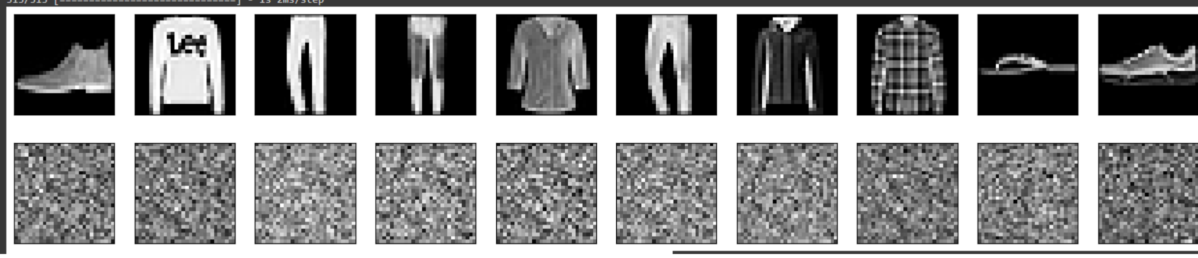
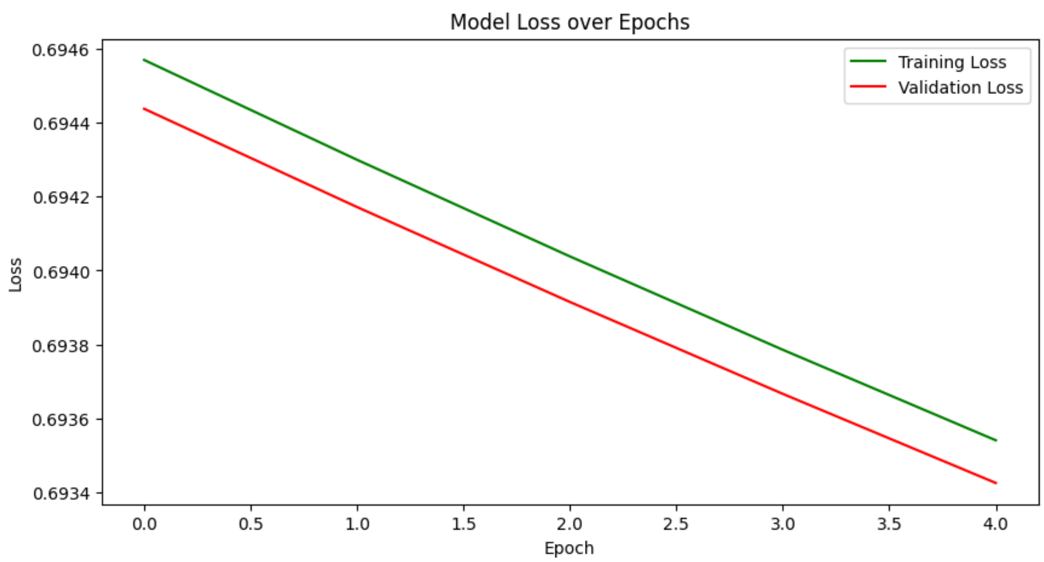
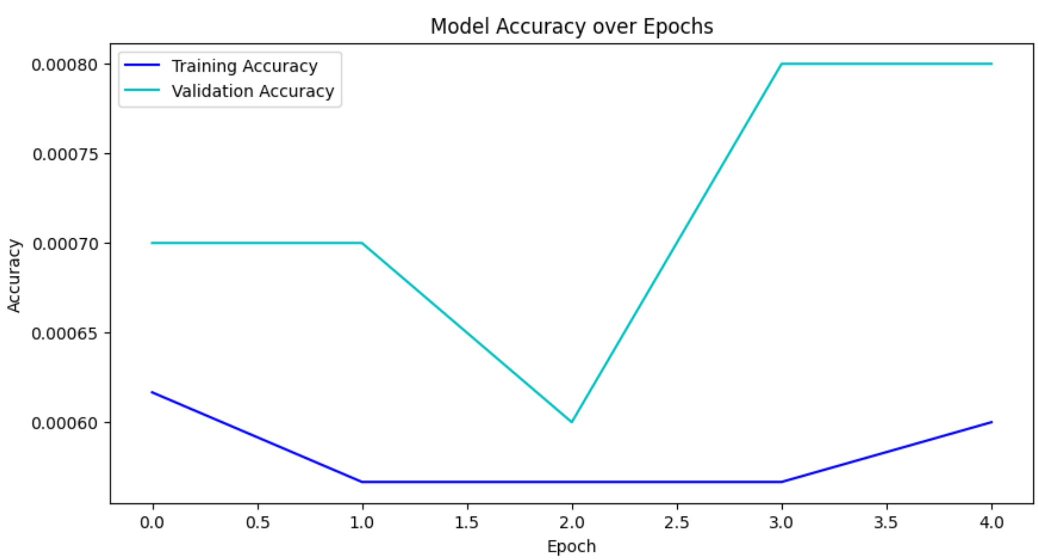
```python
[2] from keras.layers import Input, Dense
    from keras.models import Model
    from keras.datasets import fashion_mnist
    import numpy as np
    import matplotlib.pyplot as plt

    # Loading Fashion MNIST dataset
    (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
    # Normalizing the pixel values to the range of 0 to 1
    x_train = x_train.astype('float32') / 255.
    x_test = x_test.astype('float32') / 255.
    # Reshaping the data to fit the model input
    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:])))

    # Model Architecture
    encoding_dim = 32  # Compressing the image to a 32 float representation
    input_img = Input(shape=(784,))  # Input layer for flattened 28x28 Fashion MNIST images

    # Encoded layer with reduced dimensionality
    encoded = Dense(encoding_dim, activation='relu')(input_img)
    # Adding a hidden layer for enhanced feature extraction
    hidden = Dense(64, activation='relu')(encoded)
    # Decoded layer reconstructs the input from the encoded representation
    decoded = Dense(784, activation='sigmoid')(encoded)

    # Compiling the Autoencoder model
    autoencoder = Model(input_img, decoded)
    autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=['accuracy'])

    # Training the Autoencoder
    history = autoencoder.fit(x_train, x_train,
                              epochs=5,
                              batch_size=256,
                              shuffle=True,
                              validation_data=(x_test, x_test))

    # Generating reconstructed images from the test set
    decoded_imgs = autoencoder.predict(x_test)

    # Visualizing the Original and Reconstructed Images
