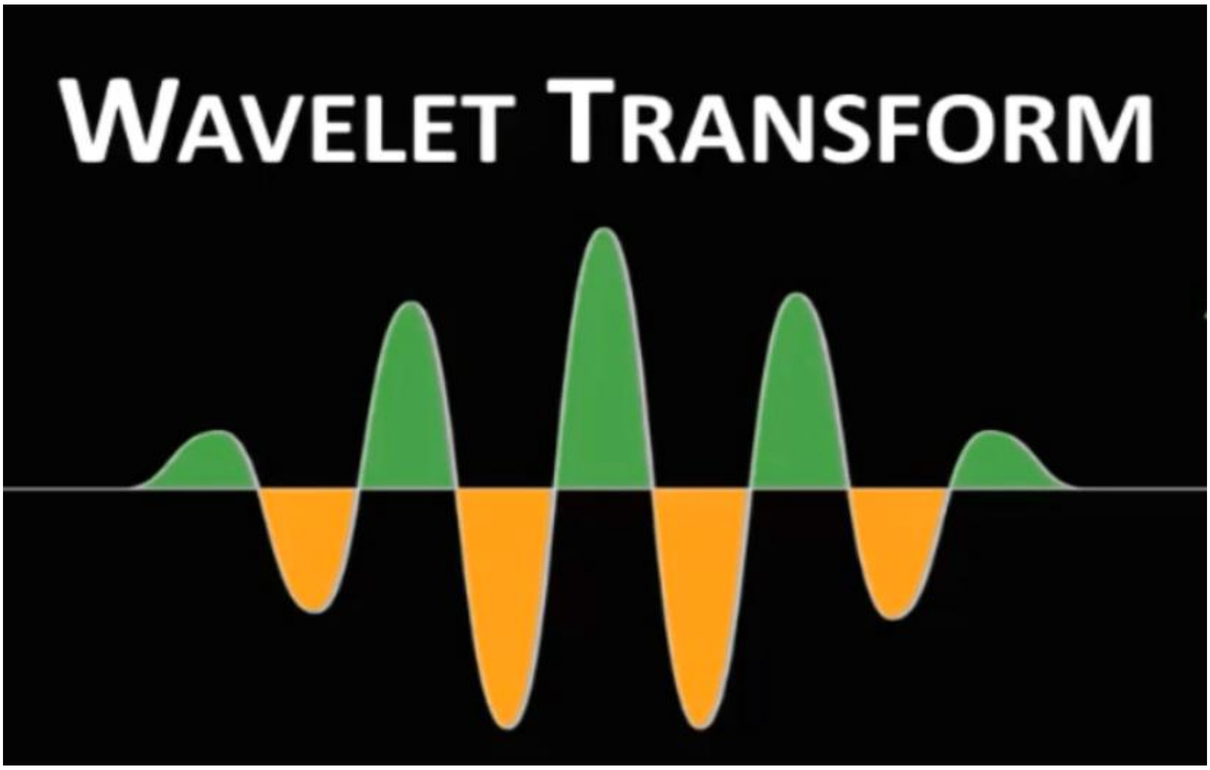
ELL 786 Report
Assignment-1



Rohan Mahala 2019MT60760

Ayush Verma 2019MT60749

Question 1.

In this experiment you will implement Huffman Coding compression algorithm on a text file.

Input : 1KB input text file.

```
hi.
we are rohan and ayush from mathematics and computing.
this is the sample text file for the question one of the assignment one.
just to be sure i will enlist all the alphabets in the following line.
abcdefghijklmnopqrstuvwxyz.
we will encode this file and then decode it to show that huffman is a lossless
technique used for data compression.
we have put good amount of efforts in this assignment and expect to obtain good
grades.
thank you.
```

Figure 1 : Input text file used for encoding and decoding

Step-1 : First we read the input text file and generate frequency corresponding to each character in the file.

```python
import random

#opening a file for generating frequencies corresponding to characters of the sample file.
file = open("freq.txt" , 'w')

#function for manipulating the characters
def getchar(char):
  if(char.isalpha()):
    char = char.lower()
    return char
  elif(char == " "):
    return "space"
  elif(char == "."):
    return "period"
  elif(char == "\n"):
    return "newline"
  return None

#if rand==1 then this function will generate frequency randomly
#if rand==0 then this function will generate frquency corresponding to the sample text file.
def generate_freq(rand):
  if(rand):
    #freq for alphabets
    for i in range(26):
      fq = random.randint(30,1000)
      add = chr(i + ord('a')) + " " + str(fq) + "\n"
      file.write(add)
```

Figure 2 : Code of freq_generator

```
    #freq for period
    fq = random.randint(30,1000)
    add = "period" + " " + str(fq) + "\n"
    file.write(add)

    #freq for newline
    fq = random.randint(30,1000)
    add = "newline" + " " + str(fq) + "\n"
    file.write(add)

    #freq for end of message
    fq = random.randint(30,1000)
    add = "EOM" + " " + str(fq) + "\n"
    file.write(add)

else:
    #reading the file for which frequency is to be generated(sample.txt)
    textfile = open("sample.txt" , 'r')

    #making and intiallizing a dictionary
    freqs = dict()
    for i in range(26):
        freqs[chr(i + ord('a'))] = 0
    freqs["space"] = 0
    freqs["EOM"] = 0
    freqs["newline"] = 0
    freqs["period"] = 0

    #iterating through the text file
    while True:
        #reading the file character by character
        char = textfile.read(1)

        #when we reach end of file then char==0, so puttinf frequency of EOM=1 and breaking from loop
        if(not char):
            freqs["EOM"] += 1
            break

        #for alphabets, space, period and newline
        char = getchar(char)
        freqs[char] += 1

    #iterating through the dictionary and writing the (character name+" "+frequency) of each character to the
    for char in freqs:
        add = char + " " + str(freqs[char]) + "\n"
        file.write(add)

file.close()
```

Figure 3 : Code of freq_generator

We see two important functions in the code of freq_generator.

- get_char(char) : This is the function used for determining the type of the char. It can be of the following types :-
    - Alphabet
    - Space
    - Period
    - Newline
    - End of message
- generate_freq(rand) : Function for generating frequency :-
    - rand = 1 : Generate frequency randomly.
    - rand = 0 : Generate frequency of the input text file.

Step-2 : We a code to build the Huffman tree and generate codewords.

```
import heapq

#Making a class for building a huffman tree
#each object of the class has character name(alpha), its frequency(freq), left child(left)
#and right child(right)
class Alphabet:
  def __init__(self , alpha , freq):
    self.alpha = alpha
    self.freq = freq
    self.left = None
    self.right = None

  def __str__(self):
    return self.alpha + " " + str(self.freq)

  def get_freq(self):
    return self.freq

  def get_alpha(self):
    return self.alpha

  def get_left(self):
    return self.left

  def get_right(self):
    return self.right

  def set_right(self,  alphabet):
    self.right = alphabet

  def set_left(self , alphabet):
    self.left = alphabet

  def __lt__(self, other):
    if(self.freq < other.freq):
      return True
    elif(self.freq == other.freq):
      return False
```

Figure 4 : Class Alphabet that will form the nodes of Huffman tree

```
#We build the huffman tree given the freqency list using heaps
def build_huffman_tree(freq1):
  freq = freq1.copy()
  heapq.heapify(freq)

  #we are removeing the all the character with zero frequency, so that it doesn't
  #involve in building the huffmann tree
  while(freq[0].get_freq() == 0):
    heapq.heappop(freq)

  while(len(freq) > 1):
    #getting and popping two elements with the least frquency
    min1 = heapq.heappop(freq)
    min2 = heapq.heappop(freq)

    #combining the elements with the least frquency, and adding it to the heap
    alpha = Alphabet("inode" , min1.get_freq() + min2.get_freq())
    heapq.heappush(freq,alpha)

    alpha.set_left(min1)
    alpha.set_right(min2)
  return freq[0]
```

Figure 5 : Function for building the Huffman tree

```python
#given an empty string(code), root of the huffman tree(root) and an empty  dictionary(codewords)
#this function will generate the codewords corresponding to the characters
def build_codewords(code , root,codewords):

  leaf = True
  if(root.right != None):
    leaf = False
    build_codewords(code + "1" , root.get_right(),codewords)

  if(root.left != None):
    leaf = False
    build_codewords(code + "0" , root.get_left(),codewords)

  #Since huffman code is a prefix code, all the characters for which code is required
  #would be the leaf nodes only.
  if(leaf) :
    codewords[root.get_alpha()] = code
```

Figure 6 : Function for generating codewords from the Huffman tree

Step-3 : Encoding the input file by first generating the frequency, building the Huffman tree, generating codewords and finally encoding the text file into encoded_text.txt.

```python
from freq_generator import getchar , generate_freq
from Rohan_Ayush_A1 import build_codewords, build_huffman_tree,Alphabet


#this file contains the text to be encoded
file = open("sample.txt" , "r")

#this will generate freqency of all the characters in freq.txt
generate_freq(0)

# Reading the frequency generated
file_freq = open("freq.txt" , 'r')
freq = []

#Creating a list of alphabet objects
for line in file_freq:
  alphafreq = list(line.split(" "))
  alpha = alphafreq[0]
  frequency = int(alphafreq[1])
  freq.append(Alphabet(alpha , frequency))
file_freq.close()

#Building the huffman tree from the list of alphabet objects
root = build_huffman_tree(freq)
print("Huffman tree built")

#Creating an empty dictionary for storing the codewords
codewords = dict()

#Building codewords from the huffman tree that we built and storing them in codewords dictionary
build_codewords("" , root,codewords)
```

Figure 7 : Generating the frequency, building the Huffman tree and generating the codewords

```
#Building codewords from the huffman tree that we built and storing them in codewords dictionary
build_codewords("" , root,codewords)

#Writing the codewords corresponding to each character in a file(codewords.txt)
result = open('codewords.txt', 'w')
for i in freq:
  if(i.get_alpha() in codewords):
    add = i.get_alpha() + " " + codewords[i.get_alpha()] + "\n"
    result.write(add)
result.close()
print("Codewords generated in codewords.txt")

#Encoding the the file(sample.txt) using the codewords generated and writing it to encoded_text.txt
file_out = open("encoded_text.txt", "w")
while True:
  char = file.read(1)
  if(not char):
    file_out.write(codewords["EOM"])
    break
  char = getchar(char)
  file_out.write(codewords[char])

print("encoding of sample.txt done in encoded_text.txt")
file_out.close()
file.close()
```

Figure 8 : Writing the codewords codewords.txt and then finally encoding the input file using the codewords

Step-4 : Decoding the encoding generated by the encoder.

```
from Rohan_Ayush_A1 import build_codewords, build_huffman_tree,Alphabet
from freq_generator import getchar , generate_freq


#this will generate freqency of all the characters in freq.txt
generate_freq(0)

# Reading the frequency generated
file_freq = open("freq.txt" , 'r')
freq = []

#Creating a list of alphabet objects
for line in file_freq:
  alphafreq = list(line.split(" "))
  alpha = alphafreq[0]
  frequency = int(alphafreq[1])
  freq.append(Alphabet(alpha , frequency))
file_freq.close()

#Building the huffman tree from the list of alphabet objects
root = build_huffman_tree(freq)


#File to be decoded
encoded_file = open("encoded_text.txt" , 'r')
```

