Representations and Representation Learning

Bryan Chiang and Jeannette Bohg

February 7, 2022

1 Overview

1.1 States and Representations

The **state** of a dynamical system is basically a compressed description and captures the key aspects of this system. These aspects may be time-varying or fixed parameters. Consider the system shown in Figure 1: a walking robot moving along a vertical plane. In this example, the time-varying quantities may consist of information about the robot's current pose (position and orientation) in space, joint configuration and velocity. Fixed parameters may be the length of the robot's limbs, the weight of each limb or joint friction. Numerous representations are possible for the system's state; for instance, we could choose to use any selection of joint angles, velocities, positions or the pose of the robot. Furthermore, these quantities could be represented in different ways. For example, orientations of the robot body could be given as Euler angles, in the angle-axis representation or as a quaternion. Positions could be provided as spatial or polar coordinates. Which representation you choose often depends on the task at hand. Finally, a state representation could also be learned which we will come back to later.

The state of a **dynamical** system is not static and evolves over time. For the example of the walker, its pose and joint configuration are time-varying. We denote the state at time t as x_t. A common assumption for such dynamical

Figure 1: Walker2D robot from the DeepMind Control Suite [4]

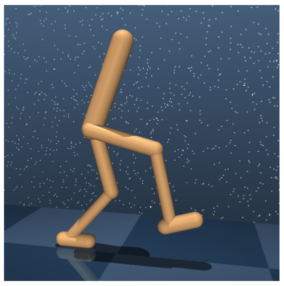


Figure 2: Probabilistic Graphical Model of a Markov chain.

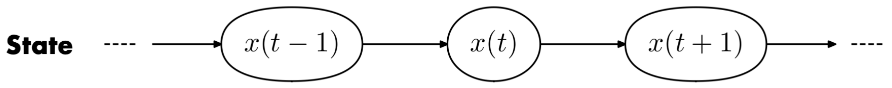
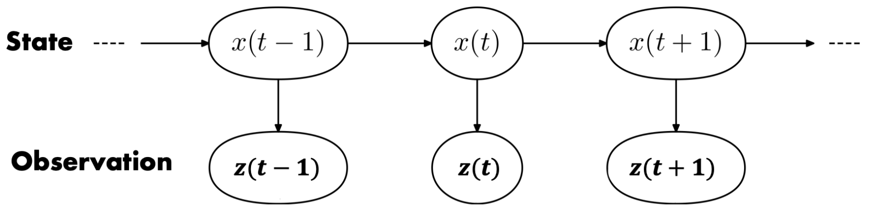


Figure 3: Probabilistic Graphical Model of a Hidden Markov Model.



systems is that they possess the **Markov property**: future states only depend on the current state, not past states. In other words, state x_{t+1} is conditionally independent of past states $x_1, x_2, \ldots, x_{t-1}$ given the current state x_t:

