Variational AutoEncoders (DRAFT)

Mark A. Austin

University of Maryland

austin@umd.edu
ENCE 688P, Spring Semester 2026

March 25, 2026

Overview

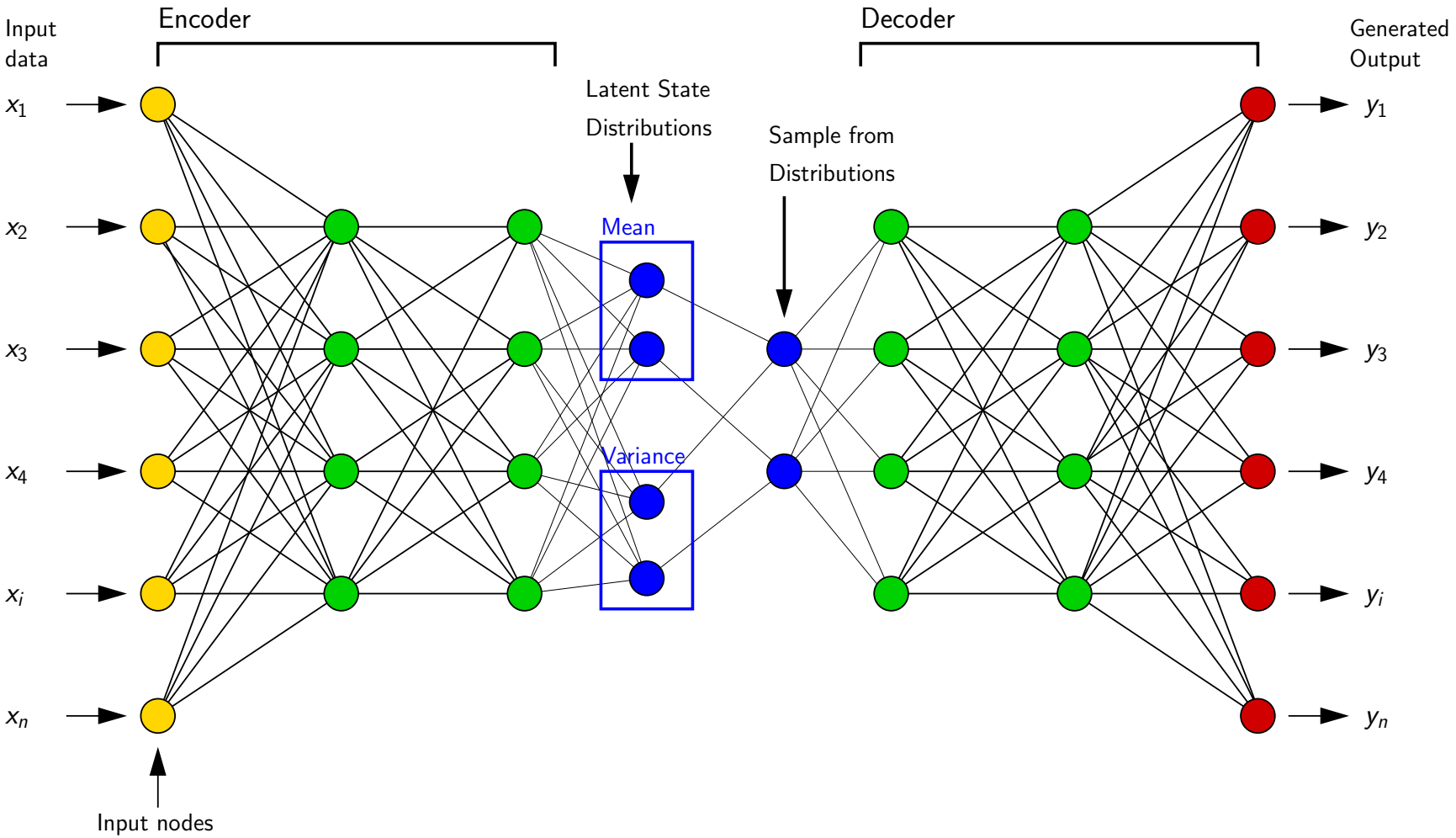
1. Variational AutoEncoders
   - Basic Idea and Applications
2. Theoretical Background
   - Convolution Operations
   - Convolution Neural Networks (CNNs) and Image Analysis
   - Probability Estimation
3. Examples
   - Example 1: Simplest Example
   - Example 2: Generation of Fashion Shapes
   - Example 3: Generation of Cartoon Faces
   - Example 4: Generation of Faces

Variational

AutoEncoders

Basic Idea and Applications

Variational AutoEncoders (Generative Models)



Basic Idea and Applications

Encoder

The encoder learns how to reduce the input dimensions and compress the input data into an encoded representation.

Decoder

The decoder learns how to reconstruct the input data from the encoded representation and be as close to the input data as possible.

Latent Space

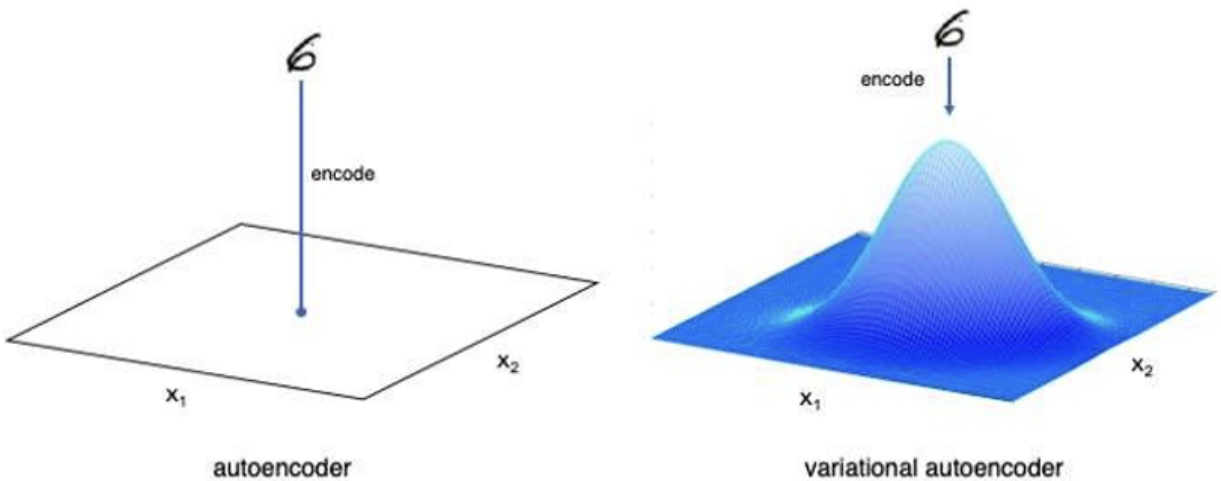
Latent space is simply a representation of compressed data in which similar points are closer together in space. This formalism is useful for learning data features and finding similar representations of data for analysis.

Classification of Machine Learning Problems

Standard Autoencoders vs. Variational Autoencoders:

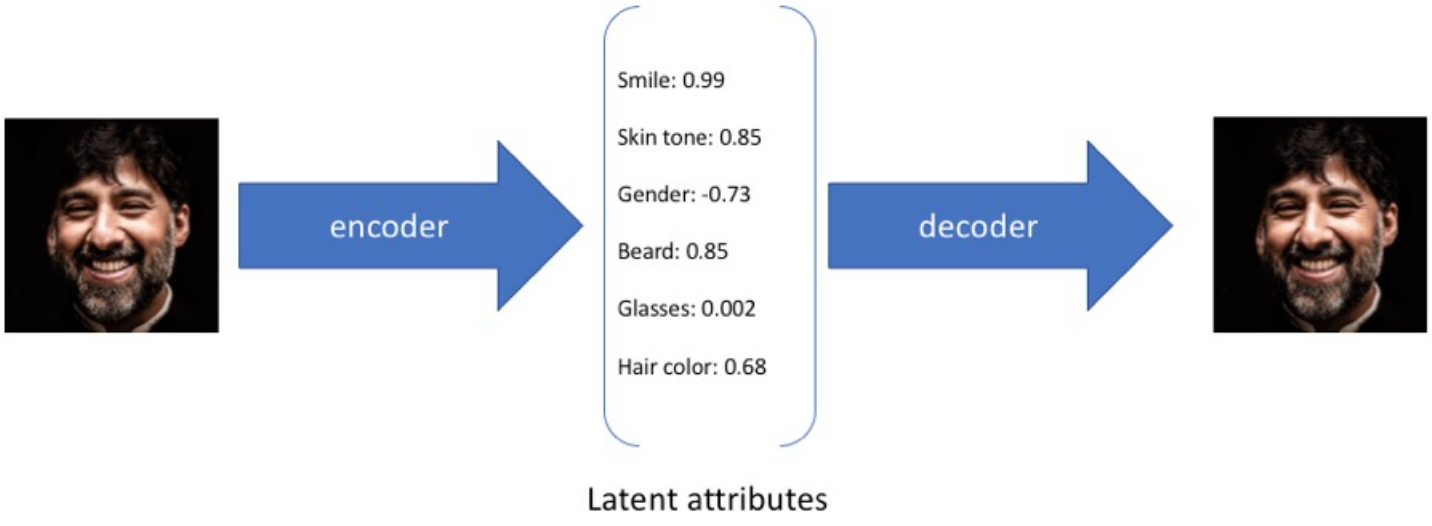
- A standard autoencoder outputs a single value for each encoding dimension.
- Variational autoencoders provide a probability distribution for each latent attribute.

Latent Space: (x_1, x_2)



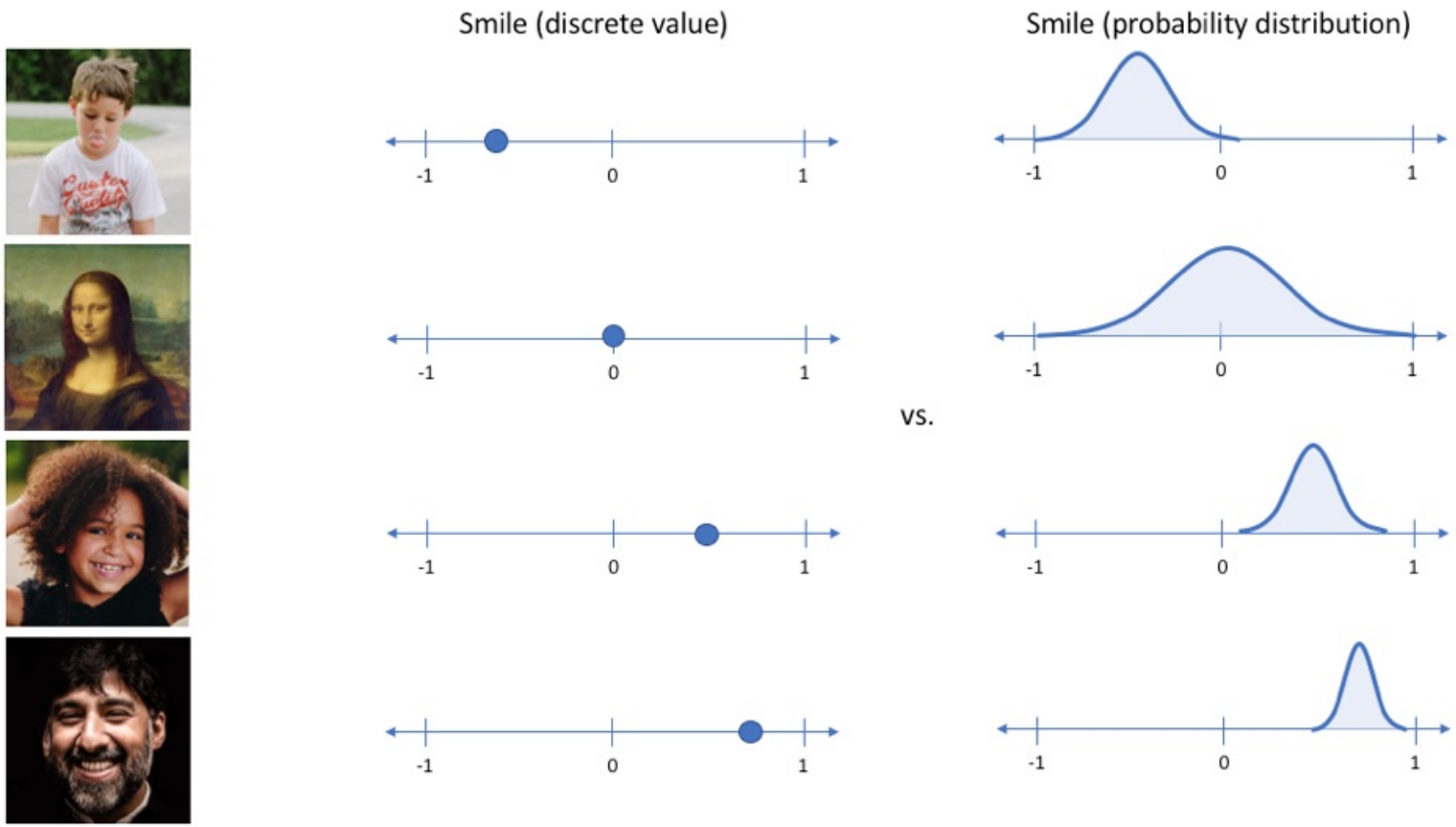
Classification of Machine Learning Problems

Example: Single value representations for latent attributes:



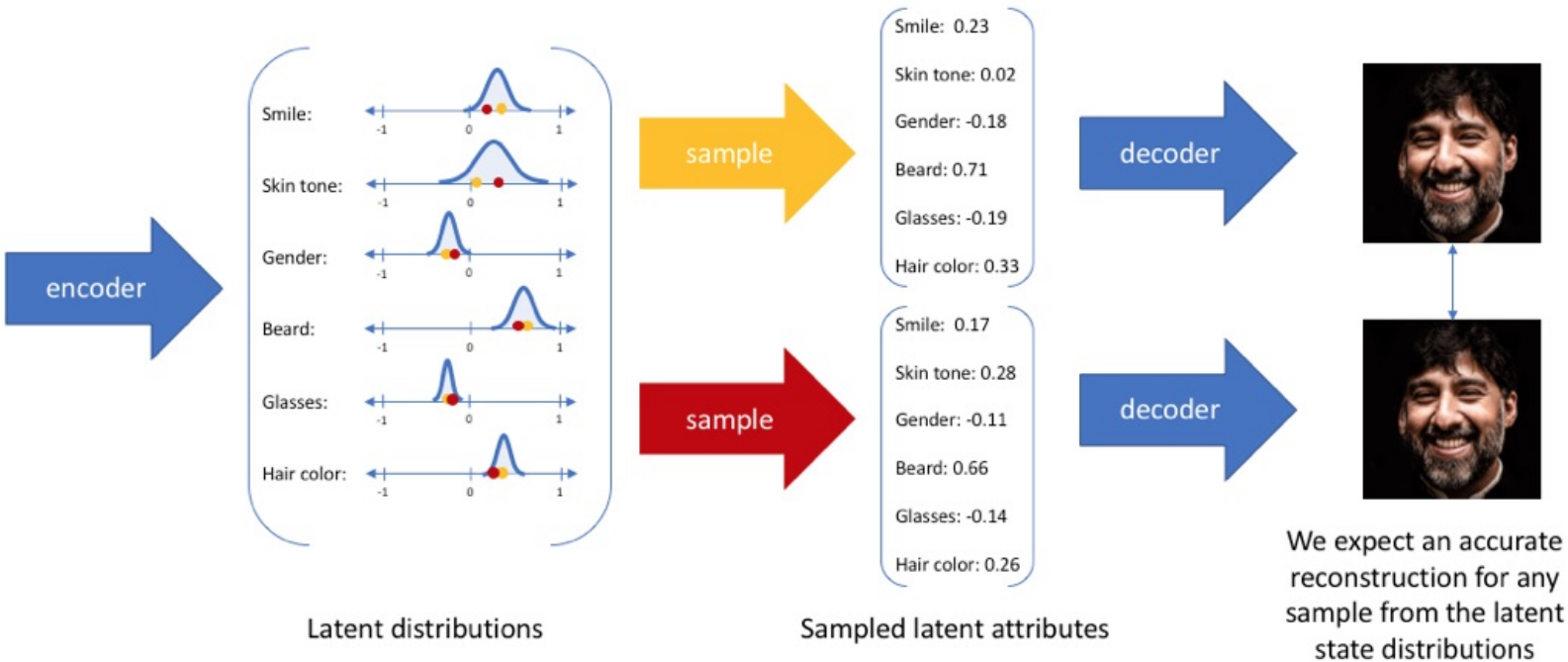
Classification of Machine Learning Problems

Discrete Value and Probability Distribution: Representations for smile latent attribute:



Classification of Machine Learning Problems

Image Reconstruction: sampled from latent distributions ...



