Neural Networks & Deep Learning - ICP-9

CS 5720 (CRN 23216)

Student ID: 700745451

Student Name: Kamala Ramesh

Execution of the provided Autoencoder

```
[1] from keras.layers import Input, Dense
    from keras.models import Model
    from keras.datasets import mnist, fashion_mnist
    import numpy as np
```

```
[2] # 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='adam', loss='binary_crossentropy',metrics=['accuracy'])
```

```
[4] (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:])))
```

```
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 [==============================] - 1s 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
```

```
autoencoder.fit(x_train, x_train,
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 1/5
235/235 [==============================] - 7s 19ms/step - loss: 0.4139 - accuracy: 0.0086 - val_loss: 0.3431 - val_accuracy: 0.0127
Epoch 2/5
235/235 [==============================] - 4s 18ms/step - loss: 0.3285 - accuracy: 0.0138 - val_loss: 0.3206 - val_accuracy: 0.0128
Epoch 3/5
235/235 [==============================] - 3s 13ms/step - loss: 0.3120 - accuracy: 0.0137 - val_loss: 0.3083 - val_accuracy: 0.0148
Epoch 4/5
235/235 [==============================] - 3s 13ms/step - loss: 0.3020 - accuracy: 0.0162 - val_loss: 0.3002 - val_accuracy: 0.0164
Epoch 5/5
235/235 [==============================] - 3s 15ms/step - loss: 0.2954 - accuracy: 0.0184 - val_loss: 0.2953 - val_accuracy: 0.0185
<keras.callbacks.History at 0x7faa4e3f0a30>
```

1. To add one hidden layer to the existing code

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

```
[2]  # 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)

     # Define the hidden layer
     hidden = Dense(784, activation='relu')(encoded)

     # "decoded" is the lossy reconstruction of the input
     decoded = Dense(784, activation='sigmoid')(hidden)
```

```
[3]  # 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='adam', loss='binary_crossentropy',metrics=['accuracy'])
```

```
(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:])))
```

```
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
```

```
[5] history = autoencoder.fit(x_train, x_train,
                    epochs=5,
                    batch_size=256,
                    shuffle=True,
                    validation_data=(x_test, x_test))
```

```
Epoch 1/5
235/235 [==============================] - 11s 42ms/step - loss: 0.3535 - accuracy: 0.0125 - val_loss: 0.3054 - val_accuracy: 0.0168
Epoch 2/5
235/235 [==============================] - 9s 37ms/step - loss: 0.2957 - accuracy: 0.0155 - val_loss: 0.2923 - val_accuracy: 0.0201
Epoch 3/5
235/235 [==============================] - 8s 35ms/step - loss: 0.2870 - accuracy: 0.0200 - val_loss: 0.2867 - val_accuracy: 0.0227
Epoch 4/5
235/235 [==============================] - 9s 39ms/step - loss: 0.2826 - accuracy: 0.0236 - val_loss: 0.2833 - val_accuracy: 0.0235
Epoch 5/5
235/235 [==============================] - 10s 41ms/step - loss: 0.2798 - accuracy: 0.0274 - val_loss: 0.2808 - val_accuracy: 0.0290
```

2. Prediction on reconstructed data and visualize the reconstructed data

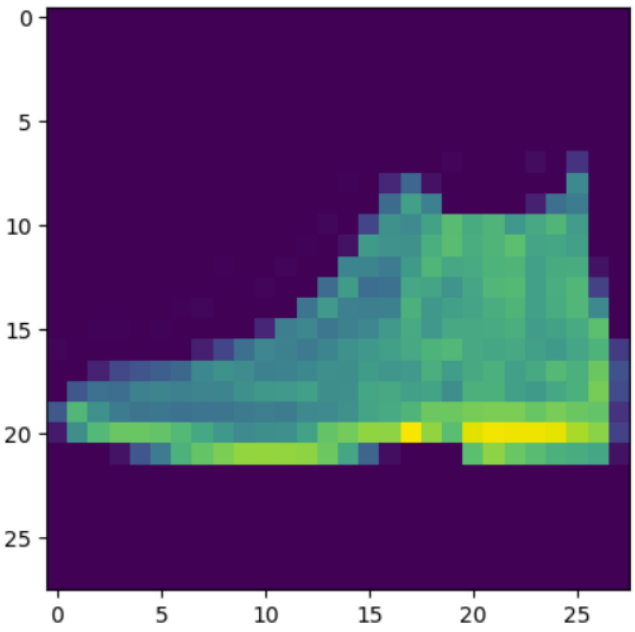
```
[6] reconstructed_data = autoencoder.predict(x_test)
```

```
313/313 [==============================] - 2s 5ms/step
```

Visualize the original data

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

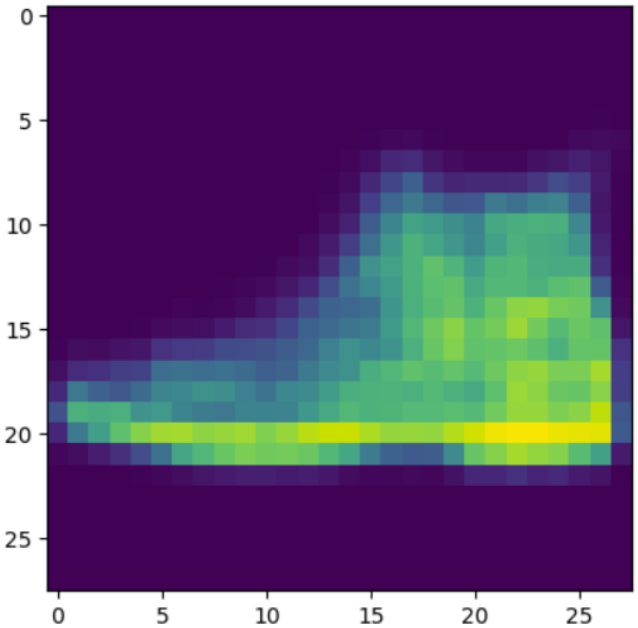
```
<matplotlib.image.AxesImage at 0x7fd3d41e6c70>
```



