E9-246 ADVANCED IMAGE PROCESSING

ASSIGNMENT 4: REPORT

NAME: HARSHIT DUBEY

SR. NO.: 04-03-06-10-51-23-1-23043

DEGREE: M.TECH AI

DEPARTMENT : COMPUTER SCIENCE AND AUTOMATION

Question 1:

Brief of Implementation:

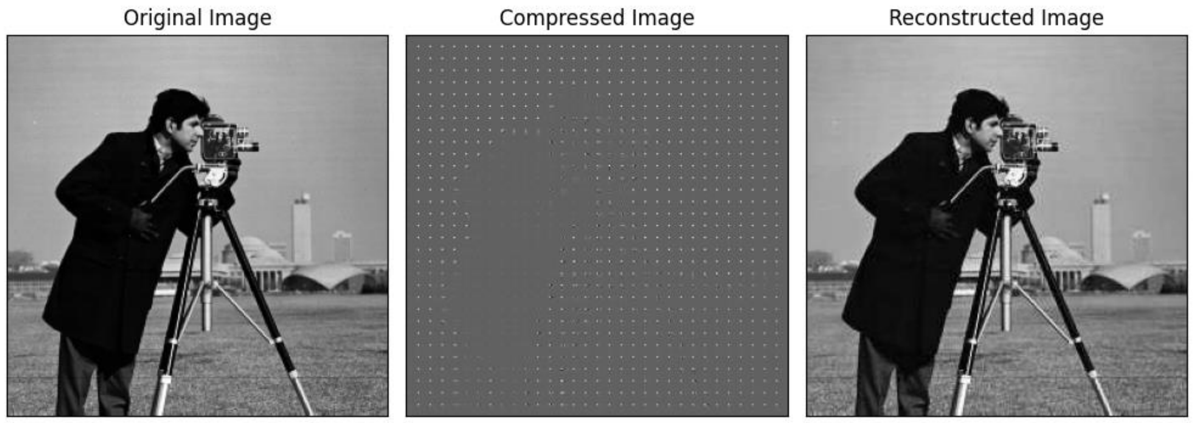
Here's a brief overview of the steps:

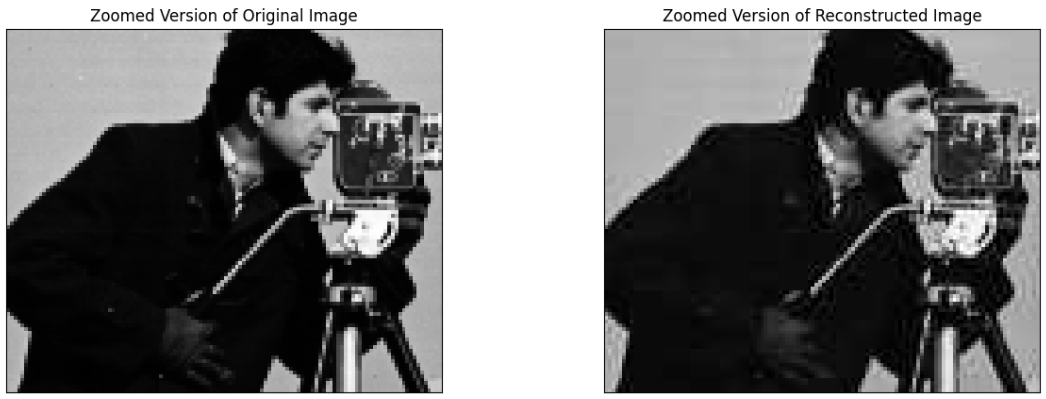
- Importing necessary libraries and loading image.
- Two quantization matrices are defined, one for a quality factor of 50 (default quantization matrix) and another for a quality factor of 80.
- **Defining the JPG_compute class**: This class is responsible for compressing, decompressing, and encoding the image. It includes the following methods:
    - Compression: This method applies the Discrete Cosine Transform (DCT) to patches of the image and then quantizes the coefficients using the provided quantization matrix.
    - Decompression: This method reconstructs the DCT coefficients from the quantized values and applies the inverse DCT to recover the image.
    - ravel_function: This method converts a 2D patch into a 1D vector using a zigzag scan.
    - decimal_to_binary: This method converts a decimal number to binary.
    - patch_encoding: This method encodes a patch using Huffman coding.
    - image_encoding: This method encodes the entire image.
- An instance of the JPG_compute class is created with the image and a quantization matrix, which is then used to compress, decompress, and encode the image into a binary string.

