Assignment 04

E9 246 Advanced Image Processing

Name: **SRIYA R G**
SR No: 22414
Date of Sub: 10.04.2024

Problem 1: JPEG Implementation (15 points)

Implement a toy version of JPEG through the following steps:

1. Transform: Compute an 8x8 discrete cosine transform (DCT) for every non-overlapping block in the input grey scale image.

2. Quantization: Use the following quantization matrix to quantize each DCT coefficient in a given 8x8 block

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Note that the quantized index of the DCT coefficient $x(i, j)$ is given by

$$y(i, j) = \left\lfloor \frac{x(i, j)}{Q(i, j)} + 0.5 \right\rfloor$$

and the reconstruction is given by $\hat{x}(i, j) = y(i, j)Q(i, j)$.

3. Lossless source coding: Use the following table to encode the quantized index corresponding to each DCT coefficient.

Quantized DCT index	Code
0	0
$-1, 1$	10x
$-3, -2, 2, 3$	110xx
$-7, -6, -5, -4, 4, 5, 6, 7$	1110xxx
...	...

The output bitstream (or file) is given by the concatenation of the sequence of bits produced for each 8x8 block.

Using the JPEG implementation described above:

1. Compute the size of the output file generated for the cameraman.tif image provided to you. Also compute the mean squared error (MSE) between the original image and reconstructed image. The reconstructed image is obtained by taking the inverse DCT for each block of quantized reconstructions of DCT coefficients. Calculate the compression ratio (defined as the ratio of the input image in bits and size of the output file in bits).

2. Compare the file size with the file size obtained with any default Python/MATLAB function for JPEG compression. Note that for this comparison, you need to control any input parameters to the default functions such that the MSE is the same for the previous case (the one that you implemented).

Problem 1- Answer

Part 1

Compression Ratio : 9.064734245967966
Original Image Size in bits: 981330
Compressed Size (Custom) in bits: 108258
MSE between original and reconstructed image: 44.54563508062465

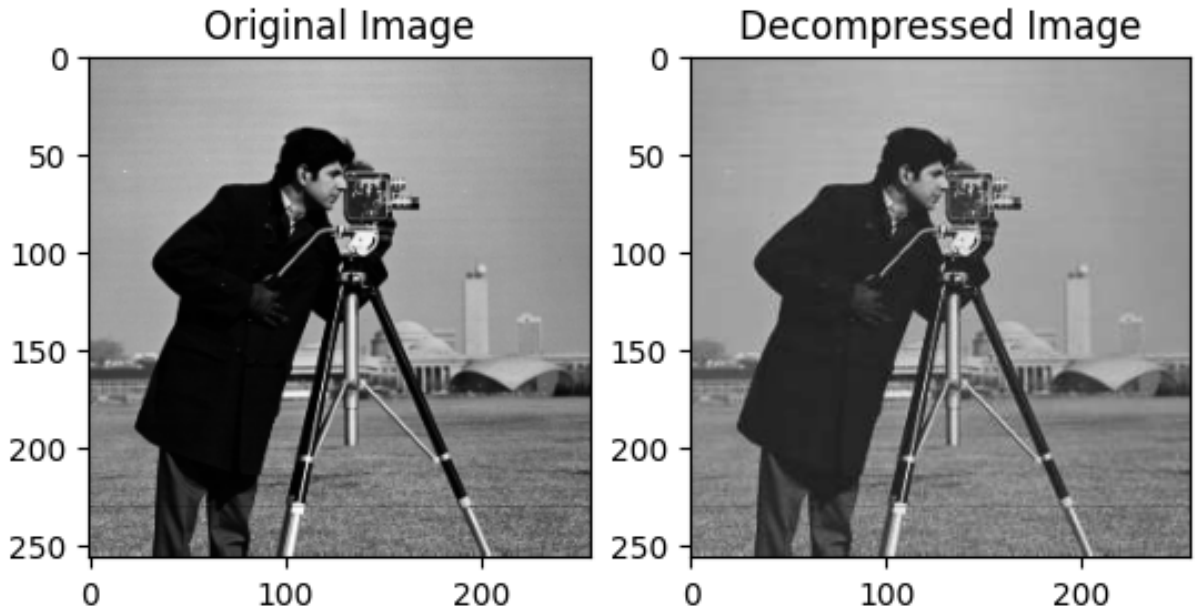

Figure 1: Original vs reconstructed image

Part 2

Custom Compression Compressed Size: 108258 bits
Default Compression - Quality: 5, Compressed Size: 1945 bits
Compression ratio using default: 504.5398457583548
MSE: 44.54563508062465

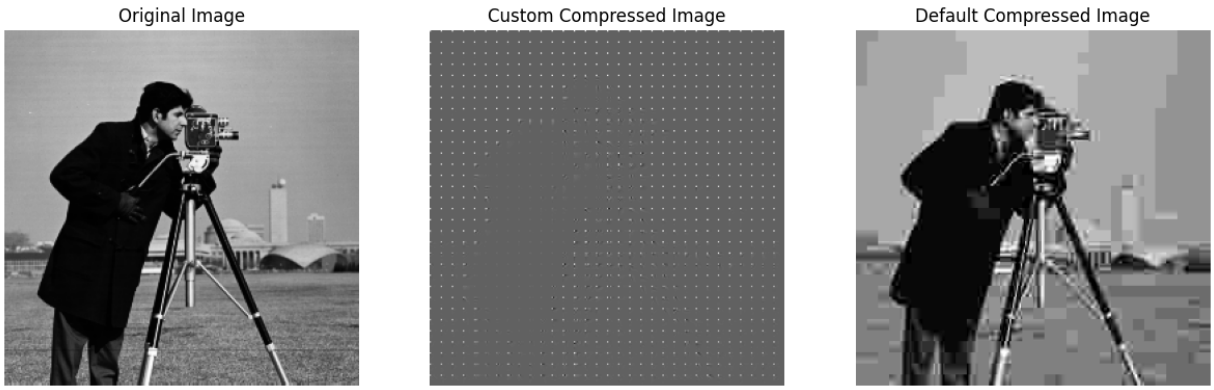

Figure 2: Enter Caption

Problem 2: Comparison of perceptual quality measures (20 points)

In this problem, you will compare PSNR, SSIM and LPIPS (2 versions) with respect to their performance in terms of correlation with human perception.

A. Algorithms: Evaluate the following metrics on the dataset mentioned in Part (b). You should obtain a single number as quality with respect to each of the metrics below for each image.

1. Peak signal to noise ratio in pixel domain
2. Single scale structural similarity index (you can use default code available in Python or MATLAB)
3. Learned perceptual image patch similarity metric (LPIPS): This is a deep learning based measure that is available online (https://pypi.org/project/lpips/). You need to compare two versions here, one that uses VGG and another that uses Alexnet.

B. Performance measurement: Download the database available at http://ece.iisc.ac.in/ rajivs/courses/aip2016/hw5.rar

The database comes with the following:

1. distorted images in the "gblur" folder
2. reference images in the "refimgs" folder
3. reference image name for every distorted image in the "gblur" folder in "refnames_blur"
4. human opinion scores in "blur_dmos"
5. indicator of whether the image in the "gblur" folder is an original image in "blur_orgs"

C. Questions

1. Compute the Spearman rank order correlation coefficient between the dmos scores in "blur_dmos" and each metric you considered in A after having removed the scores that correspond to the original images in the "gblur" folder (as mentioned earlier, this information is contained in blur_orgs).
2. Comment on the relative performances of all the indices

Problem 2- Answer

The metrics considered are Peak Signal-to-Noise Ratio (PSNR), Single Scale Structural Similarity Index (SSIM), and two versions of the Learned Perceptual Image Patch Similarity (LPIPS) metric using VGG and AlexNet architectures.

