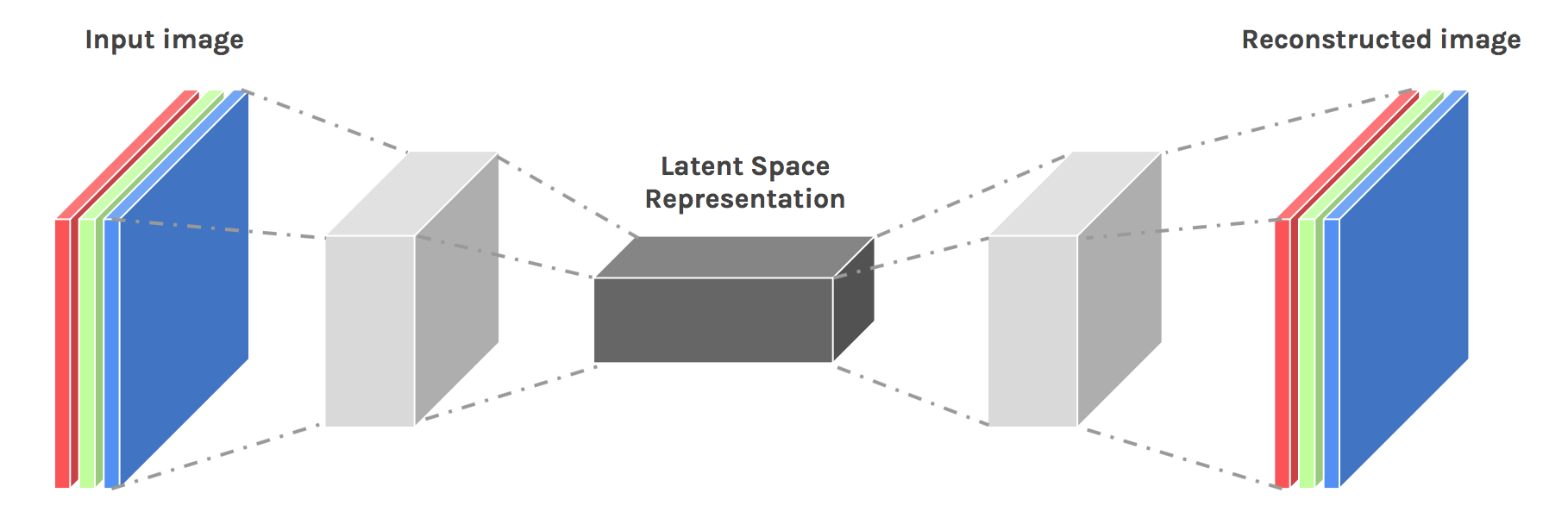
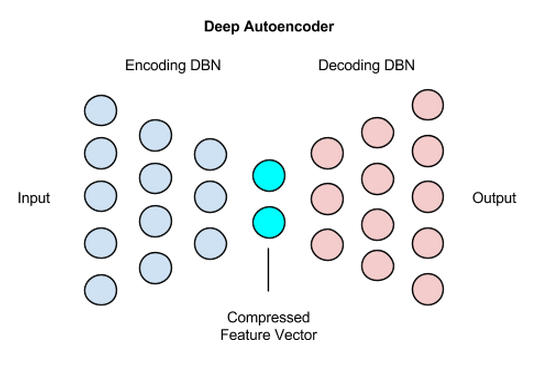
Autoencoders made simple

## What is an autoencoder?



Autoencoders are a type of generative model used for [unsupervised learning](https://en.wikipedia.org/wiki/Unsupervised_learning). Autoencoders learn some latent representation of the image and use that to reconstruct the image. What is this “latent representation”? It is another fancy term for hidden features of the image. Autoencoders, through the iterative process of training with different images tries to learn the features of a given image and reconstruct the desired image from these learned features.



On a first glance, an autoencoder might look like any other neural network but unlike others, it has a bottleneck at the centre. This bottleneck is used to learn the features of the image. An autoencoder does two tasks, it encodes an image and then decodes it.