Figure 9 : Regenerating the Huffman tree built in step-2 and reading the encoded text from step-3.

```python
#File in which decoded text will be written
decoded_file = open("decoded_text.txt" , 'w')

#decoding
node = root
while True:
  #reading the file to be decoded, char by char
  char = encoded_file.read(1)

  #if we reach end of file, we break out of the loop
  if(not char):
    break

  # char=1 means right child(that's how the code has been generated)
  if(char == '1'):
    node = node.get_right()
  # char=0 means left child
  elif(char == '0'):
    node = node.get_left()

  #If nodename != 'inode', means it is leaf node and hence it corresponds to a character
  if(node.get_alpha() != 'inode'):
    #handling new line
    if(node.get_alpha() == "newline"):
      decoded_file.write("\n")
    #Handling end of message
    elif(node.get_alpha() == "EOM"):
      break
    #handling space
    elif(node.get_alpha() == "space"):
      decoded_file.write(" ")
    #handling period
    elif(node.get_alpha() == "period"):
      decoded_file.write(".")
    #handling alphabets
    else:
      decoded_file.write(node.get_alpha())
    #reseting the node to the root of the huffman tree
    node = root
```

Figure 10 : Finally decoding the encoded_text.txt to decoded_text.txt

```
hi.
we are rohan and ayush from mathematics and computing.
this is the sample text file for the question one of the assignment one.
just to be sure i will enlist all the alphabets in the following line.
abcdefghijklmnopqrstuvwxyz.
we will encode this file and then decode it to show that huffman is a lossless
technique used for data compression.
we have put good amount of efforts in this assignment and expect to obtain good
grades.
thank you.
```

Figure 11 : decoded_text.txt

Conclusions :

- Number of bits required before Huffman encoding :-
  - Size of the input text file = 452 bytes = 452*8 bits
  - 3616 bits required before Huffman encoding.
- Number of bits required after Huffman encoding :-
  - Length of encoding of input text = 1925
  - 1925 bits required after Huffman coding.
- Compression ratio = $\frac{\text{No.of bits before encoding}}{\text{No.of bits after encoding}}$

$$= \frac{3616}{1925}$$

$$= 1.878$$

$$\approx 2$$

- Huffman encoding compresses the input text file to approximately almost half of the original size.
- Also the decoded_text.txt is exactly the same as original input text file sample.txt. This means that there is no loss of data and hence, Huffman coding is indeed a lossless compression technique.

Question 2.

In this experiment, you will implement Discrete Wavelet Transform (DWT) on an image.

Part-1 : Two dimensional Discrete Wavelet Transformation

Step-1 : Computing the n-scale two dimensional DWT with respect to haar wavelets of an input image.

```python
import pywt,numpy as np
from PIL import Image

#------------------------------------------------------------------#
# Convert a image into grayscale and load a image into numpy array #
#------------------------------------------------------------------#

def load_image(img):
  img = Image.open(img).convert('L')
  img.load()
  data = np.asarray(img, dtype="int32" )
  return data

def save_image(array, img) :
  array = array.astype(np.uint8)
  im = Image.fromarray(array)
  im.save(img)
  return im

img = load_image("Images/sample.webp")
# print(img)
# print(img.shape)
# im_out = save_image(img, "sample_out.webp")
# im_out.show()

#----------------------------------------------------------------#
#                     Multilevel Decomposition                   #
#----------------------------------------------------------------#
def Haar2D(img,lev):
  coeffs = pywt.wavedec2(img, 'haar', level=lev)
  return coeffs
```

Figure 12 : Function for computing the "lev" scale two dimensional DWT wrt Haar wavelets

Step-2 & 3 : Using an image to generate 3-scale two dimensional DWT and then reconstructing the original image using inverse two dimensional DWT.

```
#------------------------------------------------------------------#
#                     Inverse  DWT                                 #
#------------------------------------------------------------------#
def InvHaar2D(coeffs):
  reimg = pywt.waverec2(coeffs, 'haar')
  return reimg

#------------------------------------------------------------------#
#                       3-scale DWT                                #
#------------------------------------------------------------------#

# img = np.array([[1,2],[3,4]])
coeffs = Haar2D(img , 3)
'''(cA3,(cH3,cV3,cD3),(cH2,cV2,cD2),(cH1,cV1,cD1))
 where cA3 is a approximation coefficient and cHi,cVi,cDi's are detail coefficient'''

reimg = InvHaar2D(coeffs)
re_im_out = save_image(reimg, "Images/Output/sample_rec.webp")
# re_im_out.show()
```

Figure 13 : Function for inverse DWT, constructing approximation & detail coefficient and reconstructing the original image using inverse DWT

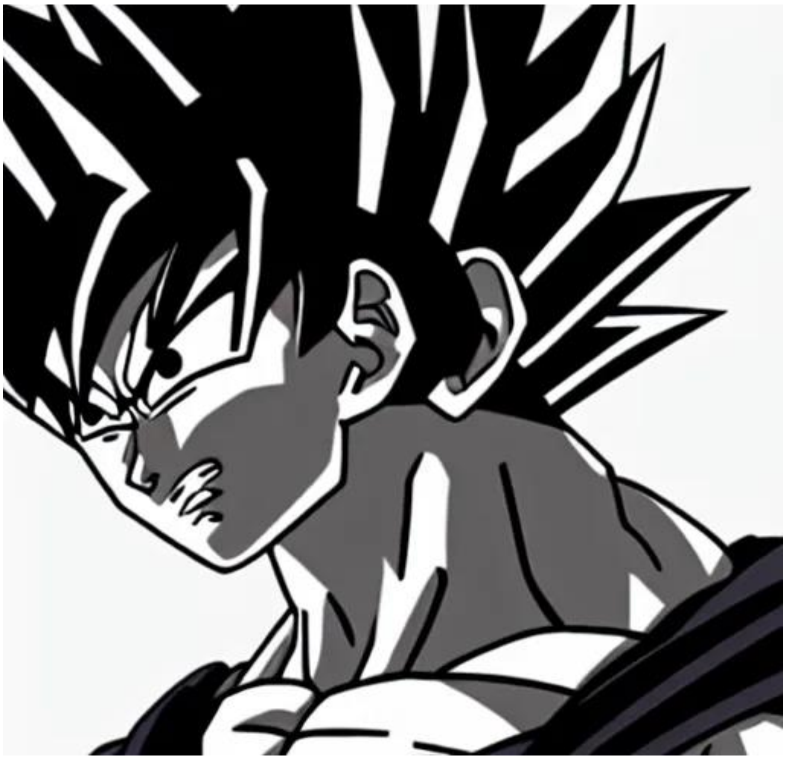


Figure 14 : Original Image

Figure 15 : Reconstructed Image

Step-4 : Scaling the the detail coefficients in the step 2.

```python
#----------------------------------------------------------------#
#            Scaling the detail coefficient                      #
#----------------------------------------------------------------#

'''In the case of the Haar wavelet, the detail coefficients are typically
scaled by a factor of 0.5. This is because the Haar wavelet
has a simple structure, and scaling the detail coefficients by a factor
of 0.5 ensures that
the energy balance is maintained in the transformed image.'''

def scale_detail_coefficients(coeffs,scaling_factor):
  for i in range(1,len(coeffs)):
    coeffs[i] = [d*scaling_factor for d in coeffs[i]]
  return coeffs

coeffs = scale_detail_coefficients(coeffs,0.5)
reimg = InvHaar2D(coeffs)
# print(reimg)
re_im_out = save_image(reimg, "Images/Output/scaled_image.webp")
# re_im_out.show()
```

Figure 16 : Scaling the detail coefficients by a factor of 0.5

Figure 17 : Scaling the detail coefficients by a factor of 0.5

Note :- All the images for this part and the subsequent parts are stored in the images folder.

Part-2 : Wavelet Transform Modifications

Step-1 : Reducing the size of our input image in half by row-column deletion, and padding it with 0s to obtain a 512 × 512 array.

```
#-----------------------------------------------------------------#
#            Wavelet Transform Modifications                      #
#-----------------------------------------------------------------#

'''Reducing the size of our input image in half by row-column deletion, and
padding it with 0s to obtain a 512 × 512 array.'''

def reduce(img):
  ''' We are reducing the image by half by taking every alternate row
  and coloumn and then paddding with zeros to make it a 512*512 size'''

  img = img[::2, ::2]
  (r,c) = img.shape
  padding = np.zeros((512, 512))
  padding[:r, :c] = img
  return padding

def zeroA(coeffs):
  coeffsA = coeffs.copy()
  coeffsA[0] = coeffsA[0]*0
  return coeffsA

def zerodetail(coeffs , detail):
  #detail = 0 --> horizonatl
  #detail = 1 --> vertical
  #detail = 2 --> diagonal

  coeffsdetail = coeffs.copy()
  for i in range(1,len(coeffsdetail)):
    coeffsdetail[i] = list(coeffsdetail[i])
    coeffsdetail[i][detail] = coeffsdetail[i][detail]*0
  return coeffsdetail

reduced_img = reduce(img)
re_im_out = save_image(reduced_img, "Images/Output/sample_reduced_and_padded.webp")
# re_im_out.show()
```

Figure 18 : Code for reducing the size of image and padding it with 0s, zeroing the approximate and detail coefficient.

Figure 19 : Reduced and Padded image

Scale = 1 :

```
#----------------------------------------#
#             Scale = 1                  #
#----------------------------------------#

coeffs = Haar2D(reduced_img , 1)

#zeroing approximatioin coefficients
coeffsA = zeroA(coeffs)
reimg = InvHaar2D(coeffsA)
re_im_out = save_image(reimg, "Images/Output/sample_scale1_reduced_and_padded_zeroA.webp")
# re_im_out.show()

#zeroing horizontal coefficients
coeffsdetail1 = zerodetail(coeffs,0)
reimg = InvHaar2D(coeffsdetail1)
re_im_out = save_image(reimg, "Images/Output/sample_scale1_reduced_and_padded_zeroH.webp")
# re_im_out.show()

#zeroing vertical coefficients
coeffsdetail2 = zerodetail(coeffs,1)
reimg = InvHaar2D(coeffsdetail2)
re_im_out = save_image(reimg, "Images/Output/sample_scale1_reduced_and_padded_zeroV.webp")
# re_im_out.show()

#zeroing both horizontal and vertical coefficients
coeffsdetail12 = zerodetail(coeffs,0)
coeffsdetail12 = zerodetail(coeffsdetail12,1)
reimg = InvHaar2D(coeffsdetail12)
re_im_out = save_image(reimg, "Images/Output/sample_scale1_reduced_and_padded_zeroHV.webp")
# re_im_out.show()
```

