Chapter 1

Conditional generative models

Draft chapter from Torralba, Isola, Freeman

In the preceding chapters, we learned about two uses of generative models: 1) as a way to synthesize realistic but novel data, and 2) as a way to learn representations of the data. Now we will introduce a third use, which is arguably the most widespread use of generative models: *as a way to solve prediction problems.*

1.1 A motivating example: image colorization

To motivate this use case, consider the following problem. We wish to colorize a black and white photo, that is, we wish to predict the color of each pixel in a black and white photo.

Now, we already have seen some tools we could apply to this problem. First we will try to solve it with least-squares regression. Second, with softmax classification. We will see that both approaches fall short of a good solution. This will motivate the necessity of conditional generative models, which turn out to solve the colorization problem quite nicely, and can produce nearly photorealistic results.

1.1.1 The failure of point prediction: multimodal distributions

We could formulate the problem as least-squares regression: train a function to output a vector of real-valued numbers, representing the R, G, and B values of every pixel in the image, then penalize the squared distance between these values and the ground truth values.

This kind of regression fits a function $f : \mathcal{X} \to \mathcal{Y}$. For every input $\mathbf{x} \in \mathcal{X}$, the output is a single **point prediction** $\hat{\mathbf{y}} \in \mathcal{Y}$, that is, we output just a *single* prediction of what the value of $\mathbf{y}$ should be.

What if there are multiple equally valid answers? Taking a color image and making it grayscale is a many-to-one mapping; for any given grayscale value, there are many different color values that could have projected to that value of gray. This means that the colorization problem is fundamentally ambiguous, and there may be multiple valid color solutions for any grayscale input. A point estimate is bound to be a poor inference in this case, since a point estimate can only predict one of the multiple possible solutions.

Figure 1.1 shows several kinds of predictions one could make given an observation of a grayscale t-shirt. The question is: "what's the color of the shirt?" To unify different kinds of prediction models, we will represent them all as outputting a distribution over the possible colors of the shirt. Let A be a random variable that represents the a value of the shirt in lab color space. Our input is just the l value of the shirt. Our predictions will be represented as $p_\theta(A|l)$. The particular shirt we are looking at comes in two colors, either teal or pink. The true data distribution, p_{data}, is therefore two delta functions, one on teal and the other on pink.

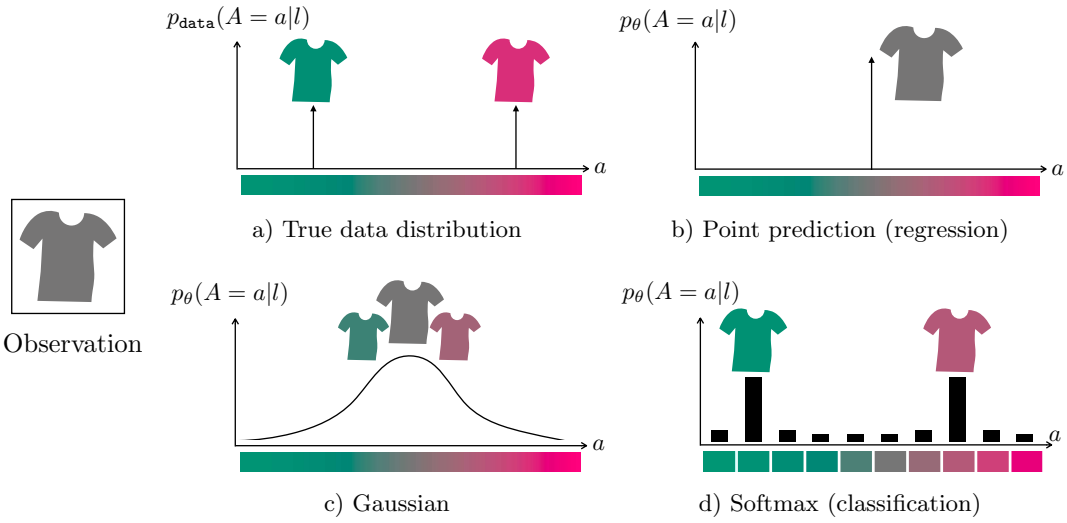


Figure 1.1: Different kinds of predictive distributions.

A point prediction does not necessarily come with probabilistic semantics, but here we will interpret a prediction of $\hat{a} = f_\theta(l)$ as implying a predictive distribution over A that has the form $p_\theta(A = a|l) = \delta(a - \hat{a})$, where δ is the Dirac delta function.

Least-squares regression results in a model that predicts the *mean* of the data distribution. Therefore, the point prediction output by a least-squares regressor will be that the shirt is gray. The probability density associated with this prediction is shown in Figure 1.1(b).

This prediction is entirely wrong! It splits the difference between the two true possibilities and comes out with something that has *zero* probability under the data distribution. As discussed in Chapter **??**, we could have done regression with a different loss function (using the L_1 loss for example, rather than L_2), and we would have come up with a different solution. But we will never get the correct solution. Because the true distribution has two equally probable modes, and a single point prediction can only ever represent one mode.

We can do better by predicting a *distribution* rather than a point estimate. An example is shown in Figure 1.1(c), where we predict a Gaussian distribution. In fact, the least-squares regression model can be described as outputting the mean of a max likelihood Gaussian fit to $p_\theta(A|l)$. Predicting the distribution then just requires also predicting the variance of the Gaussian. This gives us a better sense of the possible colors the t-shit could be, but it is still a unimodal distribution, while the data is bimodal.

Naturally, then, we may want to predict a more expressive distribution – a mixture of two Gaussians could, for example, capture the bimodal nature of the data. An easy way to predict a distribution is to predict the parameters of some parametric family of distributions. This is precisely what we did with the Gaussian fit: the parameters of a 1D Gaussian are its mean and variance, so if $\hat{\mathbf{y}} \in \mathbb{R}^2$ then $\hat{\mathbf{y}}$ suffices to parameterize a 1D Gaussian. A mixture of N Gaussians just requires outputting $3N$ numbers: the mean, variance, and weight of each Gaussian in the mixture. The loss function could then just be the data likelihood under the mixture of Gaussians parameterized by $\hat{\mathbf{y}} \in \mathbb{R}^{3N}$.

