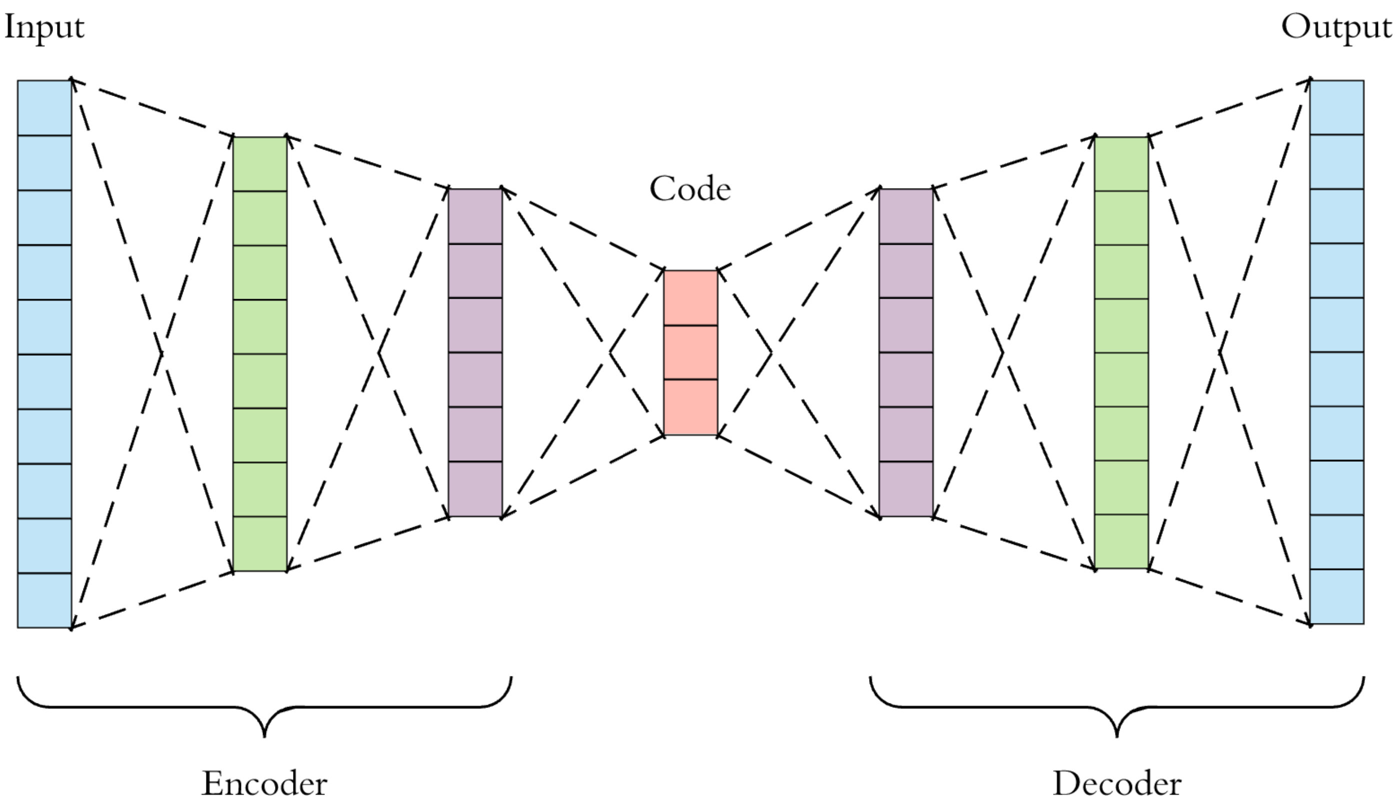
Autoencoders



```
from keras import layers
from keras.models import Model
import numpy as np
from keras.datasets import mnist
```

```
(x_train, _), (x_test, _) = mnist.load_data(path='mnist.npz')
# normalize all values between 0-1 and we will flatten the images into vectors
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
print(x_train.shape)
print(x_test.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist
11493376/11490434 [==============================] - 0s 0us/step
(60000, 784)
(10000, 784)
```