Figure 20 : Wavelet Transform Modifications for scale=1

Figure 21 : Zeroing approximation coefficient for scale=1

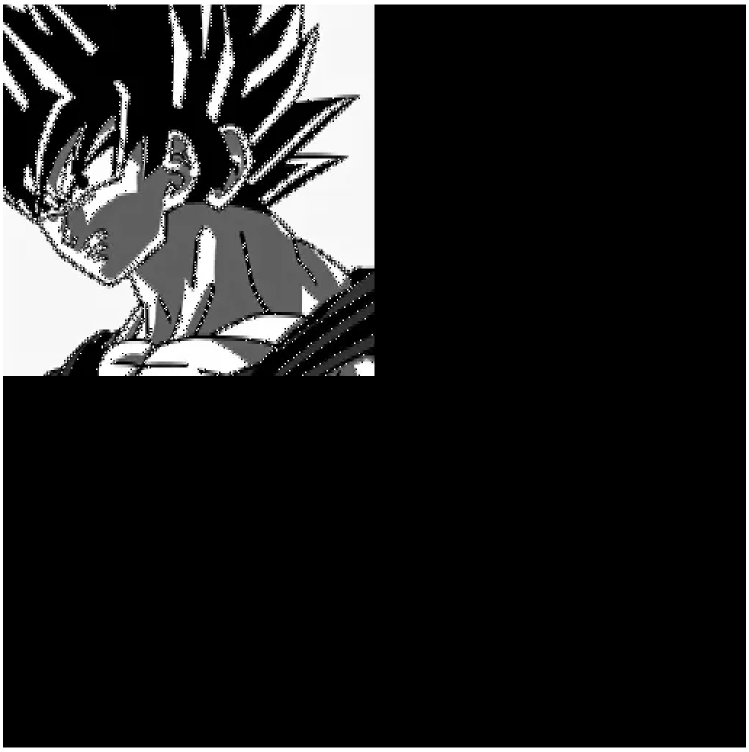


Figure 22 : Zeroing horizontal detail coefficient for scale=1

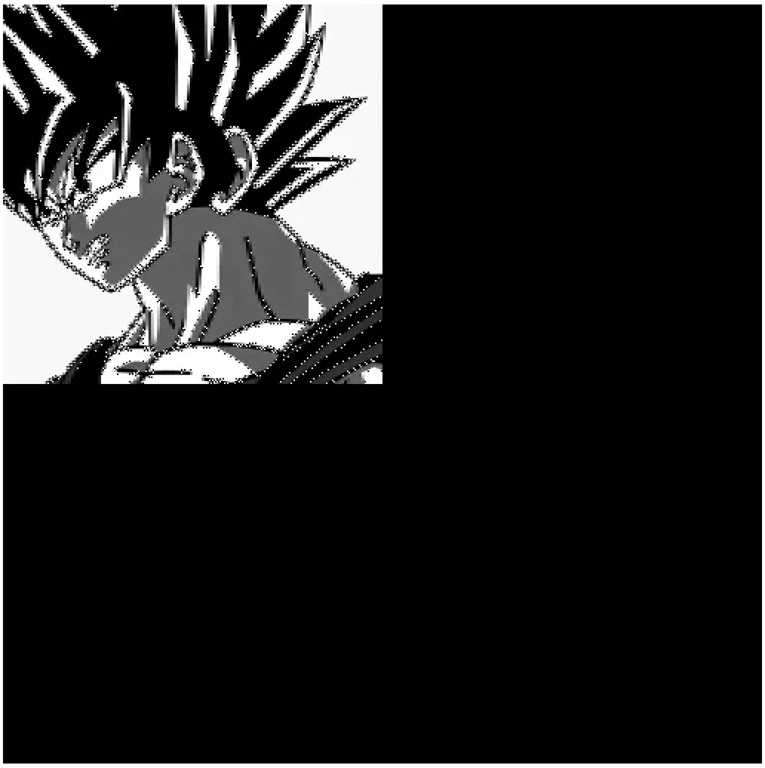


Figure 23 : Zeroing vertical detail coefficient for scale=1

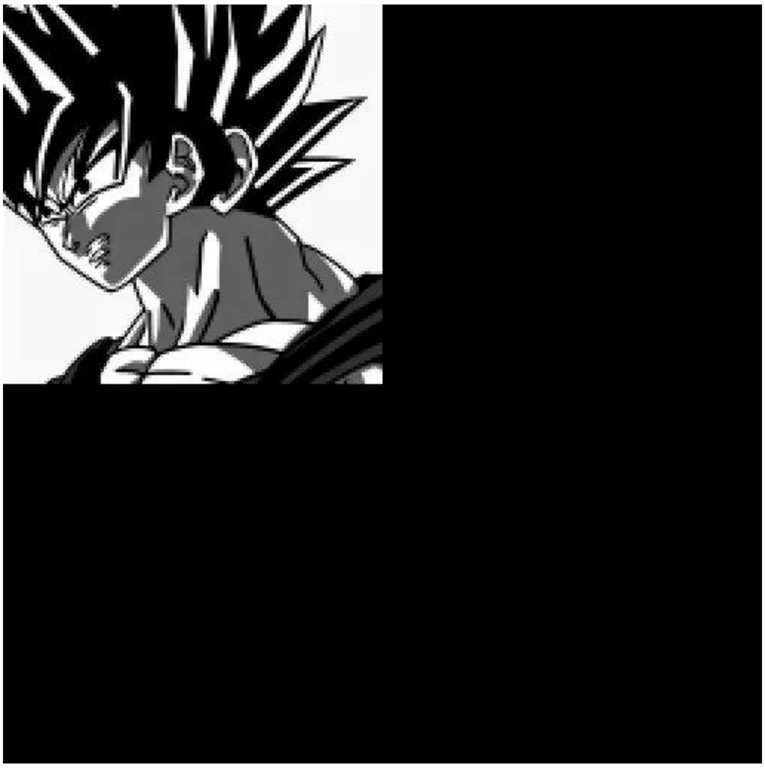


Figure 24 : Zeroing both horizontal and vertical detail coefficient for scale=1

Scale = 2 :

```python
#---------------------------------------#
#             Scale = 2                 #
#---------------------------------------#

coeffs = Haar2D(reduced_img , 2)

#zeroing approximatioin coefficients
coeffsA = zeroA(coeffs)
reimg = InvHaar2D(coeffsA)
re_im_out = save_image(reimg, "Images/Output/sample_scale2_reduced_and_padded_zeroA.webp")
# re_im_out.show()


#zeroing horizontal coefficients
coeffsdetail1 = zerodetail(coeffs,0)
reimg = InvHaar2D(coeffsdetail1)
re_im_out = save_image(reimg, "Images/Output/sample_scale2_reduced_and_padded_zeroH.webp")
# re_im_out.show()

#zeroing vertical coefficients
coeffsdetail2 = zerodetail(coeffs,1)
reimg = InvHaar2D(coeffsdetail2)
re_im_out = save_image(reimg, "Images/Output/sample_scale2_reduced_and_padded_zeroV.webp")
# re_im_out.show()

#zeroing both horizontal and vertical coefficients
coeffsdetail12 = zerodetail(coeffs,0)
coeffsdetail12 = zerodetail(coeffsdetail12,1)
reimg = InvHaar2D(coeffsdetail12)
re_im_out = save_image(reimg, "Images/Output/sample_scale2_reduced_and_padded_zeroHV.webp")
# re_im_out.show()
```

