Computer Vision I : Project 3 (10 points)

Experiments with the Deep FRAME model

Due on Nov. 8th 11:59pm

Objective. The objective of this project is to experience the multi-layered hierarchical FRAME model (sometimes also called Deep FRAME), which has a similar structure to the convolutional Neural Networks (CNN). In contrast to CNN, Deep FRAME is a generative (descriptive, to be more precise) model in the sense that (i) it extracts a representation that can reconstruct (encode) the input image; (ii) it has an explicit dictionary for the basis/filters, which you will learn and visualize in this project. Details of the model and algorithm are discussed in the textbook.

In comparison to the FRAME model that we experimented in Project 2, this model has the following difference: (a) it tries to match statistics on the responses of neurons at 2-3 layers, not just 1 layer; (b) For simplicity, instead of matching the whole histogram of a neuron's response, it only matches the expected response; and (c) instead of pursuing neurons from a hand-designed set (e.g. the Gabor filters), it learns the filters.

The number of neurons in the new model may be excessive and thus this model is dense and not sparse. It depends on the design of the structure (how many layers, how many neurons, i.e. filters, at each layer). It is still unclear at this stage how we can design a good structure for different images in different entropy regimes. So, it is left to you to explore!

1 Introduction to Hierarchical FRAME Model

Notations. Let $\mathbf{I}(x)$ be an image defined on the square (or rectangular) image domain Λ, where $x = (x_1, x_2)$ indexes the coordinates of pixels. We can treat $\mathbf{I}(x)$ as a two-dimensional function defined on Λ. We can also treat $\mathbf{I}$ as a vector if we fix an ordering for the pixels.

For a filter (or neuron) F, let $F * \mathbf{I}$ denote the filtered image or feature map, and let $[F * \mathbf{I}](x)$ denote the *filter response* or *feature* at position x.

A hierarchical FRAME model is a composition of multiple layers of linear filtering and element-wise non-linear transformation as expressed by the following recursive formula:

$$[F_k^{(l)} * \mathbf{I}](x) = h\left(\sum_{i=1}^{N_{l-1}} \sum_{y \in \mathcal{S}_l} w_{i,y}^{(l,k)} [F_i^{(l-1)} * \mathbf{I}](x+y) + b_{l,k}\right) \tag{1}$$

where $l \in \{1, 2, ..., \mathcal{L}\}$ indexes the layer. $\{F_k^{(l)}, k = 1, ..., N_l\}$ are the filters at layer l, and $\{F_i^{(l-1)}, i = 1, ..., N_{l-1}\}$ are the filters at layer $l-1$. $b_{l,k}$ is the bias term. k and i are used to index filters at layers l and $l-1$ respectively, and N_l and N_{l-1} are the numbers of filters at layers l and $l-1$ respectively. The filters are locally supported, so the range of y is within a local support $\mathcal{S}_l$ (such as a 7×7 image patch). At the bottom layer, $[F_k^{(0)} * \mathbf{I}](x) = \mathbf{I}_k(x)$, where $k \in \{R, G, B\}$ indexes the three color channels. Sub-sampling may be implemented so that in $[F_k^{(l)} * \mathbf{I}](x)$, $x \in \Lambda_l \subset \Lambda$.

We take $h(r) = \max(r, 0)$, the rectified linear unit (re-lu), as is commonly adopted in ConvNet. We define the following random field model as the hierarchical FRAME model:

$$p(\mathbf{I}; w) = \frac{1}{Z(w)} \exp\left[\sum_{k=1}^{K} \sum_{x \in \Lambda_L} [F_k^{(L)} * \mathbf{I}](x)\right] q(\mathbf{I}), \tag{2}$$

where $w = (\mathbf{w}_k, b_k, k = 1, ..., K)$ are the filters at the top layer. $q(\mathbf{I})$ is the Gaussian white noise model.

Model (2) corresponds to the exponential tilting model with scoring function

$$f(\mathbf{I}; w) = \sum_{k=1}^{K} \sum_{x \in \Lambda_L} [F_k^{(L)} * \mathbf{I}](x). \tag{3}$$

The learning of w from training images $\{\mathbf{I}_m, m = 1, ..., M\}$ can be accomplished by maximum likelihood. In our case, we have only 1 image for training, i.e. $M = 1$. Let $L(w) = \sum_{m=1}^{M} \log p(\mathbf{I}; w)/M$,

$$\frac{\partial L(w)}{\partial w} = \frac{1}{M} \sum_{m=1}^{M} \frac{\partial}{\partial w} f(\mathbf{I}_m; w) - \mathrm{E}_{p(I)}\left[\frac{\partial}{\partial w} f(\mathbf{I}; w)\right]. \tag{4}$$

The expectation can be approximated by Monte Carlo samples. One can sample from $p(\mathbf{I}; w)$ in (2) by the Langevin dynamics:

$$\mathbf{I}_{\tau+1} = \mathbf{I}_\tau - \frac{\epsilon^2}{2}\left[\mathbf{I}_\tau - \frac{\partial}{\partial \mathbf{I}} f(\mathbf{I}; w)\right] + \epsilon Z_\tau, \tag{5}$$

where τ denotes the time step, ϵ denotes the step size, assumed to be sufficiently small, $Z_\tau \sim \mathrm{N}(0, \mathbf{1})$.

