E9 246 Advanced Image Processing

Assignment 4

Name**: Niladri Dutta**

Course**: MTech AI**

SR **no.: 23112**

1. JPEG Implementation.

1. Transform: Given the grayscale image, we first break it into non-overlapping blocks of size 8x8 and then perform Discrete Cosine Transform (DCT) on each of the blocks to obtain another block of same size. This step helps to identify the components which can/cannot be perceived by the human eye easily and hence will help to throw them out in the subsequent steps for compression. The DCT basically decomposes these image blocks to 64 basis signals and its 8x8 output contains the amplitudes of those signals. Overall, in this step, just a transformation is done and hence no compression is achieved since the same data in represented in a different format.

2. Quantization: In this step, we divide each 8x8 block by the quantization matrix add 0.5 to each entry and round them down to the nearest integer. In the quantization matrix, values are generally higher down the diagonal as we can afford to lose some precision for the high frequency components as they are not perceived as good as the low frequency components by human eye. Then taking floor of the entries helps to make the values discrete which helps in the next encoding step.

3. Lossless source encoding: The quantized matrix obtained has the property that its first few entries (low frequency) are non-zero and rest of the higher frequency components are zero. So, first we do zig-zag encoding to ensure that for every block almost all of the zeros come at the last and concatenate the data of all the blocks. Then we perform run length encoding (RLE) which effectively stores the value and its consecutive number of occurrences/ run length. Thus, RLE helps to compress the image to a large extent and finally we perform the given encoding scheme to form the final output file.

4. Reconstruction: After encoding is done, we'll now decode the file to reconstruct the image. This is done by just applying the inverse of all the operations done above in reverse order and then we'll obtain the compressed image after reconstruction.

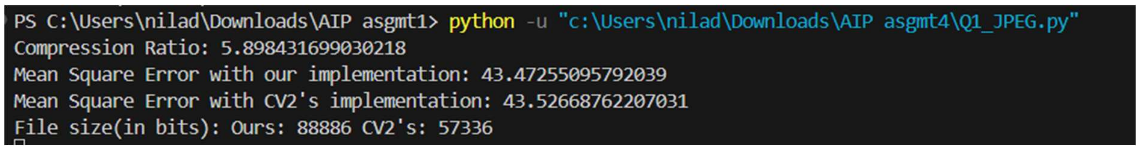
Results:

1. Compression ratio (Image size/Output file size): **5.9**

2. MSE (between original and reconstructed): **43.47**

3. File sizes in bits (with equal MSE): **Ours: 88886 and CV2's:57336**

In order to keep both the MSE's same, CV2's hyperparameter was set at **50%.**



```
PS C:\Users\nilad\Downloads\AIP asgmt1> python -u "c:\Users\nilad\Downloads\AIP asgmt4\Q1_JPEG.py"
Compression Ratio: 5.898431699030218
Mean Square Error with our implementation: 43.47255095792039
Mean Square Error with CV2's implementation: 43.52668762207031
File size(in bits): Ours: 88886 CV2's: 57336
```

2. Comparison of Perceptual quality measures.

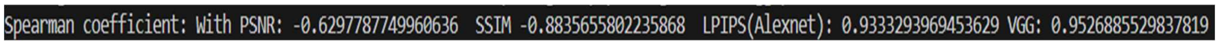
1. PSNR: This is an equivalent form of MSE and does not take into account the structure of images or the type of error signal between original and distorted. For PSNR, error at every location has same weightage which is absolutely not the way in which humans perceive image quality hence it is a very poor perceptual quality measure.

2. SSIM: This measure takes into account the local image structure and decides the similarity between them based on similarity of brightness, contrast and structure of the local patches. This index takes into account the statistical factors like mean, variance and covariances of the local patches in order to calculate all the three measures mentioned. Since it takes into account the local structure of the image, it is quite better than PSNR.

