Enhancing Low-Light Object Detection with Log RGB Image Pre-processing

Ravina Lad[1]

***Abstract*— The conventional signal processing pipelines and compression techniques applied to visual data often disrupt the inherent rules governing the interaction of light and matter, consequently limiting the potential of deep neural networks in learning vision-based tasks. This study delves into the notion that leveraging linear or log RGB images, which adhere more closely to the fundamental physics of reflection, might offer a more conducive input for deep networks. The goal of this project is to research whether attempting to undo the effects of existing signal processing pipelines provides a better input for deep networks for object detection.**

I. INTRODUCTION

Visual information is crucial for computers to understand the world around us. However, the standard ways we process images, like through JPEG compression and typical signal processing, often distort the natural behavior of light and colors. This distortion can make it tough for deep learning systems to make sense of what they're seeing.

This study takes a different path by employing raw images, maintaining their unprocessed state and subsequently converting them into three distinct encoding formats: sRGB, linear RGB, and log RGB. These encoding techniques offer varying representations of the original visual data, each with its fidelity to the natural behavior of light and colors.

Utilizing the Faster R-CNN algorithm, we aim to evaluate and compare the performance of these encoded image formats in facilitating object detection tasks. Our aim is to determine the most efficient encoding technique for improving the capabilities of deep neural networks in computer vision tasks. This investigation seeks to determine whether encoding techniques like linear or Log space, by preserving or approximating the natural physics of light and color, can significantly influence the efficiency and accuracy of object detection.

Linear RGB preserves a more direct relationship between the digital values in an image and the actual light intensity. This direct correlation can result in more accurate and realistic representations of light and color. Standard RGB, often used in display devices, does not maintain a linear relationship between digital values and light intensity. This can lead to discrepancies in higher or lower light intensities. Logarithmic encoding, as used in log RGB, expands the dynamic range, allowing for a wider representation of brightness levels. This expanded range retains more detail in both highlights and shadows, while sRGB might clip detail in extreme highlights or shadows due to its limited dynamic range, resulting in information loss. Logarithmic encoding compresses higher-intensity values while retaining more precision in lower values and is also closer to human perception.

II. DATASET

A. RAW Object Detection Dataset

For object detection algorithms in many practical applications, such as autonomous driving, the (High dynamic Range) HDR data is essential to handle complex real-world scenarios. RAW sensor data naturally stores the HDR information without additional equipment cost. The RAW NOD (Night Object Detection) dataset comprises images captured by both Nikon and Sony cameras. Nikon stores raw images with the .nef extension, while Sony utilizes the .arw extension. Each individual raw image within the dataset occupies approximately 15 to 18 Megabytes in size. In this project, exclusively .nef raw images have been utilized. This is high-quality large-scale dataset for low-light object detection using raw sensor data of outdoor images targeting low-light object detection. The dataset contains more than 7K images and 46K annotated objects (with bounding boxes) that belong to classes: person, bicycle, and car.

B. Data Processing

In the data processing phase, three distinct encoding techniques were employed:
1. sRGB Encoding: The sRGB images were saved in the JPEG format. These images underwent a gamma correction factor of 2.2 during the conversion process from the original raw images. This step resulted in the generation of non-linear standard RGB image data using rawpy library. 2. Linear RGB Conversion: The initial step was to convert the raw images into linear RGB. Here, a gamma correction of 1 was applied, signifying a

linear relationship between the digital values and the actual intensity of light. Following the conversion to linear RGB, the images were saved in the 16-bit TIFF(Tagged Image File Format) format. This format was chosen due to its ability to retain higher bit depths. Additionally, TIFF supports lossless compression, ensuring that the image data remains unaltered and precise. 3. Log Image Generation Process: This technique involved generating log images by initially reading the linear images using OpenCV with the unchanged flag. Subsequently, the logarithm of the linear image data was computed. This step of calculating the logarithm was performed prior to the conversion of all image data into tensors and their transmission to the detection pipeline. Resizing log-encoded images might affect the nuanced details stored in the compressed areas, impacting the intended representation of brightness and contrast. Linear RGB data can undergo resizing more reliably without distorting the intended representation as it maintains a linear relationship between digital values and light intensity

