```
nama : saila Julia
nim : 20220040082
```

```
#import all the dependencies
from keras.layers import Dense,Conv2D,MaxPooling2D,UpSampling2D
from keras import Input, Model
from keras.datasets import  fashion_mnist
import numpy as np
import matplotlib.pyplot as plt
```

Then we will build our model and we will provide the number of dimensions that will decide how much the input will be compressed. The lesser the dimension, the more will be the compression.

```
encoding_dim = 15
input_img = Input(shape=(784,))
# encoded representation of input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# decoded representation of code
decoded = Dense(784, activation='sigmoid')(encoded)
# Model which take input image and shows decoded images
autoencoder = Model(input_img, decoded)
```

Then we need to build the encoder model and decoder model separately so that we can easily differentiate between the input and output.

```
# This model shows encoded images
encoder = Model(input_img, encoded)
# Creating a decoder model
encoded_input = Input(shape=(encoding_dim,))
# last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))
```

```
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

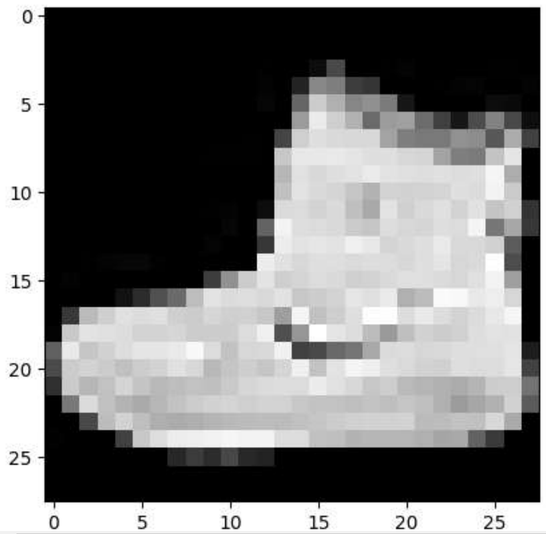
Then we need to compile the model with the ADAM optimizer and cross-entropy loss function fitment.

```
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
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)
```

```
(60000, 784)
(10000, 784)
```

```
plt.imshow(x_train[0].reshape(28,28))
```

```
<matplotlib.image.AxesImage at 0x784e53aa3520>
```

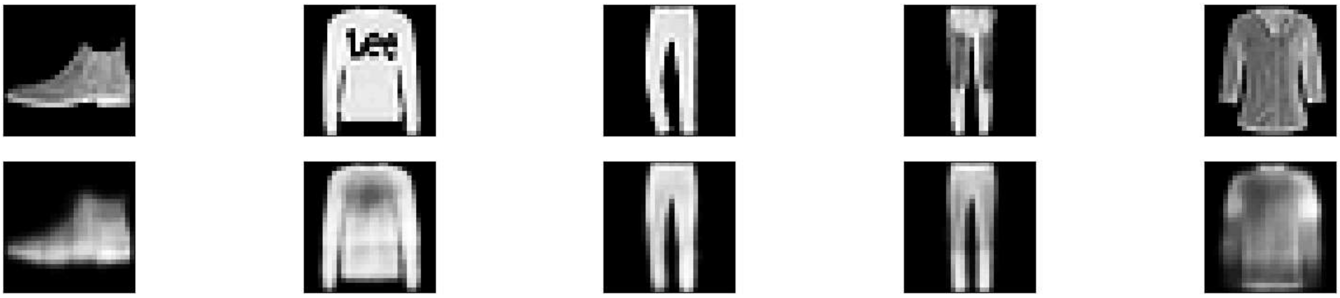


```
autoencoder.fit(x_train, x_train,
                epochs=30,
                batch_size=256,
                validation_data=(x_test, x_test))
```