$$P(x_{t+1}|x_t, x_{t-1}, \ldots, x_2, x_1) = P(x_{t+1}|x_t). \tag{1}$$

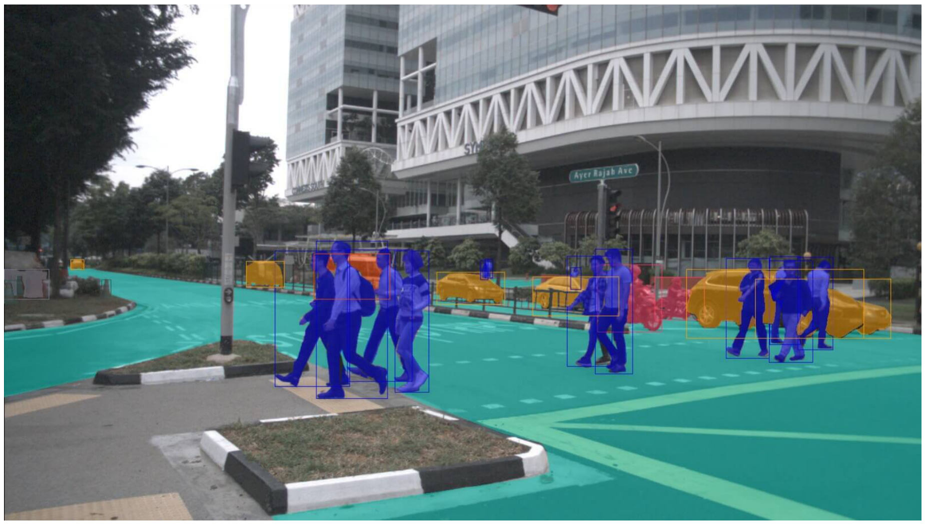
The Markov property has implications when predicting future state. If this property holds, future states only depend on the present - not on the past. Therefore, we can predict the state at the next timestamp given the current state according to specific dynamics of the system, or $f(x_t) = x_{t+1}$. For instance, if the state contains the current angular velocity of a joint, we can predict the joint angle at the next time step. $f(\cdot)$ may be a non-linear function, and we often assume that the dynamics are constant over time for simplicity. We model such systems as **Markov chains**. Figure 2 visualizes the conditional dependence structure of a Markov chain as a probabilistic graphical model in which the nodes are the random variables and the edges indicate conditional dependence.

Markov chains model dynamical systems as fully observable, i.e. the state is known. In practice, the state of a system at time t is often unknown because it cannot be directly observed. Our job is to estimate the true state from noisy sensor observations: a process called **state estimation**. Such systems are partially observable and can be modelled by **hidden Markov models** (or HMMs) since the state is "hidden". We assume that the states x_t behave as a Markov process. We also assume that observation z_t at time t only depends on the state x_t at that time. In other words, the observation z_t is conditionally independent of all previous states and observations given x_t:

$$P(z_t|x_t, x_{t-1}, \ldots, x_2, x_1) = P(z_t|x_t). \tag{2}$$

This means that the observation model $h(\cdot)$ that predicts the current observation z_t only depends on x_t: $h(x_t) = z_t$. Let us make this concrete with

Figure 4: RGB image observation of a street scene taken from the point of view of a car from NuScenes [3]. Pedestrians, cars, motorbikes and the ground are segmented and detected.



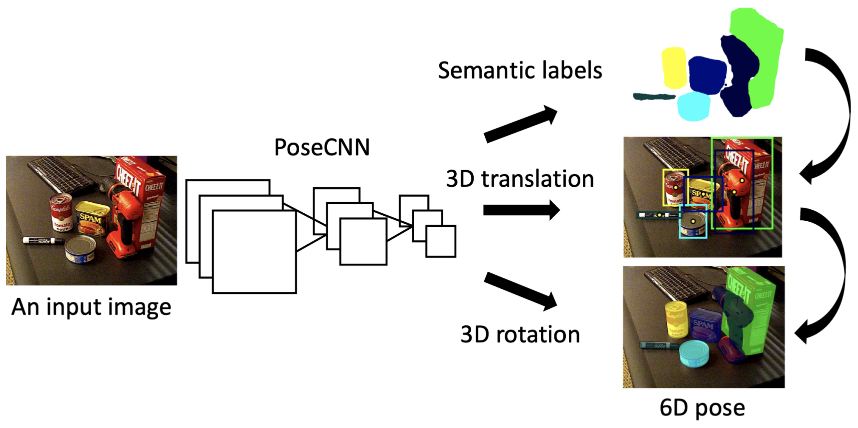
a vision example. If we are a self-driving car with the point-of-view shown in Figure 4, we would want to know the state of all the traffic participants to avoid any collisions: 3D locations, bounding boxes (size), possibly velocities of everyone. However, this information is not given to us directly and must be estimated from the sensor observations captured through for example a pair of stereo images, RGB frames, or LIDAR data.