Figure 25 : Wavelet Transform Modifications for scale=2

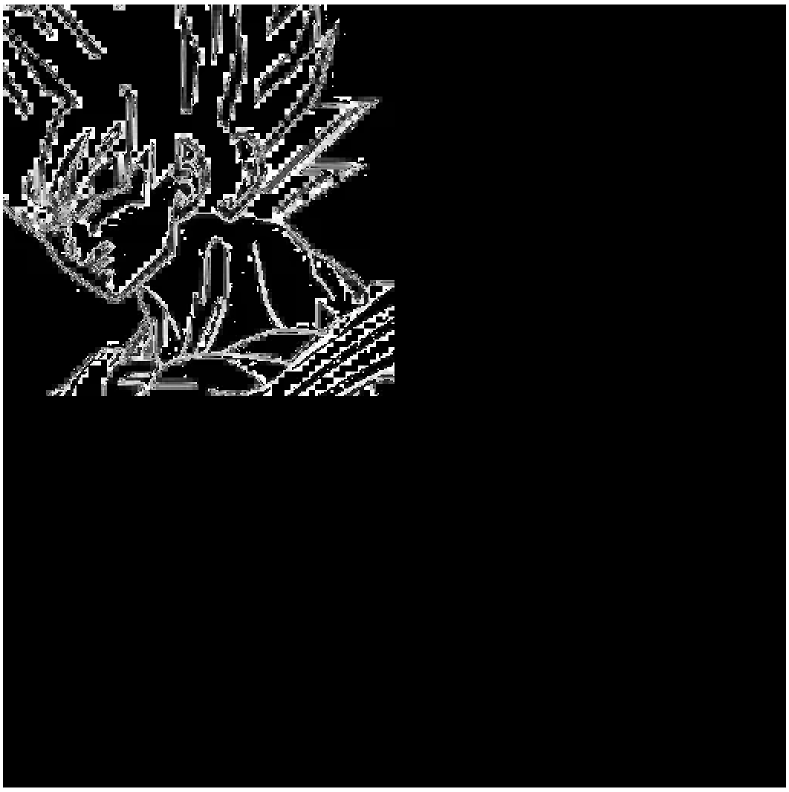


Figure 26 : Zeroing approximation coefficient for scale=2

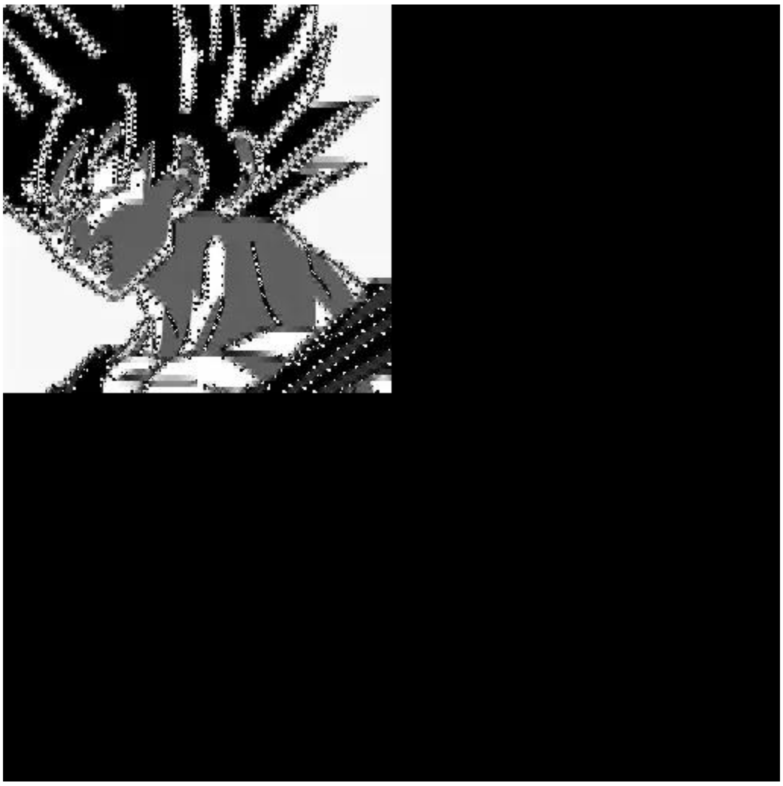


Figure 27 : Zeroing horizontal detail coefficient for scale=2

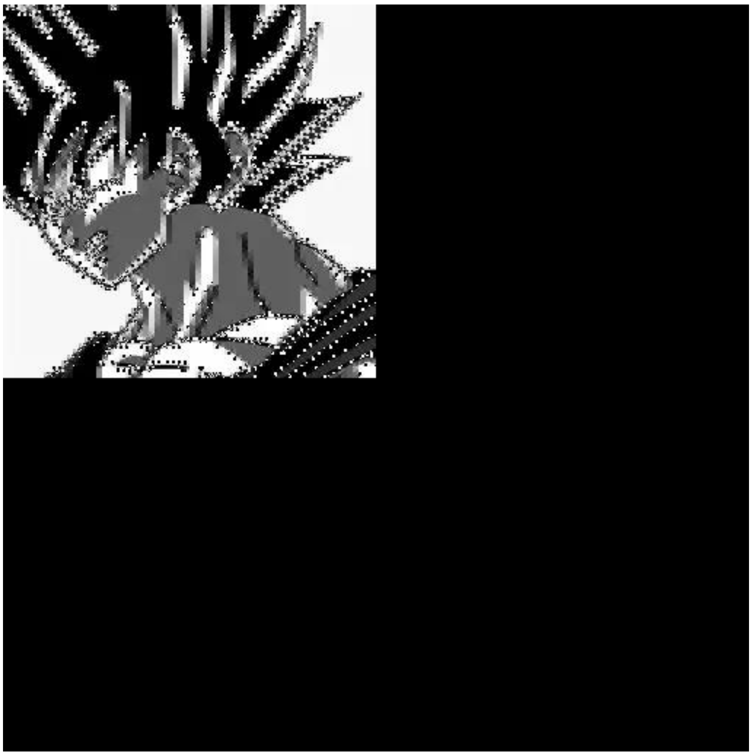


Figure 28 : Zeroing vertical detail coefficient for scale=2

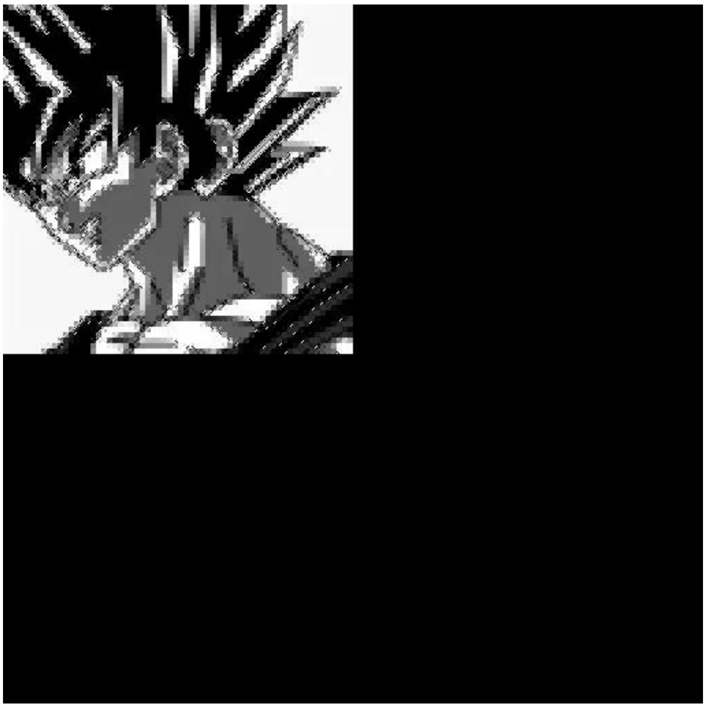


Figure 29 : Zeroing both horizontal and vertical detail coefficient for scale=2

Scale = 3 :

```
#---------------------------------------#
#            Scale = 3                  #
#---------------------------------------#

coeffs = Haar2D(reduced_img , 3)

#zeroing approximatioin coefficients
coeffsA = zeroA(coeffs)
reimg = InvHaar2D(coeffsA)
re_im_out = save_image(reimg, "Images/Output/sample_scale3_reduced_and_padded_zeroA.webp")
# re_im_out.show()

#zeroing horizontal coefficients
coeffsdetail1 = zerodetail(coeffs,0)
reimg = InvHaar2D(coeffsdetail1)
re_im_out = save_image(reimg, "Images/Output/sample_scale3_reduced_and_padded_zeroH.webp")
# re_im_out.show()

#zeroing vertical coefficients
coeffsdetail2 = zerodetail(coeffs,1)
reimg = InvHaar2D(coeffsdetail2)
re_im_out = save_image(reimg, "Images/Output/sample_scale3_reduced_and_padded_zeroV.webp")
# re_im_out.show()

#zeroing both horizontal and vertical coefficients
coeffsdetail12 = zerodetail(coeffs,0)
coeffsdetail12 = zerodetail(coeffsdetail12,1)
reimg = InvHaar2D(coeffsdetail12)
re_im_out = save_image(reimg, "Images/Output/sample_scale3_reduced_and_padded_zeroHV.webp")
# re_im_out.show()
```

Figure 30 : Wavelet Transform Modifications for scale=3

Figure 31 : Zeroing approximation coefficient for scale=3

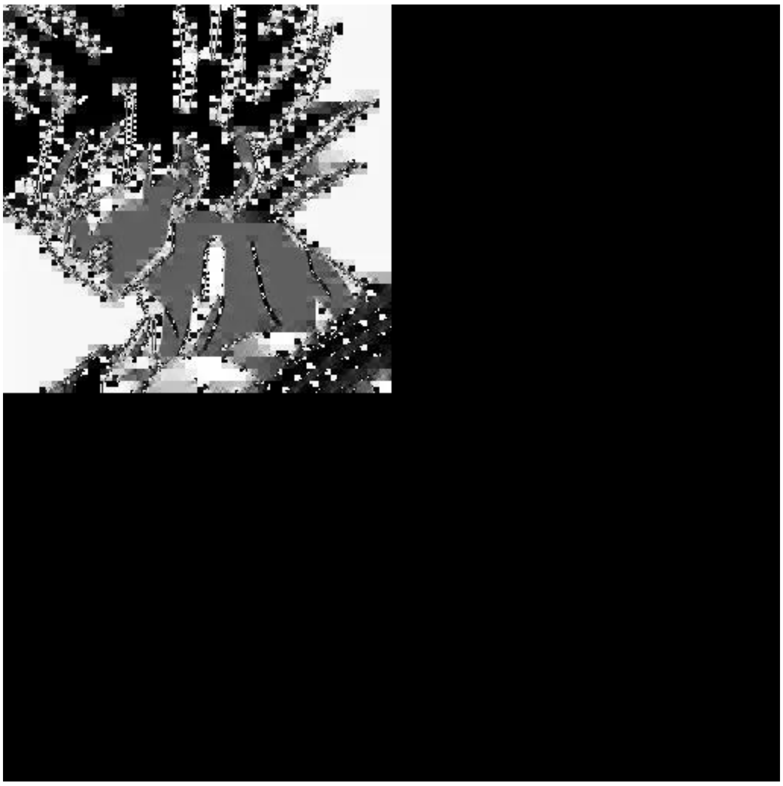


Figure 32 : Zeroing horizontal detail coefficient for scale=3

Figure 33 : Zeroing vertical detail coefficient for scale=3

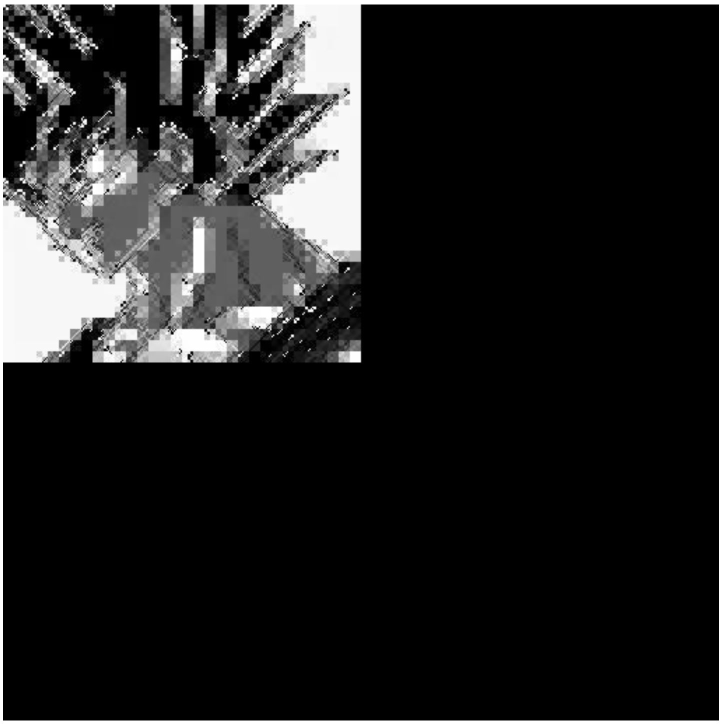


Figure 34 : Zeroing both horizontal and vertical detail coefficient for scale=3

Scale = 4 :

```
#----------------------------------------#
#              Scale = 4                 #
#----------------------------------------#

coeffs = Haar2D(reduced_img , 4)

#zeroing approximatioin coefficients
coeffsA = zeroA(coeffs)
reimg = InvHaar2D(coeffsA)
re_im_out = save_image(reimg, "Images/Output/sample_scale4_reduced_and_padded_zeroA.webp")
# re_im_out.show()


#zeroing horizontal coefficients
coeffsdetail1 = zerodetail(coeffs,0)
reimg = InvHaar2D(coeffsdetail1)
re_im_out = save_image(reimg, "Images/Output/sample_scale4_reduced_and_padded_zeroH.webp")
# re_im_out.show()

#zeroing vertical coefficients
coeffsdetail2 = zerodetail(coeffs,1)
reimg = InvHaar2D(coeffsdetail2)
re_im_out = save_image(reimg, "Images/Output/sample_scale4_reduced_and_padded_zeroV.webp")
# re_im_out.show()

#zeroing both horizontal and vertical coefficients
coeffsdetail12 = zerodetail(coeffs,0)
coeffsdetail12 = zerodetail(coeffsdetail12,1)
reimg = InvHaar2D(coeffsdetail12)
re_im_out = save_image(reimg, "Images/Output/sample_scale4_reduced_and_padded_zeroHV.webp")
# re_im_out.show()
```

