P1:

P1.1.

suppose $\vec{z} = (z_1, z_2, z_3, \cdots z_k) \in \mathbb{R}$

then, $\vec{z} - c_1 = (z_1 - c_1, z_2 - c_1, z_3 - c_1, \cdots z_k - c_1) \in \mathbb{R}^k$.

$$\text{softmax}(z - c_1)_i = \frac{\exp(z_i - c_1)}{\sum_{j=1}^{k} \exp(z_j - c_1)}$$

$$= \frac{\exp(z_i) \,/\, \exp(c_1)}{\sum_{j=1}^{k} [\exp(z_j) \,/\, \exp(c_1)]}$$

$$= \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} = \text{softmax}(z)_i$$

$\Rightarrow$ Proved. softmax(z) is shift invariance.

P1.2 rules

$$\frac{\partial e^{z_i}}{\partial e^{z_j}} = 0$$

$$\frac{\partial e^{z_i}}{\partial z_i} = e^{z_i}$$

$$h(x) = \frac{f(x)}{g(x)} \quad \text{then} \quad \frac{\partial h(x)}{\partial x} = \frac{f'(x)g(x) - g'(x)f(x)}{g(x)^2}$$

when $i = j$:

$$\frac{\partial y_i}{\partial z_j} = \frac{\exp(z_i)\sum_{j=1}^{K}\exp(z_j) - \exp(z_j)\exp(z_i)}{\left(\sum_{j=1}^{K}\exp(z_j)\right)^2}$$

$$= \frac{\exp(z_i)\left(\sum_{j=1}^{K}\exp(z_j) - \exp(z_j)\right)}{\left(\sum_{j=1}^{K}\exp(z_j)\right)^2}$$

$$= y_i - y_i y_j$$

when $i \neq j$:

$$\frac{\partial y_i}{\partial z_j} = \frac{0 - \exp(z_j)\exp(z_i)}{\left(\sum_{j=1}^{K}\exp(z_j)\right)^2}$$

$$= -y_i y_j$$

$$\Rightarrow \frac{\partial y_i}{\partial z_j} = \begin{cases} y_i - y_i y_j & - \quad i = j \\ -y_i y_j & i \neq j \end{cases}$$

Or:

$$y_i(\delta_{ij} - y_j)$$

where $\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$

P1.3 ①

$$\frac{\partial y_i}{\partial X} = \left[\frac{\partial y_i}{\partial X_1}, \frac{\partial y_i}{\partial X_2}, \cdots \frac{\partial y_i}{\partial X_j}\right]$$

For $$\frac{\partial y_i}{\partial x_j} = \sum_{x=1}^{K} \frac{\partial y_i}{\partial z_x} \cdot \frac{\partial z_x}{\partial x_j}$$

For $$\frac{\partial z_x}{\partial x_j} = \frac{\partial(\vec{w}_x \vec{x} + u_x)}{\partial x_j}$$

$$= \frac{\partial \vec{w}_x \vec{x}}{\partial x_j}$$

$$= \sum_{m=1}^{d} \left(\frac{\partial(\vec{w}_{mx} \cdot x_m)}{\partial x_j}\right)$$

$$= \Sigma_{m=1}^{d} w_{mx} \cdot \frac{\partial x_m}{\partial x_j}$$

$$= \sum_{m=1}^{d} w_{mx} \delta_{mj}$$

$$= w_{jx} \quad \text{(only keep } m=j\text{)}$$

$$\Rightarrow \frac{\partial y_i}{\partial x_j} = (y_i(\delta_{ij} - y_j)) \sum_{x=1}^{d} w_{jx}$$

$$\Rightarrow \frac{\partial y_i}{\partial x_j} = \sum_{x=1}^{d} \left(y_i (\delta_{ix} - y_x) \right) w_{jx}$$

$$= \sum_{x=1}^{d} \left(y_i \delta_{ix} w_{jx} - y_i y_x w_{jx} \right)$$

$$\Rightarrow \left[\frac{\partial y_i}{\partial \vec{x}} \right]_j = \boxed{y_i w_{ji} - \sum_{x=1}^{d} y_i y_x w_{jx}}$$

(2)

$$\frac{\partial y_i}{\partial \vec{w}_j} = \left[\frac{\partial y_i}{\partial w_{ij}}, \frac{\partial y_i}{\partial w_{2j}}, \dots, \frac{\partial y_i}{\partial w_{mj}} \right]$$

For $\frac{\partial y_i}{\partial w_{mj}} = \sum_{x=1}^{k} \frac{\partial y_i}{\partial z_x} \cdot \frac{\partial z_x}{\partial w_{mj}}$