Source: Jordan J., Variational Autoencoders, Data Science, March 2018.

Classification of Machine Learning Problems

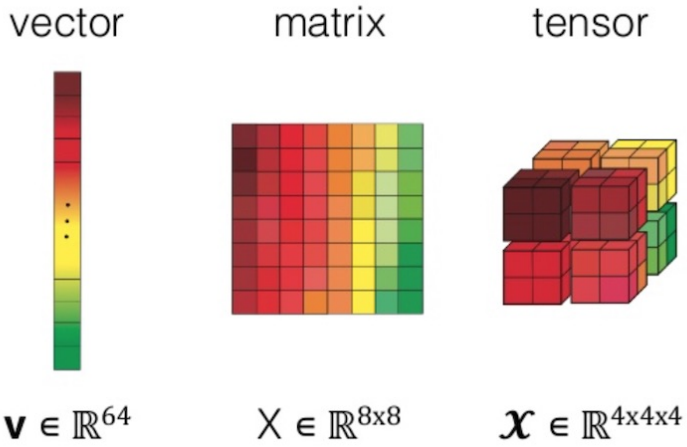
VAE Applications:

- ...
- ...
- ...
- ...

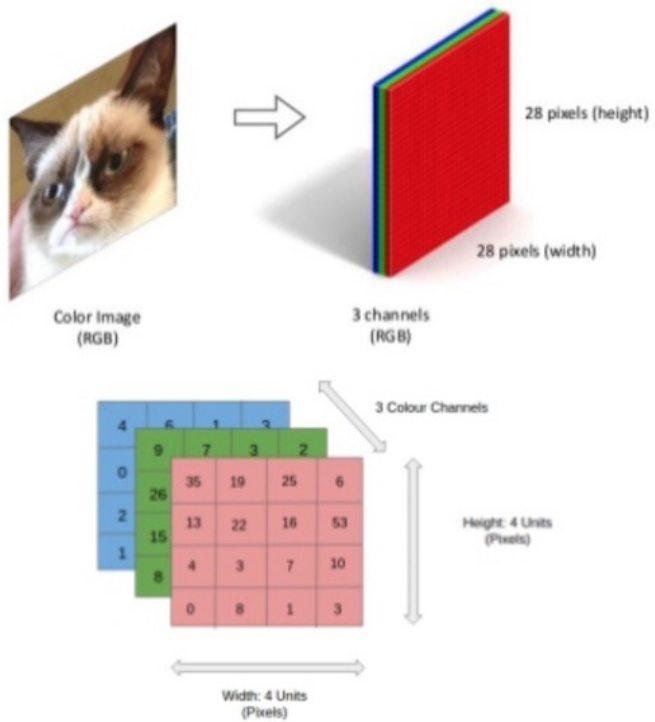
Theoretical Background

Vectors, Matrices, Tensors, Color Images

Vectors, matrices, tensors.



Color images are three-dimensional tensors.



Convolution Operations

Convolutions

Convolutions are mathematical operations that take an input signal (such as an image or datastream) and produce a transformed output signal that highlights certain features of the input.

Mathematical: A convolution operation can be represented:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau \tag{1}$$

Here:

- f and g represent the datastream/image and filter respectively.
- $*$ denotes with convolution operator.

Convolution Operations

1D Input Signals/Arrays:

- Applied to sequences or time series.
- Filter slides along the input signal and performs element-wise multiplication and accumulation at each position to produce output signal.

2D Input Signals/Arrays:

- Applied to greyscale/color images.
- Filter/kernel slides over the input array and performs element-wise multiplication and accumulation at each postion.

3D Input Signals/Arrays:

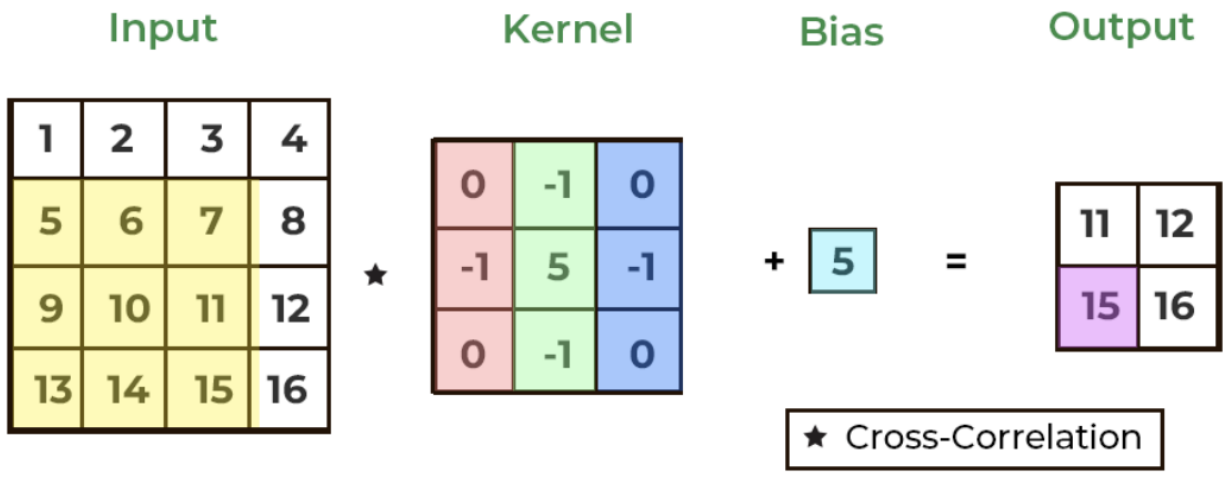
- Applied to videos and/or volumetric data.

2D Convolutions

2D Convolutions

A 2D convolution is a mathematical operation that applies a filter to an image, producing a filtered output (also called a feature map).

Simple Example: Sharpen filter ...



5 X 0 + 9 X -1 + 13 X 0 + 6 X -1 + 10 X 5 + 14 X -1 + 7 X 0 + 11 X -1 + 15 X 0 + 5 = 15

2D Convolutions

Filter/Kernel Design:

Operation	Kernel Matrix	Description
Identity	`[[0, 0, 0], [0, 1, 0], [0, 0, 0]]`	The central '1' preserves the original pixel value and the surrounding '0's ignore neighbors, leaving the image unchanged.
Edge Detection	`[[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]]`	Highlights edges by making areas with significant changes in pixel values very bright or dark, while uniform areas become black.
Sharpening	`[[0, -1, 0], [-1, 5, -1], [0, -1, 0]]`	Emphasizes differences between a central pixel and its neighbors, making the image features appear more distinct.
Blurring/Box Blur	`[[1/9, 1/9, 1/9], [1/9, 1/9, 1/9], [1/9, 1/9, 1/9]]`	Averages the pixel values with its neighbors, smoothing the image. The values sum to 1 to maintain the overall image brightness.
Sobel Edge Detection (Vertical)	`[[1, 0, -1], [2, 0, -2], [1, 0, -1]]`	Used to detect vertical edges by calculating the gradient in the x-direction. A related horizontal kernel (`[[1, 2, 1], [0, 0, 0], [-1, -2, -1]]`) detects horizontal edges.

2D Convolutions

Filter Applied to Greyscale Image:

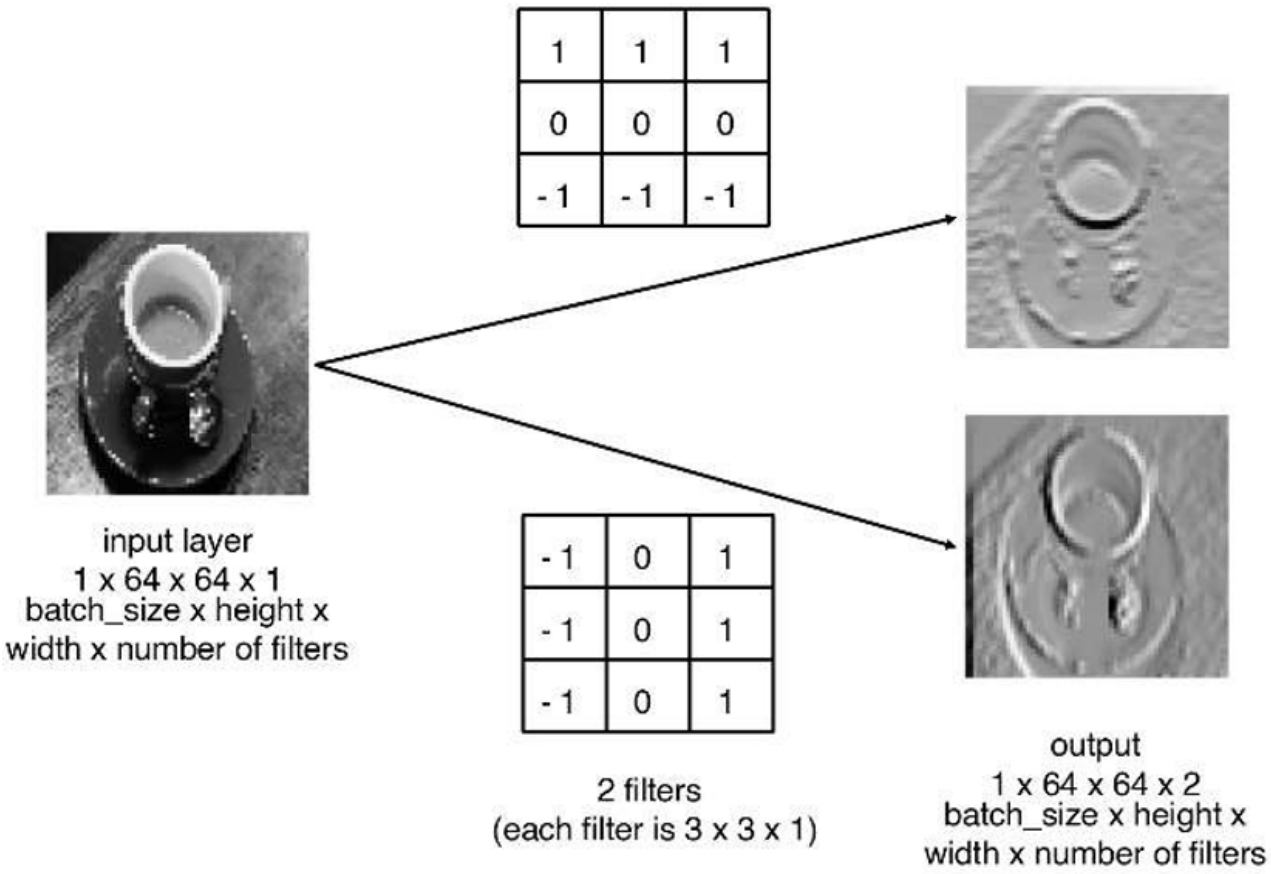


Figure 2-11. Two convolutional filters applied to a grayscale image

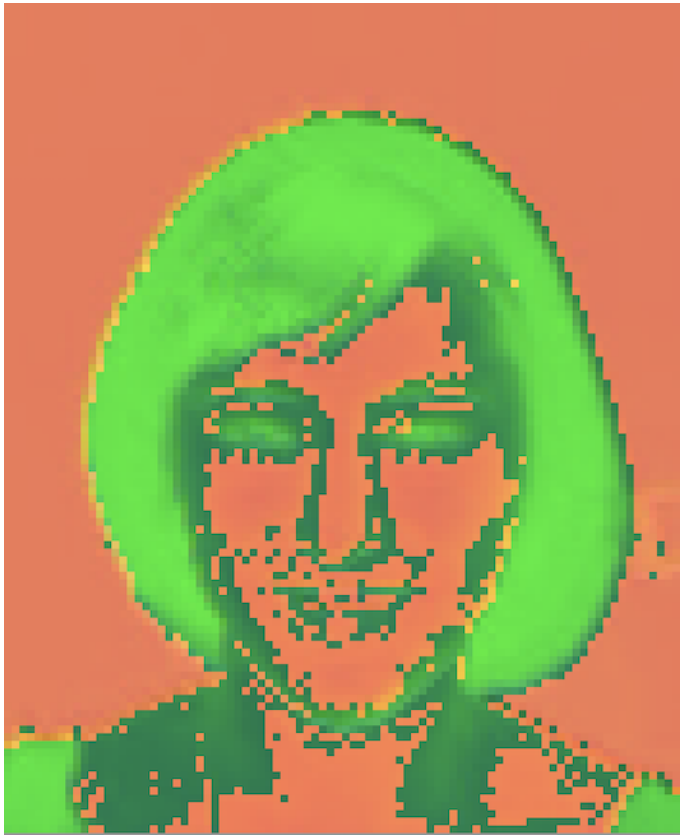
TensorFlow and Python Image Library (PIL)

```
1  # ============================================================================
2  # TestConvolution-Example03.py: Compute convolution (sharpening filter) on
3  # a real image.
4  #
5  # Source code: python-code-ai.d/convolution/TestConvolution2D-Example03.py
6  #
7  # Modified by: Mark Austin                                    January 2026
8  # ============================================================================
9
10 import torch
11 import torch.nn as nn
12 import torchvision.transforms as T
13 from PIL import Image
14
15 # main method ...
16
17 def main():
18     print("--- Read image file ... ");
19
20     imageFile = "faces/202593.jpg"
21     image = Image.open( imageFile )
22     image.show();
23
24     print("--- Convert image to torch tensor ... ");
25
26     Input = T.ToTensor()(image)
27
28     print("--- Unsqueeze image to make 4D tensor ... ");
```

TensorFlow and Python Image Library (PIL)

```
30        Input = Input.unsqueeze(0)
31        print('Input Tensor :',Input.shape)
32
33        # Assemble convolution operation ...
34
35        print("--- Assemble and compute convolution operation ... ");
36
37        conv = nn.Conv2d(in_channels=3, out_channels=3, kernel_size=3,stride=2, bias=True)
38        output = conv(Input)
39
40        print(output.shape)
41
42        Out_img = output.squeeze(0)
43
44        print("--- Convert tensor to image ... ");
45
46        Out_img = T.ToPILImage()(Out_img)
47
48        print("--- Show filtered image ... ");
49
50        Out_img.show();
51
52    # call the main method ...
53
54    if __name__ == "__main__":
55        main()
```

2D Convolutions



Convolution

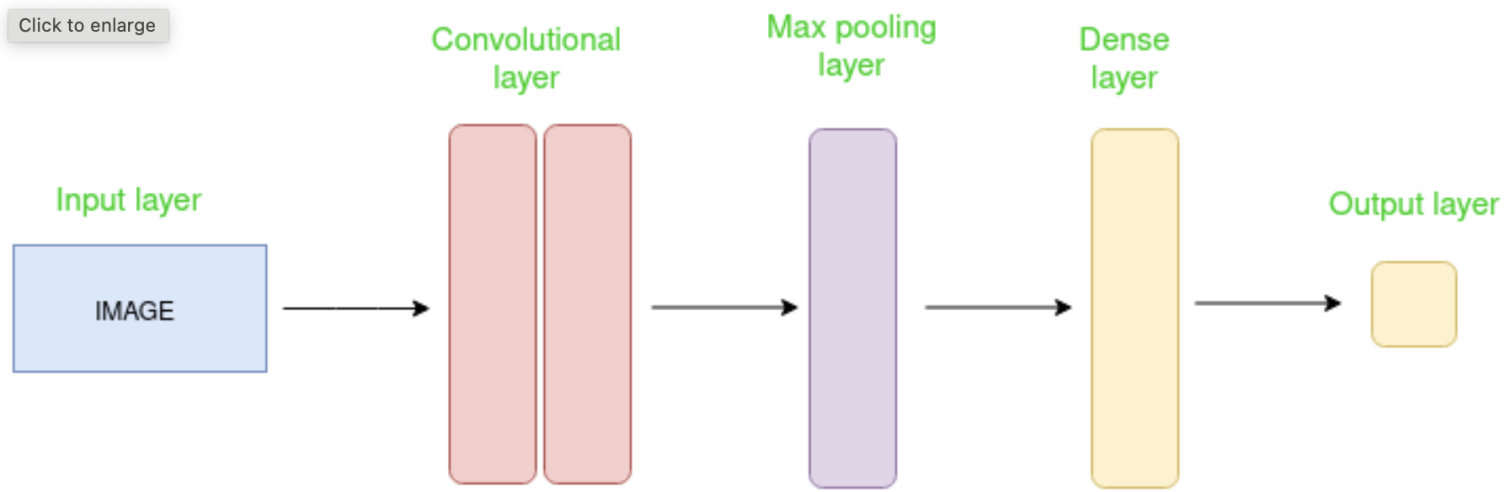
Neural Networks (CNNs)

Convolution Neural Networks (CNNs)

Convolution Neural Network

Convolution neural networks are multi-layer neural networks, designed to analyze and interpret images comprised of complex patterns, shapes and textures.