One of the most common choices for an expressive, multimodal family of distributions is the categorical distribution, $\texttt{Cat}$. This distribution applies only to discrete random variables, so to use it, the first thing we have to do is quantize our possible output values. Figure 1.1(d) shows an example, where we have quantized the a-value into 9 bins. Then the categorical distribution is simply a 9-dimensional vector of non-negative numbers that sum to 1 (a.k.a. a probability mass function). The nice thing about this parameterization is that *all* probability mass functions over k classes are members of the family $\texttt{Cat}(k)$. In other words, this is the most expressive distribution possible over a discrete random variable! That's great because it means we can use it to represent predictions with any number of modes (well, up to k, the resolution of our quantized data).

In fact, we have already seen one super important use of the categorical distribution:

softmax classification (a.k.a. multinomial logisitic regression). Now you might have a better understanding of why classification is such a ubiquitous modeling tool: it models a maximally expressive predictive distribution over a quantized decision space.

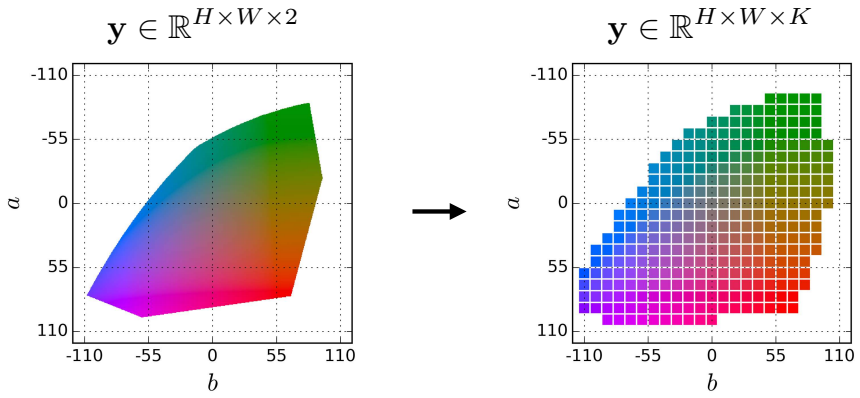
Classification to the rescue? Classification solves some of the deficiencies of point prediction. However, it comes with several new problems of its own:

- It incurs quantization error.
- It may require exponentially many classes w.r.t. the data dimensionality.

The first is not such a big issue. You can see its effect in Figure 1.1(d). The true colors are teal and bright pink but the quantization lowers the color resolution and we cannot know if the prediction is that the shirt is dull pink or bright pink as both fall in the same bin. This is why the pink shirt here appears slightly duller than in the data distribution (we colored each bin with the floor of its range). But generally this isn't such a big deal. We can always increase the resolution by increasing k. Most image formats only represent 256 possible values for a, so if we set $k = 256$ then we incur no quantization error.

Why 256 values? Because each channel in standard image formats is stored as an array of `uint8`s.

The second issue is much more severe. If we are predicting the a-value of a single pixel, then the classification approach says there are k possible values it could take on. This approach can be extended to predicting ab-values: we come up with a list of discrete color classes, like "red", "orange", "turquoise", and so forth. But to tile the two dimensional space of ab-values will require k^2 color classes, if we wish to retain a resolution of k for both a and b. Nonetheless, we can still do this using a reasonable number of classes, and it might look like this:



The funny shape of the lab color gamut is because not every ab value maps to a valid pixel color. When working with predictions over lab color space, we may clip predict ab values that fall outside the "gamut" (valid range) to the nearest in-gamut value.

The real problem comes about when predicting more than one pixel. To quantize the ab-values of N pixels requires k^{2N} classes, again assuming we want a resolution of k for the a and b value of each pixel. For a 256×256 resolution image, the number of classes required is astronomical (for $k = 10$, the number is a one followed by over 100,000 zeros).

Quantizing on a grid doesn't scale to high-dimensions, but more intelligent quantization methods can work. These more intelligent methods are called **vector quantization** or **clustering**, and we cover them in Chapter **??**.

1.1.2 The failure of independent predictions: joint structure

To circumvent this "curse of dimensionality", we can turn to factorization: rather than treating the whole configuration of pixels as a class, make *independent* predictions for each pixel in the image. From a probabilistic inference perspective, the corresponds to the following factorization of the joint distribution:

$$p_\theta(\mathbf{ab}|\mathbf{l}) = \prod_i p_\theta(ab_i|\mathbf{l}) \tag{1.1}$$

The underlying assumption of this factorization is one of conditional independence: each pixel's ab value is considered to be conditionally independent from all other pixels' ab values,

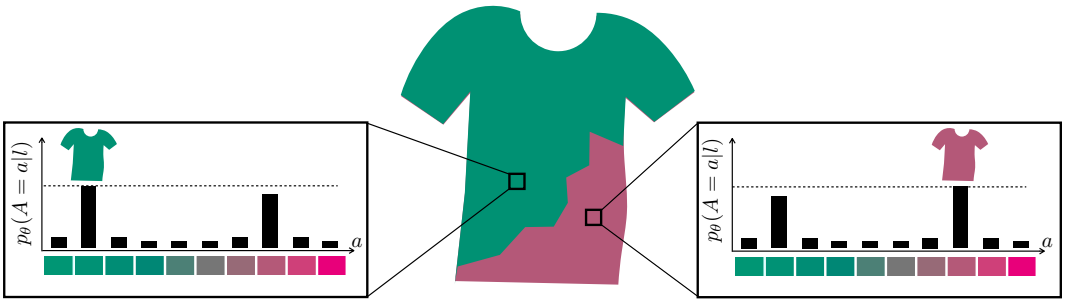


Figure 1.2: Color "flipping" can arise from a smooth underlying predictive distribution, on top of which independent choices are made.

