Circle Detection ML Challenge - Solution

Rugved Sanjay Chavan, University of Virginia

Challenge — Final Report — January 11, 2024

1 Introduction

Develop a **circle detector** capable of locating a circle in a noisy image. The model should input an image and output the circle's center coordinates (x, y) and its radius.

1.1 Helper Functions

1. `noisy_circle`: Generates a square image with a randomly drawn circle and added noise. Returns the circle's parameters (x, y, r).
2. `show_circle`: Displays the circle in an image.
3. `generate_examples`: Provides an infinite generator of example images with corresponding parameters.
4. `iou`: Calculates the Intersection Over Union (IOU) metric to evaluate the accuracy of the circle's predicted location against its actual location.

1.2 Evaluation Metric

We recommend using the accuracy based on thresholded IOU. For instance, calculate the percentage of test examples where the predicted circle closely overlaps with the actual circle, considering different IOU thresholds (e.g., IOU $\geq$ 0.5, 0.75, 0.9, 0.95). You may propose alternative metrics if they better suit your approach.

1.3 Project Guidelines

- **Timeframe**: Approximately 2 hours, excluding training time.
- **Experience Requirement**: Some experience in training CNNs.
- **Training Tools**: For GPU usage, Google Colab is suggested.
- **Model Constraints**: $\leq 10M$ parameters. Aim for high accuracy with fewer parameters if possible.
- **Adjustable Parameters**: Feel free to modify noise levels or image sizes for efficient training.
- **Report**: Include a `README.md` detailing the code functionality and the model's final metrics.

1.4 Code Quality

- Ensure high code quality.
- Use formatting tools like JetBrains or `black`.
- Include typing and comments for better clarity and understanding.

2 Methodology

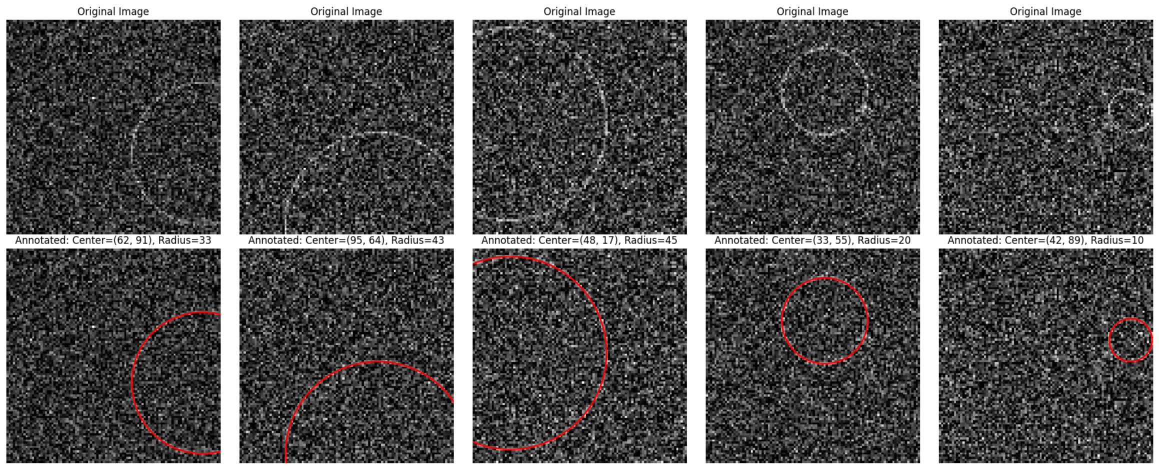
Original Image
Original Image
Original Image
Original Image
Original Image
Annotated: Center=(62, 91), Radius=33
Annotated: Center=(95, 64), Radius=43
Annotated: Center=(48, 17), Radius=45
Annotated: Center=(33, 55), Radius=20
Annotated: Center=(42, 89), Radius=10

Figure 1: Generated Original Images and Annotated with (X, Y, R) Images

2.1 Data Preprocessing

Data preprocessing is a pivotal component in methodology, aiming to optimally prepare the dataset for effective neural network training.

2.1.1 Dataset Creation

The dataset is generated using the `create_dataset` function. This function leverages a generator to produce images, each featuring a single circle against a noisy background. The parameters for dataset generation are carefully chosen to ensure diversity and representativeness:

- **Number of Samples:** The dataset comprises 10,000 images, split into 8,000 for training and 2,000 for testing.
- **Image Size:** Each image is 100×100 pixels, providing sufficient resolution for circle detection while ensuring manageable computational requirements.
- **Noise Level:** A noise level of 0.5 is applied. This level offers a balanced challenge for the model to distinguish circles amidst noise, simulating real-world conditions.
- **Circle Radius Range:** The radii of the circles are randomly chosen within a predefined range to introduce variability in the dataset.