In future lectures on optimal estimation, we will account for robot actions or control inputs in these Markov processes that affect how the state evolves. Specifically, instead of $f(x_t) = x_{t+1}$, we will have $f(x_t, u_t) = x_{t+1}$, where u_t is the control input or action taken by the robot at time t. These could be torque commands sent to a joint motor that moves the legs of the walker in Figure 1 or steering and throttle in case of a car. An interesting problem is how to decide the next best action u_t. This is called decision making. Markov chains generalize to **Markov decision processes (MDPs)**, and hidden Markov models generalize to **partially observable Markov decision processes (POMDPs)** that are used to formalize decision making. If you are interested in that subject, you should consider taking *Decision-Making under Uncertainty* (AA228/CS238) at `https://web.stanford.edu/class/aa228/cgi-bin/wp/`.

1.2 Generative and Discriminative Approaches

How can we estimate the state from observations? Let us consider the task of estimating an object's pose (state x) from an input RGB image (observation z). A 6D pose refers to both the objects 3D translation and 3D rotation (6

Figure 5: PoseCNN [6] is an example of a discriminative model.



parameters total).

We describe two approaches at a high-level and will delve into specifics later in the course. First, a generative model describes the joint probability distribution $p(z, x)$ given the observation z and state x. We compute this joint distribution using Bayes rule with the likelihood $p(z|x)$ (see Eq. 2) and prior $p(x)$. For our example of object pose estimation, we could sample an object pose $x^{(i)}$ from $p(x)$. This prior can be as simple as a uniform distribution in space or more complex and capture likely locations of this object (e.g. cars on roads rather than on the side of buildings). Given this pose sample, we can use the observation model to generate the most likely observation $h(x^{(i)}) = z^{(i)}$, i.e. the 2D projection of the object in the sampled pose onto the image plane. $p(z|x^{(i)})$ then provides a likelihood of the actual observation z given the hypothesized pose $x^{(i)}$ by comparing the differences between $z^{(i)}$ and z.

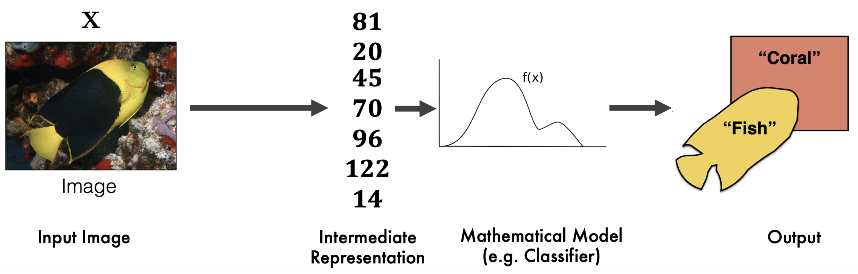
Second, a discriminative model describes the conditional probability $p(x|z)$ of the state x given the observation z. For instance, we could train a neural network, such as PoseCNN in Figure 5, to directly map an input image z to the most likely output pose x. To learn more about the differences between discriminative and generative models, please refer to the CS229 notes at `https://cs229.stanford.edu/notes-spring2019/cs229-notes2.pdf`.

2 Representation Learning

2.1 Representations in Computer Vision

We turn our attention to how the term *representations* is used in Computer Vision. Consider the high-level, semantic segmentation pipeline in Figure 6. It illustrates that data at any stage within that pipeline comes in a specific repre-

Figure 6: Visualization of input, intermediate, and output representations in a high-level computer vision pipeline for semantic segmentation [2].


X
Image
Input Image
81
20
45
70
96
122
14
Intermediate Representation
f(x)
Mathematical Model (e.g. Classifier)
"Coral"
"Fish"
Output

sentation. We will therefore introduce context to qualify the term *representation* more clearly.

