Computer Vision I: Project 4 (10 points)

Due on Dec. 16th, 11:59pm

1 Background

This project is based on Section 11.1: Deep FRAME. Read the textbook for more information.

1.1 Python Library

Please install the latest matplotlib, pillow, scikit-image, torch and torchvision. You are also welcome to utilize any libraries of your choice, **but please report them in your report (for autograder)! Again, report any customized library in the report (do not go too crazy as this will add a significant burden to TAs).**

1.2 What to hand in?

Please submit both a formal report and the accompanying code. For the report, kindly provide a PDF version. You may opt to compose a new report or complete the designated sections within this document, as can be started by simply loading the tex file to Overleaf. Your score will be based on the quality of **your results**, **the analysis** (diagnostics of issues and comparisons) of these results in your report, and your **code implementation**. You may delete all the images before handing them in, as they may be too large for the autograder.

Notice. Do not modify the function names, parameters, and returns in the given code, unless explicitly specified in this document.

1.3 Help

Make a diligent effort to independently address any encountered issues, and in cases where challenges exceed your capabilities, do not hesitate to seek assistance! Collaboration with your peers is permitted, but it is crucial that you refrain from directly examining or copying one another's code. Please be aware that you'll fail the course if our **code similarity checker**, which has found some prohibited behaviors before, detects these violations. For details, please refer to: https://yzhu.io/s/teaching/plagiarism.

2 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 $M = \tilde{M} = 1$. Hint: you can leverage an automatic differentiation engine to simplify the computation of gradients, *e.g.*, PyTorch.

3 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.

4 Code

An overview of the learning of deep FRAME model:

1. Given a target image that specifies a type of texture, our goal is to model the statistics of this texture and thereby synthesize a similar image.

2. The distribution of the image is governed by a FRAME model, with the exponential term determined by a neural descriptor (CNN). The objective is to maximize the likelihood of target image under this distribution. And the gradient w.r.t. the descriptor is derived as in Equation (4), where the first term can be computed by leveraging autograd and the second term can be approximated via Langevin dynamics.

3. Starting from a zero-initialized image, we keep iterating the synthetic image. During each round of training, we first update the synthetic image with a better one via Langevin dynamics. Then with this updated synthetic image, we compute $f(I_{tgt}) - f(I_{syn})$, backward the loss, and update the parameters of descriptor.

4. Notably, the objective to be maximized $f(I_{tgt}) - f(I_{syn})$ would probably oscillate instead of increasing monotonously. This is because sampling the synthetic image via Langevin dynamics would lead to larger $f(I_{syn})$, despite the gradient ascent when updating descriptor parameters.

The training images locate at `images/`. The experimental results will be saved at a folder named according to the configurations, *e.g.*, `rose_3layer` (modeling rose texture with a 3-layer encoder). `deep_frame.py` is the main entry function, which gives an example of learning and synthesizing images from the model. Running this python file requires two arguments, one specifying the number of layers and the other specifying the image tag. For example, train a 3-layer deep FRAME model on rose:

```
python deep_frame.py --layer 3 --tag rose
```

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 rates and sampling step size of the Langevin dynamics.

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

5 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.

Figure 3: Three images: from 1 layer to 3 layer.

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.

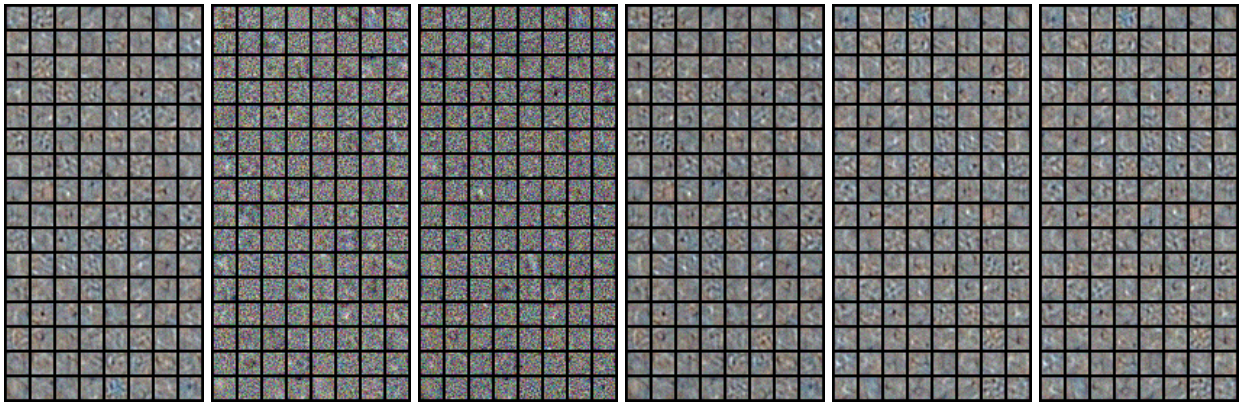

Figure 4: Filters:bark beehive coffee rose stucco water

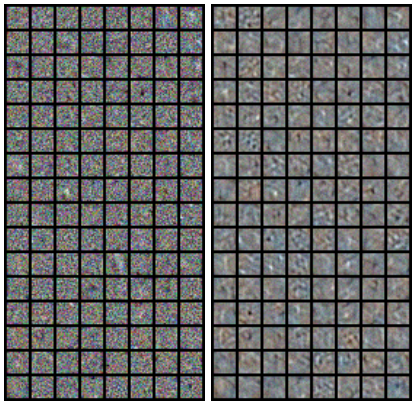

Figure 5: Langevin step size: 0.5(left) and 1.0(right)