    n = 10  # Number of images to display
    plt.figure(figsize=(20, 4))
    for i in range(n):
```

```python
    # Display original images
    ax = plt.subplot(2, n, 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 reconstructed images
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()

# Plotting the Training and Validation Loss
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], 'g-', label='Training Loss')
plt.plot(history.history['val_loss'], 'r-', label='Validation Loss')
plt.title('Model Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plotting the Training and Validation Accuracy
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], 'b-', label='Training Accuracy')
plt.plot(history.history['val_accuracy'], 'c-', label='Validation Accuracy')
plt.title('Model Accuracy over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
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 [==============================] - 0s 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 [==============================] - 0s 0us/step
Epoch 1/5
235/235 [==============================] - 5s 15ms/step - loss: 0.6946 - accuracy: 6.1667e-04 - val_loss: 0.6944 - val_accuracy: 7.0000e-04
Epoch 2/5
235/235 [==============================] - 3s 11ms/step - loss: 0.6943 - accuracy: 5.6667e-04 - val_loss: 0.6942 - val_accuracy: 7.0000e-04
Epoch 3/5
235/235 [==============================] - 3s 14ms/step - loss: 0.6940 - accuracy: 5.6667e-04 - val_loss: 0.6939 - val_accuracy: 6.0000e-04
Epoch 4/5
235/235 [==============================] - 4s 16ms/step - loss: 0.6938 - accuracy: 5.6667e-04 - val_loss: 0.6937 - val_accuracy: 8.0000e-04
Epoch 5/5
235/235 [==============================] - 3s 12ms/step - loss: 0.6935 - accuracy: 6.0000e-04 - val_loss: 0.6934 - val_accuracy: 8.0000e-04