Results:

<u>For Q matrix (Quality Factor 50):</u>

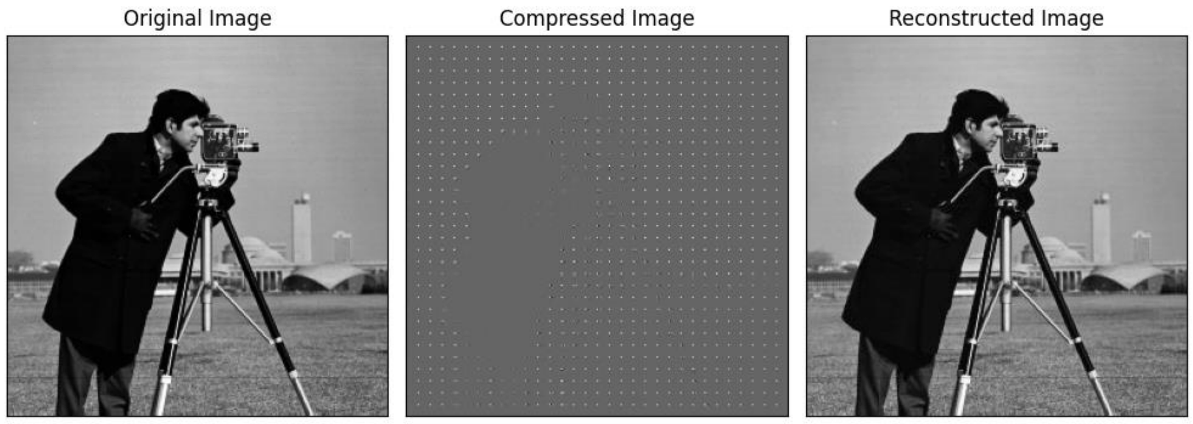
- **Mean Squared Error (MSE)**: 25.9605
- **Size of Encoded Image**: 108,260 bits
- **Size of Original Image**: 524,288 bits
- **Compression Ratio**: 4.8429




Zoomed Version of Original Image
Zoomed Version of Reconstructed Image

For Q' matrix (Quality Factor 80):

- **Mean Squared Error (MSE)**: 15.3099
- **Size of Encoded Image**: 141,006 bits
- **Size of Original Image**: 524,288 bits
- **Compression Ratio**: 3.7182



Conclusions:

From the given values, we can draw the following conclusions:

- **Quality Factor 50**: The Mean Squared Error (MSE) is 25.9605 and the compression ratio is 4.8429. This means that the image is compressed to about 20.64% of its original size, but the quality loss (MSE) is comparatively high (as compared to quality factor 80).
- **Quality Factor 80**: The MSE is 15.3099 and the compression ratio is 3.7182. This means that the image is compressed to about 26.89% of its original size. The quality loss is lower compared to a quality factor of 50 (as indicated by the lower MSE).

In conclusion, a higher quality factor results in a lower compression ratio but also a lower MSE (i.e., higher image quality after compression). Therefore, there is a trade-off between the level of compression and the resulting image quality.

Comparison with library function:

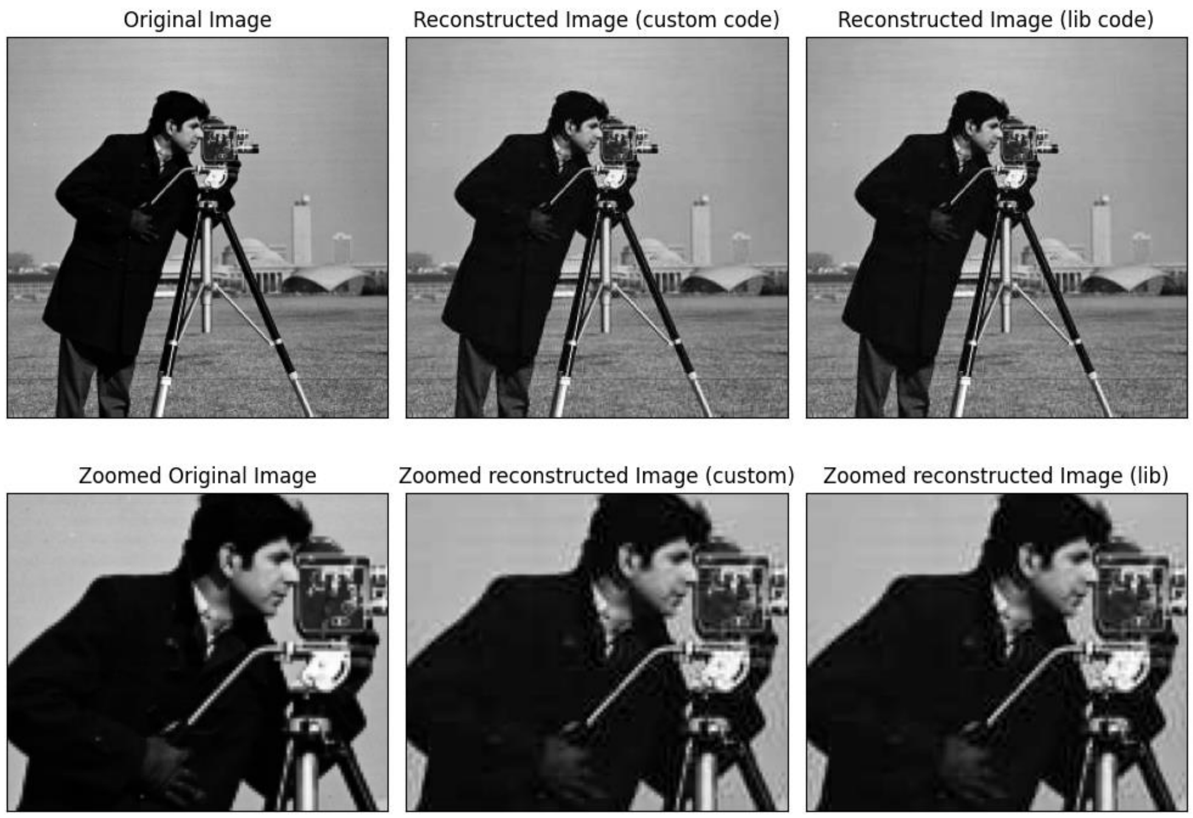
The `jpeg_compression` function takes an image and a quality factor as inputs. It compresses the image using the JPEG format with the specified quality factor. The compressed image is stored in a BytesIO object, from which the size of the compressed image in bytes is determined. The function returns the compressed image and its size.

Results:

The **Mean Squared Error (MSE)** for the custom code was 25.9605, while for the in-built code it was at 25.9923.

The **original image size** was **524,288 bits**. The **custom code** compressed this to **108,260 bits**, achieving a compression ratio of 4.8429. On the other hand, the **in-built code** compressed the image to **56,984 bits**, resulting in a significantly higher compression ratio of 9.2006.

This indicates that the built-in function employs more sophisticated techniques for compression while maintaining image fidelity.


Original Image
Reconstructed Image (custom code)
Reconstructed Image (lib code)
Zoomed Original Image
Zoomed reconstructed Image (custom)
Zoomed reconstructed Image (lib)

