```
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')
```

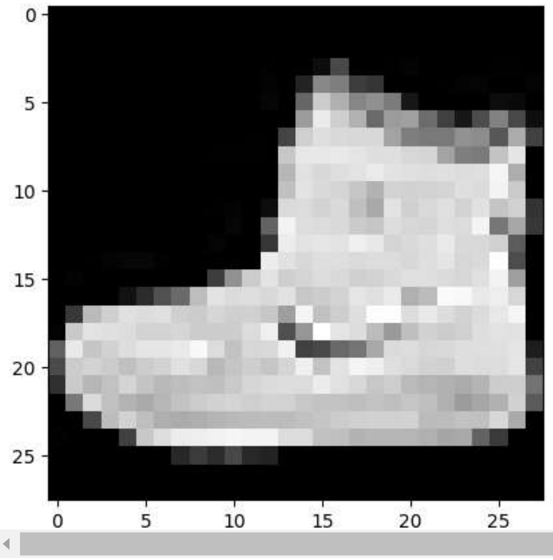
Then we need to compile the model with the ADAM optimizer and cross-entropy loss function fitment.

```
(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)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
(60000, 784)
(10000, 784)
```

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

```
<matplotlib.image.AxesImage at 0x7c29669095a0>
```



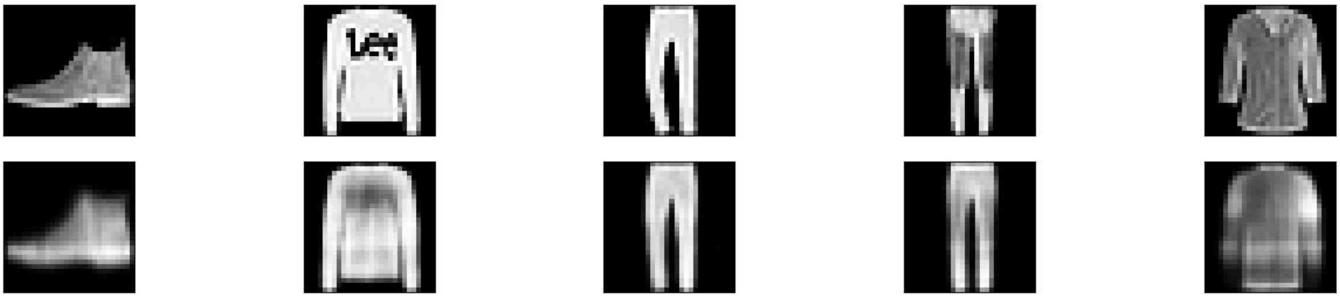
```
autoencoder.fit(x_train, x_train,
                epochs=30,
                batch_size=256,
                validation_data=(x_test, x_test))
```

```
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3536 - val_loss: 0.3366
Epoch 3/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3313 - val_loss: 0.3244
Epoch 4/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3198 - val_loss: 0.3168
Epoch 5/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3140 - val_loss: 0.3124
Epoch 6/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3078 - val_loss: 0.3078
Epoch 7/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3056 - val_loss: 0.3060
Epoch 8/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3036 - val_loss: 0.3043
Epoch 9/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.3031 - val_loss: 0.3034
Epoch 10/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3017 - val_loss: 0.3028
Epoch 11/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.3004 - val_loss: 0.3025
Epoch 12/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3004 - val_loss: 0.3022
Epoch 13/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2999 - val_loss: 0.3019
Epoch 14/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2998 - val_loss: 0.3017
Epoch 15/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2997 - val_loss: 0.3015
Epoch 16/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2997 - val_loss: 0.3016
Epoch 17/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2991 - val_loss: 0.3014
Epoch 18/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2990 - val_loss: 0.3012
Epoch 19/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2991 - val_loss: 0.3010
Epoch 20/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2982 - val_loss: 0.3008
Epoch 21/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2989 - val_loss: 0.3008
Epoch 22/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2991 - val_loss: 0.3009
Epoch 23/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2989 - val_loss: 0.3007
Epoch 24/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2980 - val_loss: 0.3005
Epoch 25/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2986 - val_loss: 0.3005
Epoch 26/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2982 - val_loss: 0.3004
Epoch 27/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2982 - val_loss: 0.3003
Epoch 28/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2981 - val_loss: 0.3003
Epoch 29/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2978 - val_loss: 0.3002
```

```
#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()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 30)	300
max_pooling2d (MaxPooling2D)	(None, 14, 14, 30)	0
conv2d_1 (Conv2D)	(None, 14, 14, 15)	4,065
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 15)	0
conv2d_2 (Conv2D)	(None, 7, 7, 15)	2,040
up_sampling2d (UpSampling2D)	(None, 14, 14, 15)	0
conv2d_3 (Conv2D)	(None, 14, 14, 30)	4,080
up_sampling2d_1 (UpSampling2D)	(None, 28, 28, 30)	0
conv2d_4 (Conv2D)	(None, 28, 28, 1)	271

