Computer Vision I : Project #3 (15 points)

This project is extremely hard and time consuming. For reference: a properly written code would take roughly 16 hours to finish running on a 5-year-old Mac Pro. START EARLY!

1 Background

This project is based on Chapter 3: Textures. You shall read the textbook and understand the underlying logistics before writing your code.

1.1 Python Library

Please install the latest cv2, PIL, numpy, scipy, matplotlib, tqdm, torch (including torch-vision), and the cython (if you want) packages.

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 1) Do not modify the function names, parameters, and returns in the given code, unless explicitly specified in this document. 2) The **maximum score** of the autograder demonstrated is 4.5, another 0.5 will leave for deeper evaluation.

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 for Project 2, detects these violations.

For details, please refer to `https://yzhu.io/s/teaching/plagiarism/`

2 Self-check Questions

You may skip these questions, as the questions below are just some help for accomplishing the project. You don't need to answer them in your report, and the correct answers to the questions should be explored by yourself.

1. Why is the FRAME model structured in the manner delineated within the referenced literature?
2. What is the definition of "sweep" in the FRAME model?
3. How to calculate the distance (difference) of two histograms within the framework of the FRAME model?
4. Why does the FRAME model work? What does the learning process minimize?
5. What is the operational mechanism of the Gibbs Sampler within the FRAME model?
6. What is the learning mechanism of the Julesz Ensemble?
7. What is the relationship between the Julesz Ensemble and the FRAME model?

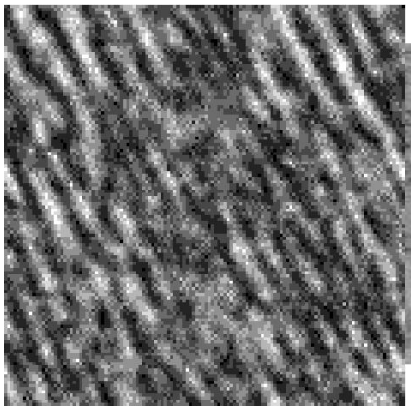
3 Questions To Answer

You should answer the following questions in your report. Please copy the questions in your report and answer **briefly** with no more than 5 sentences.
Hint: It's recommended to finish the coding part before answering the questions.

1. What is the definition of Julesz Ensemble?
2. Why is the histogram tail important in synthesis?
3. Do I need to perform convolution on the whole image while performing gibbs sampling for a single pixel?
4. In the code, what is the recommended method to generate the value of a pixel given the distribution of $p(val|I_{-s})$?
5. How is the conditional probability $p(I_s = val|I_{-s})$ computed within the Gibbs Sampler?
6. Is the FRAME model guaranteed to converge? Why and why not? (*Hint: Read Chapter 9*)
7. How to update λ in the FRAME model?
8. How to select new filters within the FRAME model?

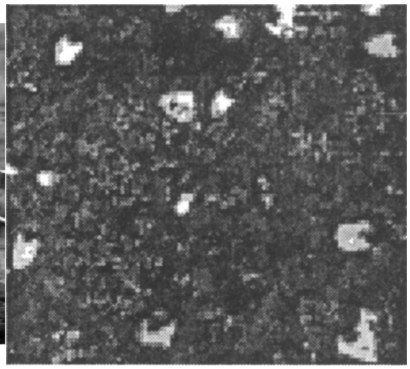
4 Part 1: Julesz Ensemble

Objective This part explores the minimax entropy learning procedure for texture modeling and synthesis. We'll study its capability and limitations. The figure below depicts 3 texture examples included in the zip file.



(a) fur

(b) horse



(c) mud

4.1 Experiment

For each input image, select a set of filters and extract the histograms of filter responses. We define the Julesz ensemble as the images that reproduce the observed histograms based on the chosen filters. The synthesis process commences with a noise image generated from a uniform distribution and progresses until it aligns with all the selected histograms. Specifically, the procedure continues to select additional filters until the filtered histograms match within a defined epsilon error. To achieve this, a Gibbs sampler with an annealing scheme is employed.

To mitigate computational complexity, it is suggested to synthesize the image in 8 gray levels [0,7] (don't forget to rescale back!) with a size of 64x64 pixels (128x128 is also okay, but it will take more time). The **torus boundary condition** addresses boundary effects. It is noteworthy that the primary focus should be on closely matching histograms at the tail bins. This involves assigning specific weights to the 15 bins, such as [8, 7, 6, 5, 4, 3, 2, 1, 2, 3, 4, 5, 6, 7, 8] (just an example!).

4.2 Instructions

4.2.1 Overview

The following is an overview procedure of the project:

1) Generate a filter bank by "filters.py."

