IMAGE TO ASCII CONVERTER

(ASCII filling)

 

**Idea :**

1. Take an image.
2. Generate an empty canvas with dimensions same as the image dimensions.
3. Using a predefined set of ASCII characters, fill the empty canvas with the same colouring indices as were there on the actual image.
4. Loop through the character set at an interval of number of characters / total number of ASCII values.
5. Cover the image looping in the range.
6. Finally, save the image.

**Modules Needed**: PIL (Pillow library of Python, which is used to process images) and math (for importing inbuilt math functions)

Code and Explanation:

# importing the packages needed for Image Processing, ASCII font setting and image generation

from PIL import Image, ImageDraw, ImageFont

import math

#storing a predefined character list to a variable to be used in the project

chars = $@B%8&WM#\*oahkbdpqwmZO0QLCJUYXzcvunxrjft/\\|()1{}[]?-\_+~<>i!lI;:,\"^`'. "[::-1]

(the [::-1] is used to remove the null characters (/0) which are stored at the end of a custom generated array, automatically)

# storing the characters as an array and repeating them after an interval whose value = character length / 256. [eg. In this case it is 128/256 = 0.5]

# scale factor defines the size of each character with respect to page aspect. One character width and height defines the ASCII character size. Simple speaking, each character is 4 units wide and 6 units high. So, we can say, all characters are repeated after an interval of 128 characters with a size of 4x6 units.

charArray = list(chars)

charLength = len(charArray)

interval = charLength/256

scaleFactor = 0.4

oneCharWidth = 10

oneCharHeight = 15

#the purpose of this function is to accept the color variable input and multiply the floored value to the interval for each character. By this way, we can control the overflow of characters.

def getChar(inputInt):

return charArray[math.floor(inputInt\*interval)]

#form a text file with a pattern that resembles the image outlines and write the pattern in file. Then the im will be used to import and serve the reference image, and fnt will be used to set the fonts as per standard windows forms available in the Lucon forms library of Windows. These are only declarations done here and none of them are being actually run here. They will be run the primary segment. This is just the assembly line.

The im.resize will help to resize the ascii characters flow line according to the image dimensions (480x480), color (in r,g,b) and hue density (h).

After these, the image will be again reset if this doesn’t satisfy the value or there is an overlap. We can therefore see the effective use of the backtracking here.

text\_file = open("Output.txt", "w")

im = Image.open("carry.jpg")

fnt = ImageFont.truetype('C:\\Windows\\Fonts\\lucon.ttf', 16)

width, height = im.size

im = im.resize((int(scaleFactor\*width), int(scaleFactor\*height\*(oneCharWidth/oneCharHeight))), Image.NEAREST)

width, height = im.size

pix = im.load()

# This segment is an auxiliary section where the image will be generated. So, the final image with imposed ASCII will need to have a character width and height, as well as image heights and widths and a colouring overlay value, initially set to 0s. The drawing head will draw the actual image (represented by d)

outputImage = Image.new('RGB', (oneCharWidth \* width, oneCharHeight \* height), color = (0, 0, 0))

d = ImageDraw.Draw(outputImage)

# Actual Logic : (The code engine or the main component)

A pointer i will print characters along height(column control loop)

A pointer j will print characters along the width(row control loop)

r,g,b values (co-ordinates) will be assigned to the pix[j,i] matrix. This matrix represents a tiny pixel of the image of “Carry”. Now, hue density is given the respective r,g,b component magnitudes of the r,g,b color vectors. Now, these values are again swapped into the pix matrix. The reason is, the new colors that it has got are not the colors of the image bits (the colors denoted by r/3,b/3…). Rather, they are image bits of the ASCII values. Now, since the spaces are created and colors are defined, we will write our predefined text file characters, that too following the method defined earlier in the getChar function. The inputInt is replaced the value of h. But as h was initially float (as you can see it is r/3 + b/3 +g/3), so its explicitly type casted to integer. The image drawing head will now get the dimensions of the image to be output, the character dimensions and font, filling colors and will be sufficiently equipped to do it’s job. The text file will finally generate the pattern to be imposed, like normal patterns with a line-break.

for i in range(height):

for j in range(width):

r, g, b = pix[j, i]

h = int(r/3 + g/3 + b/3)

pix[j, i] = (h, h, h)

text\_file.write(getChar(h))

d.text((j\*oneCharWidth, i\*oneCharHeight), getChar(h), font = fnt, fill = (r, g, b))

text\_file.write('\n')

# Finally, the image will be saved as expected.

outputImage.save('output.png')

N.B.: Please don’t forget to make enter accurate spacing else code may show errors as it did the previous day.