Generative Adversarial Networks

Kris Sankaran

Nepal Winter School in AI

December 19, 2018

Learning Objectives

- Understand basic adversarial setup
- Mathematical perspectives to help navigate sea of proposed variants
- Get a flavor of some real-world applications

First thing that comes to mind?

- GANs are popular ways to generate images

Figure: Probably the most famous GAN samples of all time. (**?**)

Or maybe this?

- GANs are popular ways to generate images

Figure: Probably the most expensive GAN sample of all time.

GANs as Sampling Algorithms

- Ignore the hype
- It's better to think of GANs as learning how to sample from probability densities $p(x)$
  - For 32×32 images, $p(x)$ is a distribution in 1024-dimensional space
- What's exciting is that they are *implicit* probability models
  - Distribution defined implicitly by sampling mechanism

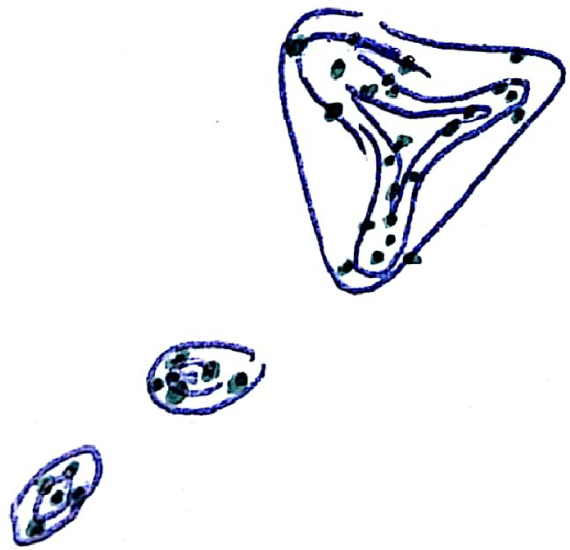


Figure: We may not be able to write simple formulas for complicated densities, but we may be able to write computer programs that sample from them.

Images and Probability

- It's common in the GANs literature to show interpolations between generated images

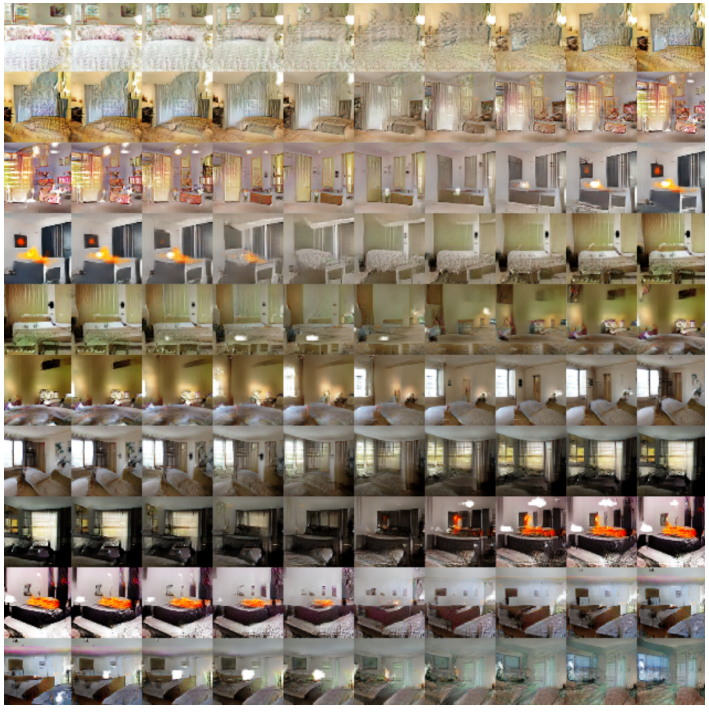


Figure: Example interpolations between LSUN images in (**?**). This was a convincing result at the time, because it showed that the samples weren't simply memorized training data.

Images and Probability

- Geometrically, these interpolated images are points along a curve in image space

Figure: The green points would each be an image, and the red line represents a sequence of gradually evolving samples, which should look like real images, if they live in a region of high probability.

