Markov Image Model on Reduced Color Spaces

YuXuan Liu **Roy Tu** **Nikhil Unni**

EE 126 - Spring 2016

1 Introduction

The Markov Model for text generation has been used in a variety of "parody generator" software, spam generation, and even DNA simulation. Motivated by this, we sought to apply the Markov Model on image generation as well. First we use dimensionality reduction on the color state space for computational tractability and efficient use of a limited data set. From there, we generated linear images by transitioning from pixel to pixel according to the Markov Model. Then we generated 2-dimensional images by conditioning on neighboring pixels. Finally we present a spiral generation algorithm to make the best use of neighboring pixels. We also explore the long term behavior of any image evolving under the Markov Model. To conclude, we point to applications such as texture generation and content-aware filling.

2 Color Space Reduction

2.1 Motivation

Most color spaces consist of three 8-bit values for red, green, and blue (RGB). This in turn corresponds to $(2^8)^3 = 16777216$ possible different colors. If we were to model a Markov chain to this color space, we would require a 16777216×16777216 matrix of transitions. Moreover because of the limited training dataset for any set of images, this transition matrix will be extremely sparse with a mostly deterministic distribution (i.e. $P_{ij} = 1$ for a significant amount of colors).

2.2 Methods

A cursory observation leads us to see that bit reduction is a simple way to reduce dimensionality. For any RGB value, we can reduce the 8-bit value to b-bits by ignore lower order bits. Thus for any b-bit RGB value V_b, we have

$$V_b = \lfloor \frac{V_8}{2^{8-b}} \rfloor 2^{8-b}$$

2.3 Experiments

We experimented with different values of b and found that $b = 4$ worked empirically well. It offered us a color space of size 4096 while maintaining reasonable quality. When $b = 3$, quality dropped dramatically

2.4 Results

Original Image:

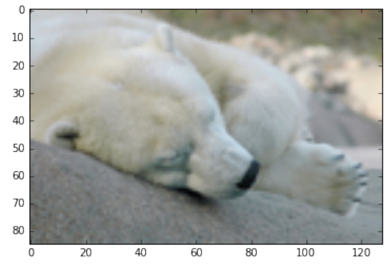


Image when $b = 4$:

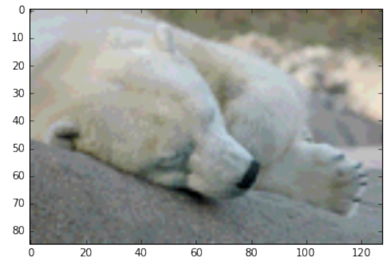
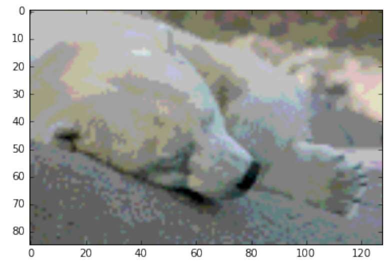


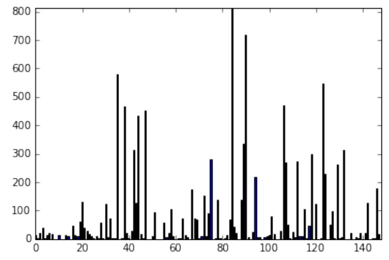
Image when $b = 3$:



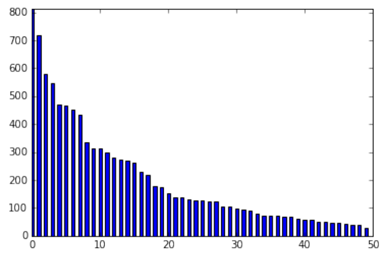
2.5 Analysis

After plotting a histogram of our 4-bit color space, we noticed that certain colors had a disproportionate frequency.