Figure 35 : Wavelet Transform Modifications for scale=4

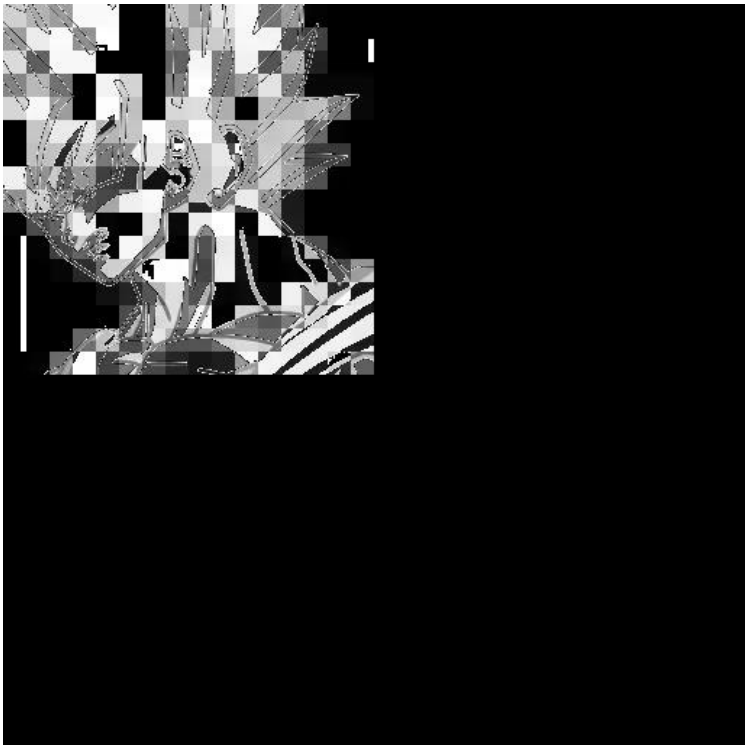


Figure 36 : Zeroing approximation coefficient for scale=4

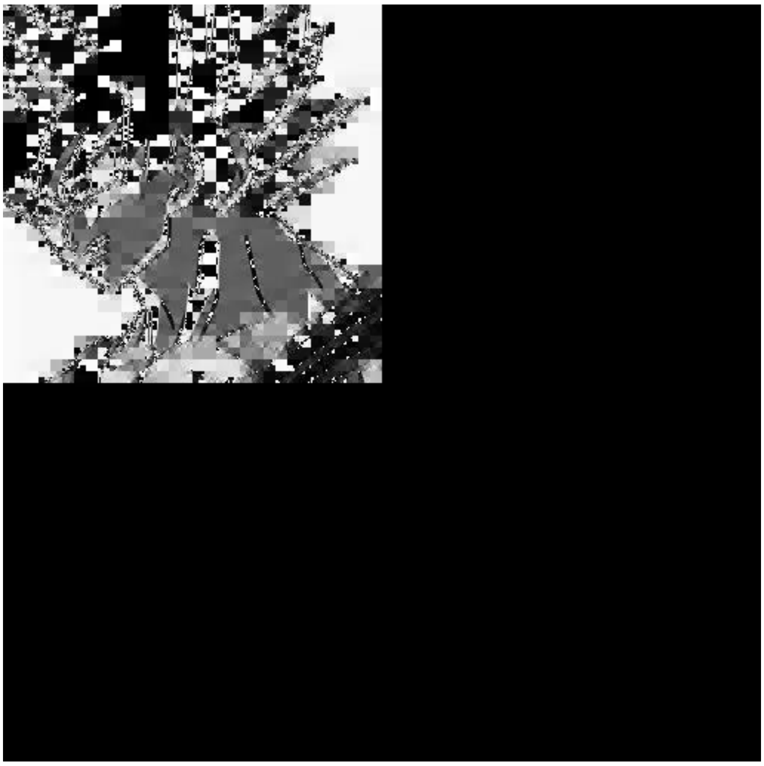


Figure 37 : Zeroing horizontal detail coefficient for scale=4

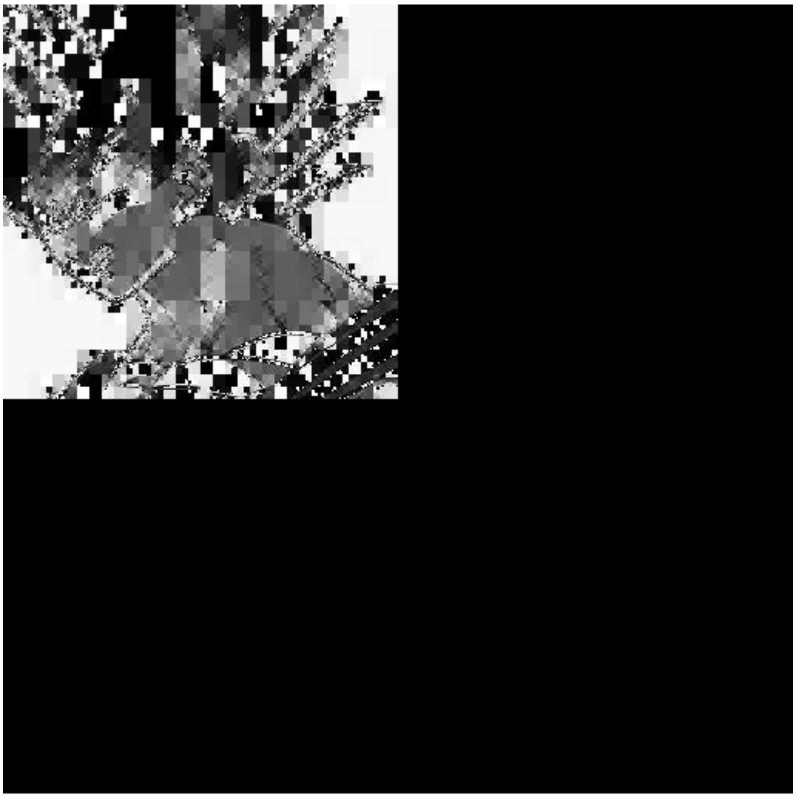


Figure 38 : Zeroing vertical detail coefficient for scale=4

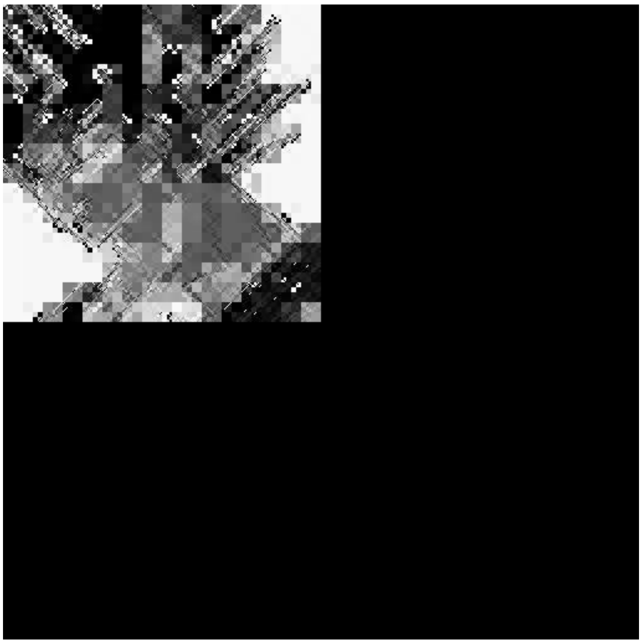


Figure 39 : Zeroing both horizontal and vertical detail coefficient for scale=4

Scale = 5 :

```
#----------------------------------------#
#             Scale = 5                  #
#----------------------------------------#

coeffs = Haar2D(reduced_img , 5)

#zeroing approximatioin coefficients
coeffsA = zeroA(coeffs)
reimg = InvHaar2D(coeffsA)
re_im_out = save_image(reimg, "Images/Output/sample_scale5_reduced_and_padded_zeroA.webp")
# re_im_out.show()


#zeroing horizontal coefficients
coeffsdetail1 = zerodetail(coeffs,0)
reimg = InvHaar2D(coeffsdetail1)
re_im_out = save_image(reimg, "Images/Output/sample_scale5_reduced_and_padded_zeroH.webp")
# re_im_out.show()

#zeroing vertical coefficients
coeffsdetail2 = zerodetail(coeffs,1)
reimg = InvHaar2D(coeffsdetail2)
re_im_out = save_image(reimg, "Images/Output/sample_scale5_reduced_and_padded_zeroV.webp")
# re_im_out.show()

#zeroing both horizontal and vertical coefficients
coeffsdetail12 = zerodetail(coeffs,0)
coeffsdetail12 = zerodetail(coeffsdetail12,1)
reimg = InvHaar2D(coeffsdetail12)
re_im_out = save_image(reimg, "Images/Output/sample_scale5_reduced_and_padded_zeroHV.webp")
# re_im_out.show()
```

Figure 40 : Wavelet Transform Modifications for scale=5

Figure 41 : Zeroing approximation coefficient for scale=5

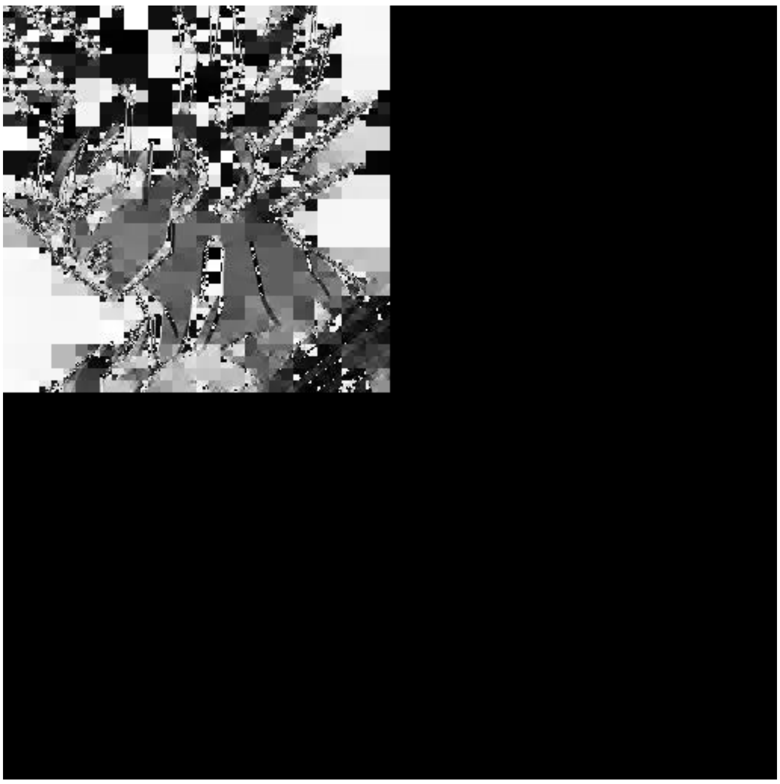


Figure 42 : Zeroing horizontal detail coefficient for scale=5

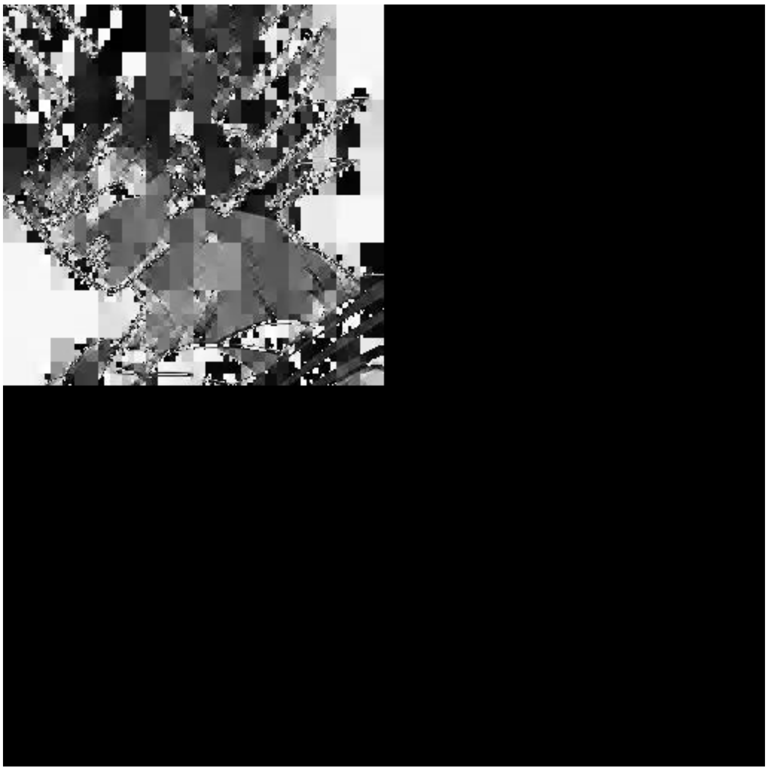


Figure 43 : Zeroing vertical detail coefficient for scale=5

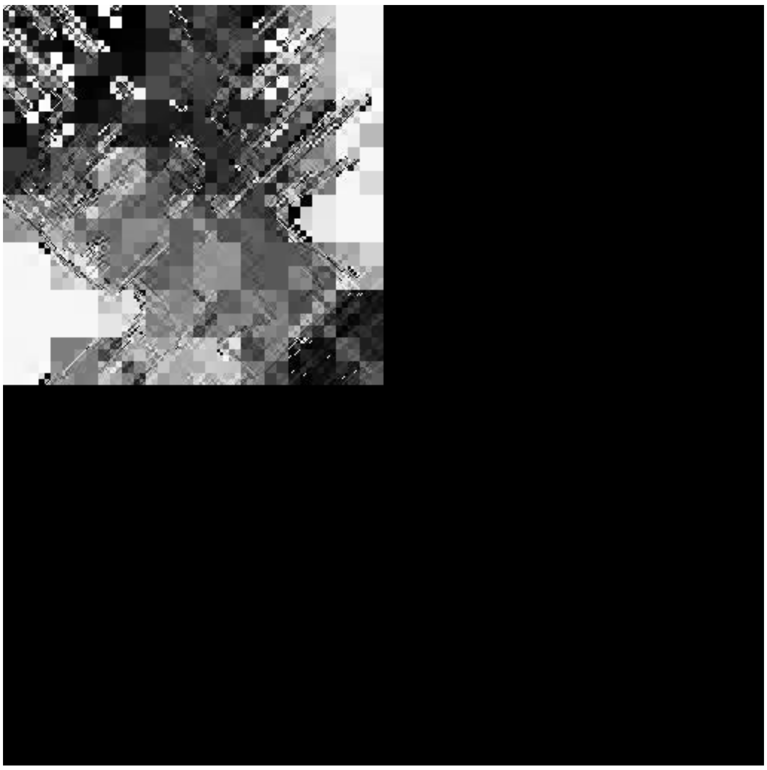


Figure 44 : Zeroing both horizontal and vertical detail coefficient for scale=5

Scale = 6 :

```
#----------------------------------------#
#              Scale = 6                 #
#----------------------------------------#

coeffs = Haar2D(reduced_img , 6)

#zeroing approximatioin coefficients
coeffsA = zeroA(coeffs)
reimg = InvHaar2D(coeffsA)
re_im_out = save_image(reimg, "Images/Output/sample_scale6_reduced_and_padded_zeroA.webp")
# re_im_out.show()


#zeroing horizontal coefficients
coeffsdetail1 = zerodetail(coeffs,0)
reimg = InvHaar2D(coeffsdetail1)
re_im_out = save_image(reimg, "Images/Output/sample_scale6_reduced_and_padded_zeroH.webp")
# re_im_out.show()

#zeroing vertical coefficients
coeffsdetail2 = zerodetail(coeffs,1)
reimg = InvHaar2D(coeffsdetail2)
re_im_out = save_image(reimg, "Images/Output/sample_scale6_reduced_and_padded_zeroV.webp")
# re_im_out.show()

#zeroing both horizontal and vertical coefficients
coeffsdetail12 = zerodetail(coeffs,0)
coeffsdetail12 = zerodetail(coeffsdetail12,1)
reimg = InvHaar2D(coeffsdetail12)
re_im_out = save_image(reimg, "Images/Output/sample_scale6_reduced_and_padded_zeroHV.webp")
# re_im_out.show()
```