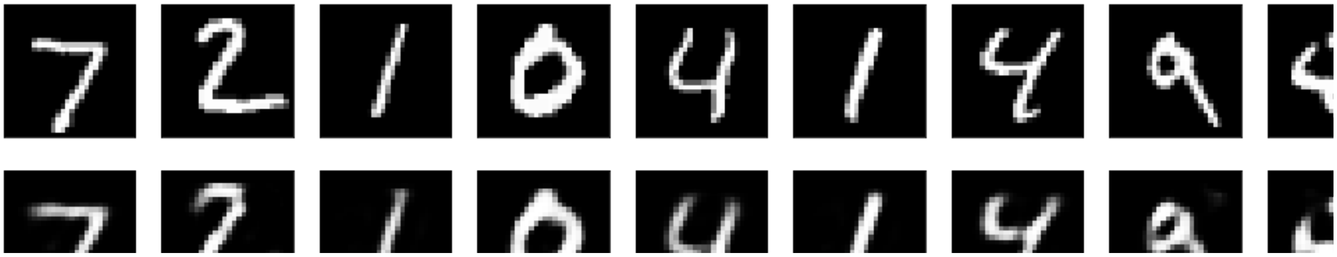
```
encoding_dim = 32
```

```
# this is our input placeholder
input_img = layers.Input(shape=(784,))

# "encoded" is the encoded representation of the input
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = layers.Dense(784, activation='sigmoid')(encoded)

# this model maps an input to its reconstruction
autoencoder_Model = Model(input_img, decoded)

# this model maps an input to its encoded representation
encoder_Model = Model(input_img, encoded)

# create a placeholder for an encoded (32-dimensional) input
encoded_input = layers.Input(shape=(encoding_dim,))
# retrieve the last layer of the autoencoder model
decoder_layer = autoencoder_Model.layers[-1]

# create the decoder model
decoder_Model = Model(encoded_input, decoder_layer(encoded_input))

autoencoder_Model.compile(optimizer='adam', loss='binary_crossentropy')


n_epochs = 20
autoencoder_Model.fit(x_train, x_train,
                epochs=n_epochs,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 1/20
235/235 [==============================] - 3s 5ms/step - loss: 0.3853 - val_loss: 0.1925
Epoch 2/20
235/235 [==============================] - 1s 3ms/step - loss: 0.1820 - val_loss: 0.1541
Epoch 3/20
235/235 [==============================] - 1s 3ms/step - loss: 0.1497 - val_loss: 0.1332
Epoch 4/20
235/235 [==============================] - 1s 3ms/step - loss: 0.1312 - val_loss: 0.1202
Epoch 5/20
235/235 [==============================] - 1s 3ms/step - loss: 0.1191 - val_loss: 0.1117
Epoch 6/20
235/235 [==============================] - 1s 3ms/step - loss: 0.1114 - val_loss: 0.1059
Epoch 7/20
235/235 [==============================] - 1s 4ms/step - loss: 0.1061 - val_loss: 0.1016
Epoch 8/20
235/235 [==============================] - 1s 3ms/step - loss: 0.1019 - val_loss: 0.0986
Epoch 9/20
235/235 [==============================] - 1s 3ms/step - loss: 0.0992 - val_loss: 0.0966
Epoch 10/20
```

```
235/235 [==============================] - 1s 3ms/step - loss: 0.0976 - val_loss: 0.0952
Epoch 11/20
235/235 [==============================] - 1s 3ms/step - loss: 0.0963 - val_loss: 0.0943
Epoch 12/20
235/235 [==============================] - 1s 3ms/step - loss: 0.0955 - val_loss: 0.0936
Epoch 13/20
235/235 [==============================] - 1s 3ms/step - loss: 0.0948 - val_loss: 0.0932
Epoch 14/20
235/235 [==============================] - 1s 3ms/step - loss: 0.0944 - val_loss: 0.0929
Epoch 15/20
235/235 [==============================] - 1s 3ms/step - loss: 0.0941 - val_loss: 0.0927
Epoch 16/20
235/235 [==============================] - 1s 3ms/step - loss: 0.0939 - val_loss: 0.0925
Epoch 17/20
235/235 [==============================] - 1s 3ms/step - loss: 0.0936 - val_loss: 0.0925
Epoch 18/20
235/235 [==============================] - 1s 3ms/step - loss: 0.0937 - val_loss: 0.0922
Epoch 19/20
235/235 [==============================] - 1s 3ms/step - loss: 0.0934 - val_loss: 0.0921
Epoch 20/20
235/235 [==============================] - 1s 3ms/step - loss: 0.0933 - val_loss: 0.0921
<tensorflow.python.keras.callbacks.History at 0x7f050d718ac8>
```

```
encoded_imgs = encoder_Model.predict(x_test)
decoded_imgs = decoder_Model.predict(encoded_imgs)

import matplotlib.pyplot as plt

n = 10  # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[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 + 1 + 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()
```