C. Comparison to Existing Data-set

RAW NOD [1] (Night Object Detection) dataset mainly contains images in low light night scenarios. Different from the PASCALRAW dataset [2] and the LOD dataset [3]. PASCALRAW dataset is of only 12-bit, which is not wide enough to handle complex lighting conditions. The LOD dataset consists of the long-exposure RGB and shortexposure RAW pairs in daily scenes, which is not fit to understand the practical HDR driving scenes. COCO-style annotation for the whole dataset and for each camera subset separately (for RAW and RAWPY JPEG images).

III. Object Detection Pipeline using Faster-RCNN

Many of the current state-of-the-art models have been constructed on the foundation established by Faster R-CNN [4] model, which continues to be one of the most cited papers in computer vision. In this project, I'll break down the implementation of Faster R-CNN from scratch, highlighting the significance of executing both training and testing phases.

I navigated challenges from raw data conversion, parsing COCO-style annotations, constructing the training pipeline, computing anchor boxes, devising evaluation metrics, and ensuring a seamless end-to-end functionality. The emphasis was on creating an easily reproducible framework amidst these complexities.

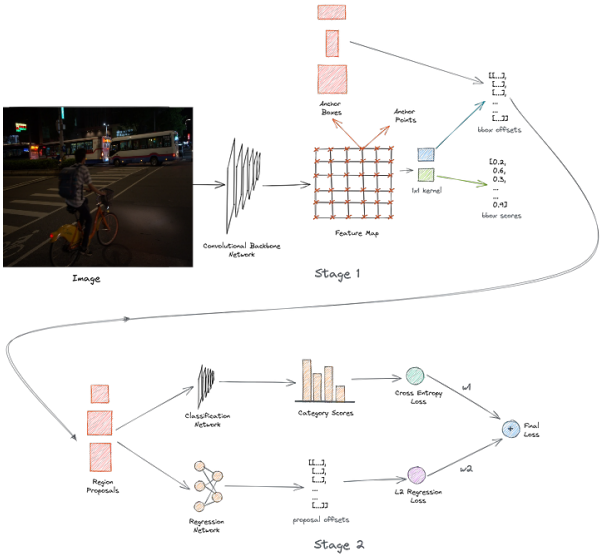

Fig. 1. Faster R-CNN Overall Architecture

A. Annotation Parser and Dataloader

I created an AnnotationParser class to read JSON files following the COCO style format. This class parses ground truth boxes in xmin, ymin, xmax, and ymax format, along with their corresponding labels. Additionally, I incorporated code to scale down the bounding box coordinates when resizing images.

In another class, ObjectDetectionDatasets, I read all these annotations and implemented padding for both bounding boxes and class labels. This step, combined with image resizing, enables batching of images. In a batch containing 'n' images with varying object counts in each, I consider the maximum number of objects across any image. I pad the remaining images with '-1' to match this maximum length. After preprocessing the dataset, I split it into three parts: training, testing, and validation sets. The split was randomized, with 70% of the data allocated for training and 15% for both testing and validation.

B. The Backbone Network

we're employing ResNet 50 as our foundational network here. In ResNet 50, each block is made up of stacks of bottleneck layers. These layers reduce the image size by half along the spatial dimension while doubling the number of channels. A bottleneck layer within ResNet 50 consists of three convolutional layers and includes a skip connection. We're specifically utilizing the initial four blocks of ResNet 50 as our backbone network.

As an image passes through this backbone network, it undergoes downsampling along the spatial dimension.

