

Κανιάρης Δημοσθένης, AM : 03116130

Κυριακίδης Δημήτριος, AM : 03117077

Τεχνολογία και Ανάλυση Εικόνων και Βίντεο

1η ομαδική άσκηση

1.Θεωρητικό Μέρος

α) Ποιό είναι το πρόβλημα και ποιές είναι οι κύριες ιδέες που πραγματεύεται το άρθρο αυτό;

Το πρόβλημα που παρουσιάζεται στο άρθρο είναι πως τα γειτονικά pixels στις εικόνες έχουν έντονο correlation με αποτέλεσμα να μην είναι ορθό να αναπαρίστανται όπως είναι η αρχική εικόνα, καθώς έχουμε πληροφορία που επαναλαμβάνεται. Επίσης δεν είναι αποδοτικό από πλευράς χώρου γιατί θέλουμε πολλά BPP (Bits Per Pixel) για να αναπαραστήσουμε το συνολικό αριθμό (0-255) της έντασης της φωτεινότητας του pixel. Για την επίλυση του προβλήματος αυτού αρκεί να βρούμε μια αναπαράσταση της εικόνας που εξαλείφει το correlation αυτό κάτι που έχει επιτευχθεί από predictive και transform τεχνικές. Οι μέθοδοι των μετασχηματισμών προσφέρουν καλύτερη συμπίεση δεδομένων όμως είναι πιο πολύπλοκη διαδικασία. Για αυτό στο συγκεκριμένο άρθρο προτείνεται μια άλλη μέθοδο που συνδυάζει χαρακτηριστικά από predictive και transform τεχνικές, την "Laplacian Pyramid Coding". Πιο συγκεκριμένα μπορεί να χαρακτηρισθεί από την "Gaussian" Πυραμίδα και την "Laplacian" Πυραμίδα, με τη πρώτη να αποτελεί τις διαδοχικές εικόνες που φιλτράρωνται από βαθυπερατά φίλτρα ενώ η δεύτερη αποτελεί τις διαδοχικές διαφορές των αντίστοιχων εικόνων της "Gaussian" πυραμίδας.

β) Ποιός είναι ο σκοπός του φίλτρου Gauss στην κατασκευή της πυραμίδας; Τι θα συνέβαινε αν είχε χρησιμοποιηθεί μόνο υποδειγματοληψία;

Ο σκοπός του φίλτρου Gauss είναι να αποκόψουμε τις μεγάλες συχνότητες και να κρατήσουμε κάποιο μέσο των γειτονικών pixel. Έτσι στη συνέχεια έχουμε μια εικόνα που έχει περάσει από ένα φίλτρο που μοιάζει με βαθυπερατό με αποτέλεσμα να μπορούμε να την κωδικοποιήσουμε με χαμηλότερο ρυθμό. Ταυτόχρονα με την αφαίρεση της βαθυπερατής εικόνας από την αρχική, κρατάμε τις έντονες διαφορές μεταξύ των pixel και διώχνουμε την έντονη συσχέτιση των γειτονικών pixel. Έτσι μπορούμε να κωδικοποιήσουμε τις διαφορές των pixel μόνο με πολύ λιγότερα BPP από ότι στην αρχική εικόνα. Αν χρησιμοποιούσαμε μόνο υποδειγματοληψία χωρίς το φίλτρο Gauss τότε θα έπρεπε να έχουμε μεγάλο ρυθμό "sample rate" στη κωδικοποίηση καθώς θα δημιουργείται το πρόβλημα του "aliasing" και θα χάνουμε πληροφορία.

γ) Πώς χρησιμοποιείται η πυραμίδα Laplacian για την επίλυση του προβλήματος που αναφέρεται στο άρθρο; Γιατί αυτή η αναπαράσταση εικόνας είναι καλύτερη από την πυραμίδα Gauss στο πλαίσιο της εργασίας που πραγματεύεται το άρθρο;

Η Laplacian αποτελείται από τις εικόνες που δείχνουν πρόβλεψη σφαλμάτων, με αποτέλεσμα να έχουν πολύ μικρότερες τιμές από τις αρχικές εικόνες και να μπορούν να κωδικοποιηθούν με πολύ λιγότερα BPP έχει δηλαδή μικρότερη εντροπία. Η αναπαράσταση της εικόνας με την Laplacian πυραμίδα είναι προτιμότερη της Gaussian καθώς μπορούμε να ανακτήσουμε την αρχική εικόνα μόνο με τη χρήση της Laplacian πυραμίδας, χωρίς σφάλμα.

δ) Περιγράψτε πώς να ανακτήσετε την εικόνα χρησιμοποιώντας μόνο την πυραμίδα Laplacian. Διατυπώστε την απάντησή σας χρησιμοποιώντας μια εξίσωση που περιγράφει αυτήν τη διαδικασία. Θα μπορέσουμε να ανακτήσουμε τέλεια την αρχική εικόνα;

Αφού χρησιμοποιούμε την πυραμίδα Laplacian τότε μπορούμε να εφαρμόσουμε την γνωστή μέθοδο ανακατασκευής της εικόνας αγνοώντας όμως τη Gaussian πυραμίδα. Πιο συγκεκριμένα θα αγνοήσουμε το ανώτερο επίπεδο της Gaussian και θα πάρουμε αυτό της Laplacian. Στη συνέχεια το υπερδειγματοληπτούμε και το προσθέτουμε στην αρχική εικόνα. Παρακάτω μπορούμε να δούμε την εξίσωση από τη δημοσίευση, στην οποία φαίνεται πως μπορούμε να ανακτήσουμε την αρχική έχοντας χάσει το L του υψηλώτερου επιπέδου της πυραμίδας Gauss. Σύμφωνα με αυτά, θα έχουμε κάποια απώλεια η οποία εξαρτάται από το πλήθος των επιπέδων της Gaussian πυραμίδας. Δηλαδή όσα πιο πολλά επίπεδα έχουμε τόσο λιγότερη πληροφορία θα έχουμε στην κορυφή της πυραμίδας Gaussian με αποτέλεσμα να χάνουμε λίγη πληροφορία. Αν όμως θεωρήσουμε ως ανώτερο επίπεδο της Laplacian αυτό της Gaussian τότε μπορούμε να ανακτήσουμε την αρχική εικόνα χωρίς απώλειες.

