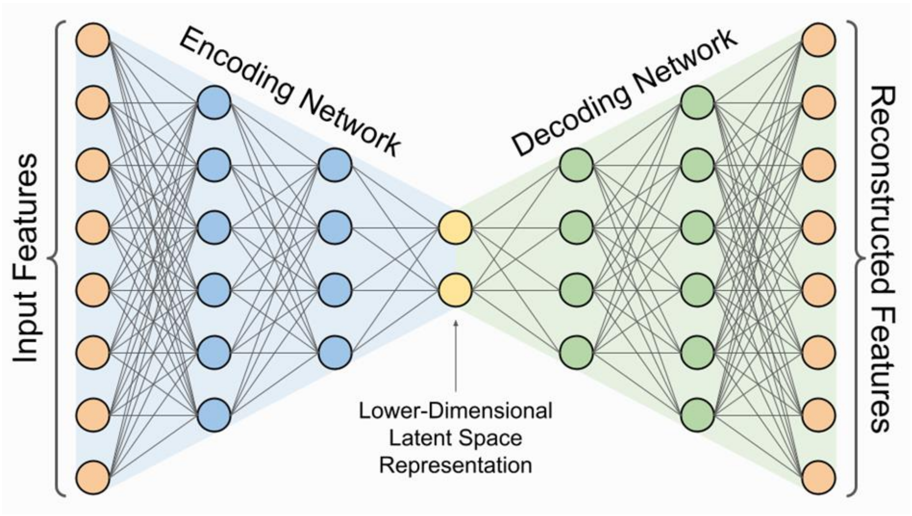
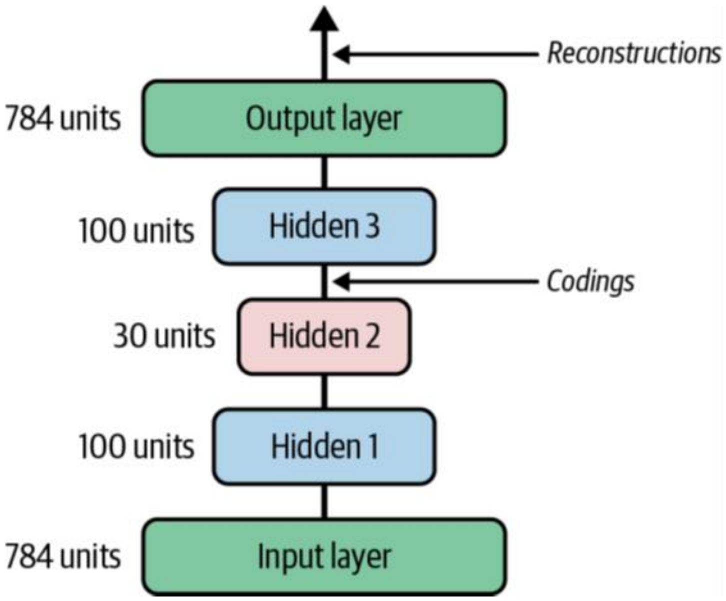
8. Învățarea reprezentărilor și învățarea generativă

