1. $\vec{z}=(z_1,\cdots,z_k)\in\mathbb{R}^k$, $\vec{y}=\text{softmax}(\vec{z})\in\mathbb{R}^k$, where $y_i=\frac{\exp(z_i)}{\sum_{j=1}^{k}\exp(z_j)}$

(1.1.) $\text{softmax}(\vec{z})=\text{softmax}(\vec{z}-C\vec{1})$, where $C\in\mathbb{R}$ and $\vec{1}=(\overbrace{1,1,\cdots,1}^{k}).\in\mathbb{R}^k$

pf) ($\Rightarrow$) $\forall i\in[1,k]$, $\text{softmax}(\vec{z})_i=\frac{\exp(z_i)}{\sum_{j=1}^{k}\exp(z_j)}$

$$=\frac{\exp(z_i)}{\exp(z_1)+\cdots+\exp(z_k)}\times\frac{\exp(-C)}{\exp(-C)}$$

$$=\frac{\exp(z_i-C)}{\exp(z_1-C)+\cdots+\exp(z_k-C)}$$

$$=\text{softmax}(\vec{z}-C\vec{1})_i.$$

($\Leftarrow$) $\forall i\in[1,k]$, $\text{softmax}(\vec{z}-C\vec{1})_i=\frac{\exp(z_i-C)}{\sum_{j=1}^{k}\exp(z_j-C)}$

$$=\frac{\exp(z_i-C)}{\exp(z_1-C)+\cdots+\exp(z_k-C)}\times\frac{\exp(C)}{\exp(C)}$$

$$=\frac{\exp(z_i)}{\exp(z_1)+\cdots+\exp(z_k)}$$

$$=\text{softmax}(\vec{z})_i.$$

(1.2.) $y_i=\text{softmax}(\vec{z})_i=\frac{\exp(z_i)}{\sum_{n=1}^{k}\exp(z_n)}=\frac{\exp(z_i)}{\exp(z_1)+\cdots+\exp(z_j)+\cdots+\exp(z_k)}:=\frac{f(\vec{z})}{g(\vec{z})}$.

① $i\neq j$

$$\frac{\partial f(\vec{z})}{\partial z_j}=\frac{\partial}{\partial z_j}(\exp(z_i))=0,\quad \frac{\partial g(\vec{z})}{\partial z_j}=\frac{\partial}{\partial z_j}(\exp(z_1)+\cdots+\exp(z_j)+\cdots+\exp(z_k))=\exp(z_j)$$

$$\frac{\partial y_i}{\partial z_j}=\frac{\partial}{\partial z_j}\left(\frac{f(\vec{z})}{g(\vec{z})}\right)=\frac{\frac{\partial f}{\partial z_j}\cdot g-f\cdot\frac{\partial g}{\partial z_j}}{g^2}=\frac{0\cdot\sum_{n=1}^{k}\exp(z_n)-\exp(z_i)\cdot\exp(z_j)}{\{\sum_{n=1}^{k}\exp(z_n)\}\cdot\{\sum_{n=1}^{k}\exp(z_n)\}}=-\frac{\exp(z_i)}{\sum_{n=1}^{k}\exp(z_n)}\cdot\frac{\exp(z_j)}{\sum_{n=1}^{k}\exp(z_n)}$$

$$=-\text{softmax}(\vec{z})_i\cdot\text{softmax}(\vec{z})_j.$$

② $i=j$

$$\frac{\partial f(\vec{z})}{\partial z_j}=\frac{\partial f}{\partial z_i}=\frac{\partial}{\partial z_i}(\exp(z_i))=\exp(z_i),\quad \frac{\partial g(\vec{z})}{\partial z_j}=\frac{\partial g}{\partial z_i}=\frac{\partial}{\partial z_i}(\exp(z_1)+\cdots+\exp(z_i)+\cdots+\exp(z_k))=\exp(z_i)$$

$$\frac{\partial y_i}{\partial z_j}=\frac{\frac{\partial f}{\partial z_j}\cdot g-f\cdot\frac{\partial g}{\partial z_j}}{g^2}=\frac{\exp(z_i)\cdot\sum_{n=1}^{k}\exp(z_n)-\exp(z_i)\cdot\exp(z_i)}{\{\sum_{n=1}^{k}\exp(z_n)\}\cdot\{\sum_{n=1}^{k}\exp(z_n)\}}=\frac{\exp(z_i)}{\sum_{n=1}^{k}\exp(z_n)}\cdot\frac{\sum_{n=1}^{k}\exp(z_n)-\exp(z_i)}{\sum_{n=1}^{k}\exp(z_n)}$$

