Image Denoising using AutoEncoders

by Varun Kashyap

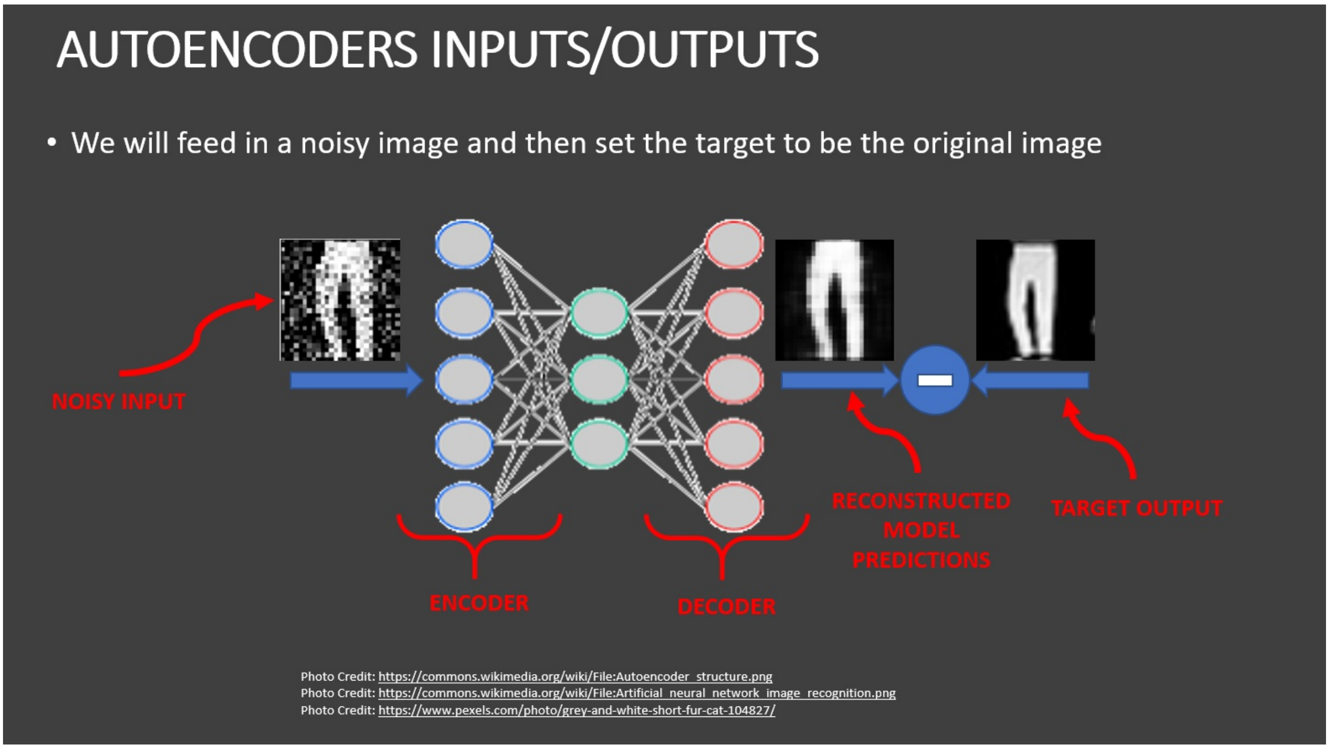
Project Overview

The project provides an insight on the following -

- Understand the theory and intuition behind using Autoencoders.
- Import Key libraries, data-set and visualize images.
- Perform image normalization and add random noise to images.
- Build an autoencoder using Keras with Tensorflow 2.0 as a backend.
- Compile and fit autoencoder model to training data.
- Assess trained model performance.

Purpose of the project is to use data encoders and ANNs to denoise images

- Input : Noisy Images from a fashion dataset.
- Output : Clean denoised images.