For $\frac{\partial z_x}{\partial w_{mj}} = \frac{\partial (\vec{w}_x \vec{x} + u_x)}{\partial w_{mj}}$

$$= \frac{\partial \vec{w}_x \vec{x}}{\partial w_{mj}}$$

$$= \sum_{n=1}^{d} \frac{\partial (w_{nx} \cdot x_n)}{\partial w_{mj}}$$

$$= \sum_{n=1}^{d} \delta_{mn} \delta_{xj} x_n$$

$$= \delta_{xj} x_n \qquad \text{(only keep when } m=n\text{)}$$

$$= \delta_{xj} x_m$$

$$\Rightarrow \frac{\partial y_i}{\partial w_{mj}} = \sum_{x=1}^{K} \left(y_i (\delta_{ix} - y_x) \delta_{xj} x_m \right)$$

$$= \sum_{x=1}^{K} y_i \delta_{ix} \delta_{xj} x_m - \sum_{x=1}^{K} y_i y_x \delta_{xj} x_m$$

$$\left[\frac{\partial y_i}{\partial w_j} \right]_m = y_i (1 - y_i) x_m - y_i y_j x_m$$

(3)

$$\left[\frac{\partial y_i}{\partial u} \right]_j = \sum_{x=1}^{d} \frac{\partial y_i}{\partial z_x} \frac{\partial z_x}{\partial u} = \sum_{x=1}^{d} \frac{\partial y_i}{\partial z_x} \cdot 1$$

$$= y_i (\delta_{ij} - y_j)$$

P2

P2.1.

$$\begin{bmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\begin{bmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$$

The matrix multiplication. let's say $V = \begin{bmatrix} V_{11} & V_{12} & V_{1n} \\ V_{21} & & \\ & & \\ V_{m1} & & V_{mn} \end{bmatrix}$

$X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$, we make $X = \begin{bmatrix} V_{11}X_1 + V_{12}X_2 + \cdots V_{1n}X_n \\ \cdots\cdots\cdots \\ V_{m1}X_1 + V_{m2}X_2 + \cdots V_{mn}X_n \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{k=J} X_i V_{ik} \\ \vdots \\ \sum_{k=1}^{k=J} X_i V_{ik} \end{bmatrix}$

In this case. $Vx = \begin{bmatrix} -\frac{1}{\sqrt{2}}X_1 + -\frac{1}{\sqrt{2}}X_2 \\ \frac{1}{\sqrt{2}}X_1 - \frac{1}{\sqrt{2}}X_2 \end{bmatrix}$

Or, in terms of geometric, V rotate s 135° anti-clockwise

P2.2.

if the square matrix with real number is an orthogonal matrix, the transpose of the matrix is the same as its inverse matrix.

Also, in this case. $V^T = \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$ $= \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$ $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$V^T \cdot V = I = V^{-1} \cdot V \Rightarrow V^T = V^{-1}$.

$V^T X = \begin{bmatrix} -\frac{1}{\sqrt{2}} x_1 + \frac{1}{\sqrt{2}} x_2 \\ -\frac{1}{\sqrt{2}} x_1 - \frac{1}{\sqrt{2}} x_2 \end{bmatrix}$

In terms of geometric, $V^T X$ rotate X 225°
anti-clockwise

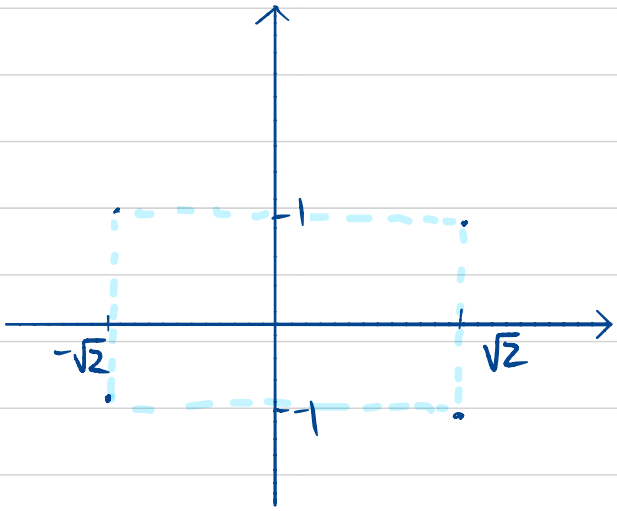
P2.3

$$\Sigma V^T \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \begin{bmatrix} -\sqrt{2} \\ -1 \end{bmatrix}$$

$$\Sigma V^T \begin{bmatrix} 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \sqrt{2} \\ -1 \end{bmatrix}$$

$$\Sigma V^T \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \begin{bmatrix} \sqrt{2} \\ 1 \end{bmatrix}$$

$$\Sigma V^T \begin{bmatrix} 0 \\ -\frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} -\sqrt{2} \\ 1 \end{bmatrix}$$