8.1 Autoencodere

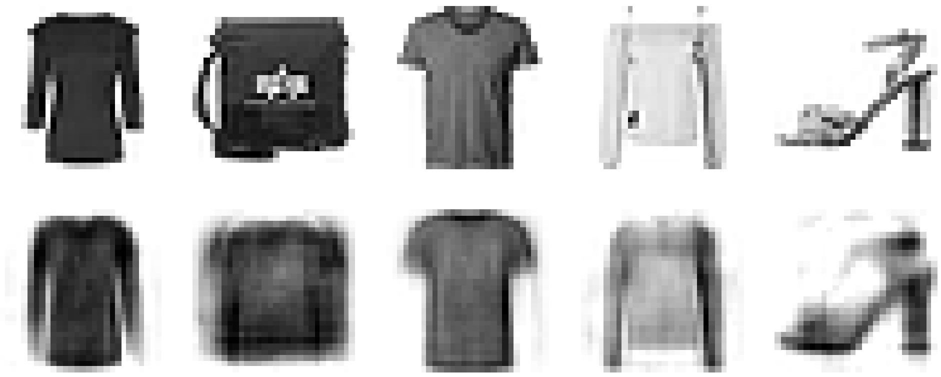


Exemplu de autoencoder pentru setul de date Fashion-MNIST



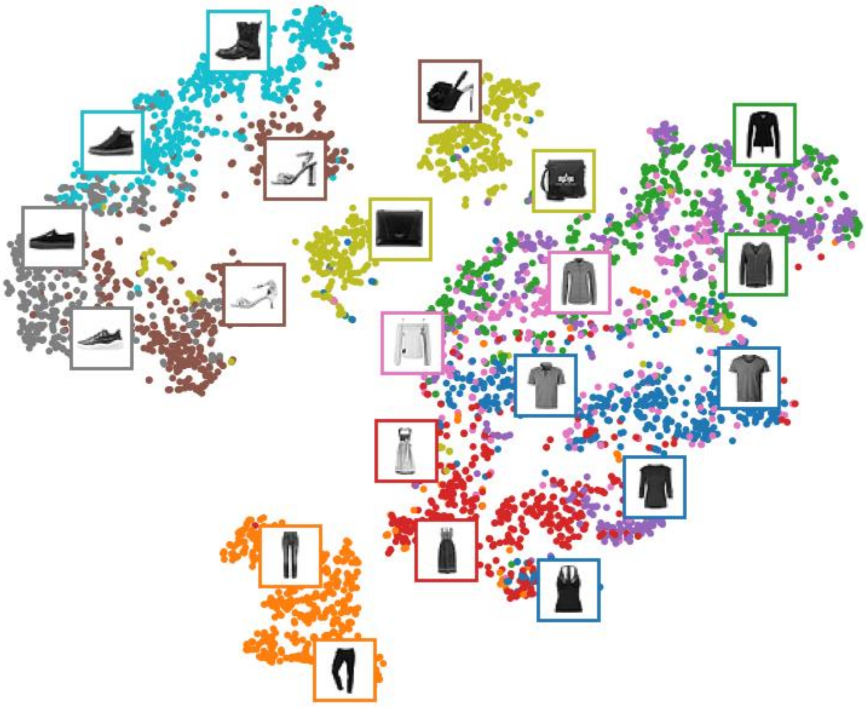
```
stacked_encoder = tf.keras.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(30, activation="relu"),
])
stacked_decoder = tf.keras.Sequential([
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(28 * 28),
    tf.keras.layers.Reshape([28, 28])
])
stacked_ae = tf.keras.Sequential([stacked_encoder, stacked_decoder])

stacked_ae.compile(loss="mse", optimizer=tf.keras.optimizers.Nadam())
history = stacked_ae.fit(X_train, X_train, epochs=20,
                         validation_data=(X_valid, X_valid))
```

