

Mark A. Austin

University of Maryland

austin@umd.edu

ENCE 688P, Spring Semester 2026

October 17, 2026

Overview

1 Variational AutoEncoders

- Basic Idea and Applications

2 Theoretical Background

- Convolution Operations
- Convolution Neural Networks (CNNs) and Image Analysis
- Probability Estimation and Sampling

3 Examples

- Example 1: Simplest Example
- Example 2: Generation of Fashion Shapes
- Example 3: Generation of Cartoon Faces
- Example 4: Generation of Faces

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ≡ ↺ 🔍 ↻

Basic Idea and Applications

Encoder

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

Decoder

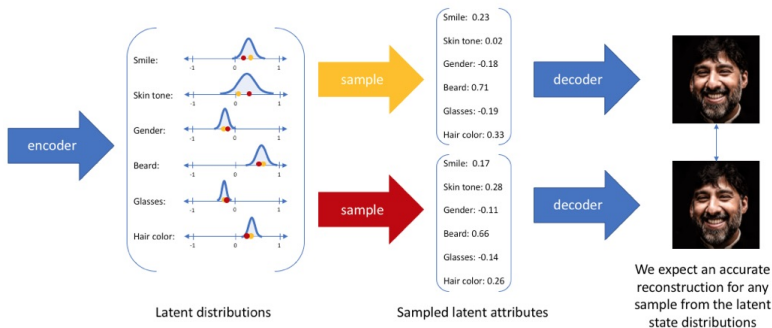
The **decoder** learns how to **reconstruct** the **input data** from the encoded representation and be as close to the input data as possible.

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

Image Reconstruction: sampled from latent distributions ...



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

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

A set of navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ≡ ↺ 🔍 ↻

Convolutions

Mathematical: A convolution operation can be represented:

Here:

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺

2D Convolutions

Simple Example: Sharpen filter ...



Filter/Kernel Design:

Filter Applied to Greyscale 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 ... ");

```

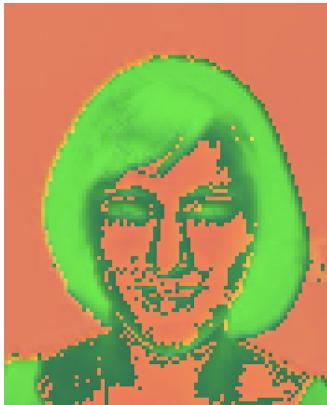
TensorFlow and Python Image Library (PIL)

```

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



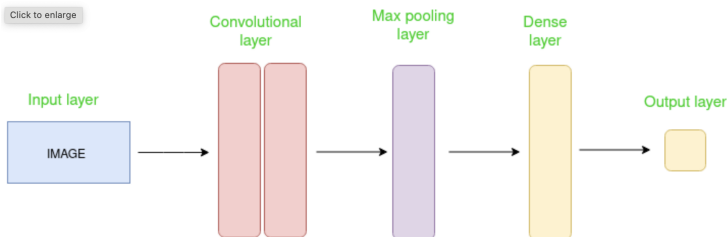
Neural Networks (CNNs)

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:



1. Introduction

Convolution Neural Networks (CNNs)

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

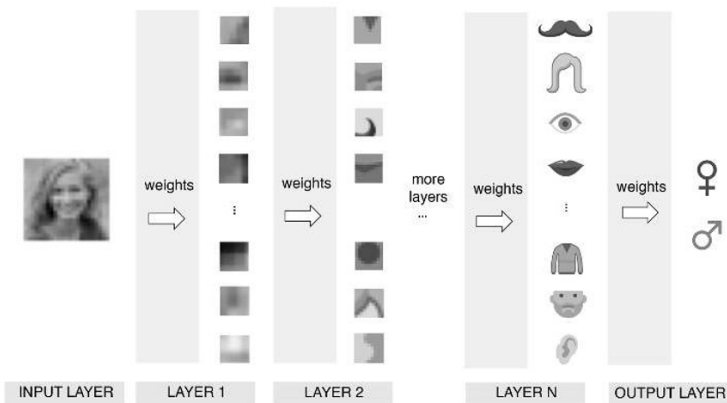
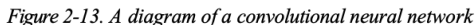