Simple CNN Architecture:



Convolution Neural Networks (CNNs)

Simple CNN Architecture Layers:

- **Input Layer:** Receive raw image data, usually as tensors with dimensions width $\times$ height $\times$ channels (e.g., $224 \times 224 \times 3$).
- **Convolution Layers:** Small learnable matrices (e.g., 5×5) slide over the image input, automatically detecting features such as edges or textures or shapes.
- **Max Pooling Layer:** Downsampling operation to reduce feature map dimensions (width/height) by selecting the maximum value from sub-regions (e.g., 2-by-2 window).
- **Dense Layers:** After feature extraction, these layers make the final prediction based on the detected patterns.
- **Softmax Output:** Converts the network output into probabilities, showing the likelihood of each class.

Convolution Neural Networks (CNNs)

Deep Learning Conceptual Diagram: Detect features in images, starting with edges and progressing to more complex patterns.

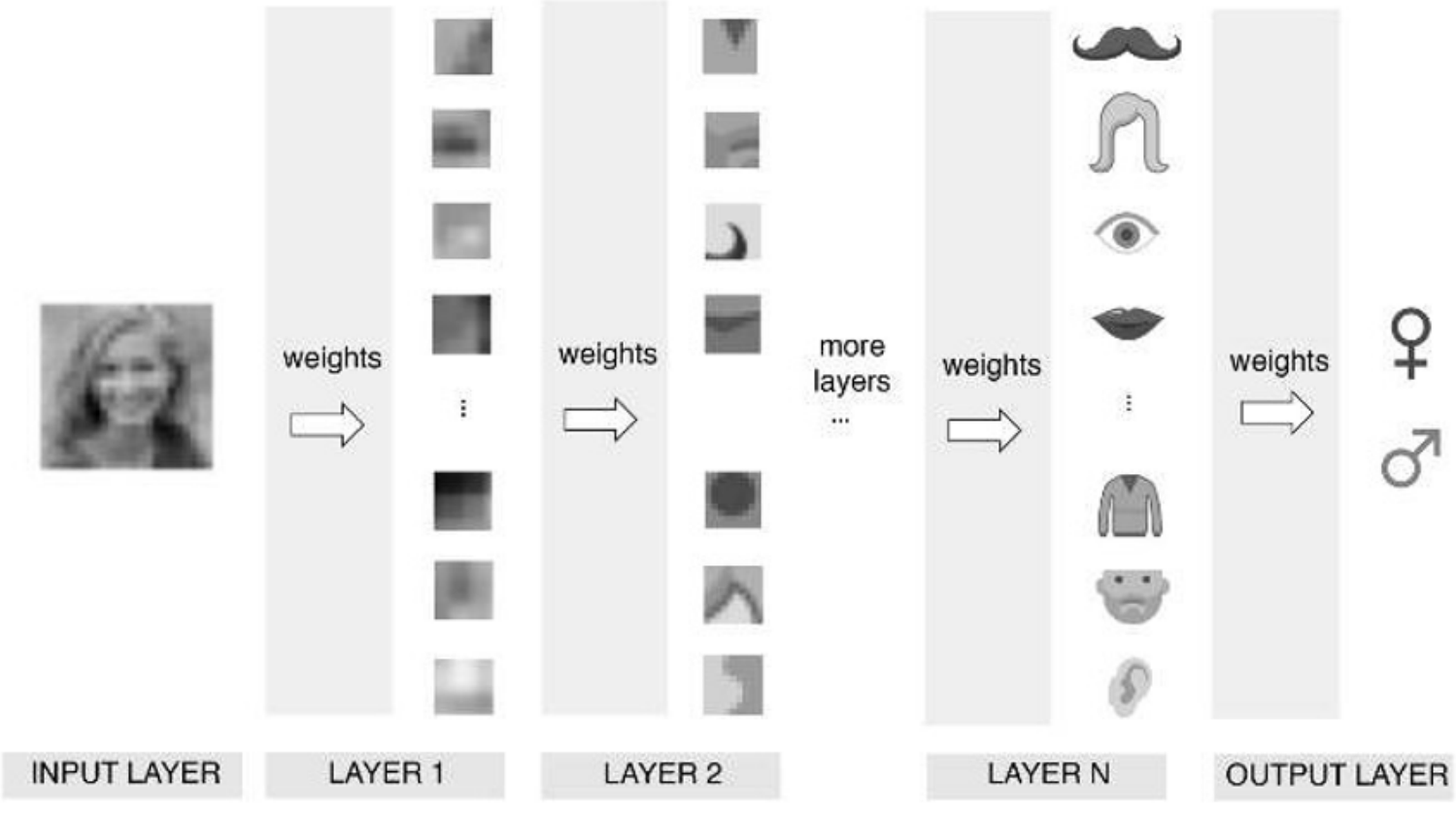


Figure 2-2. Deep learning conceptual diagram

Convolution Neural Networks (CNNs)

Convolution Architecture: Detect features in images, starting with edges and progressing to more complex patterns.

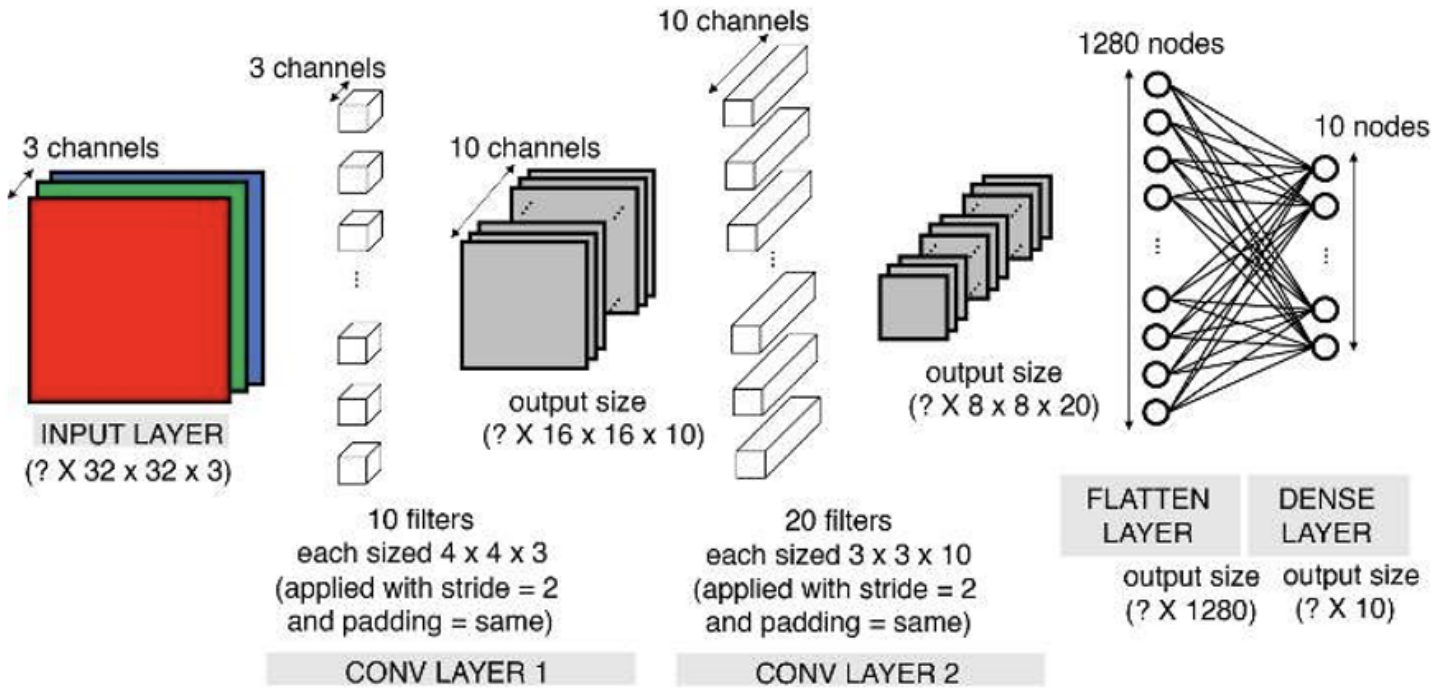


Figure 2-13. A diagram of a convolutional neural network

Convolution Neural Networks (CNNs)

Source Code Outline:

```
from tensorflow.keras import layers, models;

input_layer01   = layers.Input( shape = (None, 32, 32, 3) )
conv_layer01    = layers.Conv2D( filters=10, kernel_size=(4,4),
                                 strides=2, padding='same' ) (input_layer)

conv_layer02    = layers.Conv2D( filters=20, kernel_size=(3,3),
                                 strides=2, padding='same' ) (conv_layer01)

flatten_layer01 = layers.Flatten()( conv_layer02 )
output_layer01  = layers.Dense(units = 10, activation='softmax')( flatten_layer01 )

model = models.Model( input_layer01, output_layer01 )
```

Convolution Neural Networks (CNNs)

Source Code Outline:

....

Convolution Neural Networks (CNNs)

Convolution Architecture: Use learned information to classify the image as one of multiple predefined categories.

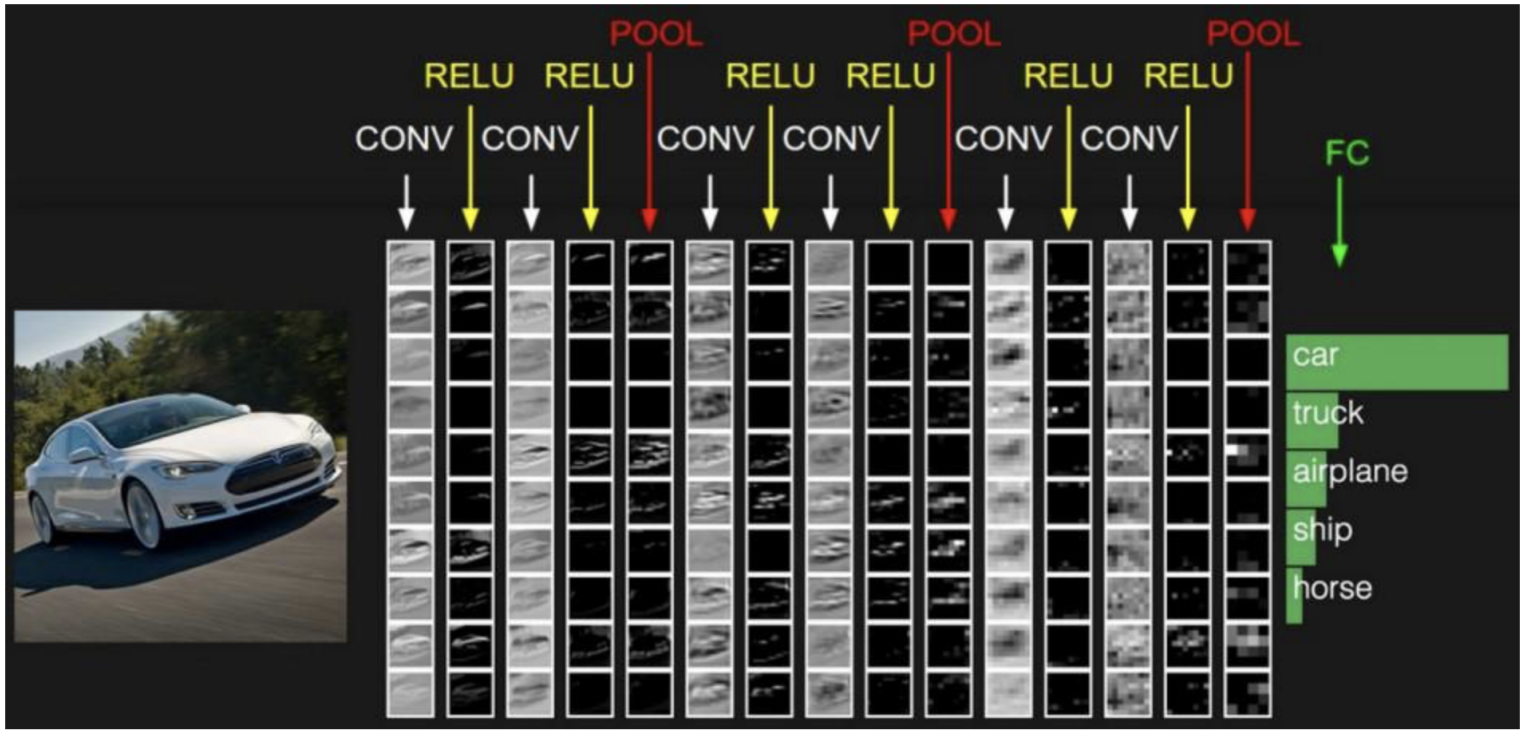


Image Source: cs231n.stanford.edu

Convolution Neural Networks (CNNs)

CNN Layers:

- **Convolution Layers:** Filters or kernels that detect features such as edges or textures.
- **ReLU Activations:** Adds nonlinearity to the model, helping to learn complex complex patterns.
- **Pooling Layers:** Reduce the dimensionality of the image (making the network more efficient while preserving important features).
- **Fully Connection Layers:** After feature extraction, these layers make the final prediction based on the detected patterns.
- **Softmax Output:** Converts the network output into probabilities, showing the likelihood of each class.

Convolution Operations/Layers

Dropout Layer

A dropout layer randomly select a fraction of input units to zero during training, thereby working to reduce overfitting and ultimately improving generalization.

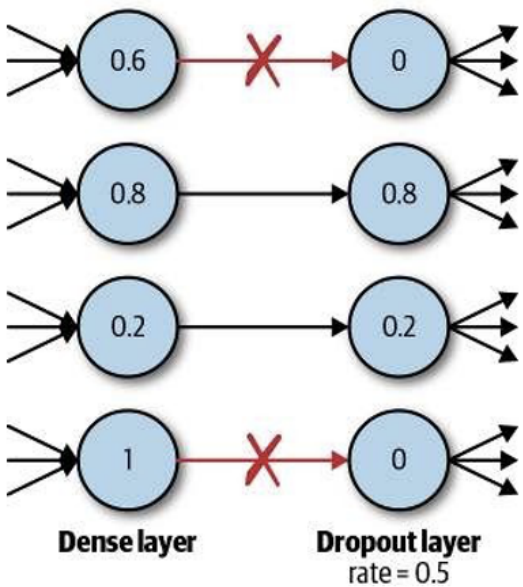


Figure 2-16. A dropout layer

CNN Summary

Advantages of CNNs:

- Automatic feature extraction.
- High accuracy in vision tasks.
- Translation invariance.