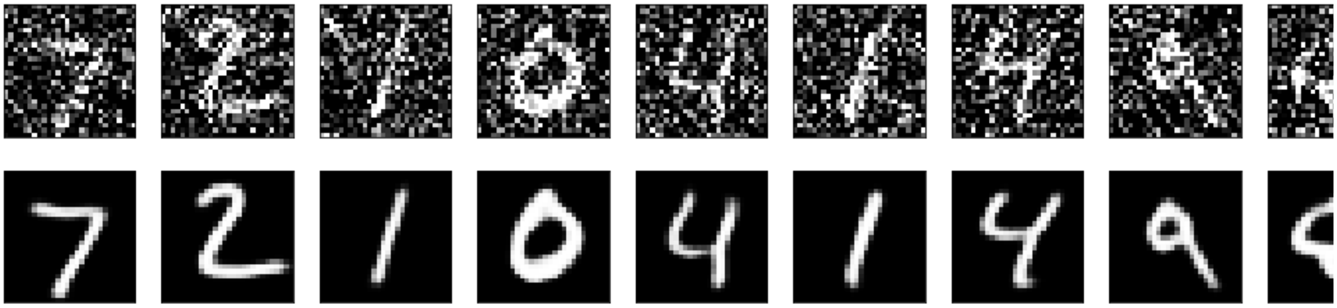


some feature in keras

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

epochs = 10
batch_size = 128


#==============================================================================
# Prepare data

(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))  # adapt this if using `channels_fir
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))  # adapt this if using `channels_first`

noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.sh
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.)

#==============================================================================
# Create Layers + Model

input_img = layers.Input(shape=(28, 28, 1))

x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (7, 7, 32)

x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
```

```
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)

# Plot model
from keras.utils import plot_model
plot_model(autoencoder, to_file='AE_model.pdf', show_shapes=True)

autoencoder.compile(optimizer=optimizers.Adam(), loss=losses.binary_crossentropy)

#===============================================================================
# Callbacks
from keras import callbacks

model_checkpoint = callbacks.ModelCheckpoint('/model.{epoch}.h5')

logger = callbacks.CSVLogger('training.log')

tensorboard = callbacks.TensorBoard(log_dir='./tensorboard')

callbacks = [model_checkpoint, logger, tensorboard]
#===============================================================================
# Train the Model
import datetime

start = datetime.datetime.now()
autoencoder.fit(x_train_noisy, x_train,
                epochs=epochs,
                batch_size=batch_size,
                shuffle=True,
                validation_data=(x_test_noisy, x_test),
                callbacks=callbacks)
end = datetime.datetime.now()
elapsed = end-start
print('Total training time: ', str(elapsed))

autoencoder.save('model.h5')
autoencoder.save_weights('model_weights.h5')
#===============================================================================
# Predict + Visualization

decoded_imgs = autoencoder.predict(x_test)