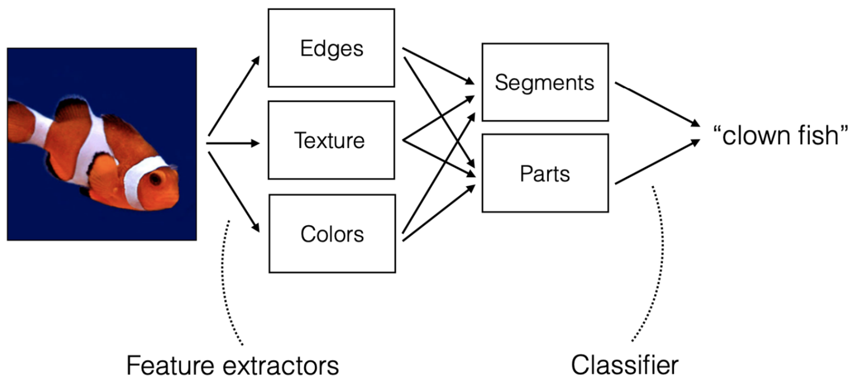
On one end of the pipeline, we have raw sensor data captured by some sensor. The **input representation** describes the raw sensor data format. In Figure 6, our input is an observation of a 3D scene within the ocean containing a fish. Depending on sensor choice, raw sensor data could be represented by 2D images, depth images, or point clouds. Even within a specific form, there are subtle differences in the input representation. For instance, color vs. grayscale, stereo vs. monocular, RGB vs. HSV image color space. On the other end of the pipeline, the information being inferred from the raw sensor data is provided in an **output representations** that succinctly describes the high-level key aspects of the scene that are necessary for the considered decision-making task. If we were a biologist and our goal was to count the number of fish we encounter, the relevant output representation may be the "Fish" label on the output side, or even a more specific label indicating the species of the fish. However, if we were a drone attempting to navigate the coral reef, we may be more interested in estimating the 6D pose and 3D bounding box of the fish, which would be useful to incorporate into our trajectory planning. In between input and output representations is the input data in some **intermediate representations**. This representation is typically a compressed, low-dimensional vector that summarizes the high-dimensional input sensory data. Intermediate representations compress the input data provided in the input representation. The data provided in the intermediate representation is then used to derive a relevant quantity in the output representations.

Some natural questions at this point are:

1. Since representations are so key for decision-making, what makes for a good representation? Bengio et al [1] proposed the following requirements for good representations.

    - A representation should be *compact*, i.e. minimal but yet
    - *explanatory*, i.e. expressive enough (sufficient) to represent and capture a large number of possible input configurations.

Figure 7: Classical CV pipeline for extracting hand-designed intermediate representations of the input data [2]. There were typically layers of intermediate representations where e.g. low-level features like short edges are grouped into high-level features like parts.



- A representation should be *disentangled*, i.e. the different explanatory factors of the input data variations should be represented independently to reflect that these factors can change independently of each other.
- A representation should also be *hierarchical* so that more abstract concepts can be explained in terms of less abstract ones, allowing feature re-use and increasing computational efficiency.
- Ultimately, this representation should *make downstream inference, prediction or decision-making problems easier.* Therefore, the quality of a representation can also be quantified by downstream performance.