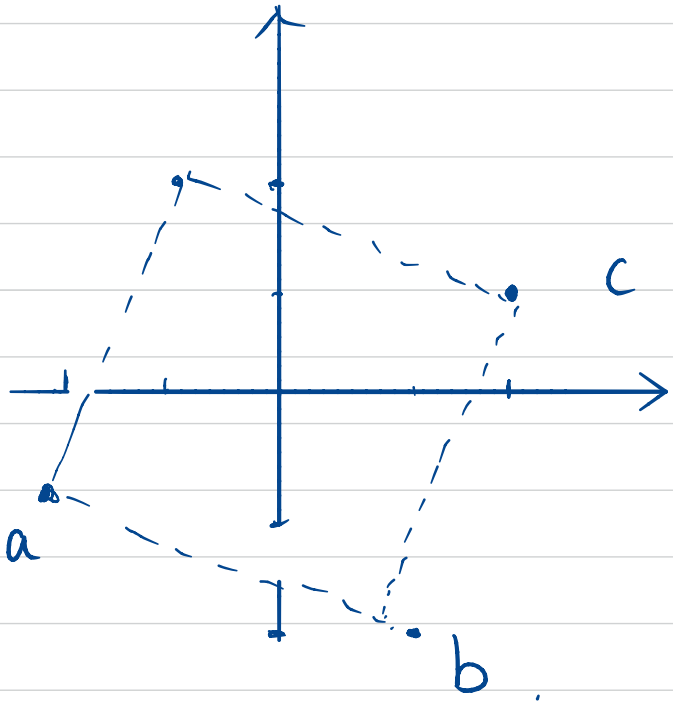


⇒ the shape of result point is a rectangle, $\begin{cases} \text{width} = 2\sqrt{2} \\ \text{height} = 2 \end{cases}$.

For instance, I have a square with corner at $(1,0), (0,1), (-1,0), (0,-1)$.

After Ux:

$$\begin{cases} (1,0) \Rightarrow \left(-\frac{\sqrt{3}}{2}, -\frac{1}{2}\right) & a \\ (0,1) \Rightarrow \left(\frac{1}{2}, -\frac{\sqrt{3}}{2}\right) & b \\ (-1,0) \Rightarrow \left(\frac{\sqrt{3}}{2}, \frac{1}{2}\right) & c \\ (0,-1) \Rightarrow \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right) & d \end{cases}$$



$\Rightarrow$ U is rotate the original vector 150° clockwise.

or 210° anti-clockwise

P.2.5

Let $B = U\Sigma V^T$,

where U and V are real and orthogonal.
The Σ is $n \times n$ and most of its entries are zero.
$\Rightarrow$ The Signular Value Decomposition (SVD).

Then V^T will rotate x, Σ rescale it and U rotate it again.

How to calculate SVD:

① Firstly, calculate eigenvalue and eigenvector for for B^TB and BB^T

② The eigenvector of BB^T will make up the column of U.
③ The eigenvector of B^TB will make up the column of V
④ Σ is the square root of eigenvalue from BB^T and B^TB., ordered decreasing as the diagonal matrix

Part3

For my code, all the code are in CS543 MP0.ipynb

The output of 3.1 is in folder: combined_output

The output for 3.2 is in folder: simple_ordered_combined_output

The output for 3.3 is in folder: hard_ordered_combined_output

3.1 Combine [5 pts].

Version 1: we can combine one pictures:

Here I take one example simple_almastatue to test the code:

- I used the PIL library

```
import numpy as np
import PIL
from PIL import Image
import glob

def combineThePicturs(inputPath, outputPath):
    #here I put all the pictures in the folder into a list
    listOfPicFiles = glob.glob(inputPath) # inputPath is like"./shredded-
images/simple_almastatue/*.png"
    #read them as pictures
    imgs = [ PIL.Image.open(i) for i in listOfPicFiles ]
    #find the min shape
    min_shape = sorted( [(np.sum(i.size), i.size ) for i in imgs])[0][1]
    imgs_comb = np.hstack( (np.asarray( i.resize(min_shape) ) for i in imgs ) )
    # save that result picture
    imgs_comb = PIL.Image.fromarray(imgs_comb)
    imgs_comb.save( outputPath ) #  outputPath is like
'./combined_output/com_simple_almastatue.png'

#run one example
combineThePicturs("./shredded-
images/simple_almastatue/*.png",'./combined_output/com_simple_almastatue.png')
```

Then we can have a picture like this:

To improve our program, we hope to have the inputPath and outputPath in the lists, so we can deal with the pictures in a loop.

```
#firstly, we put all the folder names in a list:
name_list =
['hard_almastatue','hard_building','hard_text','hard_texture','simple_almastatue',
'simple_larry-roberts']


#then, we canput the input path list:
inputPathList = []
for name in name_list:
    inputPathList.append("./shredded-images/" + name +"/*.png")

outputPathList = []
for name in name_list:
    outputPathList.append("./combined_output/com_" + name +".png")

#start the loop
#final run of part1:
```

```
for i in range(len(name_list)):
    combineThePicturs(inputPathList[i],outputPathList[i])
```

Then, we can get all the pictures simply concatenate:

3.2 Re-order [10 pts]

In the first version, we still tried our code simple_almastatue:

Here we reordered the strips:

Firstly, we store the all the first columns and last colums of one picture in the list:

```
#construct the two list that store the pix of the first colums and last colums
listOfPicFiles = glob.glob("./shredded-images/simple_almastatue/*.png")
listOfPicFiles.sort()
imgs = [ PIL.Image.open(i) for i in listOfPicFiles ]

firstColumn = []
lastColumn = []
for im in imgs:
    pix = im.load()
    first = []
    last = []
    w = im.size[0]
    h = im.size[1]
    for i in range(h):
        first.append(pix[0, i])
        last.append(pix[w-1, i])
    firstColumn.append(first)
    lastColumn.append(last)
```

Then we have the functions that can help us to find the differences between the last column of one pic and first column of another picture:

- the minimum of the differences between two pixel columns means they are matched.

```
def pixelDiff(pix1, pix2):
    """
    To compute the differences between two pixels
    """
    ans = 0
    for i in range(3):
        ans += abs(pix1[i] - pix2[i])**2
    return ans
```

```
def pixelListDiff(list1, list2):
    """
    To compute the sum of the differences between two list of pixels,
    hopefully it is the same size.
    Like: pixelListDiff(lastColumn[0], firstColumn[1])
    """
    ans = 0;
    for i in range(len(list1)):
        ans += pixelDiff(list1[i], list2[i])
    return ans
```

Start re-order:

```
#Now we need to check the lastColum of of strip and the firstColum of another
strip and computer their diff.

n = len(firstColumn)
diff_array = np.zeros([n, n])
heap = []
for i in range(n):
    for j in range(n):
        diff = pixelListDiff(lastColumn[i],firstColumn[j])
        diff_array[i][j] = diff
        if i != j:
            heap.append((diff, i, j))

heap.sort(key=lambda tup: tup[0])


order = np.zeros([n]) #order[i] = j means the next of picture i is picture j
usedJ = np.zeros([n]) #to check if the picture have already decided the pic before
it. If yes, usedJ[i] = 1. If not, If yes, usedJ[i] = 0

for i in range(n):
    order[i] = -1

#calculate the order by find the min-diff and take it out.
count = 0
for tup in heap:
    i = tup[1]
    j = tup[2]
    if i < n and order[i] == -1 and usedJ[j] == 0:
        order[tup[1]] = tup[2]
        usedJ[tup[2]] = 1
```

```
        count += 1
        if count == n-1:
            break

order = order.astype(np.int)


# now make the new orders:
# first the left most picture:
first = -1
for i in range(n):
    if usedJ[i] == 0:
        first = i
        break

order_listOfPicFiles = []
for i in range(n):
    order_listOfPicFiles.append(listOfPicFiles[first])
    first = order[first]
```

Lastly, we combine the pictures:

```
order_imgs = [ PIL.Image.open(i) for i in order_listOfPicFiles ]
min_shape = sorted( [(np.sum(i.size), i.size ) for i in order_imgs])[0][1]
imgs_comb = np.hstack( (np.asarray( i.resize(min_shape) ) for i in order_imgs ) )
# save that result picture
imgs_comb = PIL.Image.fromarray(imgs_comb)
imgs_comb.save( './ordered_simple_almastatue.png'
)#'./ordered_simple_almastatue.png'
```

To we reorganized the code into functions and run with the two simple examples:

```


#part2

def getFirstLastColumn(imgs):
    """
    construct the two list that store the pix of the first colums and last colums
    """
    firstColumn = []
    lastColumn = []
    for im in imgs:
```

```
        pix = im.load()
        first = []
        last = []
        w = im.size[0]
        h = im.size[1]
        for i in range(h):
            first.append(pix[0, i])
            last.append(pix[w-1, i])
        firstColumn.append(first)
        lastColumn.append(last)
    return firstColumn, lastColumn

def pixelDiff(pix1, pix2):
    """
    To compute the differences between two pixels
    """
    ans = 0
    for i in range(3):
        ans += abs(pix1[i] - pix2[i])**2
    return ans

def pixelListDiff(list1, list2):
    """
    To compute the sum of the differences between two list of pixels,
    hopefully it is the same size.
    Like: pixelListDiff(lastColumn[0], firstColumn[1])
    """
    ans = 0;
    for i in range(len(list1)):
        ans += pixelDiff(list1[i], list2[i])
    return ans



def getOrderedListOfPicFiles(inputPath):
    """
    This fucntion will return the ordered order_listOfPicFiles

    We need to check the lastColum of of strip and the firstColum of another strip
and computer their diff.

    """
    listOfPicFiles = glob.glob(inputPath)
    listOfPicFiles.sort()
    imgs = [ PIL.Image.open(i) for i in listOfPicFiles ]
```

```
    firstColumn, lastColumn = getFirstLastColumn(imgs)
    n = len(firstColumn)
    diff_array = np.zeros([n, n])
    heap = []
    for i in range(n):
        for j in range(n):
            diff = pixelListDiff(lastColumn[i],firstColumn[j])
            diff_array[i][j] = diff
            if i != j:
                heap.append((diff, i, j))

    heap.sort(key=lambda tup: tup[0])

    order = np.zeros([n]) #order[i] = j means the next of picture i is picture j
    usedJ = np.zeros([n]) #to check if the picture have already decided the pic
before it. If yes, usedJ[i] = 1. If not, If yes, usedJ[i] = 0

    for i in range(n):
        order[i] = -1

    count = 0
    for tup in heap:
        i = tup[1]
        j = tup[2]
        if i < n and order[i] == -1 and usedJ[j] == 0:
            order[tup[1]] = tup[2]
            usedJ[tup[2]] = 1
            count += 1
            if count == n-1:
                break

    order = order.astype(np.int)

    # now make the new orders:
    # first the left most picture:
    first = -1
    for i in range(n):
        if usedJ[i] == 0:
            first = i
            break

    order_listOfPicFiles = []
    for i in range(n):
        order_listOfPicFiles.append(listOfPicFiles[first])
        first = order[first]
    # print(listOfPicFiles)
```

```
    return order_listOfPicFiles

def combineSimpleOrderedPictures(inputPath, outputPath):
    order_listOfPicFiles = getOrderedListOfPicFiles(inputPath)
    order_imgs = [ PIL.Image.open(i) for i in order_listOfPicFiles ]
    min_shape = sorted( [(np.sum(i.size), i.size ) for i in order_imgs])[0][1]
    imgs_comb = np.hstack( (np.asarray( i.resize(min_shape) ) for i in order_imgs
) )
    # save that result picture
    imgs_comb = PIL.Image.fromarray(imgs_comb)
    imgs_comb.save( outputPath) #  './ordered_simple_almastatue.png'
```