Histogram of the color space:



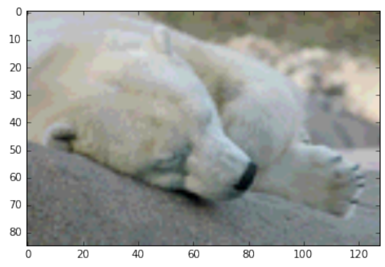
Histogram of the top 50 colors:



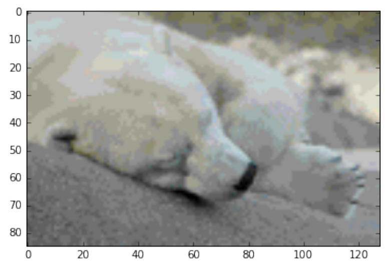
Which leads us to another color space reduction algorithm: Start with the most frequent colors, and for each color C, let M be the closest color in the colorspace CS. If $||M - C|| < LIM$ where LIM is some threshold, map C to M in CS. Otherwise, if we don't have N colors in CS yet, map C to C in CS. If there are already N colors, map C to the closest color M.

```
REDUCE(N, LIM):
  Let CS = {} empty color space
  H = Sorted histogram of colors
  for COLOR in H:
    Let M = argmin(||COLOR-C||, for C in CS)
    if ||COLOR-M|| < LIM or |CS| > N:
      CS[COLOR] = M
    else:
      CS[COLOR] = COLOR
```

Original 4-bit image:



Reduced image (N=50, LIM=1):



The reduced image and the original are almost indistinguishable. However we've reduced the size of the color space from 4096 to 50.

3 Markov Generation

3.1 Method

First we use images from ImageNET, a large dataset for computer vision systems. For every image, we consider each pixel and its 8 neighbors. For any colors i, j in the color space

$$C_{ij} = \sum_{p \in \text{pixels}} \sum_{k=1}^{8} \mathbb{1}\{N_{pk} = j\}$$

Where N_{pk} is the k'th neighbor of any pixel p and C_{ij} is the total number of transitions from color i to color j in all pixels from all images.

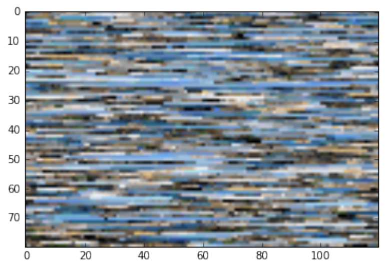
Thus the transition probability for any i, j is:

$$P_{ij} = \frac{C_{ij}}{\sum_{k=1}^{n} C_{ik}}$$

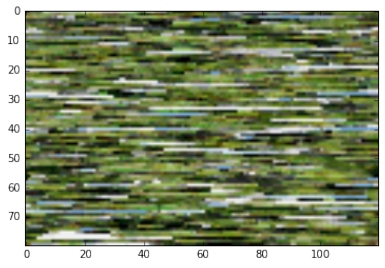
To generate images, we start by calculating transition probabilities on an ImageNET data set. Once we have the transition matrix, we choose some arbitrary starting state S_0 and sampling from the outgoing transitions from S_0. Each pixel is generated sequentially and converted into an image.

3.2 Results

Generated image from the beach dataset:



Generated image from the pasture dataset:



3.3 Analysis

The generated images appear to be very "linear" because of the Markov property. Namely, every pixel is dependent only on the pixel before it. This motivates another definition of transition probabilities conditioned on multiple pixels. If S is a set of starting pixels, then let

$$D_{Sj} = \sum_{s \in S} C_{sj}$$

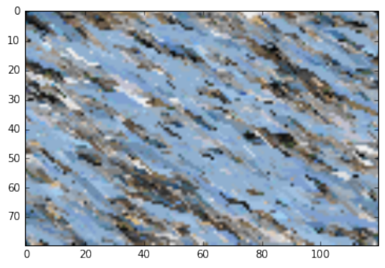
with transition probability

$$P_{Sj} = \frac{D_{Sj}}{\sum_{k=1}^{n} D_{Sk}}$$

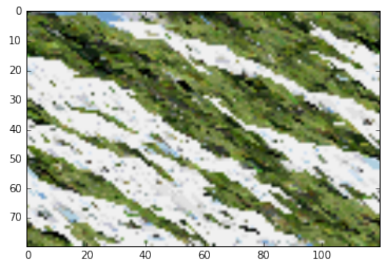
Now we start with one starting pixel in the upper left corner. Generate neighboring pixels by conditioning on $S_{ij} = \{p_{i-1,j}, p_{i,j-1}\}$ where p_{ij} is the i, j pixel.