given the observed luminance (of all pixels). This is a very common assumption in image modeling problems: in fact, *whenever* you use least-squares regression for a multidimensional prediction you are implicitly making this same assumption. To see this, suppose you are predicting a vector $\mathbf{y}$ from an observed vector $\mathbf{x}$, your prediction is $\hat{\mathbf{y}} = f(\mathbf{x})$, and you are using the L_2 loss. We can rewrite this loss as:

$$-\|\hat{\mathbf{y}} - \mathbf{y}\|_2^2 = \sum_i -(\hat{y}_i - y_i)^2 \tag{1.2}$$

$$= \log \prod_i \phi(\hat{y}_i, y_i) \tag{1.3}$$

The loss factorizes as a product over pairwise potentials. Therefore, by the Hammersley-Clifford theorem (Chapter **??**), the L_2 loss implies a probability distribution that treats all true values y_i as independent from one another, given all the predictions $\hat{y}_i$. The predictions are a function of just the input $\mathbf{x}$, so the implication is that all the true values y_i as independent from one another given the observation $\mathbf{x}$. Therefore, we have arrived at our conditional independence assumption: each dimension (pixel's) predicted value is assumed to be independent of each other dimension (other pixels') predicted values, given the observed input. This is a huge assumption and rarely true of prediction problems in computer vision!

So, whether you are using per-pixel classification, or least-squares regression, you are implicitly fitting a model that assumes independence between all the output pixels, conditioned on the input. This is called **unstructured prediction**.

A real example of color flipping from [Zhang et al. 2016]. The model is unsure whether the shirt, and the background, are blue or red, so it chaotically alternates between these two options.

This causes problems. Let's return to our t-shirt example. We will use a per-pixel color classifier and see where it fails. Since the data distribution has two equally probable modes – teal and pink – the classifier will learn to place roughly equal probability mass on these two modes. As training data and time go to infinity, the classifier should recover the exact data distribution, but with finite data and time it will only be approximate, and so we might have a case where for some luminance values the classifier places 51% chance on teal and for others it places 49% on teal. Then if, as we scan across pixels in the shirt we are observing, the luminance changes very slightly, the model predictions might wiggle back and forth between 49% and 51% teal. As our application is to colorize the photo, at some point we need to make a hard decision and output a single color for each pixel. Doing so in the present case will cause chaotic transitions from predicting pink (where $p(\text{teal}) < 0.5$) and teal (where $p(\text{teal}) > 0.5$). An example of this kind of prediction "flipping" is shown in Figure 1.2

1.2 Conditional generative models solve multimodal structured prediction

In the previous section, we learned that standard approaches to prediction are insufficient for making the kinds of predictions we usually care about in computer vision.

Conditional generative models are a general family of prediction methods that:

- Model a multimodal distribution of predictions
- Model joint structure

Methods that model joint structure in the output space are called **structured prediction** methods – they don't factorize the output into independent potentials conditioned on the input. Conditional generative modeling is a structured prediction approach that models a full distribution of possibilities over the joint configuration of outputs.

Relationship to Conditional Random Fields The conditional random fields (CRFs) from Chapter **??** are one kind of model that fits this definition. Now we will see some other ones. The big difference is that CRFs make predictions via "thinking slow": given a query observation, you run belief propagation or another iterative inference algorithm to arrive at a prediction. The conditional generative models we will see in this section "think fast": they do inference through a single forward pass of a neural net. Sometimes the "thinking fast" approach is referred to as **amortized inference**, where the idea is that the cost of inference is amortized over a training phase. This training phase learns a direct mapping from inputs to outputs that approximates the solution we would have gotten if we did exact inference on that input.

1.3 A tour of popular conditional models

We saw a bunch of unconditional generative models in the previous chapters. How can we make each conditional? It is usually pretty straightforward. This is because if you can model an arbitrary distribution over a random variable, $p(Y)$, then you can certainly model the conditional distribution $p(Y|X = \mathbf{x})$ – it's just another arbitrary distribution. Of course, we typically care about modeling $p(Y|X = \mathbf{x})$ for *all* possible settings of $\mathbf{x}$. We could, but don't want to, fit a separate generative model for each $\mathbf{x}$. Instead we want neural nets that take a query $\mathbf{x}$ as input and produce an output that models or samples from $p(Y|X = \mathbf{x})$. We will briefly cover how to do this for several popular models:

GANs We can make a GAN conditional simply both adding $\mathbf{x}$ as an input to both the generator and the discriminator:

$$\arg\min_G \max_D \mathbb{E}_{\mathbf{z},\mathbf{x},\mathbf{y}}\big[\log D(\mathbf{x}, G(\mathbf{x}, \mathbf{z})) + \log(1 - D(\mathbf{x}, \mathbf{y}))\big] \tag{1.4}$$

What this does is change the job of the discriminator from asking "is the output real or fake?" to asking "is the input-output *pair* real or fake?" An input-output pair can be considered fake for two possible reasons:

1. The output looks fake
2. The output does not match the input

If both reasons are avoided, then it can be shown that the produced samples are *iid* w.r.t. the true conditional distribution of the data $p_{\texttt{data}}(Y|\mathbf{x})$ (this follows from the analogous proof for unconditional GANs in [Goodfellow et al. 2014] since that proof applies to modeling any arbitrary distribution over Y, including $p_{\texttt{data}}(Y|\mathbf{x})$ for any $\mathbf{x}$).