Importing the Dataset and the Libraries

In [4]:

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
```

In [5]:

```
# Alternatively, you can use the same dataset made readily available by keras Using the following lines of code:
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
```

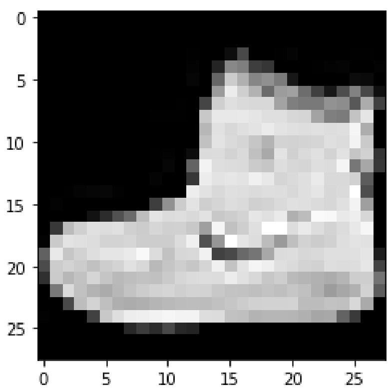
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [==================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 3s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
```

In [6]:

```
plt.imshow(X_train[0], cmap="gray")
```

Out[6]:

```
<matplotlib.image.AxesImage at 0x1dfc6139dc8>
```



In [7]:

```
X_train.shape
```

Out[7]:

```
(60000, 28, 28)
```

In [8]:

```
X_test.shape
```

Out[8]:

```
(10000, 28, 28)
```

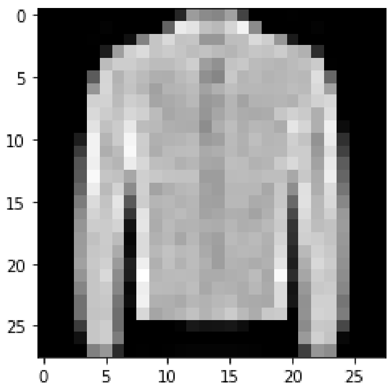
Data Visualisation Phase

In [9]:

```
# Let's view some images
i = random.randint(1,60000) # select any random index from 1 to 60,000
plt.imshow( X_train[i] , cmap = 'gray') # reshape and plot the image
```

Out[9]:

```
<matplotlib.image.AxesImage at 0x1dfc61f8ac8>
```



In [10]:

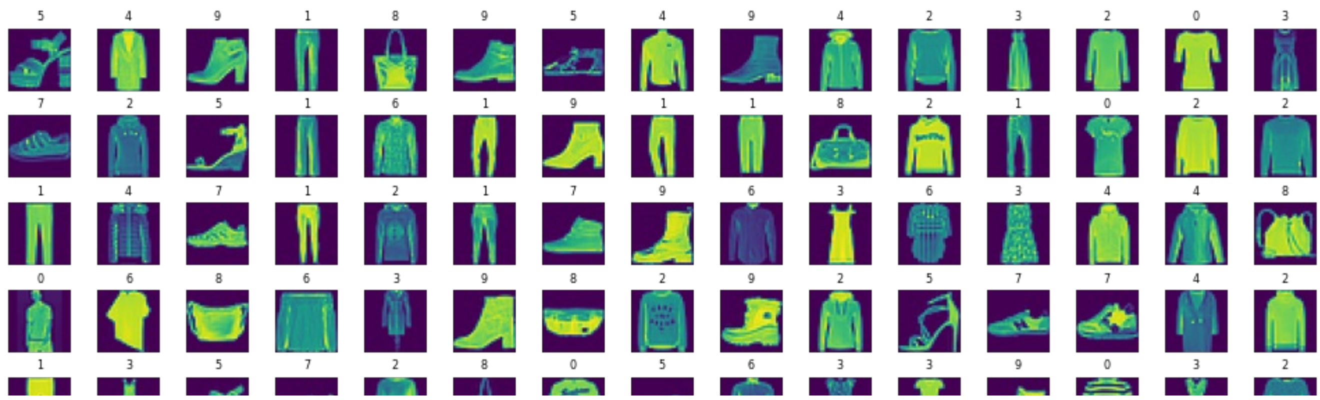
```python
label = y_train[i]
label
```

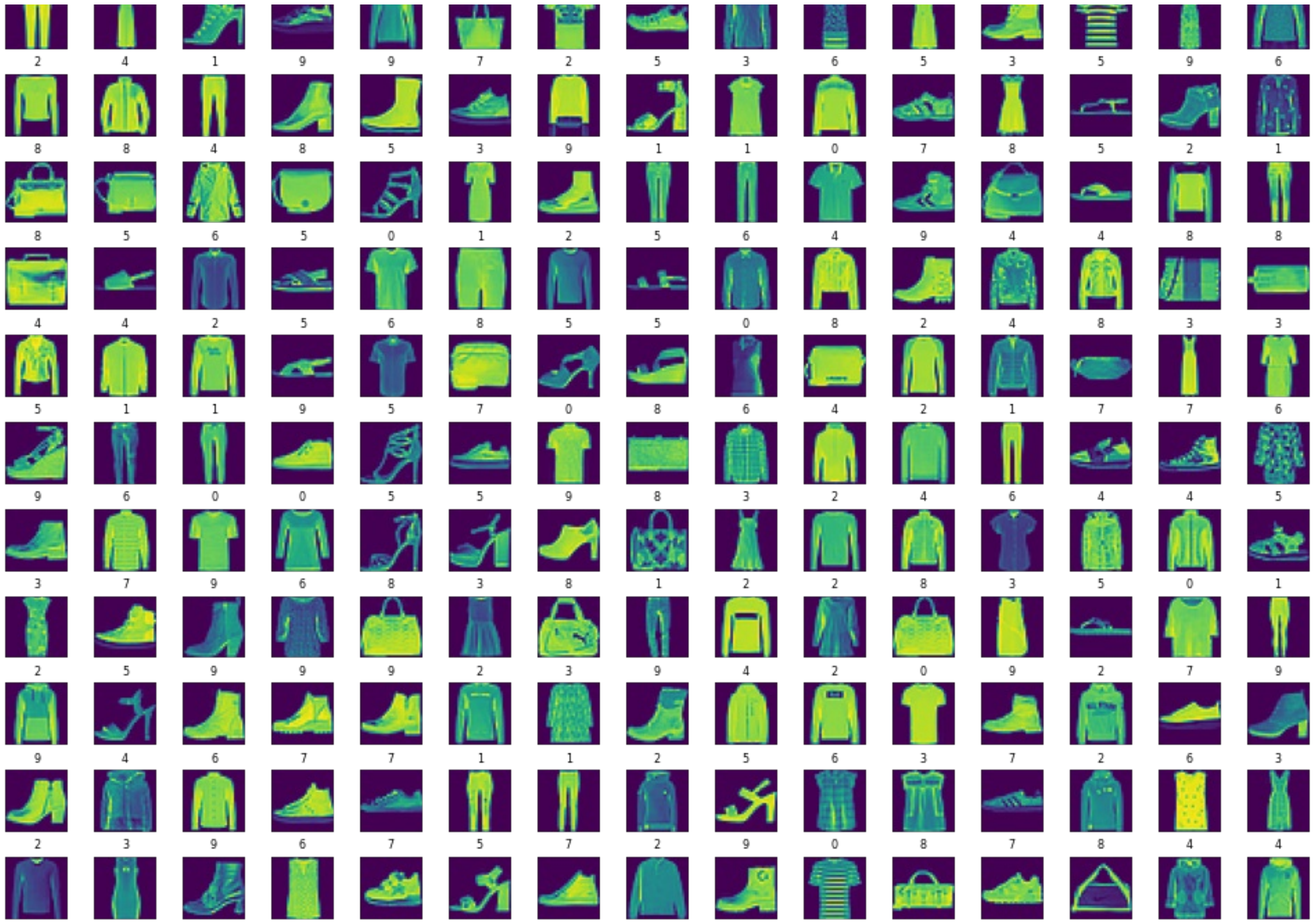
Out[10]:

```
4
```

In [11]:

```python
# Let's view more images in a grid format
# Defining the dimensions of the plot grid
W_grid = 15
L_grid = 15

# fig, axes = plt.subplots(L_grid, W_grid)
# subplot return the figure object and axes object
# the axes of the object can be used to plot specific figures at various locations

fig, axes = plt.subplots(L_grid, W_grid, figsize = (17,17))

axes = axes.ravel() # flatenning the 15 x 15 matrix into 225 array

n_training = len(X_train) # get the length of the training dataset

# Selecting a random number from 0 to n_training
for i in np.arange(0, W_grid * L_grid): # creating evenly spaces variables

    # Selecting a random number
    index = np.random.randint(0, n_training)
    # reading and displaying an image with the selected index
    axes[i].imshow( X_train[index] )
    axes[i].set_title(y_train[index], fontsize = 8)
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.4)
```