313/313 [==============================] - 1s 2ms/step
```



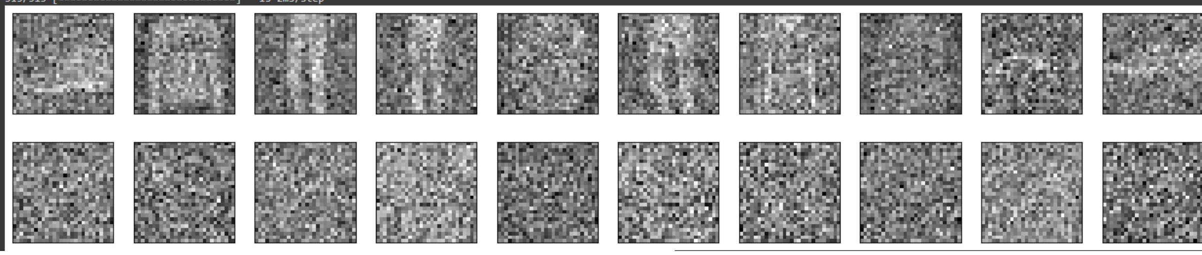


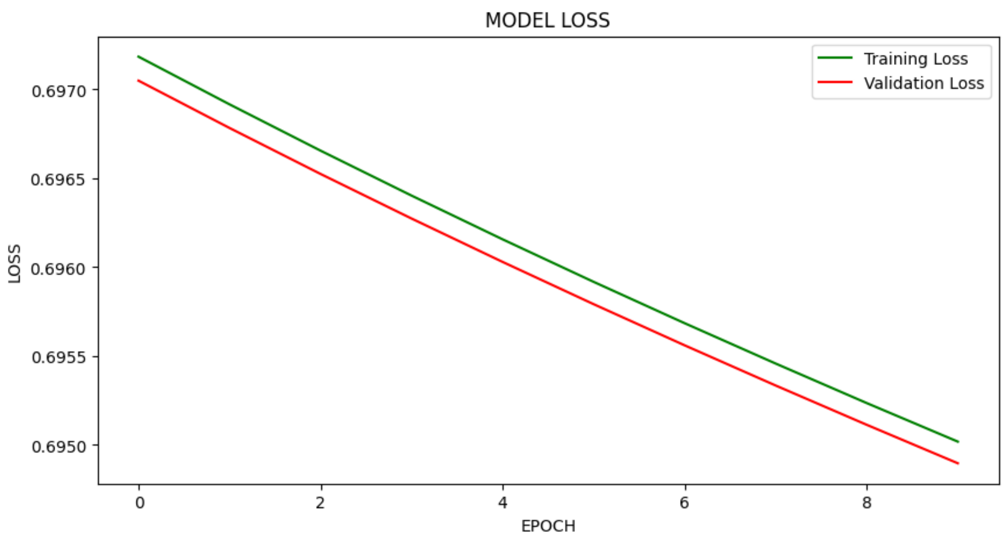


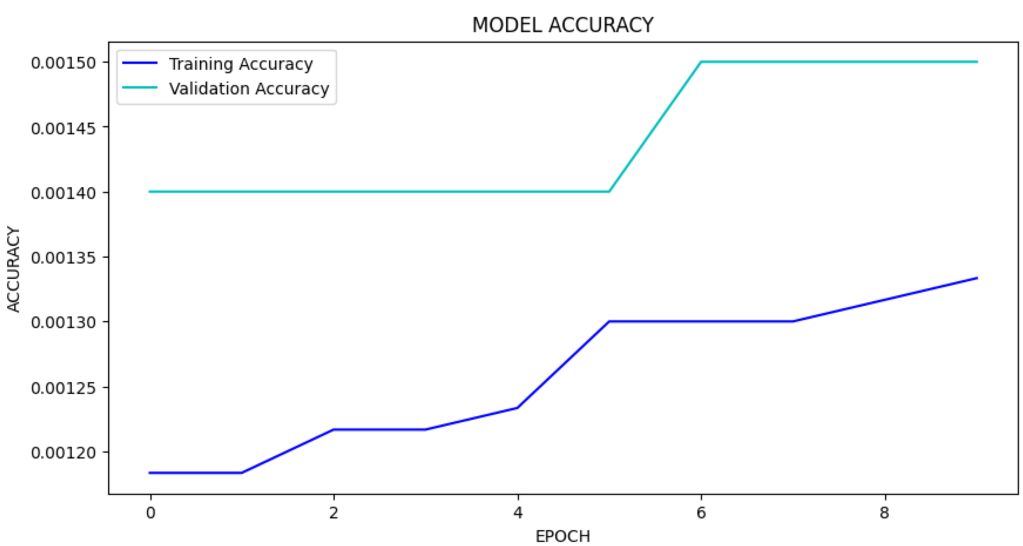
[3]

```python
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import fashion_mnist
import numpy as np
import matplotlib.pyplot as plt

# Loading the data
(x_train, _), (x_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:])))

# Introducing the Noise
noise_factor = 0.5
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)

# Model definition:

# this is the size of our encoded representations
encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats
# this is our input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)
# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)
# this model maps an input to its encoded representation
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=['accuracy'])

# Training the model
history = autoencoder.fit(x_train_noisy, x_train,
                          epochs=10,
                          batch_size=256,
                          shuffle=True,
                          validation_data=(x_test_noisy, x_test_noisy))

# Predictions on the test data
decoded_imgs = autoencoder.predict(x_test_noisy)

# Visualization of noisy and reconstructed images
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # Noisy data
    ax = plt.subplot(2, n, 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)

    # Reconstruction data
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()

# Plotting the Loss
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], 'g-', label='Training Loss')
plt.plot(history.history['val_loss'], 'r-', label='Validation Loss')
plt.title('MODEL LOSS')
plt.xlabel('EPOCH')
plt.ylabel('LOSS')
plt.legend()
