Image Processing and Pattern Recognition

Assignment 2 - Patch Denoising with the Gaussian Mixture Model

November 20, 2019

Deadline: December 05, 2019 at 23:55h.
Submission: Upload your report and your implementation to the TeachCenter. Please do not zip your files. Please use the provided framework-file for your implementation and use the provided functions.

1 Introduction

We have seen in the lecture how to denoise an image (or equivalently) patches with a variant of the Wiener Filter. The example has shown that the denoising works in general. However, it turns out that the Gaussian distribution is not very well suited for representing the prior patch-statistics of natural images. Thus, in this assignment we will replace the Gaussian with a Gaussian Mixture Model (GMM)

$$p(x, \theta) = \sum_{k=1}^{K} \alpha_k \mathcal{N}(x; \mu_k, \Sigma_k), \tag{1}$$

where $x \in \mathbb{R}^n$ is an image patch, $\theta = \{\alpha_k, \mu_k, \Sigma_k\}_{k=1}^K$ with $\alpha_k > 0$ and $\sum_k \alpha_k = 1$ are the learnable parameters and $\mathcal{N}(x; \mu_k, \Sigma_k)$ are the multivariate Gaussian distributions defined as

$$\mathcal{N}(x; \mu_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-\frac{1}{2}(x - \mu_k)^T \Sigma^{-1} (x - \mu_k)\right), \tag{2}$$

where d is the dimensionality of the Gaussian.

2 Method

We want to solve the following optimization problem for all patches x_i:

$$\hat{x}_i = \arg\min_x \frac{\lambda}{2} \|x - y_i\|_2^2 - \log\left(\sum_{k=1}^{K} \alpha_k \mathcal{N}(x; \mu_k, \Sigma_k)\right) \tag{3}$$

Thus, this optimization problem requires already a learned GMM which is used as the prior distribution $p(x; \theta)$. Note, that *learned* simply means that we have computed optimal values for the learnable parameters $\theta = \{\alpha_k, \mu_k, \Sigma_k\}_{k=1}^K$ using the Expectation-Maximization (EM) algorithm.

3 Task

Your task is to perform a patch-denoising with the model shown in (3). This requires to implement two main steps: First you need to train a GMM on a bunch of noise-free images to get the optimal parameters for the GMM, and second you need to implement the optimization arising in (3).

Learning the GMM The optimal parameters for the GMM can be computed with the EM algorithm. This algorithm consists of the two building blocks *expectation* and *maximization*. The former computes the assignment of all datapoints to the most likely Gaussian and is defined as

$$\gamma_k^j(x_i) = \frac{\alpha_k^j \mathcal{N}(x_i; \mu_k^j, \Sigma_k^j)}{\sum_{k=1}^K \alpha_k^j \mathcal{N}(x_i; \mu_k^j, \Sigma_k^j)} \tag{4}$$

and the latter updates the Gaussians based on the updates the weights α_k with

$$\alpha_k^{j+1} = \frac{1}{n} \sum_{i=1}^n \gamma_k^j(x_i), \tag{5}$$

the means μ_k with

$$\mu_k^{j+1} = \frac{\sum_{i=1}^n \gamma_k^j(x_i) x_i}{\sum_{i=1}^n \gamma_k^j(x_i)} \tag{6}$$

and the variance Σ_k with

$$\Sigma_k^{j+1} = \frac{\sum_{i=1}^n \gamma_k^j(x_i)(x_i - \mu_k^{j+1})(x_i - \mu_k^{j+1})^T}{\sum_{i=1}^n \gamma_k^j(x_i)}, \tag{7}$$

where the indices i is the index for the samples, *i.e.* the patches in our case, k is the index of a Gaussian in the GMM and j is the iteration index of the EM-algorihm. Implementing the EM-algorithm with eqs. (4) to (7) can be numerically unstable due to the exponent in the normal distribution eq. (2). Thus, we need to implement eq. (4) in the log-domain and apply the *log-sum-exp trick*. As an example, consider the implementation of