$$g = LI + EXPAND(gI + 1)$$

2.Εργαστηριακό μέρος

Α' Μέρος: Υλοποίηση αλγορίθμου

```
In [1]: import urllib.request
import cv2
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
# sudo pip3 install opencv-python

HTTP_response = urllib.request.urlopen('http://www.image.ntua.gr/~tpar/LABimage/lena.png') #Lena image
arr = np.asarray(bytearray(HTTP_response.read()), dtype=np.uint8)
BGR = cv2.imdecode(arr,-1)
#Y = cv2.cvtColor(BGR, cv2.COLOR_BGR2YCrCb)[:, :, 0]
rgb = cv2.cvtColor(BGR, cv2.COLOR_BGR2RGB) # get the Red Green Blue version of the image
Y = rgb[:, :, :] # copy to Y

#####
# take parameter a and return Generating Kernel
def GKernel(a): # a parameter
    kernel_arr = []
    kern_c = (1/4 - a/2) # create kernel parameter c
    kernel_1d = [kern_c, 1/4, a, 1/4, kern_c] # create 1d kernel
    for m in range(0, 5):
        for n in range(0, 5):
            kernel_arr.append(kernel_1d[m] * kernel_1d[n]) # create 2d kernel
    kernel_arr = np.array(kernel_arr)
    kernel_arr = kernel_arr.reshape(5, 5) # reshape array to a [5,5] array
    return kernel_arr

# for each element in reduced picture
# return the sum given from the function below (1) page 533 of the paper
def helper(I, i, j, h): # image, i,j pixel, h kernel
    if ((i == 0) and (j == 0)): # for border elements increase to avoid negatives
        i = 1
        j = 1
    elif (i == 0):
        i = 1
    elif (j == 0):
        j = 1
    sum = 0 # initiate sum
    for m in range(0, 5):
        for n in range(0,5):
            if ((2*i + m - 2 == I.shape[0]) and (2 * j + n - 2 == I.shape[1])): # if we are out of range take border ele
                i = i -1
                j = j -1
            elif ((2*i + m - 2 == I.shape[0])):
                i = i - 1
            elif ((2 * j + n - 2 == I.shape[1])):
                j = j -1
            sum = sum + h[m][n] * I[2*i + m - 2][2 * j + n - 2] # add to sum
    return int(sum)

# for each element in expanded picture
# return the sum given from the function (2) page 534 of the paper
def helper_expand(I, i, j, h): # image, i,j pixel of image, h kernel
    sum = 0 # initialize sum
    for m in range(0, 5):
        for n in range(0,5):
            first = (i-m+2)/2
            second = (j - n + 2)/2
            if ((first - int(first) == 0) and (second - int(second) == 0)): # add to sum only for integer first, second
                if (first == I.shape[0] and second == I.shape[1]): # if we have border elements avoid out of range
                    first = first - 1
                    second = second - 1
                elif (first == I.shape[0]):
                    first = first -1
                elif (second == I.shape[1]):
                    second = second - 1
                sum = sum + (h[m][n] * I[int(first)][int(second)]) # add to sum
    return int(4*sum)

#####
# def GREDUCE(I,h): # image , kernel h
#     i_rows = int(I.shape[0]/2) # reduce size of image to half
#     i_cols = int(I.shape[1]/2)
```

```

new_image = np.zeros((i_rows, i_colms, 3), dtype = int) # create an array of zeroes for the new image
# for each pixel find the sum
for i in range(0, i_rows):
    for j in range(0, i_colms):
        new_image[i][j][0] = helper(I[:, :, 0], i, j, h)
        new_image[i][j][1] = helper(I[:, :, 1], i, j, h)
        new_image[i][j][2] = helper(I[:, :, 2], i, j, h)
return new_image

#####
d #####
#####
def GEXPAND(I, h): # image, kernel h
    i_rows = int(I.shape[0]*2) # expand size to double size (per dimension)
    i_colms = int(I.shape[1]*2)
    new_image = np.zeros((i_rows, i_colms, 3), dtype = int) # create new image of zeroes
    # for each pixel find the sum
    for i in range(0, i_rows):
        for j in range(0, i_colms):
            new_image[i][j][0] = helper_expand(I[:, :, 0], i, j, h)
            new_image[i][j][1] = helper_expand(I[:, :, 1], i, j, h)
            new_image[i][j][2] = helper_expand(I[:, :, 2], i, j, h)
    return new_image

#####
g #####
#####
# create Gaussian Pyramid based on kernel h
def GPyramid(I, a, depth): # image, a parameter, depth of pyramid
    h = GKernel(a)
    L = [I] #  $L^0$  (the base of the pyramid) is the original image
    for n in range(depth-1):
        L.append(GREDUCE(L[n],h)) # add to the pyramid the reduced image of each image
    return L

#####
e #####
#####
# create the Laplacian Pyramid
def LPyramid(I, a, depth):
    G = GPyramid(I, a, depth) # create the Gaussian pyramid
    h = GKernel(a) # create the 2d kernel

    H = [G[depth-1]] #  $L^0$  (the base of the pyramid) for the laplacian pyramid
    for n in range(depth-1):
        interpolation = GEXPAND(G[depth - n - 1], h) # get the expanded image
        difference = abs(G[depth - n - 2] - interpolation + 128) # get the difference of the images
        # for pixel values above 255 initialize to 255
        whites = difference[:, :, 0]
        for i in range(whites.shape[0]):
            for j in range(whites.shape[1]):
                if(difference[i,j,0] > 255):
                    difference[i,j,0] = 255
                if(difference[i,j,1] > 255):
                    difference[i,j,1] = 255
                if(difference[i,j,2] > 255):
                    difference[i,j,2] = 255
        H.append(difference) # add the next layer of the laplacian pyramid
    H.reverse() # reverse the laplacian pyramid to get the correct order
    return H

#####
st #####
#####
# expand each layer of the laplacian pyramid and add it to the next layer
# as described in the paper page
def LPyramidDecode(L, a):
    h = GKernel(a) # calculate the kernel h
    res = L[len(L) - 1] # start with the higher level of the laplacian pyramid
    for l in reversed(range(len(L) - 1)):
        res = GEXPAND(res, h) # expand the image
        res = res + L[l] - 128 # add it to the next level
    # for the pixels that got values above 255 make them 255 to avoid errors
    whites = res[:, :, 0]
    for i in range(whites.shape[0]):
        for j in range(whites.shape[1]):
            if(res[i,j,0] > 255):
                res[i,j,0] = 255
            if(res[i,j,1] > 255):
                res[i,j,1] = 255
            if(res[i,j,2] > 255):
                res[i,j,2] = 255

```

```

        res[i,j,2] = 255
    # for the pixels that got values below 0 make them 0 to avoid errors
    blacks = res[:, :, 0]
    for i in range(blacks.shape[0]):
        for j in range(blacks.shape[1]):
            if(res[i,j,0] < 0):
                res[i,j,0] = 0
            if(res[i,j,1] < 0):
                res[i,j,1] = 0
            if(res[i,j,2] < 0):
                res[i,j,2] = 0
    return res

#####
##### e2 #####
#### The range of pixel values is divided into bins of size n
## and the quantized value C(i, j) for pixel L(i, j) is the middle
## value of the bin which contains L(i, j)
def LQuantization(L, n):
    quantized = []
    for num in range(len(L)):
        C = np.copy(L[num])
        rows = C.shape[0]
        columns = C.shape[1]
        for i in range(rows):
            for j in range(columns):
                m1 = np.floor(L[num][i][j][0] / n)
                C[i][j][0] = m1 * n + (n / 2)
                m2 = np.floor(L[num][i][j][1] / n)
                C[i][j][1] = m2 * n + (n / 2)
                m3 = np.floor(L[num][i][j][2] / n)
                C[i][j][2] = m3 * n + (n / 2)
        quantized.append(np.copy(C))
    return quantized

## The quantized image is reconstructed through the expand and
## sum procedure LPyramidDecode using C values in the place of L values

#####
# Start Main Program
print("-----\n")
depth = 5 # Number of Levels of the pyramid
a = 3/8 # a parameter for kernel

G = GPyramid(Y, a, depth) # creates gaussian pyramid
print("Creating the Laplacian Pyramid")
Lap = LPyramid(Y, a, depth) # get the laplacian pyramid
print("Laplacian pyramid Created")

LAPD = LPyramidDecode(Lap, a) # get the decoded image
print("Decoded Image ready")

print("Creating Quantized Pyramid")
quantized_image = LQuantization(Lap, 35) # get the quantized pyramid
print("Quantized Pyramid Created")

quantized_decoded = LPyramidDecode(quantized_image, a) # get the decoded quantized image
print("Quantized Image Decoded")

#####
# Start Plotting the images

# plot the original decoded and the decoded quantized image
quan_dec, (ax0, ax1) = plt.subplots(1, 2, figsize=(20, 20))
ax0.imshow(LAPD, cmap="gray") # Base of the pyramid
ax0.set_title('Decoded', fontsize=14)
ax1.imshow(quantized_decoded, cmap="gray")
ax1.set_title('Quantized', fontsize=14)

# plot the Laplacian pyramid
f, (ax0, ax1, ax2, ax3, ax4) = plt.subplots(1, 5, figsize=(20, 20))
ax0.imshow(Lap[0], cmap="gray") # Base of the pyramid
ax0.set_title('H[0]', fontsize=14)
ax1.imshow(Lap[1], cmap="gray")
ax1.set_title('H[1]', fontsize=14)
ax2.imshow(Lap[2], cmap="gray")
ax2.set_title('H[2]', fontsize=14)
ax3.imshow(Lap[3], cmap="gray")
ax3.set_title('H[3]', fontsize=14)
ax4.imshow(Lap[4], cmap="gray")