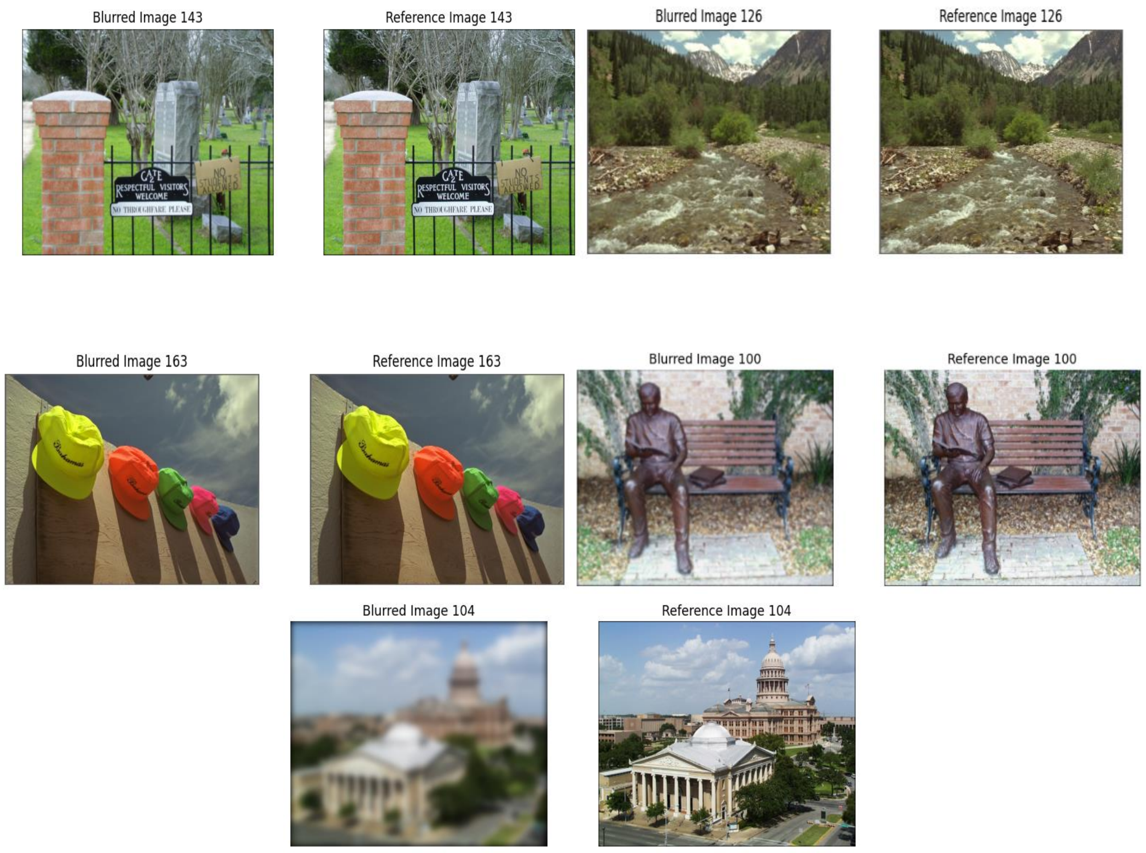
Question 2:

Brief of Implementation:

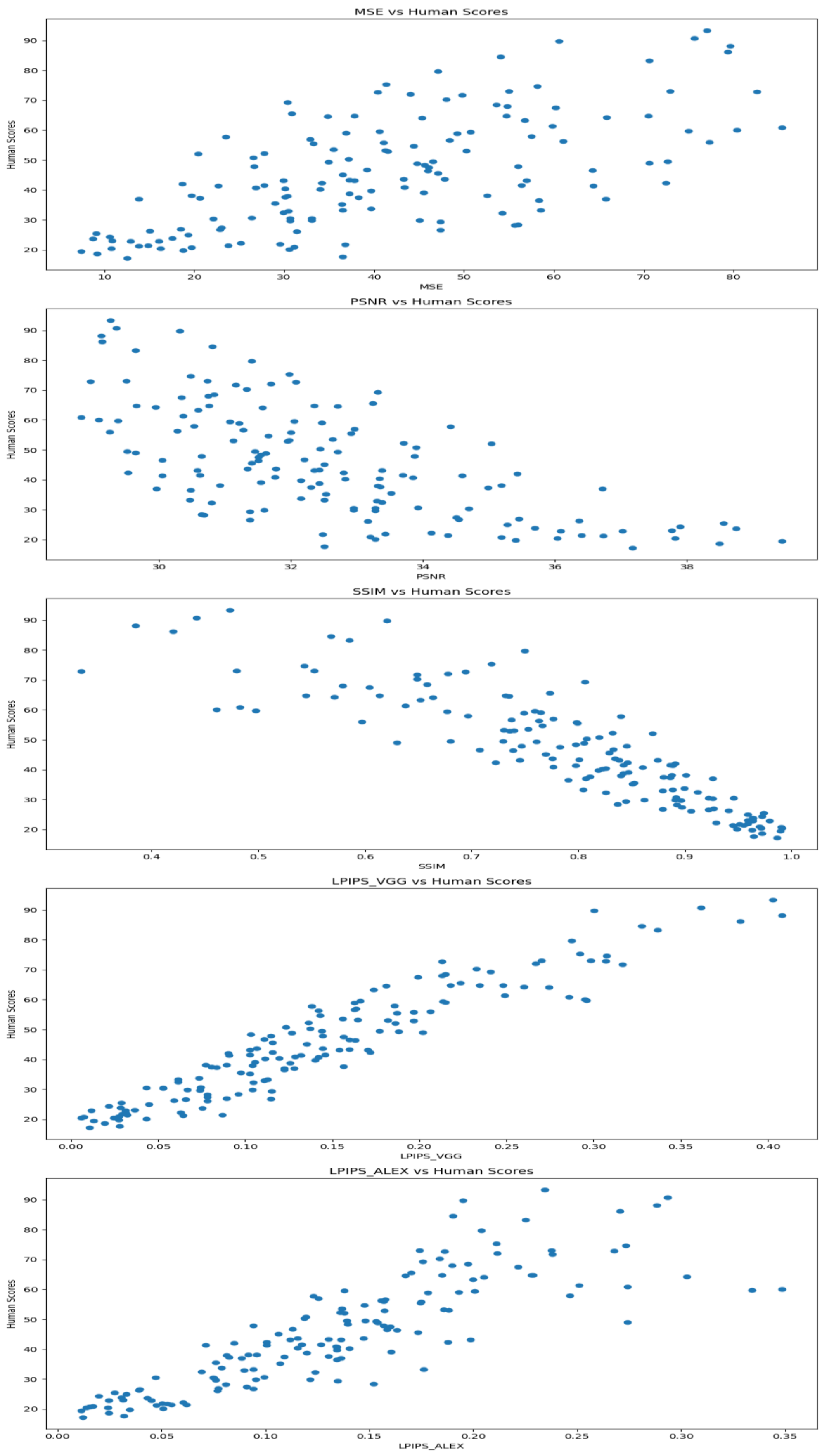
Here’s a brief overview of the steps involved:

- The code begins by loading the LPIPS models (VGG and AlexNet) and a .mat file containing human opinion scores and other data related to the images.
- Two helper functions are defined: load_data to load data from a .npy file, and compute_metrics to compute the various image quality metrics and the human rating for a pair of images.
- For each image in the dataset, the code computes the image quality metrics and the human rating, and stores these in dictionaries. The dictionaries are then saved to .npy files.
- The code then displays 5 pairs of distorted and reference images from the dataset, and it generates scatter plots comparing various image quality metrics with human scores.
- Finally, the code calculates the Spearman correlation between each image quality metric and the human scores.

Results:



Images



Scatter Plots

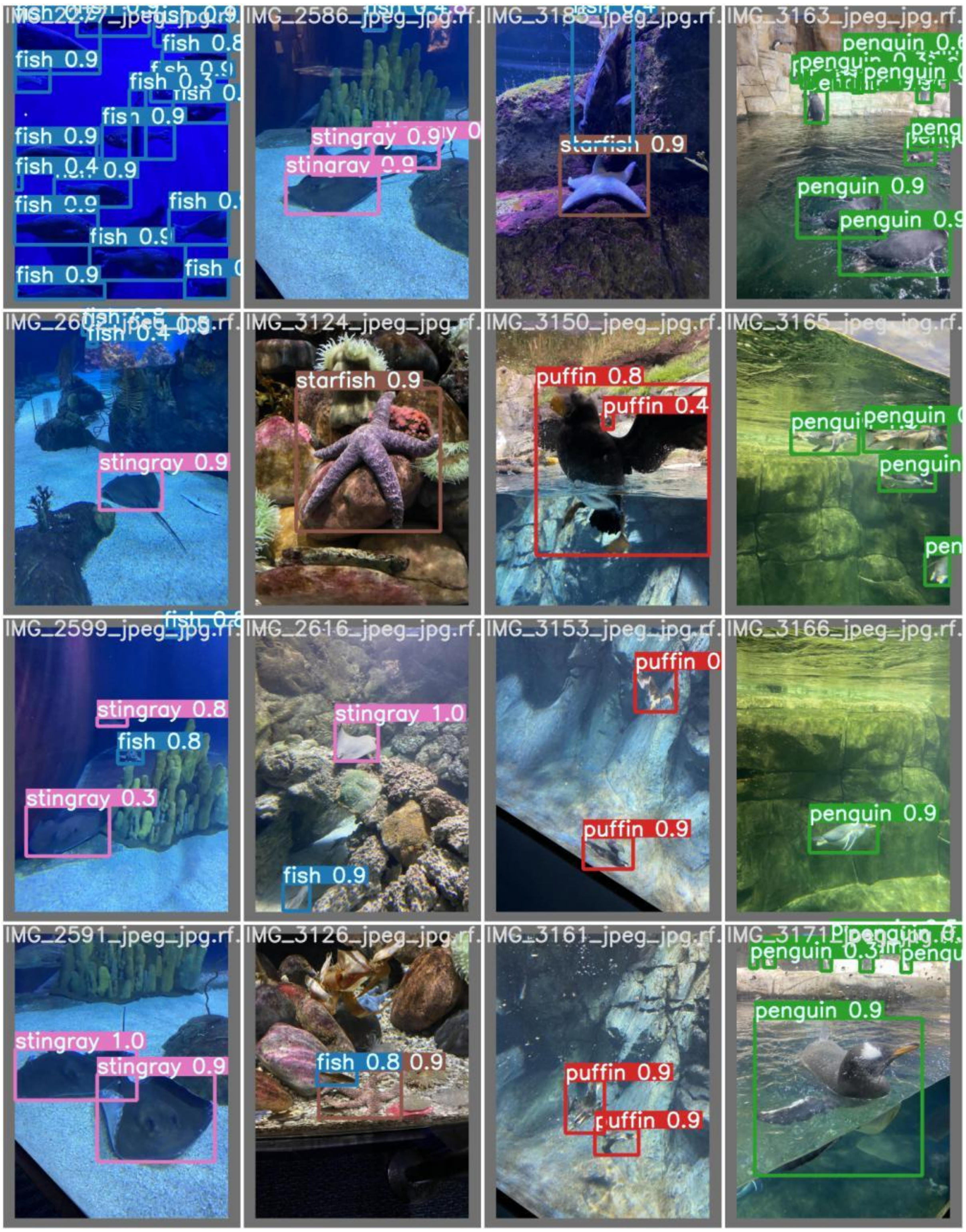
Correlation Between Image Quality Metrics and Human Scores:

Metric	Spearman Correlation
MSE	0.6607 (positive)
PSNR	-0.6607 (negative)
SSIM	-0.9181 (negative)
LPIPS_VGG	0.9439 (positive)
LPIPS_ALEX	0.8980 (positive)

Key Points:

- **MSE and PSNR:** These metrics have a weak correlation with human perception. These metrics focus solely on pixel-wise differences, which may not capture the nuances of human visual perception. Humans are more sensitive to structural distortions and artifacts than just raw intensity variations.
- **SSIM:** This metric shows a strong negative correlation with human scores. This metric incorporates aspects of luminance, contrast, and structure similarity, therefore showcasing strong correlation with human scores.
- **LPIPS_VGG and LPIPS_ALEX:** These metrics based on deep learning networks (VGG and AlexNet) exhibit strong positive correlations with human scores. Lower LPIPS scores indicate higher perceived quality, suggesting good alignment with human judgment.

Question 3:



Results

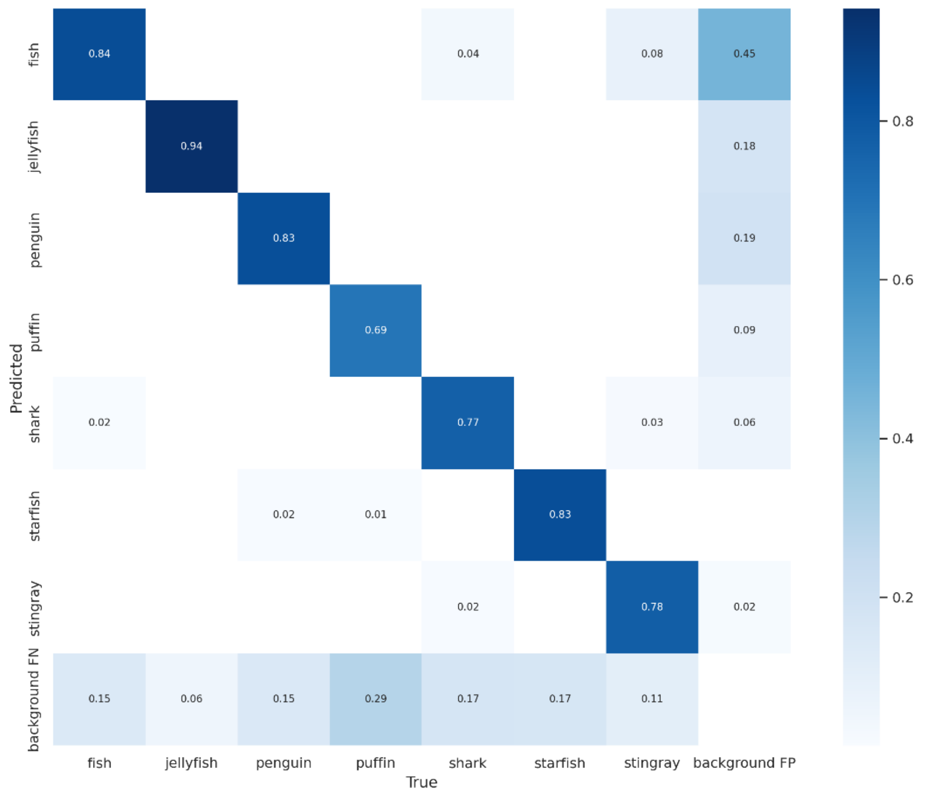
Results:

Training Dataset:

Class	P	R	mAP@0.5	mAP@[0.5:0.95]
all	0.858	0.743	0.816	0.512
fish	0.890	0.749	0.860	0.509
jellyfish	0.840	0.915	0.939	0.536
penguin	0.748	0.744	0.762	0.354
puffin	0.775	0.554	0.645	0.308
shark	0.882	0.667	0.796	0.551
starfish	1.000	0.756	0.845	0.681
stingray	0.871	0.818	0.862	0.646

<u>Plot:</u>

<u>Confusion Matrix:</u>

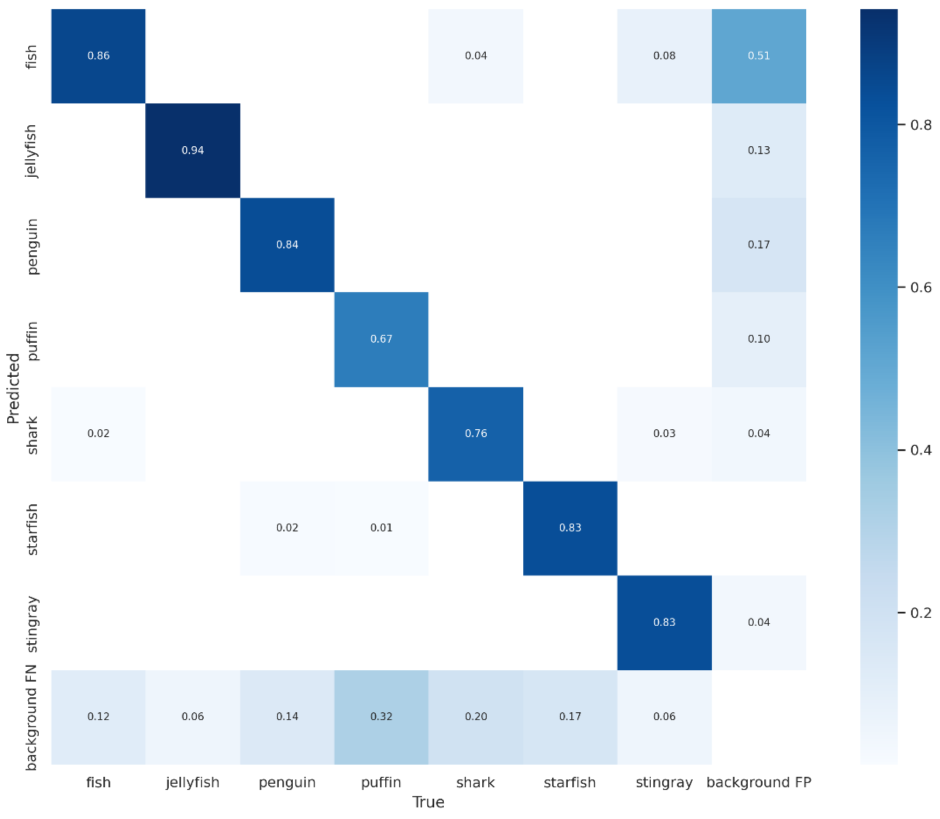


Test Dataset:

Class	P	R	mAP@.5	mAP@[0.5:0.95]
all	0.854	0.752	0.828	0.521
fish	0.879	0.792	0.875	0.510
jellyfish	0.889	0.910	0.951	0.558
penguin	0.762	0.740	0.768	0.363
puffin	0.774	0.600	0.669	0.298
shark	0.863	0.666	0.796	0.552
starfish	0.991	0.741	0.850	0.696
stingray	0.817	0.818	0.885	0.670

Plots

Confusion Matrix:



Qualitative Analysis:

Here's a more qualitative analysis based on the provided data:

Overall (all classes):

- The model performs reasonably well across all classes, with an average precision around 0.82-0.83 and recall around 0.75 on both training and test sets. This indicates a good balance between correctly identifying positive instances (high recall) and avoiding false positives (moderate precision).
- There is room for improvement, especially in the higher IoU range (mAP@0.5:0.95 around 0.52), suggesting some challenges in precisely localizing object boundaries.