3.4 Results

Generated image from the beach dataset:



Generated image from the pasture dataset:



Overall the images look much more cohesive due to the dependency on adjacent pixels.

4 Spiral Generation

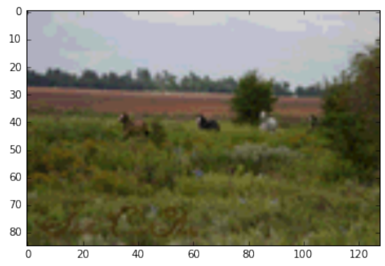
4.1 Method

As we've just seen, conditioning on neighboring pixels generated more cohesive images. Motivated by this, spiral generation conditions on the prior pixels in a spiral pattern around an initial bounding box. At each point, every pixel is conditioned on exactly 4 neighboring pixels.

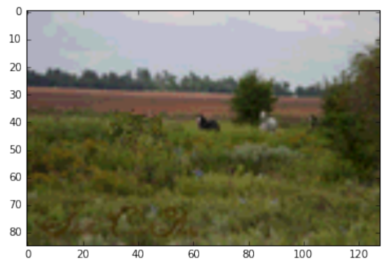
```
SPIRAL(rmin, rmax, ymin, ymax, img):
  WHILE rmin < rmax AND ymin < ymax:
    for c in [cmin, cmax]:
      S = {img[rmin-1, c], img[rmin-1, c-1],
           img[rmin-1, c+1], img[rmin, c-1]}
      img[rmin, c] = sample from P_S
      S = {img[rmax+1, c], img[rmax+1, c-1],
           img[rmax+1, c+1], img[rmax, c-1]}
      img[rmax, c] = sample from P_S
    for r in [rmin, rmax]:
      S = {img[r, cmin-1], img[r+1, cmin-1],
           img[r-1, cmin-1], img[r-1, cmin]}
      img[r, cmin] = sample from P_S
      S = {img[r, cmax+1], img[r+1, cmax+1],
           img[r-1, cmax+1], img[r-1, cmax]}
      img[r, cmax] = sample from P_S
    rmin++
    rmax--
    cmin++
    cmax--
```

4.2 Results

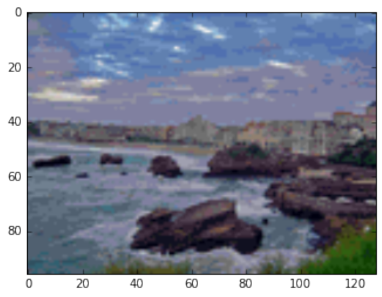
Original image:



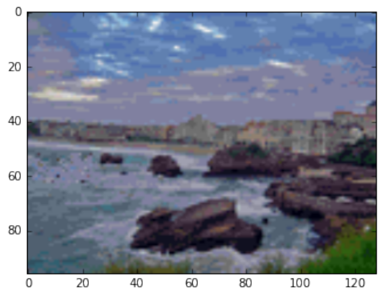
Spiral Generation on horse:
(rmin=36, rmax=50, cmin=30, cmax=50)



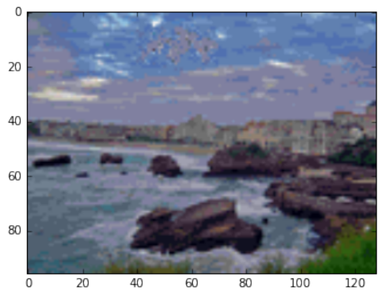
Original image:



Spiral Generation on rock:
(rmin=52, rmax=63, cmin=2, cmax=24)



Spiral Generation on sky:
(rmin=5, rmax=20, cmin=40, cmax=70)



5 GIF Generation

5.1 Motivation

Further exploring the same idea of generating images using Markov properties, we thought it would be interesting to see how an image evolves with time. At each time frame, each pixel independently transitions to another pixel value, based off of our Markov transition matrix. The result is a set of images at each time step which we can string together to make an animated GIF, which illustrates how the image transforms with time.

