# Training Data Generation

Let's generate training data for the semantic image segmentation model.

We'll cover the following



## Generating training examples #

Autonomous driving systems have to be extremely robust with no margin of error. This requires training each component model with all the possible scenarios that can happen in real life. Let's see how to generate such training data for performing semantic segmentation.

### Human-labeled data #

First, you will have to gather driving images and hire people to label the images in a pixel-wise manner. There are many tools available such as Label Box (https://labelbox.com/product/image-segmentation) that can facilitate the pixel-wise annotation of images.


Manual labelling of driving images to generate training data for semantic segmentation

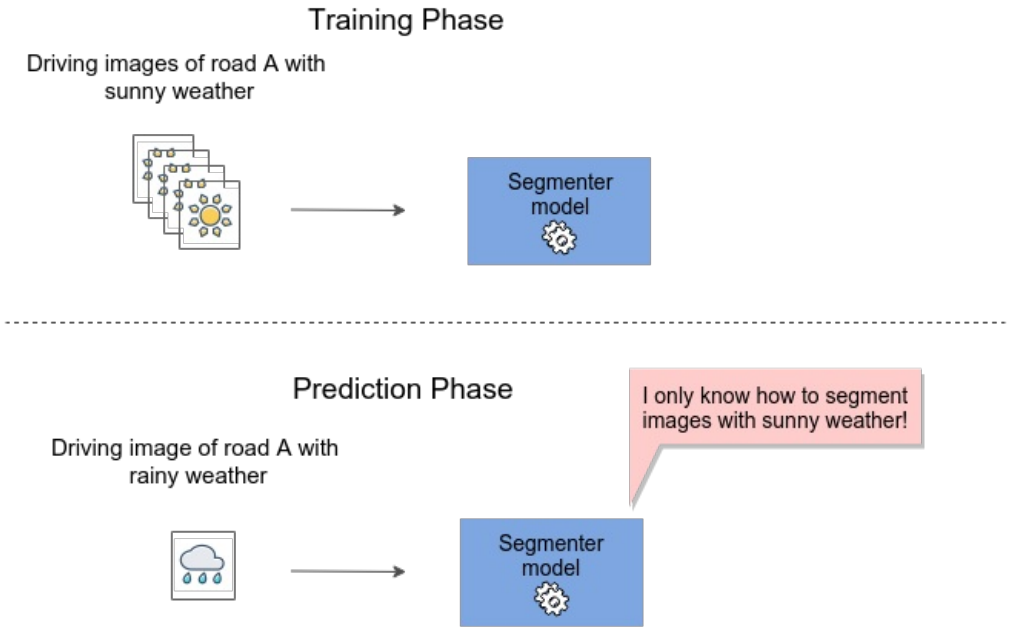
All the other parts of the image are also labelled similarly.

### Open source datasets #

There are quite a few open-source datasets available with segmented driving videos/images. One such example is the "BDD100K: A Large-scale Diverse Driving Video Database" (https://bair.berkeley.edu/blog/2018/05/30/bdd/). It contains segmented data for full frames, as shown with an example below.

BDD100K: full frame segmentation

These two methods are effective in generating manually labeled data. In most cases, your training data distribution will match with what you observe in the *real-world scene* images. However, you may not have enough data for al the **conditions** that you would like your model to make predictions for such as snow, rain, and night. For a self-autonomous vehicle to be perfect, your segmenter should work well for all the possible weather conditions, as well as cover a variety of obstacle images in the road.




A segmenter trained on a certain type of images won't be able to handle unseen conditions reliably

The sunny vs rainy condition is just one example; there can be many such situations.

It is an important area to think about how you can give your model *training data* for all conditions. One option is to manually label a lot of examples for each scenario. The second option is to use powerful data augmentation techniques to generate new training examples given your human-labeled data, especially for conditions that are missing in your training set. Let's discuss the second approach, using generative adversarial networks (GANs).

## Training data enhancement through GANs #

In the big picture, your self-driving system should compete with human intelligence when it comes to making decisions and planning movements. The segmenter can play its role in creating a reliable system by being able to accurately segment the driving scene in any condition that the vehicle experiences.