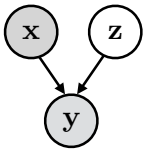
VAEs Recall that a VAE is an infinite mixture model which makes use of the following identity:

$$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p_\theta(\mathbf{x}|\mathbf{z}) p_{\mathbf{z}}(\mathbf{z}) d\mathbf{z} \tag{1.5}$$

Analogously, we can define any conditional distribution as the marginal over some latent variable:

$$p_\theta(\mathbf{y}|\mathbf{x}) = \int_{\mathbf{z}} p_\theta(\mathbf{y}|\mathbf{z},\mathbf{x}) p_{\mathbf{z}}(\mathbf{z}|\mathbf{x}) d\mathbf{z} \tag{1.6}$$

Corresponds to this graphical model:



In conditional VAEs (**cVAE**s), we restrict our attention to latent variables $\mathbf{z}$ that are independent of the inputs we are conditioning on, so we have:

$$p_\theta(\mathbf{y}|\mathbf{x}) \triangleq \int_{\mathbf{z}} p_\theta(\mathbf{y}|\mathbf{z},\mathbf{x}) p_{\mathbf{z}}(\mathbf{z}) d\mathbf{z} \qquad \triangleleft \text{ cVAE likelihood model} \tag{1.7}$$

The idea is that $\mathbf{z}$ should only encode bits of information about $\mathbf{y}$ that are *independent* from whatever $\mathbf{x}$ already tells us about y. For example, suppose that we are trying to predict the motion of a billiard ball bouncing around in a video sequence. We are given a single frame $\mathbf{x}$ and asked to predict the next frame $\mathbf{y}$. Only knowing $\mathbf{x}$ we can't know whether the ball will move up, down, diagonally, etc, but we *can* know what color the ball will be in the next frame (it must be the same as the previous frame) and the rough position on the screen of the ball (it can't have moved too far). Therefore, the only missing information about $\mathbf{y}$, given we know $\mathbf{x}$, is the velocity of the ball. Naturally, then, if the model learns to interpret $\mathbf{z}$ as coding for velocity, we would have a perfect prediction $p_\theta(\mathbf{y}|\mathbf{z},\mathbf{x})$ (one that places max likelihood on the observed next frame $\mathbf{y}$), and marginalizing over all the possible $\mathbf{z}$'s would place max likelihood on the data ($p_\theta(\mathbf{y}|\mathbf{x})$). This is what ends up happening in a cVAE (or, to be precise, it is one solution that maximizes the cVAE objective; it is not guaranteed to be the solution that is arrived to, but it is a good model of what tends to happen in practice). Figure 1.3 shows this scenario.

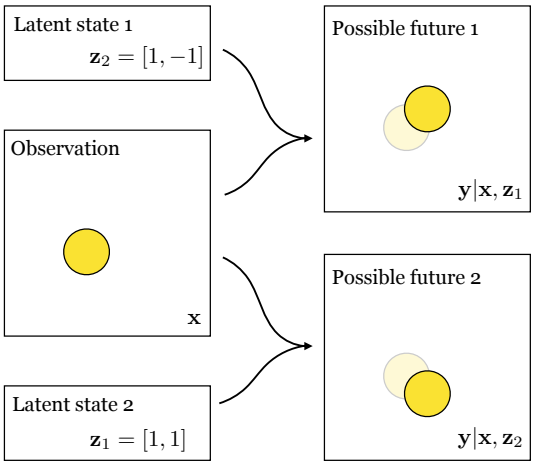


Figure 1.3: A scenario where a yellow ball is moving across a plane. The observation, $\mathbf{x}$, is a static frame. From that observation, we know what will be the color and rough position of the ball in the next frame, $\mathbf{y}$, but we don't know what direction it will have moved, because the velocity of the ball is unobserved. Therefore, velocity is a latent variable and one solution to the cVAE objective will be to encode in the model's latent variables, $\mathbf{z}$, the velocity of the ball, as is depicted here.

Just like regular VAEs, cVAEs also have an encoder, which acts to predict the optimal importance sampling distribution $p_\theta(Z|\mathbf{x},\mathbf{y})$. In practice, this means that a cVAE can be trained just like a regular VAE except that the encoder takes the conditioning information, $\mathbf{x}$, as input (in addition to $\mathbf{y}$), and the decoder also takes in $\mathbf{x}$ (in addition to $\mathbf{y}$). Figure 1.4 depicts this setting.

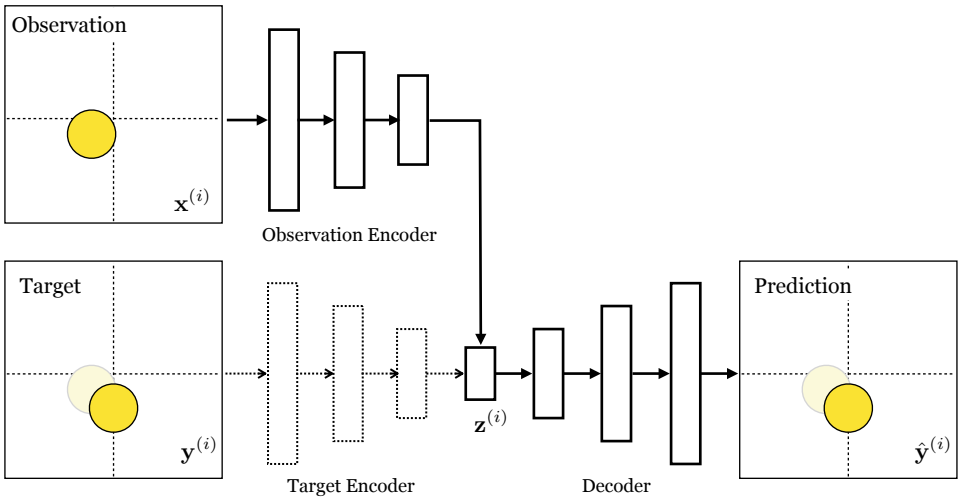


Figure 1.4: cVAE architecture. The dotted lines indicate that the "Target Encoder" is only used during training; at test time, usage follows the solid path.

Autoregressive models Autoregressive models are already modeling a sequence of conditional distributions. So, to condition them on some inputs $\mathbf{x}$, we can simply concatenate $\mathbf{x}$ as a prefix to the sequence of y's we are modeling, yielding the new sequence: $[x_1, \ldots, x_m, y_1, \ldots, y_n]$. The probability model factorizes as:

$$p_\theta(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} p_\theta(y_i|y_1, \ldots, y_{i-1}, x_1, \ldots, x_m) \tag{1.8}$$

Each distribution in this product is a prediction of the next item in a sequence given the previous items, which is no different than what we had for unconditional autoregressive models. Therefore, the tools for modeling unconditional autoregressive distributions are also appropriate for modeling conditional autoregressive distributions. From an implementation perspective, the same exact code will handle both cases, just depending on whether you prefix the sequence or not.