Table 1 presents the Spearman correlation coefficients between the DMOS scores and each quality metric. The coefficients indicate the strength and direction of the relationship between the human perception scores and the metric scores.

Table 1: Spearman Correlation Coefficients

Index	Spearman_PSNR	Spearman_SSIM	Spearman_LPIPS_VGG	Spearman_LPIPS_ALEX
Value	0.012996	-0.013398	-0.000468	-0.019298

The Spearman correlation coefficient quantifies the strength and direction of the monotonic relationship between two variables. A coefficient closer to 1 or -1 indicates a strong positive or negative correlation, respectively, while a coefficient close to 0 suggests no significant correlation.

In this table, we observe that the PSNR metric has a small positive correlation with human perception, as indicated by the positive Spearman coefficient. However, SSIM and both versions of the LPIPS metric show weaker correlations, with coefficients closer to zero. PSNR shows a small positive correlation with human perception, SSIM and LPIPS metrics exhibit weaker correlations. This implies that none of these metrics fully capture the nuances of human perception when it comes to assessing image quality.

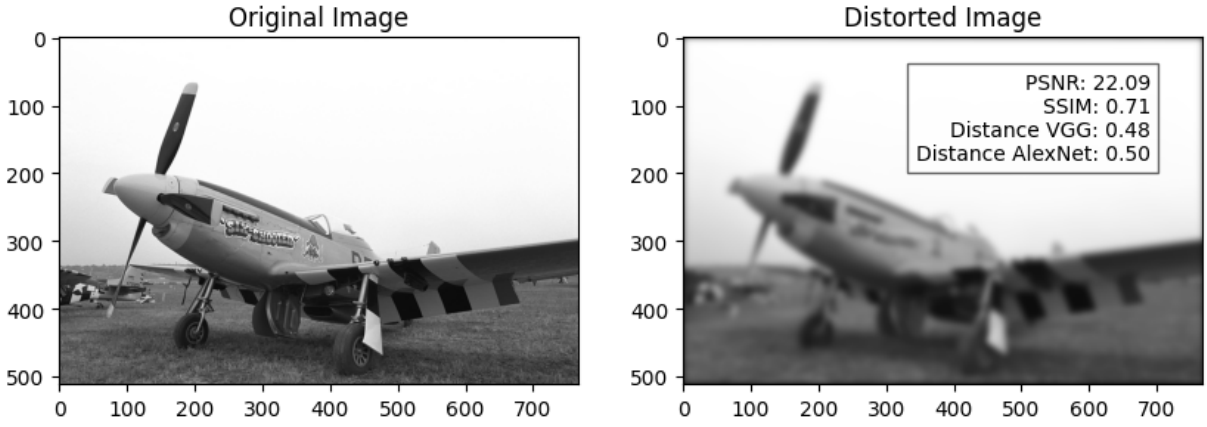

Figure 3: Example result 1

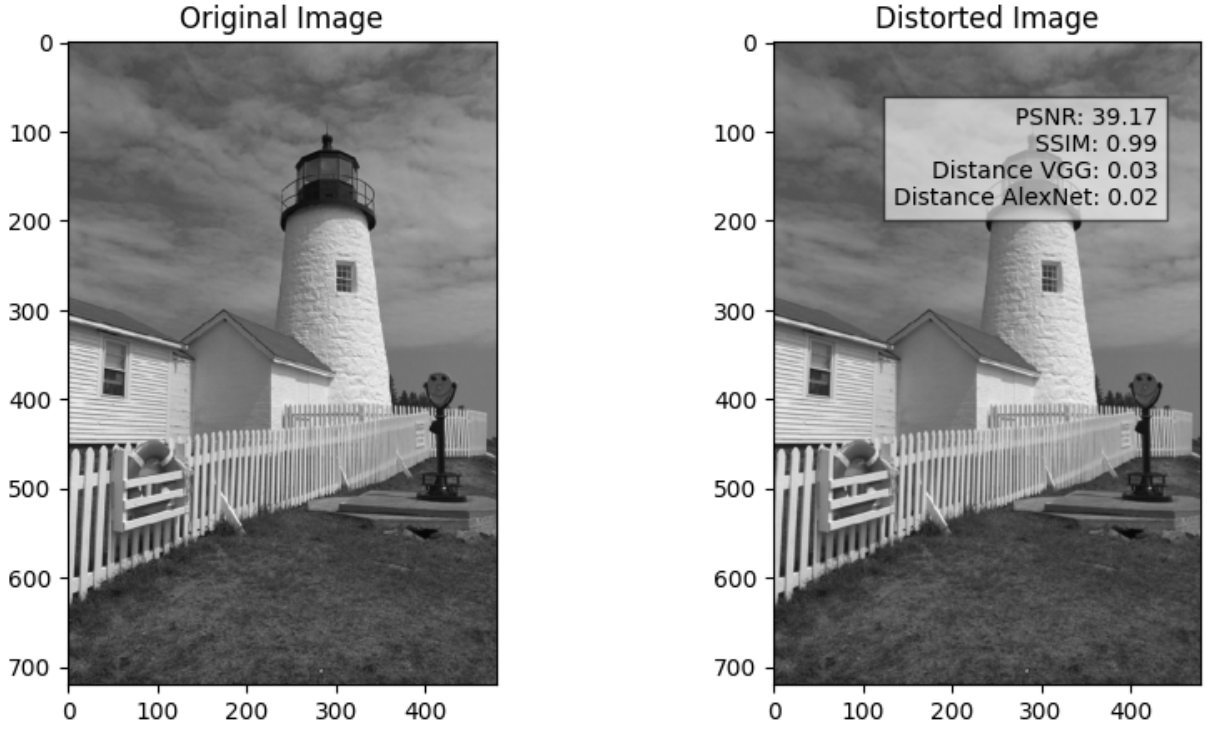

Figure 4: Example result 2

Problem 3: YOLO Object Detection (15 Points):

- Train a YOLO v7 Object Detection model on this dataset.
- The dataset has 7 classes. The classes are defined in data.yaml file. Each image has a corresponding label file with bounding box annotations. Each row in the annotation file is of the form class-id center-x center-y width height corresponding to an object in the image. The values are in zero to one normalized coordinate space. Refer YOLOv5 annotation format if any doubts.
- Evaluate the performance of the trained model on the given test set. Compute the following metrics: (i) mAP@0.5IoU; (ii) mAP@[0.5:0.95]IoU (take average of mAP at 10 IoU thresholds from 0.5 to 0.95 with a step size of 0.05). Also, give the qualitative results in your report.
- Briefly Summarize the differences between YOLO v1 (taught in class) and YOLO v7 in terms of (1) Architecture design, (2) Loss function, (3) Computation complexity. Justify how the changes proposed addresses the drawbacks in YOLO v1, if any.