2.1.2 Image Preprocessing

The preprocessing steps applied to the generated images are as follows:

- **Clipping Pixel Values:** Pixel values are clipped to the range [0, 255] to eliminate negative values, which are not meaningful in our context.
- **Normalization:** The pixel values are normalized to fall within the [0, 1] range, enhancing the convergence efficiency of the neural network during training.
- **Reshaping:** Images are reshaped to include a channel dimension, conforming to the input requirements of CNNs.

2.1.3 Data Visualization

To assess the quality of our dataset and preprocessing steps, the `visualize_data_with_annotations` function is utilized. This function displays a subset of images, illustrating both their original and annotated forms. The annotations consist of circles drawn according to the ground truth labels (center and radius). This visualization provides a clear insight into the dataset's characteristics, including the impact of the noise level and the variability in circle sizes. It is instrumental in confirming the appropriateness of our preprocessing steps and in understanding the challenges that the model will encounter during training.

2.2 Metrics and Custom Loss Functions

2.2.1 Intersection Over Union (IoU) Metric

A critical component of my model's evaluation is the Intersection Over Union (IoU) metric, specifically tailored to measure the accuracy of circle detection. The IoU metric quantifies the overlap between the predicted circle and the ground truth, offering a robust assessment of the model's performance. My implementation, `tf_iou`, computes IoU in a TensorFlow-compatible manner. This function first calculates the Euclidean distance between the predicted and actual circle centers. It then determines the IoU based on three conditions: no overlap, partial overlap, and one circle completely within another. This metric is pivotal for training and validating our models, providing a direct measure of their ability to accurately detect and delineate circles in noisy images.

2.2.2 Custom Circle Loss Function

To optimize my model's performance, I developed a `CustomCircleLoss` function. This loss function is designed to minimize the discrepancies in the predicted and actual circle parameters: center coordinates (x, y) and radius. The function is composed of three main components:

- **Center Coordinate Loss:** This part focuses on minimizing the differences between the predicted and actual center coordinates of the circles.
- **Radius Loss:** This segment aims at reducing the error in the predicted circle radii. A geometric adjustment is applied to this loss component to account for the relative importance of radius accuracy in circles of different sizes.
- **Scaling Factors:** The loss function incorporates scale factors for each component (x, y, radius), allowing for dynamic weighting of different aspects of the loss during training.

The implementation also includes an `UpdateScaleFactorsCallback`, a custom callback to adjust the scale factors dynamically during the training process. This mechanism enables the model to adaptively focus on different aspects of the circle detection task across training epochs, potentially leading to more balanced and effective learning.

2.2.3 Loss Function Configuration and Usage

The custom loss function is integrated into the model training process, allowing for a nuanced approach to circle detection. It is particularly designed to handle the complexities and variabilities inherent in my dataset, such as differing circle sizes and noise levels. The flexibility in configuring the loss function's components and scale factors offers a tailored approach to optimizing my model's performance.

2.2.4 10M Model

My first model, designed to meet the challenge's requirement of having fewer than 10 million parameters, adopts a sequential convolutional neural network (CNN) architecture. This model is specifically tailored to process the 100×100 pixel images and output the circle's center coordinates and radius.

Architecture: The model comprises multiple layers, including convolutional layers, max-pooling layers, dropout layers, and dense layers. The convolutional layers, with 64 and 128 filters of size 3×3, are designed to extract spatial features from the images. Max-pooling layers reduce the dimensionality, and dropout layers are used to prevent overfitting. The network concludes with dense layers, culminating in an output layer with three neurons corresponding to the circle's x, y coordinates, and radius.

Training: The model is compiled using the Adam optimizer and my custom loss function, which focuses on minimizing the difference between predicted and true values of the circle's parameters. The Intersection Over Union (IoU) metric is used for performance evaluation. I train this model with a dataset of 8,000 images, validating its performance on a separate set.

Parameters Count: The total number of parameters in this model is carefully managed to remain under 10 million, striking a balance between complexity and computational efficiency.

2.2.5 82M Model

The second model, significantly larger with 82 million parameters, is developed to explore the performance improvements possible with increased model complexity. In addition to the strategies used for the 10M model, this version incorporates a validation split and a ModelCheckpoint callback. The validation split allows for more rigorous monitoring of the model's performance during training. The ModelCheckpoint callback ensures that the best-performing model on the validation set is saved, capturing the most effective state of the model throughout the training epochs. After training, the best-performing model is evaluated on a separate test set to assess its generalization capabilities (80-10-10, Train-test-val). This evaluation provides insights into how the increased complexity of the model translates into real-world performance.