Figure 2-2. Deep learning conceptual diagram

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



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)

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

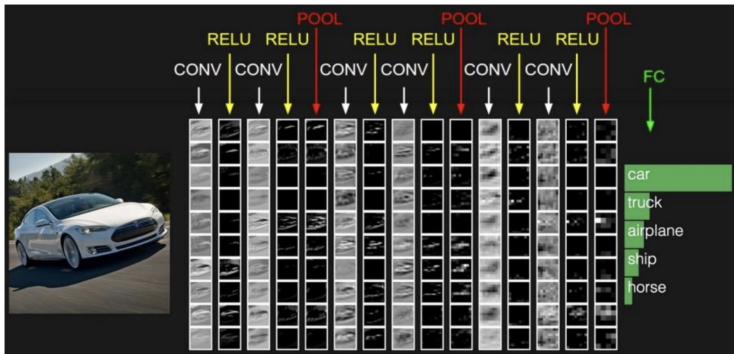


Image Source: cs231n.stanford.edu

9. Label File name: **Lab 10 - Lab 10**

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**.

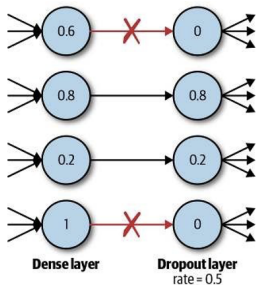


Figure 2-16. A dropout layer

CNN Summary

Advantages of CNNs:

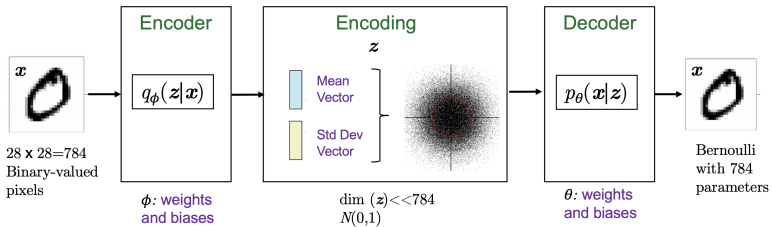
- Automatic feature extraction.
- High accuracy in vision tasks.
- Translation invariance.

Disadvantages of CNNs:

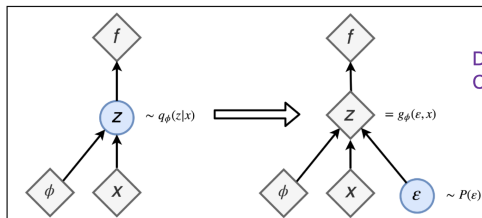
- Resource intensive training (taking days or weeks).
- Large data requirement.
- Overfitting risk – model performs well on trained data, poorly on new data.
- Traditional CNN architectures require input images to be of a fixed size.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Probability Estimation (fig01)



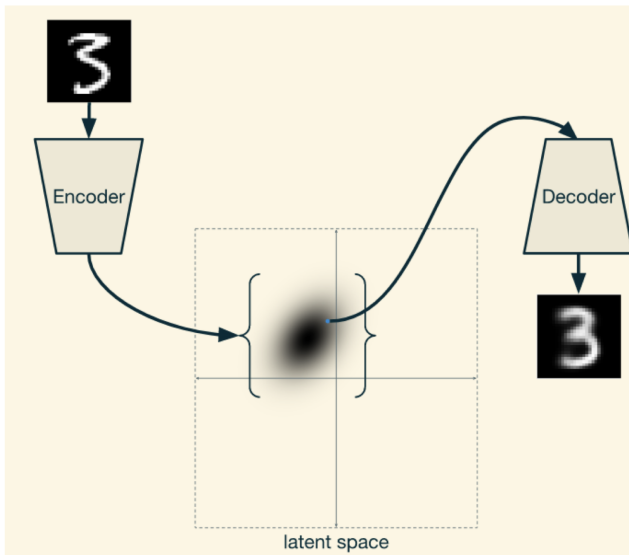
Probability Estimation (fig02)