Problem 3- Answer

Hyper parameter settings: Epochs=50 Batch size=8 Initial learning rate=0.01 decay rate=0.0005 momentum=0.937

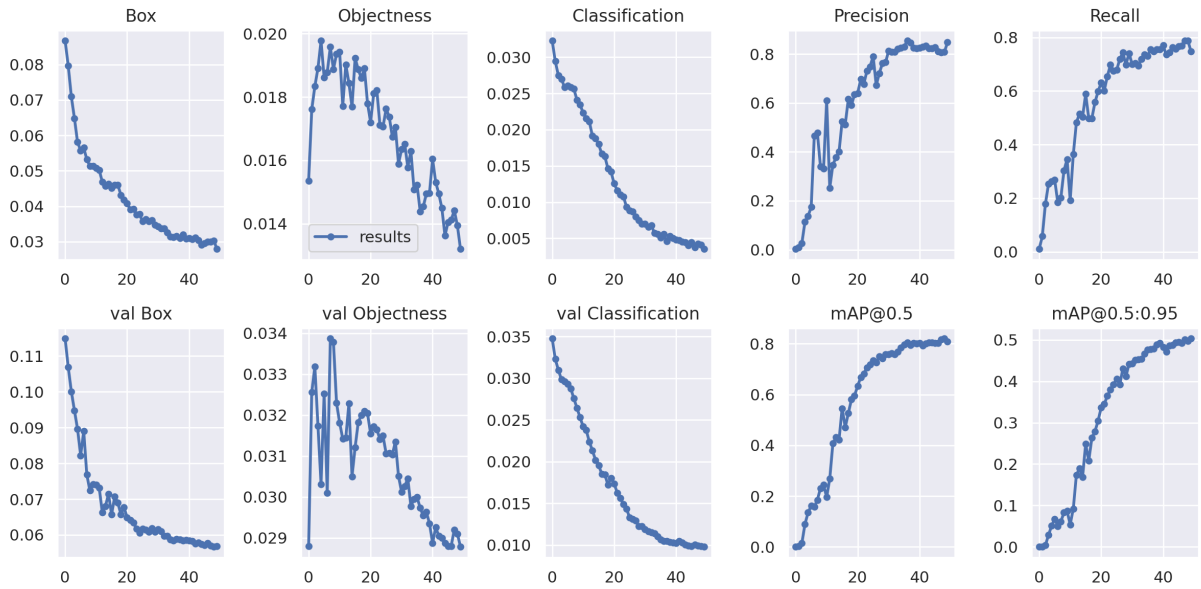

Figure 5: Results on train data

Results on validation dataset

Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:
all	127	909	0.797	0.76	0.808	0.489
fish	127	459	0.823	0.77	0.851	0.498
jellyfish	127	155	0.783	0.916	0.926	0.514
penguin	127	104	0.756	0.798	0.786	0.337
puffin	127	74	0.679	0.635	0.643	0.303
shark	127	57	0.878	0.63	0.765	0.492
starfish	127	27	0.824	0.815	0.864	0.677
stingray	127	33	0.837	0.758	0.822	0.602

Figure 6: Performance on Validation dataset- The last column gives average of mAP at 10 IoU thresholds from 0.5 to 0.95 with a step size of 0.05

