

Image Denoising using Markov Chain Monte Carlo & EM Based Methods

ISYE 6416 Final Project

*By Athanasios Taprantzis, Sri Julapally
Group 19*

Problem

Images often contain levels of noise or corruption due to phenomena such as poor lighting, compression, and downsampling. Restoring these images accurately is a challenge in computational statistics and computer vision.

Our solution:

- Implement EM-GMM and Markov Random Fields to denoise images
- Test with standardized dataset
- Create accessible user interface for people to experiment with

Approach/Methods

Markov Random Fields:

- We try and estimate posterior $p(y|x)$
- Energy of the system (neg log-likelihood):

$$E(y_{i,j}|X) = E(y_{i,j}|N(y_{i,j}))$$
$$L(y_{i,j}|X) = \sum_{z \in N(y_{i,j}) \setminus x_{i,j}} \|z - y_{i,j}\|_{n_1}^{n_1} + \lambda \|x_{i,j} - y_{i,j}\|_{n_1}^{n_1}.$$

- The probability can be formulated as: $p(y_{i,j}|N(y_{i,j})) = \frac{1}{Z} \exp(-E(y_{i,j}|X)),$

- Metropolis-Hastings to sample from the above distribution for each pixel
- We perform random walks for each pixel. State transitions are sampled from a gaussian
- We can ignore the normalisation coefficient (sum of all scores)

Approach/Methods

Expectation-Maximisation:

- **EPLL** (Expected Patch Log Likelihood):
 - Create **Prior** patches (from some clean dataset) that you try and match to the noisy image
 - Priors are usually created with GMM
 - Two sided game (as before):
 - i. Try and create patches that approximate the noisy image
 - ii. Try and create patches that approximate the prior
 - Interesting Optimisation Problem
 - i. In short, we have to introduce a trick to optimise it efficiently: “Half Quadratic Splitting”
 - Extremely effective method for sharper results (EPLL term)
 - i. Previous techniques use simple averaging of patches
- **Adaptive Image Denoising by Mixture Adaptation**
 - Extends EPLL
 - Prior is further fitted on the noisy data
 - Helps use prior patches that better suit the noisy image

$$f_{EPLL}(x|y) = \lambda \|Hx - y\|^2 - EPPLL(x)$$

$$EPPLL_p(\mathbf{x}) = \sum_i \log p(\mathbf{P}_i \mathbf{x})$$

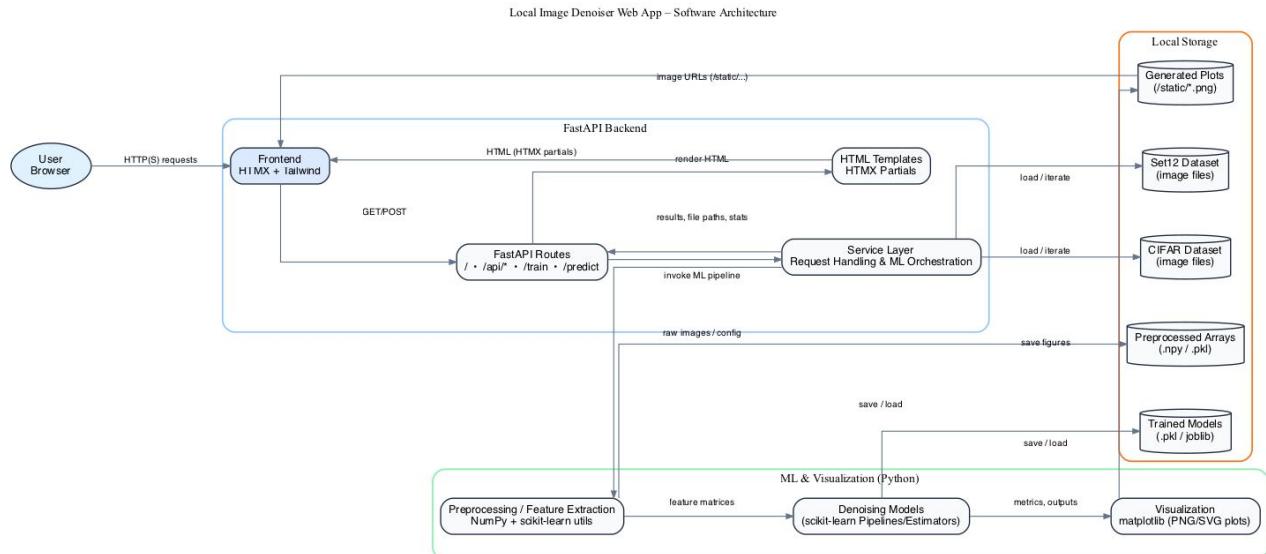
Software Architecture

Modern Web App Architecture

- Accessible to users
- Easy to deploy/update

Technologies

- Numpy/Scikit-learn
- FastAPI
- HTMX

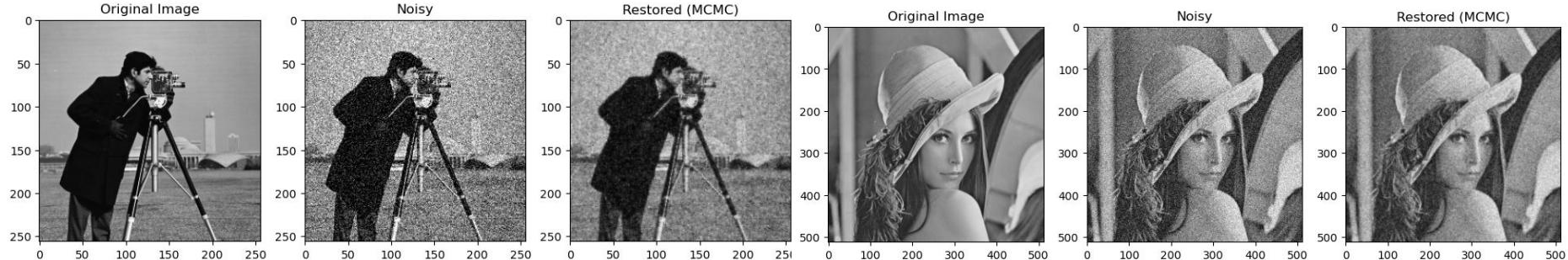


Preliminary Results/Impact

EM



MCMC



Thank you!