3. LPIPS: It uses deep neural networks to learn feature representations directly from images. Since it uses human labelled similarity scores, it has quite better performance compared to the other two indices. Since it uses deep neural networks, it can workout the complex mappings required to decide the quality of an image. This was not possible in SSIM since it uses only up to second order statistics and that too is handcrafted to have simple yet effective results. Now VGG being a bigger and more complex network performs little better than Alexnet. But overall, LPIPS is quite better than the other classical measures.

Results:

Spearman correlation coefficient with dmos: PSNR: -0.63, SSIM: -0.884, LPIPS(Alex): 0.933, LPIPS(VGG): 0.953



```
Spearman coefficient: With PSNR: -0.6297787749960636  SSIM -0.8835655802235868  LPIPS(Alexnet): 0.9333293969453629 VGG: 0.9526885529837819
```

3. YOLO Object Detection.

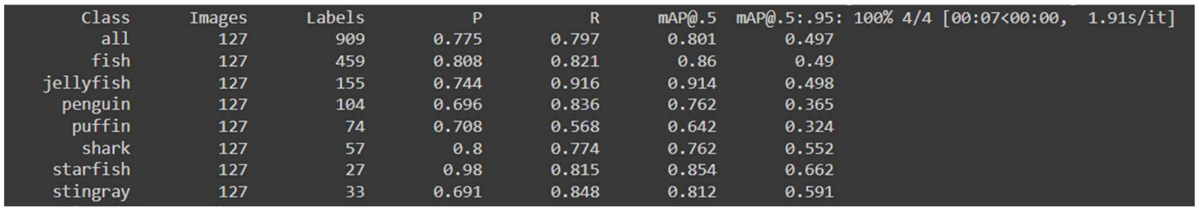
YOLO-v7 vs YOLO-v1

Architecture: Compared to YOLO-v1 the YOLO-v7 ha the following architectural changes: In the backbone of the network, it uses extended efficient layer aggregation network (E-ELAN). The design of the E-ELAN was guided by the analysis of factors that impact accuracy and speed, such as memory access cost, input/output channel ratio, and gradient path. The second architectural reform was presented as compound model scaling with the aim to cater for a wider scope of application requirements, i.e., certain applications require accuracy to be prioritized, whilst others may prioritize speed.

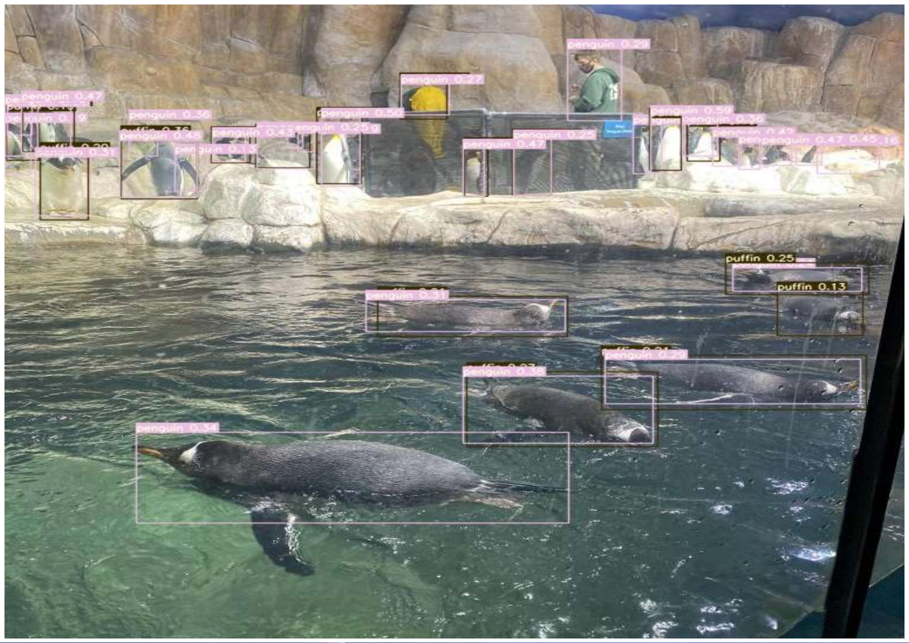
Loss: The loss function has not changed in the latest version of YOLO.