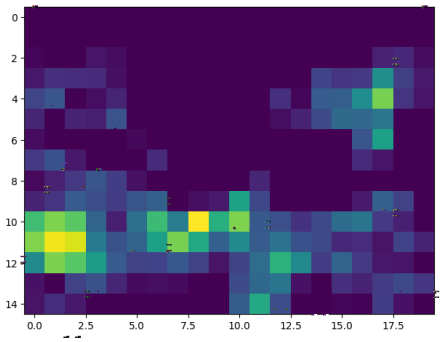

Fig. 2. feature map after passing them through the backbone

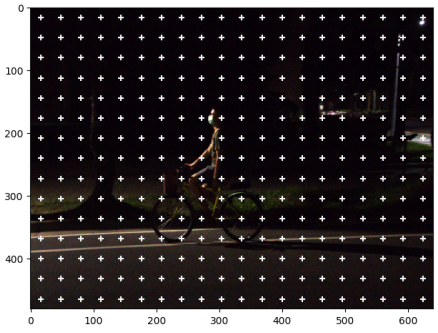

Fig. 3. Anchor Points

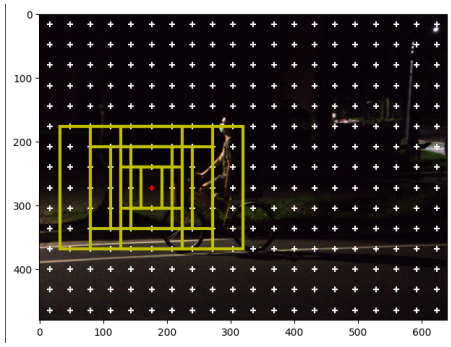

Fig. 4. Anchor Box

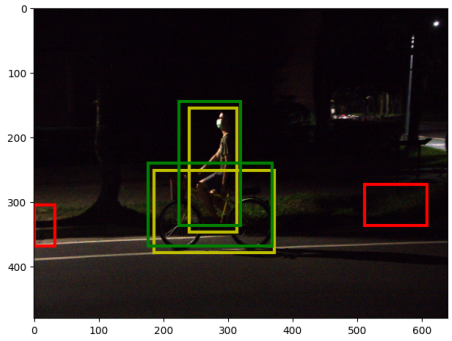

Fig. 5. Positive and Negative Anchor Boxes

For example, if we input an image sized at (640, 480) into this network, the output feature map becomes (15, 20). This process downscale the image by a factor of (32, 32). Hence, all our proposals value also will be down-scaled by the factor of 32, 32. The output is a feature-rich representation of the image given in fig 2.

C. Decoding Anchor Points

Each location on the feature map acts as an anchor point, essentially forming an array representing coordinates across the width and height dimensions. These anchor points serve as reference positions for generating bounding boxes. Check Fig 3

Using anchor scales (e.g., [2, 4, 6]) and anchor ratios (e.g., [0.5, 1, 1.5]), we create nine bounding boxes with various sizes and aspect ratios for each anchor point. The aim is to encompass diverse object shapes and sizes within an image. The specific choice of these anchor boxes typically relies on characteristics observed within the dataset being used.

D. Anchor Boxes and Offsets

Positive anchor boxes (Check Fig 4) contain an object and negative anchor boxes do not. We only need to sample a few anchor boxes for training. To sample positive anchor boxes, we select the anchor boxes that have an IoU (Intersection over union) more than 0.7 with any of the ground truth boxes. To sample negative anchor boxes, we select the anchor boxes that have an IoU less than 0.3 with any of the ground truth boxes. Usually, the number of negative samples will be far higher than the positive samples, hence will randomly sample a few to match the number of positive samples. For IOU calculation, I scaled down the ground truth boxes by factor of 32. The IoU is computed in the feature space between the generated anchor boxes and the projected ground truth boxes. Check Fig 5

The positive anchor boxes do not exactly align with the ground truth boxes. So we compute offsets between the positive anchor boxes and the ground truth boxes and train a neural network to learn these offsets. Here, we are only trying to teach the network to learn how much the anchor box is off from ground truth. We are not expecting it to predict the exact location.

