Visualizing Image Structural Similarity Using Self-Organizing Maps and HOG Features

Ernests Lavrinovics
MSc Medialogy MED08
Aalborg Univeristy, Copenhagen

Structure

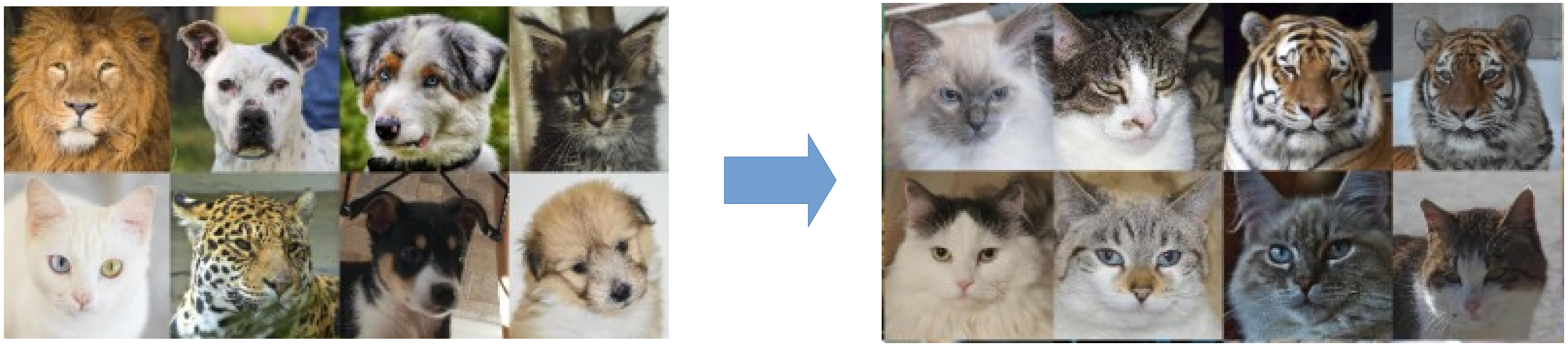
- Introduction
  - Image Features
  - Structural Similarity
  - Self-Organizing Maps (SOM)
- Implementation Details
- Dataset
- Results
- Analysis and Conclusions

Introduction
Image Features

- Features quantitatively describe an image
  - ORB, SIFT features use local corners to describe the image
    - Amount of features vary for resolution X
  - HOG splits an image into small cells and computes gradients
    - Amount of features are constant for resolution X
- Important because we cannot directly compare image values
  - i.e. identical images with varied lighting gives varied distance
- OpenCV is used for feature computation

Introduction
Structural Similarity

- Euclidean distance of HOG descriptors $$d(x, y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$
- Other metrics exist
  - Structural Similarity Index (SSIM)
  - Learned Perceptual Image Patch Similarity (LPIPS)
  - Mikowski-form distance [1]