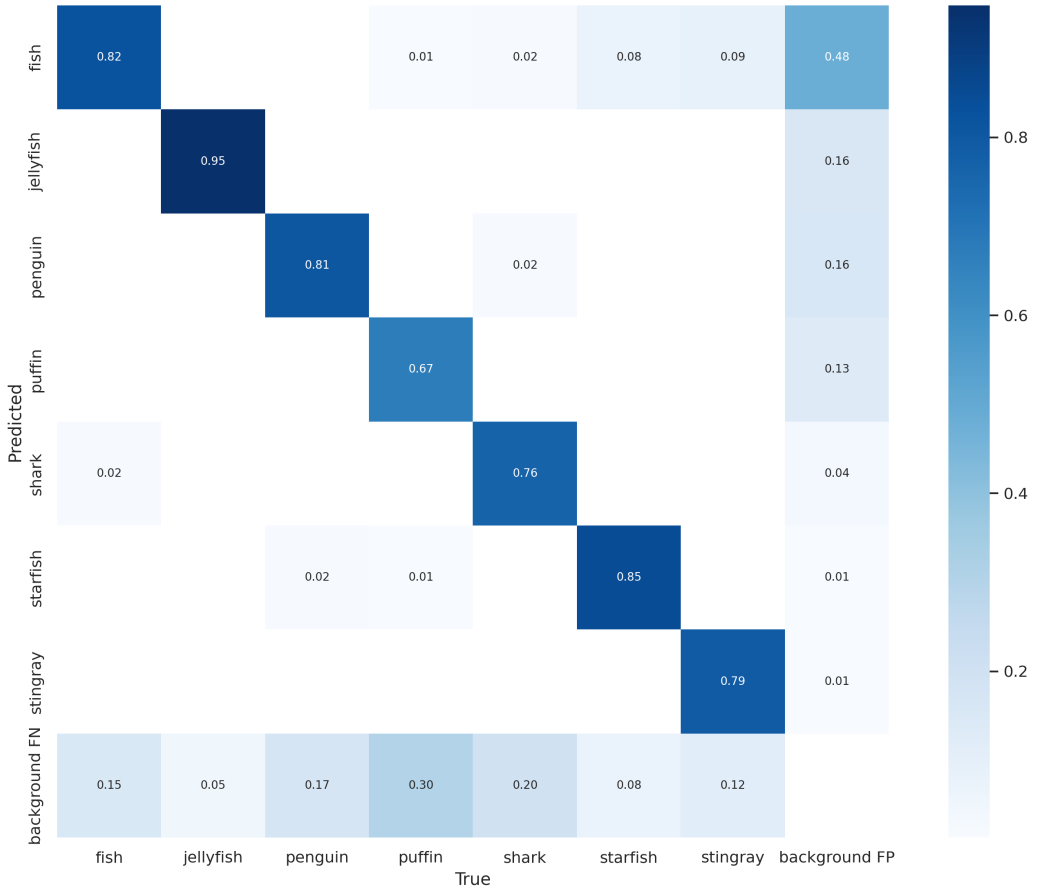

Figure 7: Confusion Matrix

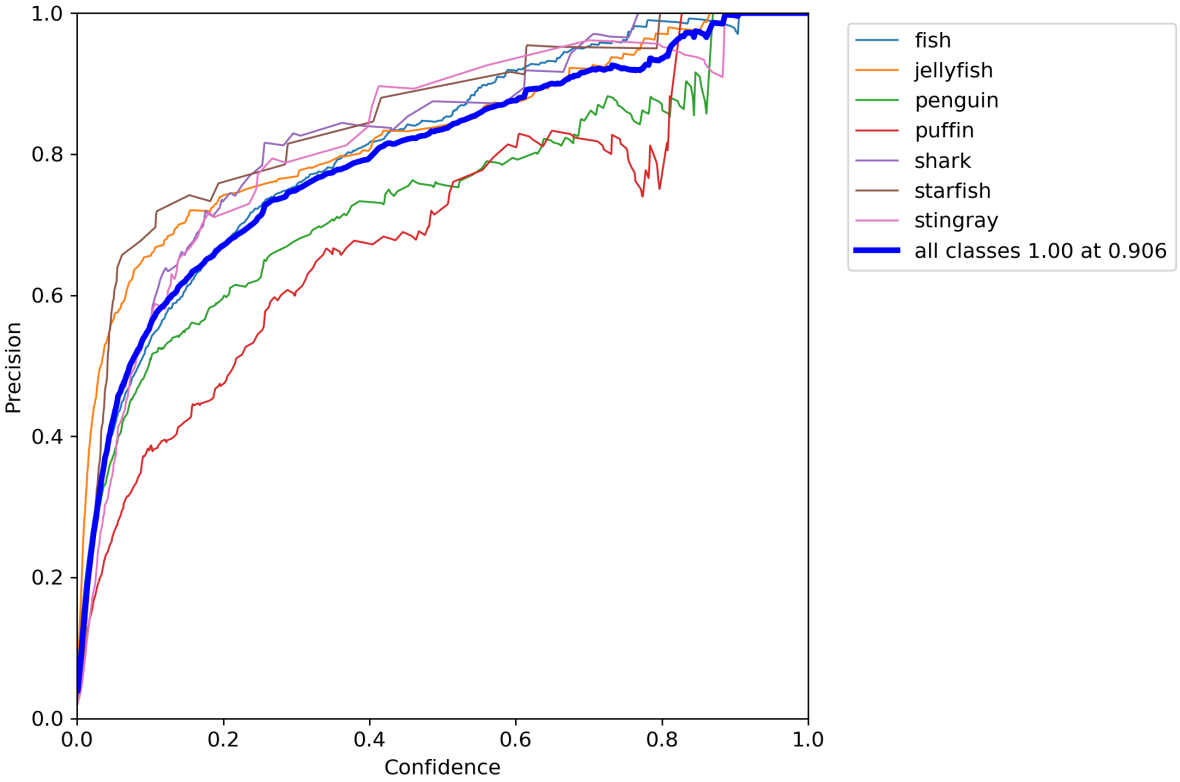

Figure 8: P curve

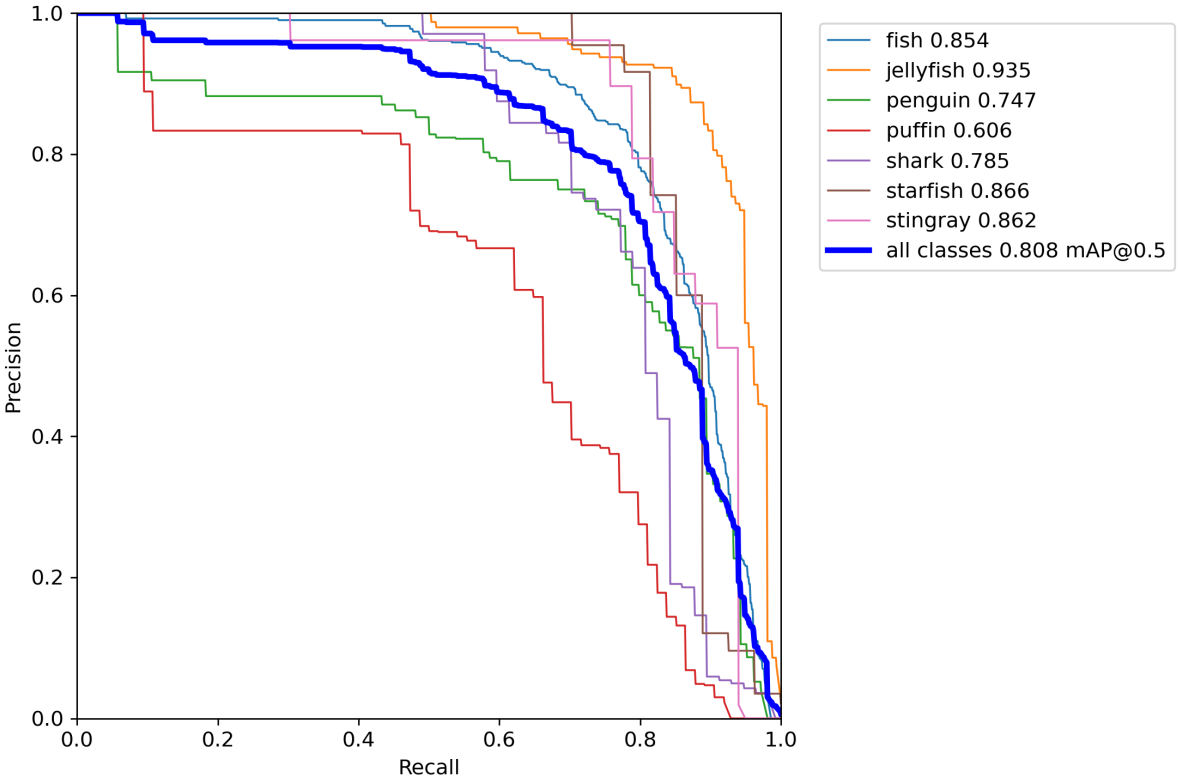

Figure 9: P R curve

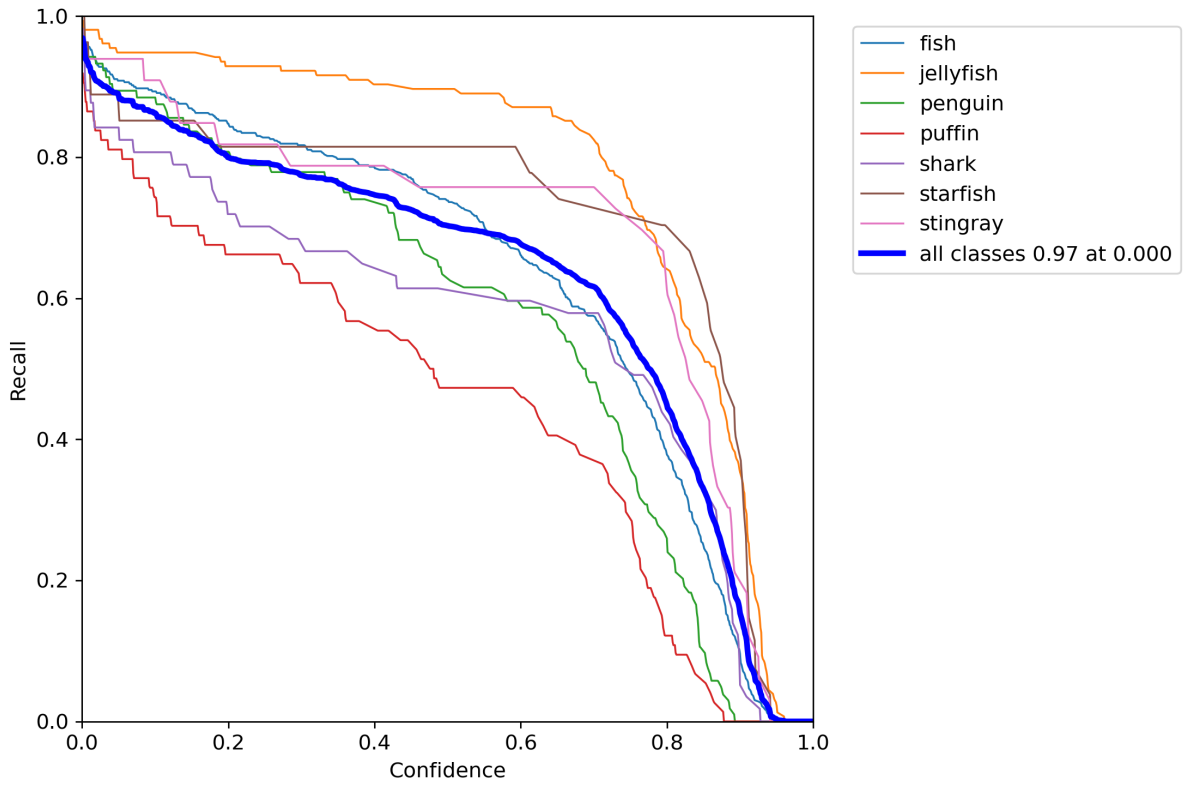

Figure 10: R curve

Results on test dataset

Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:
all	63	584	0.831	0.784	0.849	0.526
fish	63	249	0.836	0.675	0.795	0.466
jellyfish	63	154	0.757	0.929	0.908	0.555
penguin	63	82	0.85	0.707	0.82	0.358
puffin	63	35	0.736	0.557	0.651	0.269
shark	63	38	0.819	0.842	0.878	0.63
starfish	63	11	0.823	0.849	0.942	0.665
stingray	63	15	1	0.93	0.947	0.736