$$=\text{softmax}(\vec{z}_i)\cdot(1-\text{softmax}(\vec{z})_i).$$

(continued)

$$\therefore \frac{\partial y_i}{\partial z_j} = \begin{cases} -\text{softmax}(\vec{z})_i \cdot \text{softmax}(\vec{z})_j & \text{if } i \neq j \\ \text{softmax}(\vec{z})_i \cdot (1 - \text{softmax}(\vec{z})_i) & \text{if } i = j \end{cases}$$

(or, $\text{softmax}(\vec{z})_i \cdot \{\delta_{ij} - \text{softmax}(\vec{z})_j\}$ if $\delta_{ij} = 1$ if $i=j$ and 0 otherwise)

(1.3.) (note: all vectors are column vectors. all vector derivatives are row-wise.)

$$\frac{\partial}{\partial \vec{x}}(\vec{w}_i \cdot \vec{x}) = \frac{\partial}{\partial \vec{x}}(\vec{w}_i^T \vec{x}) = \vec{w}_i^T \quad \text{---------- (1)}$$

$$\frac{\partial}{\partial \vec{w}_j}(\vec{w}_i \cdot \vec{x}) = \begin{cases} \frac{\partial}{\partial \vec{w}_j}(\vec{w}_i^T \vec{x}) = \vec{0}^T & \text{if } i \neq j \\ \frac{\partial}{\partial \vec{w}_i}(\vec{w}_i^T \vec{x}) = \vec{w}_i^T & \text{if } i = j \end{cases} \Rightarrow \frac{\partial}{\partial \vec{w}_j}(\vec{w}_i \cdot \vec{x}) = \delta_{ij} \vec{x}^T \quad \text{---------- (2)}$$

① $\underset{\mathbb{R}^{1\times d}}{\frac{\partial y_i}{\partial \vec{x}}} = \sum_{j=1}^{k} \underset{\mathbb{R}}{\frac{\partial y_i}{\partial z_j}} \cdot \underset{\mathbb{R}^{1\times d}}{\frac{\partial z_j}{\partial \vec{x}}}$ ($\because$ chain rule)

$= \sum_{j=1}^{k} \frac{\partial y_i}{\partial z_j} \cdot \frac{\partial}{\partial \vec{x}}(\vec{w}_j^T \vec{x} + (\vec{u})_j)$

$= \sum_{j=1}^{k} \frac{\partial y_i}{\partial z_j} \cdot \frac{\partial}{\partial \vec{x}}(\vec{w}_j^T \cdot \vec{x})$ ($\because (\vec{u})_j$ is independent w.r.t $\vec{x}$)

$= \sum_{j=1}^{k} \frac{\partial y_i}{\partial z_j} \cdot \vec{w}_j^T$ ($\because$ by eq (1))

$= \sum_{j=1}^{k} \underbrace{\text{softmax}(\vec{z})_i \cdot \{\delta_{ij} - \text{softmax}(\vec{z})_j\}}_{\mathbb{R}} \cdot \underbrace{\vec{w}_j^T}_{\mathbb{R}^{1\times d}}$ ($\because$ by (1.2.))

② $\underset{\mathbb{R}^{1\times d}}{\frac{\partial y_i}{\partial \vec{w}_j}} = \sum_{n=1}^{k} \underset{\mathbb{R}}{\frac{\partial y_i}{\partial z_n}} \cdot \underset{\mathbb{R}^{1\times d}}{\frac{\partial z_n}{\partial \vec{w}_j}}$ ($\because$ chain rule)

$= \sum_{n=1}^{k} \frac{\partial y_i}{\partial z_n} \cdot \frac{\partial}{\partial \vec{w}_j}(\vec{w}_n^T \cdot \vec{x} + (\vec{u})_n)$

$= \sum_{n=1}^{k} \frac{\partial y_i}{\partial z_n} \cdot \frac{\partial}{\partial \vec{w}_j}(\vec{w}_n^T \vec{x})$

$= \sum_{n=1}^{k} \frac{\partial y_i}{\partial z_n} \cdot \delta_{nj} \vec{x}^T$ ($\because$ by eq. (2))

$= \frac{\partial y_i}{\partial z_j} \cdot \vec{x}^T$ ($\because$ all terms marginalized out for $n \neq j$)

$= \text{softmax}(\vec{z})_i \cdot \{\delta_{ij} - \text{softmax}(\vec{z})_j\} \cdot \vec{x}^T$ ($\because$ by (1.2.))