Visualizing the Reconstructions

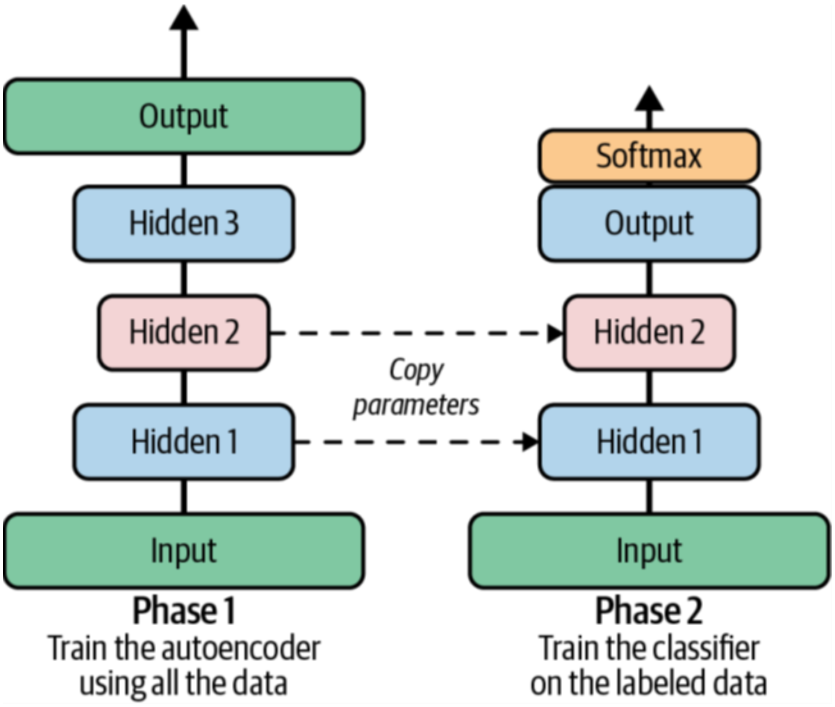


Vizualizarea setului de date Fashion-MNIST

```
from sklearn.manifold import TSNE

X_valid_compressed = stacked_encoder.predict(X_valid)
tsne = TSNE(init="pca", random_state=42)
X_valid_2D = tsne.fit_transform(X_valid_compressed)

plt.scatter(X_valid_2D[:, 0], X_valid_2D[:, 1], c=y_valid, s=10, cmap="tab10")
plt.show()
```



Preantrenarea nesupervizată utilizând autoencodere



Upsampling layer

```
import numpy as np
from keras.models import Sequential
from keras.layers import UpSampling2D


X = np.asarray([[1, 2],
          [3, 4]])

print(X)
```

```
[[1 2]
 [3 4]]
```

```
X = X.reshape((1, 2, 2, 1))

model = Sequential()
model.add(UpSampling2D(input_shape=(2, 2, 1)))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
up_sampling2d_1 (UpSampling2 (None, 4, 4, 1)           0         
=================================================================
Total params: 0
Trainable params: 0
Non-trainable params: 0
_________________________________________________________________
```

```
yhat = model.predict(X)

yhat = yhat.reshape((4, 4))

print(yhat)
```

```
[[1. 1. 2. 2.]
 [1. 1. 2. 2.]
 [3. 3. 4. 4.]
 [3. 3. 4. 4.]]
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Reshape
from keras.layers import UpSampling2D
from keras.layers import Conv2D


model = Sequential()

model.add(Dense(128 * 5 * 5, input_dim=100))

model.add(Reshape((5, 5, 128)))

model.add(UpSampling2D())

model.add(Conv2D(1, (3,3), padding='same'))

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 3200)              323200    
_________________________________________________________________
reshape_1 (Reshape)          (None, 5, 5, 128)         0         
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 10, 10, 128)       0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 1)         1153      
=================================================================
Total params: 324,353
Trainable params: 324,353
Non-trainable params: 0
_________________________________________________________________
```

Transpose convolutional layer

```
import numpy as np
from keras.models import Sequential
from keras.layers import Conv2DTranspose


X = np.asarray([[1, 2],
          [3, 4]])
```

```
print(X)
```

```
[[1 2]
 [3 4]]
```

```
X = X.reshape((1, 2, 2, 1))

model = Sequential()
model.add(Conv2DTranspose(1, (1,1), strides=(2,2),
                          input_shape=(2, 2, 1)))

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_transpose_1 (Conv2DTr (None, 4, 4, 1)           2         
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

```
weights = [np.asarray([[[[1]]]]), np.asarray([0])]

model.set_weights(weights)

yhat = model.predict(X)

yhat = yhat.reshape((4, 4))

print(yhat)
```

```
[[1. 0. 2. 0.]
 [0. 0. 0. 0.]
 [3. 0. 4. 0.]
 [0. 0. 0. 0.]]
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Reshape
from keras.layers import Conv2DTranspose
```

```
model = Sequential()

model.add(Dense(128 * 5 * 5, input_dim=100))

model.add(Reshape((5, 5, 128)))

model.add(Conv2DTranspose(1, (3,3), strides=(2,2),
                          padding='same'))

model.summary()
```

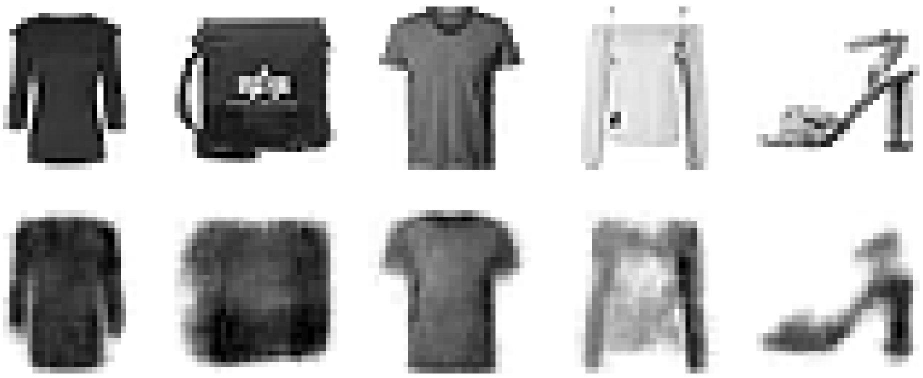
```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 3200)              323200