```

```

ax4.set_title('H[4]', fontsize=14)

# plot the Gaussian pyramid
g, (ax0, ax1, ax2, ax3, ax4) = plt.subplots(1, 5, figsize=(20, 20))
ax0.imshow(G[0]) # Base of the pyramid
ax0.set_title('G[0]', fontsize=14)
ax1.imshow(G[1])
ax1.set_title('G[1]', fontsize=14)
ax2.imshow(G[2])
ax2.set_title('G[2]', fontsize=14)
ax3.imshow(G[3])
ax3.set_title('G[3]', fontsize=14)
ax4.imshow(G[4])
ax4.set_title('G[4]', fontsize=14)

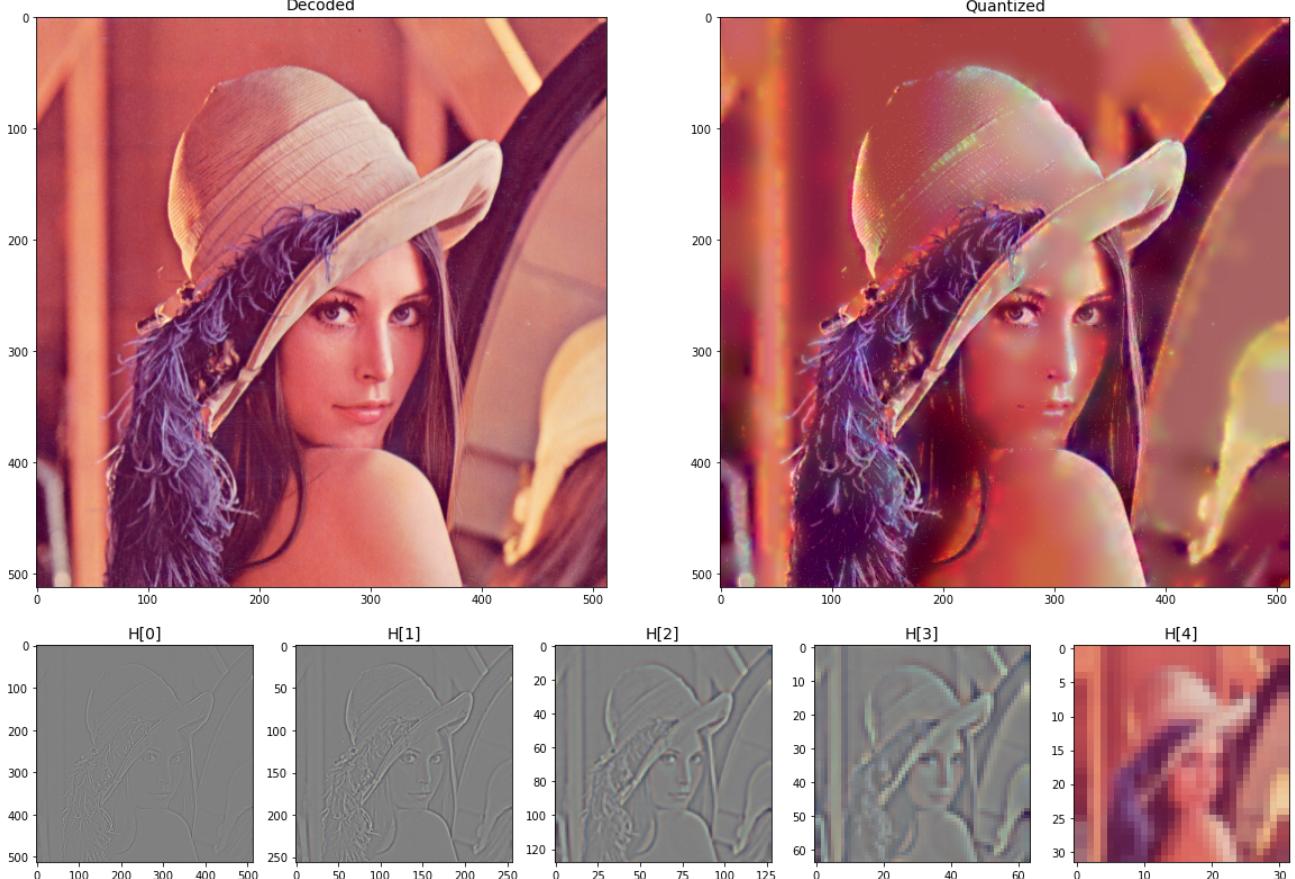
# plot the C images
q, (ax0, ax1, ax2, ax3, ax4) = plt.subplots(1, 5, figsize=(20, 20))
ax0.imshow(quantized_image[0]) # Base of the pyramid
ax0.set_title('C[0]', fontsize=14)
ax1.imshow(quantized_image[1])
ax1.set_title('C[1]', fontsize=14)
ax2.imshow(quantized_image[2])
ax2.set_title('C[2]', fontsize=14)
ax3.imshow(quantized_image[3])
ax3.set_title('C[3]', fontsize=14)
ax4.imshow(quantized_image[4])
ax4.set_title('C[4]', fontsize=14)

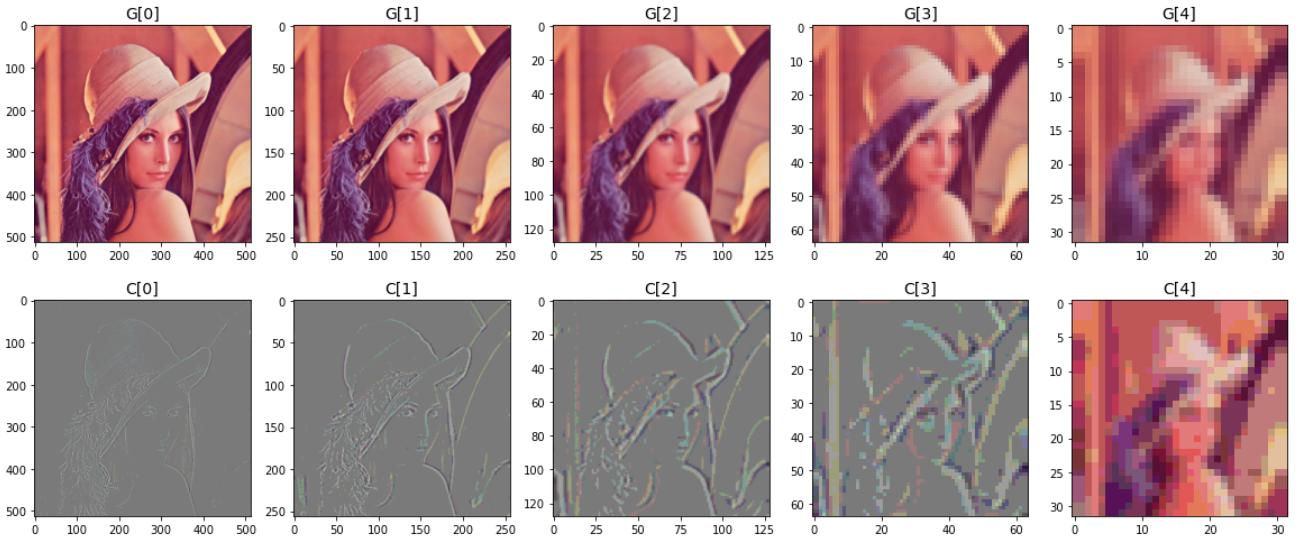
```

Creating the Laplacian Pyramid
Laplacian pyramid Created
Decoded Image ready
Creating Quantized Pyramid
Quantized Pyramid Created
Quantized Image Decoded

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Out[1]:





Παραπάνω βλέπουμε δοκιμές πάνω στην Lena. Παρατηρούμε πως η Laplacian πυραμίδα είναι πολύ αντιπροσωπευτική όπως περιμέναμε. Επίσης βλέπουμε πως η Gaussian πυραμίδα θολώνει καθώς αυξάνεται το βάθος της πυραμίδας. Επίσης βλέπουμε την decoded εικόνα που πράγματι με το μάτι δύσκολα παρατηρεί κανές διαφορά σε σχέση με την αρχική εικόνα. Τέλος, παρατηρούμε την κβαντισμένη εικόνα για την οποία έχουμε επιλέξει bins με μέγεθος 35 χρωμάτων, συνεπώς παρατηρούμε πως βγαίνει η κβαντισμένη εικόνα με 255/35 χρώματα, δηλαδή μόνο με 8 χρώματα.

B' Μέρος: Δοκιμές αλγορίθμου

α) Ελέγχετε τις υλοποιήσεις των συναρτήσεων σας *L_Pyramid* και *L_Pyramid_Decode* χρησιμοποιώντας τις εικόνες *Lena* και *camera*. Πρέπει να λειτουργούν και για έγχρωμες και για *grayscale* εικόνες.

```
In [2]: url = 'https://raw.githubusercontent.com/scikit-image/scikit-image/main skimage/data/camera.png' #camera image
HTTP_response = urllib.request.urlopen(url)
arr = np.asarray(bytearray(HTTP_response.read()), dtype=np.uint8)
BGR = cv2.imdecode(arr,-1)
rgb = cv2.cvtColor(BGR, cv2.COLOR_BGR2RGB)
camera = np.copy(rgb[:, :, :]) # get the Red Green Blue version of the camera image

HTTP_response = urllib.request.urlopen('http://www.image.ntua.gr/~tpar/LABImage/lena.png') #Lena image
arr = np.asarray(bytearray(HTTP_response.read()), dtype=np.uint8)
BGR = cv2.imdecode(arr,-1)
rgb = cv2.cvtColor(BGR, cv2.COLOR_BGR2RGB)
lena = np.copy(rgb[:, :, :]) # get the Red Green Blue of the Lena image

depth = 5 # Number of Levels of the pyramid
a = 3/8 # a parameter for kernel

Lap_lena = LPyramid(lena, a, depth) # get Lena Laplacian Pyramid
print("Lena Laplacian Pyramid Created")
LAPD_lena = LPyramidDecode(Lap_lena, a) # get Lena Decoded Image
print("Lena Decoded Image Created")

Lap_camera = LPyramid(camera, a, depth) # get Camera Laplacian Pyramid
print("Camera Laplacian Pyramid Created")
LAPD_camera = LPyramidDecode(Lap_camera, a) # get Camera Decoded Image
print("Camera Decoede Image Created")

#####
# Start Plotting

# plot Lena Laplacian Pyramid
f_lena, (ax0, ax1, ax2, ax3, ax4) = plt.subplots(1, 5, figsize=(20, 20))
ax0.imshow(Lap_lena[0], cmap="gray") # Base of the pyramid
ax0.set_title('H[0]', fontsize=14)
ax1.imshow(Lap_lena[1], cmap="gray")
ax1.set_title('H[1]', fontsize=14)
ax2.imshow(Lap_lena[2], cmap="gray")
ax2.set_title('H[2]', fontsize=14)
ax3.imshow(Lap_lena[3], cmap="gray")
ax3.set_title('H[3]', fontsize=14)
ax4.imshow(Lap_lena[4], cmap="gray")
```

```

ax4.set_title('H[4]', fontsize=14)

# Plot Camera Laplacian Pyramid
f_camera, (ax0, ax1, ax2, ax3, ax4) = plt.subplots(1, 5, figsize=(20, 20))
ax0.imshow(Lap_camera[0], cmap="gray") # Base of the pyramid
ax0.set_title('H[0]', fontsize=14)
ax1.imshow(Lap_camera[1], cmap="gray")
ax1.set_title('H[1]', fontsize=14)
ax2.imshow(Lap_camera[2], cmap="gray")
ax2.set_title('H[2]', fontsize=14)
ax3.imshow(Lap_camera[3], cmap="gray")
ax3.set_title('H[3]', fontsize=14)
ax4.imshow(Lap_camera[4], cmap="gray")
ax4.set_title('H[4]', fontsize=14)

# Plot Decoded Images
g, (ax0, ax1) = plt.subplots(1, 2, figsize=(20, 20))
ax0.imshow(LAPD_lena) # Base of the pyramid
ax0.set_title('Lena Decoded', fontsize=14)
ax1.imshow(LAPD_camera)
ax1.set_title('Camera Decoded', fontsize=14)