E. Building the Two-stage Model Architecture

1. First Stage:

Proposal Model : I have classified each of these boxes as object or background, binary classification. This stage only talks about whether an object is present in the box

or not. In order to predict their offsets from the corresponding ground truth boxes, will use 1x1 convolution layers. They are used to change the number of filters, or to serve as a regression or classification head. Hence we take two 1x1 convolutional layers and use one of them to classify each anchor box as object or background. 1x1 convolutional layer takes the feature map and produces an output where the output filters represent the predicted offsets of the anchor boxes. This is called the regression head. We only selected the positive anchor boxes and apply predicted offsets to generate region proposals, during training. Check Fig 1

Region Proposal Network The region proposal network is the stage 1 of the detector which takes the feature map and produces region proposals. Here we combine the backbone network, the sampling module, and the proposal module into the region proposal network. During both training and inference, the RPN produces scores and offsets for all the anchor boxes. During inference, we select the anchor boxes with scores above a given threshold and generate proposals using the predicted offsets.

2. Second Stage:
Classification Model: In second stage we receive region proposals, but all proposals do not have the same size. We usually resize images in image classification tasks, but the problem is resizing is not a differentiable operation, and so backpropagation cannot happen through this operation. Hence, alternative way is to divide the proposals into roughly equal subregions and apply a max pooling operation on each of them to produce outputs of same size. This is called ROI pooling. Also, Max pooling is a differentiable operation. ROI pooling is not implemented from scartch, I used torchvision.ops. We pass resized proposals through a convolutional neural network consisting of a convolutional layer followed by an average pooling layer followed by a linear layer that produces category scores. During training, we compute the classification loss using cross-entropy. During inference, we predict the object category by applying softmax function.

Non-max suppression: Non-max suppression is basically removing duplicate bounding boxes. In this technique, we first consider the bounding box with highest classification score. Then we compute IoU of all the other boxes with this box and remove the ones which have a high IoU score. The NMS processing step is implemented in the stage 1 regression network, it's not implemented from scratch, I used torchvision.ops library.

Final Stage :
We put together the region proposal network and the classification module to build the final end-to-end Faster-RCNN model.

IV. Experiments and Results

Calculating accuracy for an object detection model involves different metrics beyond traditional accuracy, as it deals with locating and classifying multiple objects within an image. I created a list of lists in which single list contains, [image unique id, class label, confidence score, xmin, ymin, xmax, ymax]. I made list of ground truth values and predicted values. In order to calculate the mean average precision. We need to set the IOU values from 50% to 95% with increment of 5%. mAP is basically area under precision and recall graph, where precision measures the proportion of correctly predicted objects among all predicted objects and recall measures the proportion of correctly predicted objects among all true objects. mAP is calculated on confidence score above 0.9 and across different IoU thresholds.
A) When I trained the model on small dataset of 22 images, Time taken = approx 25 mins. (Check Fig 9)
1. Standard RGB Train:
mAP@0.5:0.05:0.95 = 0.52645075
2. Linear RGB Train:
mAP@0.5:0.05:0.95 = 0.61831266
3. Log RGB Train:
mAP@0.5:0.05:0.95 = 0.62789305

B) Training on 550 images (Check Fig 10) :
1. Log RGB - mAP@0.5:0.05:0.95 :

0.4799	0.4721	0.4579	0.4383	0.4032	0.3435
	0.2537	0.1527	0.0404	0.0007	

Total LOG mAP@0.5:0.05:0.95 = 0.3042501

2. Linear RGB - mAP@0.5:0.05:0.95 :

0.4745	0.4638	0.4473	0.4271	0.3831	0.3234
	0.2334	0.1330	0.0334	0.0011	

Total Linear mAP@0.5:0.05:0.95 = 0.29199868

2. standard RGB - mAP@0.5:0.05:0.95 :

