CS 280 Computer Vision
Spring 2018 Efros Lecture 24

1 Lecture

Non-Parametric Visual Synthesis

> "Steal as much as possible from visual data that already exists, and then cover up the evidence that it was stolen."

One motivation comes from Shannon's 1948 approach to text synthesis: use a large text to compute probabilities of a letter/word given $N-1$ previous letters/words, turn it into a Markov chain, and (starting from a seed) repeatedly sample the chain in order to generate new text.

A visual application in a similar vein: *video textures.*

- If you have a distance and you want to turn it into a probability, you can just fake it: compute e^{-x}, where x is your distance, and get a value between 0 and 1 that you can call a probability.
- Video textures can be extended to hyperlapses; choose *spaced-out* frames which make good transitions.

Other applications: *texture synthesis, texture transfer, image analogies.* Transition in space, not time.

Synthesis with Deep Models

In one parametric method for texture synthesis, we have a manifold of the proper texture (all images that look like samples from a texture). We then come up with a mapping between the image space and some model space, such that the model captures that manifold of textures.

- Map the original texture from image space to model space.
- Map some random image to model space, presumably to a different place than that of the texture.
- Push the representation of the random image closer to the representation of the original texture, such that the model responses are similar. Effectively, this projects the random image onto something close to the natural texture manifold.

For this, we want to use strong models such as neural networks (in previous times, the basic filter bank response models were too weak for the approach to work well). We can take the features from all the layers in the network and treat them as an enormous filter bank.

- An idea: capture pairwise (second-order) statistics of these features. String the features from each layer into vectors and put them next to each other as rows in a feature matrix $F = \left[f_1, \ldots, f_N\right]^T$.
- In this case, the Gram matrix is the outer product $G = FF^T$ and captures all of the pairwise feature correlations without regard for spatial structure.
- If we tune the model such that Gram matrices of random images resemble Gram matrices of texture images, we'll end up with something that produces outputs close to the manifold of natural textures.

Pixels as Output

We've seen how we can use neural network features to create nice-looking images. How about just using neural networks as a synthesizing tool? Traditionally, neural networks funnel images from their high-dimensional

representation into a low-dimensional label (e.g. a classification label). But maybe we want the output to be high-dimensional itself; maybe we want raw, unlabeled pixels as output.

Colorization

For a task like *colorization*, we could naively split images into luma and chroma components and train the network to predict chroma from luma. Unfortunately, the results aren't vibrant; we end up with a lot of sepia tones.

This is because there are multiple modes for the correct answer; birds can be blue or they can be red. And say half of our data is blue birds, and half of our data is red birds. If our loss is something like L2, the optimizer will try to make both of the modes happy – i.e. split the difference, go to the middle. And in color space, the middle is the grays. So L2 pushes us toward averages, even if our distribution is bimodal!

One way to deal with this problem is to tessellate our color space into classes, and predict color classes for each pixel. This is better (e.g. in the case of the blue and red birds) because it will learn to put a lot of weight on blue, a lot of weight on red, and not a lot of weight in the middle. If it's a classification task, there's no notion of a "middle."

Super-Resolution

Another task: *super-resolution*. We can train on pairs of low-resolution images and their high-resolution counterparts, and predict the high-resolution image from the low-resolution one. If we minimize L2 distance between predicted pixel color and ground truth color, we'll run into a similar problem: multimodality. There are many ways to hallucinate a high-resolution image, and L2 takes an average of all of them. But what we really want to do is pick one and go with it.

So instead of doing L2 on the pixels, Johnson et al. did L2 on the features (similar to the deep texture synthesis/style transfer approach). The features are kind of like classes, so this is kind of like classification. Match covariance statistics of feature activations; distances are now in Gram matrix space!

GANs

Though the results are decent, these losses for colorization and super-resolution are somewhat ad hoc. It would be nice to have something more universal. We want a loss function that applies to all of these problems and makes realistic-looking images.

So GANs. Generate images, and use as a *universal loss* the output of a classifier which distinguishes "ground truth" from "synthetic." Jointly optimize the generator and the discriminator.

The discriminator D's loss is

$$\arg\max_D \mathbb{E}_{x,y}[\underbrace{\log D(x, G(x))}_{\text{high for fakes}} + \underbrace{\log(1 - D(x, y))}_{\text{low for real images}}]$$

x: input image
y: ground truth output image
$G(x)$: generated output image

The generator G's loss is

$$\arg\min_G \max_D \mathbb{E}_{x,y}[\log D(x, G(x)) + \log(1 - D(x, y))]$$

(G is trying to synthesize fake images that fool the *best* D, hence the max.) G gets a backpropagated signal from D, meaning it can learn how D is spotting fakes and adjust accordingly.

D is like a learned loss function.