Achieving this target requires a diverse set of training data that covers all the possible permutations of the driving scene. For instance, assume that your dataset only contains ten-thousand driving images in the snowy Montreal conditions and fifty-thousand images in the sunny California conditions. You need to devise a way to *enhance your training data* by converting snowy conditions of the ten-thousand Montreal images to sunny conditions and vice versa.

The target includes two parts:

1. Generating new training images
2. Ensuring generated images have different conditions (e.g., weather and lighting conditions).

For the first part, you can utilize GANs. They are deep learning models used for *generation of new content*. However, for the second part, i.e., to generate a different variation of the original segmented image, a simple GAN would not work. You would have to use **conditional generative adversarial networks** (cGANs) instead. Let's see why.

Minimize

# GANs

Generative adversarial networks belong to the set of *generative models* in contrast to discriminative models, i.e., they focus on learning the unknown probability distribution of the input data to **generate** similar inputs rather than just classifying inputs as belonging to a certain class.

> For instance, given panda images, a generative model will try to understand how pandas look like (learn the probability distribution) rather than trying to accurately predict if an image is of a panda or another animal. This allows you to *generate new panda images*.

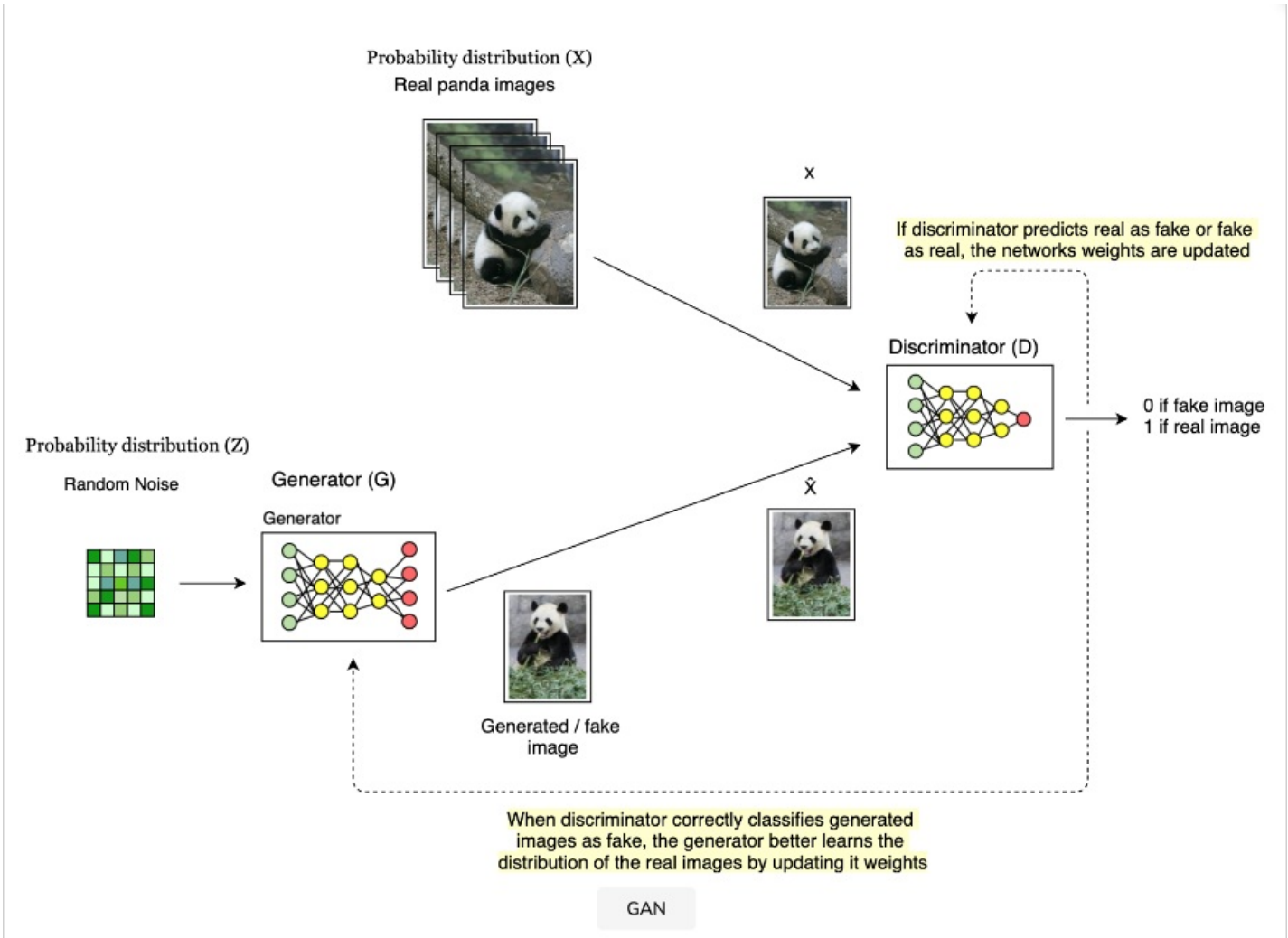
GANs are made of two deep learning models that compete against each other (hence, the term adversarial):

