goes into designing effective losses. In other words, we still have to tell the CNN what we wish it to minimize. But, just like King Midas, we must be careful what we wish for! If we take a naive approach and ask the CNN to minimize the Euclidean distance between predicted and ground truth pixels, it will tend to produce blurry results [43, 62]. This is because Euclidean distance is minimized by averaging all plausible outputs, which causes blurring. Coming up with loss functions that force the CNN to do what we really want – e.g., output sharp, realistic images – is an open problem and generally requires expert knowledge.

It would be highly desirable if we could instead specify only a high-level goal, like "make the output indistinguishable from reality", and then automatically learn a loss function appropriate for satisfying this goal. Fortunately, this is exactly what is done by the recently proposed Generative Adversarial Networks (GANs) [24, 13, 44, 52, 63]. GANs learn a loss that tries to classify if the output image is real or fake, while simultaneously training a generative model to minimize this loss. Blurry images will not be tolerated since they look obviously fake. Because GANs learn a loss that adapts to the data, they can be applied to a multitude of tasks that traditionally would require very different kinds of loss functions.

In this paper, we explore GANs in the conditional setting. Just as GANs learn a generative model of data, conditional GANs (cGANs) learn a conditional generative model [24]. This makes cGANs suitable for image-to-image translation tasks, where we condition on an input image and generate a corresponding output image.

GANs have been vigorously studied in the last two years and many of the techniques we explore in this paper have been previously proposed. Nonetheless, earlier papers have focused on specific applications, and it has remained unclear how effective image-conditional GANs can be as a general-purpose solution for image-to-image translation. Our primary contribution is to demonstrate that on a wide variety of problems, conditional GANs produce reasonable results. Our second contribution is to present a simple framework sufficient to achieve good results, and to analyze the effects of several important architectural choices. Code is available at `https://github.com/phillipi/pix2pix`.

2. Related work

Structured losses for image modeling Image-to-image translation problems are often formulated as per-pixel classification or regression (e.g., [39, 58, 28, 35, 62]). These formulations treat the output space as "unstructured" in the sense that each output pixel is considered conditionally independent from all others given the input image. Conditional GANs instead learn a *structured loss*. Structured losses penalize the joint configuration of the output. A large body of literature has considered losses of this kind, with methods including conditional random fields [10], the SSIM metric [56], feature matching [15], nonparametric losses [37], the convolutional pseudo-prior [57], and losses based on matching covariance statistics [30]. The conditional GAN is different in that the loss is learned, and can, in theory, penalize any possible structure that differs between output and target.

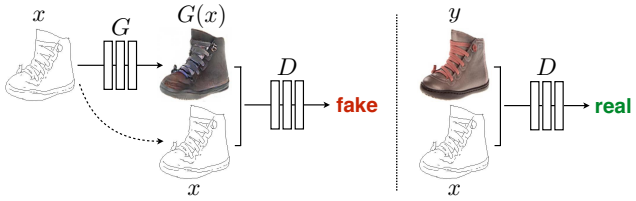

Figure 2: Training a conditional GAN to map edges→photo. The discriminator, D, learns to classify between fake (synthesized by the generator) and real {edge, photo} tuples. The generator, G, learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map.

Conditional GANs We are not the first to apply GANs in the conditional setting. Prior and concurrent works have conditioned GANs on discrete labels [41, 23, 13], text [46], and, indeed, images. The image-conditional models have tackled image prediction from a normal map [55], future frame prediction [40], product photo generation [59], and image generation from sparse annotations [31, 48] (c.f. [47] for an autoregressive approach to the same problem). Several other papers have also used GANs for image-to-image mappings, but only applied the GAN unconditionally, relying on other terms (such as L2 regression) to force the output to be conditioned on the input. These papers have achieved impressive results on inpainting [43], future state prediction [64], image manipulation guided by user constraints [65], style transfer [38], and superresolution [36]. Each of the methods was tailored for a specific application. Our framework differs in that nothing is application-specific. This makes our setup considerably simpler than most others.

Our method also differs from the prior works in several architectural choices for the generator and discriminator. Unlike past work, for our generator we use a "U-Net"-based architecture [50], and for our discriminator we use a convolutional "PatchGAN" classifier, which only penalizes structure at the scale of image patches. A similar PatchGAN architecture was previously proposed in [38] to capture local style statistics. Here we show that this approach is effective on a wider range of problems, and we investigate the effect of changing the patch size.

3. Method

The Method section comes after introduction and related work.

GANs are generative models that learn a mapping from random noise vector z to output image y, $G : z \rightarrow y$ [24]. In

contrast, conditional GANs learn a mapping from observed image x and random noise vector z, to y, $G : \{x, z\} \rightarrow y$. The generator G is trained to produce outputs that cannot be distinguished from "real" images by an adversarially trained discriminator, D, which is trained to do as well as possible at detecting the generator's "fakes". This training procedure is diagrammed in Figure 2.

3.1. Objective

Indicates that the high-level problem will be formalized in this section.

The objective of a conditional GAN can be expressed as

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z))], \quad (1)$$

where G tries to minimize this objective against an adversarial D that tries to maximize it, i.e. $G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D)$.

To test the importance of conditioning the discriminator, we also compare to an unconditional variant in which the discriminator does not observe x:

Introduction to GAN and cGAN, presuming ML expert audience.

$$\mathcal{L}_{GAN}(G, D) = \mathbb{E}_{y}[\log D(y)] + \mathbb{E}_{x,z}[\log(1 - D(G(x, z))]. \quad (2)$$

Previous approaches have found it beneficial to mix the GAN objective with a more traditional loss, such as L2 distance [43]. The discriminator's job remains unchanged, but the generator is tasked to not only fool the discriminator but also to be near the ground truth output in an L2 sense. We also explore this option, using L1 distance rather than L2 as L1 encourages less blurring:

Intuition building for approach.

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x, z)\|_1]. \quad (3)$$

Our final objective is

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G). \quad (4)$$

Without z, the net could still learn a mapping from x to y, but would produce deterministic outputs, and therefore fail to match any distribution other than a delta function. Past conditional GANs have acknowledged this and provided Gaussian noise z as an input to the generator, in addition to x (e.g., [55]). In initial experiments, we did not find this strategy effective – the generator simply learned to ignore the noise – which is consistent with Mathieu et al. [40]. Instead, for our final models, we provide noise only in the form of dropout, applied on several layers of our generator at both training and test time. Despite the dropout noise, we observe only minor stochasticity in the output of our nets. Designing conditional GANs that produce highly stochastic output, and thereby capture the full entropy of the conditional distributions they model, is an important question left open by the present work.

Intuition building for approach, including discussion of failed experiments and explanation of a potentially controversial experimental decision (to not condition on random noise z).

Use technical figures to aid a reader in understanding difficult technical ideas in the Methods.

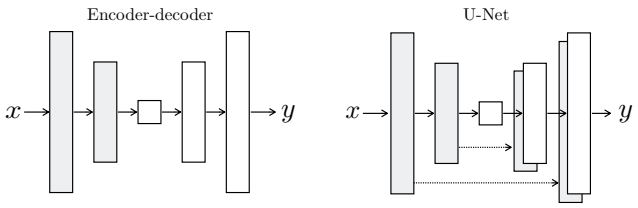

Figure 3: Two choices for the architecture of the generator. The "U-Net" [50] is an encoder-decoder with skip connections between mirrored layers in the encoder and decoder stacks.

3.2. Network architectures

Note the use of subheadings to organize content.

We adapt our generator and discriminator architectures from those in [44]. Both generator and discriminator use modules of the form convolution-BatchNorm-ReLu [29]. Details of the architecture are provided in the supplemental materials online, with key features discussed below.

3.2.1 Generator with skips

Nested levels of subheading for easy section navigation.

A defining feature of image-to-image translation problems is that they map a high resolution input grid to a high resolution output grid. In addition, for the problems we consider, the input and output differ in surface appearance, but both are renderings of the same underlying structure. Therefore, structure in the input is roughly aligned with structure in the output. We design the generator architecture around these considerations.

Many previous solutions [43, 55, 30, 64, 59] to problems in this area have used an encoder-decoder network [26]. In such a network, the input is passed through a series of layers that progressively downsample, until a bottleneck layer, at which point the process is reversed. Such a network requires that all information flow pass through all the layers, including the bottleneck. For many image translation problems, there is a great deal of low-level information shared between the input and output, and it would be desirable to shuttle this information directly across the net. For example, in the case of image colorization, the input and output share the location of prominent edges.

To give the generator a means to circumvent the bottleneck for information like this, we add skip connections, following the general shape of a "U-Net" [50]. Specifically, we add skip connections between each layer i and layer $n - i$, where n is the total number of layers. Each skip connection simply concatenates all channels at layer i with those at layer $n - i$.

3.2.2 Markovian discriminator (PatchGAN)

It is well known that the L2 loss – and L1, see Figure 4 – produces blurry results on image generation problems [34]. Although these losses fail to encourage high-

Description of technical contribution is the primary focus of the section.

frequency crispness, in many cases they nonetheless accurately capture the low frequencies. For problems where this is the case, we do not need an entirely new framework to enforce correctness at the low frequencies. L1 will already do.

This motivates restricting the GAN discriminator to only model high-frequency structure, relying on an L1 term to force low-frequency correctness (Eqn. 4). In order to model high-frequencies, it is sufficient to restrict our attention to the structure in local image patches. Therefore, we design a discriminator architecture – which we term a *Patch*GAN – that only penalizes structure at the scale of patches. This discriminator tries to classify if each $N \times N$ patch in an image is real or fake. We run this discriminator convolutionally across the image, averaging all responses to provide the ultimate output of D.

In Section 4.4, we demonstrate that N can be much smaller than the full size of the image and still produce high quality results. This is advantageous because a smaller PatchGAN has fewer parameters, runs faster, and can be applied to arbitrarily large images.

Such a discriminator effectively models the image as a Markov random field, assuming independence between pixels separated by more than a patch diameter. This connection was previously explored in [38], and is also the common assumption in models of texture [17, 21] and style [16, 25, 22, 37]. Therefore, our PatchGAN can be understood as a form of texture/style loss.

Subheading title summarizes technical focus.

3.3. Optimization and inference

To optimize our networks, we follow the standard approach from [24]: we alternate between one gradient descent step on D, then one step on G. As suggested in the original GAN paper, rather than training G to minimize $\log(1 - D(x, G(x, z))$, we instead train to maximize $\log D(x, G(x, z))$ [24]. In addition, we divide the objective by 2 while optimizing D, which slows down the rate at which D learns relative to G. We use minibatch SGD and apply the Adam solver [32], with a learning rate of 0.0002, and momentum parameters $\beta_1 = 0.5$, $\beta_2 = 0.999$.

At inference time, we run the generator net in exactly the same manner as during the training phase. This differs from the usual protocol in that we apply dropout at test time, and we apply batch normalization [29] using the statistics of the test batch, rather than aggregated statistics of the training batch. This approach to batch normalization, when the batch size is set to 1, has been termed "instance normalization" and has been demonstrated to be effective at image generation tasks [54]. In our experiments, we use batch sizes between 1 and 10 depending on the experiment.

Some details about experiments - learning rate and momentum parameters - are included. These are likely placed here because they are "universal" parameters that the authors believe to be critical to the system.

4. Experiments

The experiments section immediately follows the methods

To explore the generality of conditional GANs, we test the method on a variety of tasks and datasets, including both graphics tasks, like photo generation, and vision tasks, like semantic segmentation:

- *Semantic labels↔photo*, trained on the Cityscapes dataset [12].
- *Architectural labels→photo*, trained on CMP Facades [45].
- *Map↔aerial photo*, trained on data scraped from Google Maps.
- *BW→color photos*, trained on [51].
- *Edges→photo*, trained on data from [65] and [60]; binary edges generated using the HED edge detector [58] plus postprocessing.
- *Sketch→photo*: tests edges→photo models on human-drawn sketches from [19].
- *Day→night*, trained on [33].
- *Thermal→color photos*, trained on data from [27].
- *Photo with missing pixels→inpainted photo*, trained on Paris StreetView from [14].

Details of training on each of these datasets are provided in the supplemental materials online. In all cases, the input and output are simply 1-3 channel images. Qualitative results are shown in Figures 8, 9, 11, 10, 13, 14, 15, 16, 17, 18, 19, 20. Several failure cases are highlighted in Figure 21. More comprehensive results are available at `https://phillipi.github.io/pix2pix/`.

Data requirements and speed We note that decent results can often be obtained even on small datasets. Our facade training set consists of just 400 images (see results in Figure 14), and the day to night training set consists of only 91 unique webcams (see results in Figure 15). On datasets of this size, training can be very fast: for example, the results shown in Figure 14 took less than two hours of training on a single Pascal Titan X GPU. At test time, all models run in well under a second on this GPU.

4.1. Evaluation metrics

Evaluating the quality of synthesized images is an open and difficult problem [52]. Traditional metrics such as per-pixel mean-squared error do not assess joint statistics of the result, and therefore do not measure the very structure that structured losses aim to capture.

To more holistically evaluate the visual quality of our results, we employ two tactics. First, we run "real vs. fake" perceptual studies on Amazon Mechanical Turk (AMT). For graphics problems like colorization and photo generation, plausibility to a human observer is often the ultimate goal. Therefore, we test our map generation, aerial photo generation, and image colorization using this approach.

The Experiments section also includes evaluation metrics and results, demonstrating the blending between Methods, Experiments, and Results that happens in many CS papers.