1.4 Structured prediction in vision

Whenever you want to predict an *image*, conditional generative models are a suitable modeling choice. But how often do we really want to "predict an image"? Yes, image colorization is one example, but isn't that more of a graphics problem? Most problems in vision are about predicting labels or geometry, right?

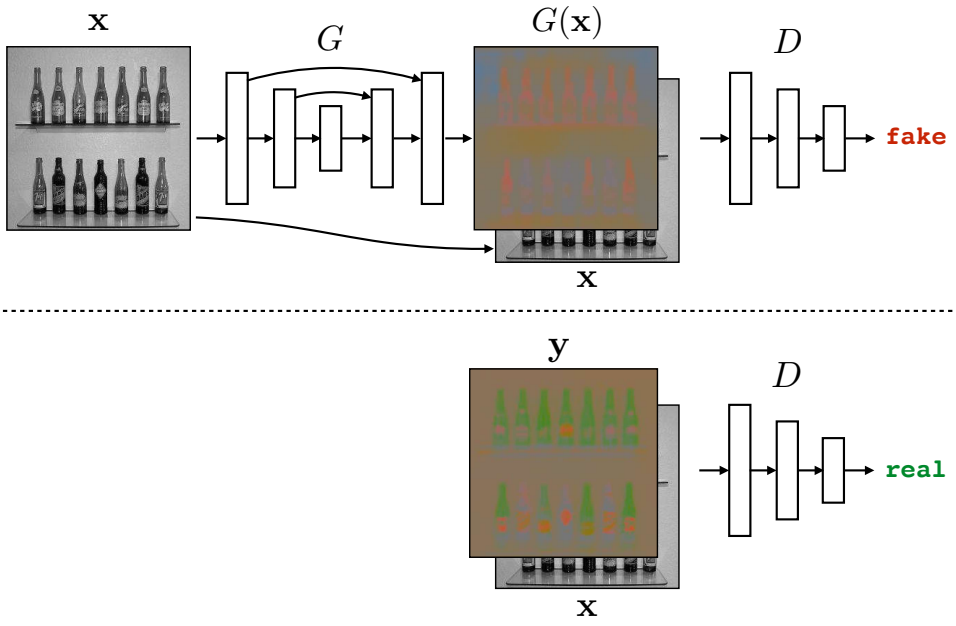
Well yes, but what does label prediction look like? The input is an image and the output is label map. In image classification the output might just be a single class, but more generally, in object detection and semantic segmentation, we want a label for each part of the image. The target output in these problems is high-dimensional and structured. Or consider geometry estimation: the output is a depth map, or a voxel grid, or a mesh, and so on. All these are high-dimensional structured objects. The tool we need for solving these problems is structured prediction, and conditional generative models are therefore a good choice. In the next sections we will see two important families of structured prediction methods used in vision.

1.5 Image-to-image translation

Image-to-image problems are mapping problems where the input is an image and the output is also an image, where we will here think of an image as any array of size $N \times M \times C$ (height by width by number of channels). These problems are super common in computer vision. Examples include colorization, next frame prediction, depth map estimation, and semantic segmentation (a per-pixel label map is an "image" just with K channels, one for each label, rather than 3 channels, one for each color channel). One way to think about all these problems is as *translating* from one view of the data to another; for example, semantic segmentation is a translation from viewing the world in terms of its colors to viewing the world in terms of its semantics. This perspective is yields the problem of **image-to-image translation** [Isola et al. 2017] – just like we can translate from English to French, can we translate from pixels to semantics, or perhaps from a photographic depiction of scene to a painting of that same scene?

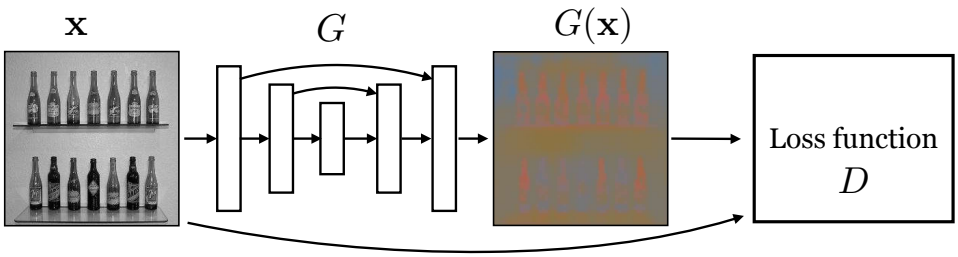
1.5.1 Image-to-image translation with a conditional GAN

One popular approach to image-to-image translation is to use a conditional GAN, as was popularized in the "pix2pix" paper [Isola et al. 2017] (whose method we will follow here). To illustrate this approach, we will return to the colorization problem. In this setting, the conditioning information (input) is a black and white image and the output is a color image. The generator there maps an image to an image so we will implement it with a image-to-image neural architecture – this could be a convolutional network, or a transformer; following pix2pix we will use a U-net (a convnet with accordion-shaped skip connections, Chapter **??**). The discriminator maps the output image to a "real vs fake" score (a scalar), so for the discriminator we will use a regular convnet (just like the ones used for image classification). Here is the full architecture: Notice that we have omitted the noise inputs here – instead



the generator only takes as input the image $\mathbf{x}$. It turns out that in this setting the noise is not really a necessary input, and many implementations omit it. This is because the input image $\mathbf{x}$ has enough "entropy" by itself, and you don't really need additional noise to create variety in the outputs of the model. The downside of this approach is that you only get one prediction $\mathbf{y}$ for each input $\mathbf{x}$.