With the helper functions ready, we can finally generate our output:

```
# Then we can start our functions:

#firstly, we put all the folder names in a list:
simple_name_list = ['simple_almastatue','simple_larry-roberts']


#then, we canput the input path list:
simple_inputPathList = []
for name in simple_name_list:
    simple_inputPathList.append("./shredded-images/" + name +"/*.png")

ordered_simple_outputPathList = []
for name in name_list:

 ordered_simple_outputPathList.append("./simple_ordered_combined_output/ordered" +
name +".png")


for i in range(len(simple_name_list)):

 combineSimpleOrderedPictures(simple_inputPathList[i],ordered_simple_outputPathLis
t[i])
```

We get the output:


ALMA MATER

3.3 Align and Re-order [15 pts]

Here I will show my ways of find the similarity:

- I slide those two strips,
  - one is make the shorter one down and overlap with the lower part of the longer one
  - one is make the shorter one up and overlap with the higher part of the longer one
- trucate the shorter one.
  - make the longer vector as the same size as the shorter one
- calculate the normalized one of each
  - First, minus the mean of the vector
  - divide the std of the vector
- flatten the vectors
  - originally, it has three colors, which is hard for us to calculate
  - Then we can have the vectors dot produc equals to one number, not a matrix
- calculate the dot product
  - I used 1 - 1- vec1 @ vec2 since if two vectors are the same, they vec1 @vec2 == 1, so I caculate the distance from 1 to vec1 @ vec2

The core functions are here:

```
def normalizedVec(list):
    list = np.asarray(list, dtype = np.float64)
    mean = np.mean(list)
    list -= mean
    norm = np.linalg.norm(list)
    normal_array = list/norm

    return normal_array

def calculateDiff1(vec1, vec2):
    """
    overlap the lower part of the longer one
    """
    if len(vec1) > len(vec2):
        vec1 = vec1[-len(vec2):]
    if len(vec1) < len(vec2):
        vec2 = vec2[-len(vec1):]
    if(len(vec1) != len(vec2)):
        print("Warning, size not the same")
    vec1 = normalizedVec(vec1)
    vec2 = normalizedVec(vec2)
```

```
    vec1 = vec1.flatten()
    vec2 = vec2.flatten()


    return 1- vec1 @ vec2

def calculateDiff2(vec1, vec2):
    """
    overlap the higher part of the longer one
    """
    if len(vec1) > len(vec2):
        vec1 = vec1[:len(vec2)]
    if len(vec1) < len(vec2):
        vec2 = vec2[:len(vec1)]
    if(len(vec1) != len(vec2)):
        print("Warning, size not the same")
    vec1 = normalizedVec(vec1)
    vec2 = normalizedVec(vec2)

    vec1 = vec1.flatten()
    vec2 = vec2.flatten()


    return 1- vec1 @ vec2




def slide(vec1, vec2):
    len1 = len(vec1)
    len2 = len(vec2)
    #make sure vec1 is smaller than vec2
    if len2 > len1:
        return slide(vec2, vec1)

    sliderange =int(len2 * 0.2)
    mindiff = calculateDiff1(vec1, vec2)
    bestslide = 0


    for s in range(sliderange):
        newvec1 = []
        newvec2 = []

        #make the left one go down and overlap the lower part of the longer one
```

```
        for i in range(len1 - s):
            newvec1.append(vec1[i])
        diff = calculateDiff1(newvec1, vec2)
        if diff < mindiff:
            mindiff = diff
            bestslide = s

        #make the left one go up and overlap the higher part of the longer one
        for i in range(s, len1):
            newvec2.append(vec1[i])
        diff = calculateDiff2(newvec2, vec2)
        if diff < mindiff:
            mindiff = diff
            bestslide = s
    return mindiff, bestslide
```

It works well.