2. How do we actually obtain intermediate and output representations? This will be discussed in the following sections.

2.2 Traditional CV and Interpretable Representations

A high-level visualization of the traditional ($\approx$ pre-2012) computer vision pipeline is shown in Figure 7. The input data is compressed into an intermediate representation which used to be combinations of hand-crafted features extracted from the input that comes in a specific input representation. The specific choices of features (descriptors) is flexible and can eb made task specific. For instance, if we believe that specific stripes or designs on a fish's body may distinguish it from other types of fish or things in the sea, then we could exploit those specific patterns by extracting edges and colors to form our intermediate representation of the raw input data. In the example shown in Figure 7, the representation of the output is a class label. To infer this class label, the intermediate representation is fed into a classifier that can either be learned or is based on more manually-defined heuristics. As a simple example, I can define a heuristic that if the histogram of colors in the image contains a high amount of oranges and whites, we may say that the image is likely to contain a clownfish. Much classical literature has focused on developing new feature extractors, often based on image processing and filtering methods. While specific methods are out of this class's scope, we provide several references for further reading at the end of this document. The primary benefit of classical feature extraction methods is that they result in **interpretable representations**: because we designed these methods by hand, we can easily explain why specific output representations were chosen. This can be especially important for downstream tasks with high stakes, perhaps in legal or medical domains. However, the downside is that coming up with these feature extractors is a tedious process, requiring significant amounts of time and domain expertise that may not be available.

2.3 Modern CV and Learned Representations

Modern computer vision methods ($\approx$ 2012-present) replace manually extracted features with **learned intermediate representations** as visualized in Figure 8. One of the most common model architecture for this purpose are *Convolutional Neural Networks* (CNN) consisting of layers of convolutional filters applied to images. These filters are learned typically using labeled training data provided a learned intermediate representation of the input data. The learned representations turn out to be much more powerful than the previously hand-designed features in providing the essential information to the subsequent classifiers.

While learned representations have shown higher accuracy in downstream tasks such as for example image classification, the downside is that they lack the interpretability of classical representations. Several methods aim to interpret learned representations to understand why they perform so well and how to improve them. Zeiler and Fergus [7] analyzed the specific image patches that activate filters the most strongly for each layer in a CNN trained on ImageNet. What they discovered was that the learned representations shown in Figure 9 resembled the traditionally extracted features (such as edges, textures, body parts shown in Figure 7).

Another approach to understanding learned intermediate representations of the input data is projecting them to a lower dimensional space so that we can plot and interpret them. One popular technique for achieving this is tSNE [5], which performs dimensionality reduction on high-dimensional intermediate representations by minimizing the KL-divergence of joint probabilities (based on data similarity) between the low-dimensional embeddings and original high-dimensional representations. We expect that data points from the same class will be geographically clustered near each other as visualized in Figure 10. We can use tSNE to sanity check that our neural network indeed learns that images

Figure 8: Learned intermediate representations of input data [2]. Components of the intermediate feature representations are not manually designed, hence the blank boxes.

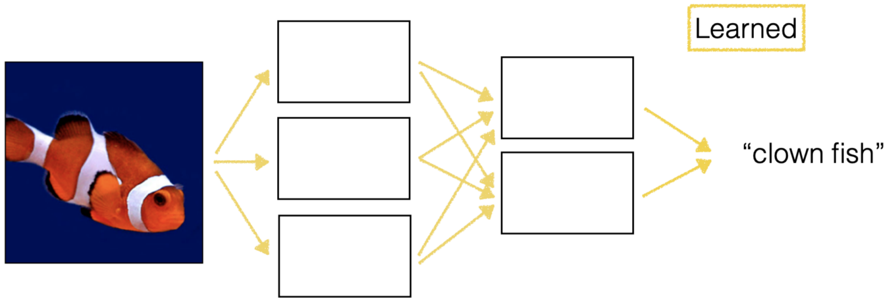


Figure 9: Different levels of learned intermediate representations represent those of a traditional pipeline from Figure 7.