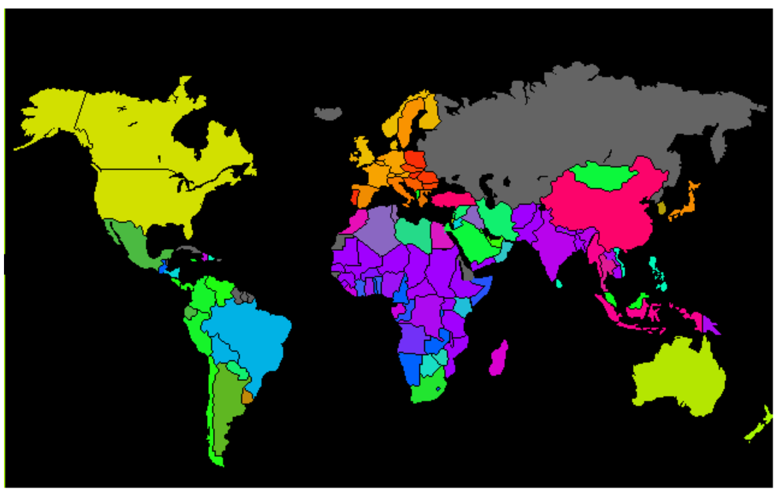
Ernests Lavrinovics
MED08, AAU-CPH

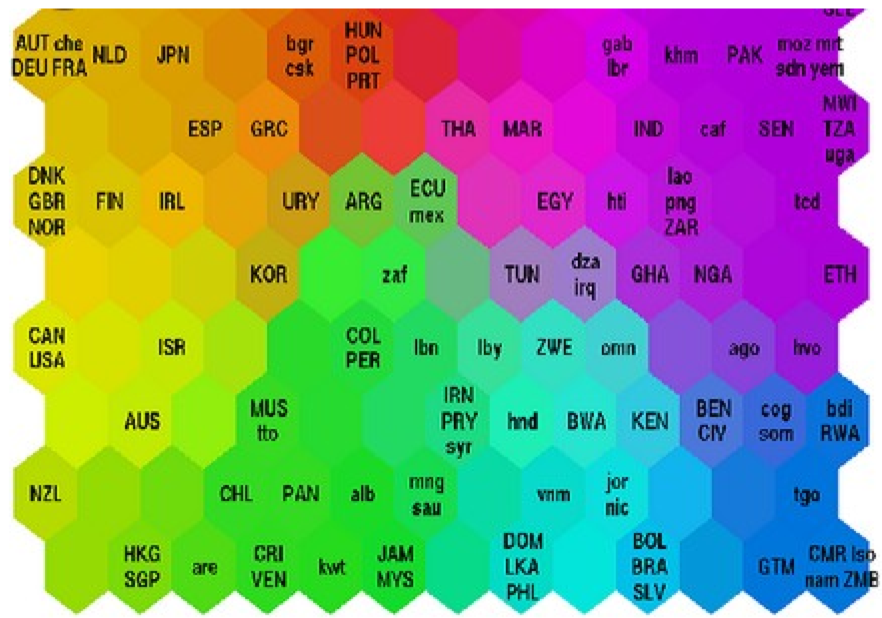
Introduction
Self-Organizing Maps (SOM)

- Algorithm originally published by T.Kohonen, sometimes referred to as *Kohonen Map*
- Unsupervised machine learning algorithm
- Used for dimensionality reduction with topological preservation
- Usually visualized as a 2D map

Introduction
Self-Organizing Maps (SOM)

- Visualizing world poverty [3]





Introduction

Self-Organizing Maps (SOM)

- Training

1) Initialize nodes with random values
2) Take a random sample from the training data
3) Find the pre-initialized node with the lowest distance to the sample. *Best Matching Unit (BMU)*
4) Take the neighbourhood of BMU
5) For each node in the neighbourhood, adjust them according to BMU
    1) New Weight = Old Weight + Learning Rate (Input Vec. - Old Weights)
6) Repeat from Step 2 for N iterations

Implementation Details
Initializing Nodes

- Randomly samples images from the training data set

```
def sampleListRandomly(input : np.ndarray, n=10):
    """ Returns a list of a randomly sampled input list """
    return np.random.choice(input, size=(n, n))
```

Implementation Details
Training

- Shuffle the training data
- Find the BMU and update weights

```python
def trainSOM(self, SOM, trainingData):
    for epoch in range(self.epochs):
        print(f"Epoch: {epoch} / {self.epochs}")
        gridImg = gallery(SOM)
        saveImg(gridImg, str(epoch))
        np.random.shuffle(trainingData)
        for trainingExample in trainingData:
            g, h = self.findBmu(SOM, trainingExample)
            SOM = self.updateWeights(SOM, trainingExample,
```

Implementation Details
Best-Matching Unit

- Find the index of the pre-initialized image with its closest distance with respect to HOG descriptor

```python
def findBmu(self, SOM, x):
    # compare euclidean distance of each element of the SOM to the input
    # and save the index of the closest element to bmu
    distances = np.zeros((SOM.shape[0], SOM.shape[1]), dtype=np.float32)
    for width in range(len(SOM)):
        for length in range(len(SOM)):
            element = SOM[width][length].features

            # Get euclidean distance of image features
            d = np.linalg.norm(element - x.features)

            # Insert the euclidean distance into the distances array
            distances[width][length] = d

    # Get the index of the minimum distance
    min_index = np.unravel_index(np.argmin(distances, axis=None), distances.shape)

    # return the coordinates of the minimum distance
    return min_index
```

Implementation Details
Updating Weights

- Key flaw of the project
- Take a random neighbor of BMU and change it to *input*
- How to update neighboring nodes when using images?

```python
def updateWeights(self, SOM, trainingExample, learningRate, radius, bmuCoord, step=1):
    g, h = bmuCoord

    # Get neighbourhood range, prevent overshooting from edges
    rWidth = (max(0, g - step), min(SOM.shape[0] - 1, g + step))
    rHeight = (max(0, h - step), min(SOM.shape[1] - 1 , h + step))

    # Get the random neighbour
    rw = np.random.randint(low=rWidth[0], high=rWidth[1])
    rh = np.random.randint(low=rHeight[0], high=rHeight[1])

    # Prevent changing the BMU itself..
    while (rw == g and rh == h):
        rw = random.randint(rWidth[0], rWidth[1])
        rh = random.randint(rHeight[0], rHeight[1])

    # Set the image at location
    SOM[rw, rh] = trainingExample
    return SOM
```

Results

Results

Analysis and Conclusions

- Obvious flaw in the code that does not follow the algorithm pseudo-code
- Multiple epochs can lead to duplicate images due to presenting every data point per epoch
- Different data set could be used with more separable features

Thank you!

References

- [1] Mei, T., Rui, Y. (2009). Image Similarity. In: LIU, L., ÖZSU, M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-39940-9_1014
- [2] T. Kohonen, "The self-organizing map," in Proceedings of the IEEE, vol. 78, no. 9, pp. 1464-1480, Sept. 1990, doi: 10.1109/5.58325.
- [3] http://www.cis.hut.fi/research/som-research/worldmap.html