Formulation

- How to transform (say, uniform) noise z into arbitrary distributions?
- Idea: Use a neural network G
- $z \to G(z)$

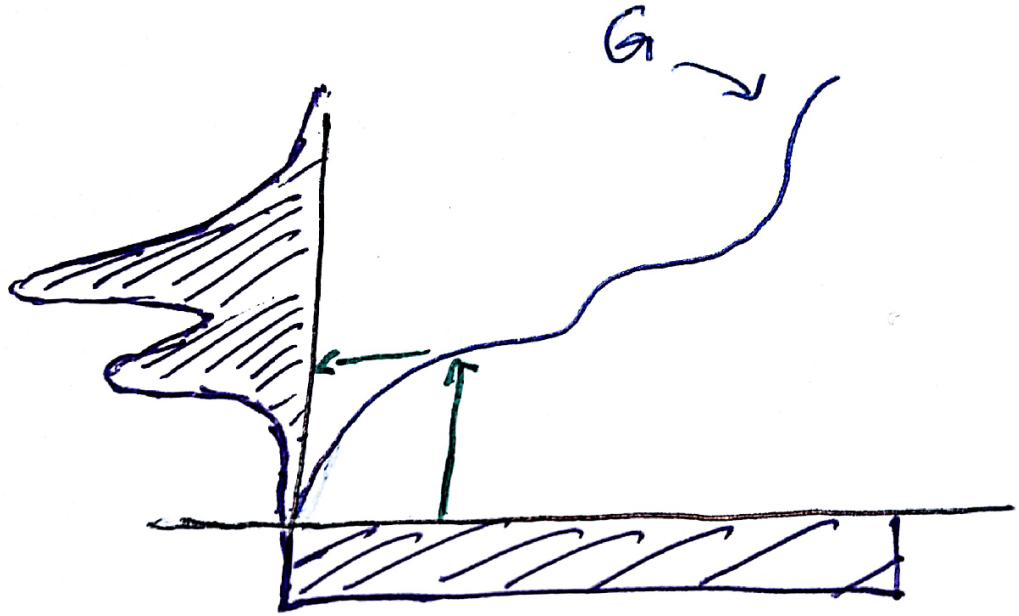


Figure: By passing samples from a probability distribution through a function, you get another probability distribution. y-values where the function is flatter will get more of the original mass, resulting in peaks in the transformed distribution.

Formulation

- How to learn a transformation to match observed data?
- Idea: Use another neural network! Call it D.
- This is pretty different from usual likelihood-based views
  - (there are connections though)