Figure 45 : Wavelet Transform Modifications for scale=6

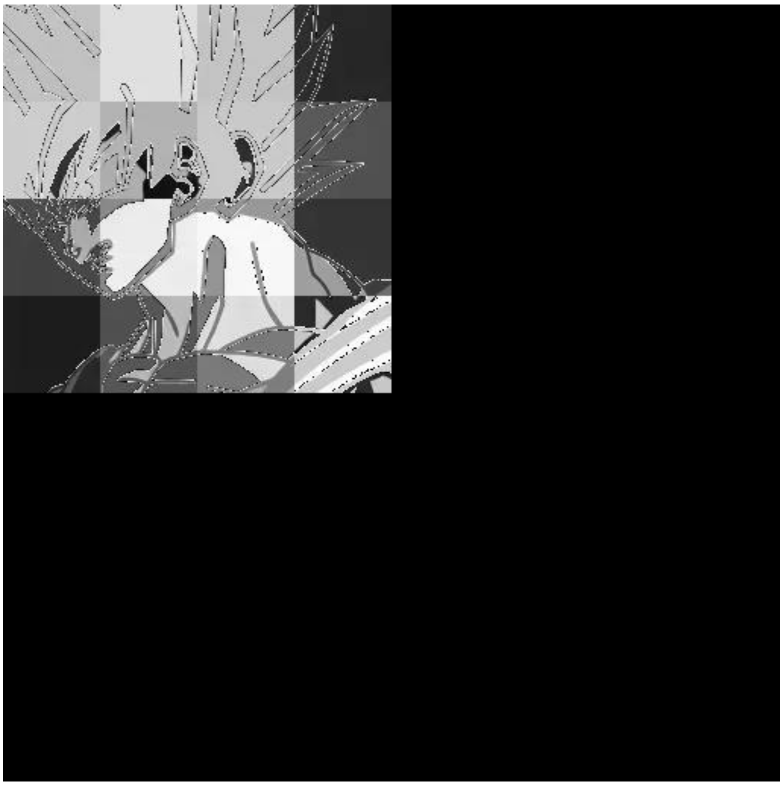


Figure 46 : Zeroing approximation coefficient for scale=6

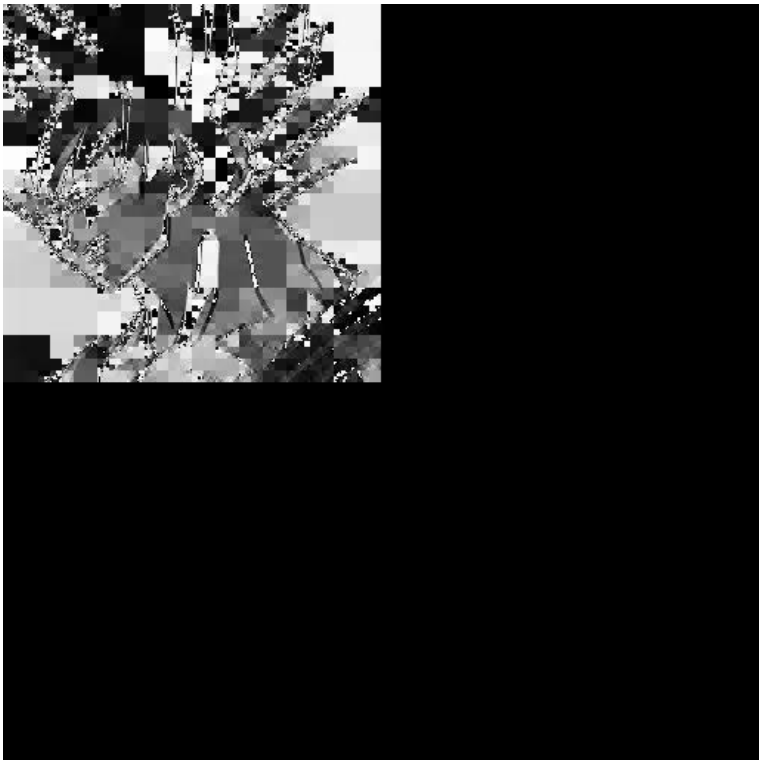


Figure 47 : Zeroing horizontal detail coefficient for scale=6

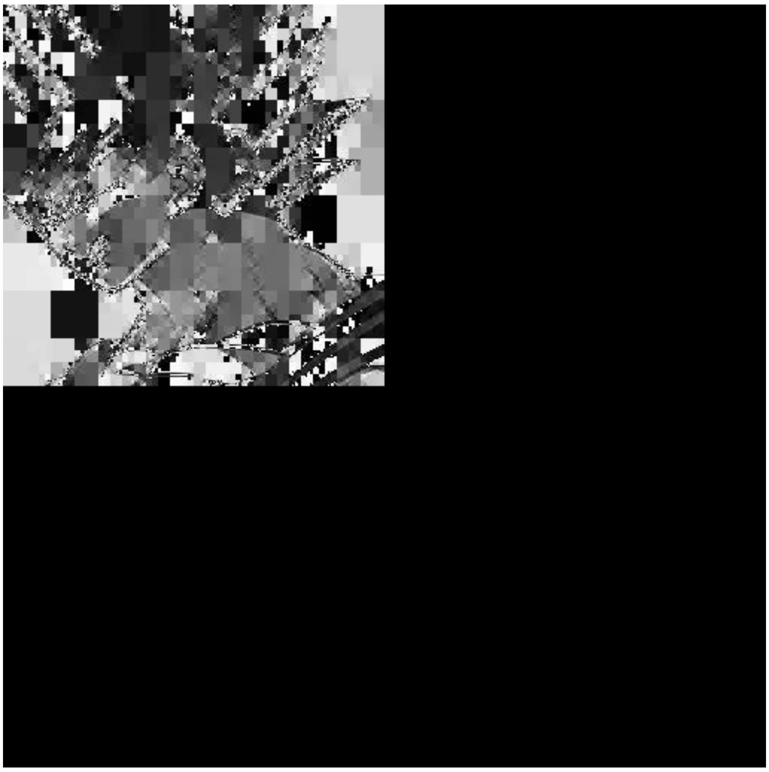


Figure 48 : Zeroing vertical detail coefficient for scale=6

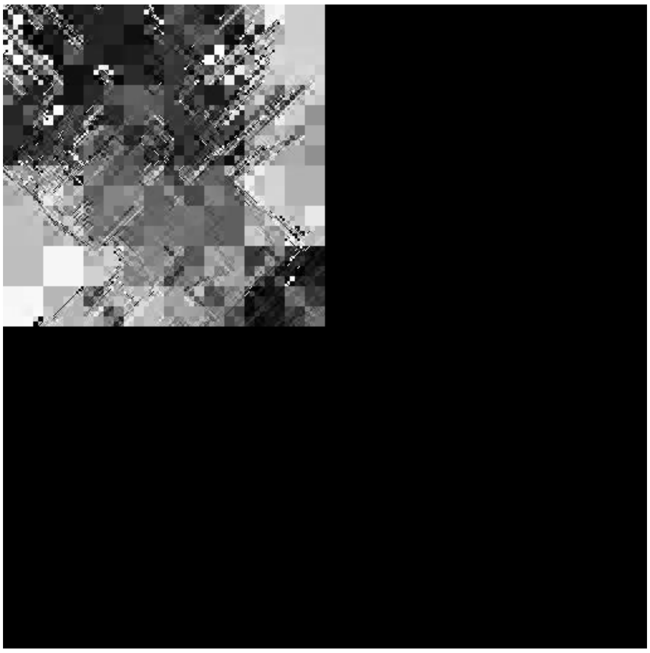


Figure 49 : Zeroing both horizontal and vertical detail coefficient for scale=6

Scale = 7 :

```
#---------------------------------------#
#            Scale = 7                  #
#---------------------------------------#

coeffs = Haar2D(reduced_img , 7)

#zeroing approximatioin coefficients
coeffsA = zeroA(coeffs)
reimg = InvHaar2D(coeffsA)
re_im_out = save_image(reimg, "Images/Output/sample_scale7_reduced_and_padded_zeroA.webp")
# re_im_out.show()


#zeroing horizontal coefficients
coeffsdetail1 = zerodetail(coeffs,0)
reimg = InvHaar2D(coeffsdetail1)
re_im_out = save_image(reimg, "Images/Output/sample_scale7_reduced_and_padded_zeroH.webp")
# re_im_out.show()

#zeroing vertical coefficients
coeffsdetail2 = zerodetail(coeffs,1)
reimg = InvHaar2D(coeffsdetail2)
re_im_out = save_image(reimg, "Images/Output/sample_scale7_reduced_and_padded_zeroV.webp")
# re_im_out.show()

#zeroing both horizontal and vertical coefficients
coeffsdetail12 = zerodetail(coeffs,0)
coeffsdetail12 = zerodetail(coeffsdetail12,1)
reimg = InvHaar2D(coeffsdetail12)
re_im_out = save_image(reimg, "Images/Output/sample_scale7_reduced_and_padded_zeroHV.webp")
# re_im_out.show()
```