Disadvantages of CNNs:

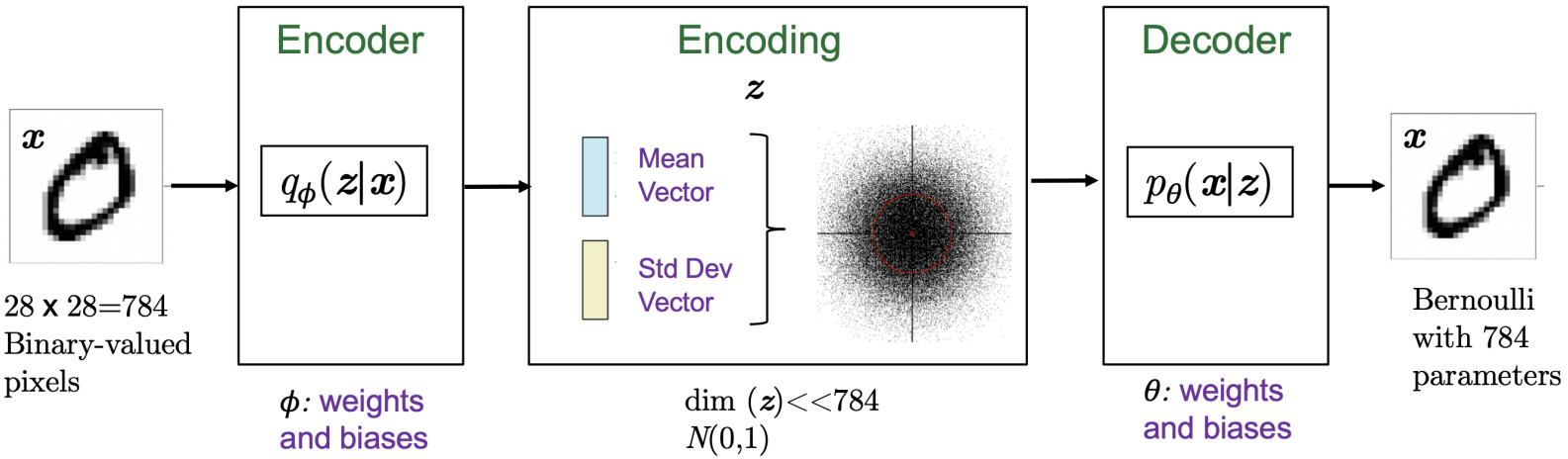
- Resource intensive training (taking days or weeks).
- Large data requirement.
- Overfitting risk – model performs well on trained data, poorly on new data.
- Traditional CNN architectures require input images to be of a fixed size.

Probability

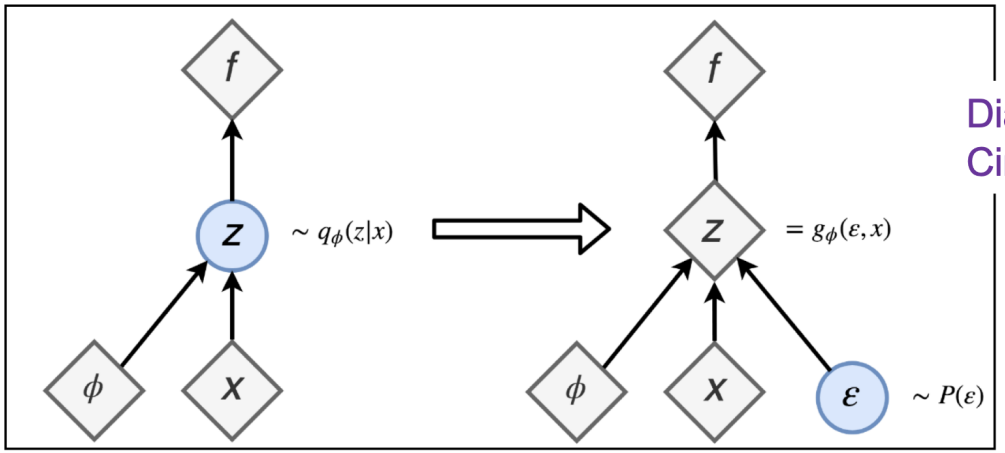
Estimation

Probability Estimation

Probability Estimation (fig01)

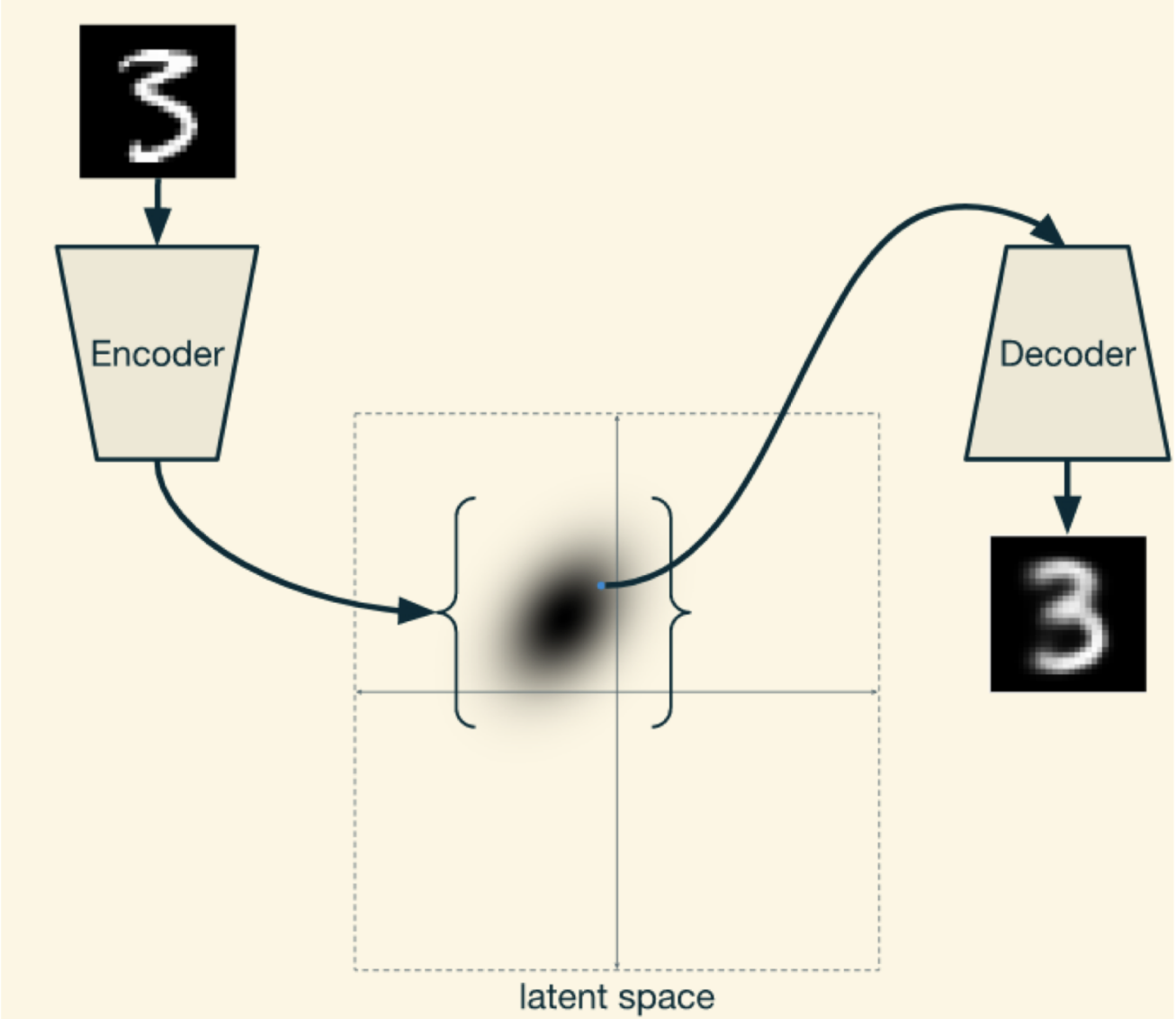


Probability Estimation (fig02)

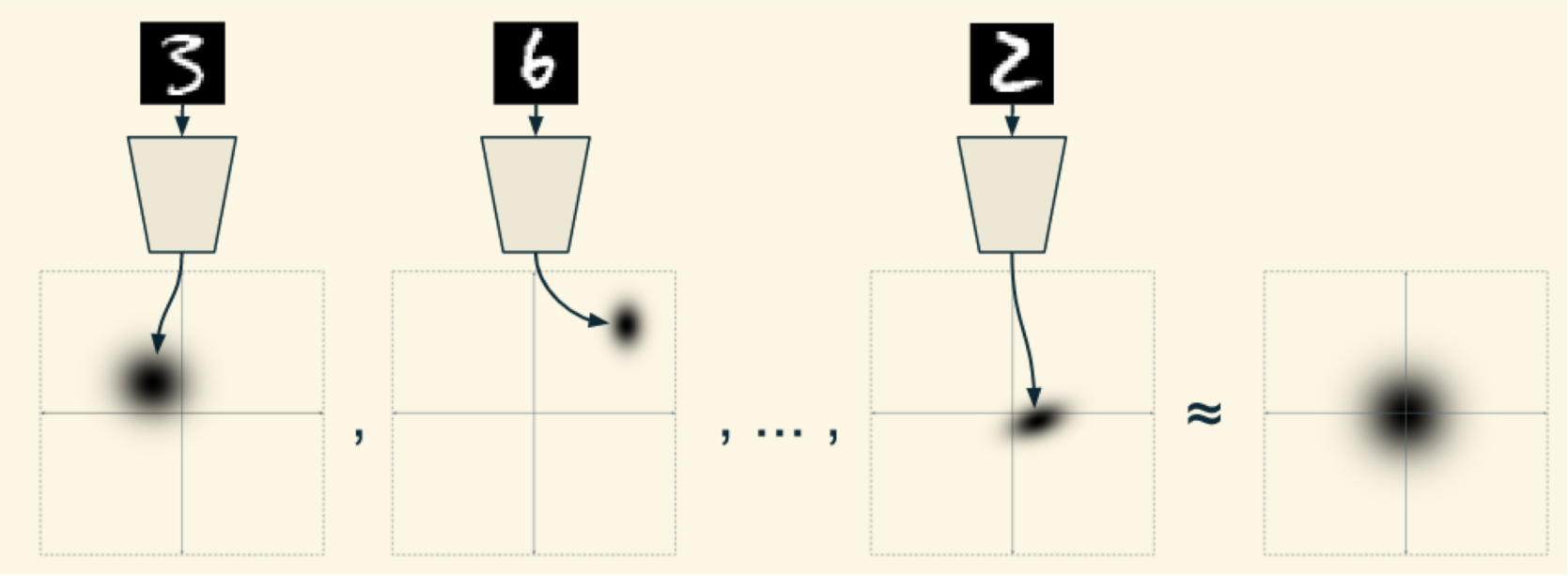


Diamonds: Deterministic dependencies
Circles: Random variables

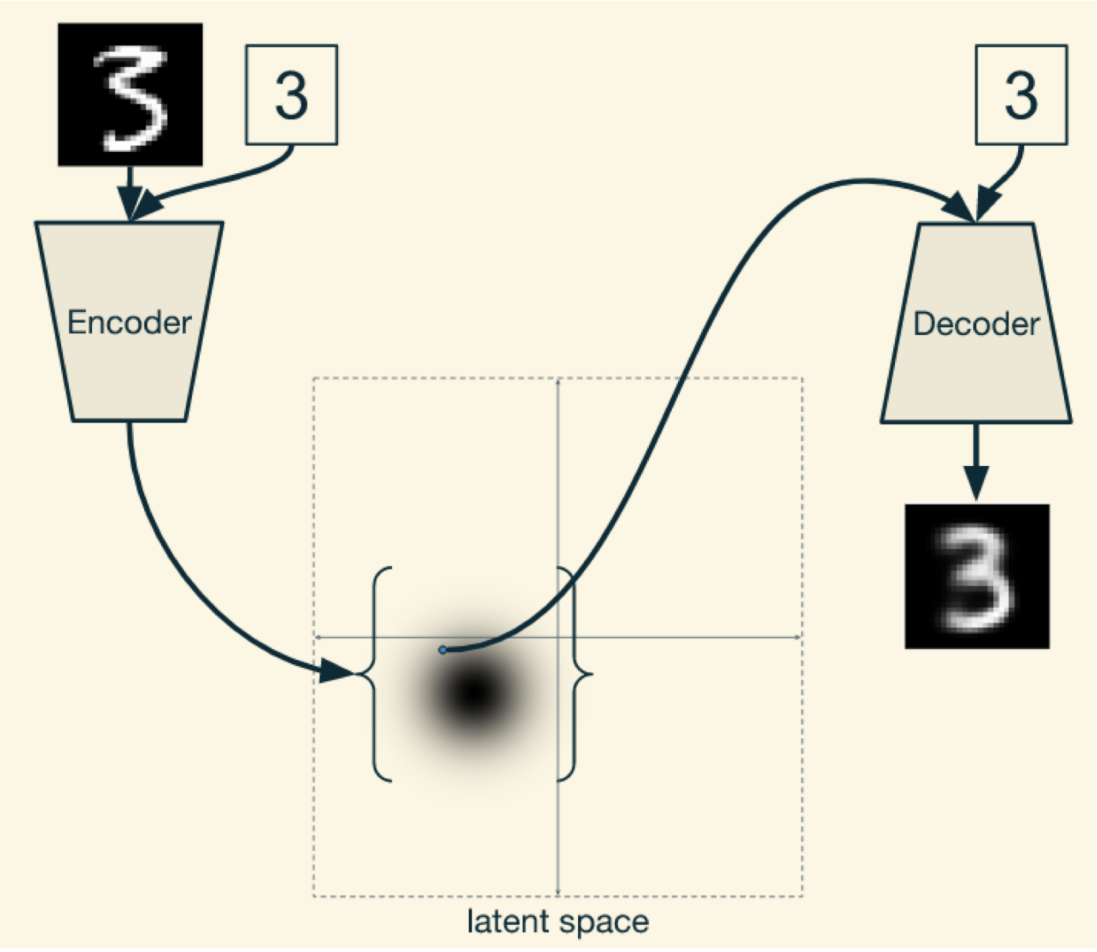
Probability Estimation (fig03)



Probability Estimation (fig04)

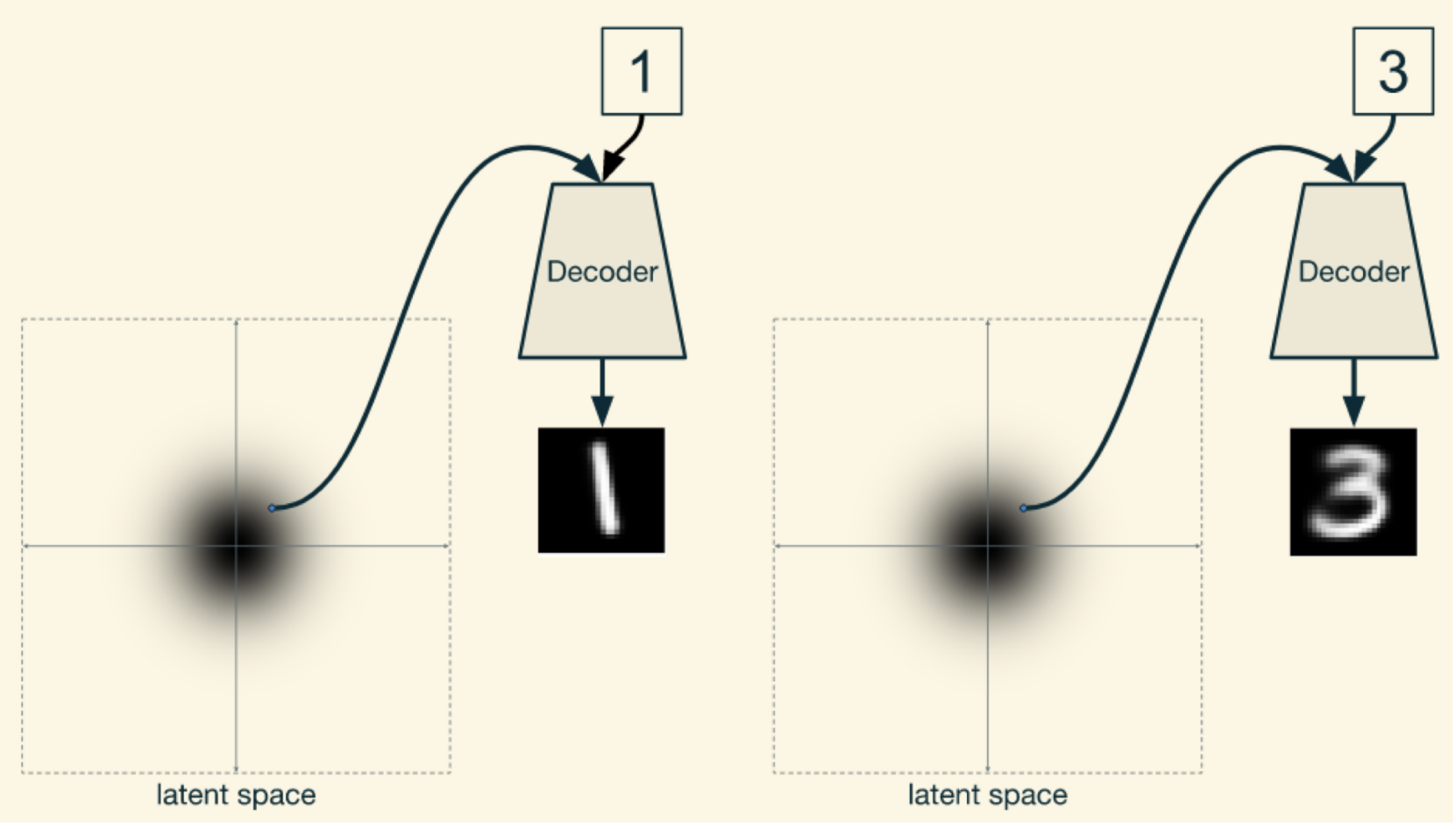


Probability Estimation (fig05)



