6CS012 – Artificial Intelligence and Machine Learning.
Lecture – 07
Representation Learning:
From Supervised {CNN} to Unsupervised {Autoencoder}.

Siman Giri {Module Leader – 6CS012}

1. What is Representation Learning?

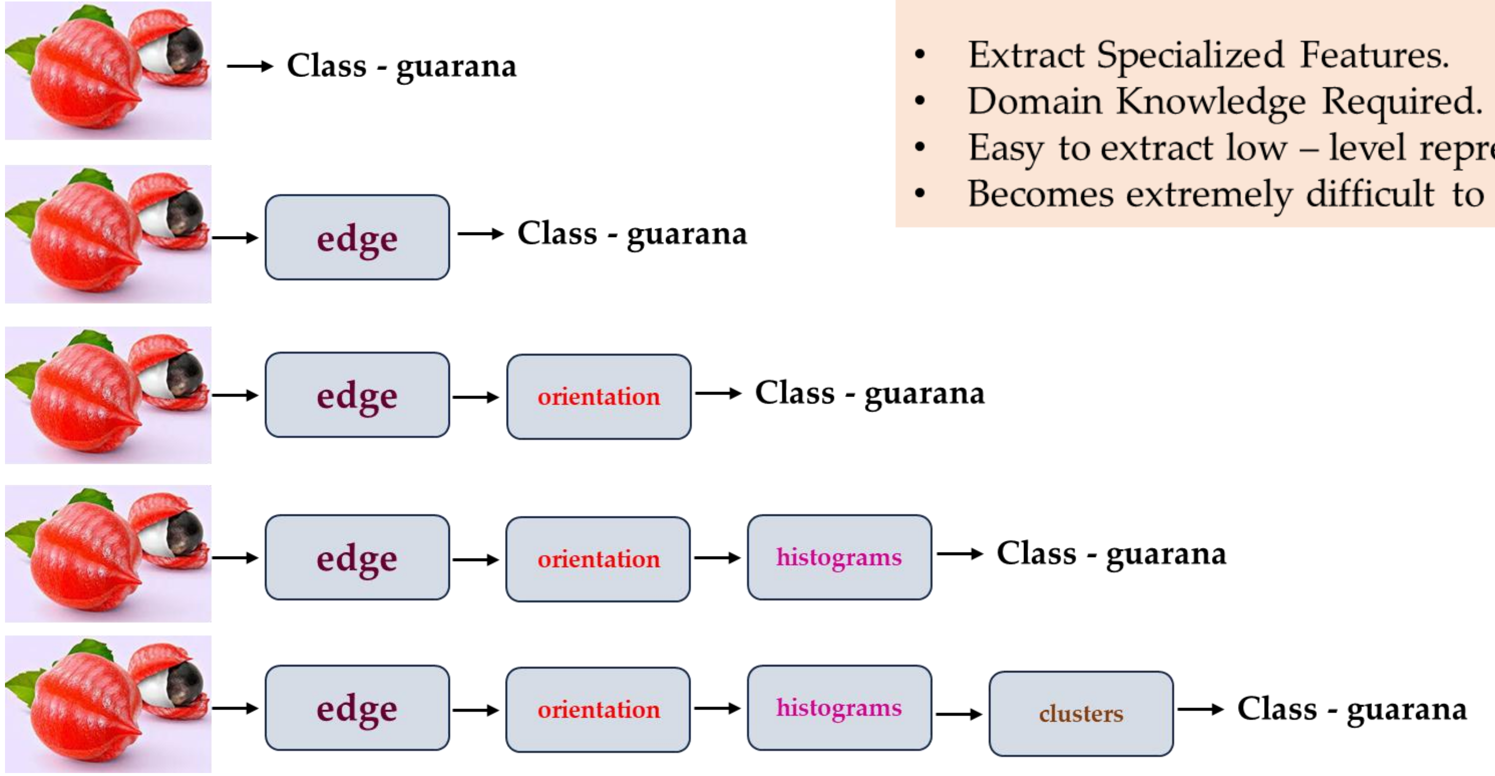
{ Deep Learning is a Representation Learning.}

1. Introduction.

- In any Learning Task, we need to learn:
  - A good set of features to represent your data,
  - A classifier on those features,
- If you design the features manually,
  - it is called Feature Engineering, very important for Machine Learning Algorithms.
- If you do it automatically,
  - it is called Feature learning or Representation Learning, which is a big part of any Deep Learning algorithms for example Convolutional Neural Network.

1.1 How to Represent Image?

- A Classical Computer Vision Perspective:

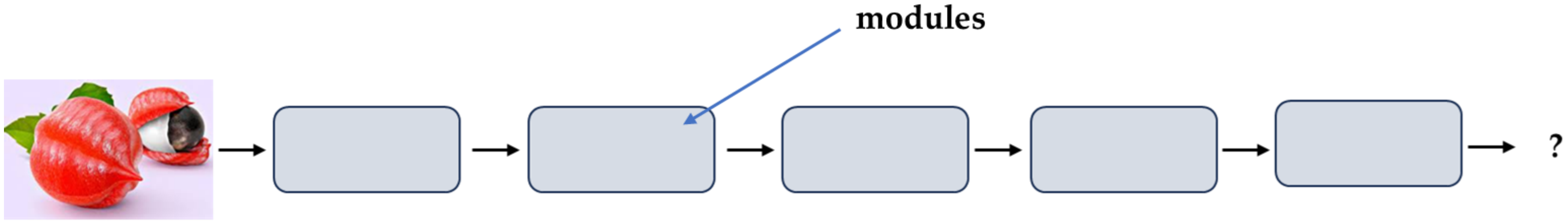


Challenges:

- Extract Specialized Features.
- Domain Knowledge Required.
- Easy to extract low – level representations but
- Becomes extremely difficult to design and extract high level representations.

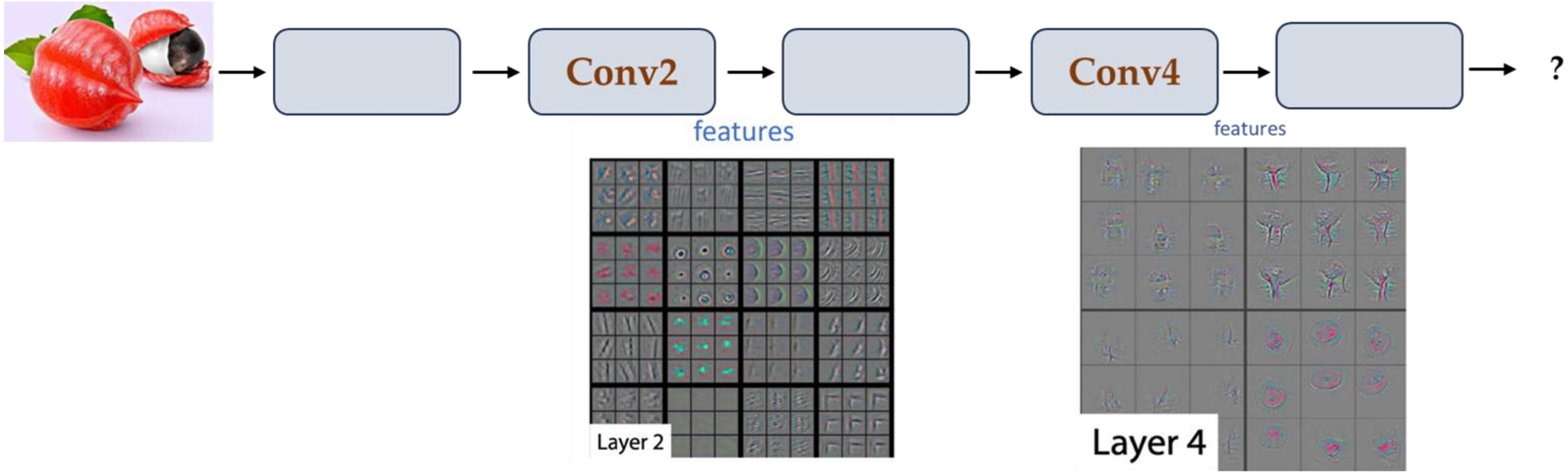
1.2 How to Represent Images with Deep Learning?

- Build a general modules instead of specialized features:
    - Compose simple modules into complex functions:



- Build multiple levels of abstractions.
- Learn from data.
- Reduce domain knowledge and feature engineering.

1.3 Multiple Levels of Representations.

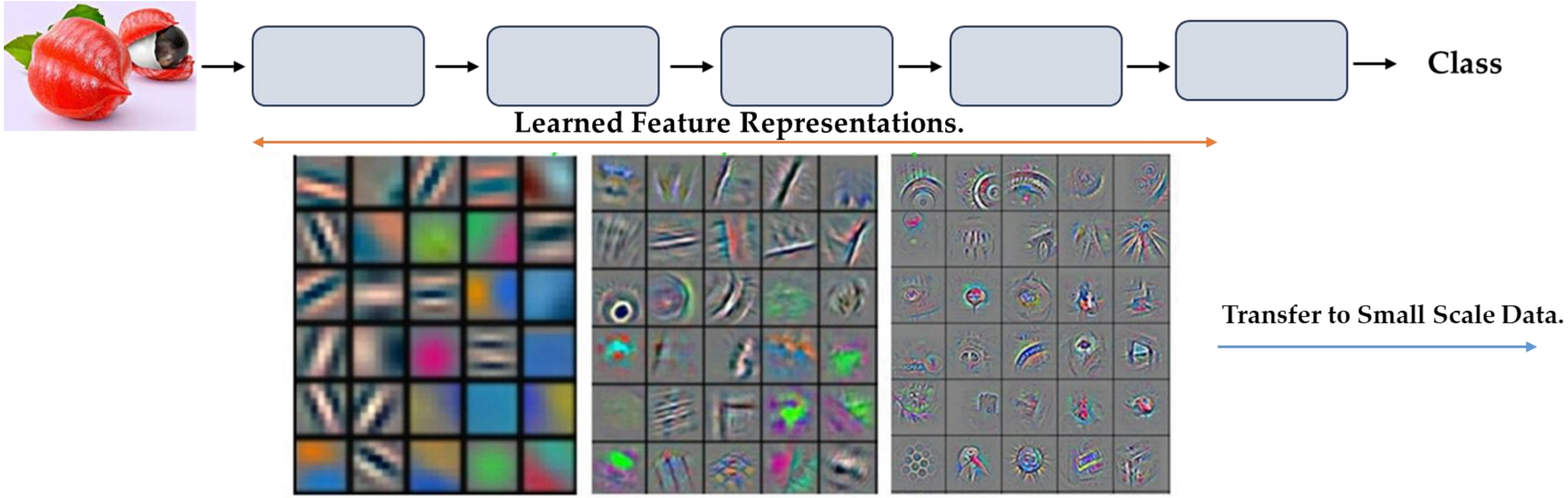


Deeper layers have "higher – level" features.