We can build up the model layer by layer. Given the filters at layers below, the top layer weight and bias parameters can be learned according to

$$\begin{aligned} \frac{\partial L(w)}{\partial w_{i,y}^{(L,k)}} = & \frac{1}{M} \sum_{m=1}^{M} \sum_{x \in \Lambda_L} \delta_{k,x}^{(L)}(\mathbf{I}_m; w) [F_i^{(L-1)} * \mathbf{I}_m](x+y) \\ & - \frac{1}{\tilde{M}} \sum_{m=1}^{\tilde{M}} \sum_{x \in \Lambda_L} \delta_{k,x}^{(L)}(\tilde{\mathbf{I}}_m; w) [F_i^{(L-1)} * \tilde{\mathbf{I}}_m](x+y), \end{aligned} \tag{6}$$

and

$$\frac{\partial L(w)}{\partial b_{L,k}} = \frac{1}{M}\sum_{m=1}^{M}\sum_{x\in\Lambda_L}\delta_{k,x}^{(L)}(\mathbf{I}_m; w) - \frac{1}{\tilde{M}}\sum_{m=1}^{\tilde{M}}\sum_{x\in\Lambda_L}\delta_{k,x}^{(L)}(\tilde{\mathbf{I}}_m; w). \tag{7}$$

where $\mathbf{I}_m$ are M observed images, and $\tilde{\mathbf{I}}_m$ are $\tilde{M}$ synthesized images sampled from the model. Here, we choose $M = \tilde{M} = 1$.

2 Experiment

For one of the input images, you will learn a hierarchical FRAME model. We define the Julesz ensemble as the set of images that reproduce the observed sufficient statistics over those designed filters. You use Langevin dynamics (code is given) to draw samples from the model. Figure 2 shows an example of synthesis. The synthesis starts from a zero image and the sampling stops when it matches all the sufficient statistics. To reduce the computational complexity, all images are resized into the size of 224 × 224 pixels.

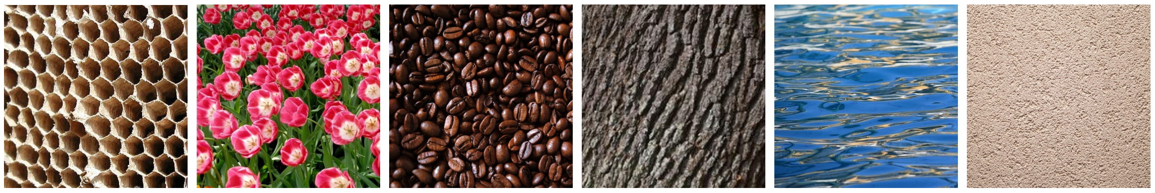

Figure 1: Six images: from low entropy(sparse regime) to high entropy (Gibbs regime).

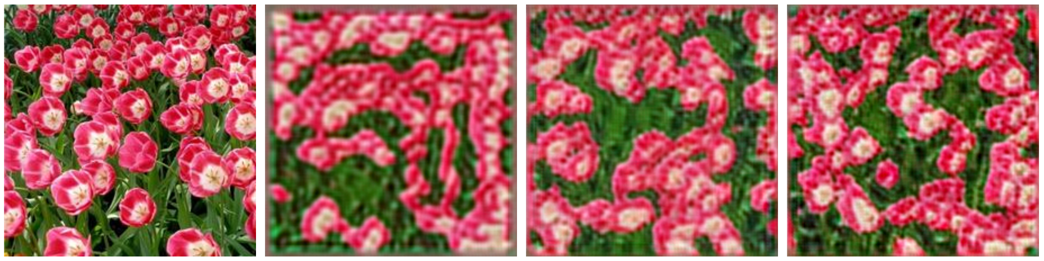

Figure 2: Synthesis example. The first image is the training image, and from left to right, the rest images are synthesized images using 1, 2, and 3 layers respectively.

3 Code

The code is located at `./code/`, and training images locate at `./data/`. The synthesized results will be saved at `.output/images/`, and the model will be saved at `./output/ckpt/`.

1. `frame_beehive_3_layer.py` is the main entry function, which gives an example of learning and synthesizing images from the model.

2. `Descriptor` is the class to design the structure of Deep FRAME model.

Modification: You design your own structures of the model with 1 and 2 convolutional layers. The example setting may not generate vivid results. You might need to adjust the learning rate and descretization step size of the Langevin dynamics.

3. `Model.train()` is the function to learn model and synthesize images.

4. A function is given to visualize filters in the first layer.

The code will determine GPU capabilities and fall back to CPU computation automatically.

4 Submission: results and analysis

In your submission, please provide the following:

1. Show the synthesized images with 1,2,3 layers on the beehive image for your best design.

2. Display the filters in the first layer using your visualization function. Compare these filters across the images. For some selected images, try to compare the learned filtered in different designs.

3. Submit your completed code.