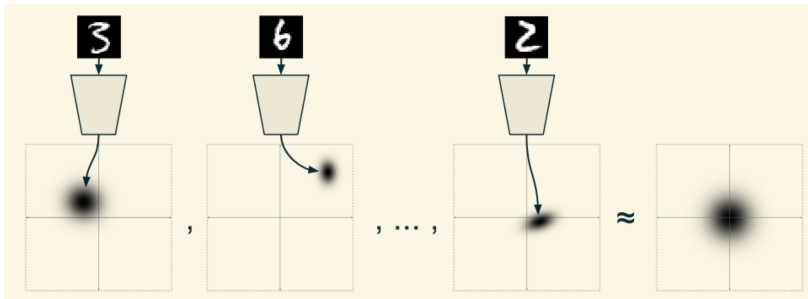
Diamonds: Deterministic dependencies

Circles: Random variables

Probability Estimation (fig03)

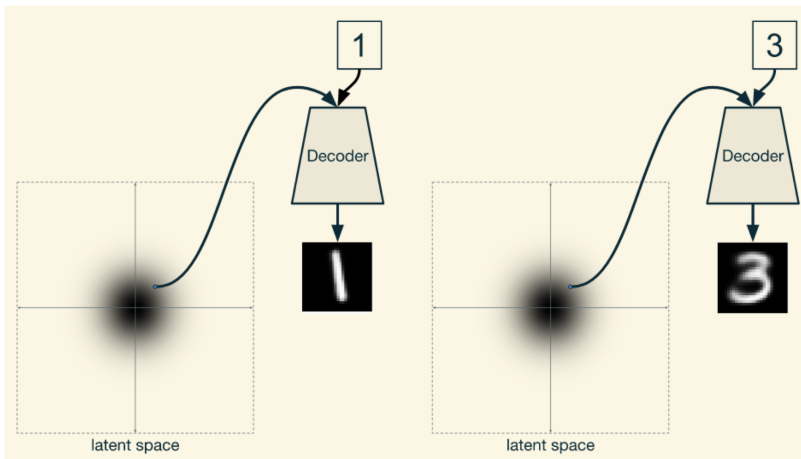


Probability Estimation (fig04)





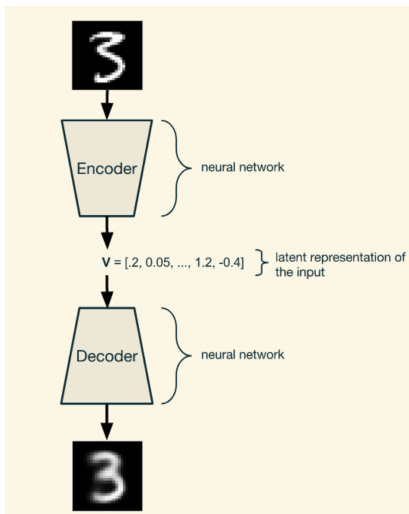
Probability Estimation (fig06)



Probability Estimation (fig07)

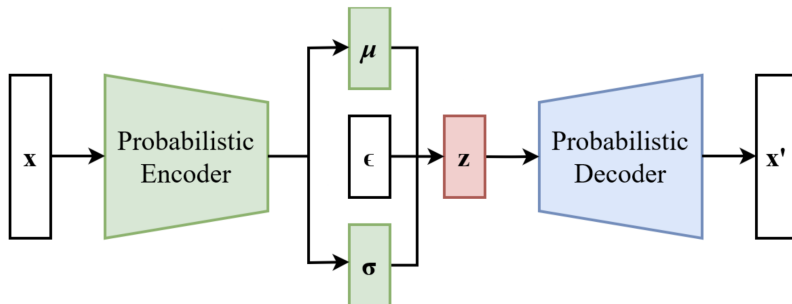


Probability Estimation (fig08)



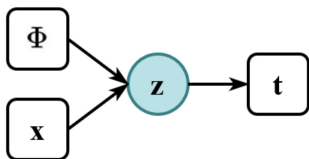
A standard autoencoder trained on MNIST digits may not provide a reasonable output when a ∇ image is input

Probability Estimation (fig09)



Probability Estimation (fig10)

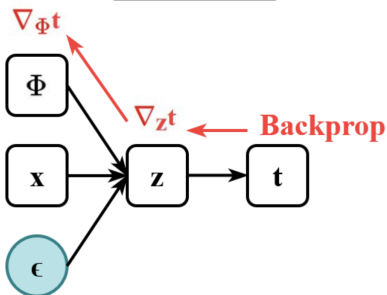
Original Formulation



Deterministic
Node

Random
Node

Reparameterized Formulation



Feedforward

Differentiation

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺

Example 1: Simplest Example

Problem Setup

1. *Journal of Management Studies*, 1990, 27, 1, 1-14.