Figure 50 : Wavelet Transform Modifications for scale=7

Figure 51 : Zeroing approximation coefficient for scale=7

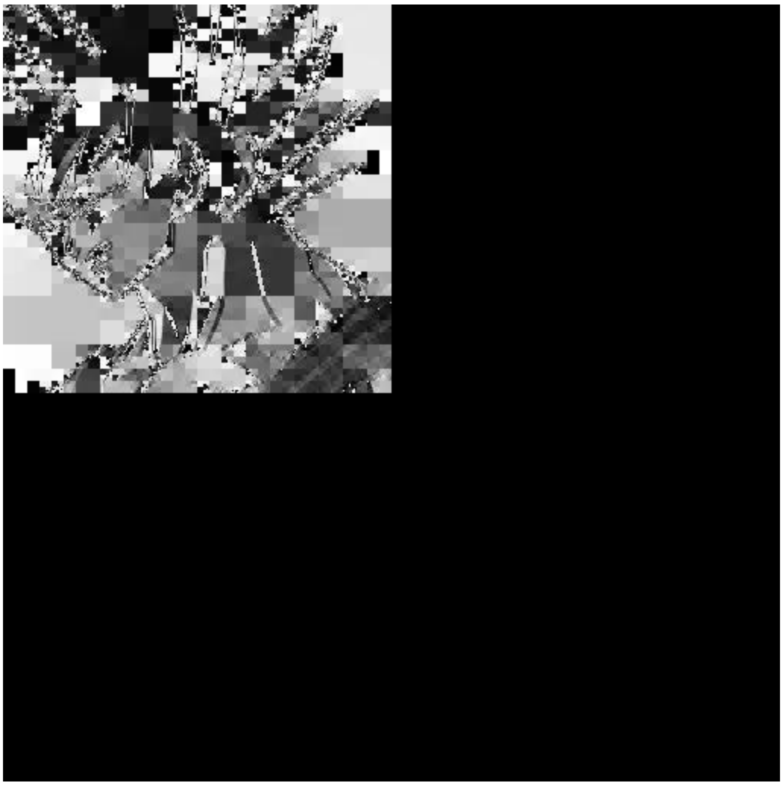


Figure 52 : Zeroing horizontal detail coefficient for scale=7

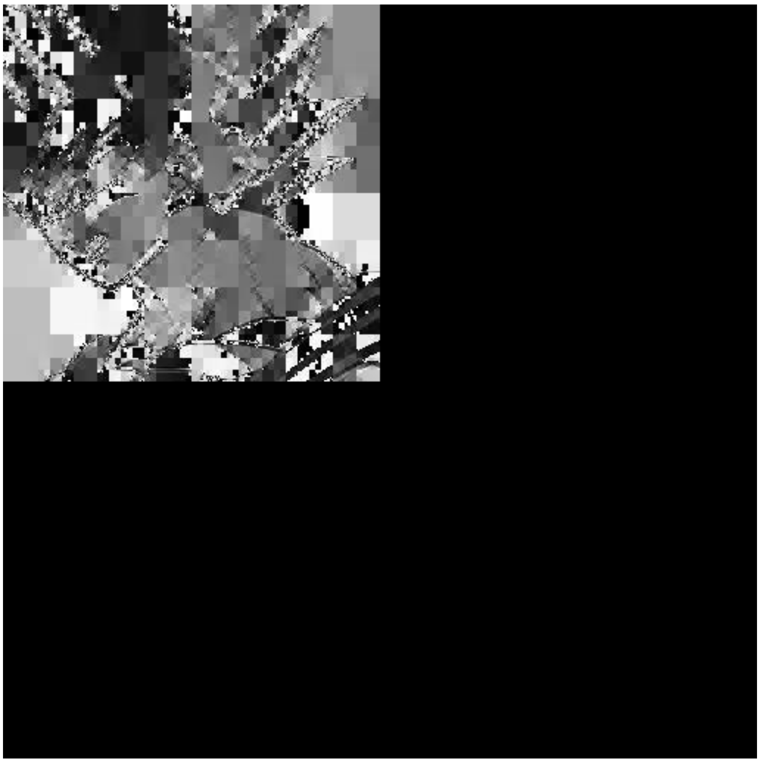


Figure 53 : Zeroing vertical detail coefficient for scale=7

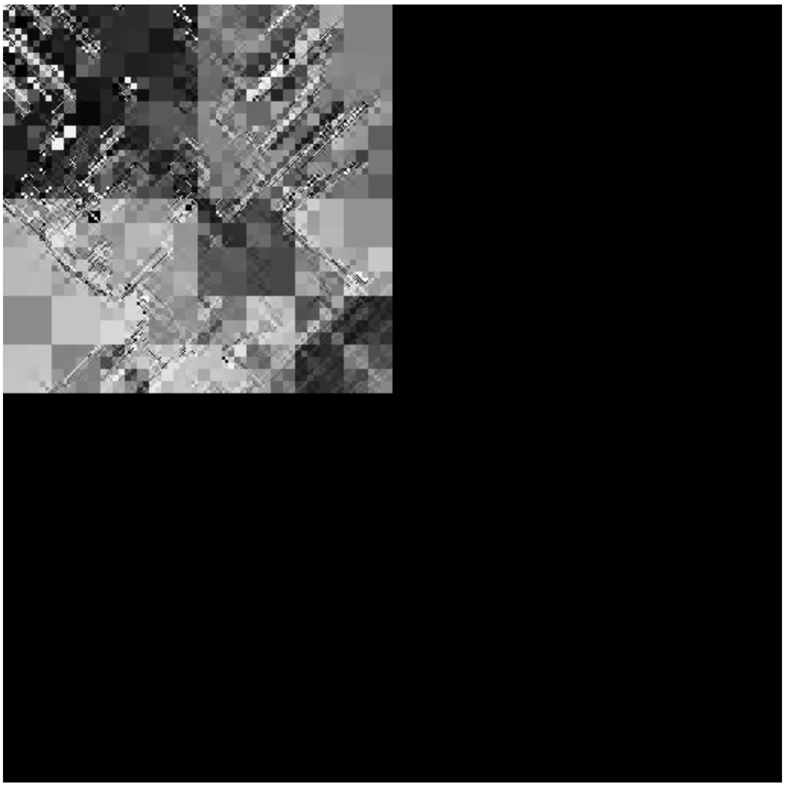


Figure 54 : Zeroing both horizontal and vertical detail coefficient for scale=7

Scale = 8 :

```
#----------------------------------------#
#            Scale = 8                   #
#----------------------------------------#

coeffs = Haar2D(reduced_img , 8)

#zeroing approximatioin coefficients
coeffsA = zeroA(coeffs)
reimg = InvHaar2D(coeffsA)
re_im_out = save_image(reimg, "Images/Output/sample_scale8_reduced_and_padded_zeroA.webp")
# re_im_out.show()


#zeroing horizontal coefficients
coeffsdetail1 = zerodetail(coeffs,0)
reimg = InvHaar2D(coeffsdetail1)
re_im_out = save_image(reimg, "Images/Output/sample_scale8_reduced_and_padded_zeroH.webp")
# re_im_out.show()

#zeroing vertical coefficients
coeffsdetail2 = zerodetail(coeffs,1)
reimg = InvHaar2D(coeffsdetail2)
re_im_out = save_image(reimg, "Images/Output/sample_scale8_reduced_and_padded_zeroV.webp")
# re_im_out.show()

#zeroing both horizontal and vertical coefficients
coeffsdetail12 = zerodetail(coeffs,0)
coeffsdetail12 = zerodetail(coeffsdetail12,1)
reimg = InvHaar2D(coeffsdetail12)
re_im_out = save_image(reimg, "Images/Output/sample_scale8_reduced_and_padded_zeroHV.webp")
# re_im_out.show()
```

Figure 55 : Wavelet Transform Modifications for scale=8

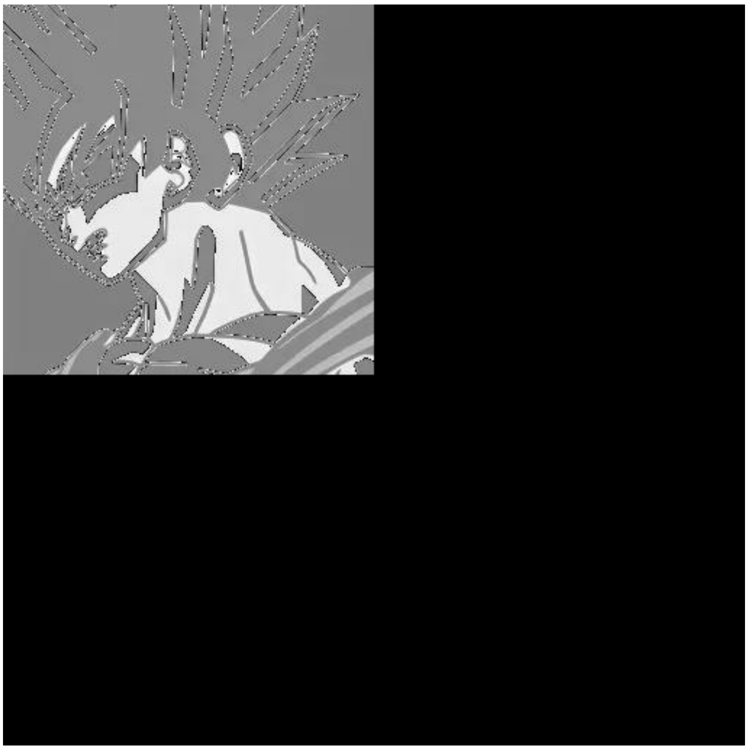


Figure 56 : Zeroing approximation coefficient for scale=8

Figure 57 : Zeroing horizontal detail coefficient for scale=8

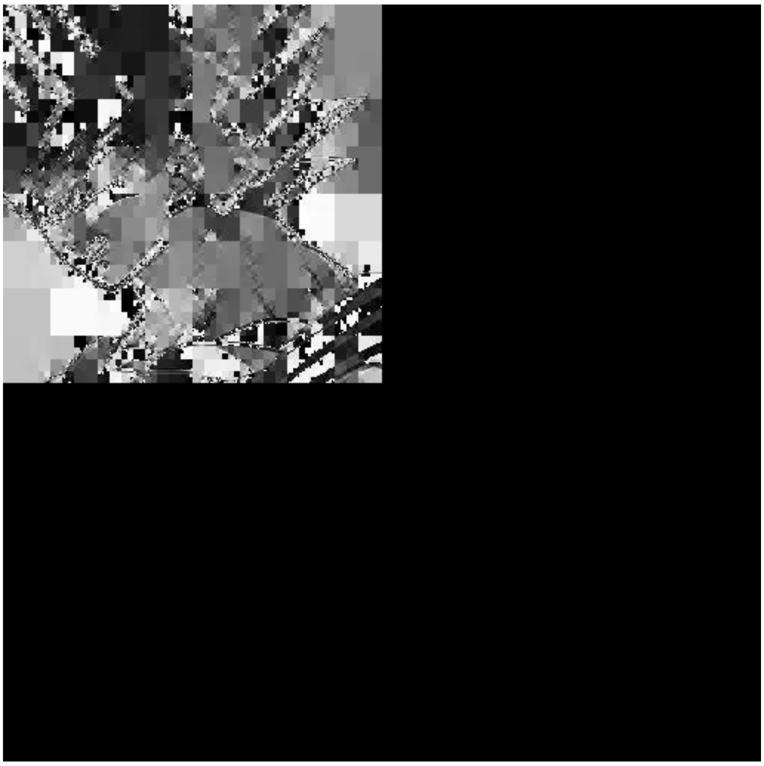


Figure 58 : Zeroing vertical detail coefficient for scale=8

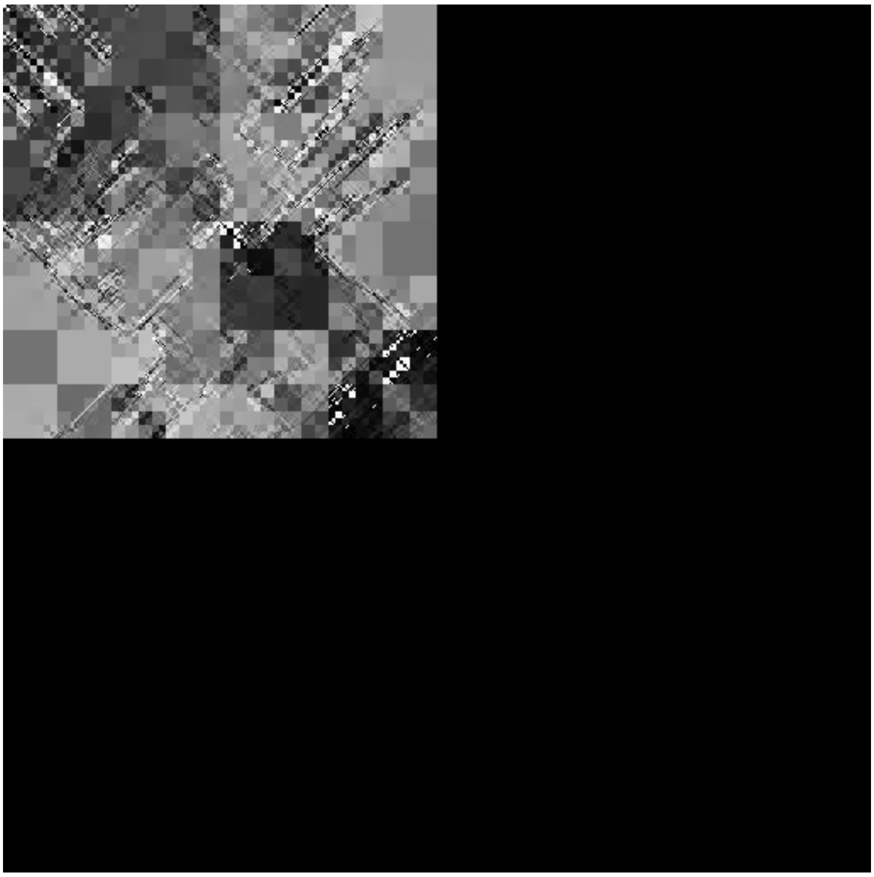


Figure 59 : Zeroing both horizontal and vertical detail coefficient for scale=8

Conclusion

- In this part, we can see that zeroing approximation coefficient leads to much greater loss of data than compared to zeroing detail coefficients.
- This is more visible in higher scales.