plt.show()

# Plotting accuracy
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], 'b-', label='Training Accuracy')
plt.plot(history.history['val_accuracy'], 'c-', label='Validation Accuracy')
plt.title('MODEL ACCURACY')
plt.xlabel('EPOCH')
plt.ylabel('ACCURACY')
plt.legend()
plt.show()
```

```
Epoch 1/10
235/235 [==============================] - 5s 17ms/step - loss: 0.6972 - accuracy: 0.0012 - val_loss: 0.6970 - val_accuracy: 0.0014
Epoch 2/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6969 - accuracy: 0.0012 - val_loss: 0.6968 - val_accuracy: 0.0014
Epoch 3/10
235/235 [==============================] - 2s 11ms/step - loss: 0.6967 - accuracy: 0.0012 - val_loss: 0.6965 - val_accuracy: 0.0014
Epoch 4/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6964 - accuracy: 0.0012 - val_loss: 0.6963 - val_accuracy: 0.0014
Epoch 5/10
235/235 [==============================] - 4s 18ms/step - loss: 0.6962 - accuracy: 0.0012 - val_loss: 0.6960 - val_accuracy: 0.0014
Epoch 6/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6959 - accuracy: 0.0013 - val_loss: 0.6958 - val_accuracy: 0.0014
Epoch 7/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6957 - accuracy: 0.0013 - val_loss: 0.6956 - val_accuracy: 0.0015
Epoch 8/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6955 - accuracy: 0.0013 - val_loss: 0.6953 - val_accuracy: 0.0015
Epoch 9/10
235/235 [==============================] - 3s 14ms/step - loss: 0.6952 - accuracy: 0.0013 - val_loss: 0.6951 - val_accuracy: 0.0015
Epoch 10/10
235/235 [==============================] - 3s 15ms/step - loss: 0.6950 - accuracy: 0.0013 - val_loss: 0.6949 - val_accuracy: 0.0015
313/313 [==============================] - 1s 2ms/step
```







Github: https://github.com/SXP36810/BigData

Youtube: https://youtu.be/GA1j8ceG_Wg