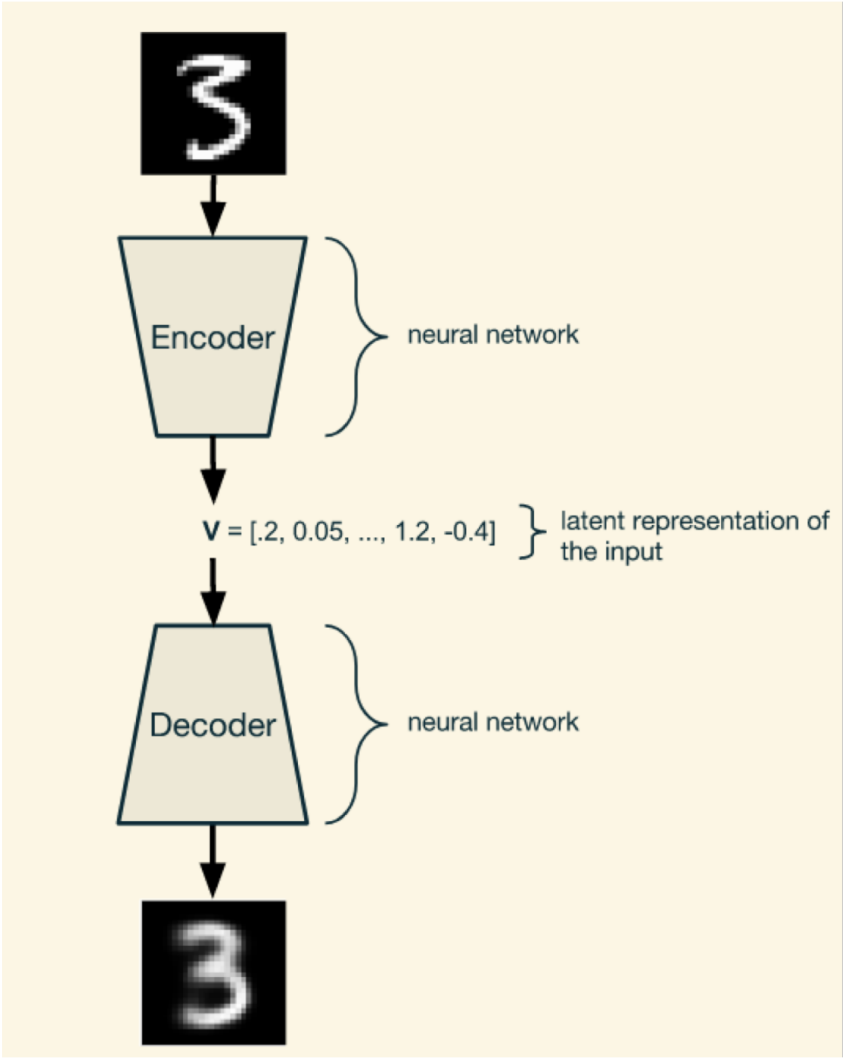
The number information is fed as a one-hot vector

Probability Estimation (fig06)



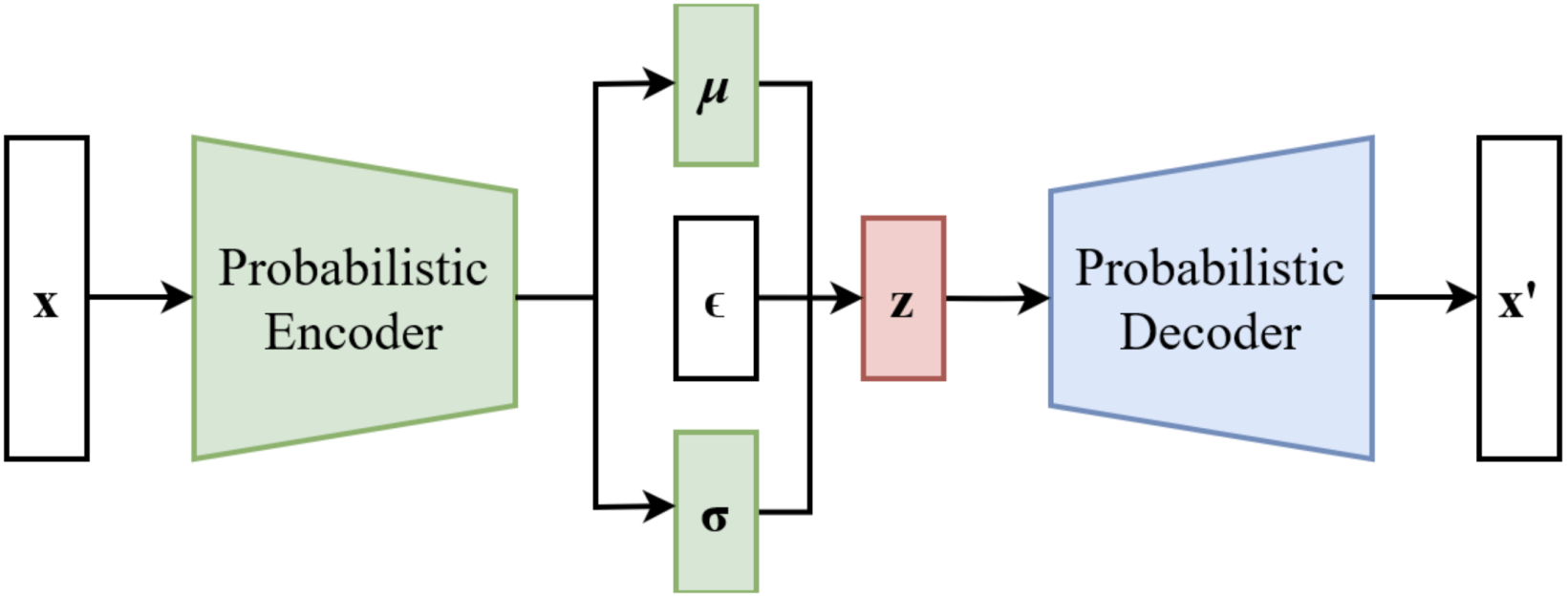
Probability Estimation (fig07)

Probability Estimation (fig08)

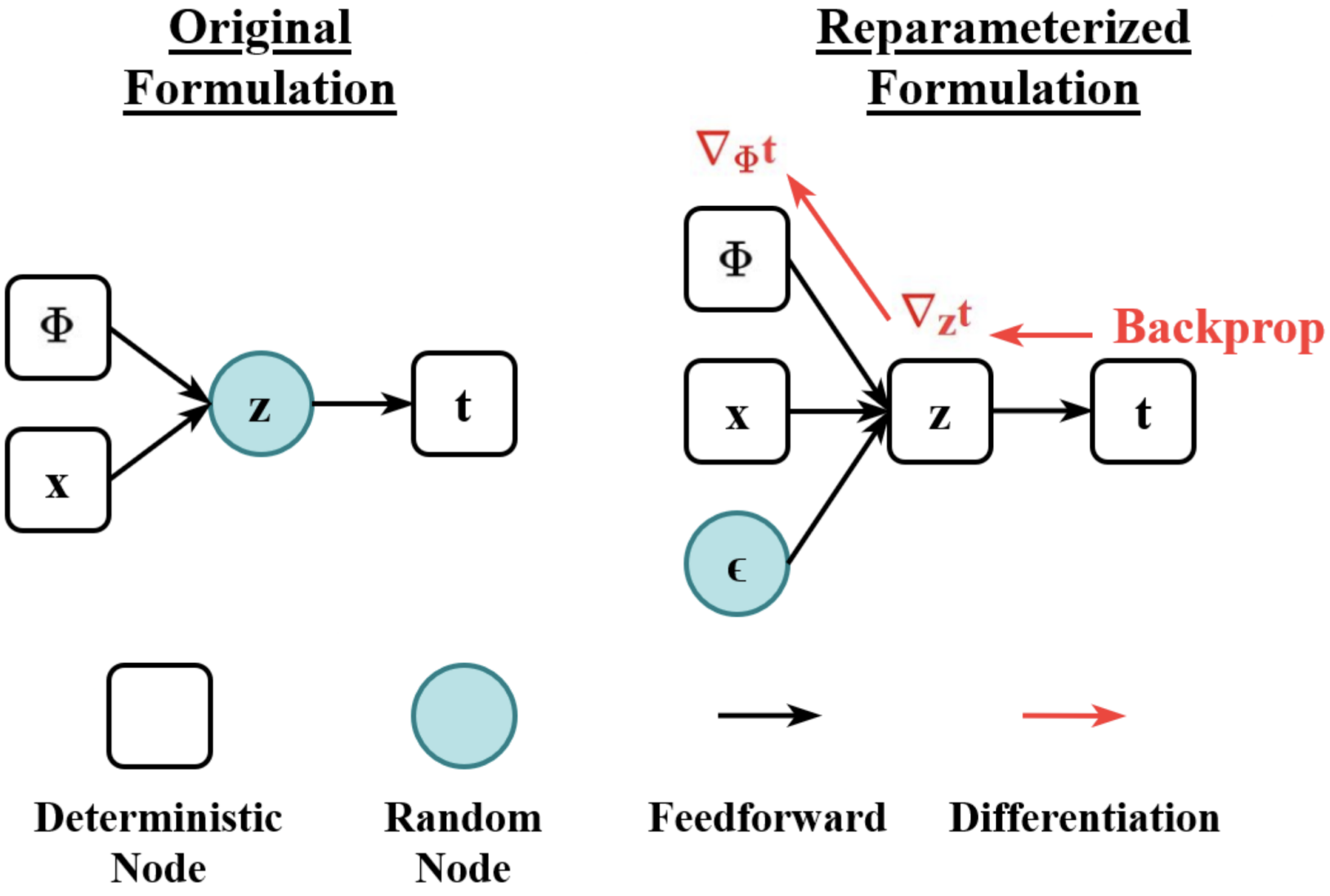


A standard autoencoder trained on MNIST digits may not provide a reasonable output when a V image is input

Probability Estimation (fig09)



Probability Estimation (fig10)



Examples

Example 1: Simplest Example

Problem Setup ....

Example 2: Generation of Fashion Shapes

Original Images: (28 x 28) pixels, greyscale.

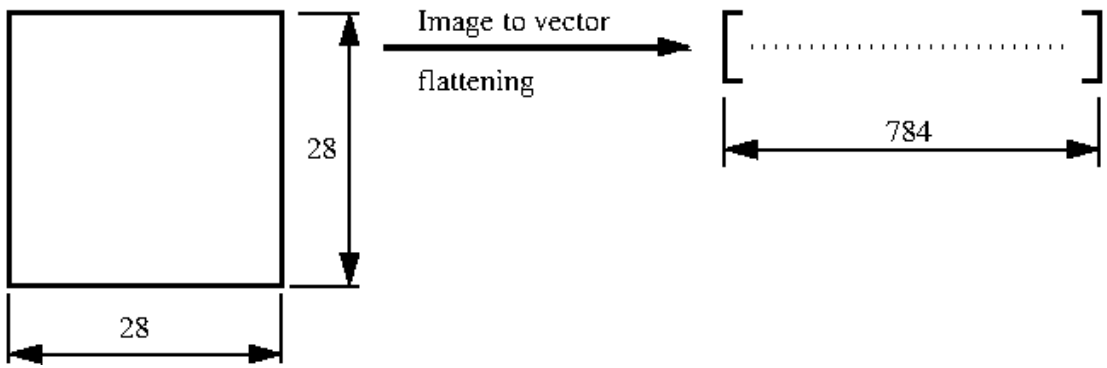
Reconstructions: (28 x 28) pixels, greyscale.

Fashion MNIST Dataset: A collection of greyscale images of clothing items, each (28 $\times$ 28) pixels, expanded to (32 $\times$ 32) pixels for processing. Individual pixels take a value 0 through 255.

Example 2: Generation of Fashion Shapes

Solution Procedure:

- Images are centered on a 28x28 grid of pixels. Individual pixels take a value 0 through 255.
- Individual 28x28 images $\longrightarrow$ 1x784 data vector.



- Create network configuration with 784 inputs/outputs, contracting down to a ten-dimensional embedding vector, i.e., 784 -> 250 -> 2 -> 250 -> 784.

Example 2: Generation of Fashion Shapes

--- Encoder summary ...
Model: "encoder"

Layer (type)	Output Shape	Param #	Connected to
encoder_input (InputLayer)	(None, 32, 32, 1)	0	-
conv2d (Conv2D)	(None, 16, 16, 32)	320	encoder_input[0][0]
conv2d_1 (Conv2D)	(None, 8, 8, 64)	18,496	conv2d[0][0]
conv2d_2 (Conv2D)	(None, 4, 4, 128)	73,856	conv2d_1[0][0]
flatten (Flatten)	(None, 2048)	0	conv2d_2[0][0]
z_mean (Dense)	(None, 2)	4,098	flatten[0][0]
z_log_var (Dense)	(None, 2)	4,098	flatten[0][0]
sampling (Sampling)	(None, 2)	0	z_mean[0][0], z_log_var[0][0]

Total params: 100,868 (394.02 KB)
Trainable params: 100,868 (394.02 KB)
Non-trainable params: 0 (0.00 B)

Example 2: Generation of Fashion Shapes

Source Code Outline:

```
Set Parameters
------------------------------------------------------------

VALIDATION_SPLIT = 0.2
IMAGE_SIZE       =  32
BATCH_SIZE       = 100
EMBEDDING_DIM    =   2
EPOCHS           =  30

Assemble Encoder
------------------------------------------------------------

encoder_input = layers.Input( shape=(IMAGE_SIZE, IMAGE_SIZE, 1), name="encoder_input" )

x = layers.Conv2D( 32, (3, 3), strides=2, activation="relu", padding="same")( encoder_input )
x = layers.Conv2D( 64, (3, 3), strides=2, activation="relu", padding="same")(x)
x = layers.Conv2D(128, (3, 3), strides=2, activation="relu", padding="same")(x)

shape_before_flattening = K.int_shape(x)[1:] # <--- the decoder will need this!
x = layers.Flatten()(x)

z_mean    = layers.Dense( EMBEDDING_DIM, name="z_mean")(x)
z_log_var = layers.Dense( EMBEDDING_DIM, name="z_log_var")(x)

z = Sampling()([z_mean, z_log_var])

encoder = models.Model(encoder_input, [z_mean, z_log_var, z], name="encoder")
```

Example 2: Generation of Fashion Shapes

--- Decoder summary ...
Model: "functional"

Layer (type)	Output Shape	Param #
decoder_input (InputLayer)	(None, 2)	0
dense (Dense)	(None, 2048)	6,144
reshape (Reshape)	(None, 4, 4, 128)	0
conv2d_transpose (Conv2DTranspose)	(None, 8, 8, 128)	147,584
conv2d_transpose_1 (Conv2DTranspose)	(None, 16, 16, 64)	73,792
conv2d_transpose_2 (Conv2DTranspose)	(None, 32, 32, 32)	18,464
decoder_output (Conv2D)	(None, 32, 32, 1)	289

Total params: 246,273 (962.00 KB)
Trainable params: 246,273 (962.00 KB)
Non-trainable params: 0 (0.00 B)

Example 2: Generation of Fashion Shapes

Source Code Outline:

```
Assemble Decoder
-----------------------------------------------------------

decoder_input = layers.Input( shape=(EMBEDDING_DIM,), name="decoder_input")

x = layers.Dense(np.prod(shape_before_flattening))(decoder_input)
x = layers.Reshape(shape_before_flattening)(x)

x = layers.Conv2DTranspose( 128, (3, 3), strides=2, activation="relu", padding="same")(x)
x = layers.Conv2DTranspose(  64, (3, 3), strides=2, activation="relu", padding="same")(x)
x = layers.Conv2DTranspose(  32, (3, 3), strides=2, activation="relu", padding="same")(x)

decoder_output = layers.Conv2D( 1, (3, 3), strides=1, activation="sigmoid",
                                padding="same", name="decoder_output",)(x)

decoder = models.Model(decoder_input, decoder_output)
```

Example 2: Generation of Fashion Shapes

Latent Space Data: `z_mean` and `z_log_var`

....

