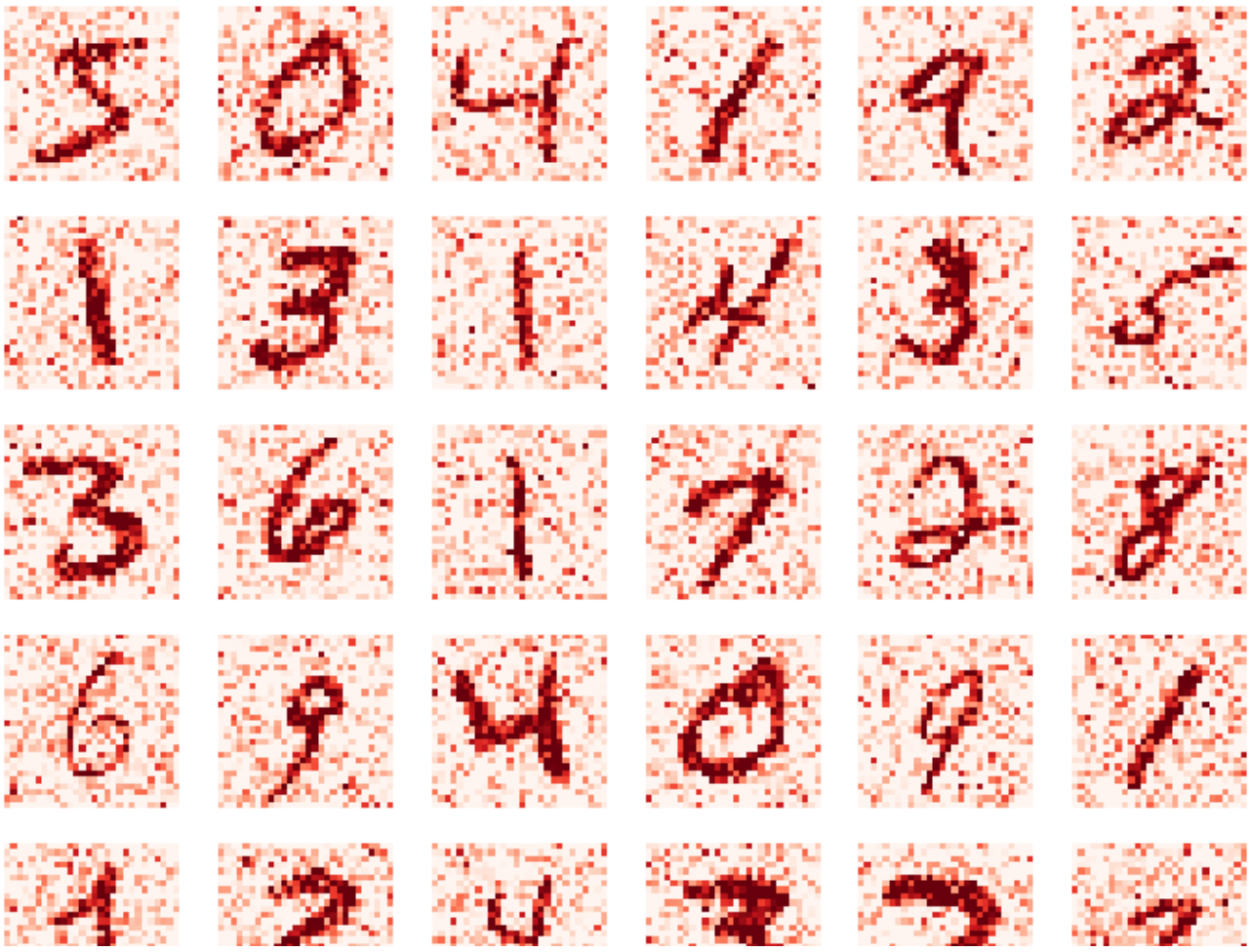
Autoencoder: An autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner. The aim of an autoencoder is to learn a representation for a set of data, typically for dimensionality reduction, by training the network to ignore signal “noise".