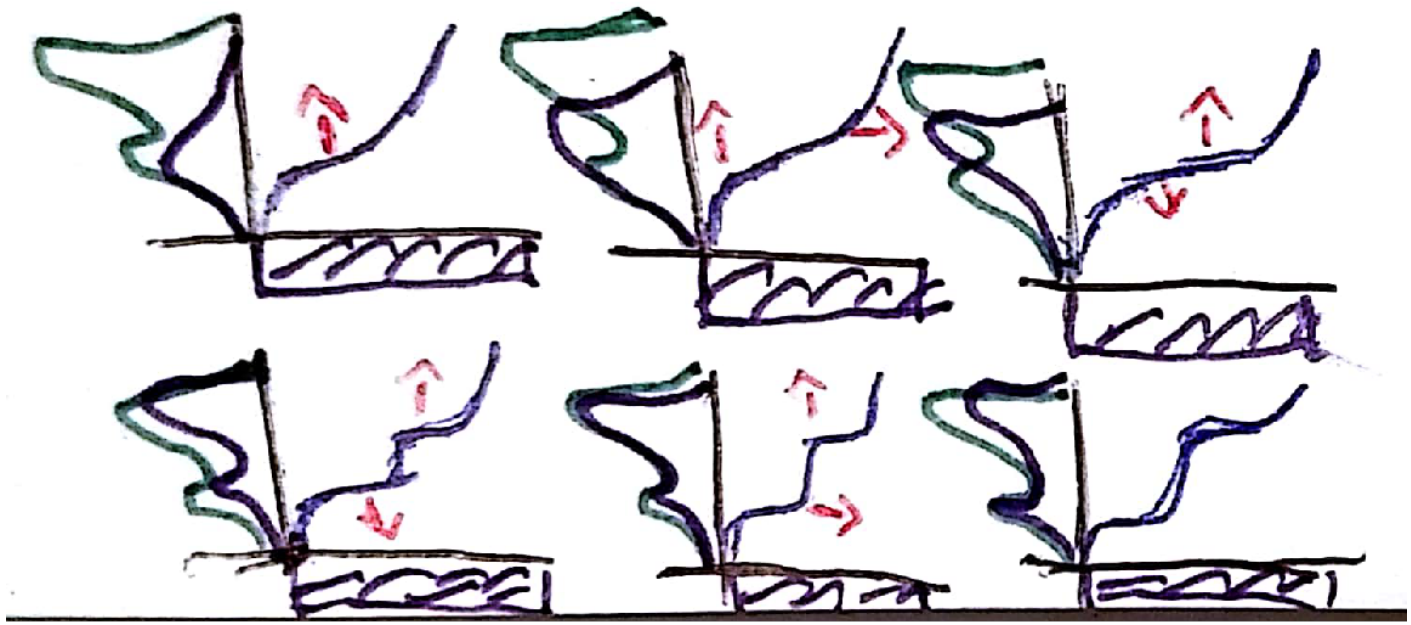


Figure: The discriminator guides the generator towards learning a good transformation, attempting to tell the difference between the true data (green) and generated (blue) distributions.

Objective Function

- How should D guide updates to G?
- Objective function

$$\min_G \max_D V(D, G) = \mathbb{E}_{p_{\text{data}}}[D(x)] + \mathbb{E}_{p(z)}[\log(1 - D(G(z)))]$$

- Alternate gradient steps, first update D, then G, then D, ...
  - D tries to be large on x and small on $G(z)$
  - G is encouraged to make D assign values near 1 to samples $G(z)$, so that the objective gets small

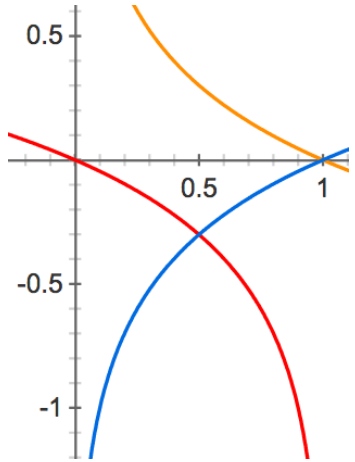


Figure: The blue, red, and orange lines are $\log x$, $\log(1 - x)$ and $-\log x$, respectively. This last function is often used instead of $\log(1 - x)$.