Data Preprocessing Phase

In [12]:

```
X_train = X_train / 255
X_test = X_test / 255
```

In [13]:

```
noise_factor = 0.3

noise_dataset = []

for img in X_train:
  noisy_image = img + noise_factor * np.random.randn(*img.shape)
  noisy_image = np.clip(noisy_image, 0., 1.)
  noise_dataset.append(noisy_image)
```

In [14]:

```
noise_dataset = np.array(noise_dataset)
```

In [15]:

```
noise_dataset.shape
```

Out[15]:

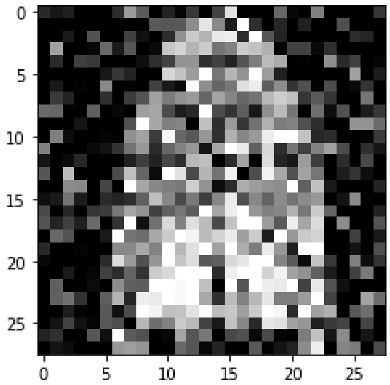
```
(60000, 28, 28)
```

In [16]:

```
plt.imshow(noise_dataset[22], cmap="gray")
```

Out[16]:

```
<matplotlib.image.AxesImage at 0x1dfca486e08>
```



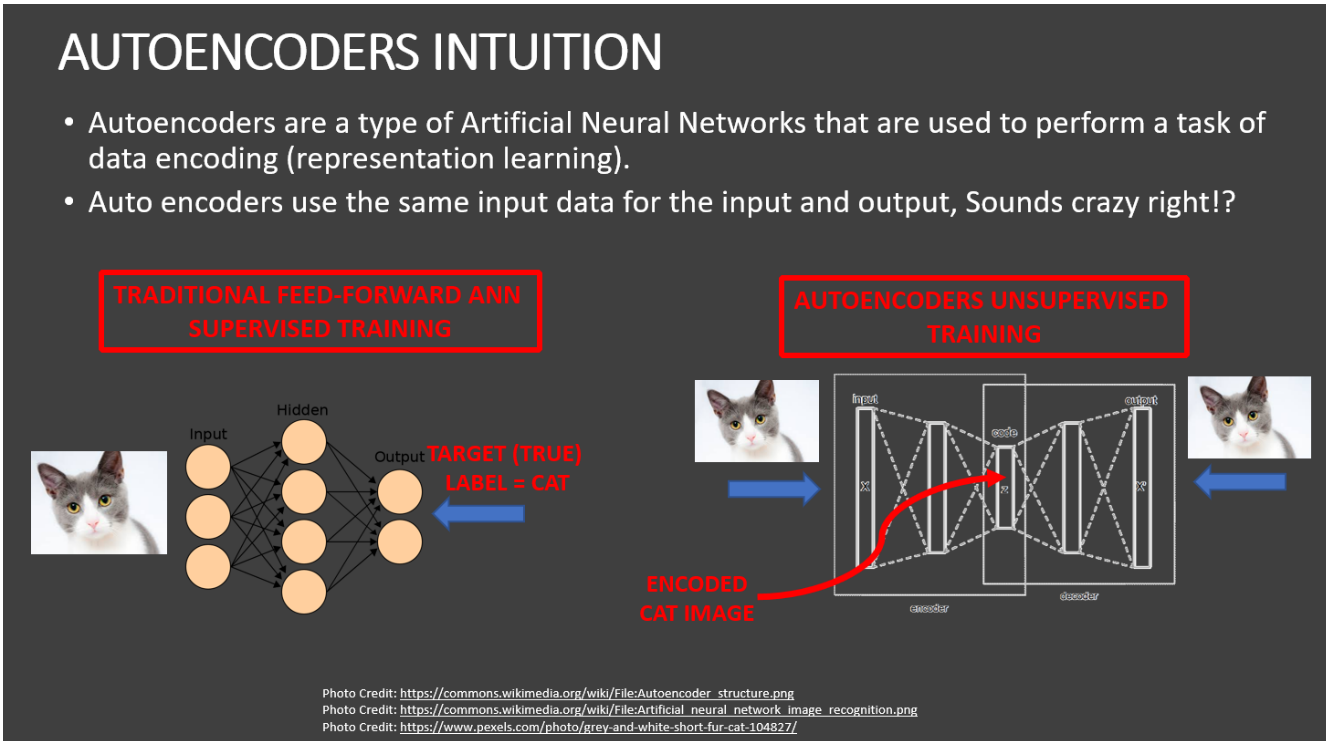
In [17]:

```
noise_test_set = []
for img in X_test:
  noisy_image = img + noise_factor * np.random.randn(*img.shape)
  noisy_image = np.clip(noisy_image, 0., 1.)
  noise_test_set.append(noisy_image)

noise_test_set = np.array(noise_test_set)
noise_test_set.shape
```

