Kohonen Self Organising Maps (SOM)

GNR 602 : Advanced Methods in Satellite Image Processing
Course Project

Durgaprasad Bhat 200070017
Rohan Kalbag 20D170033
Siddharth Anand 20D070076

Presentation Outline

Problem Statement

Problem Statement

Implement Kohonen Self-Organizing Map, with user-specified grid matrix size, and a multispectral image as input. Generate a coded image using the trained SOM as a code book

Introduction

Self Organizing Maps - Kohonen Maps

- Developed by Finnish professor and researcher **Dr. Teuvo Kohonen** in 1982
- It is an unsupervised learning method, trained using competitive learning
- **multi-dimensional data** is mapped to a compact **two-dimensional grid**
- Given the two-dimensional grid of weights, a coded representation can be generated for multidimensional data, which is much easier to store
- The multidimensional data can be restored by using the coded image, however some information is lost as we will see later.

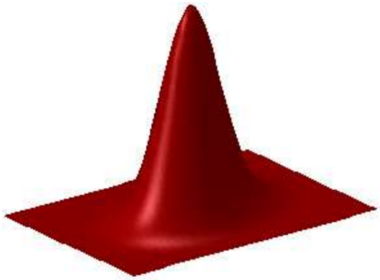
Method

Learning Rate

$$\alpha_t = \alpha_0(1 - \frac{t}{T})$$

- T : total number of training iterations
- t : is the current training iteration
- α_t : is the learning rate for the current training iteration
- α_0 : is the initial value of the learning rate (user selected)

Neighbourhood function



Neighbourhood function

$$h(\Delta x, \Delta y) = e^{\left(-\frac{1}{2}\frac{(\Delta x)^2 + \Delta y)^2}{\sigma_t^2}\right)}$$

$$\sigma_t = \sigma_0 \cdot e^{\left(\frac{-t}{T}\right)}$$

- σ_o : tunable parameter set to $\frac{(l^2+b^2)}{16}$
- t : current iteration, T : Max number of iterations

System Requirements

- Our implementation was developed on **Python 3.x**
- All the python dependencies required are in `requirements.txt`
- Made of Python libraries such as **numpy** for storing arrays, **matplotlib** for plotting, **pillow** for basic image input-output functions also, **numba**'s Just-In-Time compilation was used to accelerate the code and significantly reduce computation time
- Method to setup the python environment can be found in `README.md`

Implementation

The codebase has been modularly developed into the following functions

- `norm()`
- `find_best_matching_unit()`
- `curr_lr()`
- `neighbourhood_func()`
- `update_weights()`
- `fit()`
- `generate_coded_image()`
- `generate_image_from_coded()`

norm()

```
def norm(x, y):
    return np.sum((x - y)**2)
```

- Given two vectors x and y it returns the euclidean norm corresponding to those vectors
-
$$||x - y|| = \sum_{i=1}^{N}((x_i - y_i)^2)$$

find_best_matching_unit()

```
def find_best_matching_unit(x, length, breadth, kohonen_map):
    running_min = norm(kohonen_map[0, 0], x)
    bmu = (0, 0)
    for i in range(length):
        for j in range(breadth):
            if(norm(kohonen_map[i,j], x) < running_min):
                bmu = (i, j)
                running_min = norm(kohonen_map[i,j], x)

    return bmu
```

- Given a vector x, this function returns the index (i, j) corresponding to the closest weight vector in the kohonen map
- $$(i, j) = argmin_{i,j}(||x - w_{i,j}||)$$

curr_lr()


```
def curr_lr(learning_rate, curr_iter, max_iter):
    clr = learning_rate*(1 - (curr_iter/max_iter))
    return clr
```

- Returns the current learning rate, given the initialized learning rate
- $$\alpha_t = \alpha_o(1 - \frac{t}{T})$$

neighbourhood_fn()


```python
def neighbourhood_func(del_x, del_y, length, breadth, curr_iter, max_iter):
        sigma_t = (length**2 + breadth**2)*np.exp(-curr_iter/max_iter)/16
        return np.exp(-(del_x**2 + del_y**2)/2/sigma_t)
```

- The neighborhood function decides weights of neighbour nodes
- It uses a Gaussian function of the distance between the BMU and the current unit
- $$\sigma_t = 0.0625(l^2 + b^2)e^{-\frac{t}{T}}$$
- $$h(\Delta x, \Delta y) = e^{-\frac{(\Delta x)^2 + (\Delta y)^2}{2\sigma_t}}$$

update_weights()


```
def update_weights(bmu, x, length, breadth, kohonen_map, curr_iter, max_iter, learning_rate):
    clr = curr_lr(learning_rate, curr_iter, max_iter)
    for i in range(length):
        for j in range(breadth):
            delta_w_ij = clr*(x - kohonen_map[i, j])
            delta_w_ij = neighbourhood_func(abs(bmu[0] - i),
            abs(bmu[1] - j), length, breadth,
            curr_iter, max_iter)*delta_w_ij
            kohonen_map[i, j] += delta_w_ij
```

