Ramakrishna Mission Vivekananda Educational & Research Institute

Belur Math, Howrah, West Bengal

School of Mathematical Sciences, Department of Computer Science

Assignment - 1

M.Sc. Computer Science and Big Data Analytics Date: 15 January 2025

Course : **CS411: Applications of Computer Vision and Deep Learning**

Deadline: 5th February 2025, 11:59 P.M.

Instructor: Jimut Bahan Pal Max marks: 95

Instructions: **Attempt all the questions** Submit the code as **Name_ROLL.zip** and send it as a REPLY EMAIL **(to only *Jimut Bahan Pal*)** as Carbon Copy (CC) **jimutbahanpal@yahoo.com** AND **jpal.cs@gm.rkmvu.ac.in**. You may also CC it to your other personal email address to check whether the email has gone to the address. Using of AI tools, referring the internet, collaborations etc. are allowed given that you have properly attributed each one of those. **Submission after the deadline will fetch you -5 marks per day, so start early!**

In this assignment, we will go through the process of **ASCII art generation**. ASCII art is a process of converting images or any graphics to ASCII/Unicode text so that the image can be displayed on terminals. There are many variants of ASCII art, we will start by creating a very simple variant by taking into account the intensities of the image, and then we will move into a Machine Learning variant where the quality of the art is ***"expected"*** to be better than the previous one. We will then go a step ahead and compile videos to ASCII art by passing any video and getting a series of ASCII art on the terminal. Most of the design choices will be left to the students to decide the quality of the ASCII art produced, so please feel free to think *out of the box*.

1. **Simple ASCII Art Generator** - In this part, you will be creating a basic ASCII art generator, which will be the foundation of the next parts. (25)

   (a) Read an input image in OpenCV. Convert it to Grayscale. We will be representing the placeholder for the ASCII as cells. Cells are formed by representing the Image as rows (R) and columns (C) in the ASCII Text. Show the image as a grayscale image. **(1)**

   (b) Let's restrict the ASCII art to 80 columns for proper representation in the terminal. So, you have to scale the image properly to have a proper mapping between the 80 columns and variable number of rows which will form the total number of cells to be filled by the ASCII characters. The number of rows is to be calculated by using the proper aspect ratio, so that the original aspect ratio of the images and the texts remains intact. Say, for example, an image is of size 480x320 (WxH), and you have taken the "Courier font", which takes about 1x1 cell, so you have to convert the image to have 80 columns (unit) of width, which means you have to divide C=480/80=6. Hence, the tile size will be 6x6 here, which is the unit cell from the image, note that the unit cell from the text is always 1x1. So, you will be mapping a 6x6 tile of the image with a 1x1 text of the "Courier font." Similarly, for the rows calculation, you will need about R=320/6=53.33 rows. You can take the *ceil(R)-1*, i.e., 53 rows for this task. Therefore a 480x320 image will be mapped to an 80x53 sized Column $\times$ Rows (C$\times$R) matrix which will be the ASCII Art. (i) show the dimensions of the image, (ii) show the scaled ratio for the columns, (iii) show the scaled dimension of the image having 80 columns (iv) show the patch size dimensions, (v) show the grid having the rows$\times$columns in the actual image **(1+1+1+1+4=8)**

   (c) For mapping the cells to the font, you can have different levels of gray for this task. You can start with 10 levels of gray and then move to 70 levels of gray. The levels of gray mapping are shown in the following text:

```
# 70 levels of gray
```