One way to think about this setup is that it is a regression problem with a *learned loss function* D:



This loss function adapts to the structure of the data and the behavior of the generator to penalize just the relevant errors the generator is currently making. It can help to also add a conventional regression loss, such as the L_1 loss, to stabilize the optimization process, yielding the following objective:

$$\arg\min_G \max_D \mathbb{E}_{\mathbf{z},\mathbf{x},\mathbf{y}} \left[\log D(\mathbf{x}, G(\mathbf{x}, \mathbf{z})) + \log(1 - D(\mathbf{x}, \mathbf{y})) + \|G(\mathbf{x}) - \mathbf{y}\|_1 \right] \tag{1.9}$$

Thinking of D as a learned loss function raises new questions: what does this loss function penalize, and how can we control its properties. One of the main levers we have is the architecture of D; different architectures will have the capacity, and/or inductive bias, to penalize different kinds of errors. One popular architecture for image-to-image tasks is a "PatchGAN" discriminator, in which we score each *patch* in the output image as real or fake and take the average over patch scores as our total loss:

$$D(\mathbf{x}, \mathbf{y}) = \frac{1}{NM} \sum_{i=0}^{N} \sum_{j=0}^{M} D_{\texttt{patch}}(\mathbf{x}[i:i+k, j:j+k], \mathbf{y}[i:i+k, j:j+k]) \tag{1.10}$$

Where $k \times k$ is the patch size. Notice that this operation is equivalent to the sliding window action of convnet, so D can simply be implemented as a convnet that outputs an $N \times M \times 1$ "feature map" of real vs fake scores. Naturally, patches can be sampled at different strides and resolutions, depending on the exact architecture of the convnet D.

The PatchGAN strategy has two advantages over just scoring the whole image as real or fake with a classifier architecutre: 1) it can be easier to model if a patch is real or fake than to model if an entire image is real or fake (D needs fewer parameters), 2) there are more patches in the training data than their are images. These two properities give PatchGAN discriminators a statistical advantage over whole image discrimatinators (fewer parameters fit to more data). The disadvantage is that the PatchGAN only has the architectural capacity to penalize errors that are observable within a single patch. This can be seen by training models with different discriminator patch sizes and observing the results. We show this below for a conditional GAN trained to map from architectural label maps to building facade images (the label colors indicate whether that region of the image contains a window, door, column, etc):

1.5.2 Unpaired image-to-image translation

In the preceding section we saw how to solve image-to-image translation by treating it just like a supervised regression problem, except using a learned loss function given by a GAN discriminator. This approach works great when we have lots of training data pairs $\{\mathbf{x}, \mathbf{y}\}$ to learn from. Unfortunately, very often, paired training data is hard to obtain. For example, consider the task of translating a photo into a Cezanne painting: "what would it have looked like if Cezanne had stood here and painted this river bank?" The input and output to one algorithm that can do this (CycleGAN [Zhu et al. 2017]) is shown below:

How could this be done? We can't have solved it with paired training data because, in this setting, paired data is extremely hard to come by. We would have to resurrect Cezanne and ask him to paint for us a bunch of new scenes, one for each photo in our desired training

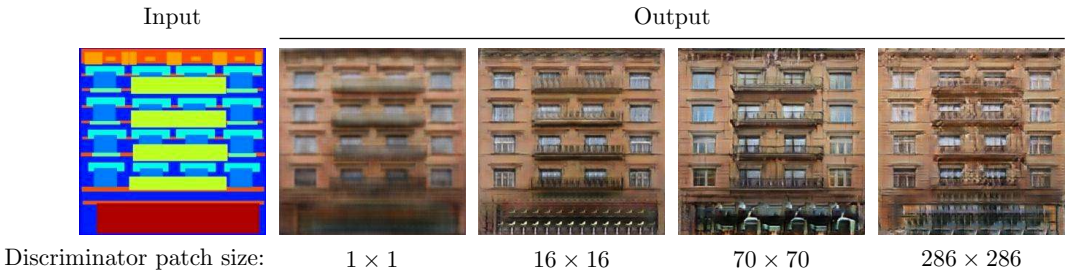


Figure 1.5: Varying the receptive field (patch size) of the convolutional discriminator affects what kinds of structure the discriminator enforces. A 1×1 discriminator can only observe a single pixel at a time and cannot penalize errors in joint pixel statistics, such as edge structure, hence the blurry results. Larger receptive fields can enforce higher order patch realism, but cannot model structure larger than the patch size (hence the tiling artifacts that are occur with a period roughly equal to the patch size). Quality degrades with the 286×286 discriminator possibly because this disciminator has too hard a task given the limited training data (in any given training set, there are fewer examples of 286×286 regions than there are of, e.g., 70×70 regions). These results are from [Isola et al. 2017].

set. But is all that effort really necessary? You and I, as humans, can certainly imagine the answer to the question above. We know what Cezanne's style looks like and can reason about the changes that would have to occur to make the photo match his style. Dabs of paint would have to replace the photographic pixels, and the colors should take on more pastel tones. We can imagine the necessary stylistic changes because we have seen many Cezanne paintings before, even though we didn't see them paired with a photograph of the exact same scenes. Let's now see how we can give a machine this same ability. We call this setting the **unpaired translation** setting, and distinguish it from **paired translation** as indicated in Figure 1.6.

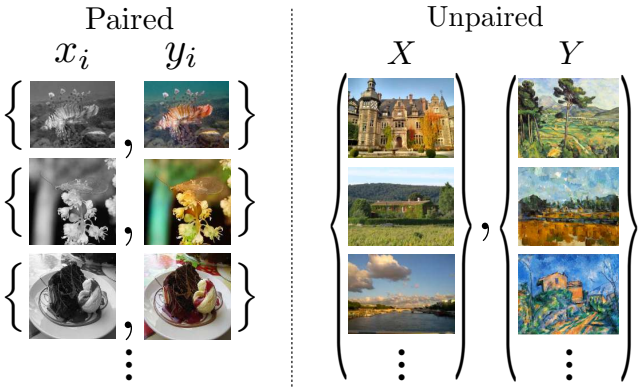


Figure 1.6: An example of paired image-to-image translation (left, colorization in this example) vs unpaired translation (right, photo-to-cezanne).

It turns out that GANs have a very useful property that makes them well-suited to solving this task. GANs train a mapping from a noise distribution to a data distribution $\mathcal{Z} \rightarrow \mathcal{X}$.