```
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3520 - val_loss: 0.3336
Epoch 3/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3278 - val_loss: 0.3217
Epoch 4/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3185 - val_loss: 0.3148
Epoch 5/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3113 - val_loss: 0.3101
Epoch 6/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3075 - val_loss: 0.3073
Epoch 7/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3047 - val_loss: 0.3053
Epoch 8/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3031 - val_loss: 0.3041
Epoch 9/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3016 - val_loss: 0.3035
Epoch 10/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3012 - val_loss: 0.3030
Epoch 11/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3006 - val_loss: 0.3028
Epoch 12/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3006 - val_loss: 0.3024
Epoch 13/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2998 - val_loss: 0.3022
Epoch 14/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2998 - val_loss: 0.3020
Epoch 15/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2996 - val_loss: 0.3018
Epoch 16/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2996 - val_loss: 0.3017
Epoch 17/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2994 - val_loss: 0.3015
Epoch 18/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2989 - val_loss: 0.3014
Epoch 19/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2991 - val_loss: 0.3013
Epoch 20/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2990 - val_loss: 0.3011
Epoch 21/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2986 - val_loss: 0.3011
Epoch 22/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2985 - val_loss: 0.3010
Epoch 23/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2994 - val_loss: 0.3010
Epoch 24/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2986 - val_loss: 0.3008
Epoch 25/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2986 - val_loss: 0.3010
Epoch 26/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2991 - val_loss: 0.3008
Epoch 27/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2989 - val_loss: 0.3007
Epoch 28/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2989 - val_loss: 0.3006
Epoch 29/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2986 - val_loss: 0.3007
Epoch 30/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2986 - val_loss: 0.3005
<keras.src.callbacks.history.History at 0x784e5393e9b0>
```

```
#After training, you need to provide the input and you can plot the results using the following code :

encoded_img = encoder.predict(x_test)
decoded_img = decoder.predict(encoded_img)
plt.figure(figsize=(20, 4))
for i in range(5):
    # Display original
    ax = plt.subplot(2, 5, 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, 5, i + 1 + 5)
    plt.imshow(decoded_img[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```



Deep CNN Autoencoder :

Since the input here is images, it does make more sense to use a Convolutional Neural network or CNN. The encoder will be made up of a stack of Conv2D and max-pooling layer and the decoder will have a stack of Conv2D and Upsampling Layer.

```
from tensorflow.keras.models import Sequential
model = Sequential()
# encoder network
model.add(Conv2D(30, 3, activation= 'relu', padding='same', input_shape = (28,28,1)))
model.add(MaxPooling2D(2, padding= 'same'))
model.add(Conv2D(15, 3, activation= 'relu', padding='same'))
model.add(MaxPooling2D(2, padding= 'same'))
#decoder network
model.add(Conv2D(15, 3, activation= 'relu', padding='same'))
model.add(UpSampling2D(2))
model.add(Conv2D(30, 3, activation= 'relu', padding='same'))
model.add(UpSampling2D(2))
model.add(Conv2D(1,3,activation='sigmoid', padding= 'same')) # output layer
model.compile(optimizer= 'adam', loss = 'binary_crossentropy')
model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 28, 28, 30)	300
max_pooling2d_4 (MaxPooling2D)	(None, 14, 14, 30)	0
conv2d_11 (Conv2D)	(None, 14, 14, 15)	4,065
max_pooling2d_5 (MaxPooling2D)	(None, 7, 7, 15)	0
conv2d_12 (Conv2D)	(None, 7, 7, 15)	2,040
up_sampling2d_4 (UpSampling2D)	(None, 14, 14, 15)	0
conv2d_13 (Conv2D)	(None, 14, 14, 30)	4,080
up_sampling2d_5 (UpSampling2D)	(None, 28, 28, 30)	0
conv2d_14 (Conv2D)	(None, 28, 28, 1)	271

Total params: 10,756 (42.02 KB)

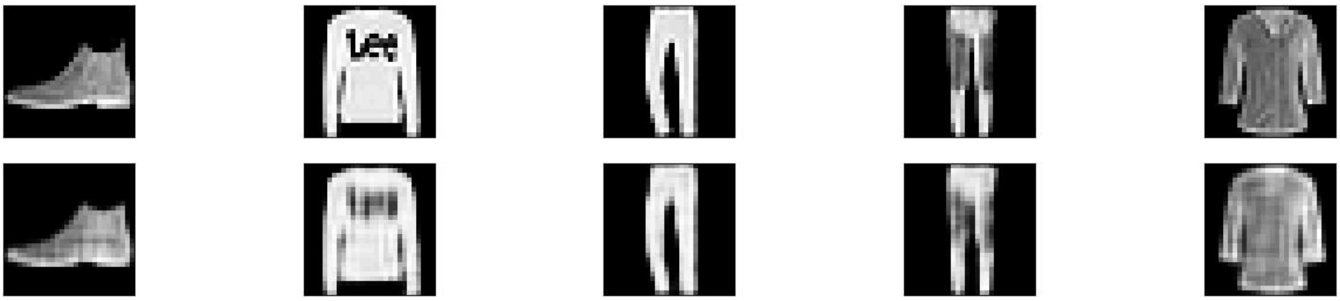
```
(x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
```

```
(x_train, _), (x_test, _) = fashion_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))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
model.fit(x_train, x_train,
                epochs=10,
                batch_size=128,
                validation_data=(x_test, x_test))
```