5.2 Method

The state space of our Markov chain is the set of unique (R,G,B) values from the original image. We use the same scheme for calculating transition probabilities as the Markov Generation section. In general, most images converge to noise. To study the convergence, we plot the entropy with respect to time. Since the images converge to mostly noise, we expect the entropy of the image to converge. Concretely, we measure the entropy of a pixel by the Shannon entropy of a 10x10 region around the pixel x:

$$S(x) = -\Sigma_{i \in N(x)} p_i log_2(p_i)$$

Where $N(x)$ is the region around the pixel, and p_i is the probability of observing the pixel within the neighborhood, which is determined by counting its frequency in the neighborhood.

5.3 Results

We measure the entropy of an image as the summation over the individual pixel entropies:

$$S = \Sigma_{x \in \text{pixels}} S(x)$$

Plotting the entropy of our GIF with respect to time, we found images converge after roughly 50 frames. And looking at Figure 3, we can see a heatmap of each pixel's entropy at T=0 and T=99.

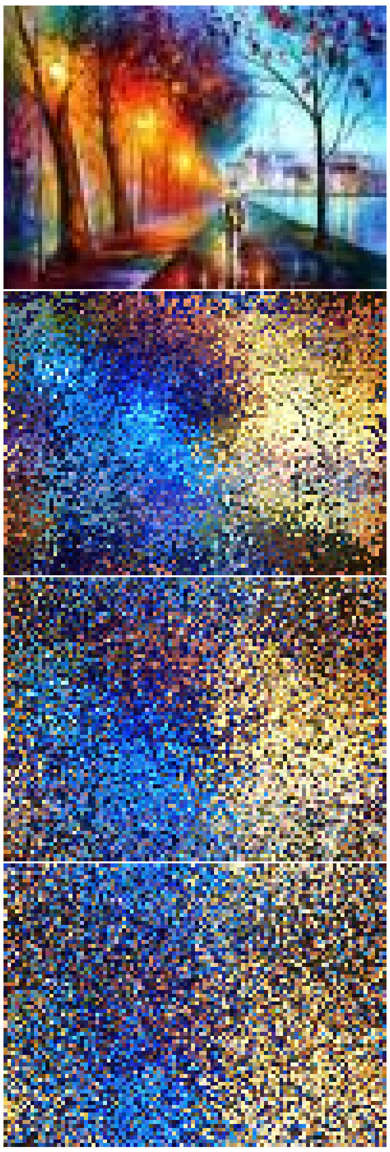


Figure 1: Sample image at times T=0, T=10, T=50, T=99

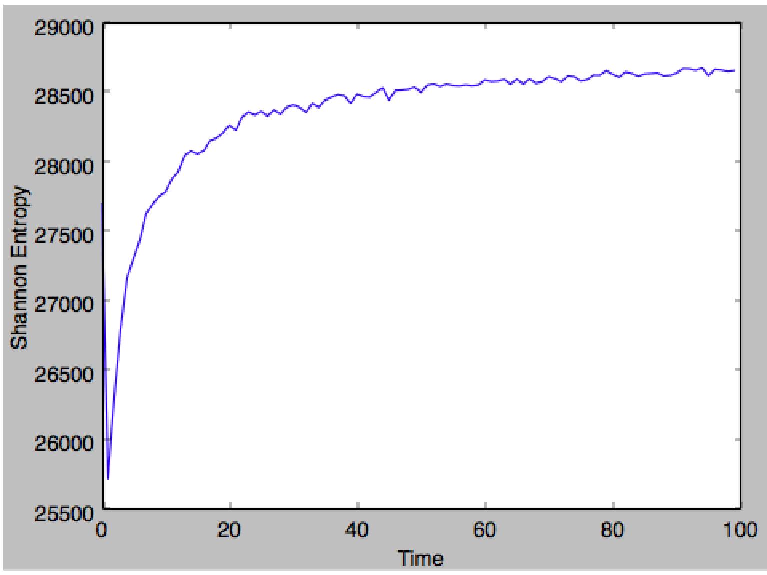


Figure 2: Entropy of painting image with respect to time

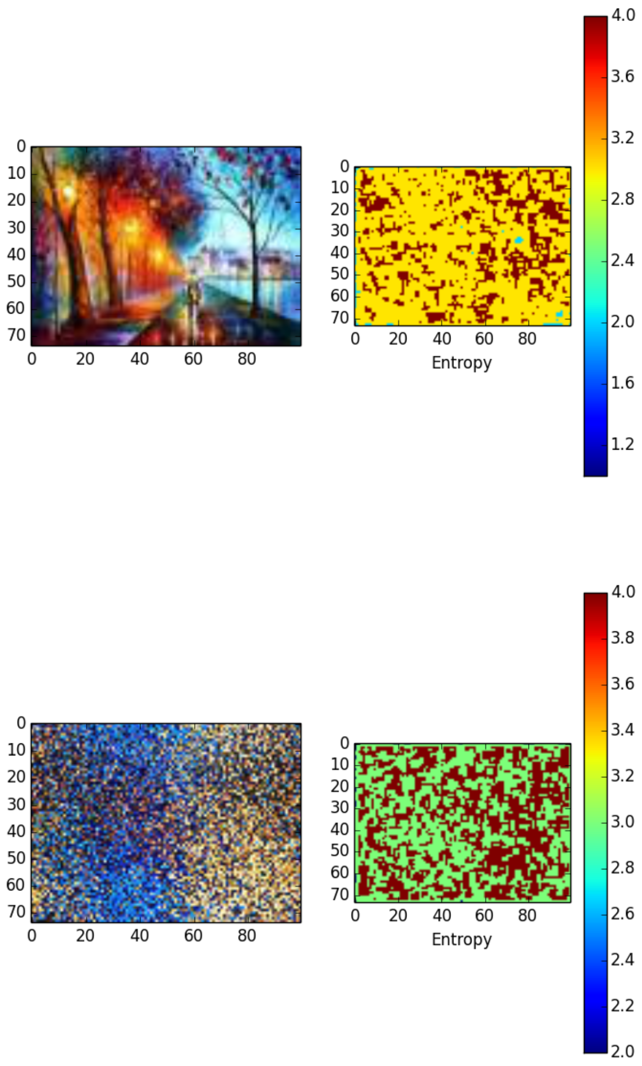


Figure 3: Heatmap of Entropy at T=0, T=99

6 Conclusions

6.1 Application

Markov Image Generation can have some notable uses. In particular, spiral generation can be used as a context-aware fill. In popular programs such as Photoshop, a context-aware brush can be used to remove blemishes or smooth out areas of an image. The Markov Image Model is also particularly good at generating textures which usually have a general homogeneity with hints of randomness.

6.2 Limitations