## Encoding an image:

An input image is taken and through a series of convolutions, the size of the image is condensed into a small vector. This condensed vector represents the features of the image from which another image can be reconstructed

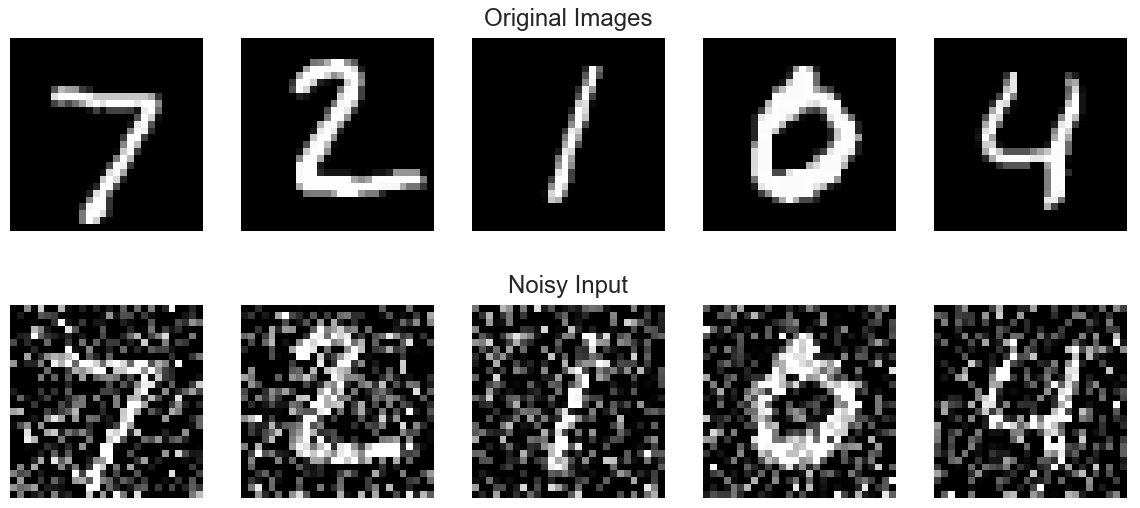
## Decoding an image:

From the condensed vector, we apply a series of deconvolution layers which blows up the size of the image and restores it back to its original size.

## What is the use of an autoencoder?

Autoencoders can be used to remove noise, perform image colourisation and various other purposes. A noisy image can be given as input to the autoencoder and a de-noised image can be provided as output. The autoencoder will try de-noise the image by learning the latent features of the image and using that to reconstruct an image without noise. The reconstruction error can be calculated as a measure of distance between the pixel values of the output image and ground truth image.



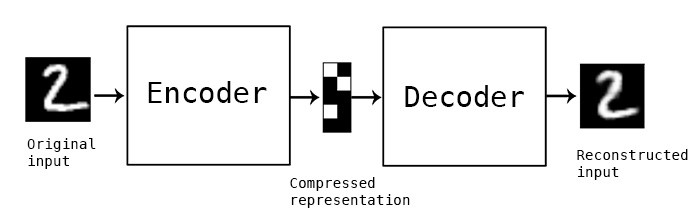


## Disadvantage:

Autoencoders are not that efficient compared to Generative Adversarial Networks in reconstructing an image. As the complexity of the images increase, autoencoders struggle to keep up and images start to get blurry.

## Code:

The python code below represents a basic autoencoder that learns the features from the mnist digits data and reconstructs them back again.



## Dependencies:

* Tensorflow
* Keras
* Numpy

|  |
| --- |
| import keras  from keras.datasets import mnist  from keras.models import Sequential  from keras.layers import Dense, Dropout, Flatten  from keras.layers import Conv2D, MaxPooling2D  from keras import backend as K  (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  img\_rows, img\_cols = 28, 28  if K.image\_data\_format() == 'channels\_first':  x\_train = x\_train.reshape(x\_train.shape[0], 1, img\_rows, img\_cols)  x\_test = x\_test.reshape(x\_test.shape[0], 1, img\_rows, img\_cols)  input\_shape = (1, img\_rows, img\_cols)  else:  x\_train = x\_train.reshape(x\_train.shape[0], img\_rows, img\_cols, 1)  x\_test = x\_test.reshape(x\_test.shape[0], img\_rows, img\_cols, 1)  input\_shape = (img\_rows, img\_cols, 1)  x\_train = x\_train.astype('float32')  x\_test = x\_test.astype('float32')  x\_train /= 255  x\_test /= 255 |

