Introductory Seminar of PyTorch for Deep Learning

Daniele Angioni, Cagliari Digital Lab 2024 - Day 5

From PyTorch to Tensorflow

From PyTorch to Tensforlow

During the seminar we learned the fundamental concepts of Deep Learning to solve different tasks, from computer vision to natural language processing.

In the following we will see first how to apply the fundamentals to solve a different task, and we will also see how to do it with another deep learning framework: Tensorflow

Tensorflow

Tensorflow is a deep learning framework that, similarly to PyTorch, make use of automatic gradient differentiation.

In Tensorflow we have an high-level API called **Keras**, that makes training and evaluating extremely easy for a user (at least for standard learning problems).

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Keras pipeline

- Load the dataset

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Keras pipeline

- Load the dataset
- Normalization

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Keras pipeline

- Load the dataset
- Normalization
- Model definition

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Keras pipeline

- Load the dataset
- Normalization
- Model definition
- Setting the optimizer and loss function

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Keras pipeline

- Load the dataset
- Normalization
- Model definition
- Setting the optimizer and loss function
- Training

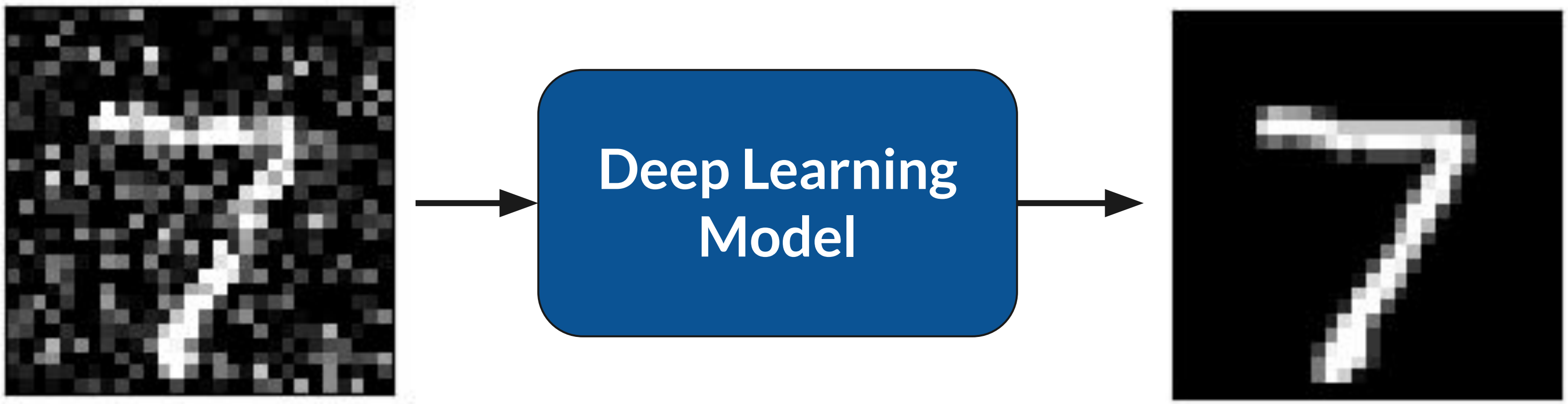
```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Keras pipeline

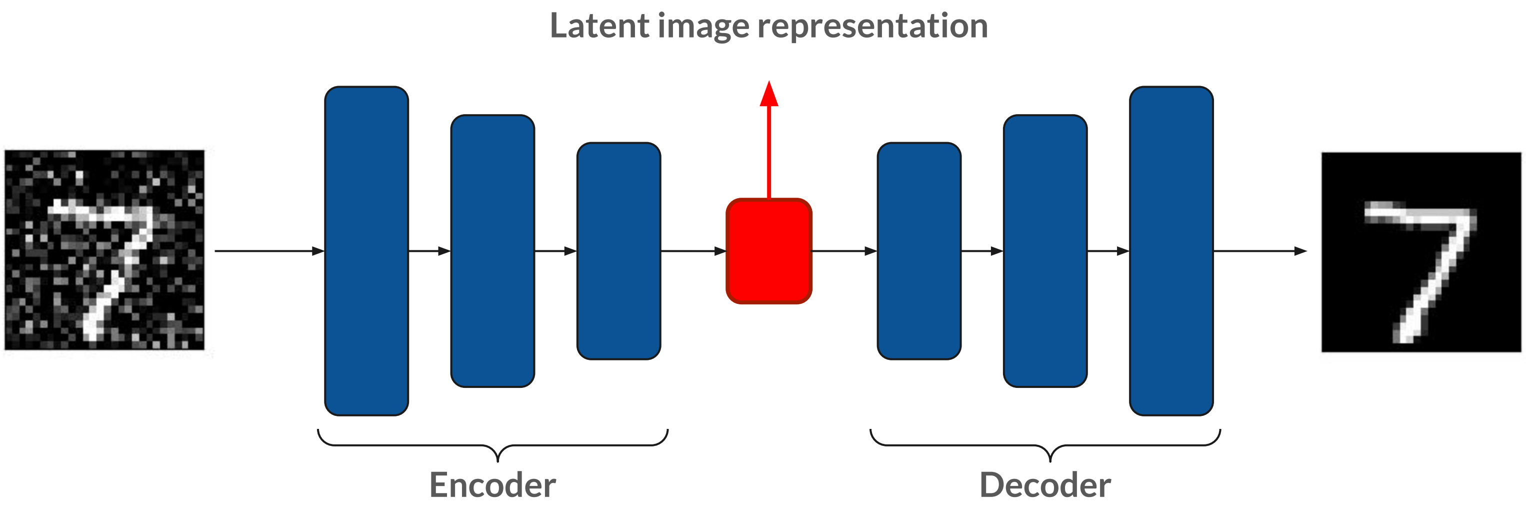
- Load the dataset
- Normalization
- Model definition
- Setting the optimizer and loss function
- Training
- Evaluating

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

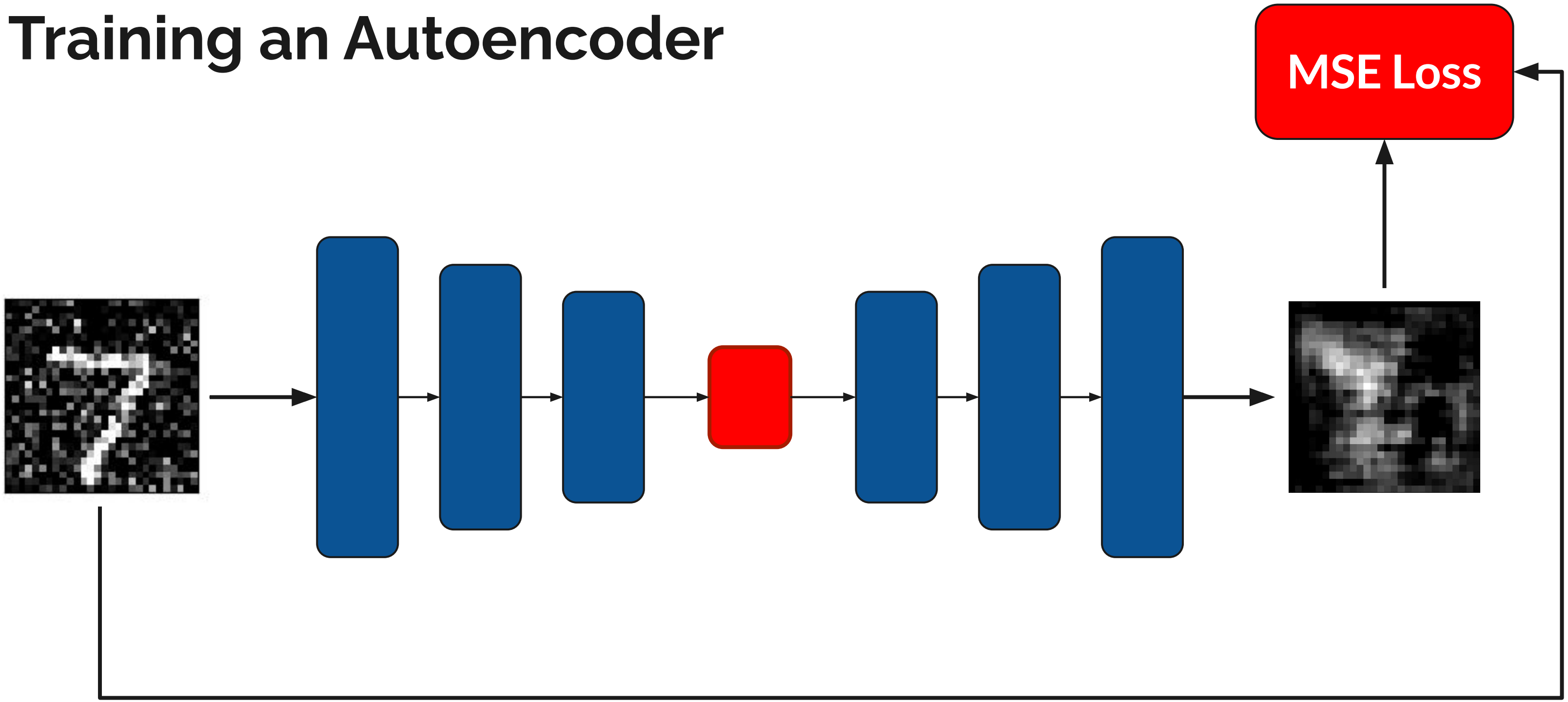
Example: Image Denoising



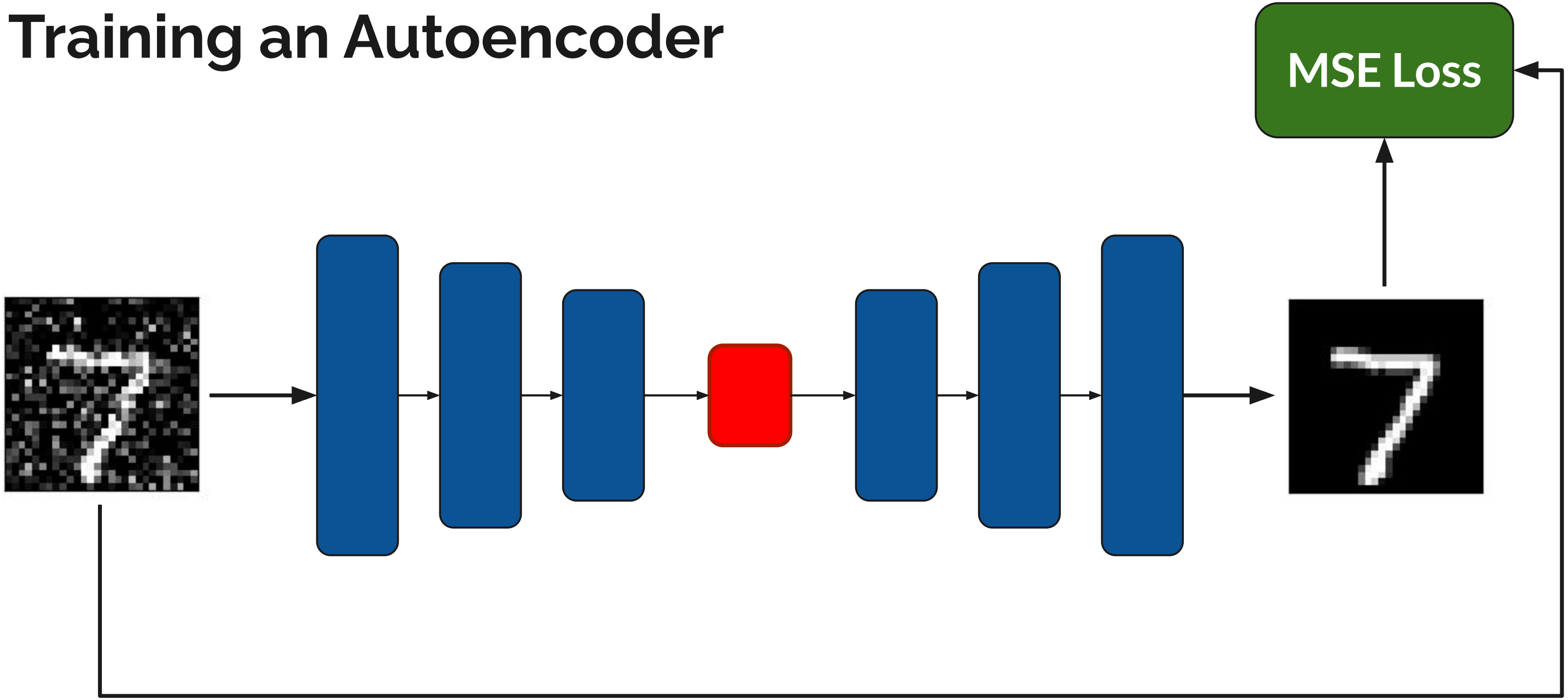
Solution: using an Autoencoder



Training an Autoencoder



Training an Autoencoder



Loading the MNIST dataset