"visualizing and Understanding Convolutional Networks", Zeiler & Fergus, ECCV 2014.

1.4 Deep Representations are Transferrable: Transfer Learning.

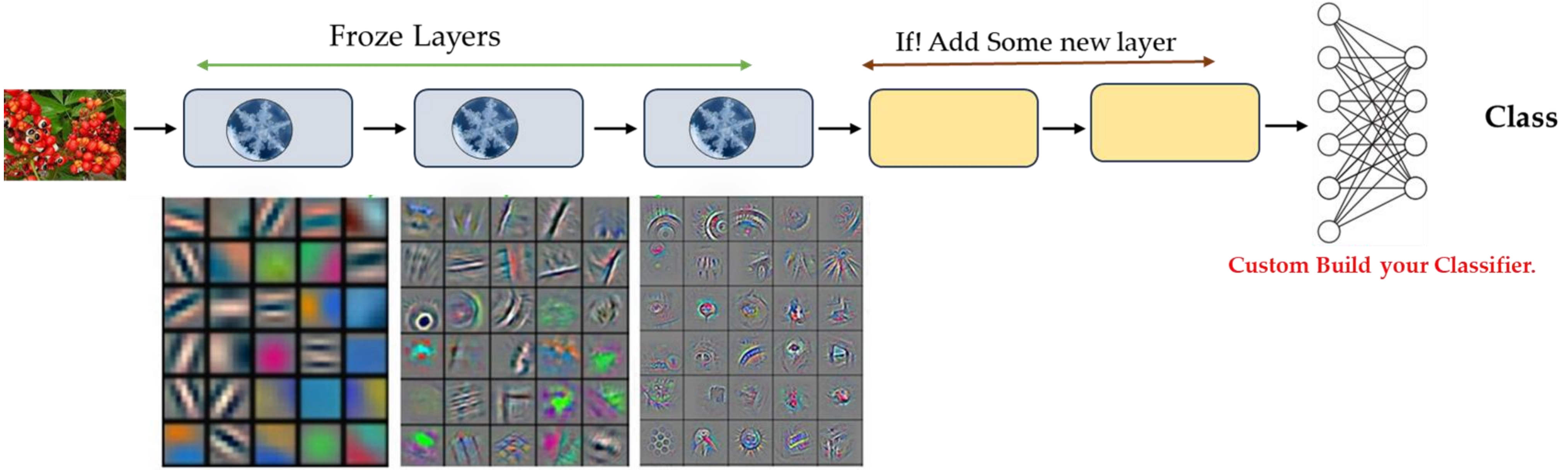
- Pre – train on Large Scale Model:
  - To learn general representations.
  - Train for a long time with large models on Large scale data.



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

1.4.1 Deep Representations are Transferrable: Transfer Learning.

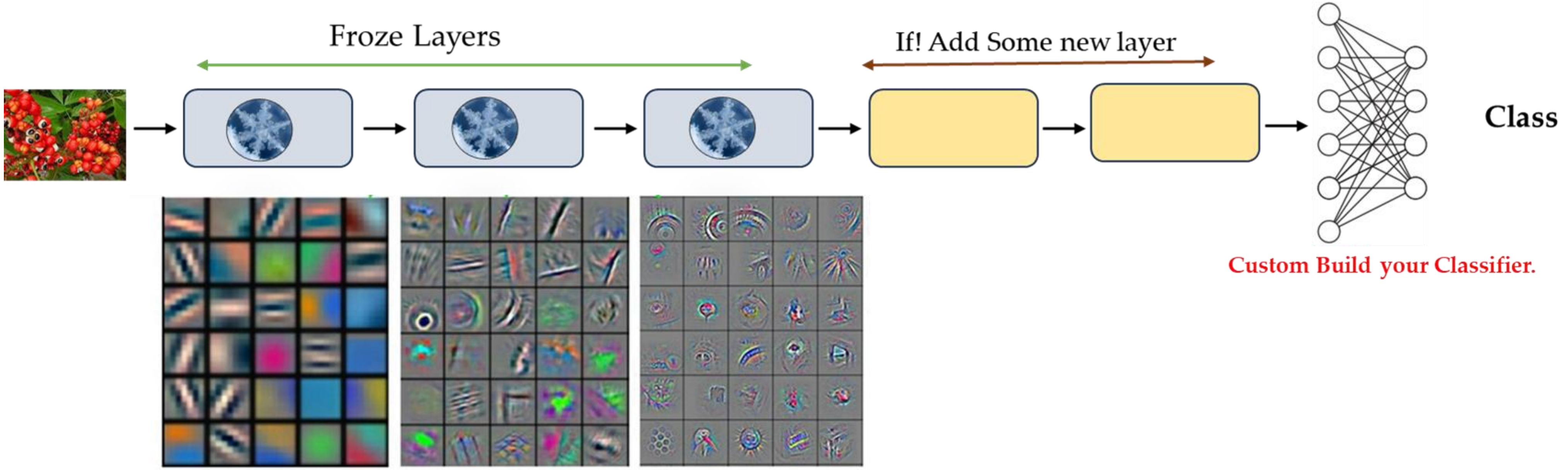
- Fine Tuning:
    - Transfer weights to specific task on small scale data
    - Train for a short time, lower learning rate.



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

1.4.1 Deep Representations are Transferrable: Transfer Learning.

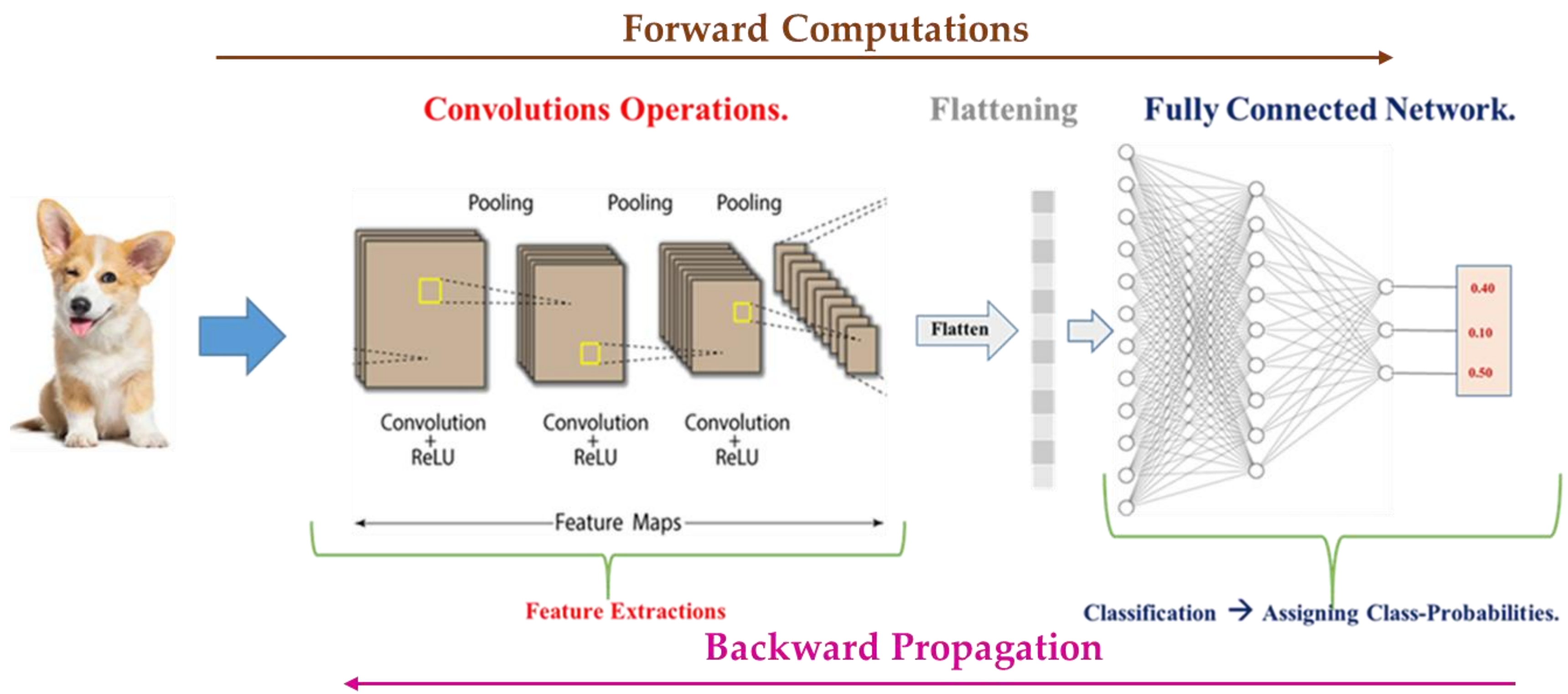
- Fine Tuning:
  - Transfer weights to specific task on small scale data
  - Train for a short time, lower learning rate.



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

How did we train the Model to learn those Representations?

1.5 How did we Train a CNN?



- In a typical **Convolutional Neural Network (CNN),**
  - features are learned by **minimizing a loss function computed from the difference between the predicted and actual class labels**.
- **But can we learn meaningful features from data *without* any labels?**

1.6 Learning Objective

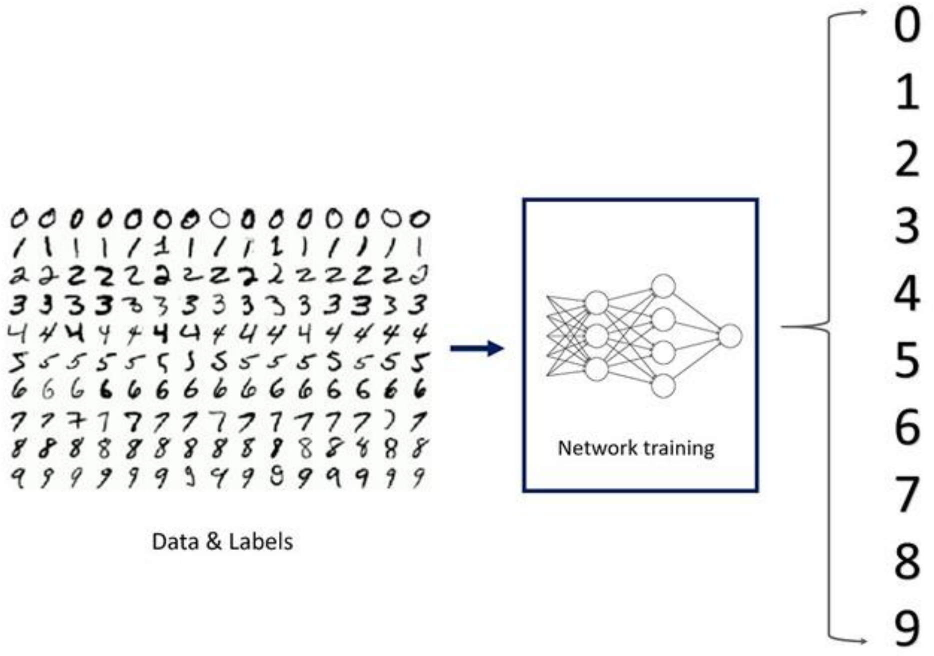
- Can we build a deep learning models
  - **which enable us to extract useful representations purely from the structure of the input data itself.**
- We will discuss
  - the underlying principles of representation learning without supervision and
  - will implement and evaluate an **autoencoder** for extracting compressed features from raw input data.