0.3996	0.3875	0.3719	0.3455	0.3154	0.2782
	0.2096	0.1013	0.0235	0.0006	

Total Standard mAP@0.5:0.05:0.95 = 0.24331062

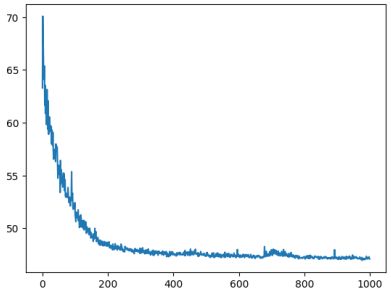

Fig. 6. Standard RGB - Training Loss

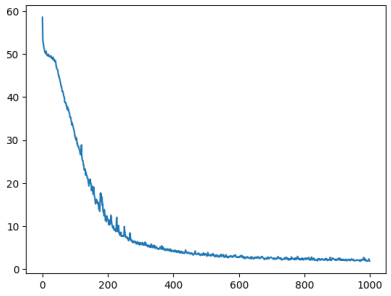

Fig. 7. Log RGB - Training Loss

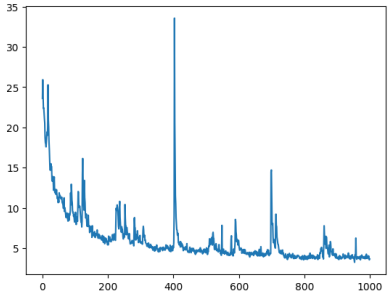

Fig. 8. Linear RGB - Training Loss

Encoding technnique	mAP@0.5	mAP@[0.5, 0.95]
sRGB	0.7181	0.52645075
Linear RGB	0.7653	0.61831266
Log RGB	0.7739	0.62789305

Fig. 9. Train mAP for subset of 22 images

Encoding technnique	mAP@0.5	mAP@[0.5, 0.95]
sRGB	0.3996	0.24331062
Linear RGB	0.4745	0.29199868
Log RGB	0.4799	0.3042501

Fig. 10. Train mAP for 550 Images

V. CONCLUSIONS

1. Obtaining raw image data can be a real challenge. Despite reaching out to various site owners who advertised raw image availability. Unfortunately, I didn't receive any positive responses.
2. Dataset I am using contains many instances of false bounding boxes and labelling. This might have lead the model to learn incorrect spatial relationships between objects and their surrounding context. This can result in inaccurate localization during inference, missing actual objects or placing bounding boxes around the wrong things.
3. Data cleaning - Manually review and correct the labeled data to remove false bounding boxes is not practically possible.
4. I had to filter out some images from the dataset because their annotations were missing. This step was necessary to ensure that all images used for analysis had complete annotations.
5. Training time for over 500 images took more than 9 hours for each encoding technique. This lengthy training duration restricted the number of hyper-parameter tuning experiments I could conduct. Additionally, due to the extensive time required, training all the models with different encoding techniques and comparing their performance consumed nearly 2 to 3 days for each evaluation cycle. This limited the scope of tuning and comparing models efficiently.
6. The logarithmic RGB model succeeded in detecting an instance of an object that both the linear and sRGB models failed to identify within the image. Specifically, in a scenario where a person was standing in a darker environment, the log space model effectively detected this presence, whereas the linear and standard RGB models overlooked it For example. 7. Switching to a different dataset with accurate and comprehensive annotations has the potential to enhance the evaluation metrics significantly. Unfortunately , I could not experiment with different datasets other than RAW-NOD (Night Object Detection) dataset.

ACKNOWLEDGMENT

I extend my heartfelt gratitude to Bruce Maxwell for invaluable guidance. His expert assistance in resolving queries and providing an opportunity to work on this project. Medium articles and the Aladdin Persson YouTube channel were instrumental in enhancing my understanding of Faster RCNN and coding IOU and mAP calculations from scratch.