```python
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np

(x_train, _), (x_test, _) = tf.keras.datasets.mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

print(x_train.shape)
print(x_test.shape)
print(type(x_train))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
(60000, 28, 28, 1)
(10000, 28, 28, 1)
<class 'numpy.ndarray'>
```

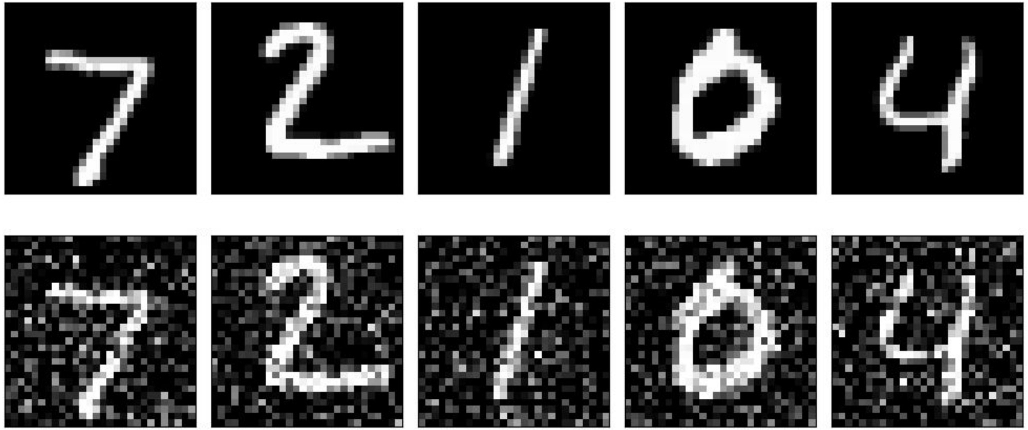
Adding the noise

```
noise_factor = 0.3
#loc and scale being mean and std
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)

#clip pixel under 0 and above 1
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

Visualizing some images

```
n_images = 5        #images to be visualized
bias = 0    #starting index from the test set for the visualization
fig, axs = plt.subplots(nrows=2, ncols=n_images, figsize=(10, 5))
for i in range(n_images):
    axs[0, i].imshow(x_test[bias + i], cmap='gray')
    axs[0, i].get_xaxis().set_visible(False)
    axs[0, i].get_yaxis().set_visible(False)

    axs[1, i].imshow(x_test_noisy[bias + i], cmap='gray')
    axs[1, i].get_xaxis().set_visible(False)
    axs[1, i].get_yaxis().set_visible(False)

fig.tight_layout()
fig.show()
```

Visualizing some images



Model architecture

```python
from keras.layers import Input, Conv2D, AveragePooling2D, UpSampling2D
from keras.models import Model
def create_autoencoder():
    encoder = tf.keras.models.Sequential([
        Conv2D(32, (5, 5), activation='relu', padding='same'),
        Conv2D(32, (5, 5), activation='relu', padding='same'),
        AveragePooling2D((2, 2), padding='same'), #14x14
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        AveragePooling2D((2, 2), padding='same'),   #7x7
        Conv2D(128, (3, 3), activation='relu', padding='same')
    ])
    # latent representation has shape (7, 7, 128)
    decoder = tf.keras.models.Sequential([
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        UpSampling2D((2, 2)), #14x14
        Conv2D(64, (5, 5), activation='relu', padding='same'),
        Conv2D(64, (5, 5), activation='relu', padding='same'),
        UpSampling2D((2, 2)), #28x28
        Conv2D(32, (5, 5), activation='relu', padding='same'),
        Conv2D(32, (5, 5), activation='relu', padding='same'),
        Conv2D(1, (3, 3), activation='relu', padding='same')
    ])
    autoencoder = tf.keras.models.Sequential([encoder, decoder])
    return autoencoder
```

Training and saving the model weights

```
from keras.optimizers import Adam, SGD
from keras.losses import MeanSquaredError

epochs = 10
autoencoder = create_autoencoder()  #create the architecture
optim = SGD(learning_rate=0.001, momentum=0.9)
loss = MeanSquaredError()
autoencoder.compile(optimizer=optim, loss=loss)

autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=128,
                shuffle=True,
                verbose=2,
                validation_data=(x_test_noisy, x_test))
autoencoder.save_weights("autoencoder.h5")
```