Example 2: Generation of Fashion Shapes

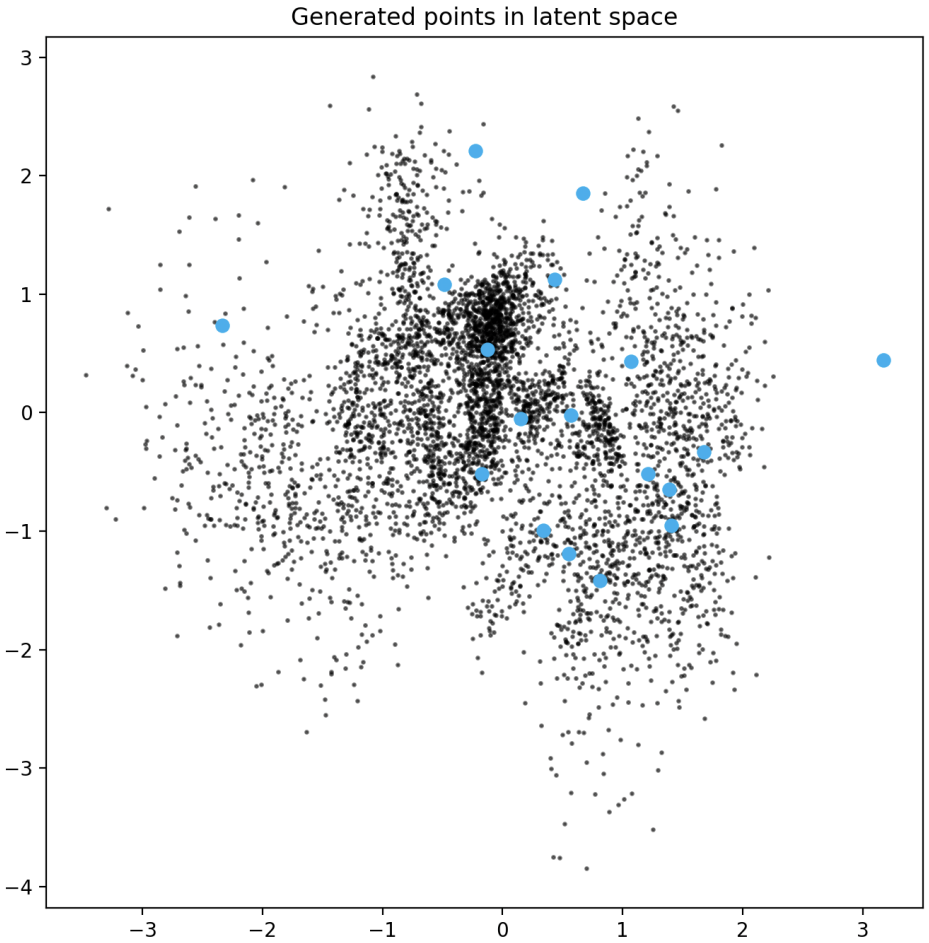
Sampling the Statistical Distribution:

```
class Sampling(layers.Layer):
    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch   = tf.shape(z_mean)[0]
        dim     = tf.shape(z_mean)[1]
        epsilon = K.random_normal(shape=(batch, dim))
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon
```

Simple Example:

Example 2: Generation of Fashion Shapes

Latent Space: Black dots show the `z_mean` value of each encoded image. Blue dots are sampled points in the latent space.



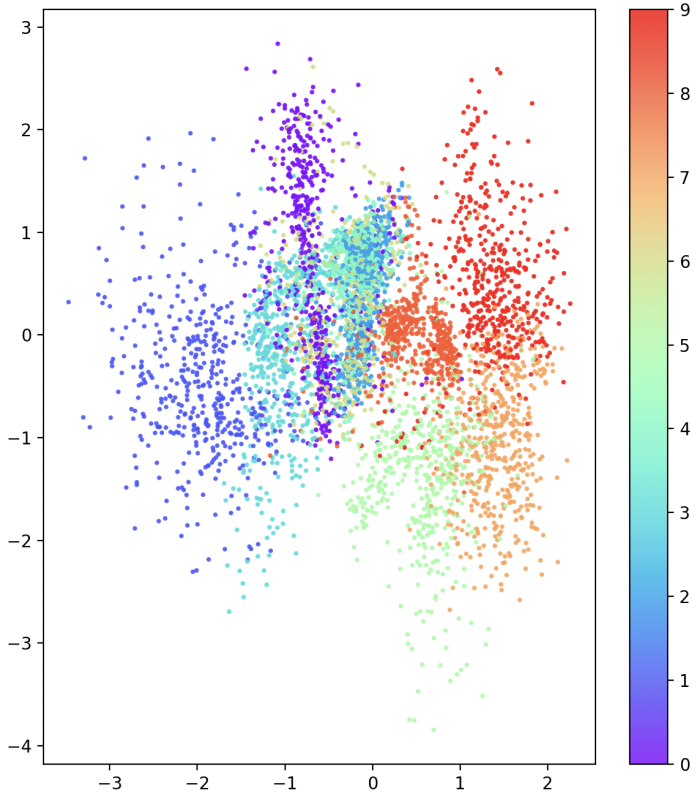
Example 2: Generation of Fashion Shapes (12)

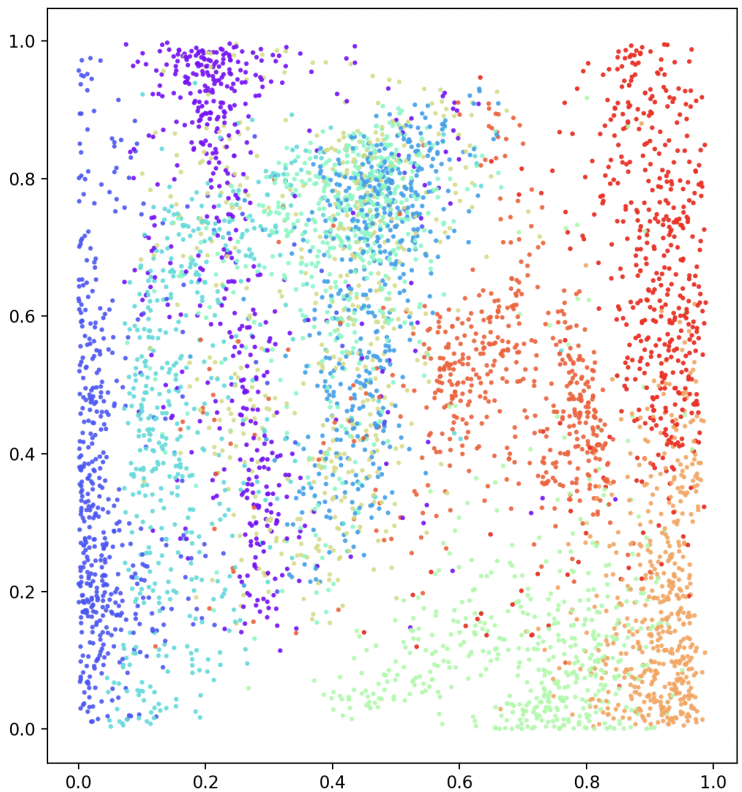
Latent Space Color Codes:

....

Example 2: Generation of Fashion Shapes (13)

Latent Space of VAE: Colored by clothing type ...

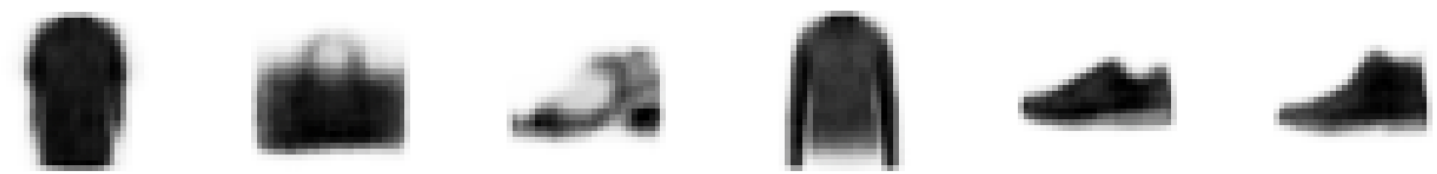




Example 2: Generation of Fashion Shapes

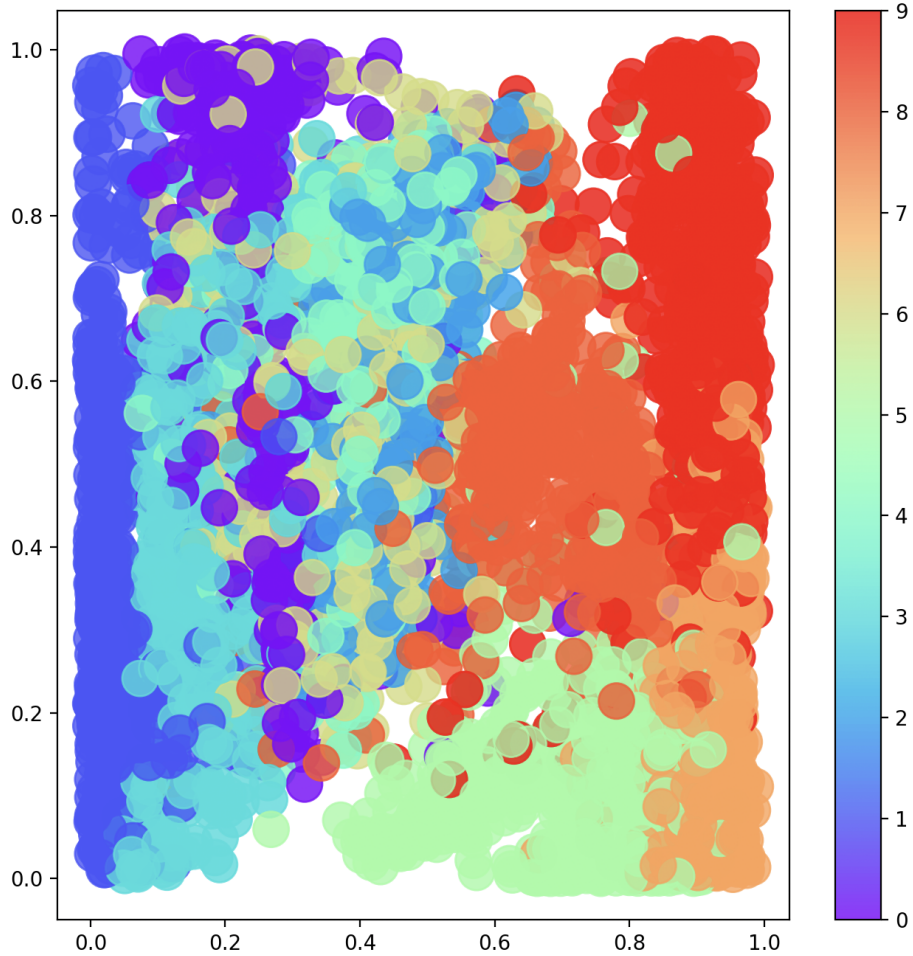
[0.7 1.9] [0.3 -1.] [3.2 0.4] [1.4 -0.6] [0.8 -1.4] [1.1 0.4]

[0.1 -0.] [0.4 1.1] [-2.3 0.7] [-0.2 -0.5] [1.2 -0.5] [-0.2 2.2]



[-0.5 1.1] [0.6 -0.] [0.5 -1.2] [-0.1 0.5] [1.4 -1.] [1.7 -0.3]

Example 2: Generation of Fashion Shapes



Example 3: Generation of Cartoon Faces

... insert pic ...

Source Code: python-code-ai.d/vae/TestVAE-Cartoon-Faces01.py

Example 3: Generation of Cartoon Faces

Encoder:

Model: "Encoder"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 256, 256, 3)	0	-
conv_1 (Conv2D)	(None, 128, 128, 32)	896	input_layer[0][0]
bn_1 (BatchNormalizatio…	(None, 128, 128, 32)	128	conv_1[0][0]
lrelu_1 (LeakyReLU)	(None, 128, 128, 32)	0	bn_1[0][0]
conv_2 (Conv2D)	(None, 64, 64, 64)	18,496	lrelu_1[0][0]
bn_2 (BatchNormalizatio…	(None, 64, 64, 64)	256	conv_2[0][0]
lrelu_2 (LeakyReLU)	(None, 64, 64, 64)	0	bn_2[0][0]
conv_3 (Conv2D)	(None, 32, 32,	36,928	lrelu_2[0][0]

Example 3: Generation of Cartoon Faces

Source Code Outline:

```
Set Parameters
------------------------------------------------------------

Assemble Encoder
------------------------------------------------------------
```

Example 3: Generation of Cartoon Faces

Decoder:

Model: "Decoder"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 200)	0
dense_1 (Dense)	(None, 4096)	823,296
Reshape (Reshape)	(None, 8, 8, 64)	0
conv_transpose_1 (Conv2DTranspose)	(None, 16, 16, 64)	36,928
bn_1 (BatchNormalization)	(None, 16, 16, 64)	256
lrelu_1 (LeakyReLU)	(None, 16, 16, 64)	0
conv_transpose_2 (Conv2DTranspose)	(None, 32, 32, 64)	36,928
bn_2 (BatchNormalization)	(None, 32, 32, 64)	256
lrelu_2 (LeakyReLU)	(None, 32, 32, 64)	0
conv_transpose_3 (Conv2DTranspose)	(None, 64, 64, 64)	36,928

Example 3: Generation of Cartoon Faces

Source Code Outline:

```
Assemble Decoder
-----------------------------------------------------------
```

Example 3: Generation of Cartoon Faces

Encoder: 1.77 million parameters.