$$\begin{aligned} r(x) &= \sum_i \exp(x_i) \\ &= \exp\left(z + \log \sum_i \exp(x_i - z)\right), \end{aligned}$$

where $z = \max_i x_i$. This ensures that the maximal value in the exponent is zero and thus, we avoid numerical problems. Equations (5) to (7) do not require any special consideration and thus can be implemented as shown. The means μ_k should be intitialized with zeros, because all our input patches are pre-processed to have zero mean. The matrix Σ_k should be a symmetric positive definite matrix. This can be achieved by initializing by first initializing a matrix $\hat{\Sigma}$ with a normal distribution and then compute

$$\Sigma = \hat{\Sigma}^T \hat{\Sigma}. \tag{8}$$

Computing a solution to (3) When we have learned the optimal parameters for the GMM it remains to perform the actual denoising task with (3). This can be done by the fast iterative approximation propsed by Zoran and Weiss. The algorithm basically iterates two steps: First it assigns the patch to the most appropriate Gaussian and second this Gaussian is then used to apply the Wiener filter. The algorithm is defined precisely in Algorithm 1. Note, that finding k_{max} requires also a numerically stable implementation. You can recycle the your implementation you used for the EM-algorithm for this task. The matrix E is used to normalize the patches and is defined as shown in the lecture:

$$E = \mathrm{id} - \frac{1}{m} \mathbf{1} \otimes \mathbf{1} \tag{9}$$

4 Framework & Tasks

We have provided a framework with some core functionality for you. It consists of the data for learning the GMM, a validation set for validating your results, a noisy test set without ground-truth and function stubs for your implementation.

Algorithm 1: Patch-Denoising with GMM and Wiener filter

Data: Learned GMM; Noisy patch y_i, α
Result: Denoised patch $\hat{x}_i$
while *not converged* **do**
`/* Compute best GMM component */`
$k_{max} = \arg\max_k \log\left(\alpha_k \mathcal{N}(Ex_i^j; \mu_k, \Sigma_k)\right)$
`/* Apply Wiener filter */`
$\tilde{x}_i = (\lambda id + E^T \Sigma_{k_{max}}^{-1} E)^{-1}(\lambda y_i + \Sigma_{k_{max}}^{-1} E\mu_{k_{max}})$
`/* Update */`
$x_i^{j+1} = \alpha x_i^j + (1-\alpha)\tilde{x}_i$
$\hat{x}_i = x_i^{j+1}$
$j = j + 1$
end

- Load the debugging dataset `simple.npy` and use it while implementing the GMM algorithm. This dataset consists datapoints in 2D and thus you can easily visualize the progress of the algorithm and it is much easier to verify the correctess of your implementation on this small dataset. You can load the data with `np.load("simple.npy")`.
- Unzip and load the datasets `train_set`, `valid_set` and `test_set.npy` and convert them to a patch representation. We have provided a loading function for you.
- Learn the GMM with the training-set and plot the means μ_k and the covariances Σ_k (we have provided a function for that in the framework)
- Use your implementation of Algorithm 1 and apply it to the noisy images of the validation set
- Compute the PSNR for all images in the validation set
- Plot the denoised images together with their respective PSNR values
- Experiment with the hyper parameters of the GMM, i.e. change the number of Gaussians and change the filter size. Show the impact of the changes on the validation set. You should have at least 2 examples for the number of Gaussians and 2 examples for the filter size.
- Discuss your findings in the report
- Is a larger number of Gaussians in the GMM and a larger filter-size always better? ("yes" or "no" are inappropriate answers here! Discuss this question and argue why you gave your answer.)
- Challenge: Use your best model and apply it to the noisy test set. Add the denoised test set images to your submission and name it `groupXX_imgY.png`, where XX is your group number and Y is the image name. We will compare your denoised results with the ground-truth (noise-free image) and compute the PSNR. The best three groups will get 5, 3, 1 bonus points and the group which achieves the highest average PSNR on the test-set will win a small present :)