Evaluating

```
autoencoder = create_autoencoder()  # instantiate the autoencoder model
out = autoencoder.predict(x_test[:1])   # build the graph
autoencoder.load_weights("autoencoder.h5")  # load the pretrained weights

n_images = 4 #images to be visualized
bias = 15 #starting index from the test set for the visualization
plt.figure(figsize=(40, 20))

input_images = x_test_noisy[bias : bias + n_images]
target_images = x_test[bias : bias + n_images]
output_imgs = autoencoder.predict(x_test_noisy[bias : bias + n_images])
output_imgs = np.clip(output_imgs, 0., 1.)

nrows = 3
figdim = 2
fig, axs = plt.subplots(nrows=nrows, ncols=n_images, figsize=(figdim*n_images, figdim*nrows))

```

Evaluating

```python
for i in range(n_images):
    axs[0, i].imshow(target_images[i], cmap='gray')
    axs[0, i].set_yticklabels([])
    axs[0, i].set_xticklabels([])

    axs[1, i].imshow(input_images[i], cmap='gray')
    axs[1, i].set_yticklabels([])
    axs[1, i].set_xticklabels([])

    axs[2, i].imshow(output_imgs[i], cmap='gray')
    axs[2, i].set_yticklabels([])
    axs[2, i].set_xticklabels([])

axs[0, 0].set_ylabel('Image')
axs[1, 0].set_ylabel('Noisy Image')
axs[2, 0].set_ylabel('Reconstructed')

fig.tight_layout()
fig.show()
```

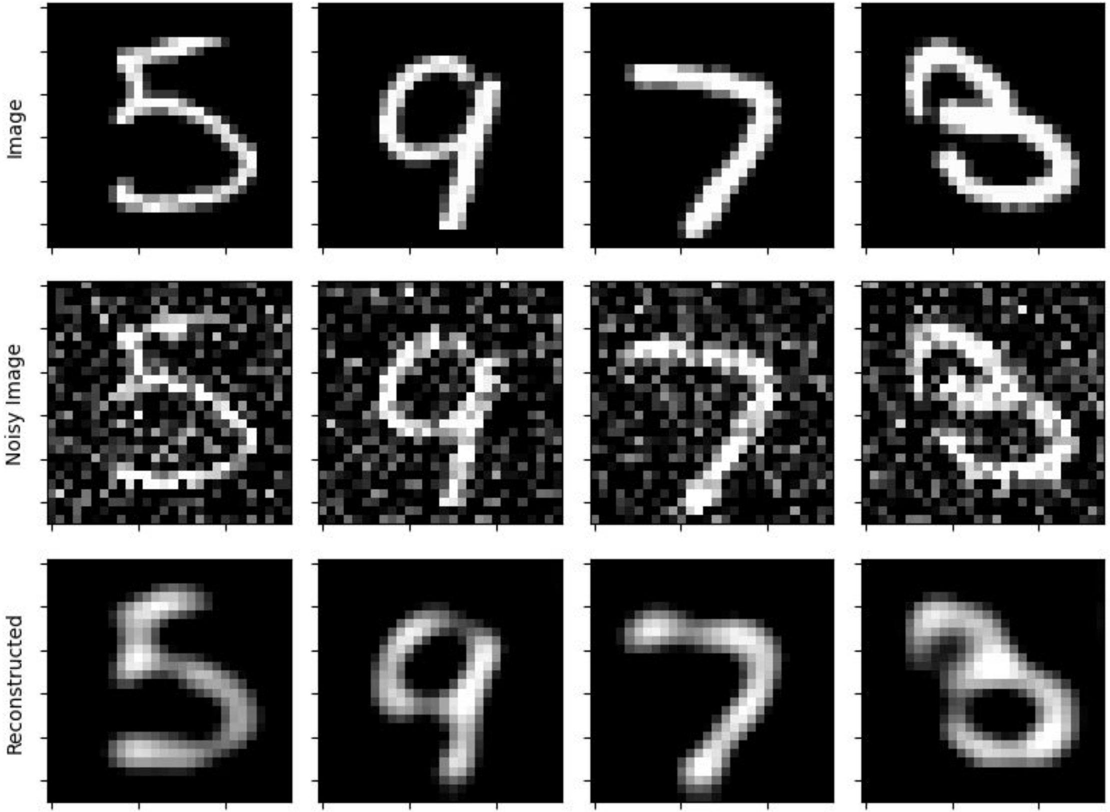


Image
Noisy Image
Reconstructed