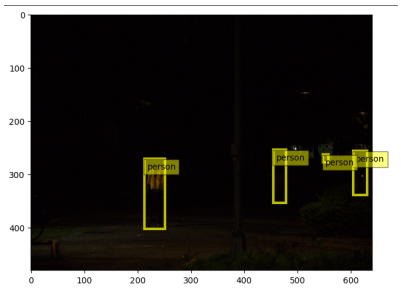

Fig. 11. Instance of an image with false label and bounding box

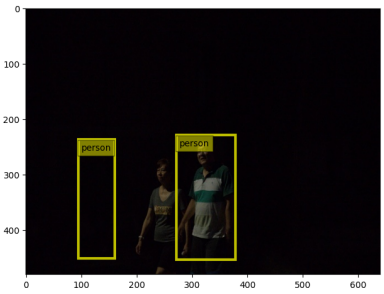

Fig. 12. Log RGB model detection instance

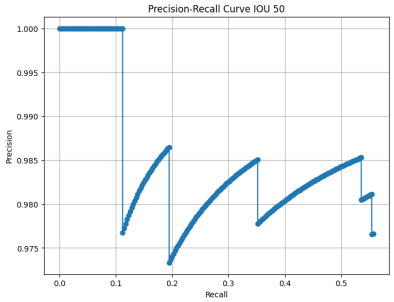

Fig. 13. Precision-recall LOG RGB IOU@50

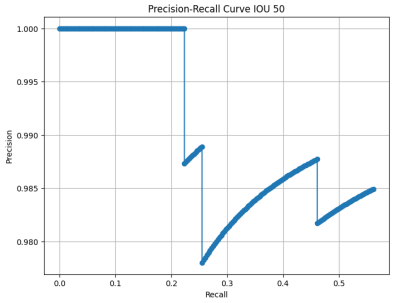

Fig. 14. Precision-recall Linear RGB IOU@50

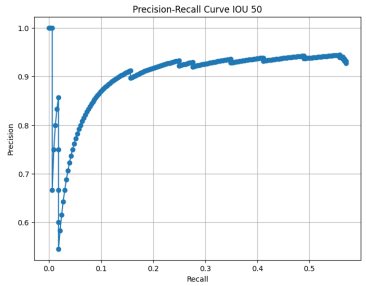

Fig. 15. Precision-recall Standard RGB IOU@50

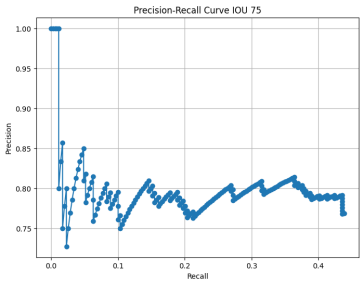

Fig. 17. Precision-recall Linear RGB IOU@75

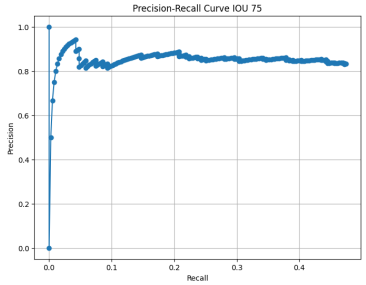

Fig. 16. Precision-recall LOG RGB IOU@75

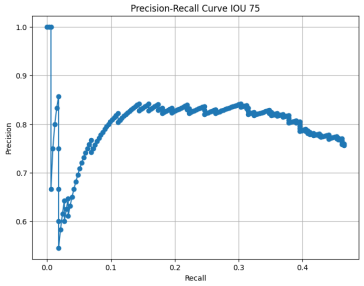

Fig. 18. Precision-recall Standard RGB IOU@75

REFERENCES

[1] Igor Morawski1 Yu-An Chen TGenISP: Neural ISP for Low-Light Machine Cognition.

[2] A Omid-Zohoor, D Ta, and B Murmann. Pascalraw: raw image database for object detection, 2014. 1, 3

[3] Hong Yang, Wei Kaixuan, Chen Linwei, and Fu Ying. Crafting object detection in very low light. In BMVC, 2021.

[4] Shaoqing Ren, Kaiming He, Jian Sun Faster R-CNN: Towards Real-time object detection with region proposal networks

[5] https://medium.com/@fractal.ai/guide-to-build-faster-rcnn-in-pytorch-42d47cb0ecd3

[6] https://blog.paperspace.com/faster-r-cnn-explained-object-detection/

[7] Link to dataset used in this project https://github.com/igor-morawski/RAW-NOD/tree/main/annotations/Nikon

[8] Xu Toward RAW Object Detection A New Benchmark and a New CVPR 2023 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15–64.