```

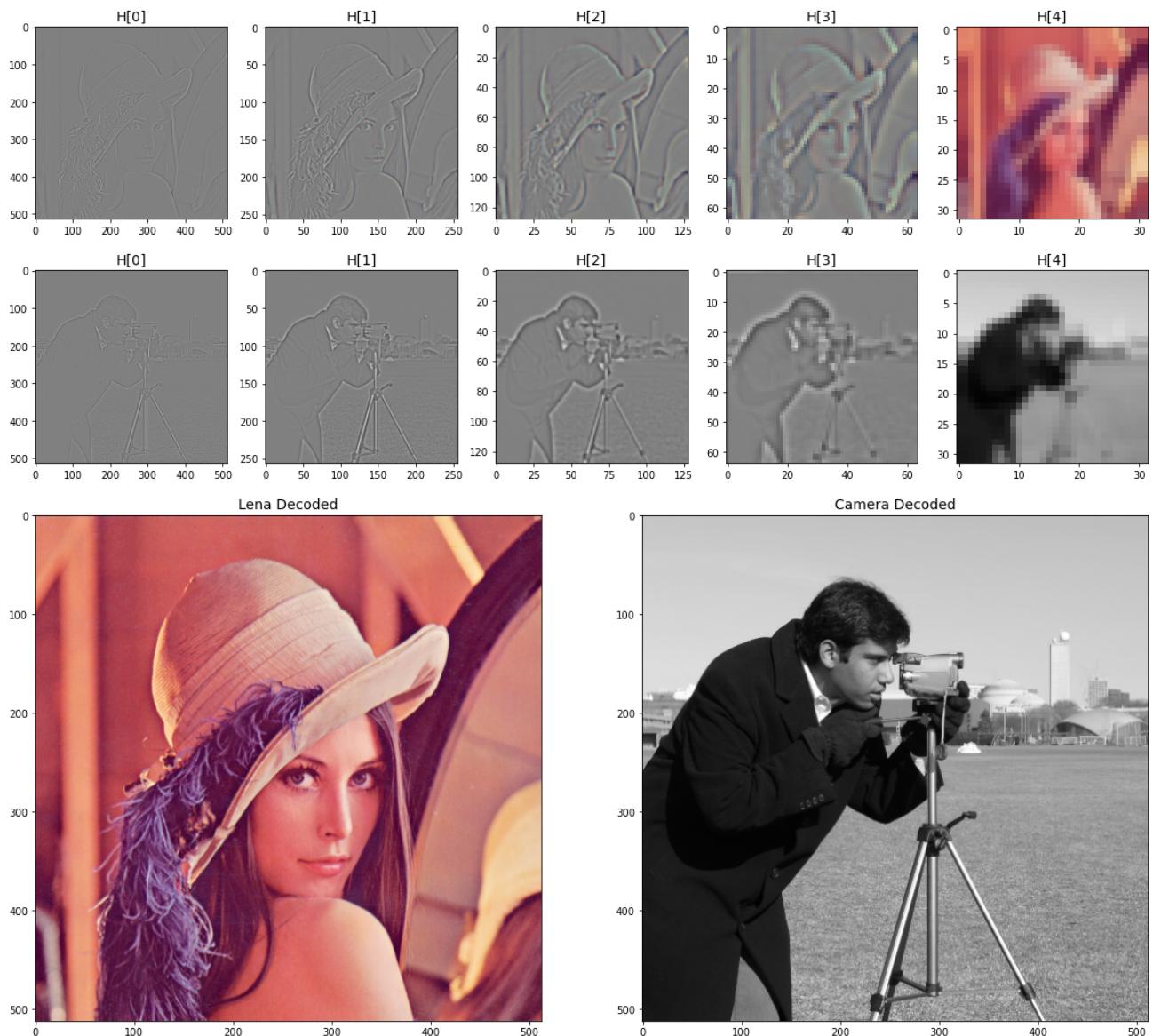
Lena Laplacian Pyramid Created

Lena Decoded Image Created

Camera Laplacian Pyramid Created

Camera Decoede Image Created

Out[2]: Text(0.5, 1.0, 'Camera Decoded')



Παραπάνω παρατηρούμε τα αποτελέσματα και για τις δύο εικόνες, τόσο για την ασπρόμαυρη "camera" όσο και για την έγχρωμη "lena".

Οι τιμές του βάθους της πυραμίδας αλλά και του α είναι επιλεγμένες κοντά στις προτεινόμενες τιμές που έχουν ειπωθεί αλλά είναι τυπικές.

β) Εμφανίστε την αρχική και την αποκωδικοποιημένη εικόνα, χρησιμοποιώντας διαφορετικά 'a' με τιμές να κυμαίνονται στο διάστημα [0.2,..,0.7].

In [3]:

```
depth = 5 # Number of Levels of the pyramid

# Create lists with original images as first elements
lena_list = [lena]
camera_list = [camera]

for a in [0.2, 0.3, 0.4, 0.5, 0.6, 0.7]:
    print("Analyzing a value: ", a)
    Lap_lena = LPyramid(lena, a, depth) # Lena Laplacian Pyramid
    lena_list.append(LPyramidDecode(Lap_lena, a)) # get Decoded Image for Lena for each a value
    Lap_camera = LPyramid(camera, a, depth) # Camera Laplacian Pyramid
    camera_list.append(LPyramidDecode(Lap_camera, a)) # get Decoded Image for Camers for each a value

# plot Lena image for each A value
f_lena, (ax0, ax1, ax2, ax3, ax4, ax5) = plt.subplots(1, 6, figsize=(20, 20))
ax0.imshow(lena_list[0])
ax0.set_title('a = 0.2', fontsize=14)
ax1.imshow(lena_list[1])
ax1.set_title('a = 0.3', fontsize=14)
ax2.imshow(lena_list[2])
ax2.set_title('a = 0.4', fontsize=14)
ax3.imshow(lena_list[3])
ax3.set_title('a = 0.5', fontsize=14)
ax4.imshow(lena_list[4])
ax4.set_title('a = 0.6', fontsize=14)
ax5.imshow(lena_list[5])
ax5.set_title('a = 0.7', fontsize=14)

# plot Camera image for each A value
f_camera, (ax0, ax1, ax2, ax3, ax4, ax5) = plt.subplots(1, 6, figsize=(20, 20))
ax0.imshow(camera_list[0])
ax0.set_title('a = 0.2', fontsize=14)
ax1.imshow(camera_list[1])
ax1.set_title('a = 0.3', fontsize=14)
ax2.imshow(camera_list[2])
ax2.set_title('a = 0.4', fontsize=14)
ax3.imshow(camera_list[3])
ax3.set_title('a = 0.5', fontsize=14)
ax4.imshow(camera_list[4])
ax4.set_title('a = 0.6', fontsize=14)
ax5.imshow(camera_list[5])
ax5.set_title('a = 0.7', fontsize=14)
```

Analyzing a value: 0.2

Analyzing a value: 0.3

Analyzing a value: 0.4

Analyzing a value: 0.5

Analyzing a value: 0.6

Analyzing a value: 0.7

Out[3]:



'Όσον αφορά τις διαφορετικές τιμές για το α, είναι δύσκολο να διακρίνει κανείς διαφορά ανάμεσα στις εικόνες ακόμα και στις

ακραίες εικόνες δηλαδή για τιμές του $\alpha = 0.2$ και $\alpha = 0.7$. Βέβαια πράγματι υπάρχει διαφορά όπως θα φανεί και παρακάτω.

γ) Εμφανίστε την αρχική και την αποκωδικοποιημένη εικόνας, χρησιμοποιώντας διαφορετικά 'depth' με τιμές να κυμαίνονται στο διάστημα [2,..,7].

In [4]:

```
a = 0.2

lena_list = [lena]
camera_list = [camera]

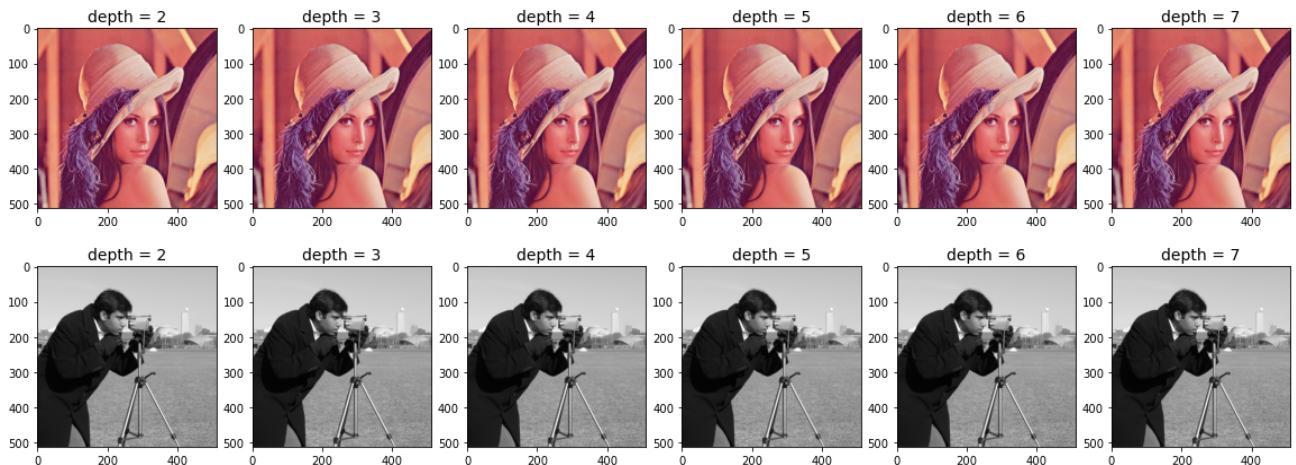
for depth in [2, 3, 4, 5, 6, 7]:
    print("Analyzing depth value : ", depth)
    Lap_lena = LPyramid(lena, a, depth)
    lena_list.append(LPyramidDecode(Lap_lena, a))
    Lap_camera = LPyramid(camera, a, depth)
    camera_list.append(LPyramidDecode(Lap_camera, a))

# plot Lena images for each depth
f_lena, (ax0, ax1, ax2, ax3, ax4, ax5) = plt.subplots(1, 6, figsize=(20, 20))
ax0.imshow(lena_list[0])
ax0.set_title('depth = 2', fontsize=14)
ax1.imshow(lena_list[1])
ax1.set_title('depth = 3', fontsize=14)
ax2.imshow(lena_list[2])
ax2.set_title('depth = 4', fontsize=14)
ax3.imshow(lena_list[3])
ax3.set_title('depth = 5', fontsize=14)
ax4.imshow(lena_list[4])
ax4.set_title('depth = 6', fontsize=14)
ax5.imshow(lena_list[5])
ax5.set_title('depth = 7', fontsize=14)

# plot Camera Images for each depth
f_camera, (ax0, ax1, ax2, ax3, ax4, ax5) = plt.subplots(1, 6, figsize=(20, 20))
ax0.imshow(camera_list[0])
ax0.set_title('depth = 2', fontsize=14)
ax1.imshow(camera_list[1])
ax1.set_title('depth = 3', fontsize=14)
ax2.imshow(camera_list[2])
ax2.set_title('depth = 4', fontsize=14)
ax3.imshow(camera_list[3])
ax3.set_title('depth = 5', fontsize=14)
ax4.imshow(camera_list[4])
ax4.set_title('depth = 6', fontsize=14)
ax5.imshow(camera_list[5])
ax5.set_title('depth = 7', fontsize=14)
```

Analyzing depth value : 2
Analyzing depth value : 3
Analyzing depth value : 4
Analyzing depth value : 5
Analyzing depth value : 6
Analyzing depth value : 7

Out[4]:



Όσον αφορά τις διαφορετικές τιμές για το depth, είναι δύσκολο να διακρίνει κανείς διαφορά ανάμεσα στις εικόνες ακόμα και στις ακραίες εικόνες δηλαδή για τιμές του depth = 2 και depth = 7. Βέβαια πράγματι υπάρχει διαφορά όπως θα φανεί και παρακάτω,

συγκεκριμένα για μικρότερο depth παρατηρείται μεγαλύτερη απόκλιση στο ιστόγραμμα της πυραμίδας.

δ) Υπολογίστε τη εντροπία και παρουσιάστε τα αντίστοιχα διαγράμματα για τα διαφορετικά 'a', και 'depth', για κάθε εικόνα και σχολιάστε επαρκώς.

```
In [ ]: import scipy.stats as st
plt.ion()
entropies = {}
variances = {}

def histograms(depth):
    print("DEPTH = ", depth)
    histogramList = []
    ## np.arange is woeful, creates stuff like this: 0.2000000000000004
    for a in [x / 100 for x in range(10, 90, 5)]: # examining more values [0.1, 0.9] step 0.05 to get a better S-a and σ
        sum_hist = np.zeros([256,1], np.float32)
        X = np.copy(lena)
        P = LPyramid(X, a, depth)
        for i in range(len(P)):
            hist = cv2.calcHist([np.float32(P[i])], [0], None, [256], [0, 256])
            sum_hist += hist

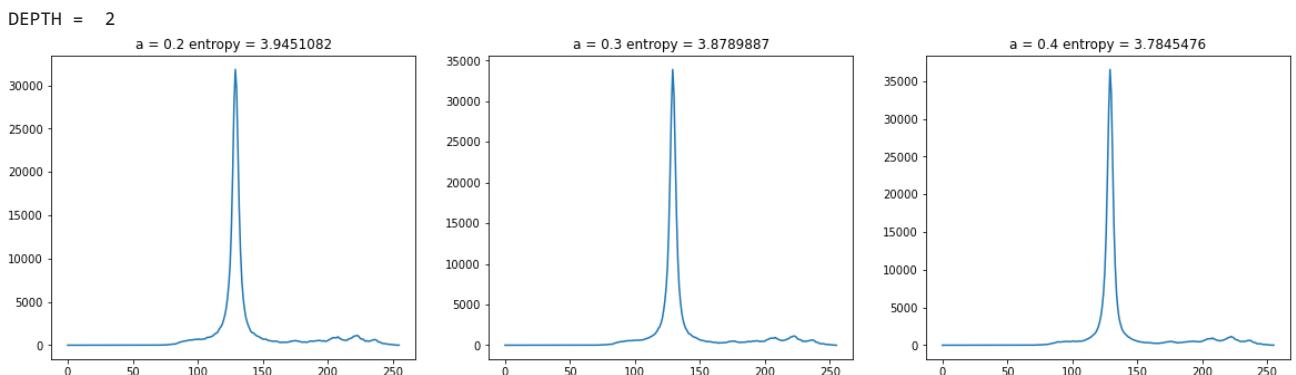
        if(a in [x / 10 for x in range(2, 8)]): histogramList.append(sum_hist)

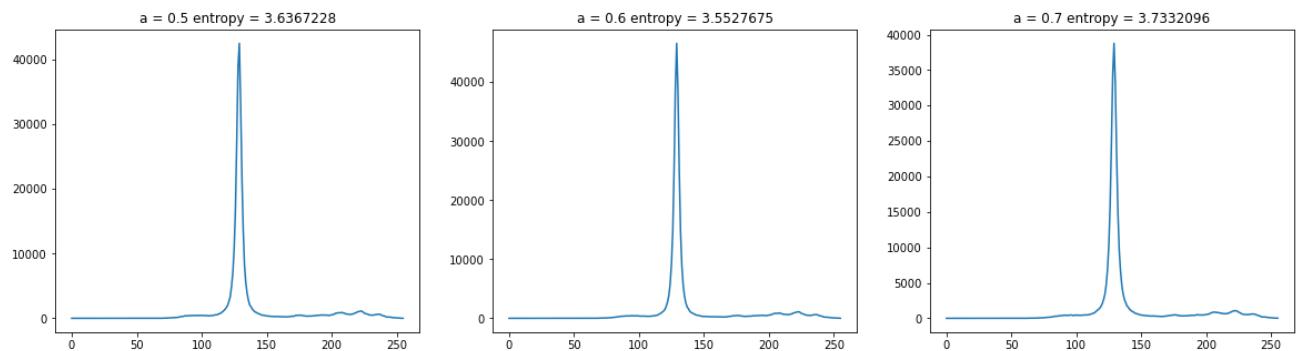
    entropy = st.entropy(sum_hist)[0]
    variance = np.var(P[0])
    entropies.update({a : entropy})
    variances.update({a : variance})

    firstRow, (ax0, ax1, ax2) = plt.subplots(1, 3, figsize=(20, 5))
    ax0.plot(histogramList[0])
    title = "a = 0.2 entropy = " + str(entropies[0.2])
    ax0.set_title(title)
    ax1.plot(histogramList[1])
    title = "a = 0.3 entropy = " + str(entropies[0.3])
    ax1.set_title(title)
    ax2.plot(histogramList[2])
    title = "a = 0.4 entropy = " + str(entropies[0.4])
    ax2.set_title(title)

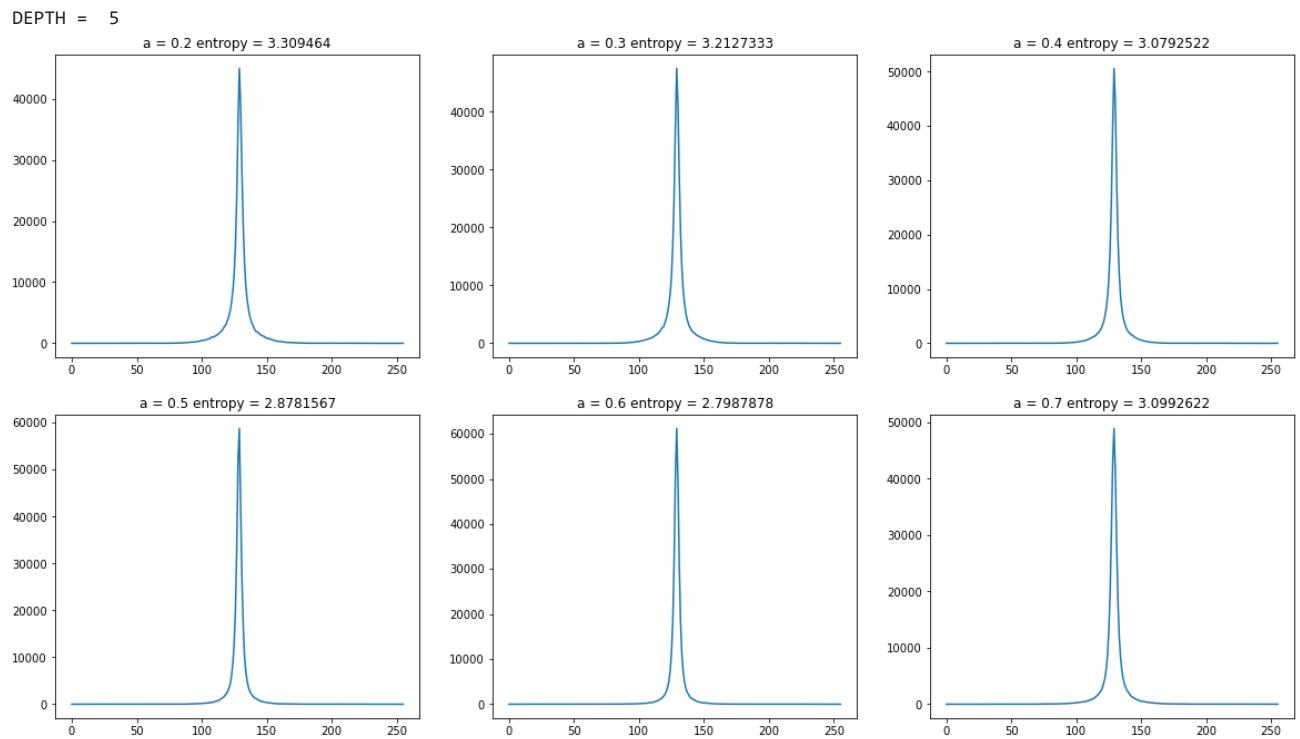
    secondRow, (ax3, ax4, ax5) = plt.subplots(1, 3, figsize=(20, 5))
    ax3.plot(histogramList[3])
    title = "a = 0.5 entropy = " + str(entropies[0.5])
    ax3.set_title(title)
    ax4.plot(histogramList[4])
    title = "a = 0.6 entropy = " + str(entropies[0.6])
    ax4.set_title(title)
    ax5.plot(histogramList[5])
    title = "a = 0.7 entropy = " + str(entropies[0.7])
    ax5.set_title(title)
```

```
In [ ]: histograms(2)
```

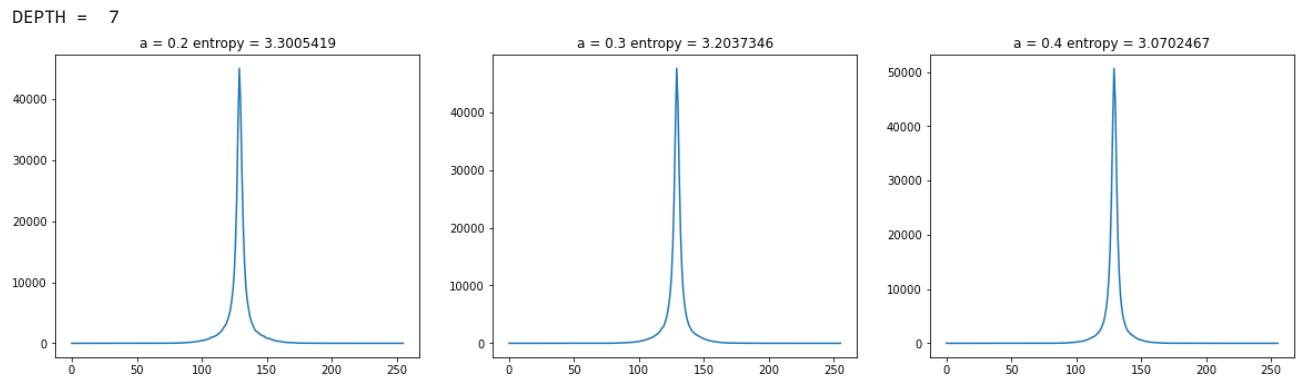


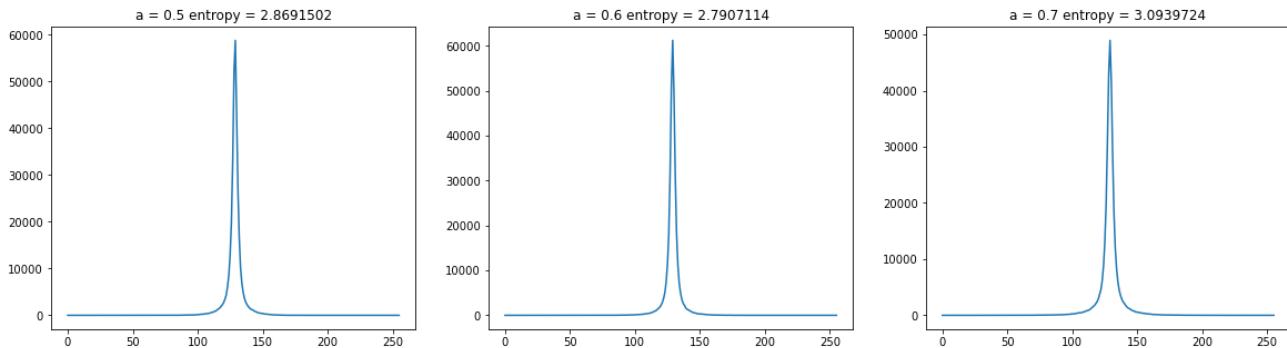


In []: `histograms(5)`



In []: `histograms(7)`

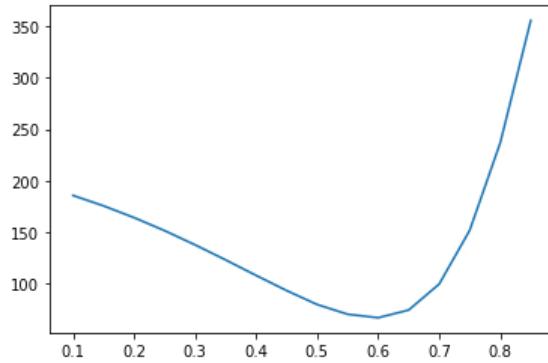
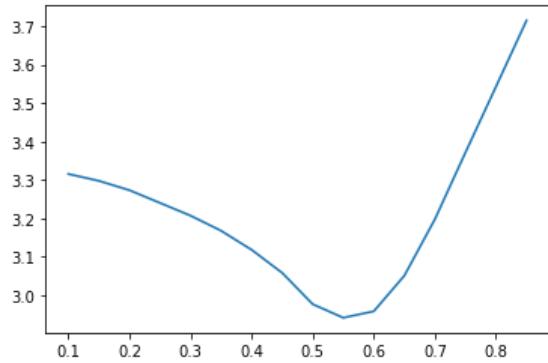




ε) Εντοπίστε το βέλτιστο 'α' χρησιμοποιώντας την εντροπία και τη διακύμανση των τιμών των εικονοστοιχείων σε κάθε επίπεδο της πυραμίδας Laplacian, για κάθε εικόνα.

In []: # Για $\text{depth} = 5$, από προηγούμενο ερώτημα

```
plt.plot(*zip(*entropies.items()))
plt.show()
plt.plot(*zip(*variances.items()))
plt.show()
```



Το ελάχιστο εντοπίζεται περίπου στο $\alpha = 0.6$

στ) Για το βέλτιστο 'α' που υπολογίσατε στο προηγούμενο ερώτημα, κβαντίστε τις εικόνες *Lena* και *camera* χρησιμοποιώντας διαφορετικά *bin size* (πραγματοποιήστε 3 διαφορετικά πειράματα για κάθε εικόνα).

In [5]: depth = 5 # Number of Levels of the pyramid

```
a = 0.6 # optimal a parameter for kernel

Lap_lena = LPyramid(lena, a, depth)
LAPD_lena = LPyramidDecode(Lap_lena, a)

Lap_camera = LPyramid(camera, a, depth)
LAPD_camera = LPyramidDecode(Lap_camera, a)

#####
plt.ion();

lenaList = []
cameraList = []
```

```

binSizes = [5, 25, 50]

for b in binSizes:
    quantLena = LQuantization(Lap_lena, b)
    quantDecLena = LPyramidDecode(quantLena, a)

    lenaList.append(quantDecLena) # saving for next question

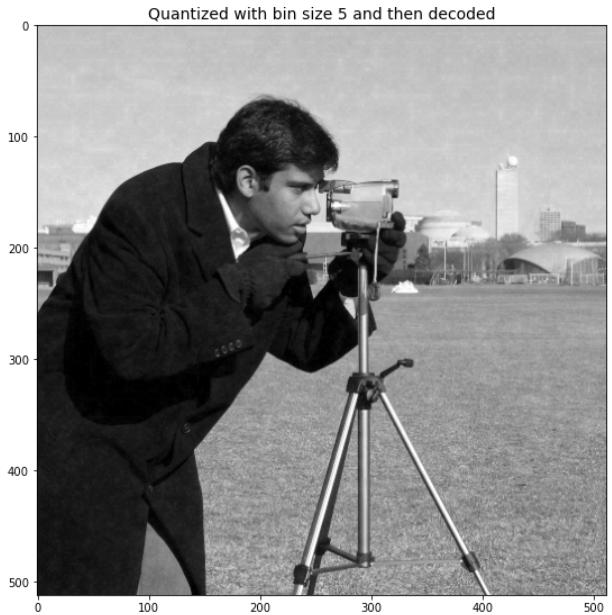
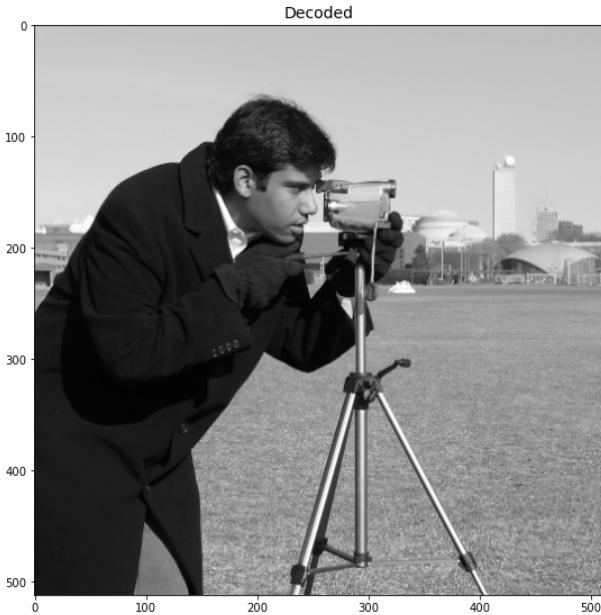
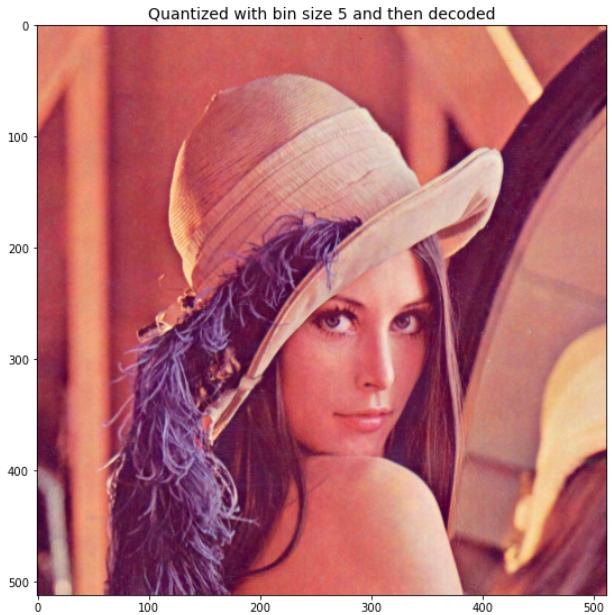
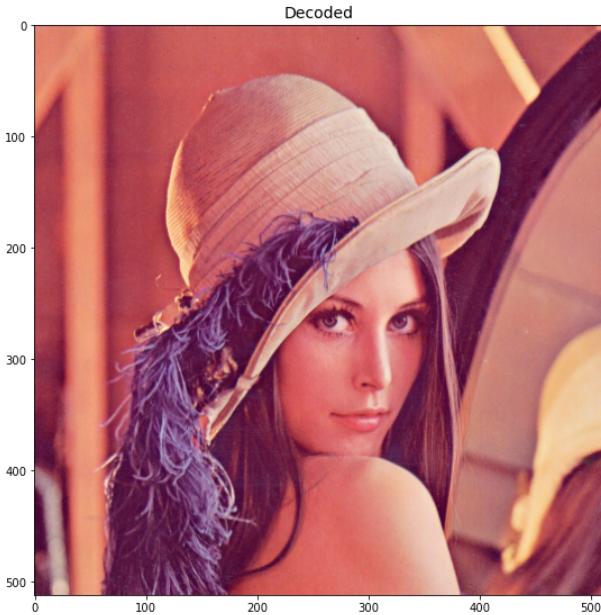
    first, (ax0, ax1) = plt.subplots(1, 2, figsize=(20, 20))
    ax0.imshow(LAPD_lena) # Base of the pyramid
    ax0.set_title('Decoded', fontsize=14)
    ax1.imshow(quantDecLena)
    title = 'Quantized with bin size ' + str(b) + ' and then decoded'
    ax1.set_title(title, fontsize=14)

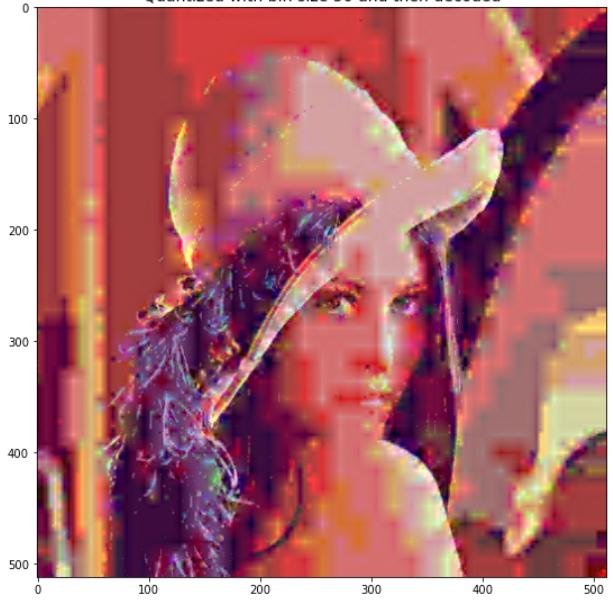
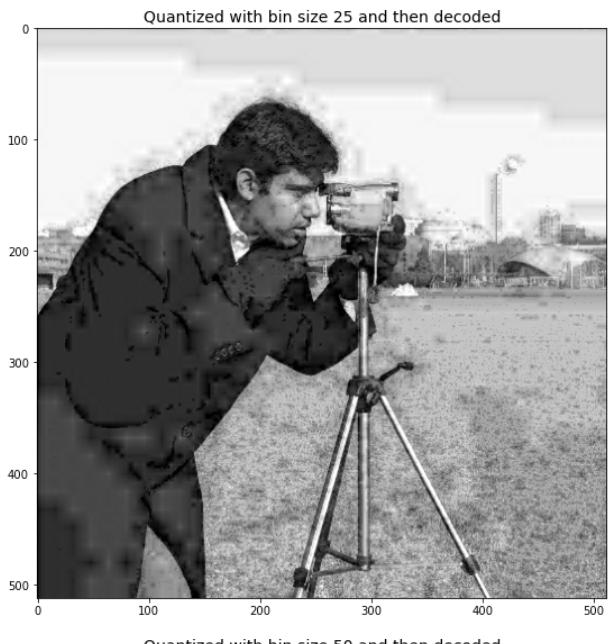
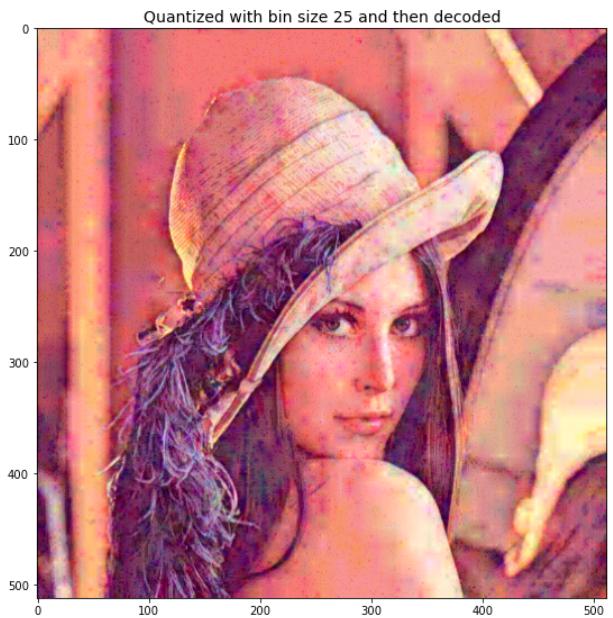
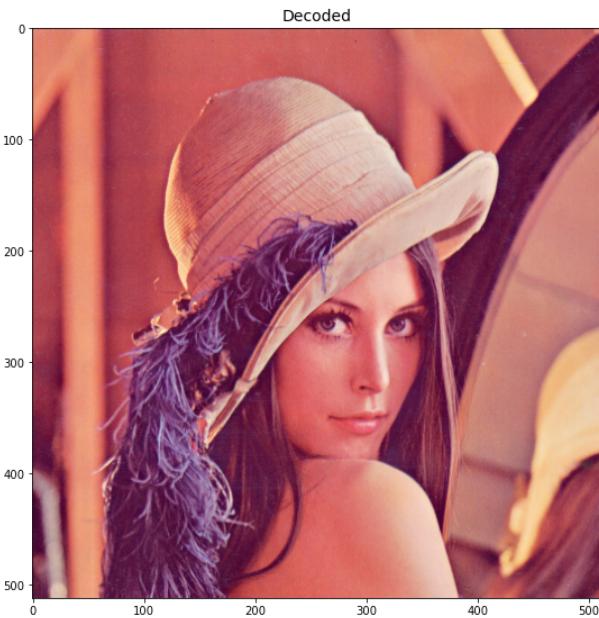
    quantCamera = LQuantization(Lap_camera, b)
    quantDecCamera = LPyramidDecode(quantCamera, a)

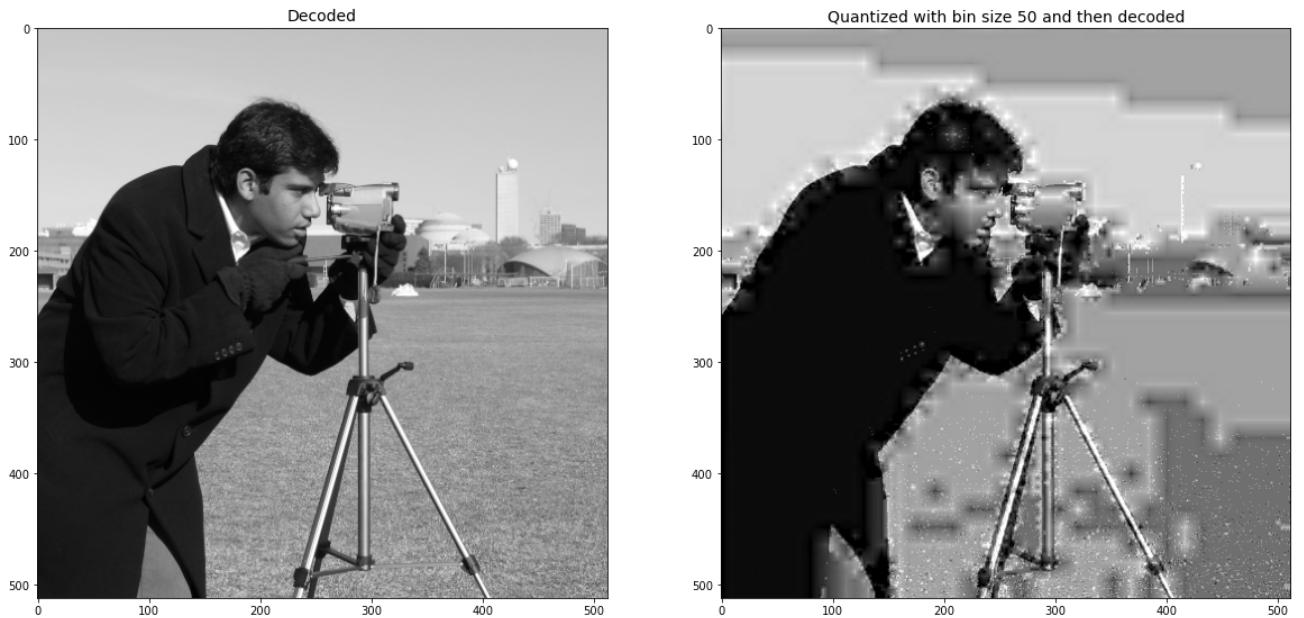
    cameraList.append(quantDecCamera)

    second, (ax2, ax3) = plt.subplots(1, 2, figsize=(20, 20))
    ax2.imshow(LAPD_camera, cmap="gray") # Base of the pyramid
    ax2.set_title('Decoded', fontsize=14)
    ax3.imshow(quantDecCamera, cmap="gray")
    ax3.set_title(title, fontsize=14)

```







Μεγαλύτερο bin size συνεπάγεται κβάντιση σε λιγότερα χρώματα και κατά συνέπεια χαμηλότερη πιστότητα της αποκωδικοποιημένης εικόνας. Παρατηρούμε πως, με σχετικά μικρό bin size, όπως 5, που επιτρέπει ~51 χρώματα, η εικόνα προκύπτουσα από αποκωδικοποίηση της κβαντισμένης πυραμίδας είναι ικανοποιητικά πιστή στην αυθεντική.

Q) Υπολογίστε και σχολιάστε το SNR της αρχικής και των ανακατασκευασμένων της εικόνων (τις 3 διαφορετικές εικόνες που προέκυψαν από τα 3 διαφορετικά πειράματα), για κάθε εικόνα (*Lena* και *camera*).

```
In [ ]: # Lab 3b-1
import math

def compute_psnr(img1, img2):
    img1 = img1.astype(np.float64) / 255.
    img2 = img2.astype(np.float64) / 255.
    mse = np.mean((img1 - img2) ** 2)
    if mse == 0:
        snr12=0
    snr12=10 * math.log10(1. / mse)

    return snr12
```

```
In [ ]: binSizes = [5, 25, 50]

print("Lena:")
for img, b in zip(lenaList, binSizes):
    print("With bin size ", b)
    snrfft = compute_psnr(img, lena)
    print('SNR FFT σε ολόκληρη την εικόνα:', snrfft)

print("\nCamera:")
for img, b in zip(cameraList, binSizes):
    print("With bin size ", b)
    snrfft = compute_psnr(img, camera)
    print('SNR FFT σε ολόκληρη την εικόνα:', snrfft)
```

```
Lena:
With bin size 5
SNR FFT σε ολόκληρη την εικόνα: 37.8201467378054
With bin size 25
SNR FFT σε ολόκληρη την εικόνα: 20.691161968403165
With bin size 50
SNR FFT σε ολόκληρη την εικόνα: 20.798435153515943
```

```
Camera:
With bin size 5
SNR FFT σε ολόκληρη την εικόνα: 36.54345799478421
With bin size 25
SNR FFT σε ολόκληρη την εικόνα: 19.628936022393283
With bin size 50
SNR FFT σε ολόκληρη την εικόνα: 20.83461732959207
```

Οι τιμές βρίσκονται εντός του αποδεκτού εύρους 20 - 40 dB.