- Updates the weight vector using neighbourhood function and learning rate given the vector x
- $$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$
- $$\Delta w_{i,j} = \alpha_t h(\Delta x, \Delta y)(x - w_{i,j})$$

fit()


```
def fit(length, breadth, img, max_iter, learning_rate, progress):
    kohonen_map = np.random.rand(length, breadth, img.shape[2])*255
    for i in range(max_iter):
        curr_iter = i
        for u in range(img.shape[0]):
            for v in range(img.shape[1]):
                x = img[u, v]
                bmu = find_best_matching_unit(x, length,
                breadth, kohonen_map)
                update_weights(bmu, x, length, breadth, kohonen_map,
                               curr_iter, max_iter, learning_rate)
        progress.update(1)
    return kohonen_map
```

- This function trains the Kohonen map using the input image
- It initializes the map randomly and then iterates through the image pixels (for a particular maximum number of iterations), finding the BMU for each pixel and updating the weights of the Kohonen map

generate_coded_image()


```python
def generate_coded_image(img, kohonen_map):
    coded_img = np.zeros(shape=(img.shape[0], img.shape[1], 2))
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            rgb_val = img[i, j]
            closest = find_best_matching_unit(
                rgb_val, kohonen_map.shape[0],
                kohonen_map.shape[1], kohonen_map)
            coded_coordinates = np.array(closest)
            coded_img[i, j] = coded_coordinates
    return coded_img
```

- This function generates a compressed image by replacing each pixel of the original image with the coordinates of its BMU in the Kohonen map
- The resulting image has two channels, corresponding to the x and y coordinates of the BMU (Code Book)

generate_coded_image()

```
def generate_coded_image(img, kohonen_map):
    coded_img = np.zeros(shape=(img.shape[0], img.shape[1], 2))
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            rgb_val = img[i, j]
            closest = find_best_matching_unit(
                rgb_val, kohonen_map.shape[0],
                kohonen_map.shape[1], kohonen_map)
            coded_coordinates = np.array(closest)
            coded_img[i, j] = coded_coordinates
    return coded_img
```

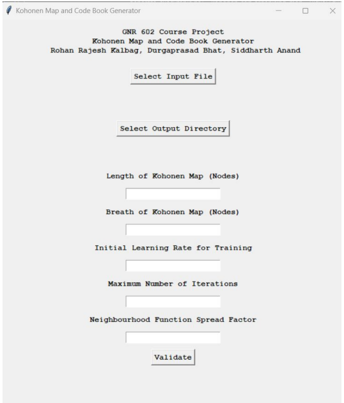
- This function generates a compressed image by replacing each pixel of the original image with the coordinates of its BMU in the Kohonen map
- The resulting image has two channels, corresponding to the x and y coordinates of the BMU (Code Book)

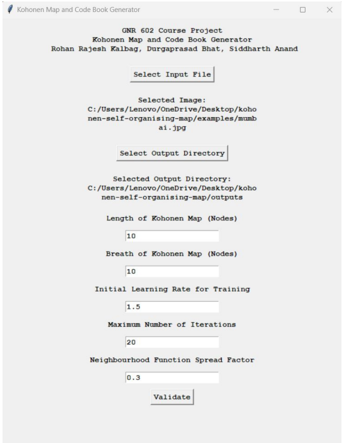
generate_image_from_coded()


```python
def generate_image_from_coded(coded_img, kohonen_map):
    reconstructed = np.zeros(
        shape=(coded_img.shape[0], coded_img.shape[1],
        kohonen_map.shape[2]))
    for i in range(coded_img.shape[0]):
        for j in range(coded_img.shape[1]):
            m = np.array(coded_image[i, j], dtype=np.int32)
            reconstructed[i, j] = kohonen_map[m[0], m[1]]
    return reconstructed
```

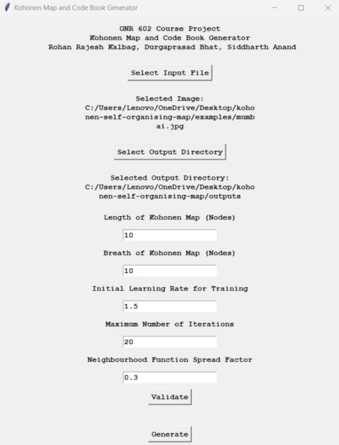
- This function tries to reconstruct an image from its coded representation and its kohonen map
- It assigns each pixel the value corresponding to the weight vector in its coded representation and returns the reconstructed image