_________________________________________________________________
reshape_2 (Reshape)          (None, 5, 5, 128)         0

_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 10, 10, 1)         1153
=================================================================
Total params: 324,353
Trainable params: 324,353
Non-trainable params: 0

_________________________________________________________________
```

Autoencodere convoluționale

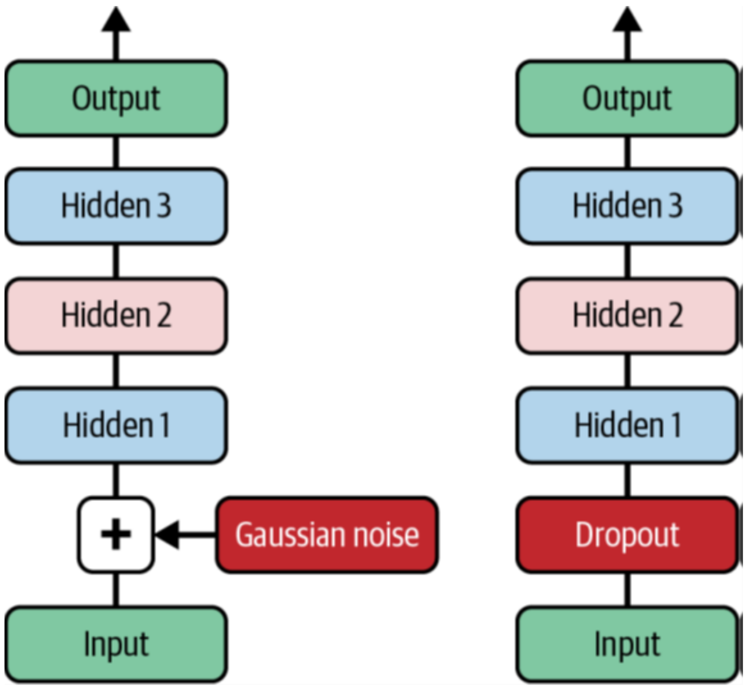
Exemplu de autoencoder convoluțional pentru setul de date Fashion-MNIST:

```
conv_encoder = tf.keras.Sequential([
    tf.keras.layers.Reshape([28, 28, 1]),
    tf.keras.layers.Conv2D(16, 3, padding="same", activation="relu"),
    tf.keras.layers.MaxPool2D(pool_size=2),  # output: 14 × 14 x 16
    tf.keras.layers.Conv2D(32, 3, padding="same", activation="relu"),
    tf.keras.layers.MaxPool2D(pool_size=2),  # output: 7 × 7 x 32
    tf.keras.layers.Conv2D(64, 3, padding="same", activation="relu"),
    tf.keras.layers.MaxPool2D(pool_size=2),  # output: 3 × 3 x 64
    tf.keras.layers.Conv2D(30, 3, padding="same", activation="relu"),
    tf.keras.layers.GlobalAvgPool2D()  # output: 30
])
conv_decoder = tf.keras.Sequential([
    tf.keras.layers.Dense(3 * 3 * 16),
    tf.keras.layers.Reshape((3, 3, 16)),
    tf.keras.layers.Conv2DTranspose(32, 3, strides=2, activation="relu"),
    tf.keras.layers.Conv2DTranspose(16, 3, strides=2, padding="same",
                                    activation="relu"),
    tf.keras.layers.Conv2DTranspose(1, 3, strides=2, padding="same"),
    tf.keras.layers.Reshape([28, 28])
])
conv_ae = tf.keras.Sequential([conv_encoder, conv_decoder])