③ $\underset{\mathbb{R}^{1\times k}}{\frac{\partial y_i}{\partial \vec{u}}} = \sum_{n=1}^{k} \frac{\partial y_i}{\partial z_n} \cdot \underset{\mathbb{R}^{1\times k}}{\frac{\partial z_n}{\partial \vec{u}}}$

$= \sum_{n=1}^{k} \frac{\partial y_i}{\partial z_n} \cdot \frac{\partial}{\partial \vec{u}}(\vec{w}_n^T \vec{x} + (\vec{u})_n)$

$= \sum_{n=1}^{k} \frac{\partial y_i}{\partial z_n} \cdot \frac{\partial}{\partial \vec{u}}((\vec{u})_n)$

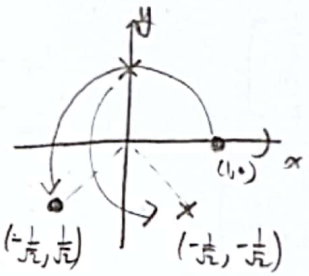
$= \sum_{n=1}^{k} \frac{\partial y_i}{\partial z_n} \cdot \vec{e}_n^T$, where $\vec{e}_n = [0, \cdots, 0, 1, 0, \cdots, 0] \in \mathbb{R}^k$

$= \sum_{n=1}^{k} \text{softmax}(\vec{z})_i \{\delta_{in} - \text{softmax}(\vec{z})_n\} \cdot \vec{e}_n^T$ where $\vec{e}_n$: unit vector in $\mathbb{R}^k$, whose n th element is one

2.

(2.1.) $V\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}-\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}-\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{bmatrix}$

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



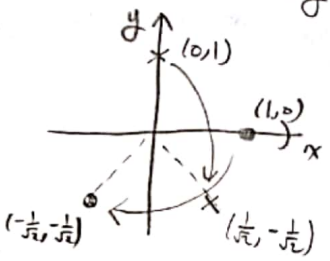
For $\forall \vec{x} \in \mathbb{R}^2$, $V\vec{x}$ does a rotation (c.c.w.) of $\vec{x}$ w.r.t. the origin by $+\frac{3}{4}\pi$.

(2.2.) $V^{-1} = \frac{1}{|V|}\begin{bmatrix}-\frac{1}{\sqrt{2}} & +\frac{1}{\sqrt{2}}\\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{bmatrix} = \frac{1}{\frac{1}{2}+\frac{1}{2}}\begin{bmatrix}-\frac{1}{\sqrt{2}} & +\frac{1}{\sqrt{2}}\\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{bmatrix} = \begin{bmatrix}-\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{bmatrix} = V^T$.

For $\forall \vec{x} \in \mathbb{R}^2$, $V^T\vec{x} = V^{-1}\vec{x}$ does a c.w. rotation of $\vec{x}$ w.r.t the origin by $+\frac{3}{4}\pi$ (= c.c.w by $-\frac{3}{4}\pi$).

e.g.) $V^T\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}-\frac{1}{\sqrt{2}}\\ -\frac{1}{\sqrt{2}}\end{bmatrix}$

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



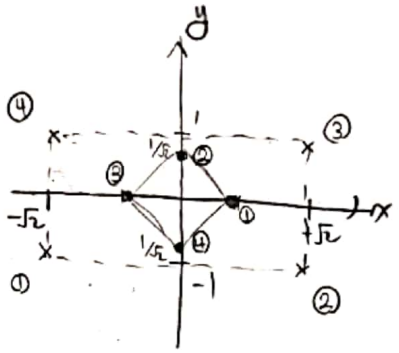
(2.3.) $\Sigma V^T = \begin{bmatrix}\sqrt{8} & 0\\ 0 & 2\end{bmatrix}\begin{bmatrix}-1/\sqrt{2} & 1/\sqrt{2}\\ -1/\sqrt{2} & -1/\sqrt{2}\end{bmatrix} = \begin{bmatrix}-2 & 2\\ -\sqrt{2} & -\sqrt{2}\end{bmatrix}$.