Figure 11: Performance on test dataset- The last column gives average of mAP at 10 IoU thresholds from 0.5 to 0.95 with a step size of 0.05

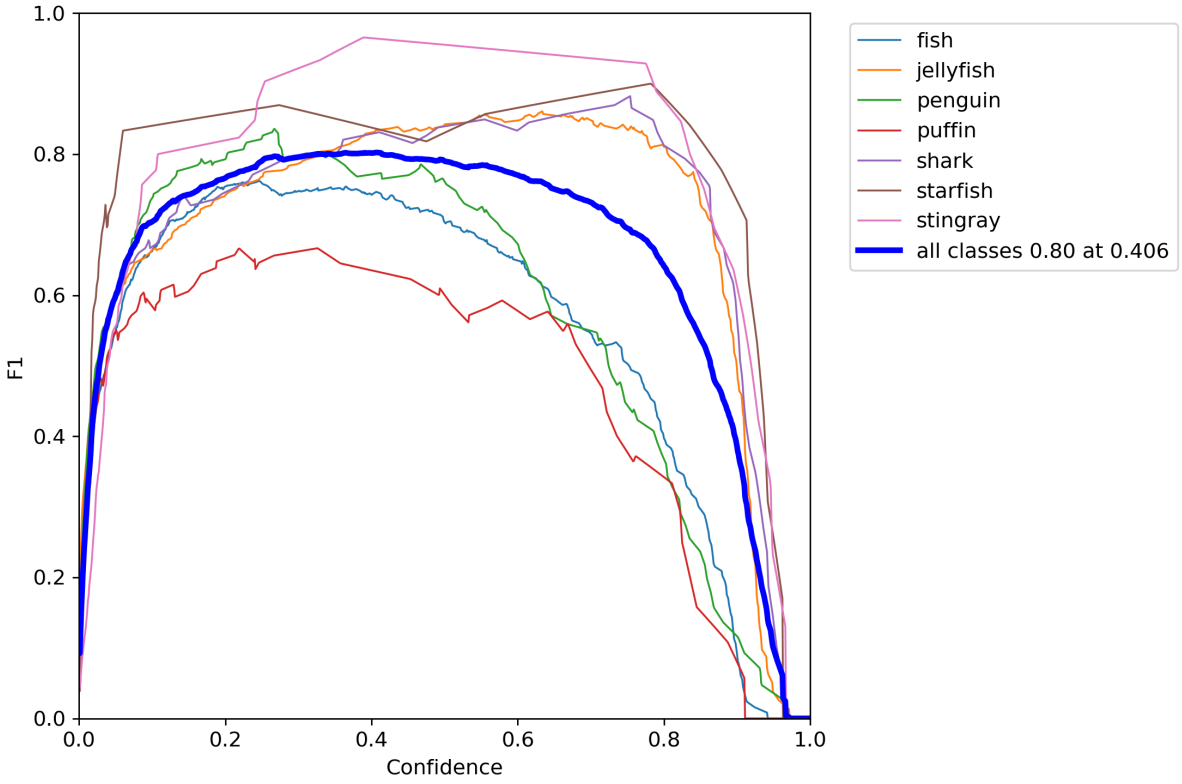

Figure 12: F1 curve

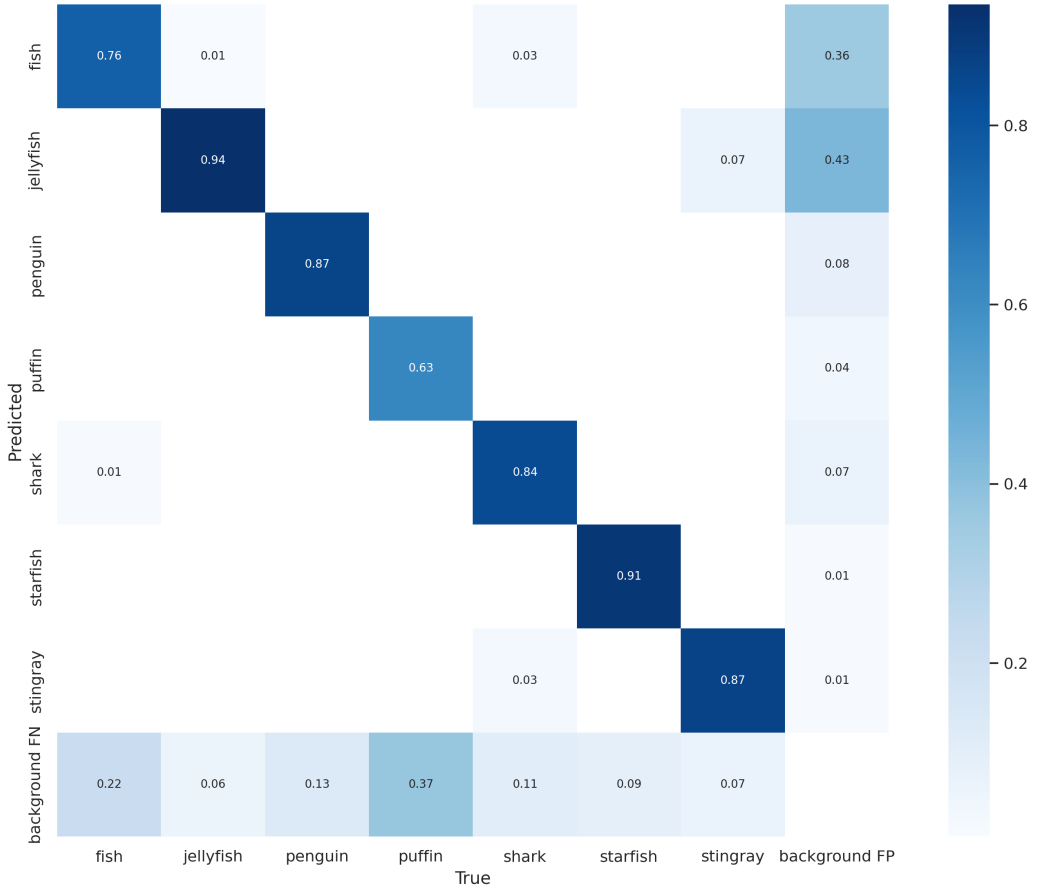

Figure 13: Confusion matrix

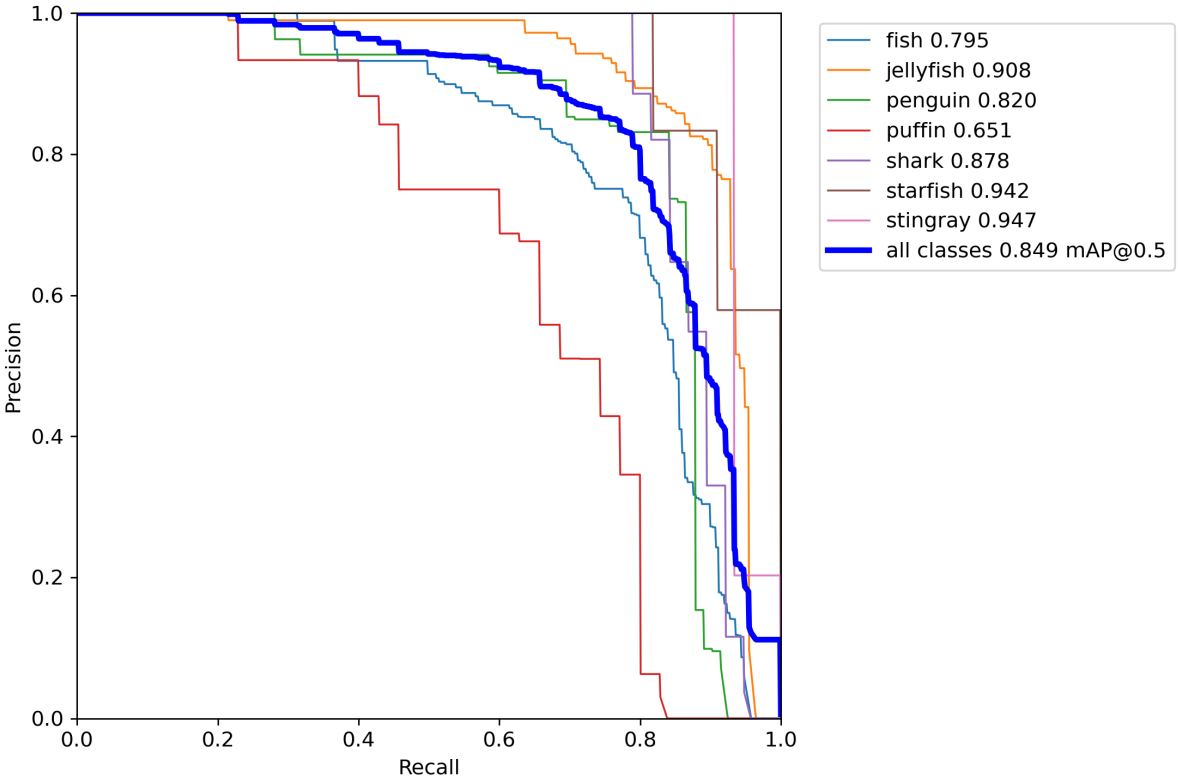

Figure 14: P R Curve

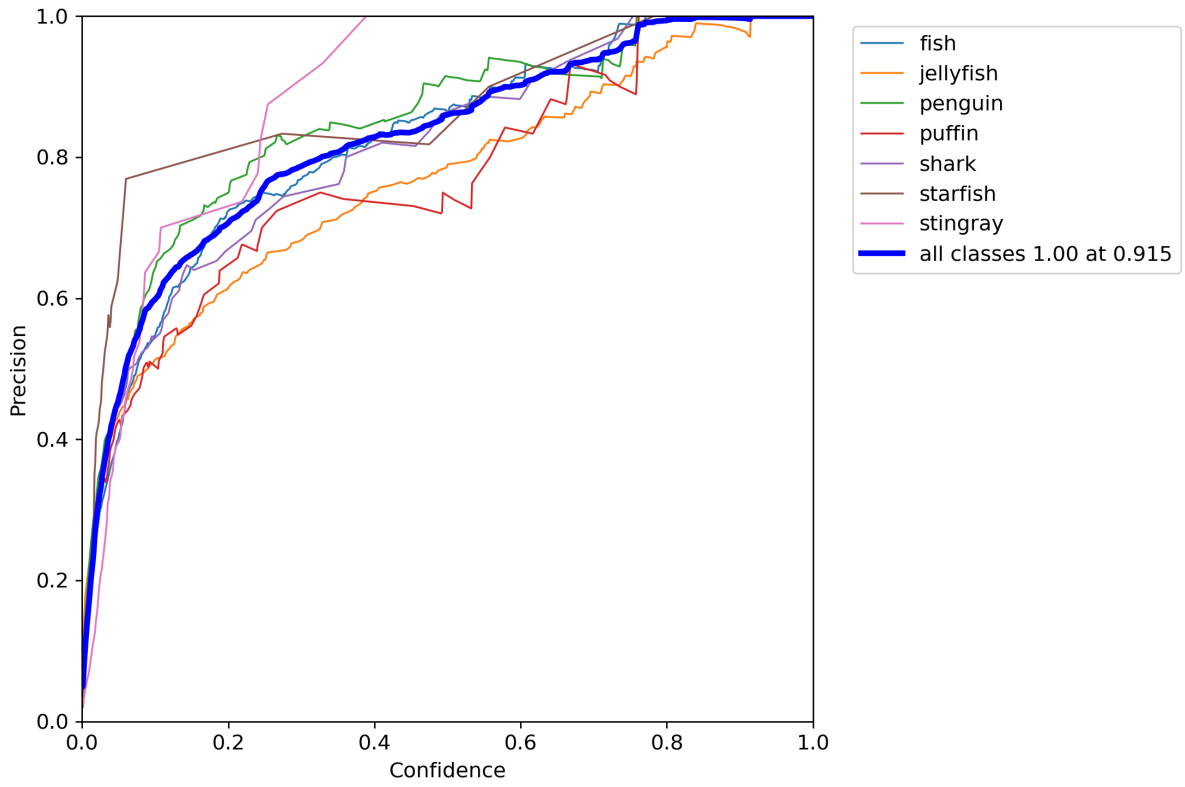

Figure 15: P curve

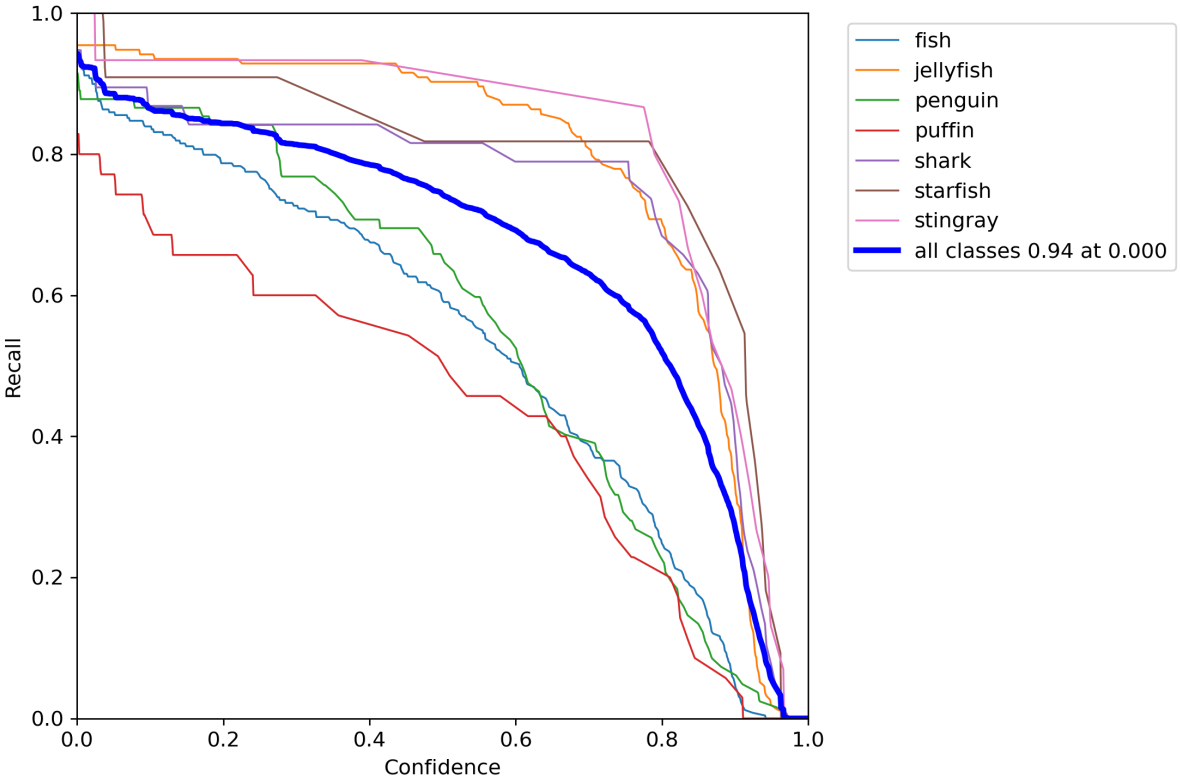

Figure 16: R curve

Qualitative Analysis on test dataset

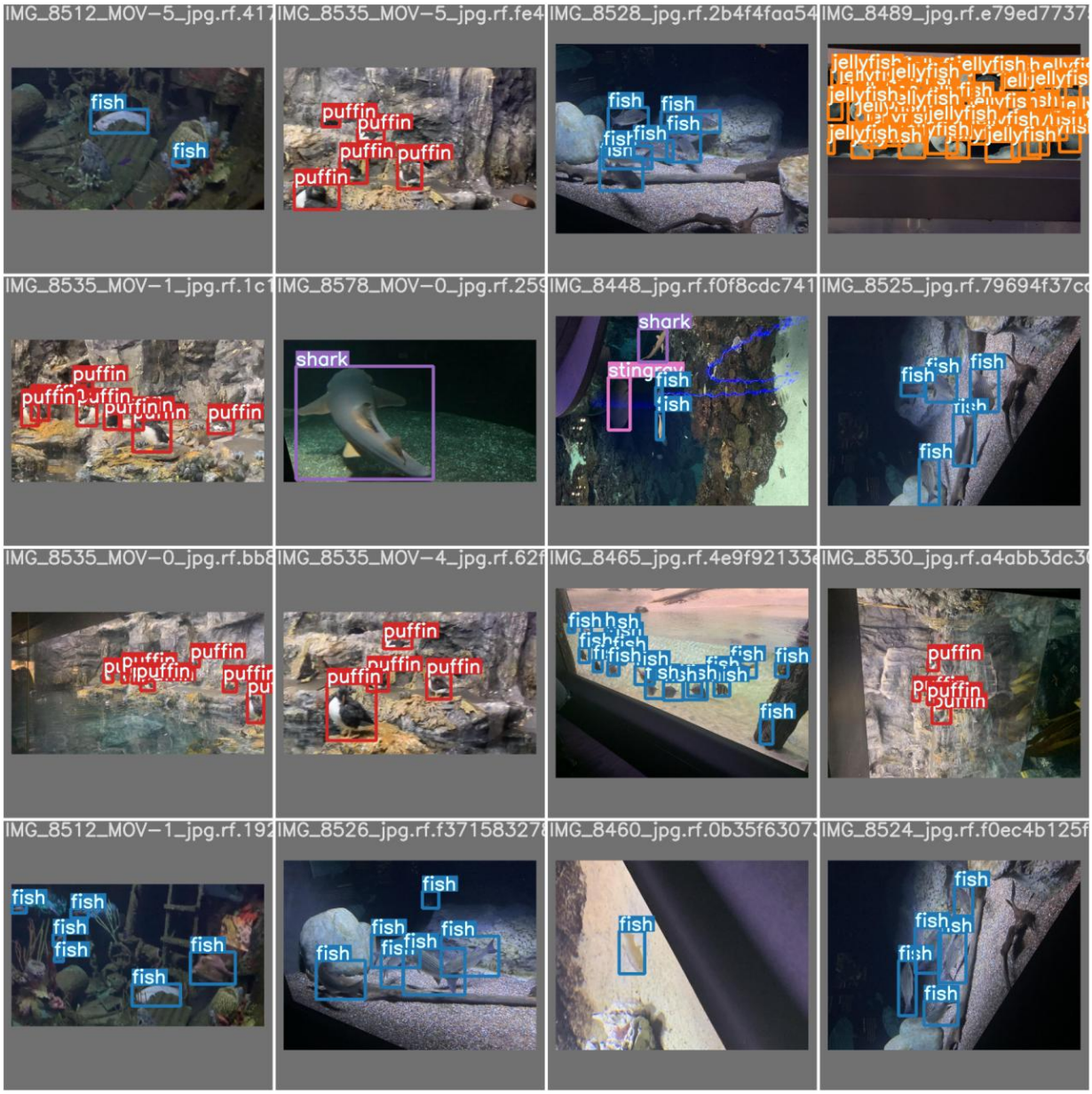

Figure 17: Test- Batch 0 ground truth labels

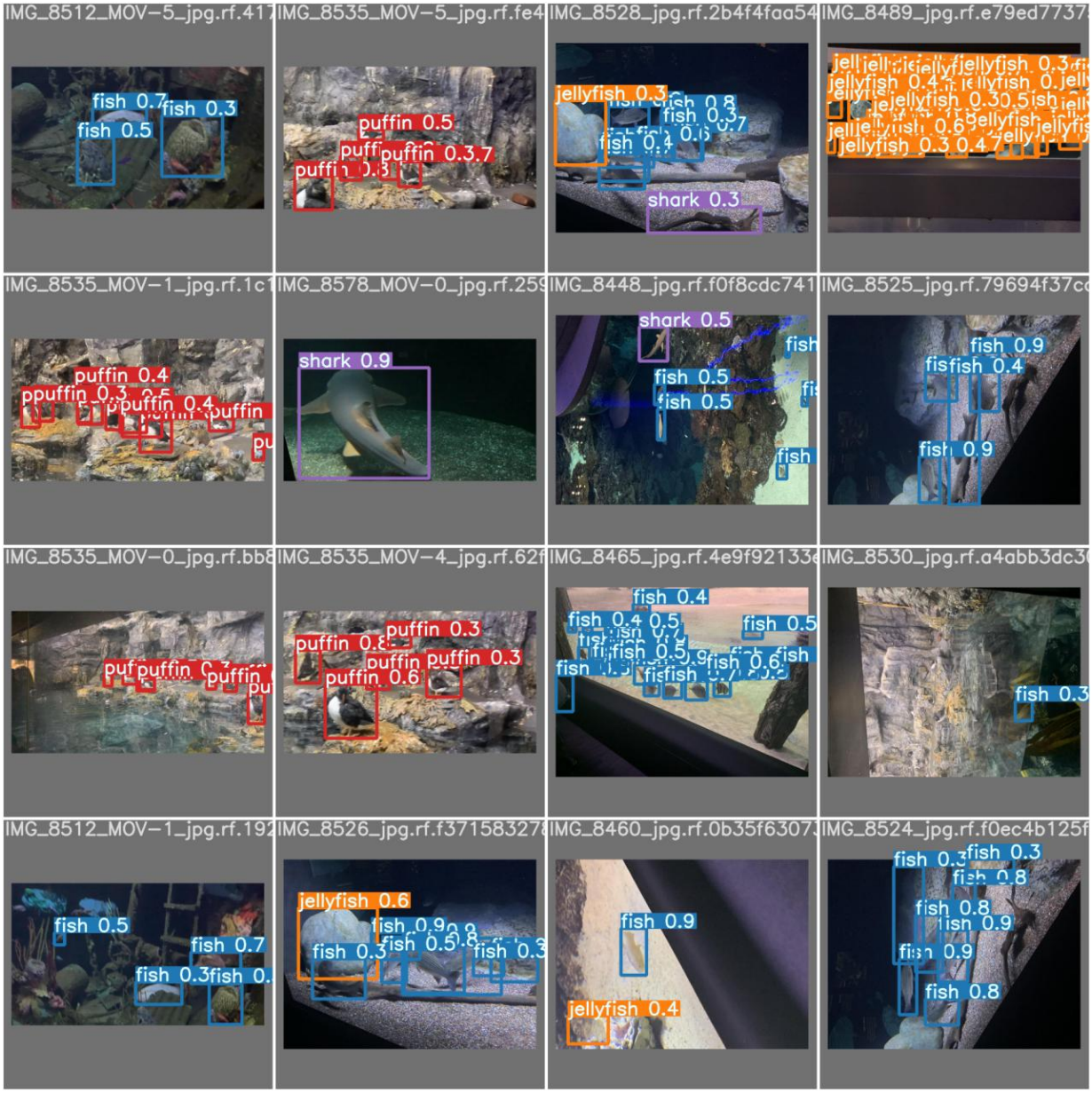

Figure 18: Test- Batch 0 predicted labels

Figure 19: Test- Batch 1 ground truth labels

Figure 20: Test- Batch 1 predicted labels

Figure 21: Test- Batch 2 ground truth labels

Figure 22: Test- Batch 2 predicted labels

Appendix

Problem 1

Transform

To compute an 8x8 Discrete Cosine Transform (DCT) for every non-overlapping block in the input grayscale image, we follow the following steps:

1. Divide image into blocks/patches: We divide the input grayscale image into non-overlapping blocks of size 8x8 using the 'patchify' function, which generates patches from the image.

- Divides an input image into smaller, non-overlapping patches of a specified size.
- Iterates through the image and extracts patches starting from the top-left corner.
- Mathematically represented as $P_{i,j} = I[i : i + h, j : j + w]$, where $P_{i,j}$ denotes the i^{th} patch starting at position (i, j), and h and w are the height and width of the patches, respectively.

Algorithm 1 Patchify Function

```
function PATCHIFY(image, h, w)
    for i ← 0 to image.shape[0] step h do