Out[17]:

```
(10000, 28, 28)
```

Theory and Intuition behind using AutoEncoders



- Auto encoders work by adding a bottleneck in the network.
- This bottleneck forces the network to create a compressed (encoded) version of the original input
- Auto encoders work well if correlations exists between input data (performs poorly if the all input data is independent)
- Great Reference: “Intro to Auto encoders by Jeremy Jordan”

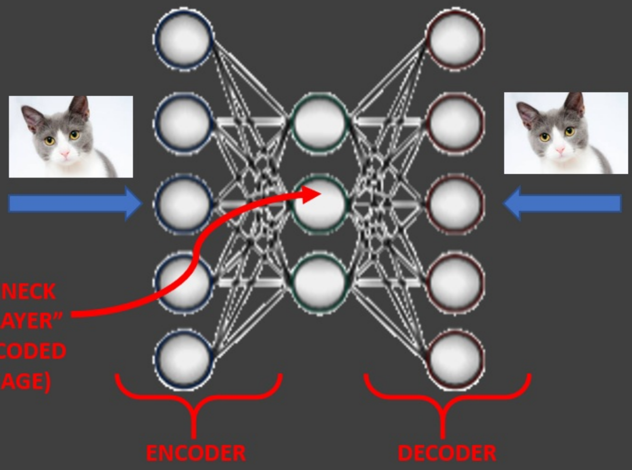


Photo Credit: https://commons.wikimedia.org/wiki/File:Autoencoder_structure.png
Photo Credit: https://commons.wikimedia.org/wiki/File:Artificial_neural_network_image_recognition.png
Photo Credit: https://www.pexels.com/photo/grey-and-white-short-fur-cat-104827/

AUTOENCODER MATH

ENCODER:

$$h(x) = sigmoid(W * x + b)$$

DECODER:

$$\hat{x} = sigmoid(W^* * h(x) + c)$$

TIED WEIGHTS:

Weights from input to hidden layer will be equal to the weights from hidden layer to output