The following table 1 summarizes the performance metrics of the 10M and 82M models on the test dataset:

Metric	10M Model	82M Model
Test Loss	8.2116	6.7374
Test Accuracy	0.7721	0.8012
Mean Absolute Error for X	3.5858	2.9991
Mean Absolute Error for Y	3.8515	3.1148
Mean Absolute Error for Radius	3.2960	3.0055
Percentage of Test Examples with IOU ≥ 0.5	87.90%	89.80%
Percentage of Test Examples with IOU ≥ 0.75	63.00%	68.90%
Percentage of Test Examples with IOU ≥ 0.9	33.30%	39.90%
Percentage of Test Examples with IOU ≥ 0.95	27.20%	31.50%

Table 1: Comparison of Test Performance Metrics

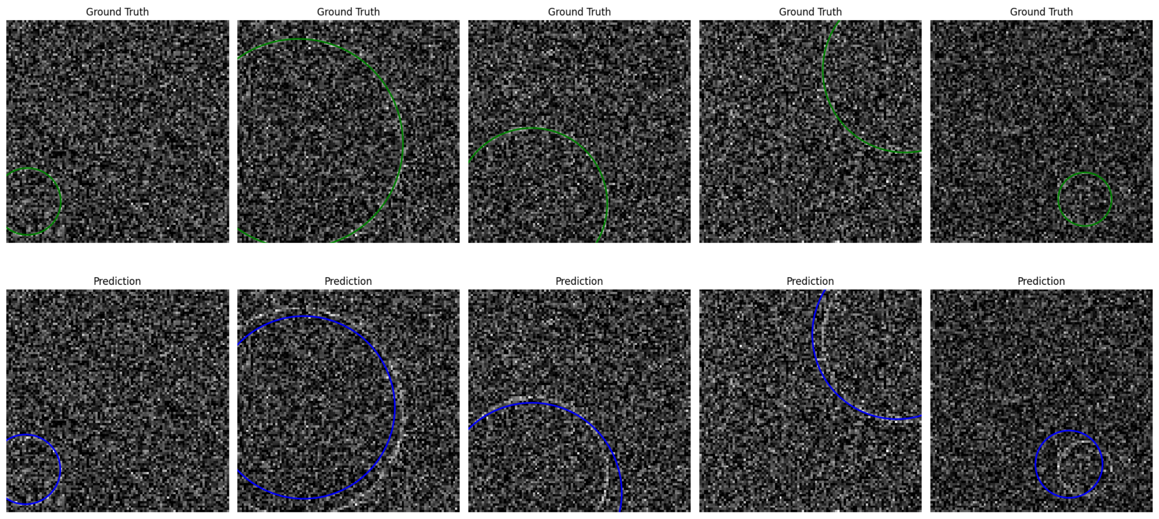

Figure 2: Prediction Results of 10M model

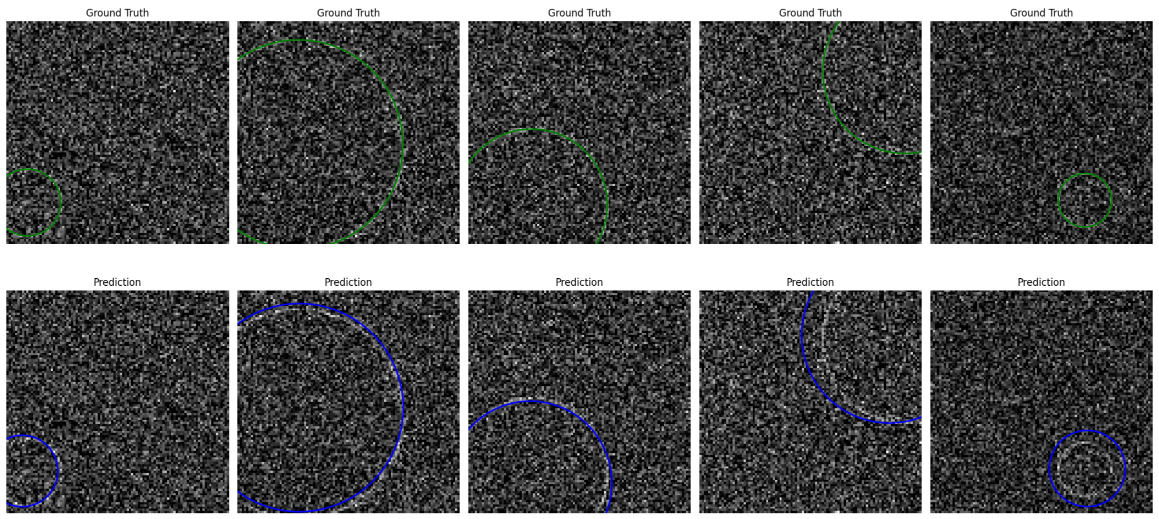

Figure 3: Prediction Results of 82M model

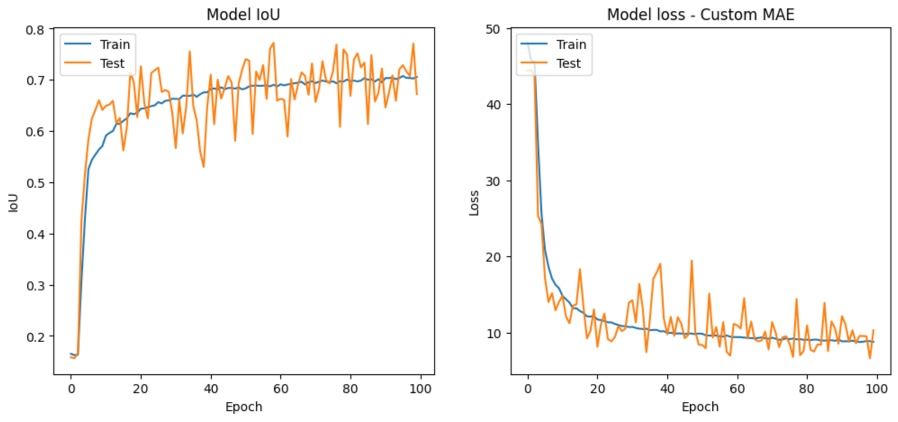

Figure 4: Training of 10M model

3 Results

Comparison of Model Performances

The comparative analysis of the 10M and 82M models, as detailed in the table, offers valuable insights into their respective performances. The 82M model shows a significant improvement with a lower test loss of 6.7374, compared to the 10M model's 8.2116, suggesting a better fit to the test data. This is further corroborated by the higher test accuracy of the 82M model (0.8012) as opposed to the 10M model (0.7721). In terms of mean absolute errors for the x-coordinate, y-coordinate, and radius, the 82M model consistently outperforms its counterpart, with values of 2.9991, 3.1148, and 3.0055 respectively, against the 10M model's 3.5858, 3.8515, and 3.2960. The 82M model also demonstrates a superior ability to accurately detect circle parameters in the image, as indicated by the higher percentages