        for j ← 0 to image.shape[1] step w do
            yield image[i : i + h, j : j + w]
        end for
    end for
end function
```

2. Discrete Cosine Transform (DCT): We define a function 'dct' to compute the 2D DCT matrix of size 8x8. This function computes the DCT matrix using the formula specified for the DCT coefficients.

- Converts spatial-domain image data into frequency-domain representation.
- Applied to image patches to separate image information into different frequency components.
- Mathematically represented as

$$F(u,v) = \frac{1}{\sqrt{N}} C(u)C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} P(x,y) \cos\left[\frac{(2x+1)u\pi}{2N}\right] \cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

  where $F(u, v)$ represents the (u, v)th coefficient of the DCT, $P(x, y)$ represents the pixel intensity, and $C(u)$ and $C(v)$ are normalization constants.

3. Apply DCT to Patches: We iterate through each patch obtained from the image, compute the DCT matrix using the 'dct' function, and apply the DCT transformation to the patch. The resulting DCT coefficients for each patch are stored in a list.

Quantization

Quantization is a crucial step in image compression, where the goal is to reduce the precision of the DCT coefficients to achieve higher compression ratios. In this process, each DCT coefficient in an 8×8 block is divided by a corresponding value in the quantization matrix, and then rounded to the nearest integer. This quantization step results in loss of information, but it allows for more efficient encoding of the image data.

Algorithm 2 Discrete Cosine Transform (DCT) Function

function DCT(n)
    $c \leftarrow \text{zeros}(n, n)$