Part-3 : Wavelet Coding

Truncating the coefficients to achieve compression for various scales.

We will truncate every coefficient between -threshold to +threshold.

Therefore :-

- If coeff< -threshold, then coeff = -threshold
- If coeff> +threshold, then coeff = +threshold
- Otherwise, no change

Error is quantized as Mean squared error between the original image and compressed image.

```
#---------------------------------------------------------------#
#             Truncating Detail coefficient                     #
#---------------------------------------------------------------#

def truncate(coeffs , threshold):
  '''we are compressing the image by truncating the detail coefficients
  into -threshold to threshold'''

  '''A lower threshold will result in a higher level of compression but
  lower image quality, as more detail coefficients will be truncated.
  On the other hand, a higher threshold will result in lower
  compression but higher image quality, as fewer detail coefficients will
  be truncated.'''

  coeffs_truncate = coeffs.copy()
  for i in range(1,len(coeffs_truncate)):
    coeffs_truncate[i] = [np.clip(d,-threshold,threshold) for d in coeffs_truncate[i]]

  return coeffs_truncate

def quantization_error(img , reimg):

  '''We have calculated the quantization error by
   the root mean square error (RMSE) between the original image and i
  ts quantized version'''
  error = np.mean((img - reimg) ** 2)
  return error
```

Figure 60 : Function for truncating the coefficients and quantization error

Scale = 1 :

```
#---------------------------------------#
#              Scale = 1                #
#---------------------------------------#


coeffs = Haar2D(img , 1)
coeffs_truncate = truncate(coeffs,100) #threshold = 100
reimg = InvHaar2D(coeffs_truncate)
re_im_out = save_image(reimg, "Images/Output/sample_scale1_reduced_and_padded_truncate.webp")
#re_im_out.show()
error = quantization_error(img , reimg)
print("Quantization_error for scale 1: ",error)
```

Figure 61 : Truncating the coefficients for scale=1

Figure 62 : Reconstructed Image after truncating the coefficients for scale=1

Scale = 2 :

```
#---------------------------------------#
#             Scale = 2                 #
#---------------------------------------#

coeffs = Haar2D(img , 2)
coeffs_truncate = truncate(coeffs,100) #threshold = 100
reimg = InvHaar2D(coeffs_truncate)
re_im_out = save_image(reimg, "Images/Output/sample_scale2_reduced_and_padded_truncate.webp")
#re_im_out.show()
error = quantization_error(img , reimg)
print("Quantization_error for scale 2: ",error)
```

Figure 63 : Truncating the coefficients for scale=2

Figure 64 : Reconstructed Image after truncating the coefficients for scale=2

Scale = 3 :

```
#---------------------------------------#
#             Scale = 3                 #
#---------------------------------------#

coeffs = Haar2D(img , 3)
coeffs_truncate = truncate(coeffs,100) #threshold = 100
reimg = InvHaar2D(coeffs_truncate)
re_im_out = save_image(reimg, "Images/Output/sample_scale3_reduced_and_padded_truncate.webp")
#re_im_out.show()
error = quantization_error(img , reimg)
print("Quantization_error for scale 3: ",error)
```

Figure 65 : Truncating the coefficients for scale=3

Figure 66 : Reconstructed Image after truncating the coefficients for scale=3

Scale = 4 :

```
#-----------------------------------------#
#              Scale = 4                  #
#-----------------------------------------#

coeffs = Haar2D(img , 4)
coeffs_truncate = truncate(coeffs,100) #threshold = 100
reimg = InvHaar2D(coeffs_truncate)
re_im_out = save_image(reimg, "Images/Output/sample_scale4_reduced_and_padded_truncate.webp")
#re_im_out.show()
error = quantization_error(img , reimg)
print("Quantization_error for scale 4: ",error)
```

Figure 67 : Truncating the coefficients for scale=4

Figure 68 : Reconstructed Image after truncating the coefficients for scale=4

Scale = 5 :

```
#---------------------------------------#
#             Scale = 5                 #
#---------------------------------------#

coeffs = Haar2D(img , 5)
coeffs_truncate = truncate(coeffs,100) #threshold = 100
reimg = InvHaar2D(coeffs_truncate)
re_im_out = save_image(reimg, "Images/Output/sample_scale5_reduced_and_padded_truncate.webp")
#re_im_out.show()
error = quantization_error(img , reimg)
print("Quantization_error for scale 5: ",error)
```

Figure 69 : Truncating the coefficients for scale=5

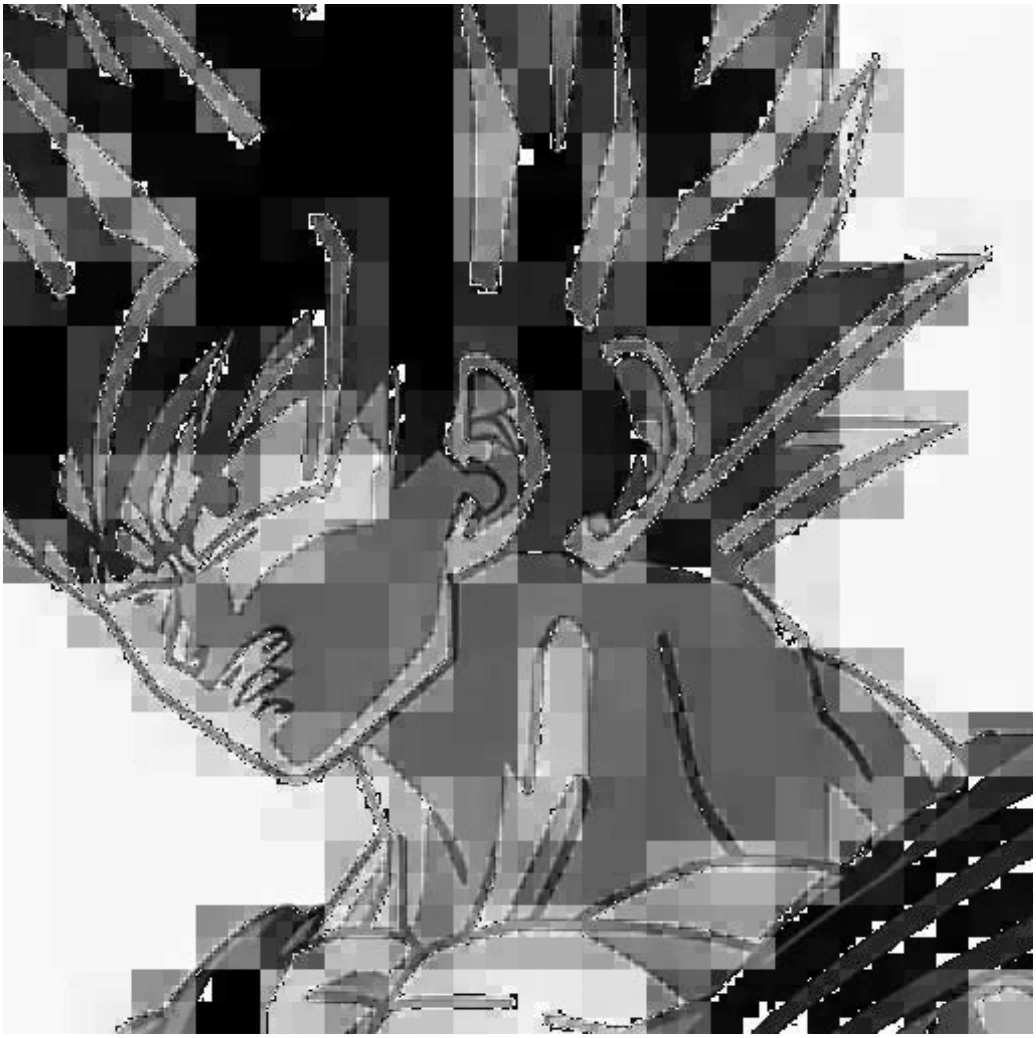


Figure 70 : Reconstructed Image after truncating the coefficients for scale=5

Scale = 6 :

```
#---------------------------------------#
#            Scale = 6                  #
#---------------------------------------#

coeffs = Haar2D(img , 6)
coeffs_truncate = truncate(coeffs,100) #threshold = 100
reimg = InvHaar2D(coeffs_truncate)
re_im_out = save_image(reimg, "Images/Output/sample_scale6_reduced_and_padded_truncate.webp")
#re_im_out.show()
error = quantization_error(img , reimg)
print("Quantization_error for scale 6: ",error)
```

Figure 71 : Truncating the coefficients for scale=6

Figure 72 : Reconstructed Image after truncating the coefficients for scale=6

Scale = 7 :

```
#---------------------------------------#
#             Scale = 7                 #
#---------------------------------------#

coeffs = Haar2D(img , 7)
coeffs_truncate = truncate(coeffs,100) #threshold = 100
reimg = InvHaar2D(coeffs_truncate)
re_im_out = save_image(reimg, "Images/Output/sample_scale7_reduced_and_padded_truncate.webp")
#re_im_out.show()
error = quantization_error(img , reimg)
print("Quantization_error for scale 7: ",error)
```

Figure 73 : Truncating the coefficients for scale=7

Figure 74 : Reconstructed Image after truncating the coefficients for scale=7

Scale = 8 :

```
#---------------------------------------#
#            Scale = 8                  #
#---------------------------------------#

coeffs = Haar2D(img , 8)
coeffs_truncate = truncate(coeffs,100) #threshold = 100
reimg = InvHaar2D(coeffs_truncate)
re_im_out = save_image(reimg, "Images/Output/sample_scale8_reduced_and_padded_truncate.webp")
#re_im_out.show()
error = quantization_error(img , reimg)
print("Quantization_error for scale 8: ",error)
```

Figure 75 : Truncating the coefficients for scale=8

Figure 76 : Reconstructed Image after truncating the coefficients for scale=8

Conclusions :

Scale	**Quantization Error**
Scale 1	3.683704376220702
Scale 2	202.57805347442624
Scale 3	932.4409230947493
Scale 4	2118.8598829209805
Scale 5	3557.418792694807
Scale 6	4881.167885924689
Scale 7	6117.3206969490275
Scale 8	8225.115728021483

- We can see that Quantization error is increasing with increase in scale.
- This is because we are truncating images with threshold 100 and this leads to propagation error.
- Also, truncating the coefficients of DWT makes this is a lossy compression.