```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 7s 8ms/step - loss: 0.3735 - val_loss: 0.2834
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.2793 - val_loss: 0.2769
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2729 - val_loss: 0.2728
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - loss: 0.2694 - val_loss: 0.2702
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 0.2675 - val_loss: 0.2684
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.2667 - val_loss: 0.2671
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2642 - val_loss: 0.2663
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.2635 - val_loss: 0.2648
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 0.2629 - val_loss: 0.2644
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - loss: 0.2613 - val_loss: 0.2630
<keras.src.callbacks.history.History at 0x784e5253db10>
```

```
pred = model.predict(x_test)
plt.figure(figsize=(20, 4))
for i in range(5):
    # Display original
    ax = plt.subplot(2, 5, 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, 5, i + 1 + 5)
    plt.imshow(pred[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

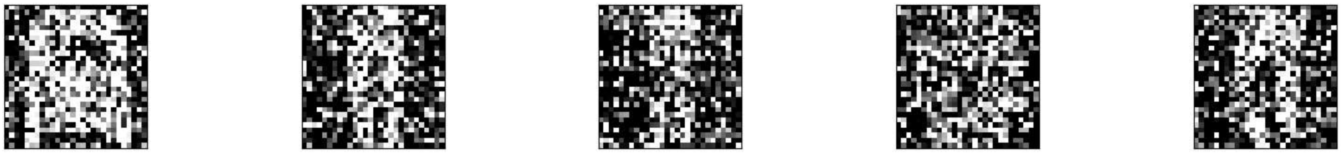
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```



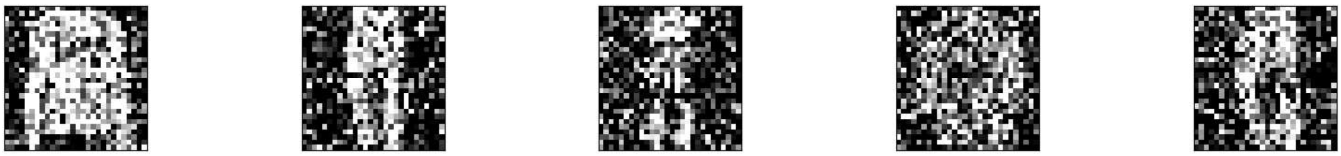
Denoising Autoencoder Now we will see how the model performs with noise in the image. What we mean by noise is blurry images, changing the color of the images, or even white markers on the image.

```
noise_factor = 0.7
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)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)

plt.figure(figsize=(20, 2))
for i in range(1, 5 + 1):
    ax = plt.subplot(1, 5, i)
    plt.imshow(x_test_noisy[i].reshape(28, 28))
    plt.gray()
```

```
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



```
# Add Gaussian noise to the images
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0, scale=1, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0, scale=1, size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0.0, 1.0)
x_test_noisy = np.clip(x_test_noisy, 0.0, 1.0)

plt.figure(figsize=(20, 2))
for i in range(1, 5 + 1):
    ax = plt.subplot(1, 5, i)
    plt.imshow(x_test_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



```
model = Sequential()
# encoder network
model.add(Conv2D(35, 3, activation= 'relu', padding='same', input_shape = (28,28,1)))
model.add(MaxPooling2D(2, padding= 'same'))
model.add(Conv2D(25, 3, activation= 'relu', padding='same'))
model.add(MaxPooling2D(2, padding= 'same'))
#decoder network
model.add(Conv2D(25, 3, activation= 'relu', padding='same'))
model.add(UpSampling2D(2))
model.add(Conv2D(35, 3, activation= 'relu', padding='same'))
model.add(UpSampling2D(2))
model.add(Conv2D(1,3,activation='sigmoid', padding= 'same')) # output layer
model.compile(optimizer= 'adam', loss = 'binary_crossentropy')
model.fit(x_train_noisy, x_train,
                epochs=5,
                batch_size=128,
                validation_data=(x_test_noisy, x_test))
```

```
Epoch 1/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 7s 10ms/step - loss: 0.3840 - val_loss: 0.3112
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - loss: 0.3066 - val_loss: 0.3050
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.3019 - val_loss: 0.3029
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 0.2998 - val_loss: 0.3009
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - loss: 0.2981 - val_loss: 0.2996
<keras.src.callbacks.history.History at 0x784e53b2d060>
```

```
pred = model.predict(x_test_noisy)
plt.figure(figsize=(20, 4))
for i in range(5):
    # Display original
    ax = plt.subplot(2, 5, 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, 5, i + 1 + 5)
    plt.imshow(pred[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step
```