2. Background.

{Types of Learning!!!}

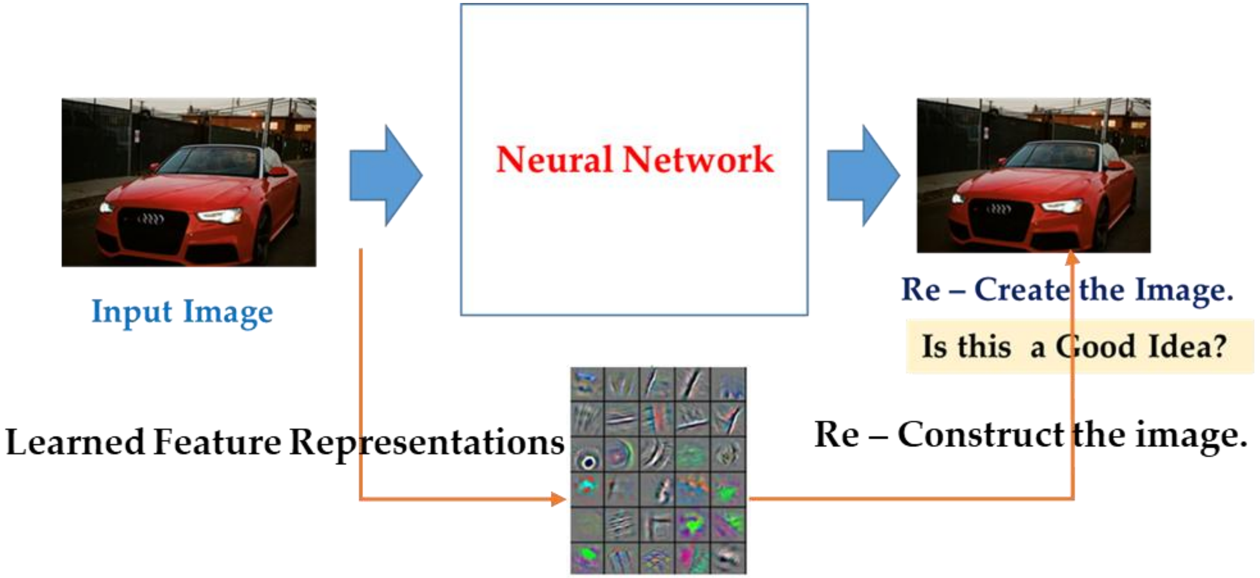
2.1 Recap: Supervised Learning.

- **Data** : {**X, Y**} where **X** is **data (feature)** and **Y** is **label (target)**
- **Goal** : Learn a **Function** to map **X → Y**
- **How** : Minimizing a **loss function**:
  - Loss function : divergence {difference} between **Y** and $\hat{\mathbf{Y}}$.
- **Example**: Classification.



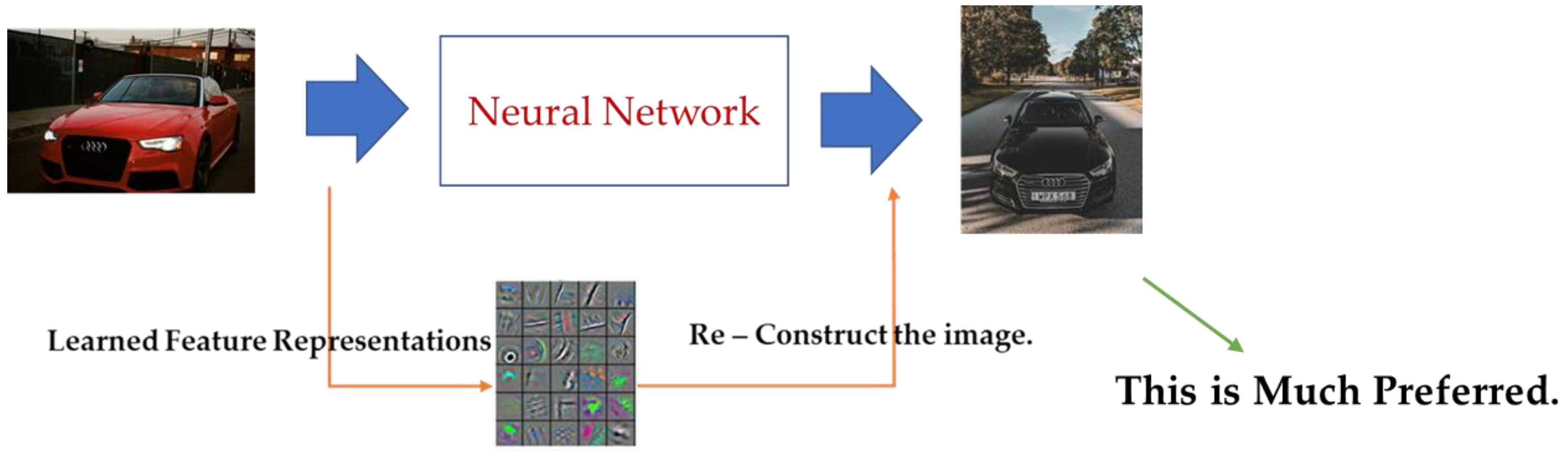
2.2 Recap: Unsupervised Learning.

- **Data** : {X} where X is **data (feature)** and {Y is **label (target) which is not available.**}
- **Goal** : Learn an underlying hidden structure {Representation} of the data. Why?
- **Example Task** : Such that we can re – create it self.
  - $X \rightarrow \tilde{X}$



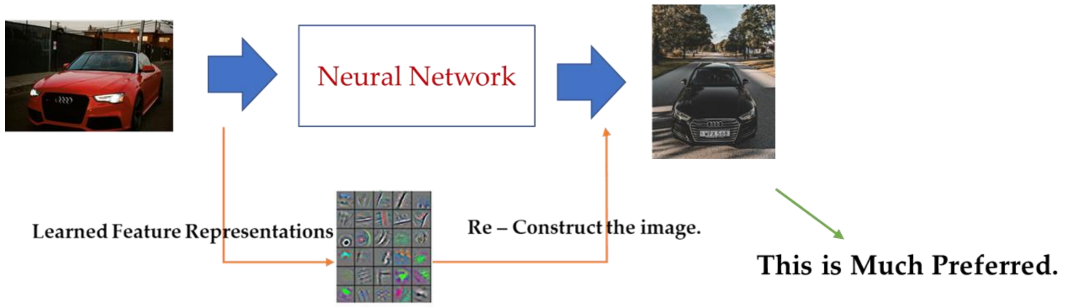
2.2.2 Unsupervised to Generative Model.

- We want to learn an approximation of the identity function such that
  - the reconstructed output closely resembles the original input i.e. $\hat{\mathbf{x}} \approx \mathbf{x}$,
- However, to ensure meaningful feature extraction rather than a trivial replication of the input,
  - the model must **learn a representation** that captures the **most salient features** of the **data**.
- This **learned representation** should **preserve essential structural and semantic information**, enabling the generation of an output that closely resembles the input while avoiding mere duplication.
  - Idea of Generative Models.



2.3 Generative Models: Idea.

- Given **training data**, generate **new samples** from **same distribution**.
  - Addresses **density estimation**, a **core problem** in unsupervised learning.
- **Density Estimation**:
  - Density estimation is the problem of **reconstructing** the **probability density function** using a set of **given data points**.
  - Namely, we observe $\{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$ as our datapoints and
    - we want to recover the underlying **probability density function** generating our **dataset**.
    - A classical approach of density estimation is the "**histogram**".



This is what we call a Generative Models!!!!

2.3.1 Generative Models: A basic Intuition.

- Given training data, generate new samples from same distribution.
  - Addresses density estimation, a core problem in unsupervised learning.

Training Data { ~ $p_{data}(X)$ }

Generated Samples { ~ $p_{model}(X)$ }

- Want to learn $\mathbf{p_{model}(X)}$ similar to $\mathbf{p_{data}(X)}${density estimation problem.}
- **How?**

Disclaimer:
This should not be understood as copy pasting or duplicating the input.

2.3.2 Generative Models.

- Given **training data**, generate **new samples** from **same distribution**.
  - Addresses **density estimation**, a **core problem** in unsupervised learning.

Training Data { ~ $p_{data}(X)$ }

Generated Samples { ~ $p_{model}(X)$ }

- Want to learn $\mathbf{p_{model}}(\mathbf{X})$ similar to $\mathbf{p_{data}}(\mathbf{X})${density estimation problem.}
- **How?**
  - **Explicit density estimation**:
    - explicitly define and solve for $\mathbf{p_{model}}(\mathbf{X})$
      - **Normalizing Flows**
      - **Autoregressive Models – ChatGpt**
      - **Diffusion Models – Dall . E**
  - **Implicit density estimation**:
    - learn model that can sample from $\mathbf{p_{model}}(\mathbf{X})$ without explicitly defining it.
    - **Autoencoder {Topics of our Interest}**

2.4 Why Generative Models?

- Realistic samples for artwork, super-resolution, colorization, chatbot {Q n A}etc.