They do this without knowing the pairing between $\mathbf{z}$'s and $\mathbf{x}$'s in the training data. Instead this pairing is *emergent*. Which $\mathbf{z}$-vector corresponds to which $\mathbf{x}$ is not predetermined but self-organizes to satisfy the GAN objective (all $\mathbf{z}$-vectors should map to realistic images, and, collectively they should map to the data distribution).

Now consider training a GAN to map from $\mathbf{x}$'s to $\mathbf{y}$'s:

$$\arg\min_G \max_D \mathbb{E}_{\mathbf{x},\mathbf{y}}\big[\log D(G(\mathbf{x})) + \log(1 - D(\mathbf{y}))\big] \tag{1.11}$$

Note that this is a regular GAN objective, rather than a conditional GAN, except that we have relabeled the variables compared to Equation **??**, with $\mathbf{y}$ playing the role of $\mathbf{x}$ and $\mathbf{x}$ playing the role of $\mathbf{z}$

Such a GAN would self-organize so that the outputs are realistic $\mathbf{y}$'s and so that different inputs map to different outputs (collectively they must map to the data distribution over $\mathbf{y}$'s. No paired supervision is required to achieve this.

Such a GAN may indeed learn the correct mapping from $\mathbf{x}$'s to $\mathbf{y}$'s – that's one of the solutions that minimizes the loss – but there are many symmetries in this learning problem, i.e. many different mappings achieve equivalent loss. For example, consider that we permute the mapping: if $\{\mathbf{x}_i, \mathbf{y}_i\}$ is the true mapping then consider we instead recover $\{\mathbf{x}_i, \mathbf{y}_{i+1}\}$ after learning (circularly shifting at the boundary). This mapping achieves equal loss to the true mapping, under the standard GAN objective (Equation 1.5.2). This is because any permutation in the mapping does not affect the output *distribution*, and the GAN objective only cares about the output distribution.

Put another away, the GAN discriminator is different than a normal loss function. Rather than checking if the output matches a target *instance* (that $G(\mathbf{x}_i)$ matches $\mathbf{y}_i$), it checks if the output is part of an admissible set – the set of things that look like realistic $\mathbf{y}'s$. This property makes GANs perfect for working with unpaired data, where we don't have instance-level supervision but only have set-level superversion.

We call this "unpaired" learning rather than unsupervised because we still have supervision in the form of labels indicating which *set* each datapoint belongs to. That is, we know whether each image is a $\mathbf{x}$ or a $\mathbf{y}$.

So, a regular GAN objective, applied to mapping from $\mathcal{X}$ to $\mathcal{Y}$, solves the unpaired translation problem, but it does not distinguish between the correct mapping and any permutation of this mapping. To break the symmetry we need to add additional constraints or inductive biases. One that works especially well is **cycle-consistency**, which was introduced in this context by CycleGAN [Zhu et al. 2017], DualGAN [**?**], and DiscoGAN [**?**], which are all essentially the same model. The idea of cycle-consistency is that if we translate from $\mathcal{X}$ to $\mathcal{Y}$ and then translate back – from $\mathcal{Y}$ to $\mathcal{X}$ – we should arrive where we started. Think about translating from English to French, for example. If we translate "apple" to French, "apple"→"pomme", and then translate back, "pomme"→"apple", we arrive where we started. We can check the quality of a translation system, for a language we are unfamiliar with, by using this trick. If backtranslation does not return the word we started with then something went wrong with the translation model. This is because we expect language translation to be roughly one-to-one: for each word in English there is usually an equivalent word in French. Cycle-consistency losses are regularizers that encourage a learend mapping to be roughly one-to-one. The cycle-consistency losses from CycleGAN is simply:

In natural language processing, this idea is known as "backtranslation" [**?**].

$$\mathcal{L}_{\texttt{cyc}}(\mathbf{x}; G, F) = \|\mathbf{x} - F(G(\mathbf{x}))\|_1 \tag{1.12}$$

$$\mathcal{L}_{\texttt{cyc}}(\mathbf{y}; G, F) = \|\mathbf{y} - G(F(\mathbf{y}))\|_1 \tag{1.13}$$

Adding this loss to Equation 1.5.2 yields the complete CycleGAN objective:

$$\arg\min_{G,F} \max_{D_\mathcal{X}, D_\mathcal{Y}} \mathbb{E}_{\mathbf{x},\mathbf{y}}\big[\log D(G(\mathbf{x})) + \log(1 - D(\mathbf{y})) + \|\mathbf{x} - F(G(\mathbf{x}))\|_1 + \tag{1.14}$$

$$\log D(F(\mathbf{y})) + \log(1 - D(\mathbf{x})) + \|\mathbf{y} - G(F(\mathbf{y}))\|_1\big] \tag{1.15}$$

This model translates from domain $\mathcal{X}$ to domain $\mathcal{Y}$ and back, such that the outputs in domain $\mathcal{Y}$ look realistic according to a domain $\mathcal{Y}$ discriminator, and vice versa for domain $\mathcal{X}$, *and* the mappings are cycle-consistent: one complete cycle from $\mathcal{X}$ to $\mathcal{Y}$ and back should arrive where it started (and, again, vice versa). A diagram for this whole process is given in Figure 1.7.

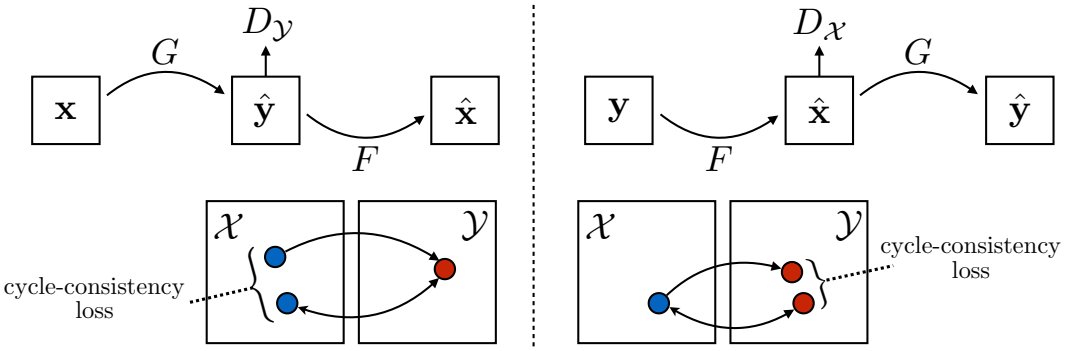


Figure 1.7: CycleGAN schematic (figure derived from [Zhu et al. 2017]).