Althought the hard_texture one is hard to tell from us, we can easily see another three work well:

IN! ;T M

:oup RO.

al ligence Gr July 7, 966

emo Intell 100, 1!

Artifici . No.

Vision M

TH E SUM ON PROJECT

Se MER R VISI pert

ymOut Pa

su mm visi on proj ect is an at tem pt to use our s umm er wor ke

The ely er the cons tru ction of a si gnificant part of a visual sy stem.

effective p ticular n th task was ch osen pa rtly b ecause it can b e s egmented y an d in

The part icular task will allow individual ste s to work independently to

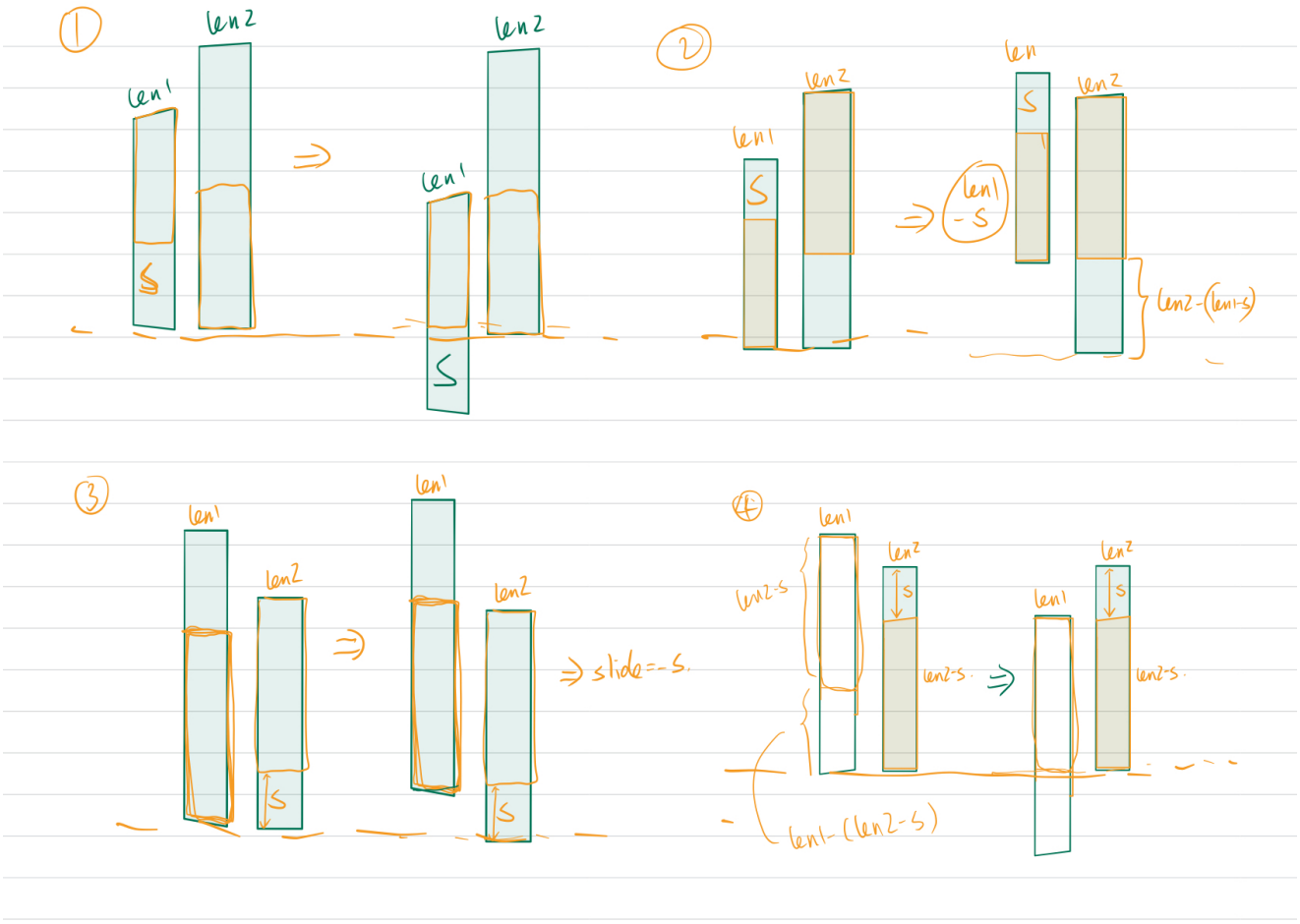
e probl em s which ch w ill al so be t

su b-probl em s of the const ruction o f a sy stem complex enough to ye

pa rt i he cons .. n cc en h a r

So here I am trying to find the ways to improve the current algorithm and and stitch the shreds together have the slide canculated.