conv_ae.compile(loss="mse", optimizer=tf.keras.optimizers.Nadam())
history = conv_ae.fit(X_train, X_train, epochs=10,
                      validation_data=(X_valid, X_valid))
```

Visualizing the Reconstructions



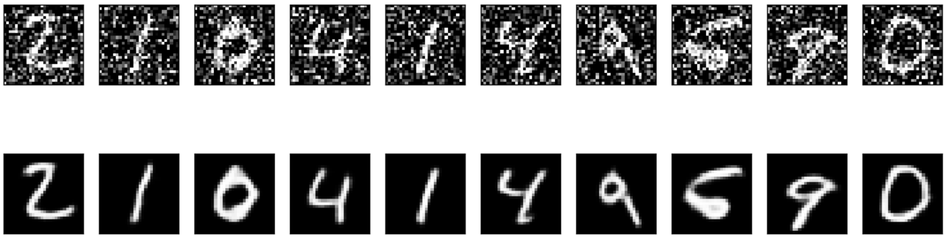
Autoencodere de eliminare a zgomotului



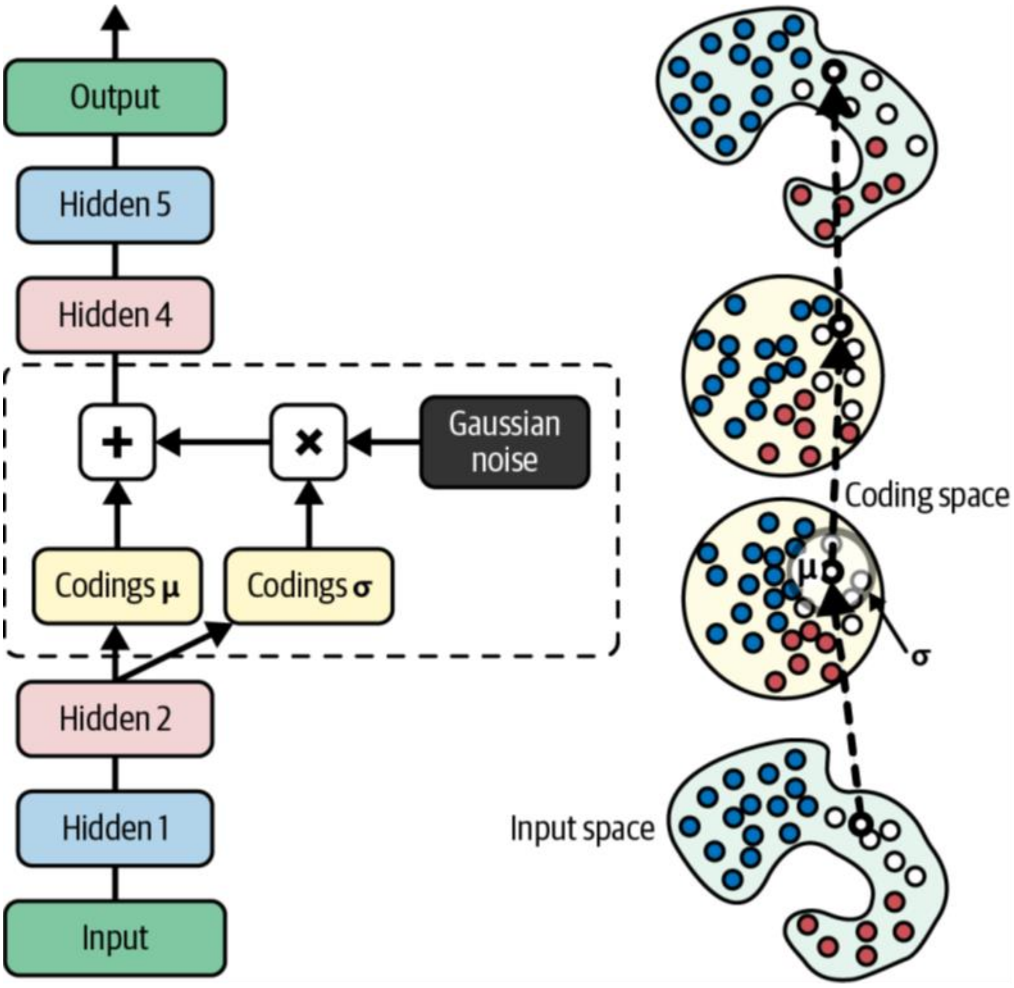
Aplicație: reducerea zgomotului imaginilor

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras import models, layers

(X_train, _), (X_test, _) = mnist.load_data()

X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255

X_train = np.expand_dims(X_train, axis=-1)
X_test = np.expand_dims(X_test, axis=-1)

noise_factor = 0.5
X_train_noisy = X_train + noise_factor * np.random.normal(
    loc=0.0, scale=1.0, size=X_train.shape)
X_test_noisy = X_test + noise_factor * np.random.normal(
    loc=0.0, scale=1.0, size=X_test.shape)

X_train_noisy = np.clip(X_train_noisy, 0, 1)
X_test_noisy = np.clip(X_test_noisy, 0, 1)

encoder = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3),
                  padding='same', activation='relu',
                  input_shape=X_train.shape[1:]),
    layers.MaxPooling2D(pool_size=(2, 2), padding='same'),
    layers.Conv2D(filters=32, kernel_size=(3, 3),
                  padding='same', activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2), padding='same')
    ])

decoder = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3),
                  padding='same', activation='relu',
                  input_shape=encoder.layers[-1].output_shape[1:]),
    layers.UpSampling2D(size=(2, 2)),
    layers.Conv2D(filters=32, kernel_size=(3, 3),
                  padding='same', activation='relu'),
    layers.UpSampling2D(size=(2, 2)),
    layers.Conv2D(filters=1, kernel_size=(3, 3),
                  padding='same', activation='sigmoid')
    ])

autoencoder = models.Sequential([encoder, decoder])
```