— () —



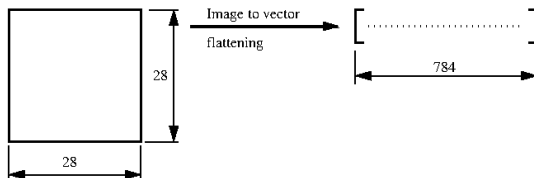
1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 26



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

Solution Procedure:

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



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

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)

Source Code Outline:

```
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:

Assembly 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

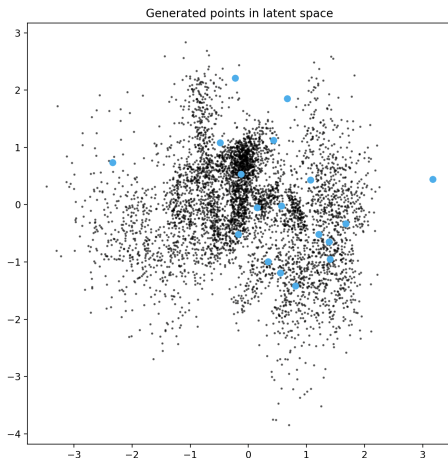
Sampling the Statistical Distribution:

```
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

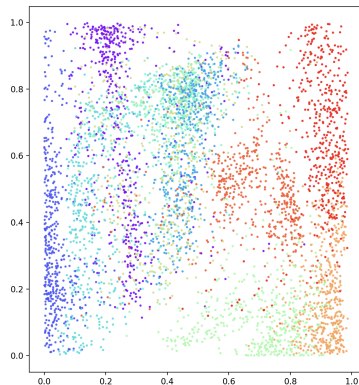
Latent Space: Black dots show the `z_mean` value of each encoded image. Blue dots are sampled points in the latent space.



Latent Space Color Codes:

• • • •

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



Example 2: Generation of Fashion Shapes



[1.1 0.4]

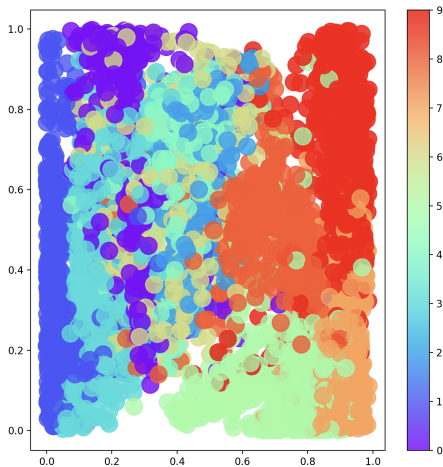


[-0.2 2.2]



[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, 32)	36,928	lrelu_2[0][0]

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

Source Code Outline:

Example 3: Generation of Cartoon Faces

Encoder: 1.77 million parameters.

Decoder: 974,000 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 (BatchNormalization)	(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 (BatchNormalization)	(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 (BatchNormalization)	(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 (BatchNormalization)	(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 (BatchNormalization)	(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: 1,769,592 (6.75 MB)
Non-trainable params: 536 (2.25 KB)

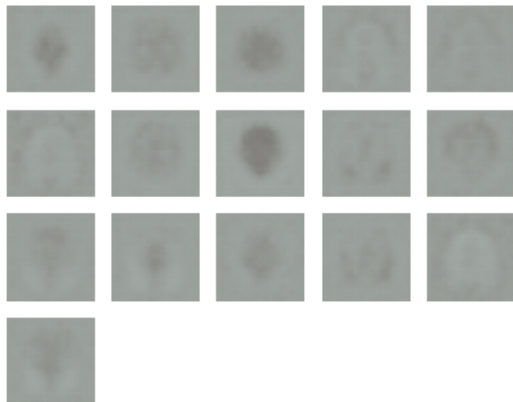
Model: "Decoder"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 200)	0
dense_1 (Dense)	(None, 4096)	823,200
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: 974,387 (3.64 MB)
Trainable params: 952,859 (3.64 MB)
Non-trainable params: 448 (1.75 KB)