- Generative models of time-series data can be used for simulation and planning (reinforcement learning applications!)
- **Training generative models can also**
  - **enable inference of latent representations that can be useful as general features.**
    - **This is what AutoEncoder does.{ This is what we will learn to do this week.}**

3. Introduction to Auto – Encoder.

3.1 Before we start:

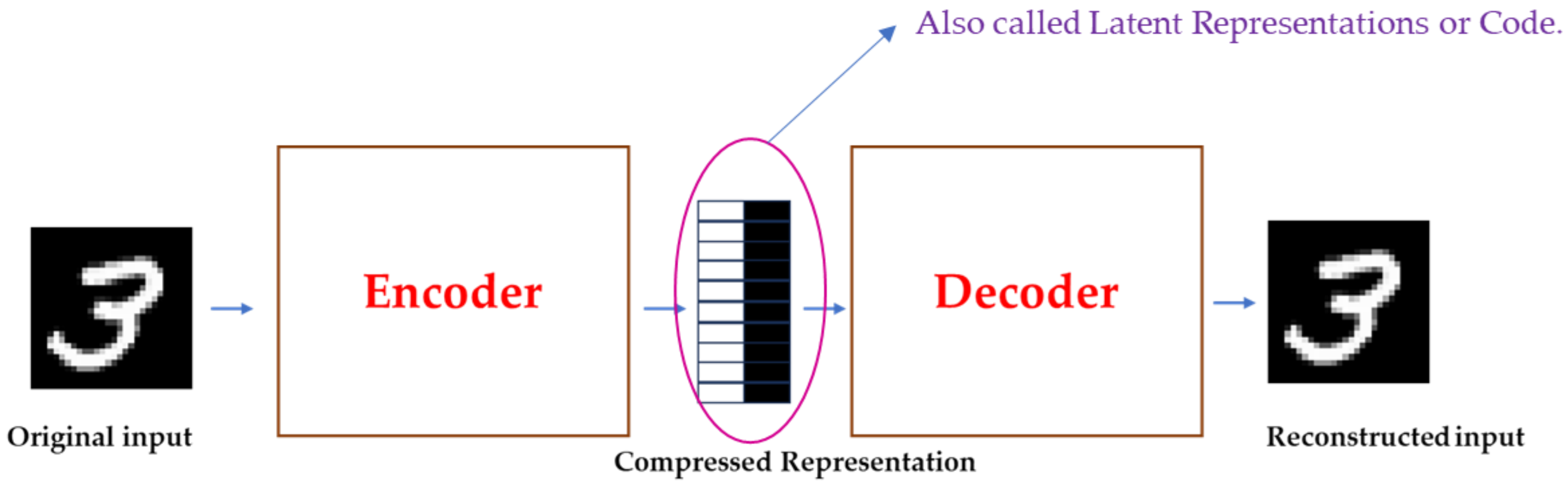
- **Are Autoencoder a Generative Model?**
  - Yes, autoencoders learn to reconstruct their input, making them a form of generative model.
  - However, their generative capability is limited,
    - as they primarily reproduce the variations of the input rather than generating entirely new samples.
  - Despite this limitation, autoencoders remain widely useful in applications such as
    - **dimensionality reduction**, **anomaly detection**, **and denoising,**
    - where precise feature learning and reconstruction are more important than generating novel samples.

Expectation

Reality

3.2 Autoencoders: Introduction.

- An autoencoder is a type of neural network architecture designed to efficiently compress (encode) input data down to its essential features,
  - then reconstruct (decode) the original input from this compressed representation.
- An autoencoder typically consists of three blocks:
  - **Encoder Layer**: to the compress the input data into a compressed representations.
  - **Bottleneck layer:** or code or latent representations: to represent the compressed input.
  - **Decoder Layer**: to reconstruct the encoded image back to the original dimension.



3.3 Encoder – Decoder Architecture: Why?

- **Convolutional Neural Networks (CNNs)** traditionally follow a **canonical architecture**,
  - where the **spatial dimensions** of feature maps progressively decrease as depth increases.
  - This design is well-suited for **supervised learning**, where the primary goal is to **extract hierarchical features** and
    - map them to **class probabilities**.
- However, in Unsupervised learning, **where class labels are unavailable**
  - our **objective shifts from classification** to **learning meaningful representations**
    - that preserve essential features of the input.
  - Thus, instead of assigning class we seek to reconstruct the input itself, **necessitating a different architectural approach**.
    - This leads to Encoder – Decoder framework where:
      - **Encoder compress the input into a latent representation,**
      - **Decoder reconstructs it, enabling the model to capture the underlying structure of the data without explicit supervision.**

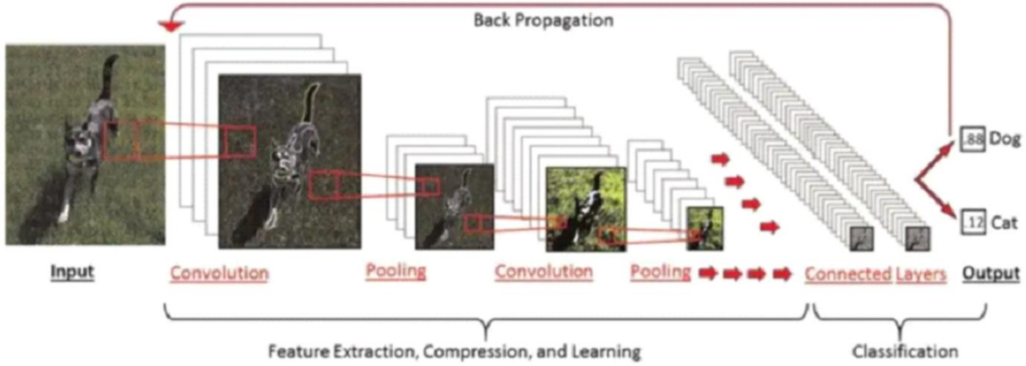


Image from Mickael Komendyak Blog post.

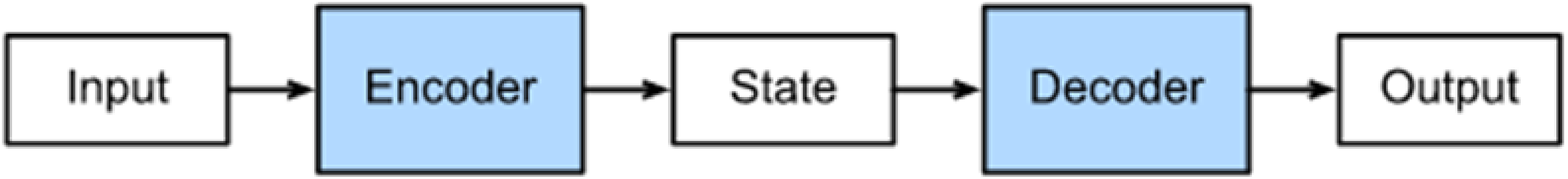
3.3.1 Introduction to Encoder – Decoder Architecture.

Encoder

- This process allows the encoder **to capture the most relevant information** {feature} **from the inputs** and produce a **fixed-size** **representation** of it.
- Encodes the important feature information in some latent dimension {generally smaller than input}
- Input can be image or text or others.

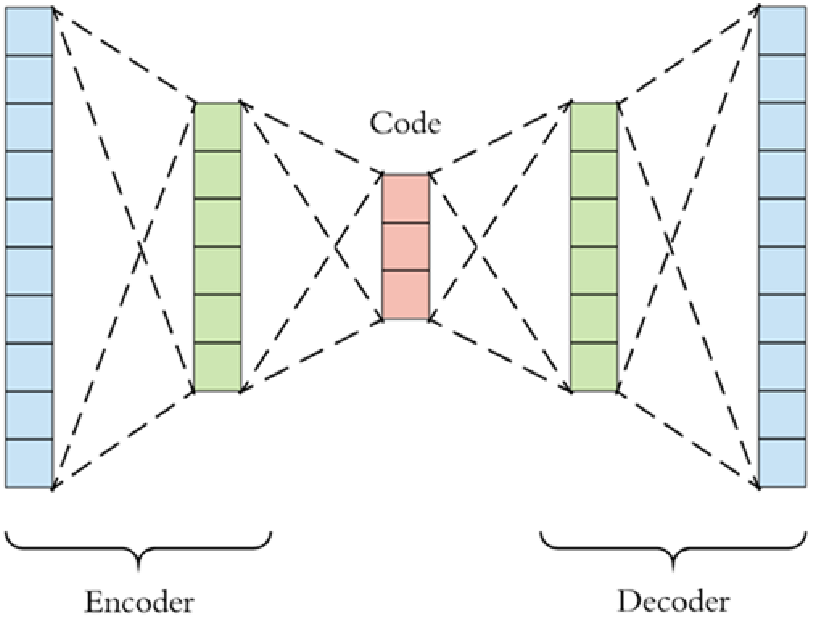
Decoder

- The decoder, on the other hand, takes the **encoded representation** produced by the encoder and **generates an output**.
- Generated output can be image or text or other.



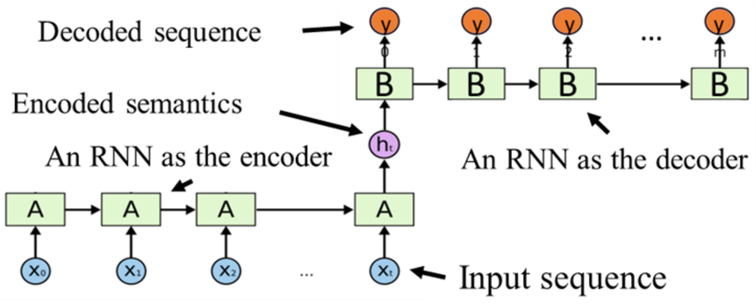
3.3.2 Encoder – Decoder Architecture.

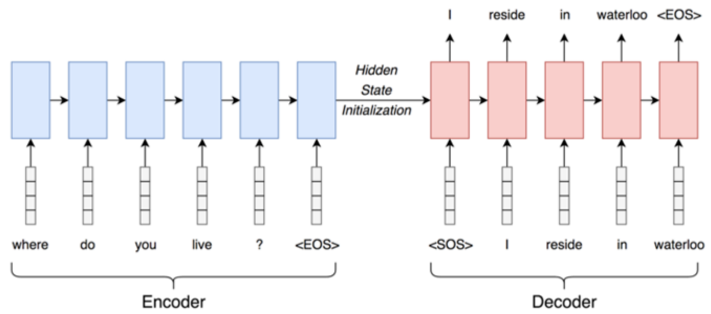
- **Training Encoder – Decoder Architecture:**
  - The encoder-decoder architecture can be trained end-to-end using backpropagation and (stochastic) gradient descent to minimize a loss function that compares the predicted output to the ground truth.
  - In many applications, the {**loss function**} is designed to penalize differences in the {**pixel values**} between the **predicted and ground truth images**.