|  |
| --- |
| import numpy as np  temp = []  for img in x\_train:  t = []  for row in img:  for i in row:  t.append(i)  temp.append(t)  x\_train = []  x\_train = temp  x\_train = np.array(x\_train)  x\_train = x\_train.reshape(60000,784)  model = Sequential()  model.add(Dense(784,activation='relu',input\_dim=784))  model.add(Dense(256,activation='relu'))  model.add(Dense(128,activation='relu'))  model.add(Dense(256,activation='relu'))  model.add(Dense(784,activation='relu'))  model.compile(loss=keras.losses.mean\_squared\_error,  optimizer=keras.optimizers.RMSprop(lr=0.0001, rho=0.9, epsilon=None, decay=0.0),  metrics = ['accuracy'])  model.fit(x\_train,x\_train,verbose=1,epochs=10,batch\_size=256)  model.save('C:\\python\\auto\_en.h5') |

Epoch 1/10

60000/60000 [==============================] - 17s 281us/step - loss: 0.0538 - acc: 0.0119

Epoch 2/10

60000/60000 [==============================] - 20s 339us/step - loss: 0.0314 - acc: 0.0128

Epoch 3/10

60000/60000 [==============================] - 20s 339us/step - loss: 0.0264 - acc: 0.0128

Epoch 4/10

60000/60000 [==============================] - 18s 305us/step - loss: 0.0238 - acc: 0.0134

Epoch 5/10

60000/60000 [==============================] - 18s 298us/step - loss: 0.0221 - acc: 0.01411s - lo

Epoch 6/10

60000/60000 [==============================] - 19s 314us/step - loss: 0.0209 - acc: 0.0135

Epoch 7/10

60000/60000 [==============================] - 23s 390us/step - loss: 0.0198 - acc: 0.0134

Epoch 8/10

60000/60000 [==============================] - 22s 360us/step - loss: 0.0190 - acc: 0.0138

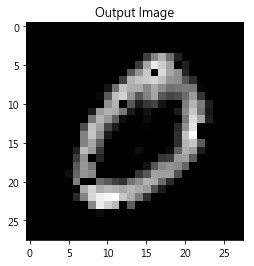
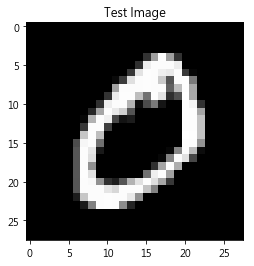
Epoch 9/10

60000/60000 [==============================] - 22s 367us/step - loss: 0.0183 - acc: 0.0136

Epoch 10/10

60000/60000 [==============================] - 24s 392us/step - loss: 0.0177 - acc: 0.01360s - loss: 0.0177 - ac

|  |
| --- |
| from keras.models import load\_model  import matplotlib.pyplot as plt  model = load\_model('C:\\python\\auto\_en.h5')  test = x\_train[1].reshape(1,784)  y\_test = model.predict(test)  inp\_img = []  temp = []  for i in range(len(test[0])):  if((i+1)%28 == 0):  temp.append(test[0][i])  inp\_img.append(temp)  temp = []  else:  temp.append(test[0][i])  out\_img = []  temp = []  for i in range(len(y\_test[0])):  if((i+1)%28 == 0):  temp.append(y\_test[0][i])  out\_img.append(temp)  temp = []  else:  temp.append(y\_test[0][i])    inp\_img = np.array(inp\_img)  out\_img = np.array(out\_img)    plt.imshow(inp\_img, cmap='gray')  plt.title('Test Image')  plt.show()  plt.imshow(out\_img, cmap='gray')  plt.title('Output Image')  plt.show() |



## Conclusion:

Even though autoencoders might struggle to keep up with GANs, they are highly efficient in certain tasks such as anomaly detection and others. This is still a burgeoning field of neural network.