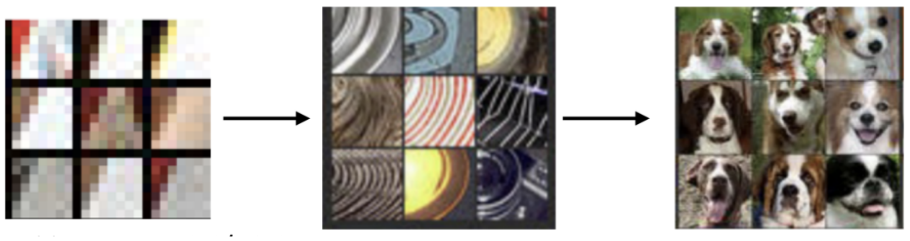
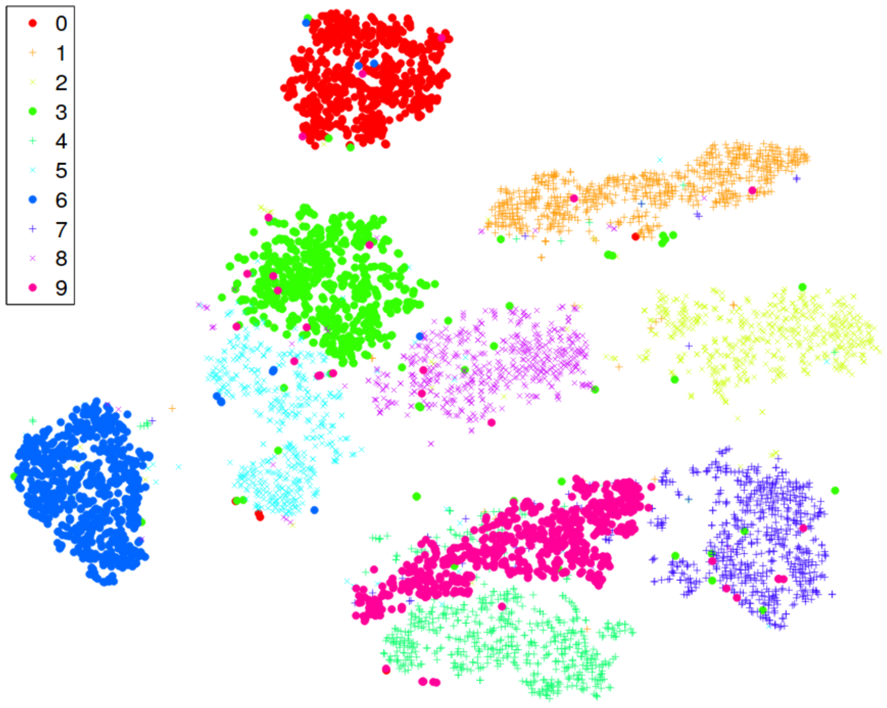


Figure 10: Visualization of tSNE embeddings on the MNIST dataset [2].

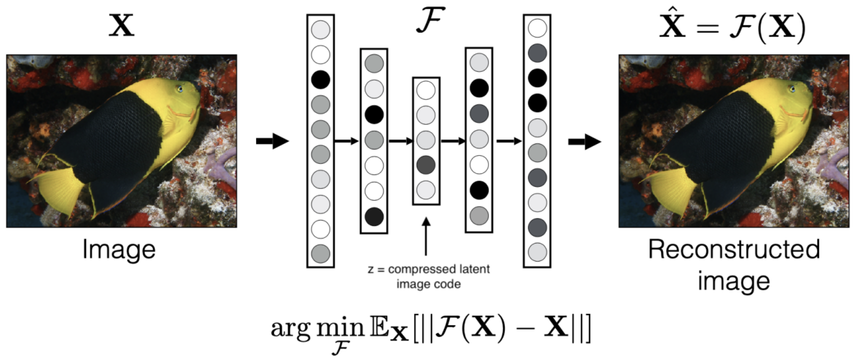


with the same label are also visually similar.