1. Generator (Forger)

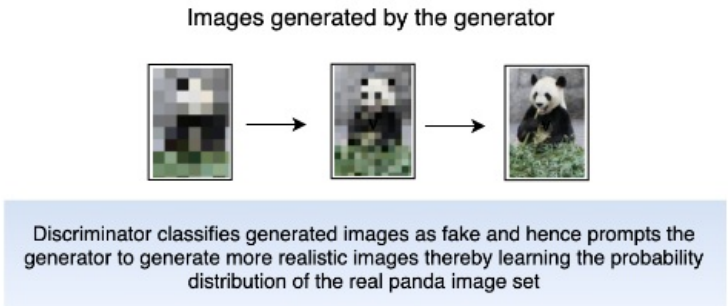
   This neural network receives a sample from a random distribution $Z$ (noise). It transforms it into something (a panda) that looks like it came from the target/true distribution(panda images). This is done by learning a mapping between the input samples and the images in the true distribution, during the training phase.

2. Discriminator (Detective)

   This neural network receives images from the true distribution $X$ as well as the new/fake images $\hat{X}$ generated by the generator. The goal is to discriminate between real and generated images. The network is trained to output $0$ for a generated image and $1$ for a real image.




The generator's tries to fool the discriminator into thinking that its generated images belong to the true distribution. On the other hand, the discriminator tries to catch all the fake/generated images. This encourages the generator to learn the true probability distribution more closely and generate such real looking images that the discriminator's output converges to $0.5$, i.e., it is not able to distinguish fake from real.




The generator's output improves throughout the training iterations.

## Supervised or unsupervised learning?

You saw that the generator model of the GAN learns the target probability distribution (looks for patterns in the input data) to generate new content; this is known as *generative modeling*.

Generative modeling is an unsupervised task where the model is not told what kind of patterns to look for in the data and there is no error metric to improve the model. However, the *training process of the GAN is posed as a supervised learning problem* with the help of a discriminator. The discriminator is a supervised