Model: "Encoder"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 256, 256, 3)	0	-
conv_1 (Conv2D)	(None, 128, 128, 32)	896	input_layer[0][0]
bn_1 (BatchNormalizatio…	(None, 128, 128, 32)	128	conv_1[0][0]
lrelu_1 (LeakyReLU)	(None, 128, 128, 32)	0	bn_1[0][0]
conv_2 (Conv2D)	(None, 64, 64, 64)	18,496	lrelu_1[0][0]
bn_2 (BatchNormalizatio…	(None, 64, 64, 64)	256	conv_2[0][0]
lrelu_2 (LeakyReLU)	(None, 64, 64, 64)	0	bn_2[0][0]
conv_3 (Conv2D)	(None, 32, 32, 64)	36,928	lrelu_2[0][0]
bn_3 (BatchNormalizatio…	(None, 32, 32, 64)	256	conv_3[0][0]
lrelu_3 (LeakyReLU)	(None, 32, 32, 64)	0	bn_3[0][0]
conv_4 (Conv2D)	(None, 16, 16, 64)	36,928	lrelu_3[0][0]
bn_4 (BatchNormalizatio…	(None, 16, 16, 64)	256	conv_4[0][0]
lrelu_4 (LeakyReLU)	(None, 16, 16, 64)	0	bn_4[0][0]
conv_5 (Conv2D)	(None, 8, 8, 64)	36,928	lrelu_4[0][0]
bn_5 (BatchNormalizatio…	(None, 8, 8, 64)	256	conv_5[0][0]
lrelu_5 (LeakyReLU)	(None, 8, 8, 64)	0	bn_5[0][0]
flatten (Flatten)	(None, 4096)	0	lrelu_5[0][0]
mean (Dense)	(None, 200)	819,400	flatten[0][0]
log_var (Dense)	(None, 200)	819,400	flatten[0][0]

Total params: 1,770,128 (6.75 MB)
Trainable params: 1,769,552 (6.75 MB)
Non-trainable params: 576 (2.25 KB)

Decoder: 974,000 parameters.

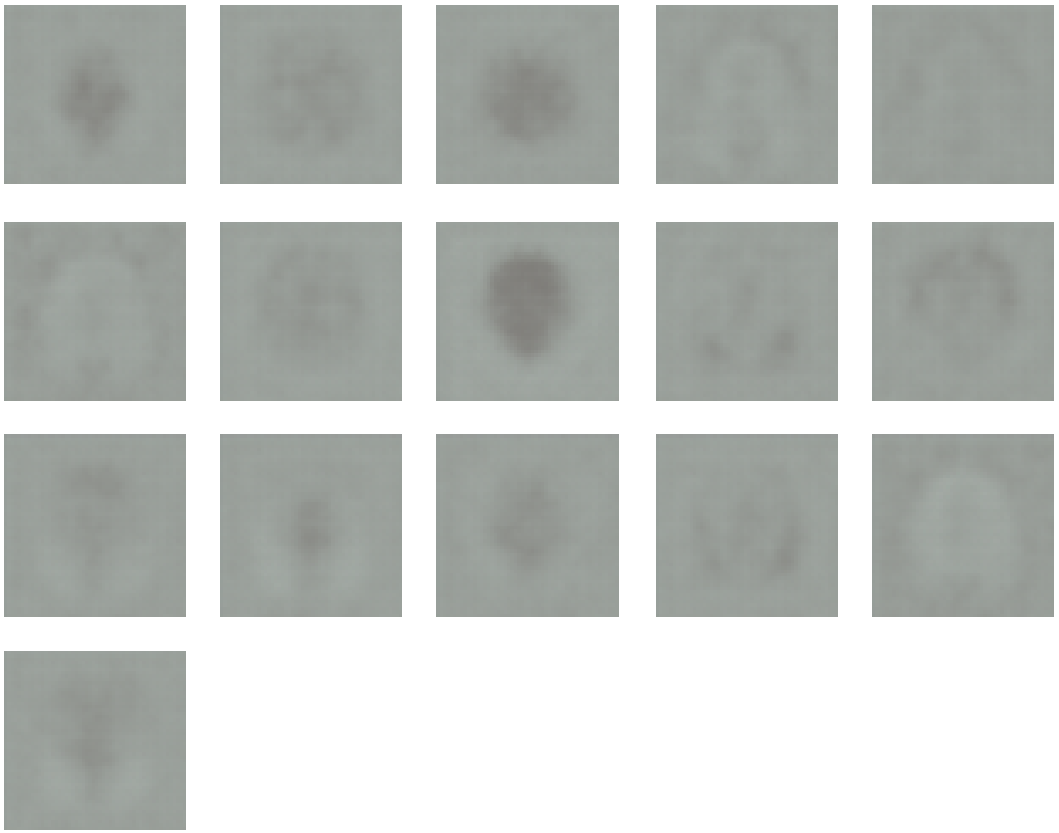
Model: "Decoder"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 200)	0
dense_1 (Dense)	(None, 4096)	823,296
Reshape (Reshape)	(None, 8, 8, 64)	0
conv_transpose_1 (Conv2DTranspose)	(None, 16, 16, 64)	36,928
bn_1 (BatchNormalization)	(None, 16, 16, 64)	256
lrelu_1 (LeakyReLU)	(None, 16, 16, 64)	0
conv_transpose_2 (Conv2DTranspose)	(None, 32, 32, 64)	36,928
bn_2 (BatchNormalization)	(None, 32, 32, 64)	256
lrelu_2 (LeakyReLU)	(None, 32, 32, 64)	0
conv_transpose_3 (Conv2DTranspose)	(None, 64, 64, 64)	36,928
bn_3 (BatchNormalization)	(None, 64, 64, 64)	256
lrelu_3 (LeakyReLU)	(None, 64, 64, 64)	0
conv_transpose_4 (Conv2DTranspose)	(None, 128, 128, 32)	18,464
bn_4 (BatchNormalization)	(None, 128, 128, 32)	128
lrelu_4 (LeakyReLU)	(None, 128, 128, 32)	0
conv_transpose_5 (Conv2DTranspose)	(None, 256, 256, 3)	867

Total params: 954,307 (3.64 MB)
Trainable params: 953,859 (3.64 MB)
Non-trainable params: 448 (1.75 KB)

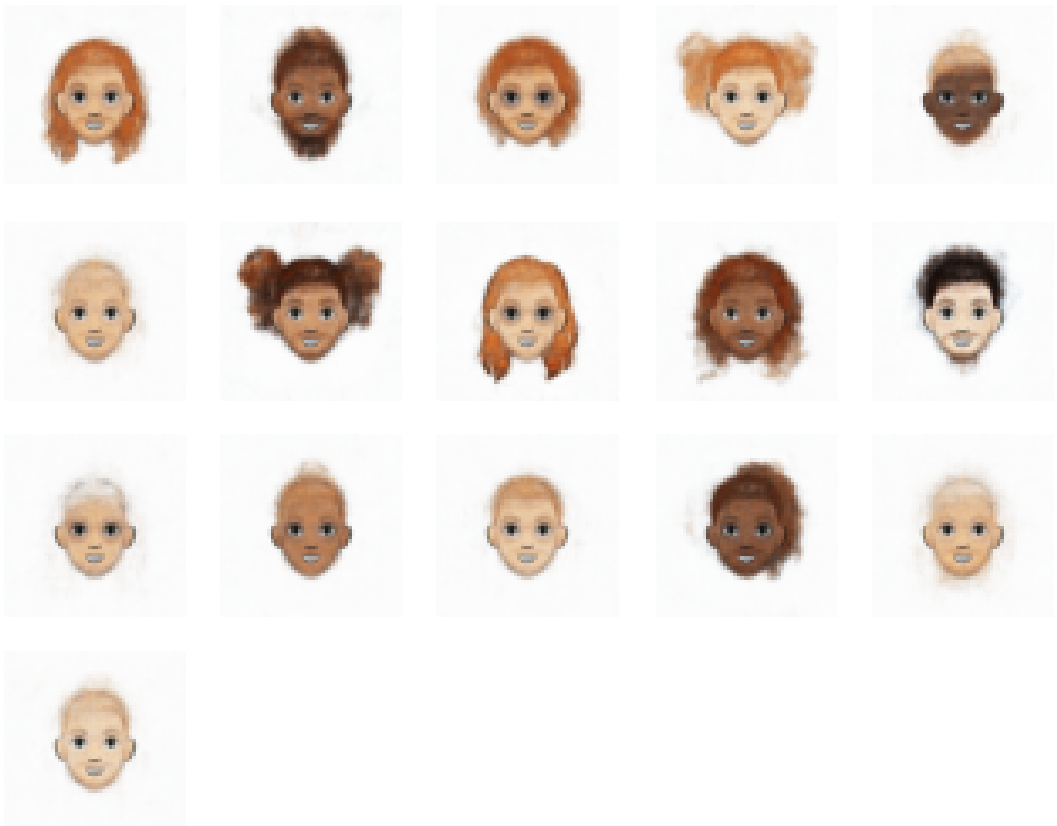
Example 3: Generation of Cartoon Faces

Iteration: 1



Example 3: Generation of Cartoon Faces

Iteration: 30



Example 4: Generation of Faces

CelebA Dataset: Collection of 200,000+ color images (i.e., three input channels of RGB instead of one greyscale):

Figure 3-15. Some examples from the CelebA dataset[5]

Example 4: Generation of Faces

Attributes and Values: Faces are annotated with labels

Sample Image:

Sample Attributes and Values:

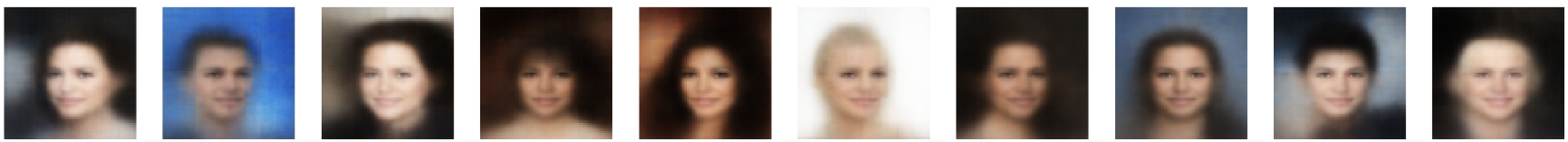
```
image_id            -->    202599.jpg
5_o_Clock_Shadow    -->           -1
Arched_Eyebrows     -->            1
Attractive          -->            1
Bags_Under_Eyes     -->           -1
Bald                -->           -1
Bangs               -->           -1

... Wavy_Hair, Wearing_Earrings,
Wearing_Hat, Wearing_Lipstick,
Wearing_Necklace, Wearing_Necktie, Young
```

Source Code: python-code-ai.d/vae/TestVAE-Faces01.py

Example 4: Generation of Faces

Original Images: (178 x 218) pixels, depth = 3.

Reconstructions: (64 x 64) pixels, depth = 3.



Example 4: Generation of Faces

Latent Space: ...

Example 4: Generation of Faces

Encoder: 20 layers; 253k parameters.

...

Model: "encoder"

Layer (type)	Output Shape	Param #	Connected to
encoder_input (InputLayer)	(None, 64, 64, 3)	0	–
conv2d (Conv2D)	(None, 32, 32, 64)	1,792	encoder_input[0][0]
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256	conv2d[0][0]
leaky_re_lu (LeakyReLU)	(None, 32, 32, 64)	0	batch_normalization[0][0]
conv2d_1 (Conv2D)	(None, 16, 16, 64)	36,928	leaky_re_lu[0][0]
batch_normalization_1 (BatchNormalization)	(None, 16, 16, 64)	256	conv2d_1[0][0]
leaky_re_lu_1 (LeakyReLU)	(None, 16, 16, 64)	0	batch_normalization_1[0][…
conv2d_2 (Conv2D)	(None, 8, 8, 64)	36,928	leaky_re_lu_1[0][0]
batch_normalization_2 (BatchNormalization)	(None, 8, 8, 64)	256	conv2d_2[0][0]
leaky_re_lu_2 (LeakyReLU)	(None, 8, 8, 64)	0	batch_normalization_2[0][…
conv2d_3 (Conv2D)	(None, 4, 4, 64)	36,928	leaky_re_lu_2[0][0]
batch_normalization_3 (BatchNormalization)	(None, 4, 4, 64)	256	conv2d_3[0][0]
leaky_re_lu_3 (LeakyReLU)	(None, 4, 4, 64)	0	batch_normalization_3[0][…
conv2d_4 (Conv2D)	(None, 2, 2, 64)	36,928	leaky_re_lu_3[0][0]
batch_normalization_4 (BatchNormalization)	(None, 2, 2, 64)	256	conv2d_4[0][0]
leaky_re_lu_4 (LeakyReLU)	(None, 2, 2, 64)	0	batch_normalization_4[0][…
flatten (Flatten)	(None, 256)	0	leaky_re_lu_4[0][0]
z_mean (Dense)	(None, 200)	51,400	flatten[0][0]
z_log_var (Dense)	(None, 200)	51,400	flatten[0][0]
sampling (Sampling)	(None, 200)	0	z_mean[0][0], z_log_var[0][0]

Total params: 253,584 (990.56 KB)
Trainable params: 252,944 (988.06 KB)
Non-trainable params: 640 (2.50 KB)

Example 4: Generation of Faces

Source Code Outline:

```
Set Parameters
----------------------------------------------------------

IMAGE_SIZE = 64
CHANNELS   = 3
BATCH_SIZE = 128
Z_DIM      = 200
EPOCHS     = 2
NUM_FEATURES  = 64
LEARNING_RATE = 0.0005
LOAD_MODEL    = False

Assemble Encoder
----------------------------------------------------------

encoder_input = layers.Input(
    shape=(IMAGE_SIZE, IMAGE_SIZE, CHANNELS), name="encoder_input"
)

x = layers.Conv2D(NUM_FEATURES, kernel_size=3, strides=2, padding="same")(
    encoder_input
)

x = layers.BatchNormalization()(x)
x = layers.LeakyReLU()(x)
x = layers.Conv2D(NUM_FEATURES, kernel_size=3, strides=2, padding="same")(x)
x = layers.BatchNormalization()(x)
```

Example 4: Generation of Faces

Source Code Outline: continued ...

```
x = layers.LeakyReLU()(x)
x = layers.Conv2D(NUM_FEATURES, kernel_size=3, strides=2, padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU()(x)
x = layers.Conv2D(NUM_FEATURES, kernel_size=3, strides=2, padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU()(x)
x = layers.Conv2D(NUM_FEATURES, kernel_size=3, strides=2, padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU()(x)

shape_before_flattening = K.int_shape(x)[1:]  # <--- the decoder will need this!

x = layers.Flatten()(x)
z_mean    = layers.Dense(Z_DIM, name="z_mean")(x)
z_log_var = layers.Dense(Z_DIM, name="z_log_var")(x)
z         = Sampling()([z_mean, z_log_var])

encoder = models.Model(encoder_input, [z_mean, z_log_var, z], name="encoder")
```

Example 4: Generation of Faces

Decoder: 21 layers; 240k parameters.