```
 Total params: 10,756 (42.02 KB)
 Trainable params: 10,756 (42.02 KB)
```

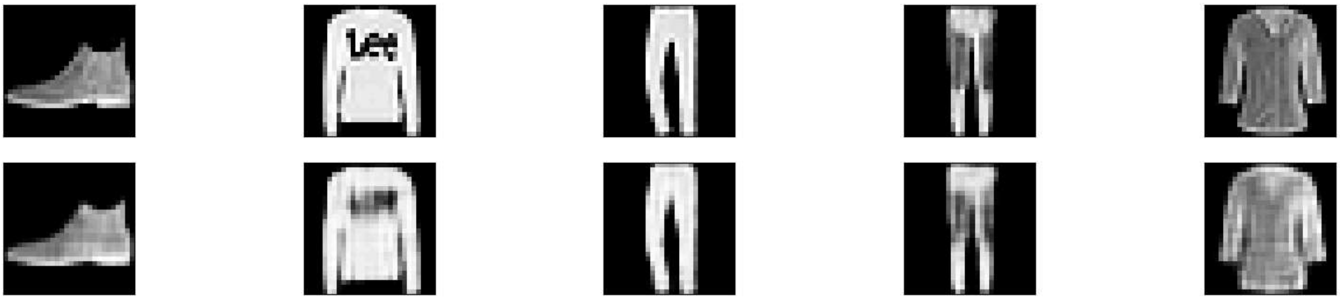
```
(x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
```

```
(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 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - loss: 0.3612 - val_loss: 0.2808
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - loss: 0.2762 - val_loss: 0.2745
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.2713 - val_loss: 0.2714
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 0.2687 - val_loss: 0.2700
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 0.2664 - val_loss: 0.2669
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - loss: 0.2641 - val_loss: 0.2651
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - loss: 0.2631 - val_loss: 0.2637
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.2608 - val_loss: 0.2626
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.2604 - val_loss: 0.2617
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.2598 - val_loss: 0.2611
<keras.src.callbacks.history.History at 0x7c292b917700>
```

```
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()
```

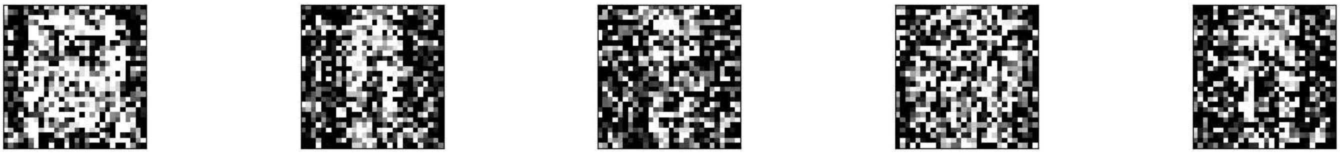
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
```



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.

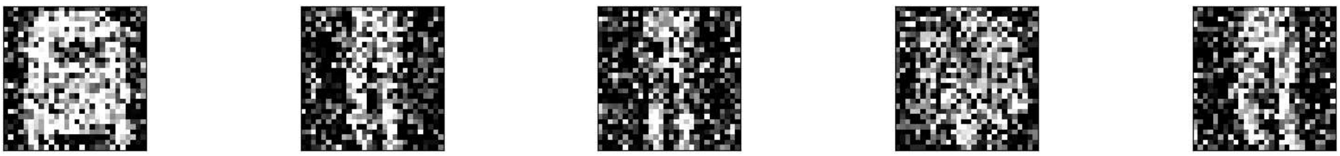
```
noise_factor = 0.7
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 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 ━━━━━━━━━━━━━━━━━━━━ 9s 11ms/step - loss: 0.3843 - val_loss: 0.3117
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.3076 - val_loss: 0.3063
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - loss: 0.3031 - val_loss: 0.3035
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 0.3012 - val_loss: 0.3022
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.2992 - val_loss: 0.3002
<keras.src.callbacks.history.History at 0x7c292c6cfe20>
```

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