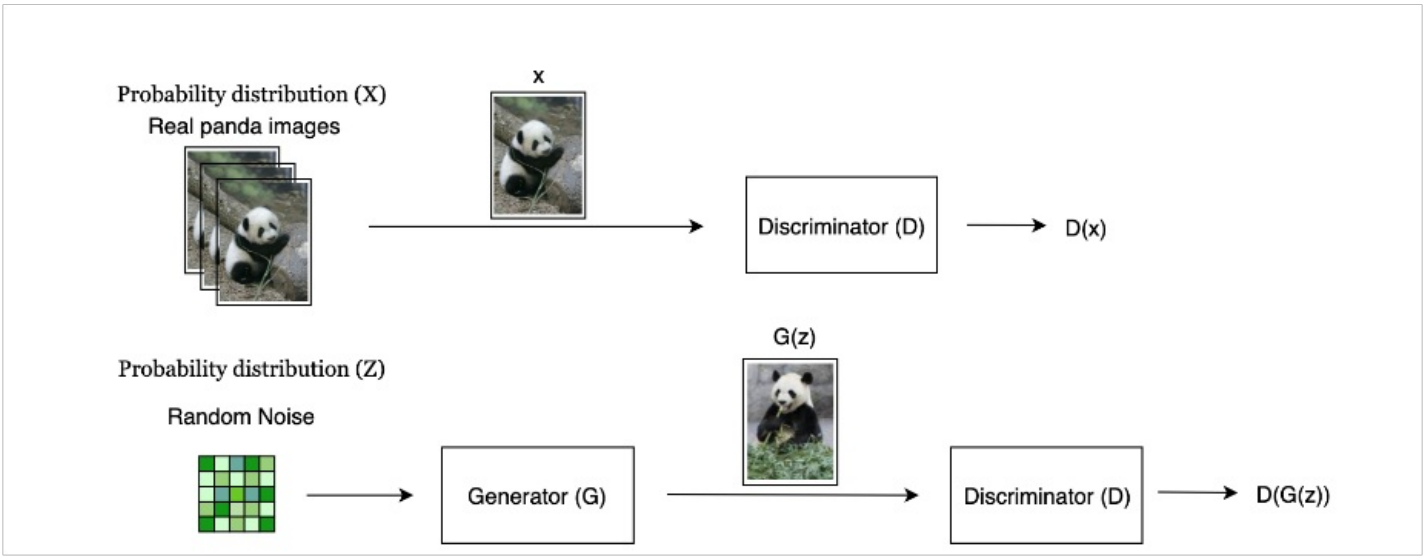
classifier that distinguishes between real and generated images. This allows us to devise a **supervised loss function for the GAN**.

## Loss Function

The *binary cross entropy loss function* for the GAN is as follows:

$$min_G\ max_D\ \{E_{x\sim p_{data}(x)}[log\ D(x)] + E_{z\sim p_z(z)}[log(1 - D(G(z)))]\}$$

Its made up of two terms. Let's see what each part means (This is the combined loss for both discriminator and generator network).




- x is an image from the true distribution
- z is the random input to the generator
- $p_{data}(x)$ is the probability distribution of the real images.
- $p_z(z)$ is the probability distribution of the generated images.
- G(z) is the generator's output when given noise z.
- D(x) is the discriminator's probability (judgement) that the *real data instance x* is real.
- D(G(z)) is the discriminator's probability (judgement) that the *generated instance x* is real.
- $E_x$ is the expected value over all real data instances.
- $E_z$ is the expected value over random inputs to the generator.

> The expected value of a variable X is the average of all values of X weighted by their occurrence probability.

The GAN's objective is to replicate the $p_{data}$ with $p_z$. Binary cross entropy helps to measure the distance between these two distributions. Let's see how this works.

The term *D(x)* is the probability that a data point belongs to *class real*, whereas the term *1-D(G(z))* is the probability of belonging to *class fake*. The loss function works by measuring how far away from the actual value (real 1 or fake 0) the prediction is for each class and then averages the errors (Expectation) to obtain the final loss.

## Discriminator's aim

The discriminator wants to distinguish between real and fake. Earlier, you mentioned the final form of the loss. Let's see how the discriminator's loss function is derived from the original form of the binary cross-entropy loss as follows:

$$Loss(\hat{y}, y) = [ylog\hat{y} + (1-y)log(1-\hat{y})]$$

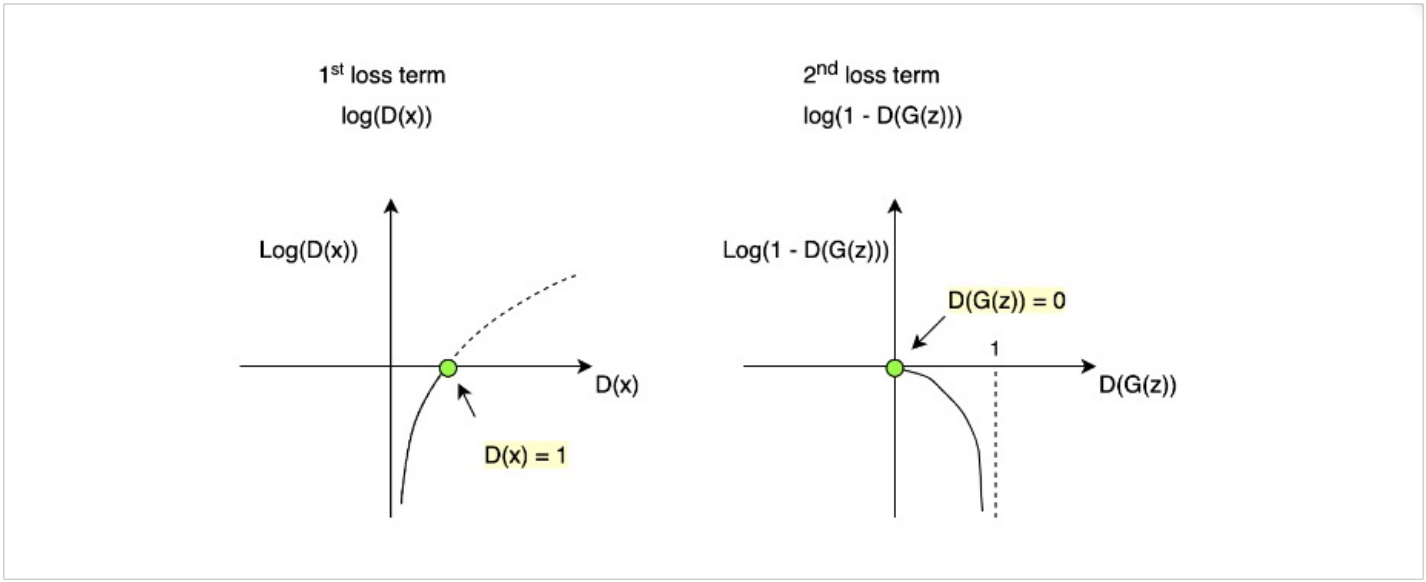
When you have the image coming from the true distribution, the function takes the form:

$$\begin{aligned} Loss(D(x), 1) &= [1 * log(D(x)) + (1-1) * log(1 - D(x)] \\ &= log(D(x)) - (1) \end{aligned}$$

When you have the image coming from the generator, the function takes the form:

$$\begin{aligned} Loss(D(G(z)), 0) &= [0 * log(D(G(z)) + (1-0) * log(1 - D(G(z))] \\ &= log(1 - D(G(z)) - (2) \end{aligned}$$

The terms (1) and (2) are added to achieve the final loss. To see why the discriminator maximises this loss ( $max_D$ part), you need to plot both terms.



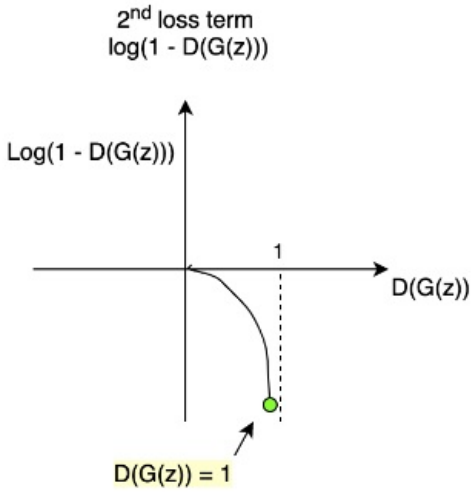

You can observe that the discriminator needs to maximize the $1^{st}$ term in order to correctly predict "1" for the real input image. Also, it needs to maximize the second term to correctly predict "0" for the generated input image.

## Generator's Aim

The generator aims to generate such an image that the discriminator classifies it as real or 1. The generator's loss function is the same as when the discriminator receives generated image, i.e.,

$$Loss(D(G(z)), 0) = log(1 - D(G(z))$$

However, unlike the discriminator, the generator minimizes this term . Let's look at the plot again.




As you can see in the log plot of the second term, in order to fool the discriminator (D(G(z))=1), you need to minimize the term.

## Training the GAN

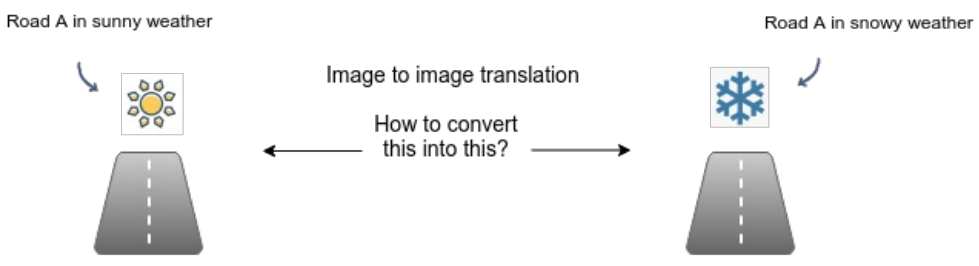
The generator and discriminator are trained alternatively.