To find the right slide, here I change the slide and calculateDiff1, calculateDiff2 functions, and divide it into four the situations :



To help us to calculate, we suppose that the bottom of the two adjacant strips are vertical, and we always calculate the relatively change of the left bottom one when the right bottom is fixed.

Then we can divide it into four situation,

- The left one is shorter and move down
- The left one is shorter and move up
- The left one is longer and move down
- The left one is longer and move up

Then, we improve our functions into the followings:

```
def normalizedVec(list):
    list = np.asarray(list, dtype = np.float64)
    mean = np.mean(list)
    list -= mean
    norm = np.linalg.norm(list)
    normal_array = list/norm

    return normal_array
```

```
def calculateDiff1(vec1, vec2):
    """
    overlap the lower part of the longer one
    """
    if len(vec1) > len(vec2):
        vec1 = vec1[-len(vec2):]
    if len(vec1) < len(vec2):
        vec2 = vec2[-len(vec1):]
    vec1 = normalizedVec(vec1)
    vec2 = normalizedVec(vec2)
    vec1 = vec1.flatten()
    vec2 = vec2.flatten()


    return 1- vec1 @ vec2

def calculateDiff2(vec1, vec2):
    """
    overlap the higher part of the longer one
    """

    if len(vec1) > len(vec2):
        vec1 = vec1[:len(vec2)]
    if len(vec1) < len(vec2):
        vec2 = vec2[:len(vec1)]

    vec1 = normalizedVec(vec1)
    vec2 = normalizedVec(vec2)
    vec1 = vec1.flatten()
    vec2 = vec2.flatten()

    return 1- vec1 @ vec2


def slide(vec1, vec2):
    """
    we suppose the initial state it is bottom of the strips are at the same level
    s is positive means left one goes up
    s is negative means left one goes down
    """
    len1 = len(vec1)
    len2 = len(vec2)

    mindiff = calculateDiff1(vec1, vec2)
    bestslide = 0
```

```
    if len1 >= len2:
        sliderange =int(len2 * 0.2)
        for s in range(sliderange):
            newvec1 = []
            newvec2 = []

            #situation1
            #make the left one go down and overlap the lower part of the longer
one
            for i in range(len1 - s):#upper part the left one
                newvec1.append(vec1[i])

            diff = calculateDiff1(newvec1, vec2)
            if diff < mindiff:
                mindiff = diff
                bestslide = -s

            #situation2
            #make the left one go up and overlap the higher part of the longer one

            for i in range(s, len1):#lower part of left one
                newvec2.append(vec1[i])

            diff = calculateDiff2(newvec2, vec2)
            if diff < mindiff:
                mindiff = diff
                bestslide = (len2 - (len1-s))



    if len1 < len2:
        sliderange =int(len1 * 0.2)
        for s in range(sliderange):
            newvec1 = []
            newvec2 = []

            #situation3
            #make the left one go up
            for i in range(len2 - s): #upper part of right one, shorter one
                newvec1.append(vec2[i])
            diff = calculateDiff1(newvec1, vec1)
            if diff < mindiff:
                mindiff = diff
                bestslide = s
```

```
            #situation4
            #make the left one go down
            for i in range(s, len2): #lower part of right one, shorter one
                newvec2.append(vec2[i])
            diff = calculateDiff2(newvec2, vec1)
            if diff < mindiff:
                mindiff = diff
                bestslide = (len1 - (len2-s)) * -1


    return mindiff, bestslide
```

Also, to avoid the strips will appear again and help our greedy function, I includeded the union find algorithm to avoid loop for the pictures.

```
def union(parent, a, b):
    roota = find(parent,a)
    rootb = find(parent,b)
    if roota == rootb:
        return False
    if roota < rootb:
        parent[roota] = rootb
        return True
    if rootb < roota:
        parent[rootb] = roota
        return True

def find(parent, a):
    if parent[a] != a:
        parent[a] = find(parent,parent[a])
    return parent[a]
```

Also, for the last part, we add the padding to the top and bottom of each strip, so that the total height of each picture are the same. So, we can stitch the shreds together.

```
def add_padding(pil_img, top, right, bottom, left, color):
    """
    black(0,0,0)
    """
    width, height = pil_img.size
    new_width = width + right + left
    new_height = height + top + bottom
    result = Image.new(pil_img.mode, (new_width, new_height), color)
    result.paste(pil_img, (left, top))
    return result
```