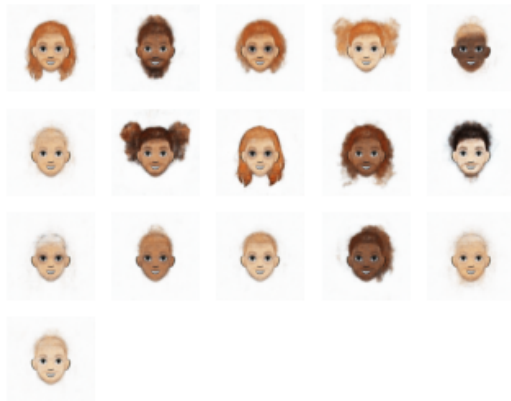
Example 3: Generation of Cartoon Faces

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 dataset⁵

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](https://github.com/eriklindern-co/python-code-ai.d/vae/TestVAE-Faces01.py)

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][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][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][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][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 (988.96 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)
```

Source Code Outline:

Assemble Decoder

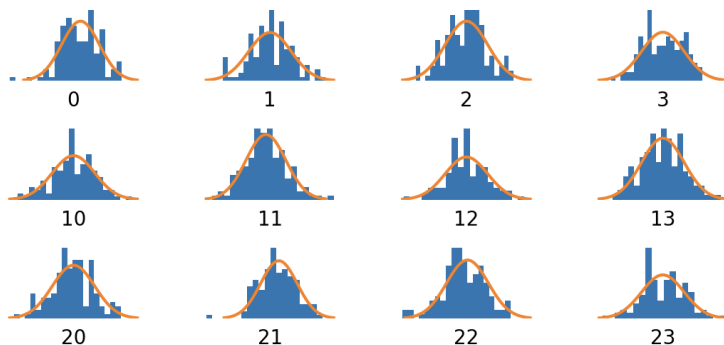
```
decoder_input = layers.Input(shape=( Z_DIM,), name="decoder_input")

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", padding="same")(x)
decoder = models.Model(decoder_input, decoder_output)
```


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





◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺

Generate New Faces:

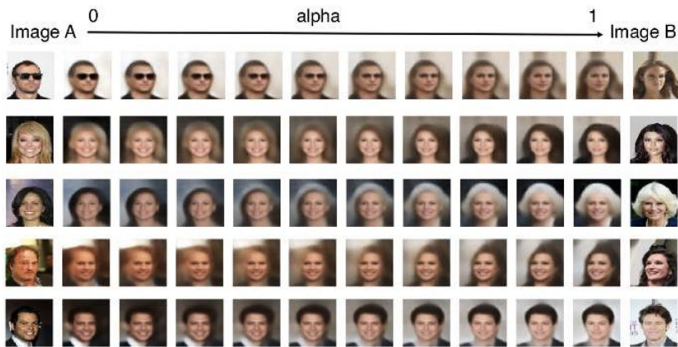


Figure 3-22. Morphing between two faces

Example 4: Generation of Faces (faces03)

Generate New Faces:

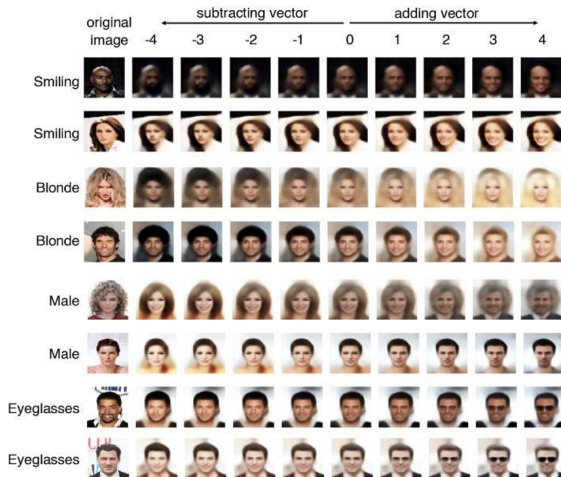


Figure 3-21. Adding and subtracting features to and from faces

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.