For training the discriminator, the weights of the generator model are frozen. Two mini-batches are taken. One is from the real images data. The other consists of generated images obtained by feeding random noise samples to the generator. There are no labels associated with this data. However, the discriminator still learns due to the formulation of the loss function, which that we discussed earlier. The discriminator updates its weights to *maximize* the terms leading to fake images being classified as 0 and real as 1.

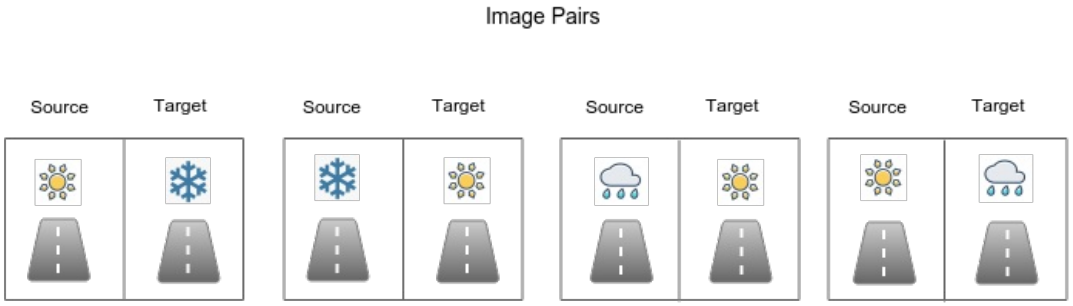
For training the generator, the weights of the discriminator model are frozen. A mini-batch is taken from the noise. Once again there are no labels, but the model updates its weights to *minimize* the loss term, leading to fake images being classified as 1.

If you train a GAN on a set of driving images, you do not have any control over the generator's output, i.e., it would just generate driving images similar to the images present in the training dataset. However, that is not particularly useful in your case.

Therefore, we want to perform **image-to-image translation** (a kind of data augmentation) to enhance our training data, i.e., you want to map features from an input image to an output image. For instance, you want to learn to map driving images with one weather condition to another weather condition, or you may want to convert day time driving images to night time images and vice versa. This can be done with a cGAN which is a deep neural network that extends the concept of GAN.