    for $i \leftarrow 0$ **to** n **do**
        for $j \leftarrow 0$ **to** n **do**
            if $i = 0$ **then**
                $c[i, j] \leftarrow \sqrt{\frac{1}{n}}$
            else
                $c[i, j] \leftarrow \sqrt{\frac{2}{n}} \cdot \cos\left(\frac{(2j+1) \cdot i \cdot \pi}{2n}\right)$
            end if
        end for
    end for
    return c
end function

The given quantization matrix Q is used to quantize each DCT coefficient in an 8×8 block. It has the following values:

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

To quantize a DCT coefficient $x(i, j)$ in the (i, j)th position of the 8×8 block, the quantized index $y(i, j)$ is computed using the formula:

$$y(i, j) = \text{round}\left(\frac{x(i, j)}{Q(i, j)}\right)$$

where $Q(i, j)$ is the corresponding value in the quantization matrix. Additionally, to reconstruct the coefficient $x(i, j)$ from its quantized index $y(i, j)$, the formula is:

$$\hat{x}(i, j) = y(i, j) \times Q(i, j)$$

This quantization process introduces quantization error, which is one of the main sources of image

Algorithm 3 Quantization

function QUANTIZE(DCT coefficient, Q)
    Quantized index $\leftarrow \left\lfloor \frac{\text{DCT coefficient}}{Q} + 0.5 \right\rfloor$
    return Quantized index
end function
function RECONSTRUCT(Quantized index, Q)
    Reconstructed coefficient $\leftarrow$ Quantized index $\times Q$
    return Reconstructed coefficient
end function

distortion in lossy compression algorithms. The choice of the quantization matrix determines the

trade-off between compression ratio and image quality, with higher quantization values leading to higher compression but lower image fidelity.