1.6 Image-to-text and text-to-image

In the previous section we saw how to translate between different image domains. Could we be even more ambitious and translate between entirely different data *modalities*? It turns out that this is indeed possible and is an area of growing interest. For any type of data X, and any other type Y, there is, increasingly, a method X2Y (and Y2X) that translates between these two data types. Go ahead and pick your favorite data types – X,Y could be images and sounds, music and lyrics, amino acid sequences and protein geometry – then google "X to Y with a deep net" and you will probably find some interesting work that addresses this problem. X2Y systems are powerful because they forge links between disparate domains of knowledge, allowing tools and discoveries from one domain to become applicable to other linked domains.

One of the most popular and useful X2Y systems is image-to-text. This is a useful ability because text (i.e. human language) is a ubiquitous and general-purpose interface between almost all domains of human knowledge (we use language to describe pretty everything we do and everything we know). So, if we can translate images to text then we link imagery to essentially all other domains of knowledge.

It's also possible to do translation in the opposite direction: text-to-image.

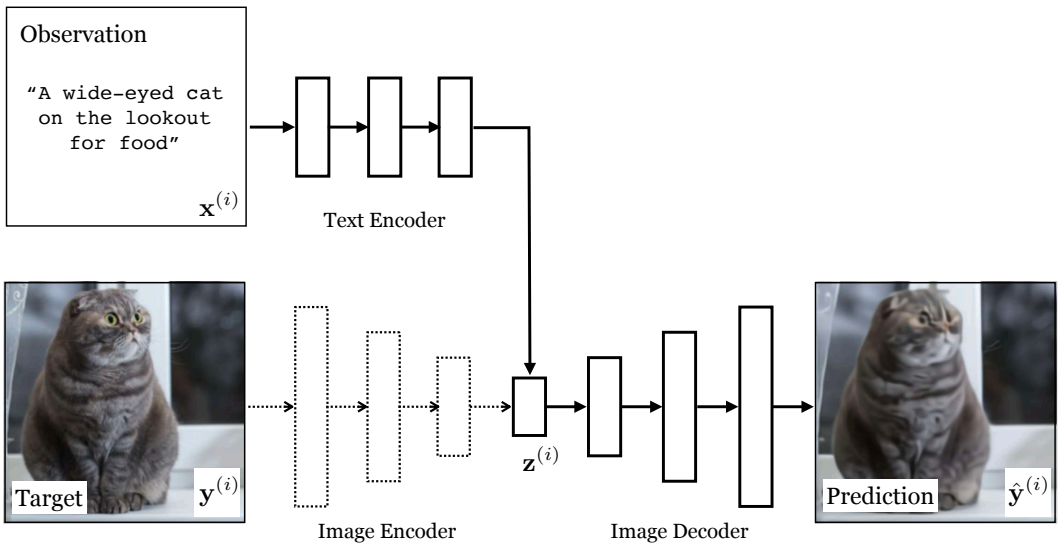


Figure 1.8: DALL-E training (cat image from [Ramesh et al. 2021]).

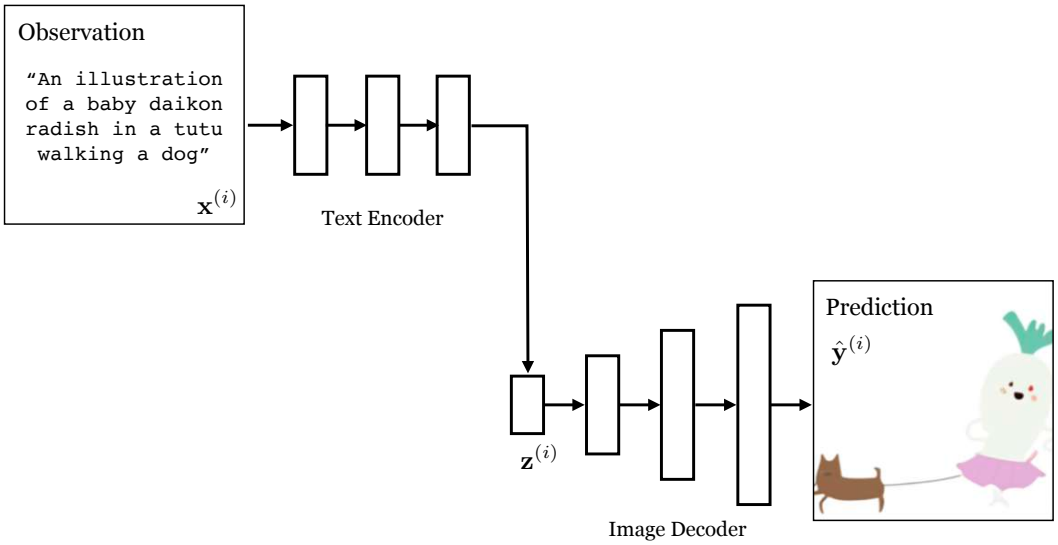


Figure 1.9: DALL-E training (text and generated image from [Ramesh et al. 2021]).

1.7 Concluding remarks

Many problems in vision can be phrased as structured prediction, and conditional generative models are a great approach to these problems. They provide a unified and simple solution to a diverse array of problems. A current trend in computer vision (and all of AI) is to replace bespoke systems for specific kinds of structured prediction – object detectors, image captioning systems, etc – with general-purpose conditional generative models. This is one reason why this book does not focus much on special-purpose systems that solve specific problems in vision – because the trend is toward general-purpose modeling tools.

Bibliography

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.

A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *European conference on computer vision*, pages 649–666. Springer, 2016.

J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.