of test examples meeting various IOU thresholds. For instance, 89.80% of test examples in the 82M model meet the IOU 0.5 threshold, compared to 87.90% in the 10M model. Similar trends are observed for higher IOU thresholds, highlighting the 82M model's enhanced precision. However, these advancements in accuracy and error rates come at the expense of increased model complexity and computational demand, underlining the critical balance between performance and resource utilization in model development as shown in Fig 2, Fig 3, Fig 4 and Fig 5.

4 Conclusion

This project successfully addressed the challenge of developing a circle detector for noisy images. Employing deep learning techniques, two models were constructed: a 10M parameter model and a more complex 82M

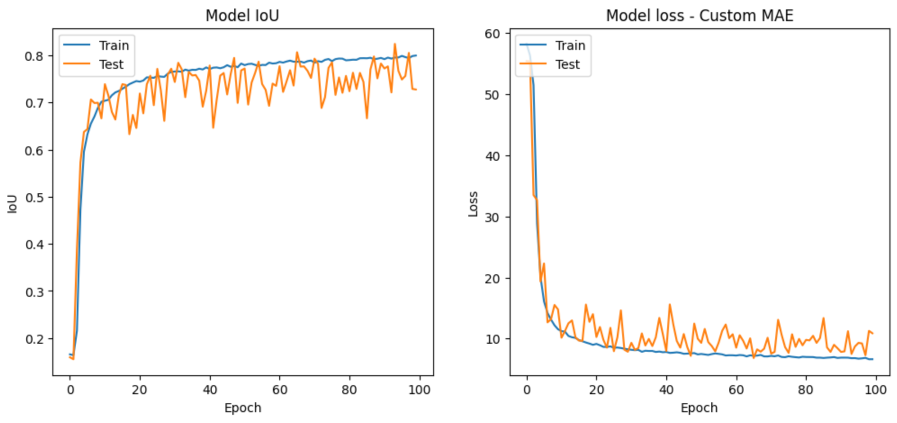

Figure 5: Training of 82M model

parameter model. Both models were rigorously evaluated using Intersection Over Union (IoU) metrics, with the 82M model demonstrating superior accuracy and lower error rates in circle detection.

The use of custom loss functions and a carefully structured training process played a crucial role in optimizing the models' performance within the constraints of computational efficiency and accuracy. The results highlight the effectiveness of these models in detecting circles despite the presence of noise, showcasing a balance between model complexity and performance.

References

[1] Anish Mittal, Anush Krishna Moorthy, Alan Conrad Bovik, "No-Reference Image Quality Assessment in the Spatial Domain", IEEE Transactions on Image Processing, 2012.

[2] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, "Deep Learning", Nature, 2015.