0.0.1 Lossless encoding

The `encoding` function is used to encode the quantized values obtained after quantization of Discrete Cosine Transform (DCT) coefficients. It follows a variable-length encoding scheme, where shorter codes represent smaller quantized values and longer codes represent larger quantized values.

The function takes a single value, *value*, as input, which represents the quantized DCT coefficient.

Algorithm 4 Encoding Function

```
function ENCODING(value)
    steps ← [0, 1, 3, 7, 15, 31, 63, 127, 255]
    value ← int(value)
    code ← 0
    if |value| = 0 then
        code ← 0
    else
        for i in range(8) do
            if steps[i] < |value| ≤ steps[i + 1] then
                num ← i + 1
                code ← num × [1]
                code ← code + [0]
                if value > 0 then
                    code ← code + bin(|value|)[2 :]
                else
                    code ← code + str(bin(steps[i + 1] + value)[2 :].rjust(i + 1,′ 0′))
                end if
                code ← int(″.join(str(i) for i in code))
            end if
        end for
    end if
    return code
end function
```

In this algorithm, *steps* represents the upper bounds for each quantization range. The value of *value* is first converted to an integer to ensure compatibility with subsequent operations. The function then iterates over the quantization ranges defined by *steps*, and for each range, it determines the appropriate number of bits needed to represent the value. Based on this, it constructs a binary code representing the encoded value.

The encoded value is returned as a string, representing the compressed representation of the quantized DCT coefficient.