With the helper functions ready, we can start process the pictures :

```
def combineHardOrderedPictures(inputPath,outputPath):
    # inputPath = "./shredded-images/hard_building/*.png"
    listOfPicFiles = glob.glob(inputPath)
    listOfPicFiles.sort()
    imgs = [ PIL.Image.open(i) for i in listOfPicFiles ]

    #to compute the first column, last coloum and the height of each picture in
orders
    firstColumn = []
    lastColumn = []
    picHeight = []
    for im in imgs:
        pix = im.load()
        first = []
        last = []
        w = im.size[0]
        h = im.size[1]
        picHeight.append(h)
        for i in range(h):
            first.append(pix[0, i])
            last.append(pix[w-1, i])
        firstColumn.append(first)
        lastColumn.append(last)

    # the total number of pictures
    n = len(firstColumn)

    # store the information in the heap, which here we just list and sort it.
    diff_array = np.zeros([n, n])
    heap = []
```

```
    for i in range(n):
        for j in range(n):
            if i != j:
                diff, s = slide(lastColumn[i],firstColumn[j])
                diff_array[i][j] = diff
                heap.append((diff, i, j, s))

    heap.sort(key=lambda tup: tup[0])


    order = np.zeros([n]) #order[i] = j means the next of picture i is picture j
    usedJ = np.zeros([n]) #to check if the picture have already decided the pic
before it. If yes, usedJ[i] = 1. If not, If yes, usedJ[i] = 0
    usedI = np.zeros([n]) #to check if the picture have already decided the pic
after it.
    parent = np.zeros([n]) #used for union find
    slide_amount = np.zeros([n]) #to store the sliding amount of this strip

    for i in range(n):
        order[i] = -1
        parent[i] = i

    parent = parent.astype(np.int)

    count = 0
    for tup in heap:
        i = tup[1]
        j = tup[2]
        if usedJ[j] == 0 and usedI[i] == 0:
            if union(parent, i, j):
                order[i] = j
                slide_amount[i] = tup[3]
                usedJ[j] = 1
                usedI[i] = 1
                count += 1
                if count == n-1:
                    break


    order = order.astype(np.int)



    # now make the new orders:
    # first the left most picture:
    first = -1
```

```
    for i in range(n):
        if usedJ[i] == 0:
            first = i
            break

    order_listOfPicFiles = []
    order_slideamount = []
    pic_order = []

    for i in range(n):
        order_listOfPicFiles.append(listOfPicFiles[first])
        pic_order.append(first)
        order_slideamount.append(slide_amount[first])
        first = order[first]


    bottom_index_list = np.zeros([n])
    top_index_list = np.zeros([n])
    cur = 0
    for i in range(1,len(order_slideamount)):
        bottom_index_list[i] = bottom_index_list[i-1] - order_slideamount[i-1]
    bottom_index_list = bottom_index_list.astype(np.int)


    toppest = np.max(top_index_list)

    bottomest = np.min(bottom_index_list)

    top_padding = np.zeros([n])
    bottom_padding = np.zeros([n])



    for i in range(len(bottom_index_list)):
        top_padding[i] = toppest - top_index_list[i]
        bottom_padding[i] = bottom_index_list[i] - bottomest

    top_padding = top_padding.astype(np.int)
    bottom_padding = bottom_padding.astype(np.int)



    order_imgs = []


    for i in range(len(order_listOfPicFiles)):
```

```
        newimg =
add_padding(PIL.Image.open(order_listOfPicFiles[i]),top_padding[i],0,bottom_paddin
g[i], 0, (0,0,0))
        order_imgs.append(newimg)

    min_shape = sorted( [(np.sum(i.size), i.size ) for i in order_imgs])[0][1]
    imgs_comb = np.hstack( (np.asarray( i.resize(min_shape) ) for i in order_imgs
) )
    # save that result picture
    imgs_comb = PIL.Image.fromarray(imgs_comb)
    display(imgs_comb)
    imgs_comb.save(outputPath)
```

Then we can use a loop to process for the hard pictures:

```
# Then we can start our functions:

#firstly, we put all the folder names in a list:
hard_name_list = ['hard_almastatue','hard_building','hard_text','hard_texture']

#then, we canput the input path list:
hard_inputPathList = []
for name in hard_name_list:
    hard_inputPathList.append("./shredded-images/" + name +"/*.png")

ordered_hard_outputPathList = []
for name in hard_name_list:
    ordered_hard_outputPathList.append("./hard_ordered_combined_output/ordered" +
name +".png")


for i in range(len(hard_name_list)):

 combineHardOrderedPictures(hard_inputPathList[i],ordered_hard_outputPathList[i])
```

Then we have our final results:

[redacted]

Artificial Intelligence Group July 7, 1966
Vision Memo. No. 100.

THE SUMMER VISION PROJECT

Seymour Papert

The summer vision project is an attempt to use our summer workers effectively in the construction of a significant part of a visual system. The particular task was chosen partly because it can be segmented into sub-problems which will allow individuals to work independently and yet pa[redacted]ate i[redacted]he constru[redacted]on o[redacted]system comp[redacted] enough to be a r[redacted]l