2.4 Unsupervised and Self-Supervised Learning

In a traditional supervised learning formulation, we can train a model for a specific inference task such that it minimizes a loss function (e.g. image classification accuracy or pose estimation) on a training dataset of D datapoints $\{(x_i, y_i)\}_D$, where x_i is the i^{th} datapoint in a given input representation (e.g. images or 3D point clouds) and y_i is the label in the output representation (e.g. object categories or 6D poses). In practice, data is abundant. However, labels are expensive to acquire and can require specialized knowledge. Can we learn meaningful representations from data *without* labels? Consider the **autoencoder** architecture in Figure 11. The goal of an autoencoder is to learn to perfectly reconstruct the input image. How well such an encoder is able to achieve this is measured by the reconstruction loss, which is defined as the difference between the input data $\mathbf{X}$ and the output $\mathcal{F}(\mathbf{X})$, both in the same representation. Here, the intermediate representation $\mathbf{z}$ of the input data is a lower-dimensional vector, and is located at the middle of the network in Figure 11. Intuitively, we expect that if the second half of the network $\mathcal{F}$ is able

Figure 11: Autoencoder architecture for unsupervised learning [2]. $\mathcal{F}$ is the autoenconder, $\mathbf{X}$ is the input image, $\hat{\mathbf{X}}$ is the reconstructed input image.



$\mathbf{X}$
Image
$\mathcal{F}$
z = compressed latent image code
$\hat{\mathbf{X}} = \mathcal{F}(\mathbf{X})$
Reconstructed image
$\arg\min_{\mathcal{F}} \mathbb{E}_{\mathbf{X}}[||\mathcal{F}(\mathbf{X}) - \mathbf{X}||]$

to reconstruct the input from $\mathbf{z}$, then $\mathbf{z}$ is a useful and informative compressed version of the input data. For this reason, we say that $\mathbf{z}$ is the **bottleneck**.

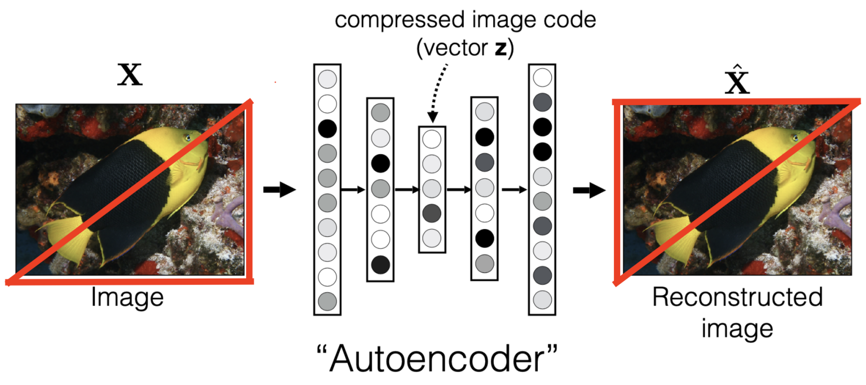
We say that an autoencoder performs **unsupervised learning** since it requires no external labels aside from the input image (which may be viewed as the label itself). In a similar vein, **self-supervised learning** masks out part of the input to use as the output label. For instance, as shown in Figure 12, we can keep all pixels below the image diagonal as the input, and all the pixels above the diagonal as the output.

Given the lower portion, the task would be to reconstruct the upper portion of the image. The hope is that the network would still learn meaningful representations about the input images to achieve this task. This has been empirically shown to some extent in the following manner. Given an autoencoder trained on a large amount of unlabeled data, we can obtain the compressed intermediate representation $\mathbf{z}$ with the first half of the network (also called the **encoder**). Then, with only a small amount of labeled data, we can train a classifier on top of the intermediate representation $\mathbf{z}$ to accurately predict the output representation since the autoencoder has already done the heavy lifting of learning a meaningful representation for the input.

References

[1] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, aug 2013.

Figure 12: Self-supervised learning setup. The goal is to use the lower triangle to reconstruct the upper triangle.



[2] Phillip Isola Bill Freeman. Representation learning. `http://6.869.csail.mit.edu/sp21/lectures/L14/14_rep_learning.pdf`, 2021. Illustrations used with permission. Accessed: 2022-01-31.

[3] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *CVPR*, 2020.

[4] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. Deepmind control suite. *CoRR*, abs/1801.00690, 2018.

[5] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[6] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*, 2017.

[7] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.