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

# 10 levels of gray
'@%#*+=-:. '
```

So, how you are going to map this is, in general, you will be taking the 6x6 patch from the image. Now, average the values (intensities) of this patch. This will make the average value between 0-255. If the values are not between 0-255, you need to re-normalize between this value, by dividing by maximum and multiplying by 255. Now, if you are using 10 levels of gray, then have a slab of 255/10=25.5 for each of the letters. So, the initial character array/list will be ['@', '%', '#', '*', '+', '=', '-', ':', '.' , ' '], so if a patch's average is in between 0-25.5 then it will be replaced by '@', similarly, if the patch's average is 90, then we will take the ceil(90/25.5)-1=4-1=3, hence, 3rd index i.e., '*' and replace the patch with this text. Show the 10 levels of gray value (i.e., the averaged value) in the grid of the image having rows $\times$ columns (here columns=80), and also map the gray value from the 10 levels of gray to the actual mapping. Similarly, do this for the 70 levels of gray, where the actual mapping refers to the selection of the characters from the list provided. **(1+3+1+3=8)**

(d) Create a script that takes the image file <file_name.png> as an argument, and also the <text_name.txt> as an argument and converts the image file to this text file, so using cat <text_name.txt> in the Linux terminal, we can see the ASCII art generated. You need to do this for the 10-level and 70-level text format and save the generated ASCII art in txt as FILENAME_10_level.txt and FILENAME_70_level.txt and also save the screenshot of the ASCII art as FILENAME_10_level.png and FILENAME_70_level.png, if <FILENAME> was your input file's name without .png extension. Keep the original input image too. Do this for another image and save the 5 files, i.e., 10 level text and image, 70 level text and image and the original image files. Make a report and put the images in there along with the question numbers and the parts. Show all the 5 files **(8)**

2. **Data Science based ASCII Art Generator** - Now that you have created the Simple ASCII art generator by using the Average value of the patches, we will modify this function to get a more (40)

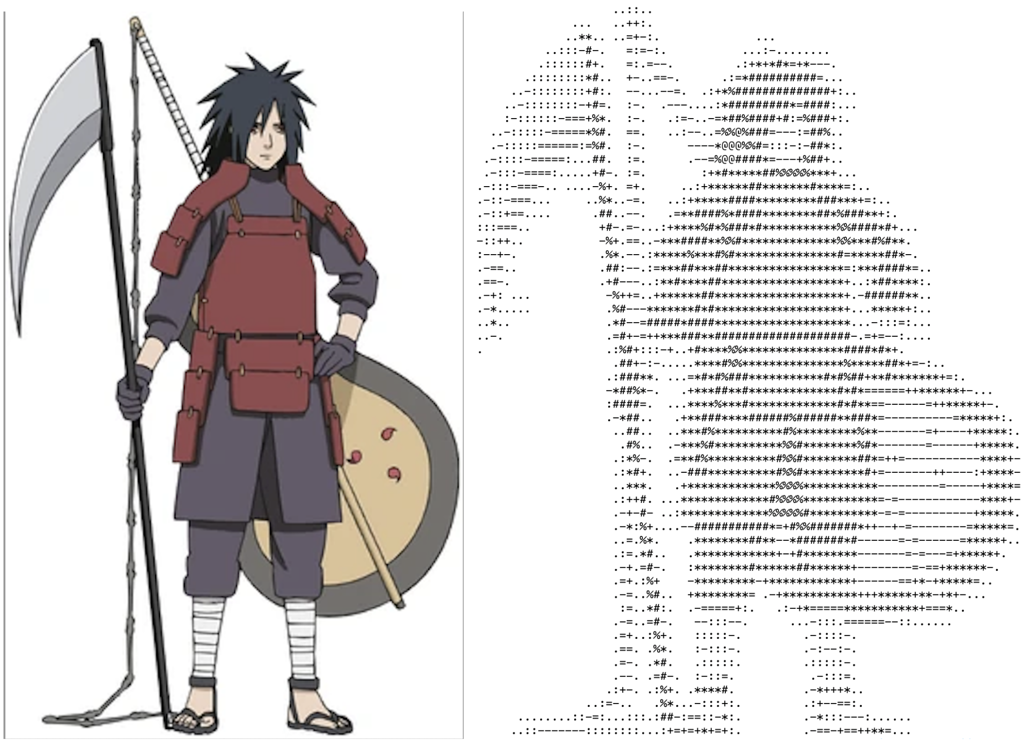
Figure 1: Simple ASCII art generated (on right) by passing a figure (on left) using '10 levels of Grey'.

meaningful and representational value for each patch using Machine Learning. In this particular setting, we will use PCA; you can use any standard library of your choice. Please feel free to make suitable assumptions about the design choices which suit your purpose. This part is **'expected'** to give better results than the previous versions.

(a) Make a new function that takes an image patch and finds the PCA along with the number of components as input. Make specific assumptions and design decisions by yourself on the selection of libraries for this task. The output/return value of this function will be a *k*-length vector, where *k* is the number of components in the PCA. For example, if you take a patch of size 7x7 as input to this function which has the character 'a' (image) in it, then the output of the function with another character which is close to 'a' will have similar value, i.e., the L2 norm of the two vectors will be small. The same goes for '1' and 'l,' where I have passed 'one' and the letter 'l' as an image. Show how the PCA function works, for example, pass PCA('A', 6) which will convert the character 'A' to the image and apply PCA on it, and have a PCA reduced vector of size k. Show the PCA vector value, with different sizes of vector, i.e., 3,5,7. Now with the PCA vectors got from the font and the PCA vectors got from the patch, find the different mapping and show the different texts got using the different number of PCA components. **(2+8=10)**