n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i+1)
    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 + 1 + 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()
```

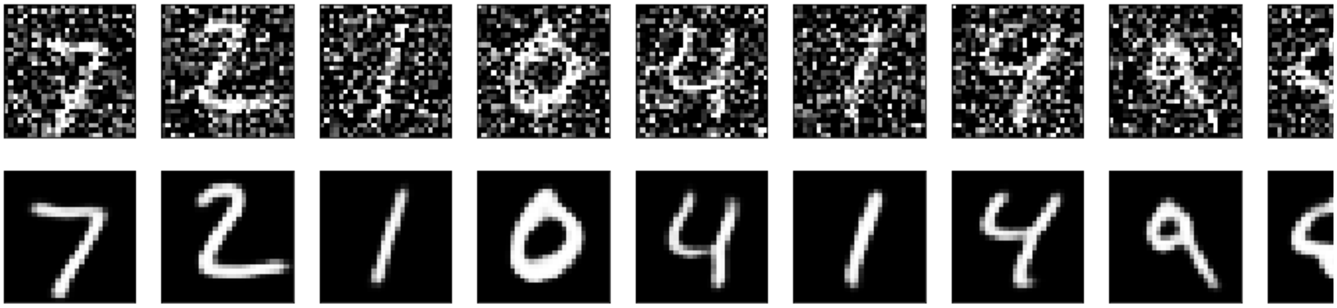
```
Epoch 1/10
469/469 [==============================] - 9s 7ms/step - loss: 0.2621 - val_loss: 0.1169
Epoch 2/10
469/469 [==============================] - 3s 6ms/step - loss: 0.1153 - val_loss: 0.1083
Epoch 3/10
469/469 [==============================] - 3s 6ms/step - loss: 0.1090 - val_loss: 0.1052
Epoch 4/10
469/469 [==============================] - 3s 7ms/step - loss: 0.1054 - val_loss: 0.1030
Epoch 5/10
469/469 [==============================] - 3s 7ms/step - loss: 0.1036 - val_loss: 0.1017
Epoch 6/10
469/469 [==============================] - 3s 7ms/step - loss: 0.1023 - val_loss: 0.1004
Epoch 7/10
469/469 [==============================] - 3s 7ms/step - loss: 0.1012 - val_loss: 0.0995
Epoch 8/10
469/469 [==============================] - 3s 7ms/step - loss: 0.1002 - val_loss: 0.0993
Epoch 9/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0995 - val_loss: 0.0984
Epoch 10/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0989 - val_loss: 0.0978
Total training time:  0:00:36.943574
```



load model

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

epochs = 10
batch_size = 128
#================================================================================
# Prepare data

(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))  # adapt this if using `channels_fir
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))  # adapt this if using `channels_first`

noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.sh
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.)


# Load entire model
from keras.models import load_model

autoencoder_loaded = load_model('model.h5')
autoencoder_loaded.compile(optimizer=optimizers.Adam(), loss=losses.binary_crossentropy)
autoencoder_loaded.summary()

#================================================================================
# Predict + Visualization

decoded_imgs = autoencoder_loaded.predict(x_test)

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

```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (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_2 (Conv2D)            (None, 7, 7, 32)          9248
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 32)        9248
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 28, 28, 1)         289
=================================================================
Total params: 28,353
Trainable params: 28,353
Non-trainable params: 0
_________________________________________________________________
```



load weights

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

```
epochs = 10
batch_size = 128
#===============================================================================
# Prepare data

(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))  # adapt this if using `channels_fir
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))  # adapt this if using `channels_first`

noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.sh
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.)


# Create Layers + Model

input_img = layers.Input(shape=(28, 28, 1))

x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (7, 7, 32)

x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder_loaded = Model(input_img, decoded)

# Load model weights
autoencoder_loaded.load_weights('model_weights.h5')
autoencoder_loaded.compile(optimizer=optimizers.Adam(), loss=losses.binary_crossentropy)
autoencoder_loaded.summary()

#===============================================================================
# Predict + Visualization

decoded_imgs = autoencoder_loaded.predict(x_test)

n = 10
plt.figure(figsize=(20, 4))
```

```
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i+1)
    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 + 1 + 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()
```

```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 14, 14, 32)        9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 7, 7, 32)          9248
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 14, 14, 32)        9248
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 28, 28, 1)         289
=================================================================
Total params: 28,353
Trainable params: 28,353
Non-trainable params: 0
_________________________________________________________________
```