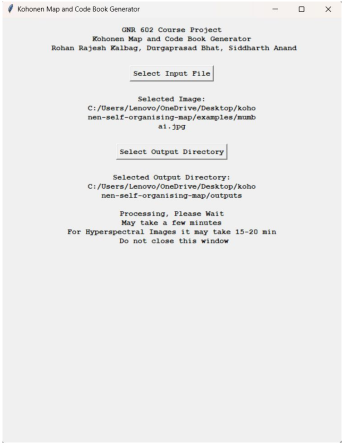
GUI (Graphical User Interface)





GUI (Graphical User Interface)





Results

Results Obtained

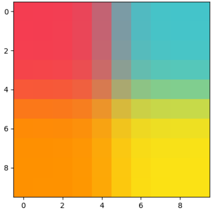
We discuss the experiments conducted using Kohonen Self-Organizing Map (SOM) on different images, including an initial color image (4 color kite), a Mumbai satellite image, and a hyperspectral image of Kennedy Space Center

Experiment 1: 4 Color Kite Image

- **Image source**: `https://www.schemecolor.com/4-colors-kite.php`
- **Kohonen map size**: 10x10
- **Learning rate**: 0.1
- **Number of iterations**: 10
- Saved the Kohonen map as 'initimg_kohonen.png' and the coded image as 'initimg_coded_image.npy'
- Reconstructed the original image from the coded image and Kohonen map, saved as 'initimg_restored.png'
- **Results**: Kohonen map successfully captured 4 colors of the original image, and the reconstructed image is identical to the original image

Results: Experiment 1

4 Color Kite Image
(Resized to: 200X200)



Kohonen Map

Reconstructed Image

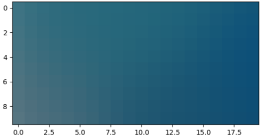
Experiment 2: Mumbai Satellite Image

- **Image source**: `https://www.esa.int/ESA_Multimedia/Images/2005/03/Bombay_seen_by_Proba_satellite`

Experiment	Learning Rate	Spread of Neighborhood Function	Kohonen Map Size	No, of iterations
2.1	0.5	16	10 X 20	100
2.2	0.5	32	15 X 25	50
2.3	0.5	48	20 X 35	50
2.4	0.3	48	20 X 35	50
2.5	0.25	48	20 X 35	100

Results: Experiment 2.1

Satellite Image of Mumbai
(Resized to: 200X200)

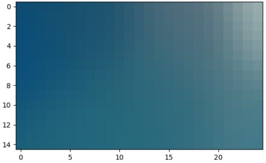


Kohonen Map

Reconstructed Image

Results: Experiment 2.2

Satellite Image of Mumbai
(Resized to: 200X200)

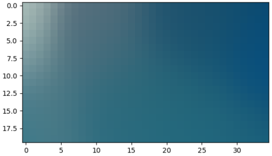


Kohonen Map

Reconstructed Image

Results: Experiment 2.3

Satellite Image of Mumbai (Resized to: 200X200)

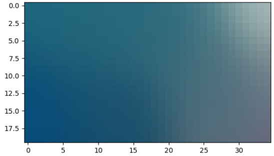


Kohonen Map

Reconstructed Image

Results: Experiment 2.4

Satellite Image of Mumbai (Resized to: 200X200)

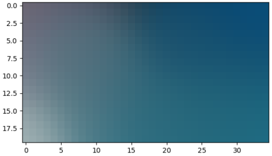


Kohonen Map

Reconstructed Image

Results: Experiment 2.5

Satellite Image of Mumbai
(Resized to: 200X200)



Kohonen Map

Reconstructed Image

Results and Derivatives

We have experimented with five different sets of parameters for image reconstruction. As visible, there is a gradual improvement in the image quality with the use of better parameters.
The first reconstructed image shows a lot of blue tint, which indicates that the parameters used were not optimal for image reconstruction. As we move to the last image, we can see a very clear reconstruction, which is a result of using better parameter choices.
Our experiment shows that with better parameter choices, we can significantly improve the quality of reconstructed images from the Kohonen Map.

Experiment 3: Hyperspectral Image

- **Image source**: `https://www.ehu.eus/ccwintco/index.php?title=Hyperspectral_Remote_Sensing_Scenes#Kennedy_Space_Center_.28KSC.29`
- **Kohonen map size**: 10x10
- **Learning rate**: 0.4
- **Number of iterations**: 50
- Loaded the hyperspectral image from 'KSC.mat'
- Saved the coded image as 'KSC_coded_image.npy'
- **Results**: Kohonen map successfully captured features in the hyperspectral image (KSC_coded_image.npy)

Thank You!