3.3.3 Encoder – Decoder Architecture Application.

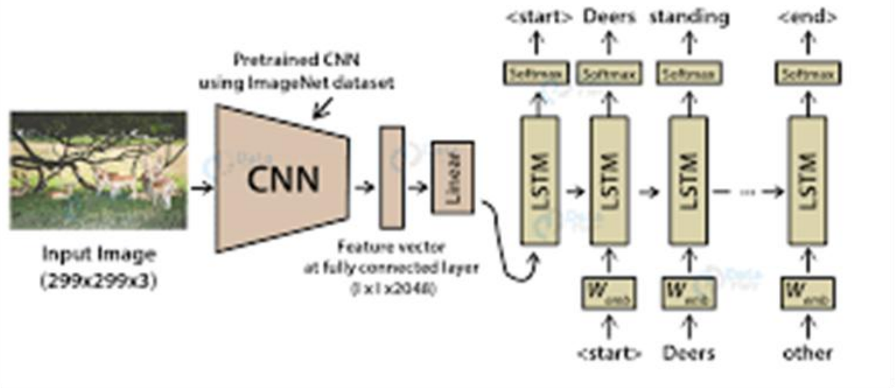
- **Encoder – Decoder Architecture → Examples.**
  - **Sequence – to Sequence Model {Neural Machine Translation}**
  - **Encoder:**
    - The encoder turns each item into a corresponding hidden vector containing the item and its context.
      - **extract** and **compress** the **semantics** from the **input sequence**
    - In RNN in some variants like {LSTM or GRU is Used}
  - **Decoder:**
    - The decoder reverses the process, turning the vector into an output item, using the previous output as the input context.
      - **generate** a **sequence** based on the **input semantics**
    - In RNN in some variants like {LSTM or GRU is Used}
    - { Week – 9's Discussion}





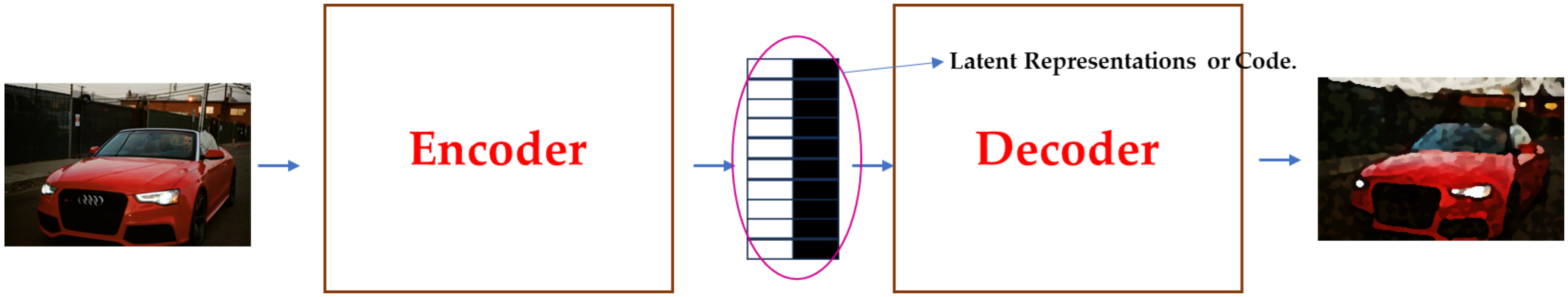
3.3.4 Encoder – Decoder Architecture Application.

- **Encoder – Decoder Architecture → Examples.**
  - **Image Captioning**
  - **Encoder:**
    - The encoder turns each image into a corresponding hidden vector containing the discriminative features.
      - extract and compress the features from the input images
    - An CNN in some variants and with transfer learning is used.
  - **Decoder:**
    - The decoder reverses the process, turning the vector into an output item, using the previous output as the input context.
      - generate a sequence based on the input features.
    - An RNN in some variants like {LSTM or GRU is Used}



3.4 Getting back to Autoencoder!!!

- Autoencoders form a very specific subset of **encoder – decoder architectures** that are trained via unsupervised learning to **reconstruct their own input data**.
  - **Input and output dimensions are the same.**
- Using unsupervised machine learning, autoencoders are trained to discover the **latent variables** of the input data.
- Learning g(f (x)) = x (identity function) everywhere is not very useful,
  - Thus, autoencoders are designed to be unable to copy perfectly.
- Autoencoders learn useful properties of data and can prioritize which aspects of input should be copied.

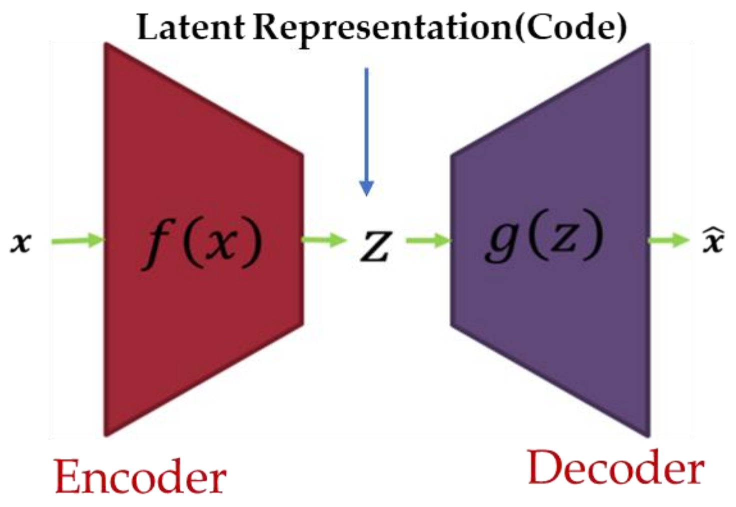


Terminology Alert!!!

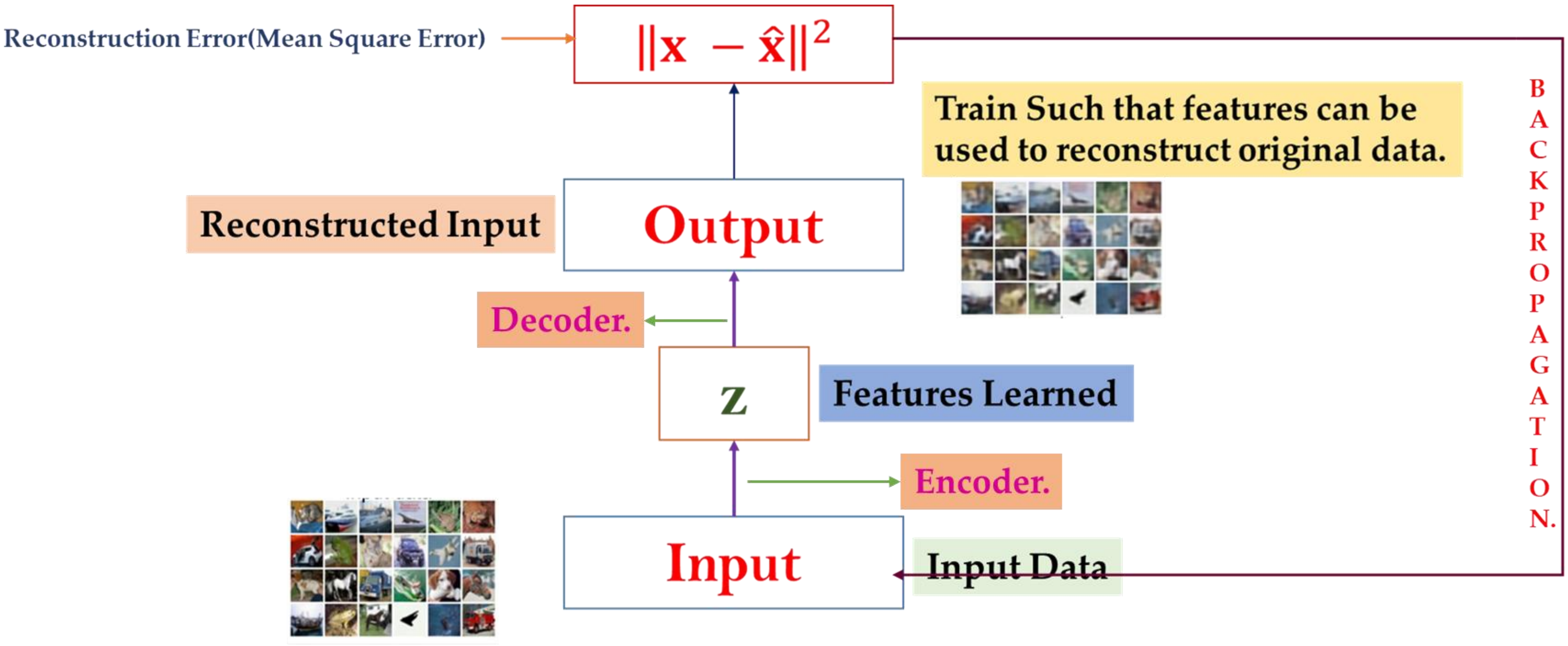
- **Latent Variable(Origin-Latin): "lie hidden":**
  - **In statistics:**
    - latent variables are variables that **can only be inferred indirectly through a mathematical model** from **another observable variable** which can be **measured directly**.
- **Latent Representation or Space or embeddings or code:**
  - **It is a vector space spanned by the latent variables.**

3.5 AutoEncoder in Practice.

- **Idea**
  - Given data $\mathbf{x}$ (**no labels**) we would like to learn the functions $\mathbf{f}$(**encoder**) and $\mathbf{g}$(**decoder**) where:
    - $\mathbf{f(x) = act(wx + b) = z}$
  - **Here:**
    - Act – activation functions.
    - **Z** latent representation or code.
  - Here:
    - $\mathbf{\hat{x}}$ **is** $\mathbf{x'}$**s reconstructions.**
    - $\mathbf{h(x) = g(f(x))} \rightarrow$ **approximation of identity function.**

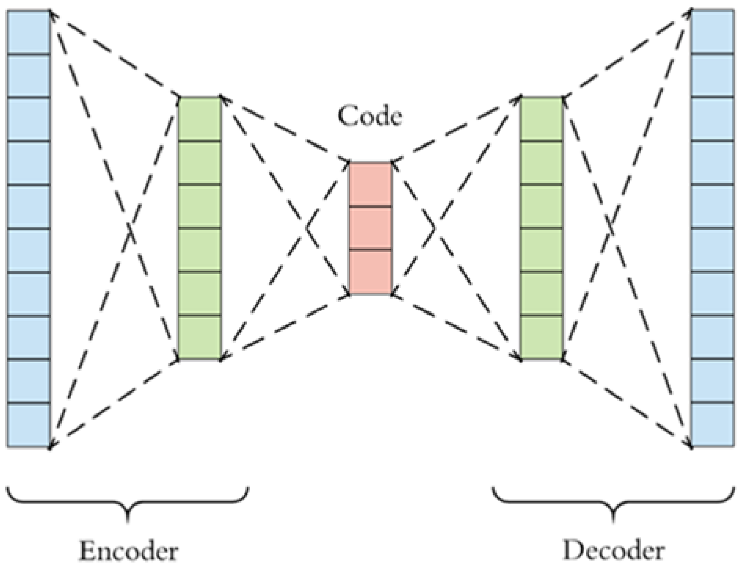


3.6 Training an Autoencoder.



3.6.1 Training Auto – Encoders: Hyper – parameters.

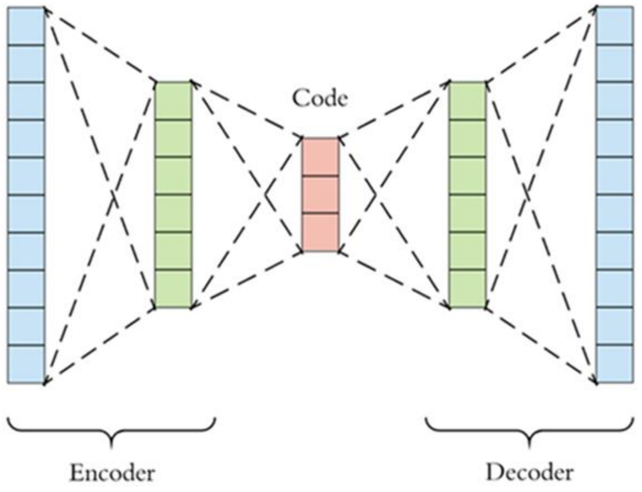
- There are **four hyperparameters** that need to be set when training an autoencoder:
  - **Code Size (Dimension of Latent Representations)**:
    - This refers to the number of nodes in the middle or bottleneck layer. By choosing a smaller size than the input dimension, we force the model to learn the most important feature representations of the data.
  - **Number of Layers**:
    - This typically refers to the number of fully connected (dense) or convolutional layers used to build the encoder and decoder.
    - The autoencoder can be made as deep as necessary, depending on the complexity of the data.



3.6.1 Training Auto – Encoders: Hyper – parameters.

- **Number of Nodes per Layer**:
  - In a standard (stacked) autoencoder, the number of nodes decreases in successive layers of the encoder and increases back in the decoder.
  - The decoder is usually **symmetric** to the encoder in terms of layer structure.
- **Loss Function**:
  - Common choices include Mean Squared Error (MSE) for continuous data and Binary Cross Entropy (BCE) for binary or normalized data.

- Autoencoders are typically trained using **backpropagation**, combined with **mini-batch gradient descent**.



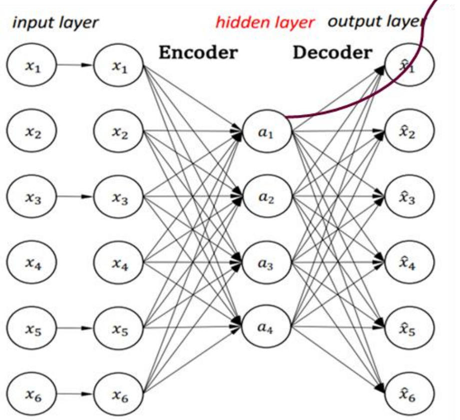
4. Designing an Auto –Encoder.

{ Putting Neural Network in Encoder and Decoder}

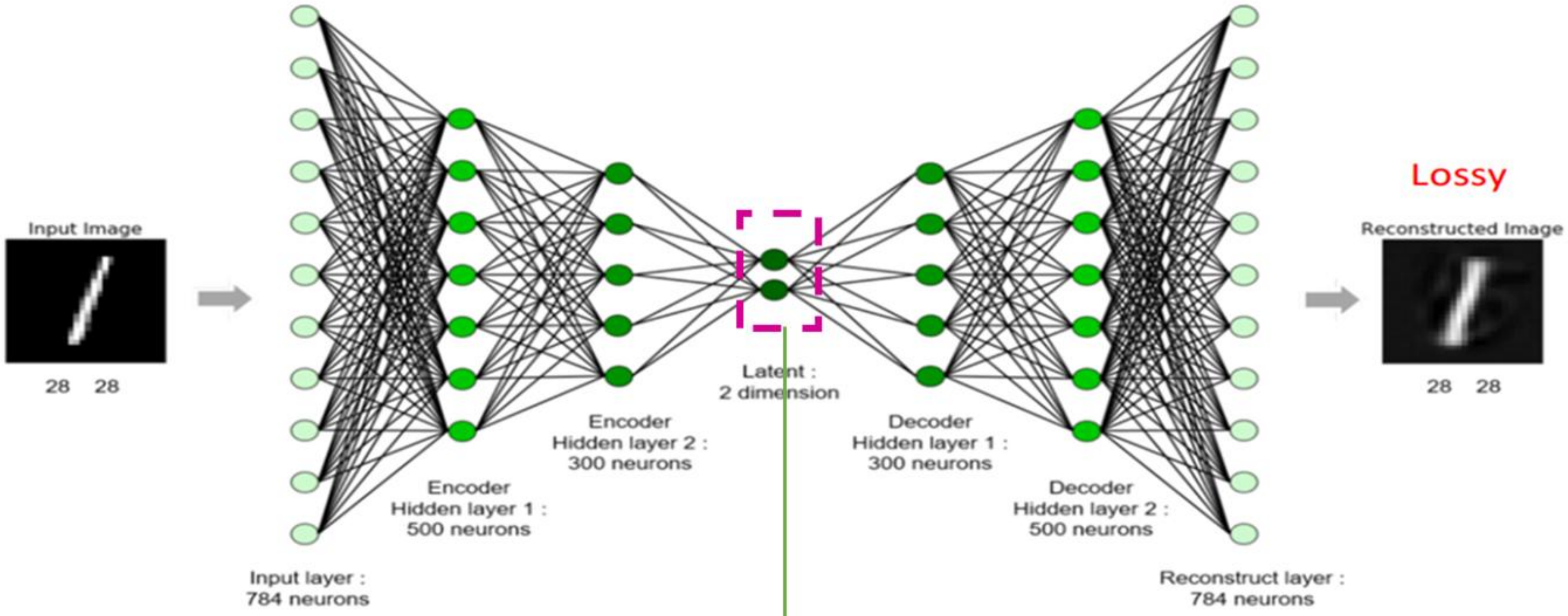
4.1 Vanilla Autoencoder.

- A Vanilla Autoencoder is the simplest form of autoencoder, composed of:
  - An **Encoder composed of Fully Connected Neural Network (Dense) layer** that maps the **input x** to a **lower dimensional latent representations z.**
  - A **Decoder also composed of Fully Connected Neural Network** that **reconstructs** the input from the latent code z to **produce x̂.**
- **Objective:**
  - To learn a **compressed representation of the data** by mining the reconstruction loss:
    - $\mathcal{L} = \|\mathbf{x} - \hat{\mathbf{x}}\|^2$
- Symmetric Architecture:
  - The **decoder usually mirrors the encoder** in terms of layers and nodes.
- **Use case:**
  - Dimensionality Reduction.
  - Feature Extraction.

Learn This!!!



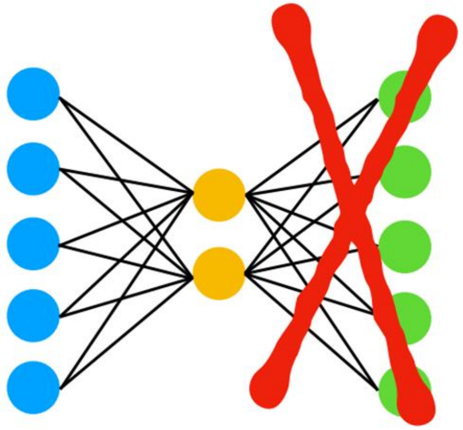
4.2 Vanilla Autoencoder: Dimensionality Reduction.



- Using an autoencoder with a **2-dimensional latent space**,
  - we reduce the original 784-dimensional image representation to just 2 dimensions.
    - Each **image is now encoded as a 2D point, capturing its most salient features.**

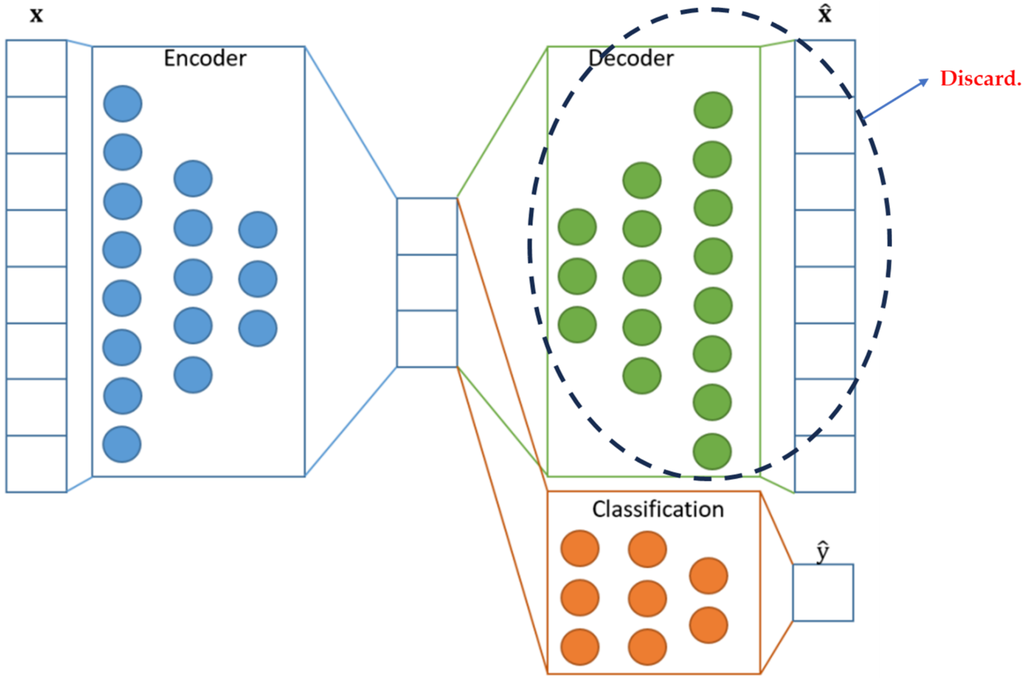
4.3 Vanilla Autoencoder: Feature Extraction.

- **How it works as a Feature Extractor?**
  - **Training the autoencoder:**
    - The model is trained to minimize the difference between the input and its reconstruction.
    - During this process, the **encoder** learns to capture the most informative features about the data in its latent space (i.e., the most important factors that explain the variability in the data).
  - **Using the Latent Representations:**
    - Once the model is trained, the encoder part is used to **extract features** from new data.
    - The latent space (encoded representation) becomes a **compressed version** of the input data,
      - containing the **relevant features for further tasks**.



4.3.1 Vanilla Autoencoder: Feature Extraction.

- **Feature Extraction for Downstream Tasks**:
  - **Classification**:
    - You can use the encoded features as inputs to another classifier (e.g., SVM, logistic regression).
  - **Clustering**:
    - The features can be clustered (e.g., using k-means) to identify groups within the data.
  - **Anomaly Detection**:
    - The reconstructed error can be used for anomaly detection, where a high error indicates that the input is an outlier.



4.4 Denoising Autoencoder.

- A **Denoising Autoencoder (DAE)** is a variant of the vanilla autoencoder designed to learn **robust feature representations** by reconstructing **clean inputs from noisy versions**.
- **Motivation:**
  - While vanilla autoencoders can learn compressed representations, they may **simply memorize** the input.
  - To encourage **generalization** and **robustness**,
    - denoising autoencoders are trained to **recover the original input from a corrupted version**.
- **How it works?**
  - Add **noise or corruption** to Input:
    - A stochastic corruption process (**e.g. Gaussian noise**)
      - is applied to **the input x, producing noise free $\hat{x}$.**
    - The original paper by Y. Bengio et. Al. (Learning Deep Architectures 2009) also suggest to use
    - **random masking similar to Dropout:**
      - During training, the **input data x** is corrupted by applying **dropout** to the input layer,
        - meaning **some parts of the input are randomly set to zero (dropped out**).
        - This was done to **simulate the presence of missing or noisy data**.
      - {DropOut was introduced by Alex et. al in AlexNet Paper in 2012}

4.4.1 Denoising Autoencoder: Working.

- **Encode the Noisy Input:**
  - $\mathbf{z} = \mathbf{Encoder}(\tilde{\mathbf{x}})\ \{\tilde{\mathbf{x}} \rightarrow \mathbf{Noisy\ Input}\}$
- **Decode to Reconstruct the Clean Input:**
  - $\hat{\mathbf{x}} = \mathbf{Decoder}(\mathbf{z})$
- **Train to minimize Reconstruction Loss:**
  - $\mathcal{L} = \|\mathbf{x} - \hat{\mathbf{x}}\|^2$
- Use Cases:
  - **Image Denoising.**
  - **Robust Feature Extraction.**

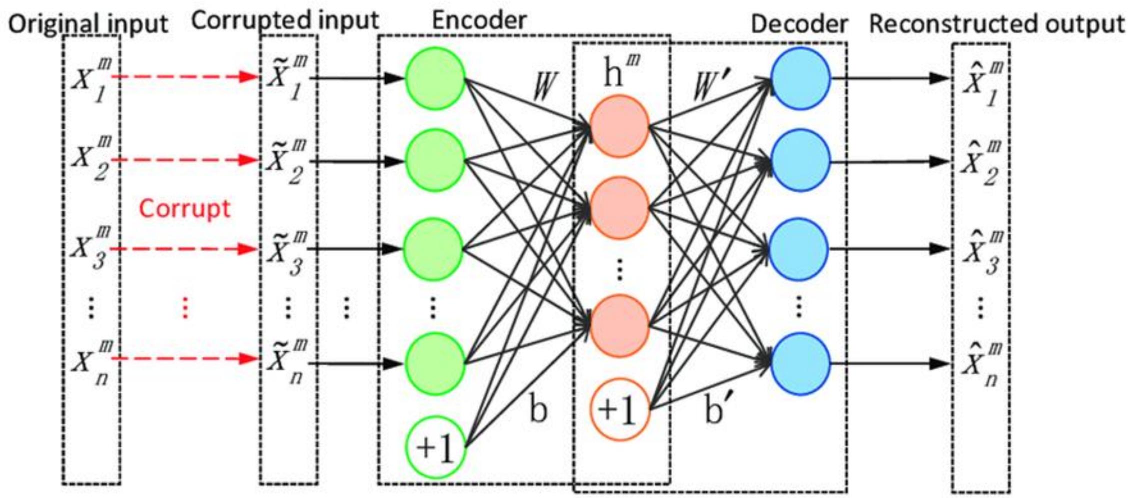


Image from Internet Subject to Copyright.

4.4.3 Denoising Autoencoder: Application.

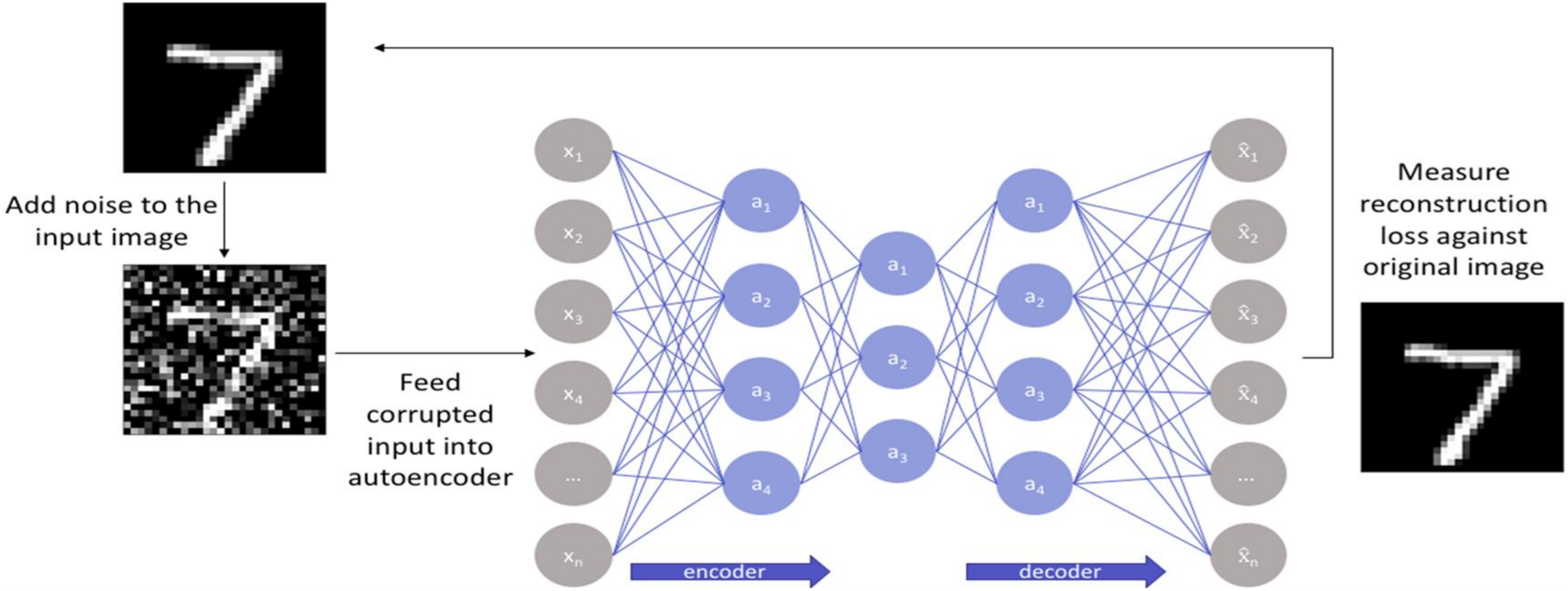
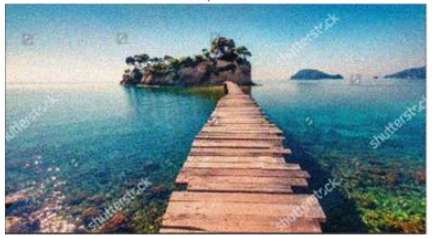


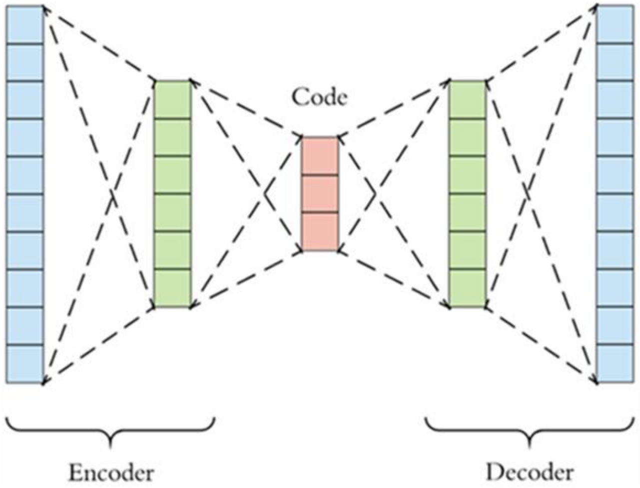
Image from Internet: Subject to copyright.

4.4.4 A Real-world application

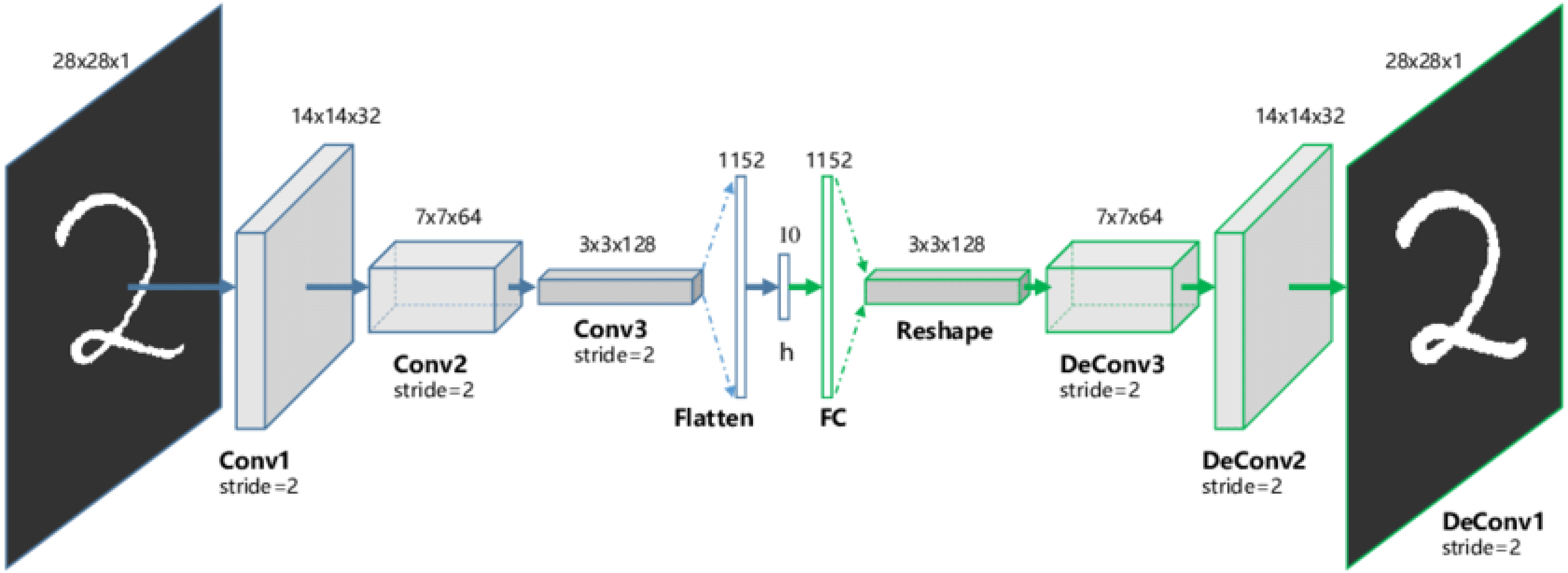
- Water Mark Removal:

Add WaterMark and Noise.





5. Convolutional Autoencoder.



5.1 Convolutional Autoencoder.

- Encoder-
  - 1 or more convolutional layers.
- Decoder-
  - 1 or more Transposed convolutional layers.

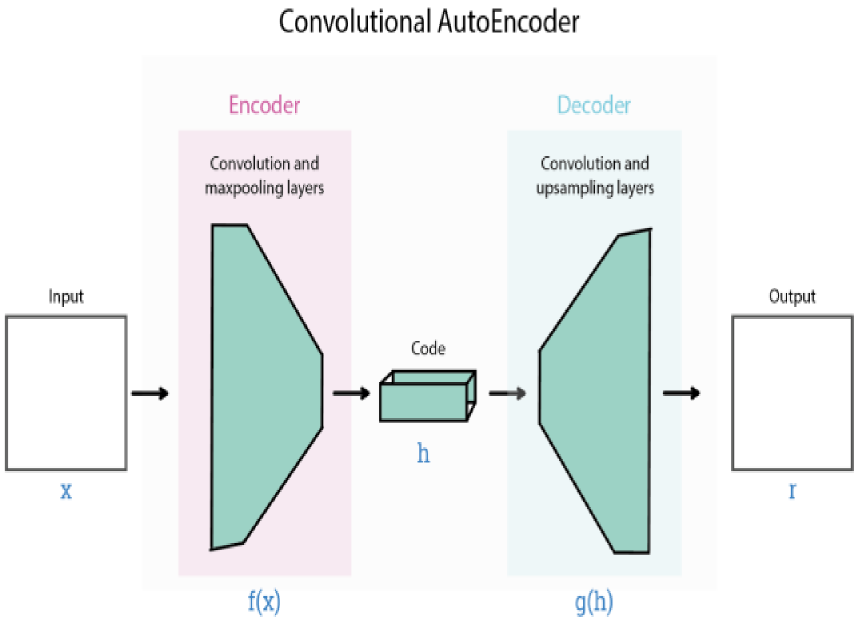


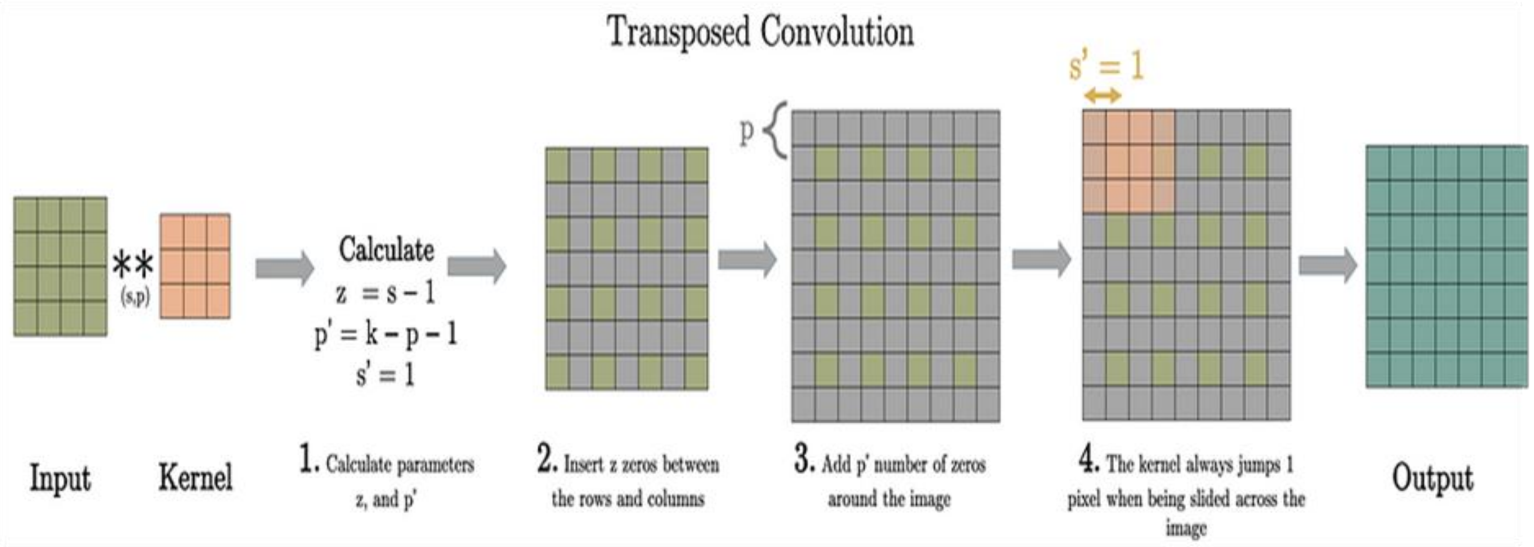
Fig. 3. General architecture of a convolutional autoencoder (CAE).

5.2 Transposed Convolutions.

- Allows us to increase the size of the **output feature map** compared to the **input feature map**.
- The output size of a transposed convolutional layer can be computed using the following formula:
  - output_size = (**input_size - 1**) * **stride** + **kernel_size** - 2 * **padding**
  - where:
    - **input_size**: the size of the input tensor along the spatial dimensions (width and height)
    - **stride**: the stride of the transposed convolution operation
    - **kernel_size**: the size of the transposed convolution kernel along the spatial dimensions
    - **padding**: the amount of zero padding added to the input tensor along the spatial dimensions
- Synonyms:
  - often also (**incorrectly**) called "**deconvolution**" (mathematically, deconvolution is defined as the inverse of convolution, which is different from transposed convolutions)
  - the term "unconv" is sometimes also used
  - aka “upsampling” layer.

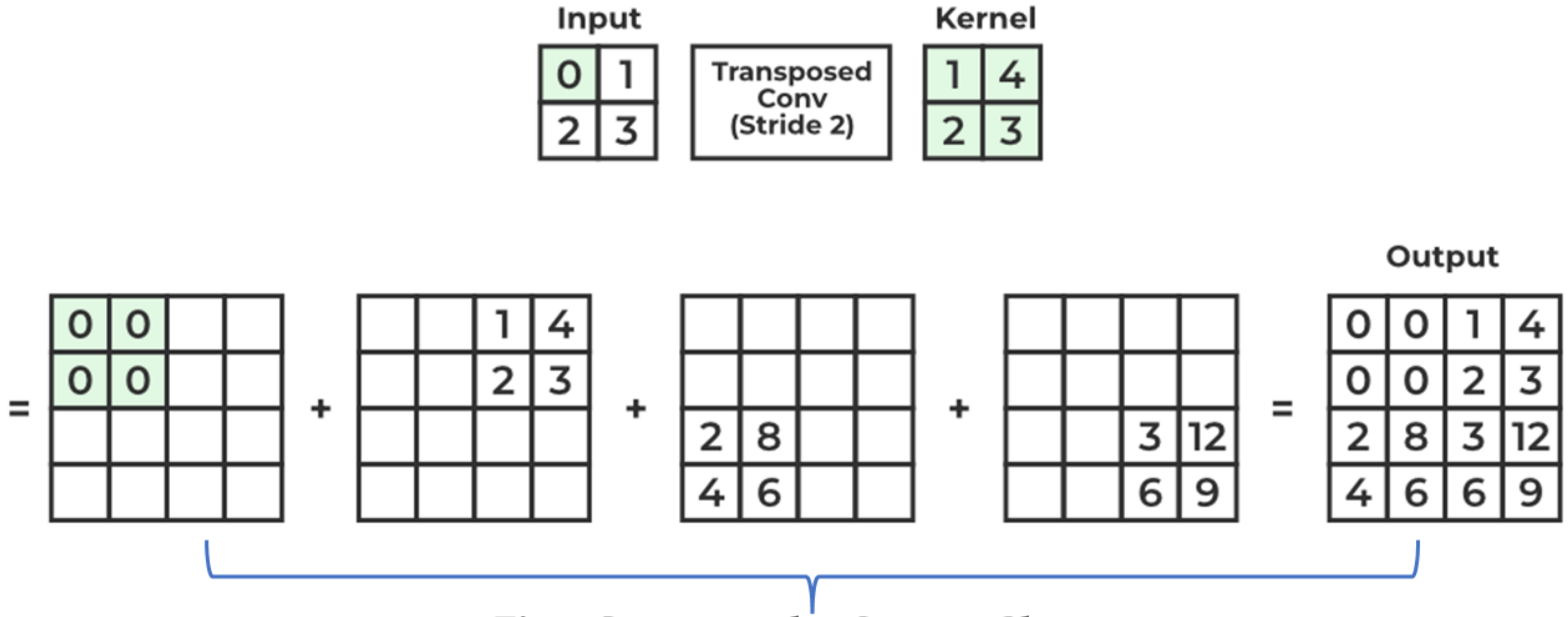
5.3 Transposed Convolutions – In Practice.

- Implementing a transposed convolutional layer can be better explained as a 4-step process:
  - **Step 1:** Calculate new {**hyper**} **parameters z and p'**
  - **Step 2:** Between each row and columns of the input, insert z number of zeros. This increases the size of the input to **(2*i-1)x(2*i-1)**
  - **Step 3:** Pad the modified input image with **p' number of zeros**
  - **Step 4:** Carry out **standard convolution** on the **image generated from step 3** with a **stride length of 1**



5.4 Transposed Convolutions-Demo.

- **How Transposed Convolution Works?**



First Compute the Output Shape.

5. Some Codes.

Vanilla Autoencoder.

```
# This is the dimension of the original space
input_dim = 10

# This is the dimension of the latent space (encoding space)
latent_dim = 2

encoder = Sequential([
    Dense(128, activation='relu', input_shape=(input_dim,)),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(latent_dim, activation='relu')
])

decoder = Sequential([
    Dense(64, activation='relu', input_shape=(latent_dim,)),
    Dense(128, activation='relu'),
    Dense(256, activation='relu'),
    Dense(input_dim, activation=None)
])
```

```
autoencoder = Model(inputs=encoder.input, outputs=decoder(encoder.output))
autoencoder.compile(loss='mse', optimizer='adam')
```

CAE.

```
input = layers.Input(shape=(28, 28, 1))

# Encoder
x = layers.Conv2D(32, (3, 3), activation="relu", padding="same")(input)
x = layers.MaxPooling2D((2, 2), padding="same")(x)
x = layers.Conv2D(32, (3, 3), activation="relu", padding="same")(x)
x = layers.MaxPooling2D((2, 2), padding="same")(x)

# Decoder
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu", padding="same")(x)
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu", padding="same")(x)
x = layers.Conv2D(1, (3, 3), activation="sigmoid", padding="same")(x)

# Autoencoder
autoencoder = Model(input, x)
autoencoder.compile(optimizer="adam", loss="binary_crossentropy")
autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 14, 14, 32)        9248
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 28, 28, 32)        9248
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 1)         289
=================================================================
Total params: 28,353
Trainable params: 28,353
Non-trainable params: 0
_________________________________________________________________
```

At the end!!!!

- Autoencoders learn data representation in an **unsupervised**/ **self-supervised** way.
    - Learned features are able to capture salient properties of data
- Different with vanilla autoencoder, in sparse autoencoder, the number of hidden units can be greater than the number of input variables.
    - **Under - complete** and **Over - complete** Architectures.
- Can also be stacked to create → deep autoencoders.

- **"You will implement Autoencoders in Tutorial → Please come with laptop."**

Thank You