2) The set of chosen filters S is initially empty.

3) Randomly sample a synthesized image I (sample the intensity of each pixel uniformly from [0,7]).

4) For each unselected filter, calculate its response histogram on both the input image and the currently synthesized image I. Ensure that the bin range of the histogram corresponds to the minimum and maximum response values of the input image. Subsequently, compute the weighted difference (error) between the two histograms.

5) If the highest error is smaller than a threshold, go to step 8. Otherwise, choose the filter with the highest error and add it to the chosen set S.

6) Use the Gibbs function to sample a new image I', and replace I with I'. (Hint: the input of the Gibbs function includes the synthesized image I from the last iteration, chosen filters set S, histograms of chosen filters' responses on the input image, and the corresponding bin ranges of the histograms.)

7) Compute the total error between the new I and the input image based on the new set S. This is the error you must show for the report. Then, go to step 4.

8) Output the synthesized image I as the final result.

4.2.2 Details

1. "filters.py" contains a Python function "get_filters()", which generates a set of filters in the format of matrices. **Guidance**:

   - You need to add more filters. For instance, you should at least add the Dirac delta function whose response is the intensity of a given pixel.

2. "julesz.py" contains a function "get_histogram()", which computes the normalized filter response histogram on an image. **Guidance:**

   - You should fill in the "conv()" function and the "get_histogram()" function.
   - Noticed that the "conv()" function returns the image's filtered response which holds the **same shape** as the input image. Therefore, you should consider carefully the padding method.
   - The "get_histogram()" function should be designed to allow the specification of the bin range, which corresponds to the maximum and minimum responses of the filter on the input image when computing the histogram. Note that the output histogram should be normalized by (image height) * (image width). Therefore, you need to multiply the normalized histograms by (image height) * (image width) to compute the difference (error) between two histograms.
   - Note that when calculating the histogram of each filtered response, you should use the same bin range for both the input image and the synthesized image.

3. **In this step, you can modify the "gibbs.py" file (including deleting all the codes and changing the file name).** We present two approaches for implementing the Gibbs sampling process: (1) Python Style (Roughly 3 hours per image with 64×64 size, Recommended) and (2) C Style (much faster). **Python Style (Recommended):** "gibbs.py" contains a function "gibbs_sample()", which implements a Gibbs sampling process for the image synthesis. **Guidance** :

   Python Style:

   (a) "gibbs_sample()" is a function that sweeps the image, and there is a function "pos_gibbs_sample_update()" which handles a single pixel of the image.

   (b) In "pos_gibbs_sample_update()", be careful while coding the process of calculating the conditional probability. (Hint: It's considerably faster to modify the specific bin responses in the existing histogram rather than performing convolution on the entire image and creating a new histogram from scratch.)

   (c) When computing the histograms of the input image and the synthesized image, you must use the same bin range. For this, you may compute the maximum and minimum responses of the response of a filter on the input image, and then divide the range equally to obtain the interval of each bin. Use these bin ranges for both input and synthesized images.

   (d) Assign different weights to the bins when you calculate the difference between two histograms, e.g. `abs(hists_syn[x] - hists_ori[x]) * w[x]`. Next, leverage the weighted difference to calculate the "energy" of the annealing process.

   (e) Tune the coefficient of the simulated annealing process to improve the synthesis.

   (f) Noticed that in the book, the selection of the location to be updated follows a uniform distribution. However, in this context, the locations may be chosen sequentially, such as following the order of `(0,0), (0,1), (0,2), ..., (0, img_width-1), (1,0), ...`.

   (g) If you have no idea, it is recommended to read section 3 of the paper "*Exploring Texture Ensembles by Efficient Markov Chain Monte Carlo- Towards a "Trichromacy" Theory of Texture*" by *Song-Chun Zhu, Xiu Wen Liu, Ying Nian Wu.*

(h) **Speed Up Tips**

i. If your implementation is done by using Torch, it'll be much faster to change to numpy.

ii. Use the **cython** library! Please first read the basic tutorial, and understand some basic logistics of **cython**. Please be cautious that the **"pyximport" method** is the **preferred approach** as opposed to the **"setup.py" method**. The latter may generate different files on different operating systems, and therefore please use the **"pyximport" method** . Now, you are almost ready to speed up your Gibbs sampling, please read the working with numpy section of the document. You have done a great job, and please speed up your Gibbs sampling process with Cython!

C Style: Sorry that we do not have a hint code for C-style. If you feel that Python is not enough you can try to study the **ctypes** library and implement the algorithm in C. It is a much harder way and you may get crazy with "segmentation fault," but you'll be much faster than Python and I believe you'll learn a lot!