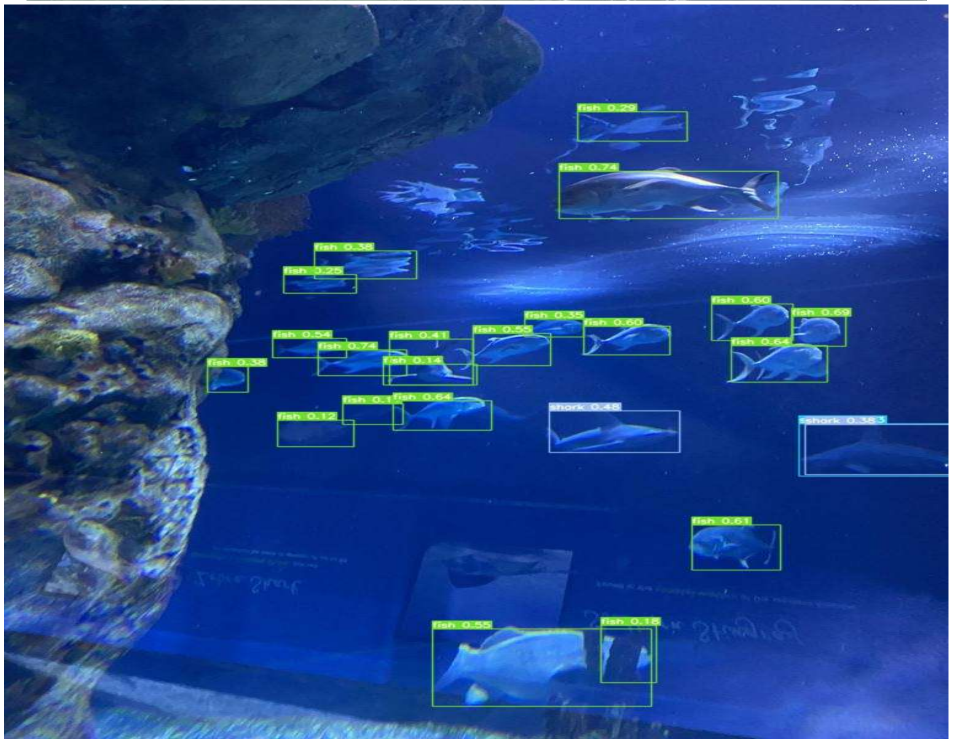
Computation complexity: YOLO-v7 is at least 3x faster than YOLO-v1 due to various factors like use of batch normalization, hardware accelerators, better backbones, etc. All YOLO-v7 variants surpassed the compared object detectors in accuracy and speed in the range of 5–160 FPS.

Results:



```
     Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 4/4 [00:07<00:00,  1.91s/it]
       all         127         909       0.775       0.797       0.801       0.497
      fish         127         459       0.808       0.821        0.86        0.49
 jellyfish         127         155       0.744       0.916       0.914       0.498
   penguin         127         104       0.696       0.836       0.762       0.365
    puffin         127          74       0.708       0.568       0.642       0.324
     shark         127          57         0.8       0.774       0.762       0.552
  starfish         127          27        0.98       0.815       0.854       0.662
  stingray         127          33       0.691       0.848       0.812       0.591
```


penguin 0.29
penguin 0.27
penguin 0.47
penguin 0.36
penguin 0.56
penguin 0.59
penguin 0.31
penguin 0.47
penguin 0.25
puffin 0.25
puffin 0.13
penguin 0.31
penguin 0.29
penguin 0.38
penguin 0.34


fish 0.29
fish 0.74
fish 0.38
fish 0.25
fish 0.60
fish 0.69
fish 0.64
fish 0.35
fish 0.60
fish 0.55
fish 0.54
fish 0.74
fish 0.41
fish 0.14
fish 0.38
fish 0.64
fish 0.12
shark 0.48
shark 0.38
fish 0.61
fish 0.55
fish 0.18