-۳.۱

نحوه عملکرد تابع اول:(convertImageToBinary)

به طور خلاصه، این تابع یک فایل تصویر را به عنوان ورودی دریافت میکند، هر پیکسل را بر اساس شدت آن پردازش میکند، و یک نمایش دودویی از تصویر را بازمیگرداند. این نمایش دودویی یک لیست است که هر عنصر آن به یک پیکسل متناظر است و مقدار آن یا ۱ - (سفید) یا ۱ (سیاه) است.

نحوه عملکرد تابع دوم(generateNoisyImages)

این تابع به عنوان بخشی از پردازش تصویر، تصاویر ورودی را با اضافه کردن نویز تصادفی تغییر میدهد و تصاویر نویزی را در فرمتPEG ذخیره می کند. توضیحات عملکرد تابع به شرح زیر است:

ابتدا لیست مسیر فایل تصاویر تعیین شده، سپس با استفاده از توابع مورد نیاز برای بارگیری وترسیم تصاویر استفاده می شود. مقداری نویز تصادفی به هر پیکسل افزوده میشود. تصاویر نویزی با استفاده از تابع image.save در فرمت JPEG ذخیره می شوند. مسیر و نام فایلهای جدید بر اساس تصاویر اصلی وارد شده به تابع تعیین میشوند. پس از ایجاد هر تصویر نویزی، یک پیام چاپ میشود تا اعلام کند که تصویر نویزی برای تصویر ورودی ایجاد و ذخیره شده است.

به طور کلی، این تابع به تصاویر ورودی نویز افزوده و تصاویر نویزی را ذخیره می کند تا برای آزمایش وتحلیل الکوریتمها یا مدل های

مختلف استفاده شوند.

بخش 2:

از شبکه Hamming استفاده می کنیم:

ابتدا توابع مورد نیازمان را تعریف می کنیم: بردار vector را به ماتریسی با ابعاد a وb تبدیل می کنیم

```
import os
from math import sqrt

def change(vector, a, b):
  vector = np.array(vector)
  matrix = vector.reshape((a, b))
  return matrix
```

ماتریس را در بردار ضرب کرده و حاصلضربشان را با مقدار آستانه جمع میکند..

```
def product(matrix, vector, T):
  result_vector = []
  for i in range(len(matrix)):
    sum = 0
    for j in range(len(vector)):
      sum = sum + matrix[i][j] * vector[j]
    result_vector.append((sum + T))
  return result_vector
```

مقادیر بردار را با مقدار آستانه مقایسه میکند

```
def action(vector, T, Emax):
  result_vector = []
  for value in vector:
    if value <= 0:
      result_vector.append(0)
    elif 0 < value <= T:
      result_vector.append(Emax*value)
    elif value > T:
      result_vector.append(T)
  return result_vector
```

درایه های بردار یه جز درایه‌iام را با هم جمع میکنیم

```
def sum(vector, j):
  total_sum = 0
  for i in range(0, len(vector)):
    if i != j:
      total_sum = total_sum + vector[i]
  return total_sum
```

ترم بردار هارا محاسبه میکنیم

```
def norm(vector, p):
  difference = []
  for i in range(len(vector)):
    difference.append(vector[i] - p[i])
  sum = 0
  for element in difference:
    sum += element * element
  return sqrt(sum)
```

حال با استفاده از تابعconvertImageToBinaryتصاویر را به حالت باینری تبدیل میکنیم. سپس یک تصویر نویزی را انتخاب میکنیم تا شبگه همینگ را تست کنیم که تشخیص درستی دارد یا نه.

```
path = [
    '/content/1.jpg',
    '/content/2.jpg',
    '/content/3.jpg',
    '/content/4.jpg',
    '/content/5.jpg',
]
```

برای همه تصاویر این کار را امتحان میکنیم

```
x = []
for i in path:
  x.append(convertImageToBinary(i))
```

```
image_path = "/content/noisy3.jpg"
y = convertImageToBinary(image_path)

print(os.path.basename(image_path))
```

متغیرها،ماتریس وزن و ماتری سیناپس شبکه عصبی را تعریف میکنیم. و در ادامه شبکه عصبی را آموزش میدهیم. آموزش تا زمانی که نرم بردارهایyوpبزرگتر ازEmaxباشد ادامه مییابد

```
k = len(x)
a = 96
b = 96
q = change(y, a, b)
plt.matshow(q)
m = len(x[0])
T = m / 2
Emax = 0.000001
U = 1 / Emax

w = [[(x[i][j]) / 2 for j in range(m)] for i in range(k)]
e = round(1 / len(x), 1)
E = [[0 for j in range(k)] for i in range(k)]


for i in range(k):
  for j in range(k):
    if j == i:
      E[i][j] = 1.0
    else:
      E[i][j] = -e

s = [product(w, y, T)]
p = action(s[0], U, Emax)
y = [p]
i = 0
j = []
p = [0 for j in range(len(s[0]))]

while norm(y[i], p) >= Emax:
  s.append([0 for j in range(len(s[0]))])
  for j in range(len(s[0])):
    s[i + 1][j] = y[i][j] - e*sum(y[i], j)
  y.append((action(s[i + 1], U, Emax)))
  i += 1
  p = y[i - 1]

result_index = y[len(y) - 1].index(max(y[len(y) - 1])) + 1
q = change(x[result_index - 1], a, b)

from matplotlib import pyplot as plt
from matplotlib import image as img

img.imsave('output.jpg', q)
output_img = Image.open('output.jpg')
```

```
output_img = output_img.transpose(Image.FLIP_TOP_BOTTOM)
output_img = output_img.transpose(Image.ROTATE_270)
output_img.save('output.jpg')

plt.show()
image = img.imread('output.jpg')
print('\n' + '\n')
plt.imshow(image)
plt.show()
```

تصاویر خروجی:

حال مشاهده میکنیم که با افزایش مقدارnoise_factorشبکه دچار اختلال میشود. مثال به ازایnoise_factor=5000

مشاهده میشود که شبکه به جای حرف ج حرف و را تشخیص داده است

قسمت3:

با الهام از تابع دوم تابع جدیدی با اسمgenerateMissedPointImage ساختیم که این تابع همانند تابع قبلی یک مقدار رندوم را به مقادیر قرمز و سبز و آبی اضافه میکند با این تفاوت که مقادیر رندوم فقط زمانی اضافه میشوند که پیکسل مدنظر مشکی بوده

باشد.

در واقع در این حالت برخی از پیکسل های مشکی را سفید در نظر گرفتیم به همین دلیل داده دچار اختشاش شده است.

```
from PIL import Image, ImageDraw
import random

def generateMissedPointImages():
    # List of image file paths
    image_paths = [
        "/content/1.jpg",
        "/content/2.jpg",
        "/content/3.jpg",
        "/content/4.jpg",
        "/content/5.jpg"
    ]

    for i, image_path in enumerate(image_paths, start=1):
        missedpoint_image_path = f"/content/missedpoint{i}.jpg"
        getMissedPointBinaryImage(image_path, missedpoint_image_path)
        print(f"MissedPoint image for {image_path} generated and saved as
{missedpoint_image_path}")

def getMissedPointBinaryImage(input_path, output_path):
    """
    Add noise to an image and save it as a new file.

    Args:
        input_path (str): The file path to the input image.
        output_path (str): The file path to save the missedpoint image.
    """
    # Open the input image.
    image = Image.open(input_path)

    # Create a drawing tool for manipulating the image.
    draw = ImageDraw.Draw(image)
```

```
    # Determine the image's width and height in pixels.
    width = image.size[0]
    height = image.size[1]

    # Load pixel values for the image.
    pix = image.load()

    # Define a factor for introducing missedpoint.
    missedpoint_factor = 10000000

    # Loop through all pixels in the image.
    for i in range(width):
        for j in range(height):

            # Generate a random missedpoint value within the specified
factor.
            rand = random.randint(-missedpoint_factor, missedpoint_factor)

            # Add the missedpoint to the Red, Green, and Blue (RGB) values
of the pixel.
            red = pix[i, j][0]
            green = pix[i, j][1]
            blue = pix[i, j][2]
            if red == 0 & green == 0 & blue == 0:
              red = red + rand
              green = green + rand
              blue = blue + rand

            # Ensure that RGB values stay within the valid range (0-255).
              if red < 0:
                  red = 0
              if green < 0:
                  green = 0
              if blue < 0:
                  blue = 0
              if red > 255:
                  red = 255
              if green > 255:
                  green = 255
              if blue > 255:
                  blue = 255

            # Set the pixel color accordingly.
            draw.point((i, j), (red, green, blue))
```

```
    # Save the noisy image as a file.
    image.save(output_path, "JPEG")

    # Clean up the drawing tool.
    del draw

# Generate missedpoint images and save them
generateMissedPointImages()
```