$$W^* = W^T$$

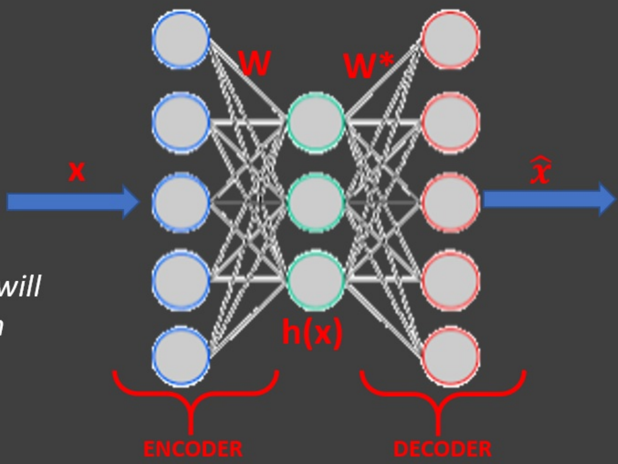
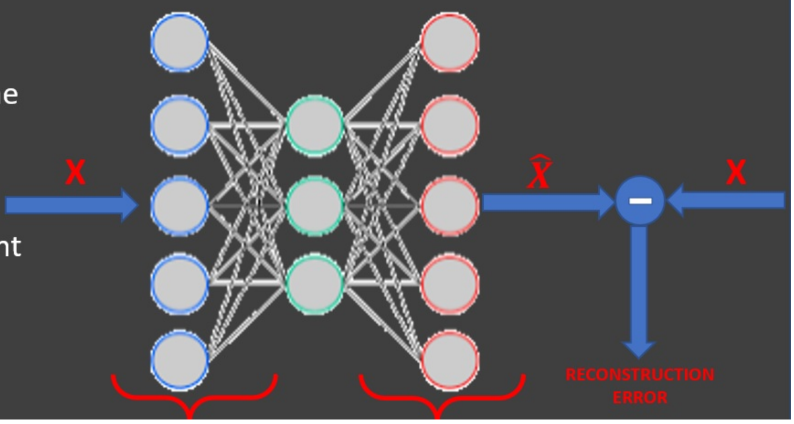
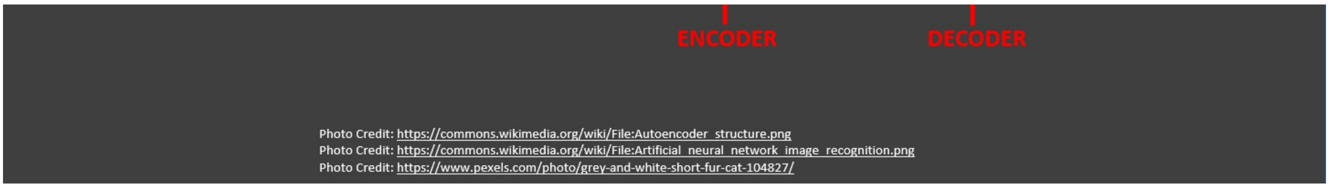


Photo Credit: https://commons.wikimedia.org/wiki/File:Autoencoder_structure.png
Photo Credit: https://commons.wikimedia.org/wiki/File:Artificial_neural_network_image_recognition.png
Photo Credit: https://www.pexels.com/photo/grey-and-white-short-fur-cat-104827/

RECONSTRUCTION ERROR

- Auto encoders objective is to minimize the reconstruction error which is the difference between the input X and the network output $\hat{X}$
- Auto encoders dimensionality reduction (latent space) is quite similar to PCA (Principal Component Analysis) if linear activation functions are used





Building and Training the AutoEncoder Deep Learning Model

In [18]:

```
autoencoder = tf.keras.models.Sequential()

#Encoder
autoencoder.add(tf.keras.layers.Conv2D(filters=16, kernel_size=3, strides=2, padding="same", input_
shape=(28, 28, 1)))
autoencoder.add(tf.keras.layers.Conv2D(filters=8, kernel_size=3, strides=2, padding="same"))

#Encoded image
autoencoder.add(tf.keras.layers.Conv2D(filters=8, kernel_size=3, strides=1, padding="same"))

#Decoder
autoencoder.add(tf.keras.layers.Conv2DTranspose(filters=16, kernel_size=3, strides=2, padding="same
"))
autoencoder.add(tf.keras.layers.Conv2DTranspose(filters=1, kernel_size=3, strides=2, activation='si
gmoid', padding="same"))
```

In [19]:

```
autoencoder.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(lr=0.001))
autoencoder.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 14, 14, 16)        160
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 7, 7, 8)           1160
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 8)           584
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 14, 14, 16)        1168
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 28, 28, 1)         145
=================================================================
Total params: 3,217
Trainable params: 3,217
Non-trainable params: 0
_________________________________________________________________
```