Model: "functional"

Layer (type)	Output Shape	Param #
decoder_input (InputLayer)	(None, 200)	0
dense (Dense)	(None, 256)	51,456
batch_normalization_5 (BatchNormalization)	(None, 256)	1,024
leaky_re_lu_5 (LeakyReLU)	(None, 256)	0
reshape (Reshape)	(None, 2, 2, 64)	0
conv2d_transpose (Conv2DTranspose)	(None, 4, 4, 64)	36,928
batch_normalization_6 (BatchNormalization)	(None, 4, 4, 64)	256
leaky_re_lu_6 (LeakyReLU)	(None, 4, 4, 64)	0
conv2d_transpose_1 (Conv2DTranspose)	(None, 8, 8, 64)	36,928
batch_normalization_7 (BatchNormalization)	(None, 8, 8, 64)	256
leaky_re_lu_7 (LeakyReLU)	(None, 8, 8, 64)	0
conv2d_transpose_2 (Conv2DTranspose)	(None, 16, 16, 64)	36,928
batch_normalization_8 (BatchNormalization)	(None, 16, 16, 64)	256
leaky_re_lu_8 (LeakyReLU)	(None, 16, 16, 64)	0
conv2d_transpose_3 (Conv2DTranspose)	(None, 32, 32, 64)	36,928
batch_normalization_9 (BatchNormalization)	(None, 32, 32, 64)	256
leaky_re_lu_9 (LeakyReLU)	(None, 32, 32, 64)	0
conv2d_transpose_4 (Conv2DTranspose)	(None, 64, 64, 64)	36,928
batch_normalization_10 (BatchNormalization)	(None, 64, 64, 64)	256
leaky_re_lu_10 (LeakyReLU)	(None, 64, 64, 64)	0
conv2d_transpose_5 (Conv2DTranspose)	(None, 64, 64, 3)	1,731

Total params: 240,131 (938.01 KB)
Trainable params: 238,979 (933.51 KB)
Non-trainable params: 1,152 (4.50 KB)

Example 4: Generation of Faces

Source Code Outline:

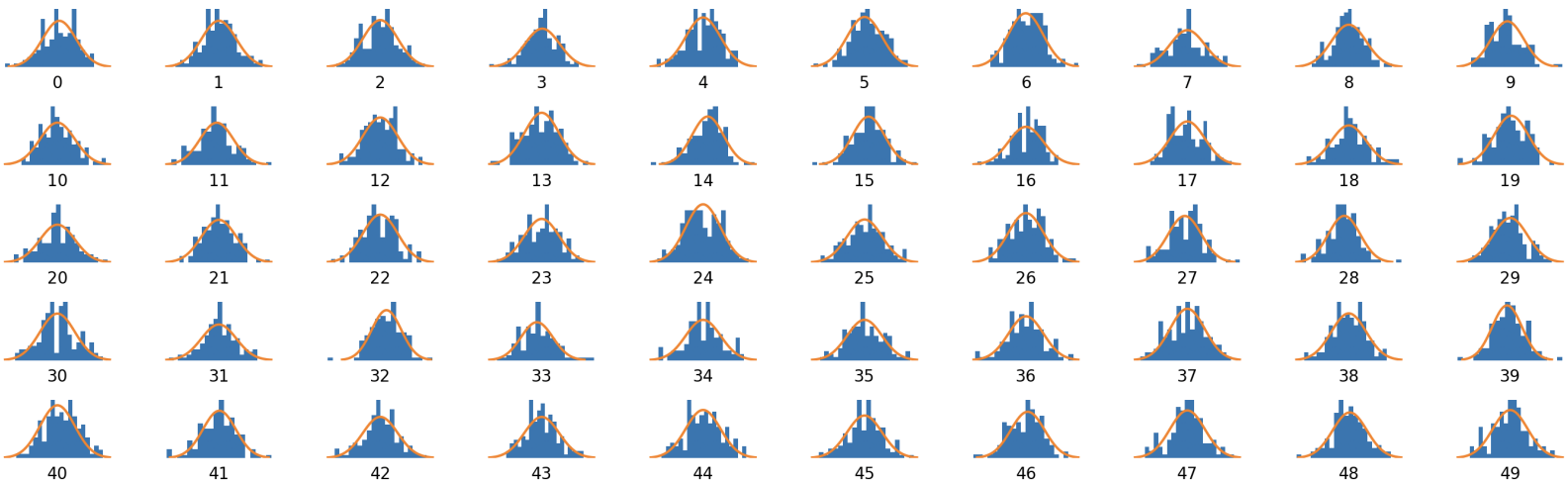
```
Assemble Decoder
-----------------------------------------------------------

decoder_input = layers.Input(shape=( Z_DIM,), name="decoder_input")

x = layers.Dense(np.prod(shape_before_flattening))(decoder_input)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU()(x)
x = layers.Reshape(shape_before_flattening)(x)
x = layers.Conv2DTranspose( NUM_FEATURES, kernel_size=3, strides=2, padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU()(x)
x = layers.Conv2DTranspose( NUM_FEATURES, kernel_size=3, strides=2, padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU()(x)
x = layers.Conv2DTranspose( NUM_FEATURES, kernel_size=3, strides=2, padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU()(x)
x = layers.Conv2DTranspose( NUM_FEATURES, kernel_size=3, strides=2, padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU()(x)
x = layers.Conv2DTranspose( NUM_FEATURES, kernel_size=3, strides=2, padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU()(x)

decoder_output = layers.Conv2DTranspose( CHANNELS, kernel_size=3, strides=1, activation="sigmoid", pad
decoder = models.Model(decoder_input, decoder_output)
```

Example 4: Generation of Faces

Latent Space Parameters: ...

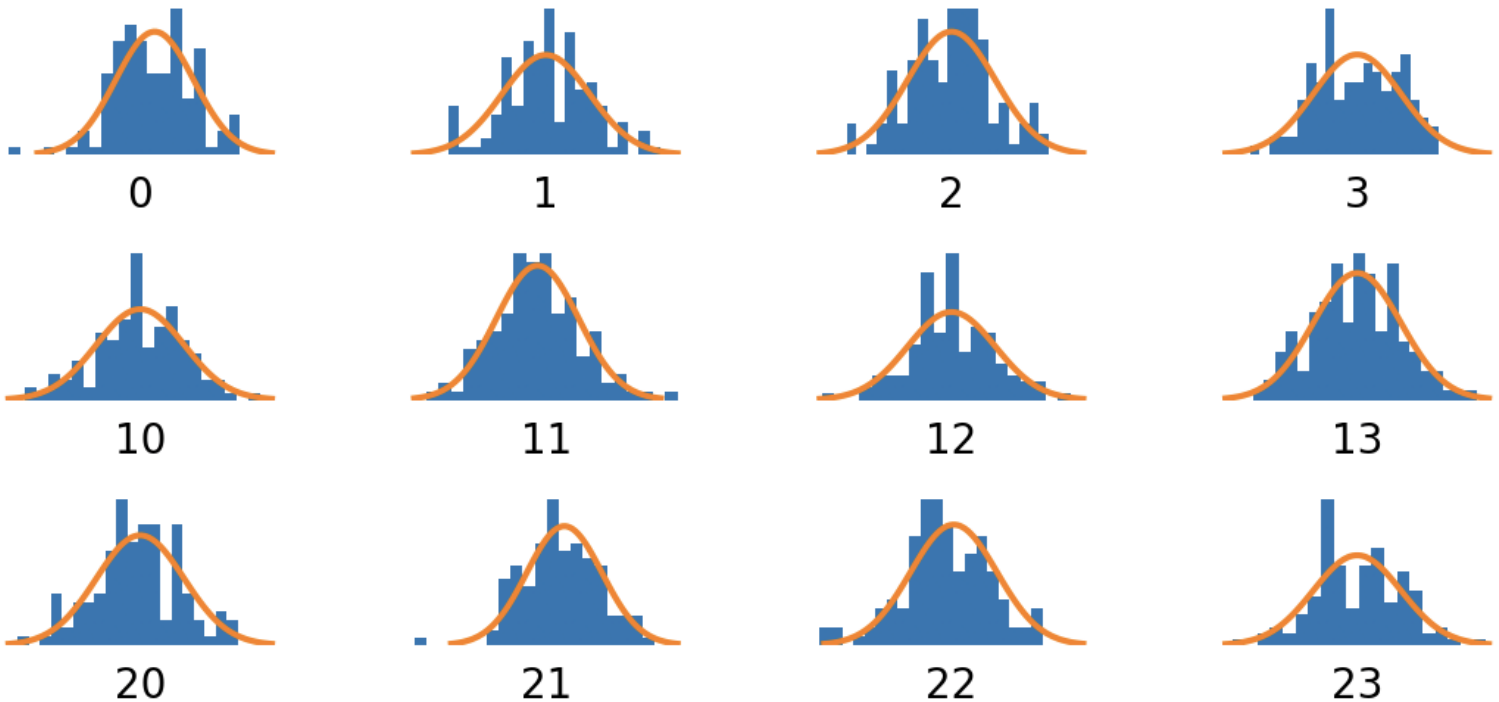
Example 4: Generation of Faces

Latent Distributions: Distributions of point for the first 50 dimensions of the latent space.



Example 4: Generation of Faces (fig 04)

Latent Distributions: Zoomed view of 12 dimensions of the latent space.



Example 4: Generation of Faces (faces01)

Generate New Faces:

Generating New Faces

To generate new faces, we can use the code in Example 3-9.

Example 3-9. Generating new faces from the latent space

```
n_to_show = 30

znew = np.random.normal(size = (n_to_show, VAE.z_dim))  (1)

reconst = VAE.decoder.predict(np.array(znew))  (2)

fig = plt.figure(figsize=(18, 5))
fig.subplots_adjust(hspace=0.4, wspace=0.4)
for i in range(n_to_show):
    ax = fig.add_subplot(3, 10, i+1)
    ax.imshow(reconst[i, :, :, :])  (3)
    ax.axis('off')

plt.show()
```

Example 4: Generation of Faces (faces02)

Generate New Faces:

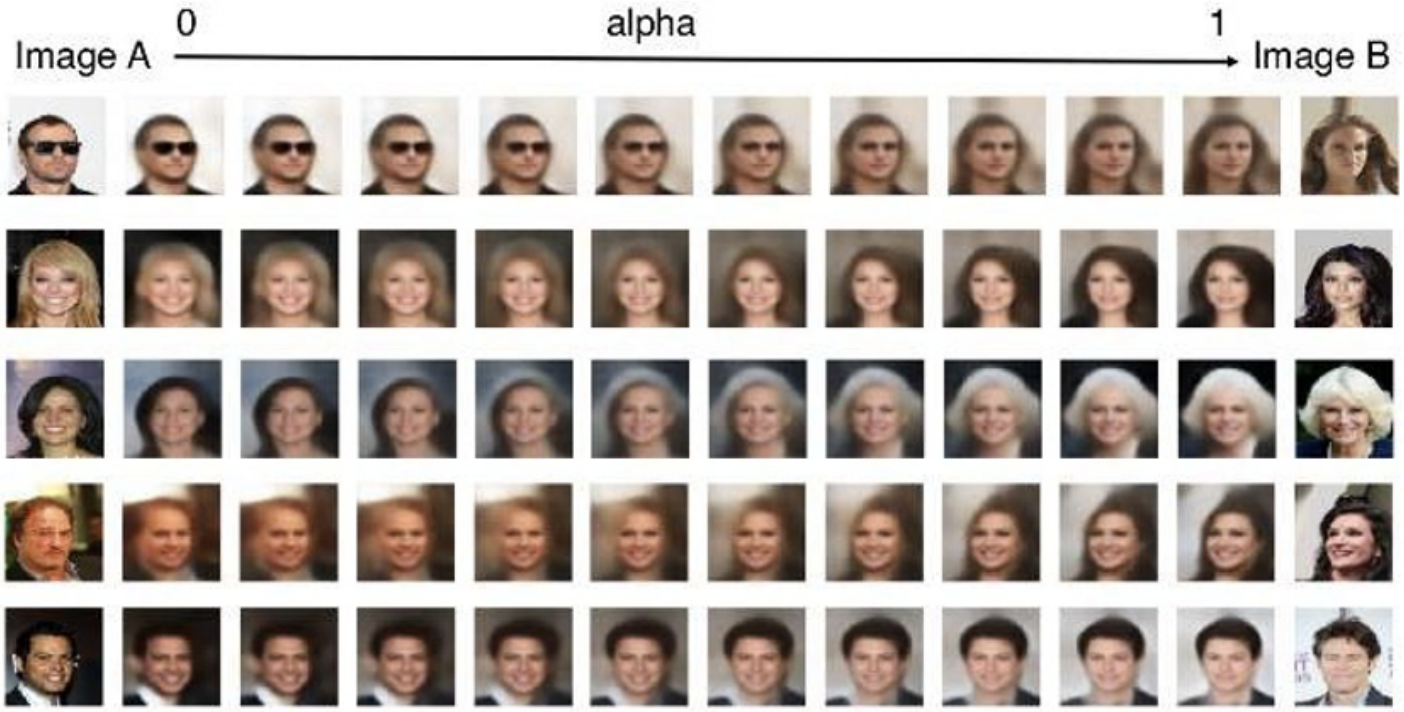


Figure 3-22. Morphing between two faces

Example 4: Generation of Faces (faces03)

Generate New Faces:

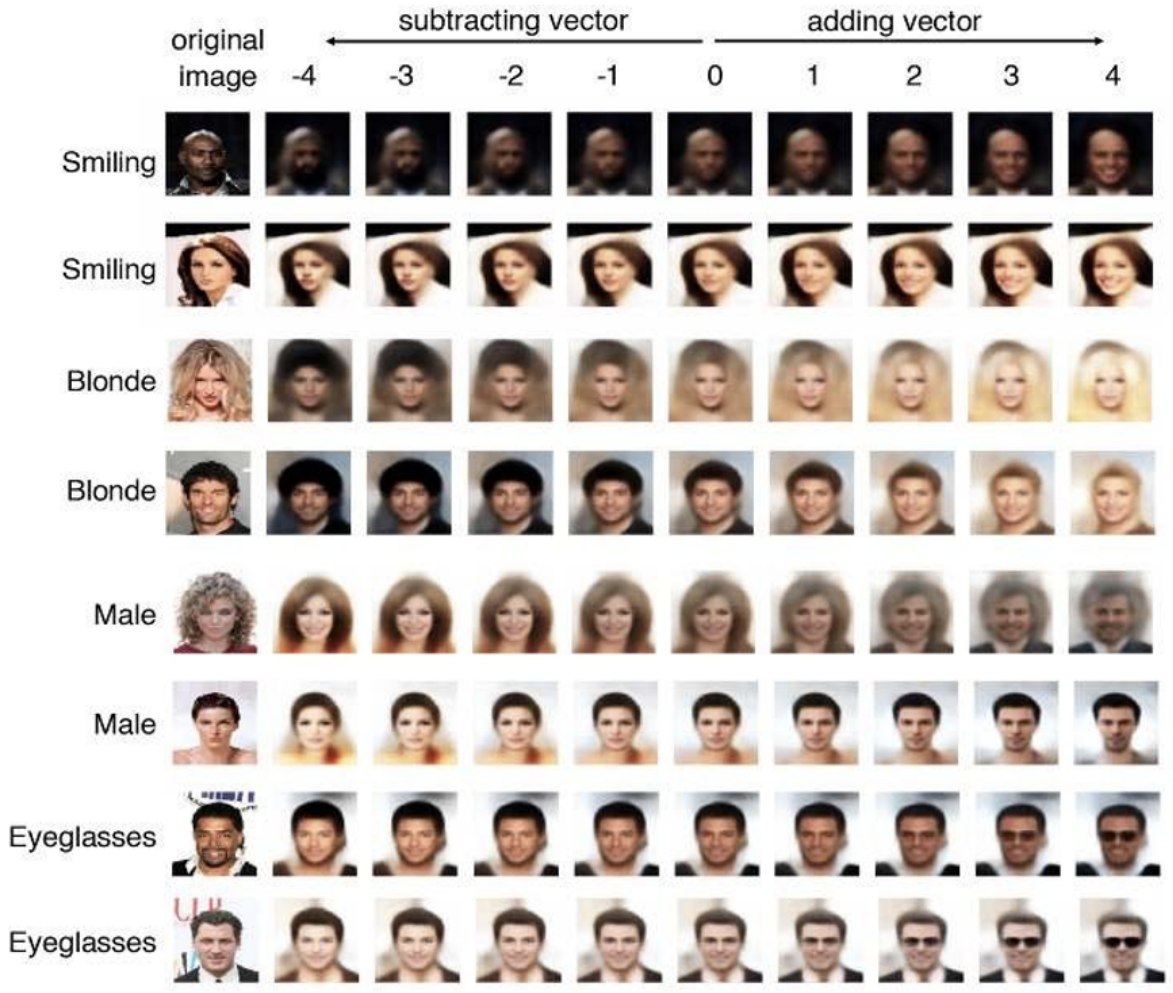


Figure 3-21. Adding and subtracting features to and from faces

References

- Foster D., Generative Deep Learning: Teaching Machines to Paint, Write, Compose and Play (2nd Edition), O'Reilly, 2024.
- Jordan J., Variational Autoencoders, Data Science, March 2018.
- Liu, Z., Large-Scale CelebFaces Attributes (CelebA) Dataset, 2015. See: http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html.
- Nugroho H., Susanty M., Irawan A., Koyimatu M., and Yunita A., Fully Convolutional Variational Autoencoder for Feature Extraction of Fire Detection System, Journal of Computer Science and Engineering, Vol. 13, No. 1, 2020.
- Watt J., Borhani R., Katsaggelos A.K., Machine Learning Refined, Second Edition, Cambridge University Press, 2020.