The Markov Model has limitations because of its memory-less property. Even by conditioning on all neighboring pixels, newly generated pixels only have a "nearsighted" view of the image. Therefore, its very difficult to generate complex image structures beyond simple textures, since encoding states of d pixels grows by $O(n^d)$, where n is the size of the color space. Any system based around the Markov model must dramatically limit the color space in order to be computationally tractable.

6.3 Future Areas of Study

In the future, we hope to use blocks of pixels at a time. By replacing the color space with a $m \times n$ pixel block space, we can capture more of the image structure within the Markov Model. However this also increases the complexity of our state space and will require more training data.

Also worth considering is augmenting our Markov Model with another data structure. For instance, since Markov models are sufficiently powerful to encode local relationships between pixels, one can imagine training a quadtree structure over an image, where the leaves of the quadtree are local Markov models. While a scheme like this may require some thought to construct, the general strategy is clear: augment a structure capable of handling higher-level image features with Markov models trained on patches of the image.

A final thought on an application out of the scope of this paper is whether Markov models trained on textures can be used as classifiers. Imagine models trained over several homogenous texture images – brick, dirt, grass, etc. One can imagine a system where a candidate patch of a test image is assigned a label based on what texture it corresponds to. A naive strategy would be to first get a set of labeled textures and train Markov models over them, and then use each model to expand the patch of the test image, using a distance metric to penalize models that greatly alter the test image. The model that alters the test image the least would most likely correspond to the correct texture for that patch.