(b) Take 'two' fonts from your choice and create a function that will produce images by passing them letters as an argument. For example, if I use 'Calibri' as a font, then I will get Calibri-style images of a particular character, and the same goes for 'Times New Roman.' We want this since we want diversity in the textual images as a form of uni-code. How this is different from the previous part-1 (b) is, that we are using two fonts in the textual visualization of the ASCII art to have rich representation. Show this for the two fonts working using different PCA components (3,5,7). So, now, two fonts, 1 image, 6 files having ASCII arts as the texts should be presented. **(10)**

(c) Now, once you have created the above two functions, you will pool the two fonts' available images into a folder, which will be a search space directory/database. Use proper naming to the 'letters' and store them. You will extract the patch from the input image (which is converted to grayscale) and pass it to the function which will give *k*-length vectors as output, let us call this ***V***. Similarly you will pass all of the 'letter' images into that function to get a database of *k*-length vectors, let us call this as ***L***. You will compare each of L_i's with ***V*** to find the closest of all the L_i's, i.e., using the $L1$ norm, and select the mapping that has the lowest $L1$ distance. This way you will find the closest-looking mapping for the ASCII art image. **(10)**

(d) Fill up the image with the closest-looking mapping and store them in a similar way using similar techniques from the previous part into a text file. Repeat Part-1(d) for this task and store the files in a similar way, here you will store the original image, the text file generated, and the text file's PNG version. Compare the two images, i.e., the ASCII art got from Part 1 and the ASCII art got from Part-2, and see whether the things are improving or not. Put the images generated into the report and write how it is better in terms of representational capabilties. **(10)**

3. **Movie ASCII Art Generator** - Now that you have created the PCA ASCII art generator by using the PCA of the patches, we will modify that to get an ASCII art of a video. (Think of a hypothetical situation where you have downloaded a video file on a remote FTP Server, and you have to check the authenticity of the file on the terminal itself, you will check the first few frames in the terminal. Well, you have to modify this much more to make it work like that, but this works fine, for verifying the semantic content of the video via terminal, also gives a hacker-like feel. You can even modify this assignment to have 8-bit color code encoding to make it work, but that's a bit complicated, let's focus on the easy part first.) (30)

(a) Select a video of say 20 seconds of your choice and modify/add functions to the previous script which will take the video, convert it to frames, convert the frames to gray-scale images and apply PCA ASCII art generator on it to create ASCII texts per frames and store it in a folder. You may choose the one which is performing better from Task-1 and Task-2. So, for getting full marks in this part, if you successfully convert the video into frames which has a folder named frames and all the frames i.e., 1.png, 2.png, 3.png etc., and another folder having names as ascii_art having 1.txt, 2.txt etc., which corresponds to the frames extracted, then you will have full marks. **(10)**

(b) Create a script which will read these texts and display a movie in the terminal itself. Use some sort of video recording tools to capture this and save the file as VIDEO.mp4 or any media which can be opened in a browser. Or, read the text files one by one and display it on the terminal, clearing the previous frame's text file giving an animation effect. **(10)**

(c) Create a report (of the insights and paste the results according to the numbering of questions) and have all the files in order. The details of the report will be left to the user's choice, do anything interesting and submit the ASCII art generated in the report. **The best report will be show-cased in the course website page along with the quality of the ASCII art generated in the first version, the PCA version and the movie version.** **(10)**