Make sure that your code can be run with "python julesz.py" for autograding!

4. Fill in the "julesz()" method of "julesz.py", which should include the steps described in Section 4.2.1.

4.3 Submission

1. Show the input images of your program, and report their sizes.
2. Show the set of selected filters (enlarge them to make them visible) and print the sequence of images you synthesize by increasing the filter number.
3. Plot the errors of the histograms of filtered responses over the iterations (after choosing a new filter).

Hints for getting nice results:

1. Match the histogram closely, especially at the tails.
2. Try other filters not in the file.
3. You can debug by setting a small image size.
4. It's enough to generate good images of size 64×64 by using filters of size 3×3. However, for 128×128 input images, 3×3 filters aren't enough.
5. For the grass figure, 128×128 size will result in a more beautiful result, but we won't deduct your score for using 64×64.

5 Part 2: FRAME Model

Congratulations on finishing Part 1, you are marvelous!

5.1 Objective

This section examines the Filters, Random Field, and Maximum Entropy (FRAME) model, which constitutes a rigorously formulated mathematical framework designed to characterize textures by integrating filtering and random field modeling. In this model, Maximum Entropy is harnessed to establish the probability distribution governing the image space, while Minimum Entropy is employed for the judicious

selection of filters from a comprehensive filter bank. The amalgamation of these two facets gives rise to the overarching principle known as the "min-max entropy principle."

Revisiting the Principle of Maximum Entropy for constructing a probability distribution denoted as p over a set of random variables, denoted as X, involves the following considerations. Let $\phi_n(x)$ denote a set of known functions such that $E_p\left[\phi_n\left(x\right)\right] = \mu_n$. The collection of constraints to be satisfied is denoted as $\Omega = \{p\left(x\right) | E_p\left[\phi_n\left(x\right)\right] = \mu_n,\, n = 1, \ldots, N\}$.

The Principle of Maximum Entropy states that the choice of the probability distribution $p \in \Omega$, which maximizes the entropy $H(p)$, is a judicious one. As a consequence, this yields a solution in the form of $p\left(x;\Lambda\right) = \frac{1}{Z(\Lambda)} \exp\left\{-\sum_{n=1}^{N} \lambda_n \phi_n\left(x\right)\right\}$, where Λ represents the Lagrange multipliers $\{\lambda_1, \lambda_2, \ldots, \lambda_N\}$, and the partition function $Z\left(\Lambda\right)$ is defined as $Z\left(\Lambda\right) = \int \exp\left\{-\sum_{n=1}^{N} \lambda_n \phi_n\left(x\right)\right\} dx$.

From this foundational principle, the Filters, Random Fields, and Maximum Entropy (FRAME) model can be derived (see "Computer Vision – Statistical Models for Marr's Paradigm" or "Filters, Random Fields and Maximum Entropy (FRAME): Towards a Unified Theory for Texture Modeling").

5.2 Instructions

To learn the FRAME model, you must select filters and learn Λ. Note that in this part, only the **fur image of size** 64×64 is needed.

1. Read the relevant chapter in the textbook.
2. Create a copy of your codes in part 1 ("filters.py" to "filters_frame.py", "gibbs.pyx" to "gibbs_frame.pyx").
3. Adjust your Gibbs sampling code to compute the term $-\sum_n \lambda_n \phi_n(x)$.
4. Create a copy "frame.py" of "julesz.py". Adjust the code to define Λ and update $\{\lambda_1, \lambda_2, \ldots, \lambda_N\}$ by gradient ascend on $\log p(x;\Lambda)$, i.e.,

$$\frac{d\lambda_n}{dt} = \frac{d \log p(x;\Lambda)}{d\lambda_n} = E_{p(x;\Lambda)}\left[\phi_n\left(x\right)\right] - \mu_n, n = 1, 2, \ldots, N.$$

5. Deprecate the "weight" item for calculating the difference (error) of 2 histograms in part 1, as it's not needed for part 2.
6. Set the threshold of the algorithm to a reasonable scale so that the program will not take too long. As a reference, the threshold of 0.1 will take roughly 7 hours.
7. Change the "julesz()" function to "frame()" and adjust it according to the book.
8. Make sure that your code can be run by "python frame.py" for autograding!

5.3 Submission

1. Plot the estimated $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_N\}$ over learning iterations.
2. Show the set of selected filters and print the sequence of images that you synthesize by increasing the filter number.

Hints If the implementation is deemed accurate, you are advised to wait several procedural steps in order to observe the efficacy of the FRAME model.