GAN Lab

https://poloclub.github.io/ganlab/

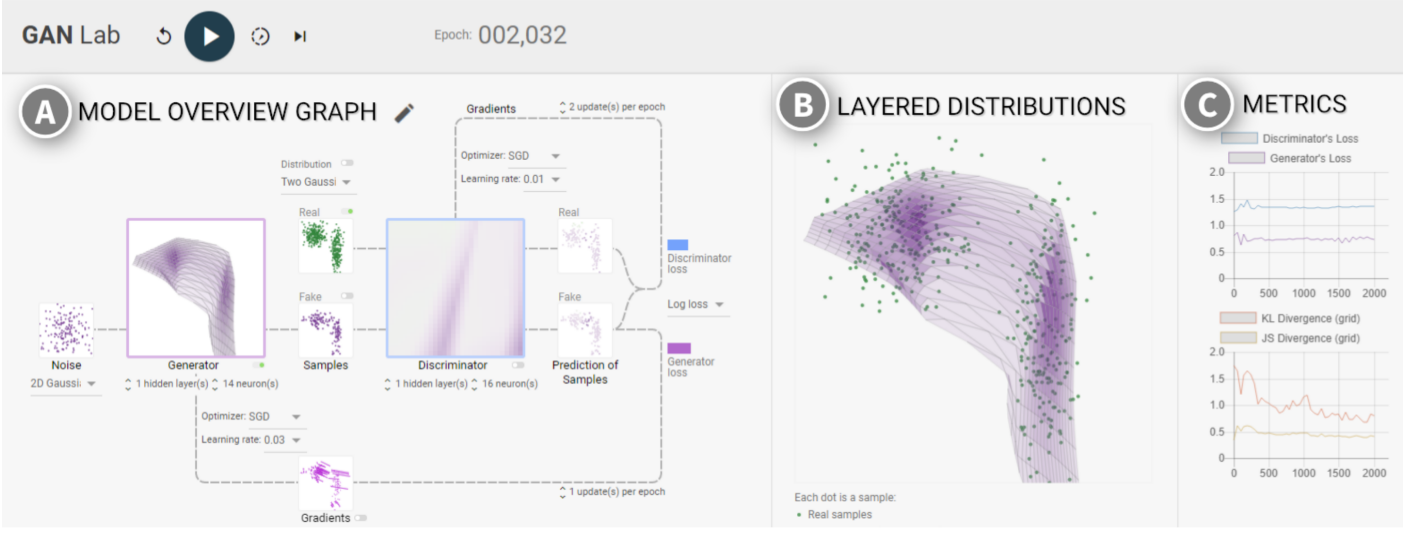


Figure: A visualization of the whole process, on a few different toy datasets.

Training difficulties

- GANs have a bad reputation for being finnicky to train
- If the discriminator D is "too strong," it might not communicate useful information

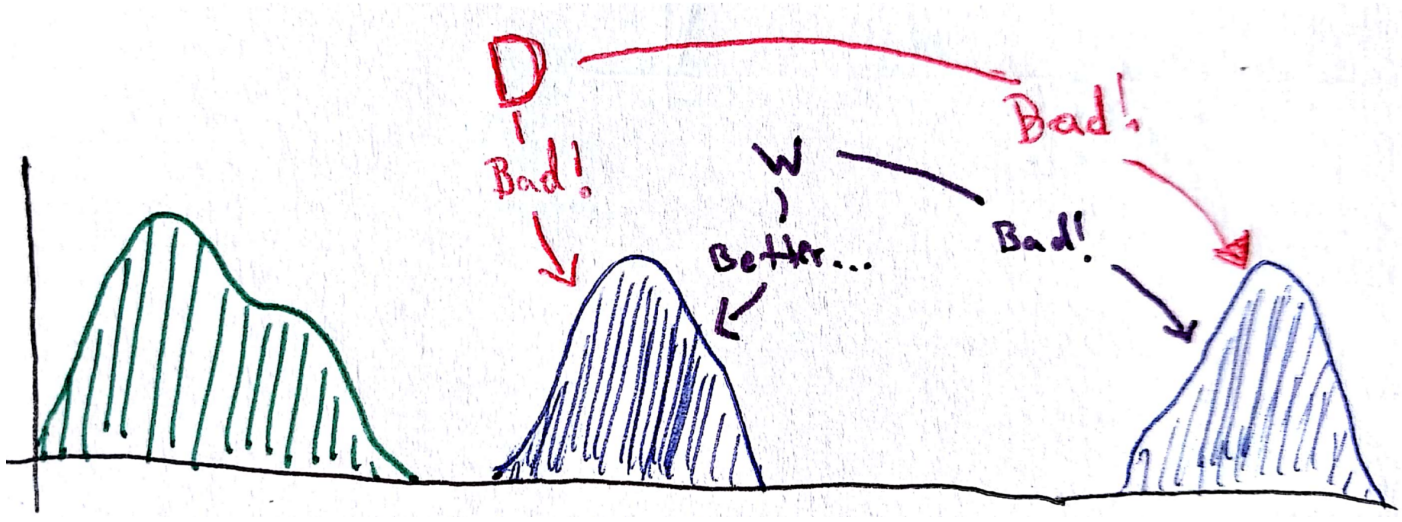


Figure: We would like a critic W that tells the generator when it's getting closer, even when it's obviously different from the target.

Wasserstein Distance

- Formalize this idea using distances $d(\mathbb{P}_1, \mathbb{P}_2)$ between probability distributions
- KL-divergence compares densities *vertically*
- Wasserstein compares densities *horizontally*
- Wasserstein distance tells us something even when they are far away!

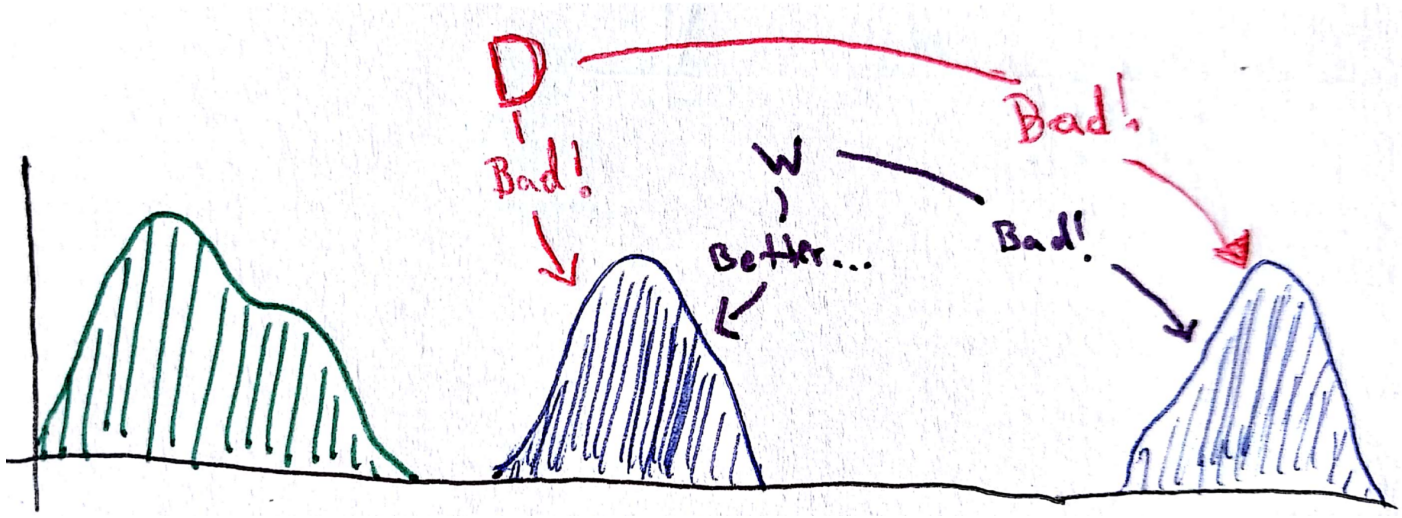


Figure: We would like a critic W that tells the generator when it's getting closer, even when it's obviously different from the target.

Wasserstein Objective

Wasserstein distance between data X and generated $\tilde{X}$ distributions is defined by

$$d\left(\mathbb{P}_X, \mathbb{P}_{\tilde{X}}\right) = \max_{\|f\|_L \leq 1} \mathbb{E}_X\left[f\left(x\right)\right] - \mathbb{E}_{\tilde{X}}\left[f\left(\tilde{x}\right)\right]$$

where $\|f\|_L \leq 1$ is the set of functions whose slopes are always less than 1.

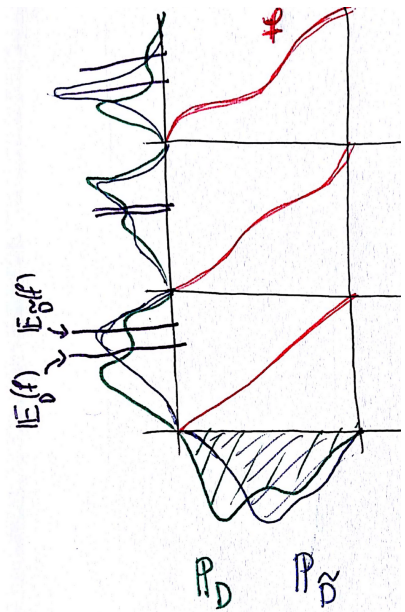


Figure: You can interpret the objective as computing differences in means after passing samples through a variety of (not too wild) test functions.

Wasserstein Training

- In terms of the transformed noise $G(z)$, this looks like

$$d\left(\mathbb{P}_X, \mathbb{P}_{\tilde{X}}\right) = \max_{\|f\| \leq 1} \mathbb{E}_X\left[f(x)\right] - \mathbb{E}_Z\left[f\left(G(z)\right)\right]$$

- Finding the optimal f for any particular choice of G is complicated...
- As an alternative, do gradient updates
  - Critic f inches towards its maximizing value of $d\left(\mathbb{P}_X, \mathbb{P}_{\tilde{X}}\right)$.
  - Generator G descends towards smaller wasserstein distances
  - Constraint $\|f\|_L \leq 1$ maintained either by clipping or regularization

Implicit Learning

- Alternative perspective: GANs as likelihood-free inference / implicit learning
- Remember generative mechanism,

$$z \sim q(z)$$
$$x|z := G_\theta(z)$$

- Marginal density $q_\theta(x)$ of data generated by this process is not analytically available

Estimating Density Ratios

- Let $p^*(x)$ be the true data generating density
- Let y be an indicator of true vs. simulated data
  - $p(x|y=1) = p^*(x)$
  - $p(x|y=0) = q_\theta(x)$
- We can't evaluate either p^* or q_θ directly, but by Bayes' rule,

$$\begin{aligned}\frac{p^*(x)}{q_\theta(x)} &= \frac{p(y=1|x)\,p(x)}{p(y=1)}\,\frac{p(y=0)}{p(y=0|x)\,p(x)} \\ &= \frac{p(y=1|x)}{p(y=0|x)}\,\frac{1-\pi}{\pi}\end{aligned}$$

  where $\pi = p(y=1)$.
- We just need a classifier $D_\varphi(x)$ to tell difference between truth and simulated

Learning

- Lots of options: just specify what you are trying to do (ratio matching, density difference) and how you are going to get there (cross-entropy, f-divergence, ...)
- For example, if you try estimating density ratios using Bernoulli loss,

$$\begin{aligned}
&\mathbb{E}_{p(x|y)p(y)}\left[y \log D_\varphi(x) + (1-y)\log\left(1 - D_\varphi(x)\right)\right] \\
=&\pi \mathbb{E}_{p(x|y=1)}\left[\log D_\varphi(x)\right] + (1-\pi)\,\mathbb{E}_{p(x|y=0)}\left[\log\left(1 - D_\varphi(x)\right)\right] \\
=&\pi \mathbb{E}_{p^*(x)}\left[\log D_\varphi(x)\right] + (1-\pi)\,\mathbb{E}_{q_\theta(x)}\left[\log\left(1 - D_\varphi(x)\right)\right]
\end{aligned}$$

which is exactly the usual loss for GANs, if we set $\pi = \frac{1}{2}$.

Applications: Image-to-Image translation

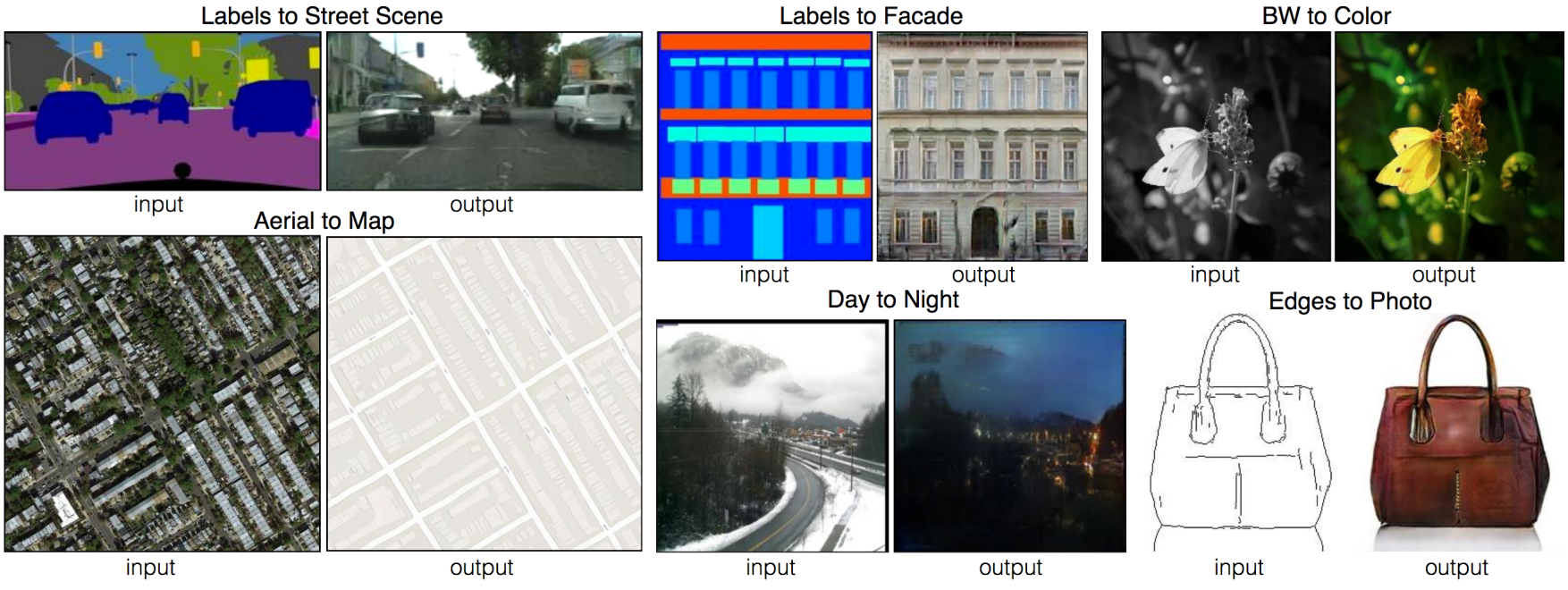
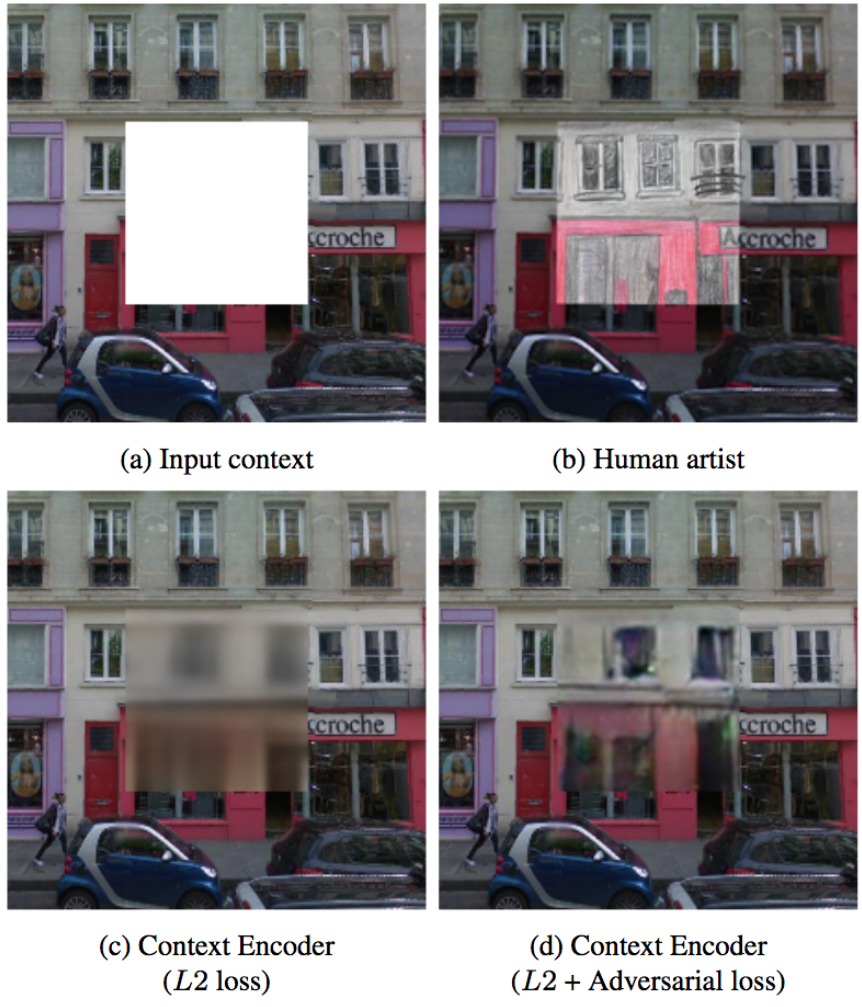


Figure: Adding on a GAN objective when learning to map one class of images to another results in realistic looking translated images.

Applications: Inpainting



(a) Input context (b) Human artist

(c) Context Encoder ($L2$ loss) (d) Context Encoder ($L2$ + Adversarial loss)

Figure: Similarly, when applied to inpainting, the GAN loss leads to much less blurry images than the traditional ℓ^2 loss.

Applications: Super-Resolution

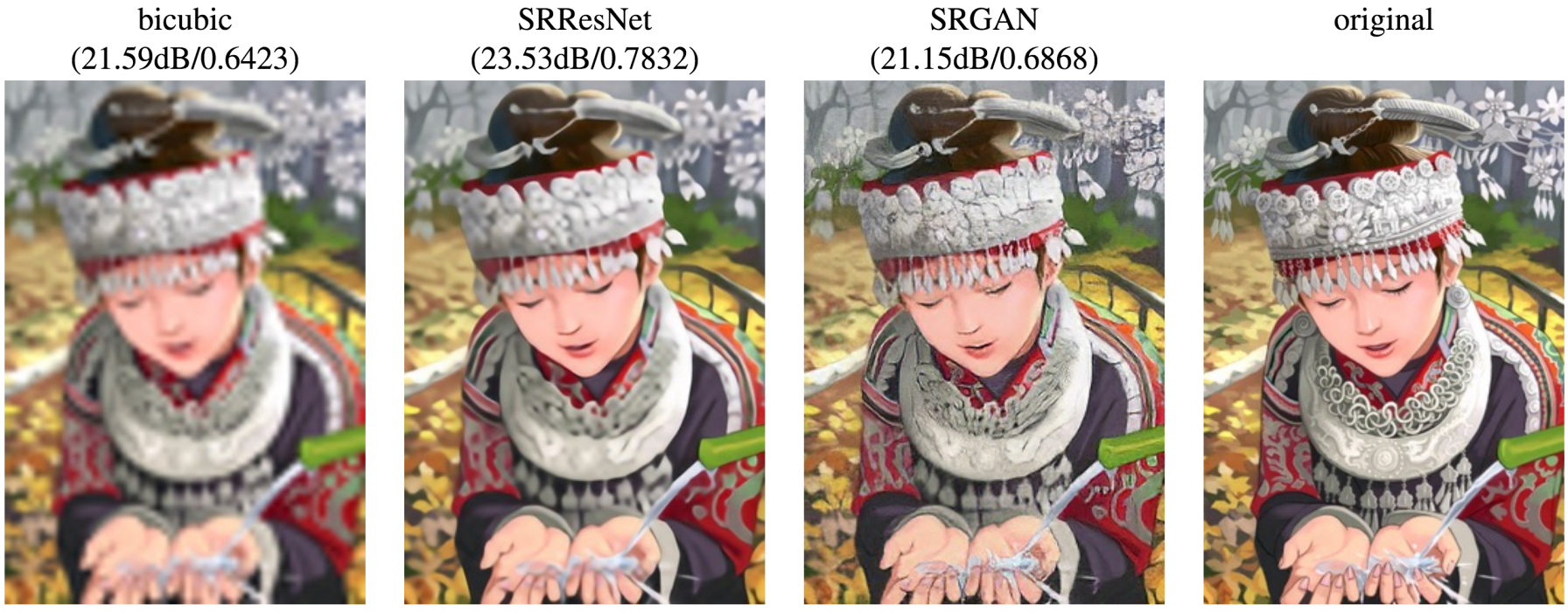


Figure: It's interesting to note that some of the details that appear in the GAN super-resolved image were not present in the original high-resolution version.

Conclusion

- GANs are generic unsupervised learning procedure that don't require explicit probability densities
- They learn to transform simple distributions into those that resemble training data
- They can be used in many computer vision problems