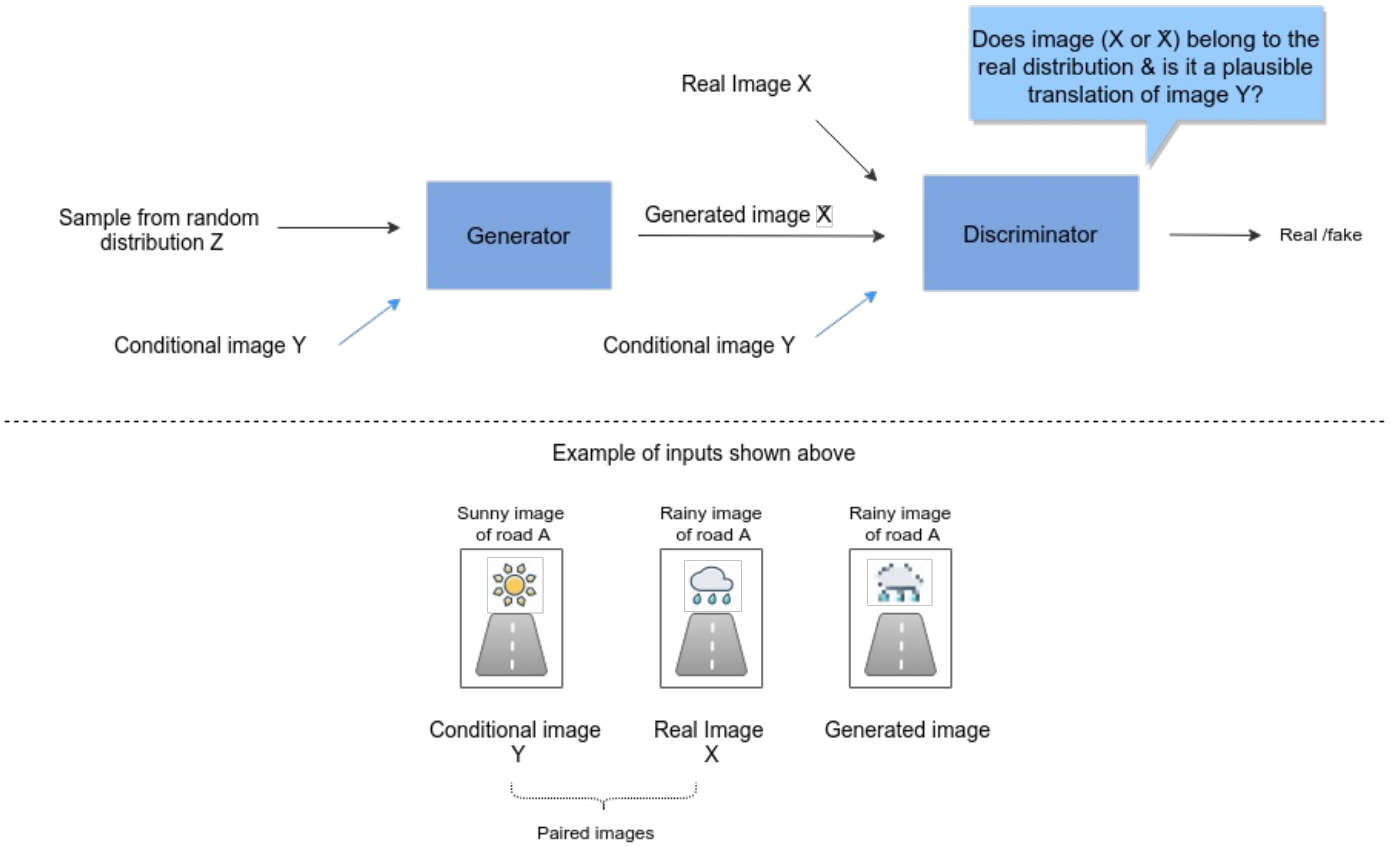
For image translation, your training data should consist of paired images. For example, image pairs could be of the same road during day and night. They could also be of the same road during winter and summer.




Training data for cGAN

In cGAN, you give the source image that you want to translate to the generator, along with an additional input (target image as a condition). This will guide the generator's output image to be of the same place but with a different weather/light condition.

You will also feed the source image to the discriminator so that apart from its usual task of discriminating between real and generated images, it will also see if the given image is a plausible translation of the source image.




cGAN: both generator and discriminator are conditioned on y (image to be translated)

The loss function for cGAN is as follows:

$$min_G\ max_D\ \{E_{x \sim p_{data}(x)}[log\ D(x|y)] + E_{z \sim p_z(z)}[log(1 - D(G(z|y)))]\}$$

To understand this loss function you can refer to the expandable section on GANs above. You see that the cGAN loss function differs from the GAN loss function as the discriminator's probability D(x) or D(G(z)) is now conditioned on y.

Once you have trained the cGAN for the image translation task, you can now feed it the driving images you manually collected through the generator model to enhance the training data.

Using a similar approach, you can build multiple cGAN models for different conditions, including generating night images, cloudy images, snow images etc., and enhance our training data set with examples of all conditions.

> This chapter does not have a feature engineering lesson, because the convolutional layers (in the CNN) extract features themselves.

## Targeted data gathering #

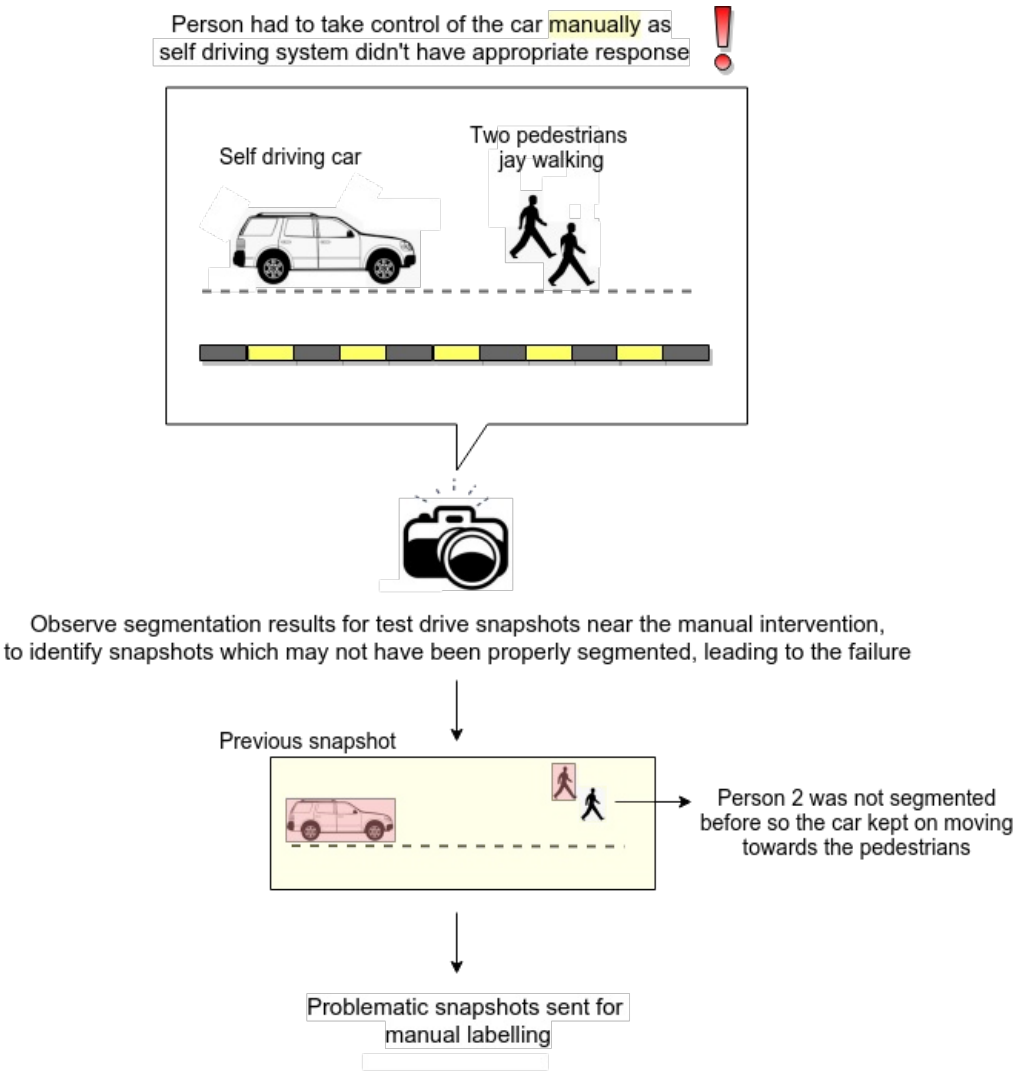
Earlier in the chapter, we discussed manual intervention as a metric for our end-to-end self-driving car system. You can use this metric to identify scenarios where your segmentation model is not performing well and learn from them.

> Here, you aim to gather training data specifically on the areas where you fail while testing the system. It will help to further enhance the training data.

You will test the self-driving car, with a person at the wheel. The vehicle has the capability to record the video of this test drive. After the test drive is over, you will observe the system and look for instances where it did not perform well and the person had to intervene.

Consider a scenario where you observe that the person had to manually intervene when two pedestrians were jaywalking. You will check the performance of the segmentation model on, let's say, the last twenty frames before the manual intervention took place. You might see that segmentation failed to predict one of the two pedestrians. You will select snapshots of those instances from the recording and ask the manual labelers to labels those snapshots since it means that the training data lacks such examples.

You will then focus on collecting such examples and ask the manual labelers to label them.




Why you need targeted data gathering

After this, you can also apply data augmentation on the newly gathered training examples to further improve your model's performance.

---


← Back

Architectural Components

Next →

Modeling

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/training-data-generation__self-driving-car-image-segmentation__grokking-the-machine-learning-interview)