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

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

③ $\Sigma V^T\begin{bmatrix}-1/\sqrt{2}\\ 0\end{bmatrix} = \begin{bmatrix}-2 & 2\\ -\sqrt{2} & -\sqrt{2}\end{bmatrix}\begin{bmatrix}-1/\sqrt{2}\\ 0\end{bmatrix} = \begin{bmatrix}+\sqrt{2}\\ 1\end{bmatrix}$.

④ $\Sigma V^T\begin{bmatrix}0\\ -1/\sqrt{2}\end{bmatrix} = \begin{bmatrix}-2 & 2\\ -\sqrt{2} & -\sqrt{2}\end{bmatrix}\begin{bmatrix}0\\ -1/\sqrt{2}\end{bmatrix} = \begin{bmatrix}-\sqrt{2}\\ 1\end{bmatrix}$.



∴ These points form a rectangle.

(continued)

$\cos\theta = -\frac{\sqrt{3}}{2}$, $\sin\theta = -\frac{1}{2}$

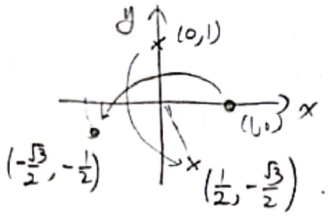
(2.4.)

$$U = \begin{bmatrix} -\frac{\sqrt{3}}{2} & \frac{1}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix}$$

As similar to (2.1.), for $\forall \vec{x} \in \mathbb{R}^2$, $U\vec{x}$ does a c.c.w. rotation of a point $\vec{x}$ w.r.t. the origin by $\frac{7}{6}\pi$.

e.g. $U\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -\frac{\sqrt{3}}{2} \\ -\frac{1}{2} \end{bmatrix}$

$U\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ -\frac{\sqrt{3}}{2} \end{bmatrix}$



(2.5.)

$$A = U\Sigma V^T = \begin{bmatrix} -\frac{\sqrt{3}}{2} & \frac{1}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} -2 & 2 \\ -\sqrt{2} & -\sqrt{2} \end{bmatrix} = \begin{bmatrix} \sqrt{3} - \frac{\sqrt{2}}{2} & -\sqrt{3} - \frac{\sqrt{2}}{2} \\ 1 + \frac{\sqrt{6}}{2} & -1 + \frac{\sqrt{6}}{2} \end{bmatrix}$$

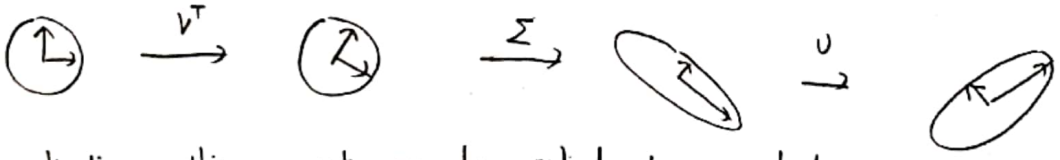
If $\forall B \in \mathbb{R}^{n\times n}$ can be decomposed by $B = U\Sigma V^T$ (where $U, \Sigma, V \in \mathbb{R}^{n\times n}$) as above, then we can interpret $B\vec{x}$ as a composition of rotation, re-scaling, and rotation of $\vec{x} \in \mathbb{R}^n$ (V^T) (Σ) (U) w.r.t. the origin in $\mathbb{R}^n$.

For example, if $n = 3$, we can imagine a sphere $S = \{\vec{x} \mid \|\vec{x}\| = R\}$.

① By V^T, S is rotated.

② By Σ, S is re-scaled along each axis.

③ By U, S is rotated.



For generalization, this concept can be applied to any higher order space. (n-dim)

MP0-Q3 by Jongwon Lee (jongwon5)

Description on parameters

```
usage: main.py [-h] [--src-dir SRC_DIR] [--dst-dir DST_DIR]
               [--img-name {simple_almastatue,simple_larry-
roberts,hard_almastatue,hard_texture,hard_text,hard_building}]
               [--mode {None,ssd,zncc}]

parameters for image processing

optional arguments:
  -h, --help            show this help message and exit
  --src-dir SRC_DIR     directory to read shredded images
  --dst-dir DST_DIR     directory to write concatenated image
  --img-name {simple_almastatue,simple_larry-
roberts,hard_almastatue,hard_texture,hard_text,hard_building}
                        image to concatenate
  --mode {None,ssd,zncc}
                        matching method to use
```

How to use

Before get started, please unzip this project into the directory `MP0/`, which should already include a folder `shredded-images/`.

1. To create the answer for (3.1.), please execute:

```
python main.py --img-name simple_{IMGNAME}
```

2. To create the answer for (3.2.), please execute:

```
python main.py --img-name simple_{IMGNAME} --mode ssd
```

3. To create the answer for (3.3.), please execute:

```
python main.py --img-name hard_{IMGNAME} --mode zncc
```

You may be able to see combined strips in the directory `results/`.

Examples

Let's reconstruct the Alma Statue.

1. Combine [5 pts]

```
$ python main.py --img-name simple_almastatue
> Namespace(dst_dir='results/', img_name='simple_almastatue', mode=None,
src_dir='../shredded-images/')
> collapsed time: 0.0000 [s]
```

You may be able to see the weird combined image in an unsorted order. This has been done by simply reading all strips in the designated directory (function `_read_strips(self, src_dir)`), sort randomly (function `sort_strips_random(self)`), and combine these (function `combine_strips(self)`).

2. Re-order [10 pts]

Now, let's sort the strips using *the sum of squared differences*.

```
$ python main.py --img-name simple_almastatue --mode ssd
> Namespace(dst_dir='results/', img_name='simple_almastatue', mode='ssd',
src_dir='../shredded-images/')
> collapsed time: 0.0247 [s]
```

It seems like the strips are sorted properly. The sorting algorithm is implemented in a member function `sort_strips_ssd` of `ImgUnshredder` class. Below is the pseudocode of `sort_strips_ssd` function.

```
input : strips_unsorted, strips_sorted
goal  : move all elements from strips_unsorted to strips_sorted with a
proper order.

do
    for all strip_being_compared in strips_unsorted:
        max_nssd <- -inf
        similarest_strip <- None
        append_direction <- None

        if nssd(rightmost(strips_sorted), leftmost(strip_being_compared)) >
max_nssd then
            max_nssd <- nssd(rightmost(strips_sorted),
leftmost(strip_being_compared))
            similarest_strip <- strip_being_compared
            append_direction <- right
        else if nssd(rightmost(strip_being_compared),
leftmost(strips_sorted)) > max_nssd then
            max_nssd <- nssd(rightmost(strip_being_compared),
leftmost(strips_sorted))
            similarest_strip <- strip_being_compared
            append_direction <- left
        end
    end

    move similarest_strip from strips_unsorted to strips_sorted, by
appending on the append_direction

until len(strips_unsorted) == 0
```

where negative SSD can be computed as below:

```
function nssd(arr1, arr2)
    return -sum(l2_norm(arr1 - arr2))
end
```

If we assume that there are n strips to be sorted, this greedy SSD algorithm as the time complexity of *O(n**2)*, since it needs to compare (n-1) + (n-2) + ... + 1 strips for all iterations.

Except Alma Statue, Larry Roberts's picture as been successfully reconstructed as well.

3. Align and Re-order [15 pts]

Now let's move on to the unsorted strips with offset and try to reconstruct it based on *zero mean normalized cross correlation* algorithm.

```
$ python main.py --img-name hard_almastatue --mode zncc
> Namespace(dst_dir='results/', img_name='hard_almastatue', mode='zncc',
src_dir='../shredded-images/')
> collapsed time: 160.3736 [s]
```

It seems like the strips are properly aligned sorted as well. The sorting algorithm is implemented in a member function `sort_strips_zncc` of `ImgUnshredder` class. Below is the pseudocode of `sort_strips_zncc` function.

```
input : strips_unsorted, strips_sorted, relative_offset
goal  : move all elements from strips_unsorted to strips_sorted with a
proper order, while recording the relative offset between left to right
strips in relative_offset.

do
    for all strip_being_compared in strips_unsorted:
        max_zncc <- 0
        similarest_strip <- None
        similarest_offset <- None
        append_direction <- None

        for all offset in offset_range
            if zncc(rightmost(strips_sorted),
leftmost(strip_being_compared), offset) > max_zncc then
                max_zncc <- -ssd(rightmost(strips_sorted),
leftmost(strip_being_compared))
                similarest_strip <- strip_being_compared
                similarest_offset <- offset
                append_direction <- right
            else if -ssd(rightmost(strip_being_compared),
leftmost(strips_sorted), offset) > max_zncc then
                max_zncc <- -ssd(rightmost(strip_being_compared),
leftmost(strips_sorted))
                similarest_strip <- strip_being_compared
                similarest_offset <- offset
                append_direction <- left
            end
        end
    end
```

```
    move similarest_strip from strips_unsorted to strips_sorted, by
appending on the append_direction
    add similarest_offset in relative_offset, by appending on the
append_direction

until len(strips_unsorted) == 0
```

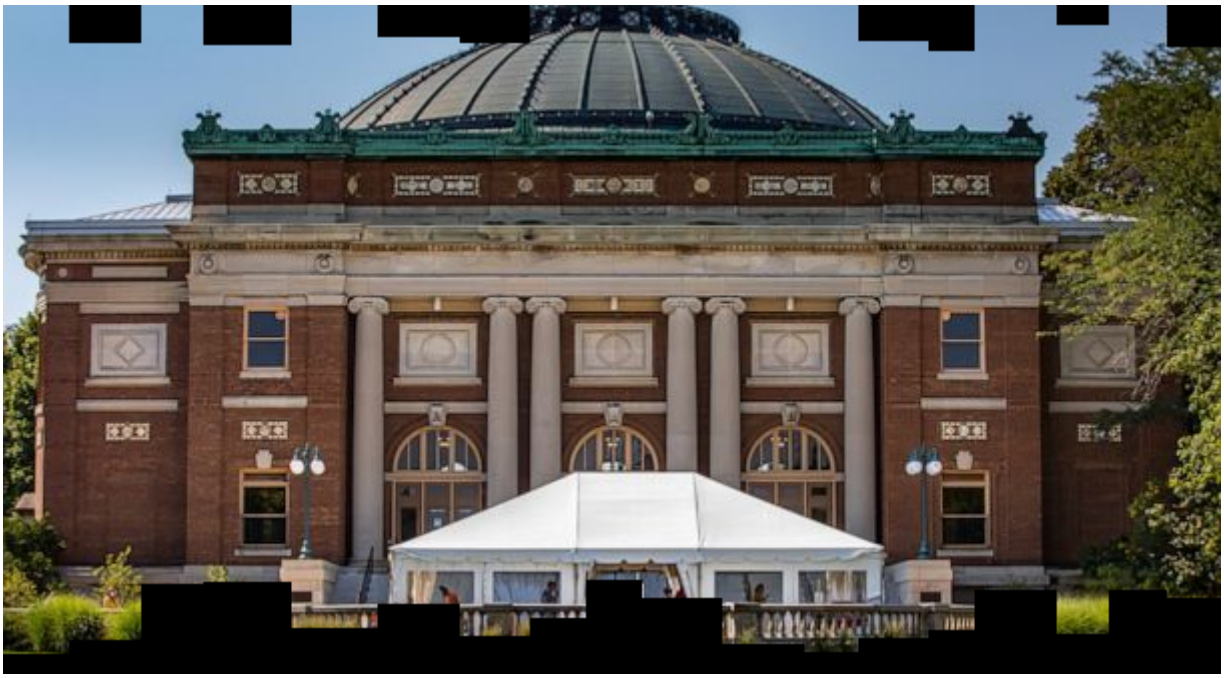
where ZNCC can be computed as below:

```
function zncc(arr1, arr2)
    sum_zncc = 0
    for n in range(size(arr1)):
        sum_zncc += (arr1[n] - avg(arr1)) * (arr2[n] - avg(arr2)) /
(std(arr1) * std(arr2))

    return sum_zncc / size(arr1)
end
```

It takes significantly longer time than SSD algorithm. This is due to the fact that ZNCC with offset algorithm has the time complexity of *O(n**2 * o)* whereas SSD has *O(n**2)* (*n* : the number of strips, *o* : the pixel offset to be investigated), which may have resulted in such a significant computation burden.

Like Alma Statue, Other unsorted and misaligned images has been successfully reconstructed as well.



[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] i[redacted] construction o[redacted] system comp[redacted] enough to be a r[redacted]