In [20]:

```
autoencoder.fit(noise_dataset.reshape(-1, 28, 28, 1),
                X_train.reshape(-1, 28, 28, 1),
                epochs=10,
                batch_size=200,
                validation_data=(noise_test_set.reshape(-1, 28, 28, 1), X_test.reshape(-1, 28, 28, 1
)))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 40s 661us/sample - loss: 0.3972 - val_loss: 0.3257
Epoch 2/10
60000/60000 [==============================] - 35s 590us/sample - loss: 0.3161 - val_loss: 0.3142
Epoch 3/10
60000/60000 [==============================] - 40s 669us/sample - loss: 0.3097 - val_loss: 0.3099
Epoch 4/10
60000/60000 [==============================] - 37s 621us/sample - loss: 0.3062 - val_loss: 0.3071
```

```
60000/60000 [==============================] - 37s 621us/sample - loss: 0.3062 - val_loss: 0.3071
Epoch 5/10
60000/60000 [==============================] - 35s 581us/sample - loss: 0.3042 - val_loss: 0.3058
Epoch 6/10
60000/60000 [==============================] - 36s 607us/sample - loss: 0.3031 - val_loss: 0.3048
Epoch 7/10
60000/60000 [==============================] - 36s 605us/sample - loss: 0.3023 - val_loss: 0.3042
Epoch 8/10
60000/60000 [==============================] - 36s 595us/sample - loss: 0.3016 - val_loss: 0.3036
Epoch 9/10
60000/60000 [==============================] - 35s 585us/sample - loss: 0.3010 - val_loss: 0.3029
Epoch 10/10
60000/60000 [==============================] - 36s 606us/sample - loss: 0.3006 - val_loss: 0.3025
```

Out[20]:

```
<tensorflow.python.keras.callbacks.History at 0x1dfc5bca4c8>
```

Performance Evaluation of the Trained Model

In [21]:

```python
evaluation = autoencoder.evaluate(noise_test_set.reshape(-1, 28, 28, 1), X_test.reshape(-1, 28, 28,
1))
print('Test Accuracy : {:.3f}'.format(evaluation))
```

```
10000/10000 [==============================] - 5s 500us/sample - loss: 0.3025
Test Accuracy : 0.303
```

In [22]:

```python
predicted = autoencoder.predict(noise_test_set[:10].reshape(-1, 28, 28, 1))
```

In [23]:

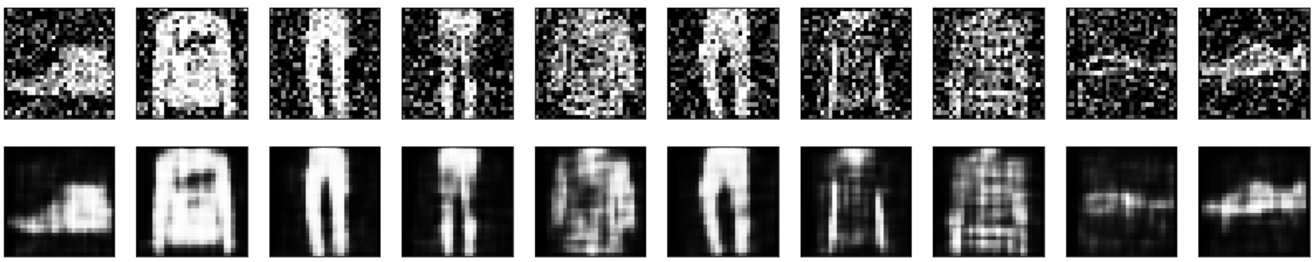
```python
predicted.shape
```

Out[23]:

```
(10, 28, 28, 1)
```

In [24]:

```python
fig, axes = plt.subplots(nrows=2, ncols=10, sharex=True, sharey=True, figsize=(20,4))
for images, row in zip([noise_test_set[:10], predicted], axes):
    for img, ax in zip(images, row):
        ax.imshow(img.reshape((28, 28)), cmap='Greys_r')
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
```



0 = T-shirt/top 1 = Trouser 2 = Pullover 3 = Dress 4 = Coat 5 = Sandal 6 = Shirt 7 = Sneaker 8 = Bag 9 = Ankle boot

The top row is the row of noisy images, the bottom row consists of denoised images.

In [ ]: