Project: ASCII Art Animation

Team : #awfullyPYTHONIC

Members: 金淑媛（517120910199） 王倩淑（517120910204）

杨思佳（517120910206） 尹悦舟（517120910208）

1 Introduction

1.1 Function

ASCII Art Animation can change a flash into a new one combined by symbols. After finding the way to change a picture into ASCII Art, we considered whether we could make a short video or a GIF into the same style. It can help gain new feelings and will certainly increase the fun of watching an animated picture or a simple video.

1.2 Methods

First, it is important for us to get some modules ---- OpenCV, numpy, os, argparse, subprocess.

1) OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.
2) The numpy system has a powerful N dimensional array object. This tool can be used to store and process large matrices, which is much more efficient than the nested list structure of Python itself.
3) The os module provides a portable way of using operating system dependent functionality.
4) The argparse module makes it easy to write user-friendly command-line interfaces and automatically generates help and usage messages and issues errors when users give the program invalid arguments.
5) The subprocess module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.

(From https://docs.python.org/3/)

Second, we decided the main route we would take in the next few weeks. And the steps are clearly showed in part 3.

Third, we ran it and found some problems. We discussed together and have solved some of them. Through this process, we tried our best to find solutions and turned to the Internet for help.

1.3 Key factor

Roughly, our code consists of two parts ---- to turn animation into pictures and to change those pictures into new styles. And the key factor we believe is to convert each frame with lots of colors into Grayscale images.

2 Project processes

2.1 How did we make decisions?

We discussed together to make most decisions. We had offline discussions, usually after the class was over. Offline discussions mostly focused on the general steps to take and everyone can air her opinions freely as long as she wanted. Also, we spent large amount of time discussing online. We posed questions, solved it together and revised some details through online discussion.

2.2 Our allocation

金淑媛 mainly focused on making PPT.

王倩淑 mainly focused on writing and revising the code.

杨思佳 mainly focused on writing and revising the report.

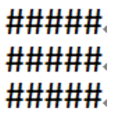
尹悦舟 mainly focused on making PPT.

And we presented together!

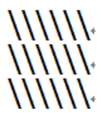
3 Algorithm Description

3.1 Explore ways to convert pictures into ASCII character paintings

An ASCII character painting is a combination of series of characters. We think of characters as relatively large pixels. A character can express a color. Look at the following two series of characters:



```
#####
#####
#####
```



```
\\\\\\
\\\\\\
\\\\\\
```

We can see that the whole color of the first picture is a little darker. According to this, different characters are used to represent different gray values. The type and number of characters can be adjusted according to our own needs, but it is best to see obvious changes after going:

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

The final display is black and white character painting, so the color pictures need to be converted in to black and white. Here we introduce the concept of gray value.

Gray value refers to the color depth at the midpoint of a black-and-white image. The range is generally 0 to 255 (255 for white and 0 for black). Therefore, a black-and-white image is also called a gray image. (from interactive encyclopedia)

We can use the following formula to convert RGB values of pixels into gray-scale values:

$$Gray = 0.2126 * r + 0.7152 * g + 0.0722 * b$$

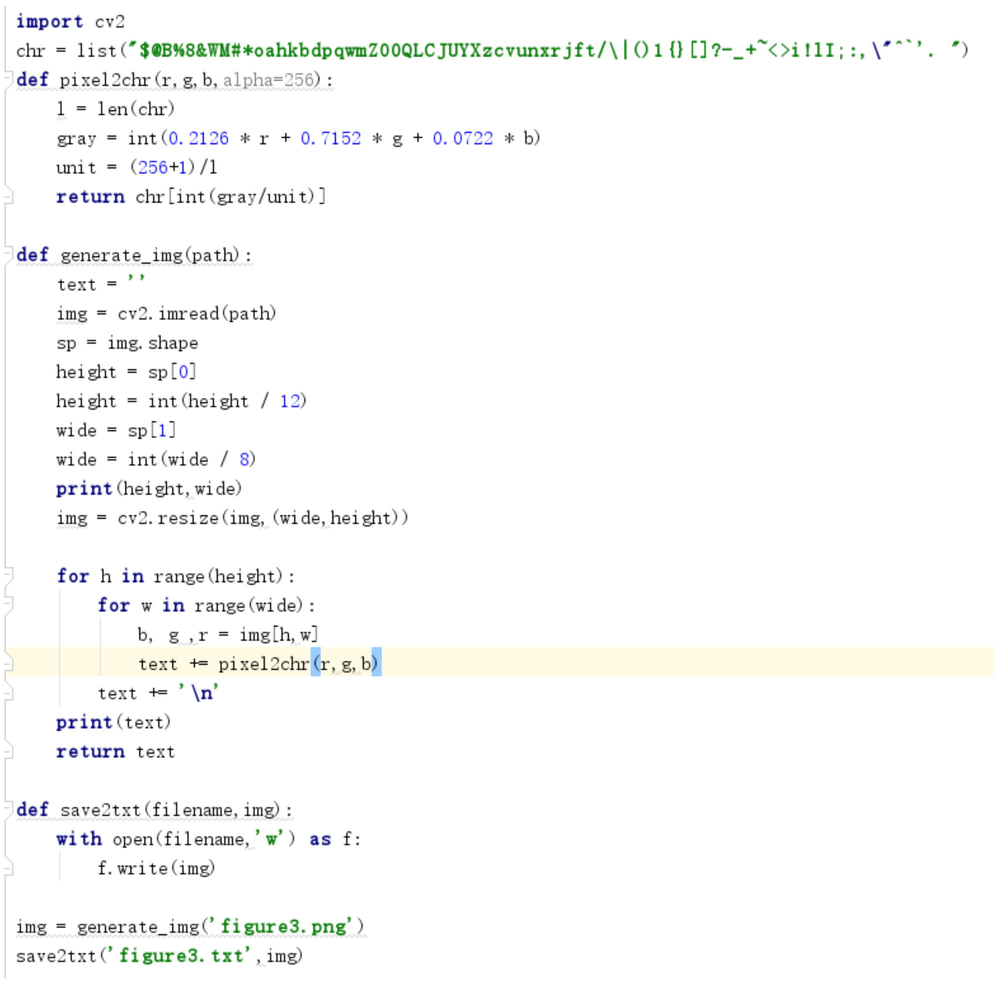
Or we can directly use the function in cv2.

On the basis of the above contents, we use the following ideas for conversion:

First, correspond each character to the gray value of a section.

Second, divide the picture into appropriate pixel blocks, calculate the gray value of each block line by line, and correspond to the characters.

Last, write characters line by line to a .txt file and generate pictures.



```
import cv2
chr = list("$@B%8&WM#*oahkbdpqwmZO0QLCJUYXzcvunxrjft/\|()1{}[]?-_+~<>i!lI;:,\"^`'. ")
def pixel2chr(r, g, b, alpha=256):
    l = len(chr)
    gray = int(0.2126 * r + 0.7152 * g + 0.0722 * b)
    unit = (256+1)/l
    return chr[int(gray/unit)]

def generate_img(path):
    text = ''
    img = cv2.imread(path)
    sp = img.shape
    height = sp[0]
    height = int(height / 12)
    wide = sp[1]
    wide = int(wide / 8)
    print(height, wide)
    img = cv2.resize(img, (wide, height))

    for h in range(height):
        for w in range(wide):
            b,  g , r = img[h, w]
            text += pixel2chr(r, g, b)
        text += '\n'
    print(text)
    return text

def save2txt(filename, img):
    with open(filename, 'w') as f:
        f.write(img)

img = generate_img('figure3.png')
save2txt('figure3.txt', img)
```

(code of testing phase)

The following is the effect of the conversion:

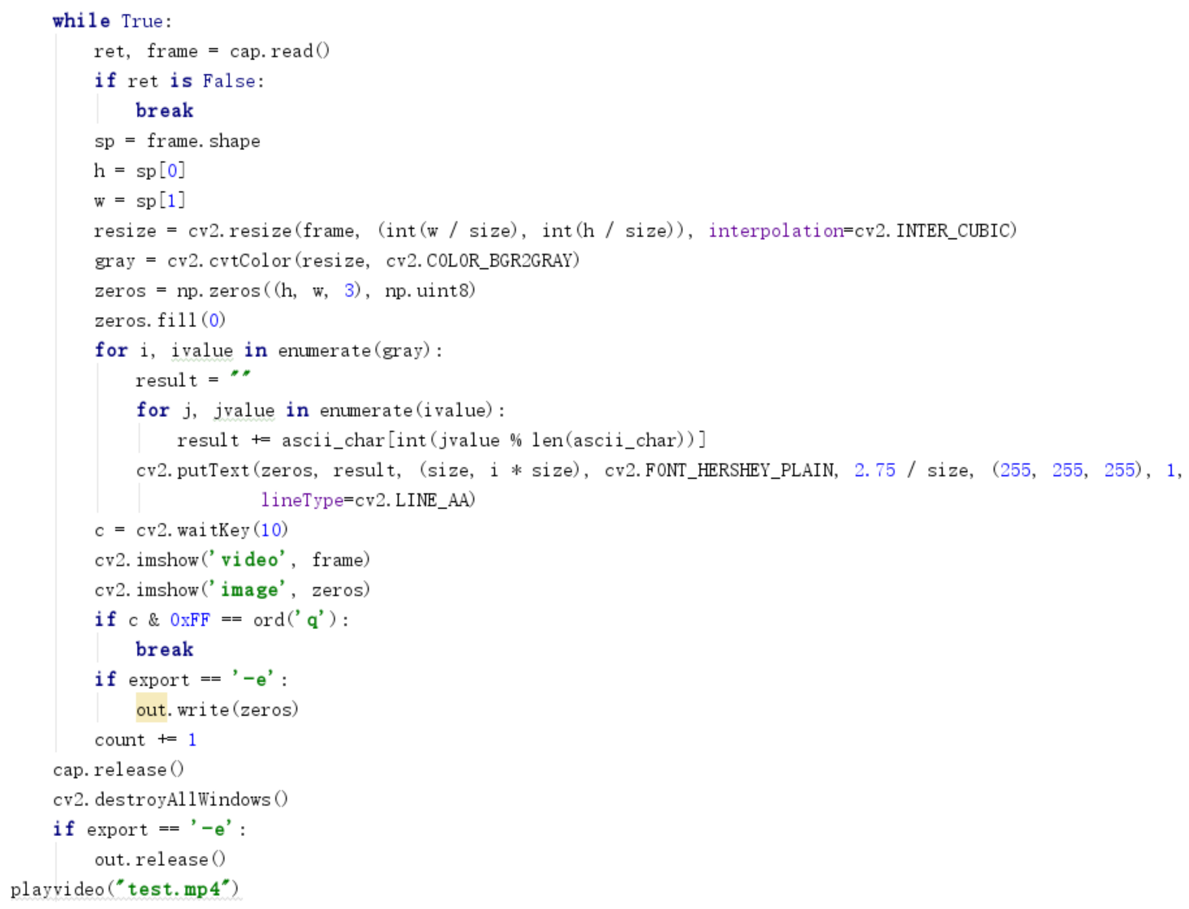
The original picture

The converted picture

3.2 Explore ways to convert video to character video

We all know that video is made up of many frames of pictures. Based on previous results, to convert video into character video, the core step is to read every frame in the video in a loop and then convert it into a character picture.

```
import ...
def playvideo(video_idx, size = 5, export = None):
    size = int(size)
    cap = cv2.VideoCapture(video_idx)
    ascii_char = list("$B%8&WM#*oahkbdpqwmZO0QLCJUYXzcvunxrjft/\|()1{}[]?-_+~<>i!lI;:, ")
    if export is '-e':
        fourcc = cv2.VideoWriter_fourcc(*'MJPG')
        out = cv2.VideoWriter('output.avi', fourcc, cap.get(cv2.CAP_PROP_FPS), (640, 360))
    count=0
    cv2.namedWindow('video')
    cv2.namedWindow('image')
```

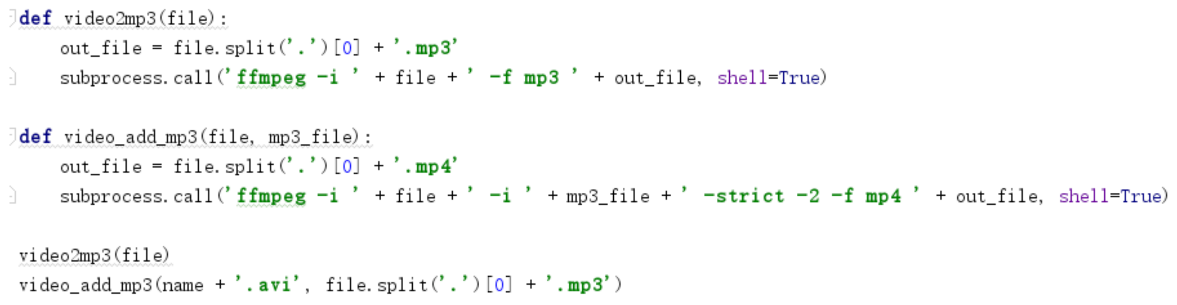


```python
    while True:
        ret, frame = cap.read()
        if ret is False:
            break
        sp = frame.shape
        h = sp[0]
        w = sp[1]
        resize = cv2.resize(frame, (int(w / size), int(h / size)), interpolation=cv2.INTER_CUBIC)
        gray = cv2.cvtColor(resize, cv2.COLOR_BGR2GRAY)
        zeros = np.zeros((h, w, 3), np.uint8)
        zeros.fill(0)
        for i, ivalue in enumerate(gray):
            result = ""
            for j, jvalue in enumerate(ivalue):
                result += ascii_char[int(jvalue % len(ascii_char))]
            cv2.putText(zeros, result, (size, i * size), cv2.FONT_HERSHEY_PLAIN, 2.75 / size, (255, 255, 255), 1,
                        lineType=cv2.LINE_AA)
        c = cv2.waitKey(10)
        cv2.imshow('video', frame)
        cv2.imshow('image', zeros)
        if c & 0xFF == ord('q'):
            break
        if export == '-e':
            out.write(zeros)
        count += 1
    cap.release()
    cv2.destroyAllWindows()
    if export == '-e':
        out.release()
playvideo("test.mp4")
```

(code of testing phase)

Although the animation without sound is able to watch, people may get tired of seeing it twice.

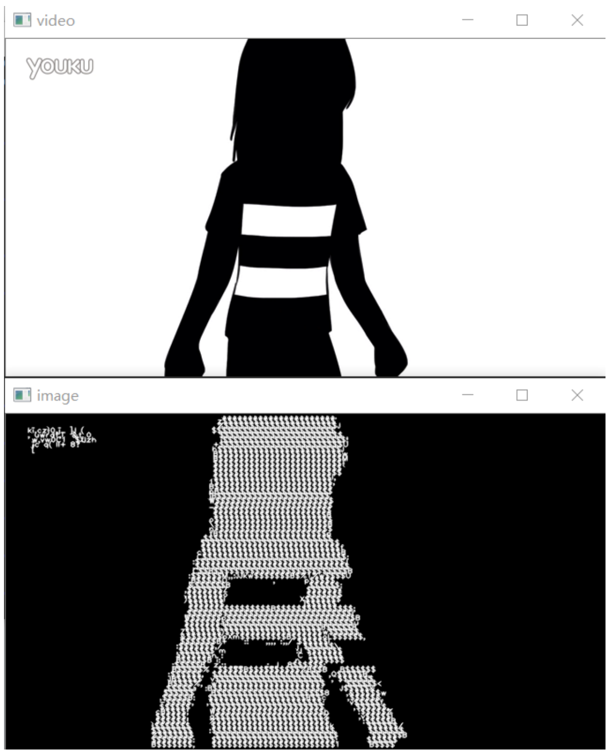
So then we add original sound to it.



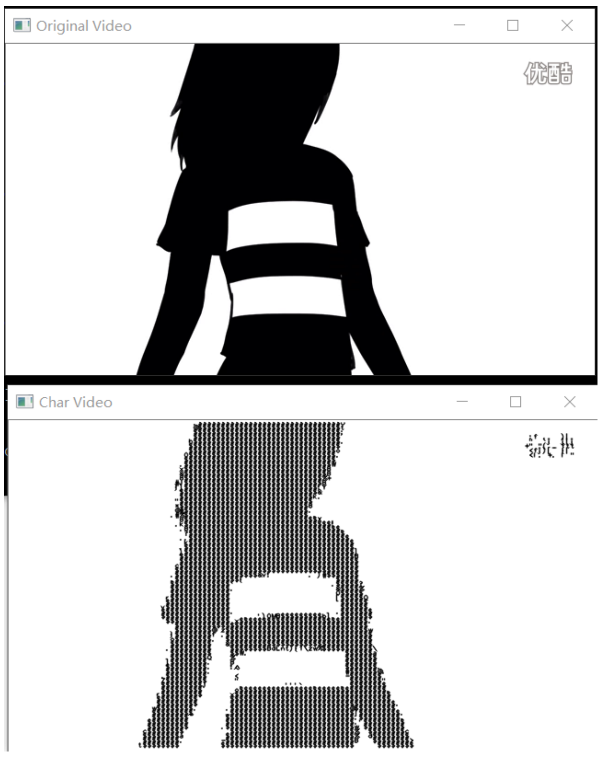
```python
def video2mp3(file):
    out_file = file.split('.')[0] + '.mp3'
    subprocess.call('ffmpeg -i ' + file + ' -f mp3 ' + out_file, shell=True)

def video_add_mp3(file, mp3_file):
    out_file = file.split('.')[0] + '.mp4'
    subprocess.call('ffmpeg -i ' + file + ' -i ' + mp3_file + ' -strict -2 -f mp4 ' + out_file, shell=True)

video2mp3(file)
video_add_mp3(name + '.avi', file.split('.')[0] + '.mp3')
```

3.3 Problems we met and solved

In the above process, we let the computer produce characters line by line, this method applies well in image transferring, but when it comes to video transferring, we met a problem, the image of the video has a good conversion effect near the left side, but there is a distortion of the edge on the right side. Just look at the following one frame of the video (the above is the original fragment, the below is the transferred one):

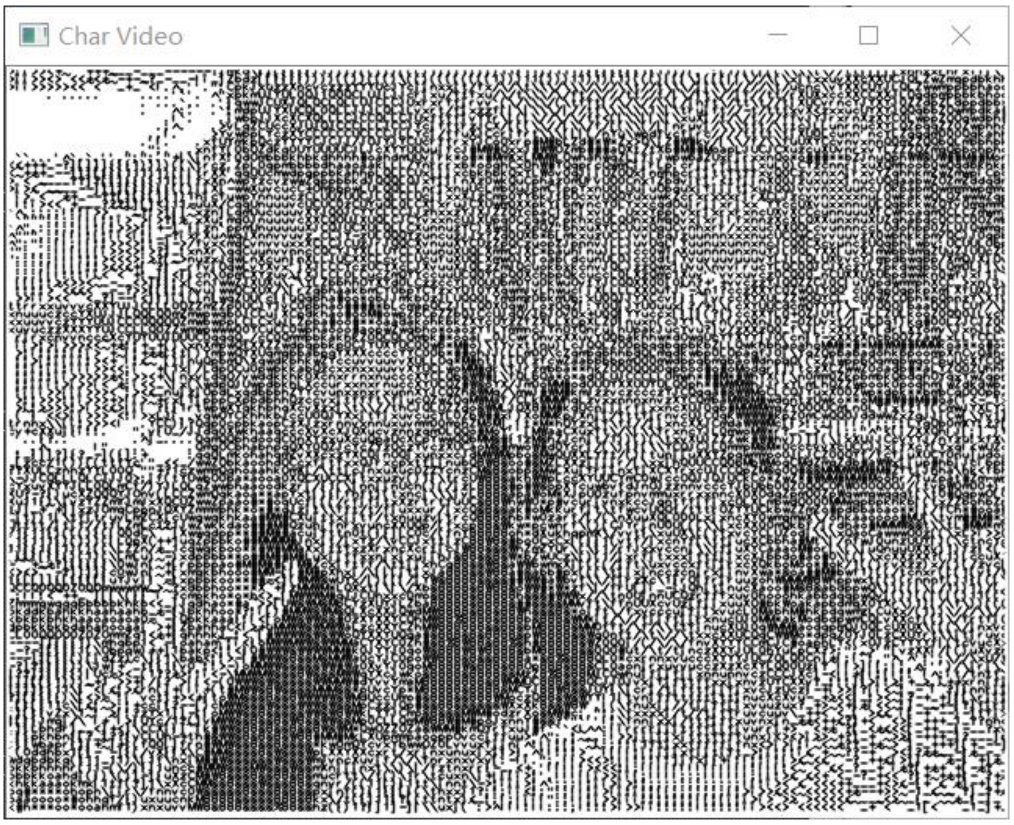


Finally, we find that it is because different characters have different width. At the beginning, we try to find another font in which every character has the same width, but we failed. So we change the computer writing method to make it write characters one by one. Surprisingly, the problem is settled, and the speed of the transformation seems not be affected.



3.4 Brainstorm to add functions and refine code

Since we can convert simple animations very well, how about people in real life like us? So we added new lines of codes to ask the computer to open its camera and make real-time conversion of captured images into ASCII paintings. Although it is not clear, considering one character has to represent a large range of gray values, the effect is satisfying.



And the last thing we do is to combine all the above functions in one project. We use the command line of anaconda to do this.

```
# initialize command parameters
parser = argparse.ArgumentParser()
parser.add_argument('file')
parser.add_argument('-r', '--ratio', type=int, nargs='?', default=10)
parser.add_argument('-s', '--save', type=bool, nargs='?', default=False, const=True)
parser.add_argument('-n', '--name', default='character')

# get parameters
args = parser.parse_args()
file = args.file
ratio = args.ratio
save = args.save
name = args.name
```

4 Problems and conclusion

4.1 Problems we want to solve

As what we show above, all the conversions we made are just black and white. Although it can be viewed as a character of ASCII art animation, but we still want to see the effect with original colors.

Because of the plentiful colors in real world, we failed to complete this step timely.

Also, we find it a little bit slowly to convert our video, usually 4~5 minutes for a video of 1~2 minutes long. It seems inefficient as most videos are not those fragments but much longer and complete ones. Converting those long and colorful ones into ASCII art animation will be impossible.

And when we use the camera to make real-time conversion, we cannot save it by using the function we made (-s). Also, when writing the code, we had to define two different functions to succeed in running the code. We wonder why did this happen and if we have any way to solve them as well.

5.2 Conclusion

During the discussion, we found a topic that all of us four were interested in ---- ASCII art animation. After learning by ourselves and trying lots of times, we have got a satisfying result. This was a great experience for us to look into the mysterious Python world, and we have improved our ability and interest in Python as well.