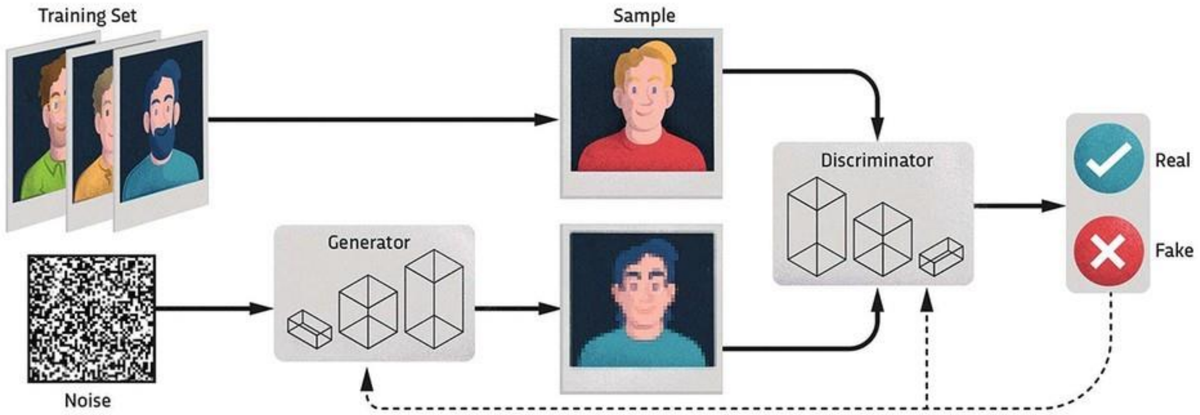
```
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

history = autoencoder.fit(X_train_noisy, X_train,
                epochs=100,
                batch_size=128,
                shuffle=True,
                validation_data=(X_test_noisy, X_test))

decoded_imgs = autoencoder.predict(X_test_noisy)


n = 10
plt.figure(figsize=(20, 4))
for i in range(1, n + 1):
    # Display original
    ax = plt.subplot(2, n, i)
    plt.imshow(X_test_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



Autoencodere variaționale



8.2 Rețele generative adversative



Exemplu de DCGAN pentru setul de date Fashion-MNIST:

```
codings_size = 100

generator = tf.keras.Sequential([
    tf.keras.layers.Dense(7 * 7 * 128),
    tf.keras.layers.Reshape([7, 7, 128]),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2DTranspose(64, kernel_size=5, strides=2,
                                    padding="same", activation="relu"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2DTranspose(1, kernel_size=5, strides=2,
                                    padding="same", activation="tanh"),
])
discriminator = tf.keras.Sequential([
    tf.keras.layers.Conv2D(64, kernel_size=5, strides=2, padding="same",
                        activation=tf.keras.layers.LeakyReLU(0.2)),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Conv2D(128, kernel_size=5, strides=2, padding="same",
                        activation=tf.keras.layers.LeakyReLU(0.2)),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(1, activation="sigmoid")
])
gan = tf.keras.Sequential([generator, discriminator])

discriminator.compile(loss="binary_crossentropy", optimizer="rmsprop")
discriminator.trainable = False
gan.compile(loss="binary_crossentropy", optimizer="rmsprop")
```