Individual class:

- **Fish:** The model exhibits high precision (~0.89) but slightly lower recall (~0.75-0.79), suggesting it may be conservative in predicting fish instances, potentially missing some true positives. The performance is consistent across training and test sets.
- **Jellyfish:** This class stands out with very high recall (0.91-0.95), indicating the model effectively captures most jellyfish instances. The precision is also good (~0.84-0.89), and the class achieves the highest mAP@0.5 (~0.95), demonstrating strong overall performance.
- **Penguin:** Both precision and recall are moderate (~0.74-0.76), with mAP values around 0.35-0.37, indicating this class is more challenging for the model. The performance is consistent across datasets.
- **Puffin:** This class has the lowest performance among all classes, with precision around 0.77, recall around 0.55-0.60, and low mAP values (~0.30). The model struggles to accurately identify and localize puffin instances, likely due to their unique appearance.
- **Shark:** The model exhibits good precision (~0.86-0.88) but lower recall (~0.67), suggesting it may be conservative in predicting sharks, potentially missing some true positives. The mAP values (~0.55) indicate moderate overall performance.
- **Starfish:** While the precision is perfect (1.0) on the training set, indicating no false positives, the recall of 0.756 suggests the model misses some starfish instances. The test set shows lower but still good precision (0.991) and recall (0.741).
- **Stingray:** Both precision and recall are consistently high (~0.82) across datasets, with mAP values around 0.65-0.67, indicating good overall performance for this class.

In summary, the model performs well on classes like jellyfish and stingray, moderately on fish, shark, and starfish, and comparatively poorly on penguin and puffin.

Comparison :

The evolution from YOLO v1 to YOLO v7 has brought significant improvements in architecture design, loss function, and computation complexity, addressing some of the drawbacks of the original YOLO v1 model.

- **Architecture design:**
  - YOLO v1: Used a single fully convolutional network to predict bounding boxes and class probabilities.
  - YOLO v7: Adopts a more sophisticated architecture with advanced features, enabling better feature extraction and representation.

These architectural changes improve the model's ability to capture complex spatial and channel-wise relationships, leading to better object detection performance.

- **Loss function:**
  - YOLO v1: Utilized a single loss function that combined classification, localization, and confidence components.
  - YOLO v7: Employs more advanced loss functions, such as the IoU-based loss (e.g., GIoU, DIoU), which directly optimize the intersection over union (IoU) between predicted and ground truth bounding boxes.

These improved loss functions provide better gradient signals for bounding box regression, addressing the localization challenges faced by YOLO v1.

- **Computation complexity:**
  - YOLO v1: Had a relatively high computational complexity due to its fully convolutional nature and the need to evaluate the entire image at multiple scales.
  - YOLO v7: Incorporates techniques like model pruning, quantization, and efficient attention mechanisms to reduce computational complexity without sacrificing performance.

These optimizations make YOLO v7 more efficient and suitable for deployment on resource-constrained devices or real-time applications.

<u>The changes proposed in YOLO v7 address several drawbacks of YOLO v1:</u>

1. Improved architecture design: The incorporation of advanced architectural enhances the model's ability to capture long-range dependencies and focus on relevant features, leading to better object detection performance.

2. Advanced loss functions: The use of IoU-based losses directly optimizes the model's bounding box regression capability, addressing the localization challenges faced by YOLO v1, which relied on a more simplistic loss function.

3. Reduced computation complexity: The introduction of techniques like model pruning, quantization, and efficient attention mechanisms helps reduce the computational burden of YOLO v7 compared to YOLO v1, making it more suitable for real-time applications or deployment on resource-constrained devices.

4. Data Augmentation Strategy:

YOLO v1 relied on traditional, predefined data augmentation techniques like random cropping, flipping, and scaling, which may not be optimally tailored to the specific dataset and task.

YOLO v7 incorporates the Trainable Bag of Freebies concept, which allows the model to learn and adapt its own augmentation strategies during training. An additional Augmentation Network module generates augmentation parameters based on the input image and the current state of the model. The loss function is modified to encourage the Augmentation Network to produce augmentations that improve the model's performance. This dynamic and adaptive augmentation approach addresses the limitations of fixed, generic augmentation techniques used in YOLO v1, potentially leading to better generalization and robustness.

Overall, the evolution from YOLO v1 to YOLO v7 represents a significant improvement in object detection capabilities, addressing key limitations of the original YOLO model while enhancing performance, accuracy, and efficiency.