```
import numpy as np

import matplotlib.pyplot as plt

from keras.layers import Conv2D, Input, Dense, Dropout, MaxPool2D, UpSampling2D
from keras.models import Model
from keras.datasets import mnist, cifar10

%matplotlib inline


(train, _), (test, _) = mnist.load_data()

# scaling input data
train = train.reshape([-1,28,28,1]) / 255
test = test.reshape([-1,28,28,1]) / 255
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mni
11493376/11490434 [==============================] - 0s 0us/step
```

```
# Adding noise to data
noise = 0.3
train_noise = train + noise * np.random.normal(0, 1, size=train.shape)
test_noise = test + noise * np.random.normal(0, 1, size=test.shape)

train_noise = np.clip(train_noise, 0, 1)
test_noise = np.clip(test_noise, 0, 1)


# sample noisy image

rows = 5 # defining no. of rows in figure
cols = 6 # defining no. of colums in figure
subplot_size = 2

f = plt.figure(figsize=(subplot_size*cols,subplot_size*rows)) # defining a figure

for i in range(rows*cols):
    f.add_subplot(rows,cols,i+1) # adding sub plot to figure on each iteration
    plt.imshow(train_noise[i].reshape([28,28]),cmap="Reds")
    plt.axis("off")
plt.savefig("digits_noise.png")
```



```
# Encoder
inputs = Input(shape=(28,28,1))

x = Conv2D(32, 3, activation='relu', padding='same')(inputs)
x = MaxPool2D()(x)
x = Dropout(0.3)(x)
x = Conv2D(32, 3, activation='relu', padding='same')(x)
encoded = MaxPool2D()(x)


# Decoder

x = Conv2D(32, 3, activation='relu', padding='same')(encoded)
x = UpSampling2D()(x)
x = Dropout(0.3)(x)
x = Conv2D(32, 3, activation='relu', padding='same')(x)
x = UpSampling2D()(x)
decoded = Conv2D(1, 3, activation='sigmoid', padding='same')(x)


autoencoder = Model(inputs, decoded)
autoencoder.compile(optimizer='rmsprop', loss='binary_crossentropy')

autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
input_1 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
dropout (Dropout)            (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
_________________________________________________________________
dropout_1 (Dropout)          (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
_________________________________________________________________
```

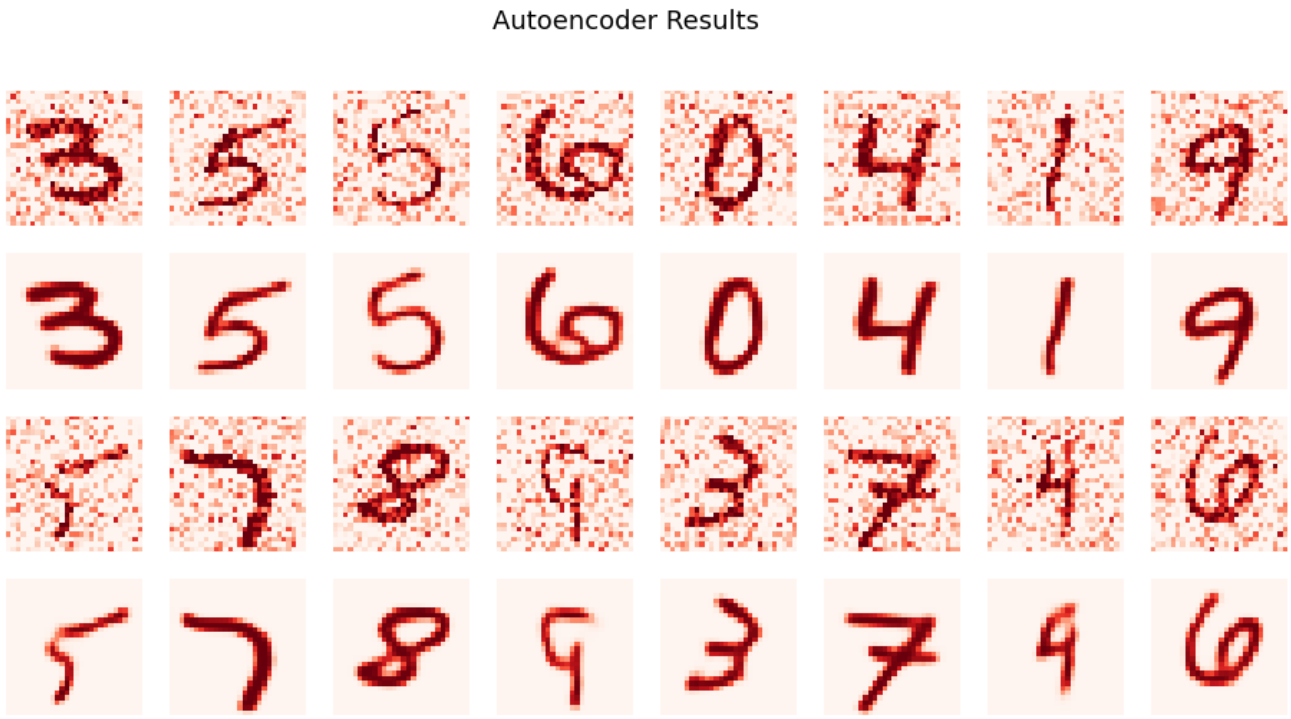
```
epochs = 50
batch_size = 256

history = autoencoder.fit(train_noise,
                train,
                epochs=epochs,
                batch_size=batch_size,
                shuffle=True,
                validation_data=(test_noise, test)
               )
```

```
Epoch 22/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0852 - val_loss:
Epoch 23/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0851 - val_loss:
Epoch 24/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0852 - val_loss:
Epoch 25/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0850 - val_loss:
Epoch 26/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0848 - val_loss:
Epoch 27/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0847 - val_loss:
Epoch 28/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0846 - val_loss:
Epoch 29/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0845 - val_loss:
Epoch 30/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0845 - val_loss:
Epoch 31/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0843 - val_loss:
```

```
Epoch 32/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0843 - val_loss:
Epoch 33/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0842 - val_loss:
Epoch 34/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0841 - val_loss:
Epoch 35/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0842 - val_loss:
Epoch 36/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0841 - val_loss:
Epoch 37/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0840 - val_loss:
Epoch 38/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0839 - val_loss:
Epoch 39/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0840 - val_loss:
Epoch 40/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0840 - val_loss:
Epoch 41/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0838 - val_loss:
Epoch 42/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0838 - val_loss:
Epoch 43/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0836 - val_loss:
Epoch 44/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0837 - val_loss:
Epoch 45/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0836 - val_loss:
Epoch 46/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0836 - val_loss:
Epoch 47/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0837 - val_loss:
Epoch 48/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0835 - val_loss:
Epoch 49/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0836 - val_loss:
Epoch 50/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0835 - val_loss:
```

```
num_imgs = 16
rand = np.random.randint(1, 100)

test_images = test_noise[rand:rand+num_imgs] # slicing
test_desoided = autoencoder.predict(test_images) # predict


# Visualize test images with their denoised images

rows = 2 # defining no. of rows in figure
cols = 8 # defining no. of colums in figure

f = plt.figure(figsize=(2*cols,2*rows*2)) # defining a figure

for i in range(rows):
    for j in range(cols):
        f.add_subplot(rows*2,cols, (2*i*cols)+(j+1)) # adding sub plot to figure on each i
        plt.imshow(test_images[i*cols + j].reshape([28,28]),cmap="Reds")
        plt.axis("off")
```

```
            plt.axis( off )

        for j in range(cols):
            f.add_subplot(rows*2,cols,((2*i+1)*cols)+(j+1)) # adding sub plot to figure on eac
            plt.imshow(test_desoided[i*cols + j].reshape([28,28]),cmap="Reds")
            plt.axis("off")

    f.suptitle("Autoencoder Results",fontsize=18)
    plt.savefig("test_results.png")

    plt.show()
```



✓ 2s completed at 9:41 PM