Visualize the reconstructed data

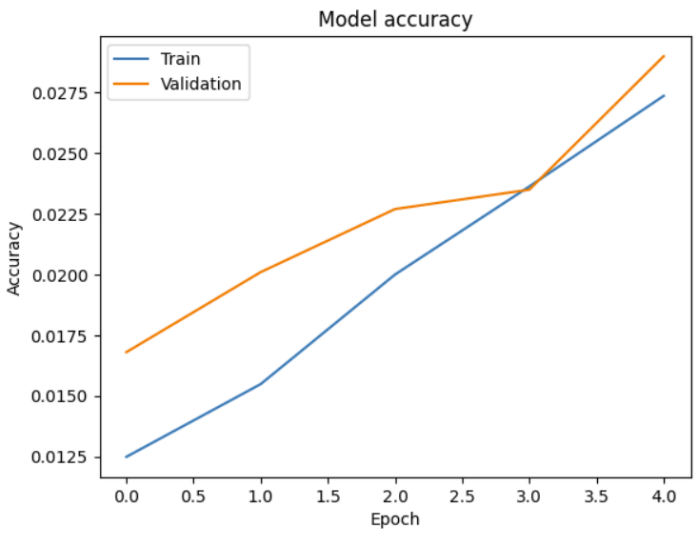
```
plt.imshow(reconstructed_data[0].reshape(28, 28))
```

```
<matplotlib.image.AxesImage at 0x7fd3d4ac7e20>
```

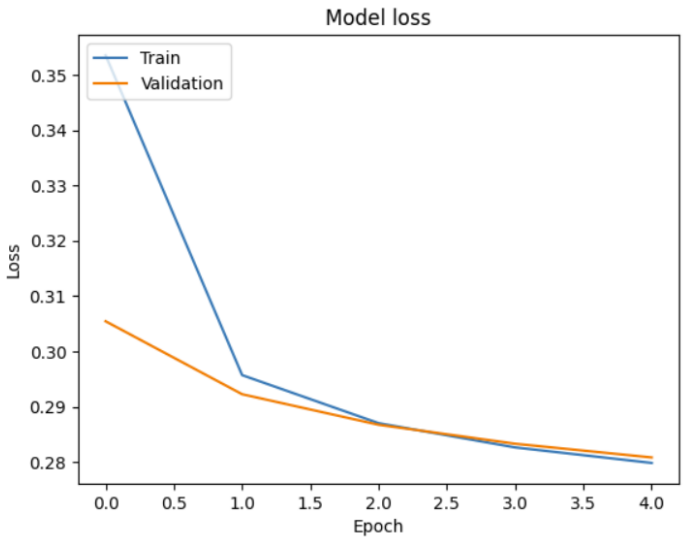


Accuracy and Loss Plot

```
# Plot the training and validation accuracy over epochs
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



```
# Plot the training and validation loss over epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



3. Repeating the above scenarios for Denoising Encoder

```
[1] 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
```

```
[2] # 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)

    # Define the hidden layer
    hidden = Dense(784, activation='relu')(encoded)

    # "decoded" is the lossy reconstruction of the input
    decoded = Dense(784, activation='sigmoid')(hidden)
```

```
[3] # 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='adam', loss='binary_crossentropy',metrics=['accuracy'])
```

```
(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 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)
```

```
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 [==============================] - 1s 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
```

```
history = autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=128,
                shuffle=True,
                validation_data=(x_test_noisy, x_test_noisy))
```

```
Epoch 1/10
469/469 [==============================] - 13s 25ms/step - loss: 0.3414 - accuracy: 0.0120 - val_loss: 0.2856 - val_accuracy: 0.0088
Epoch 2/10
469/469 [==============================] - 11s 24ms/step - loss: 0.3041 - accuracy: 0.0150 - val_loss: 0.2660 - val_accuracy: 0.0101
Epoch 3/10
469/469 [==============================] - 12s 25ms/step - loss: 0.2975 - accuracy: 0.0176 - val_loss: 0.2551 - val_accuracy: 0.0106
Epoch 4/10
469/469 [==============================] - 11s 23ms/step - loss: 0.2940 - accuracy: 0.0203 - val_loss: 0.2479 - val_accuracy: 0.0099
Epoch 5/10
469/469 [==============================] - 11s 23ms/step - loss: 0.2920 - accuracy: 0.0231 - val_loss: 0.2436 - val_accuracy: 0.0129
Epoch 6/10
469/469 [==============================] - 12s 26ms/step - loss: 0.2907 - accuracy: 0.0242 - val_loss: 0.2405 - val_accuracy: 0.0118
Epoch 7/10
469/469 [==============================] - 12s 26ms/step - loss: 0.2899 - accuracy: 0.0259 - val_loss: 0.2386 - val_accuracy: 0.0113
Epoch 8/10
469/469 [==============================] - 12s 25ms/step - loss: 0.2894 - accuracy: 0.0277 - val_loss: 0.2376 - val_accuracy: 0.0114
Epoch 9/10
469/469 [==============================] - 12s 25ms/step - loss: 0.2889 - accuracy: 0.0277 - val_loss: 0.2358 - val_accuracy: 0.0112
Epoch 10/10
469/469 [==============================] - 12s 25ms/step - loss: 0.2885 - accuracy: 0.0280 - val_loss: 0.2351 - val_accuracy: 0.0112
```

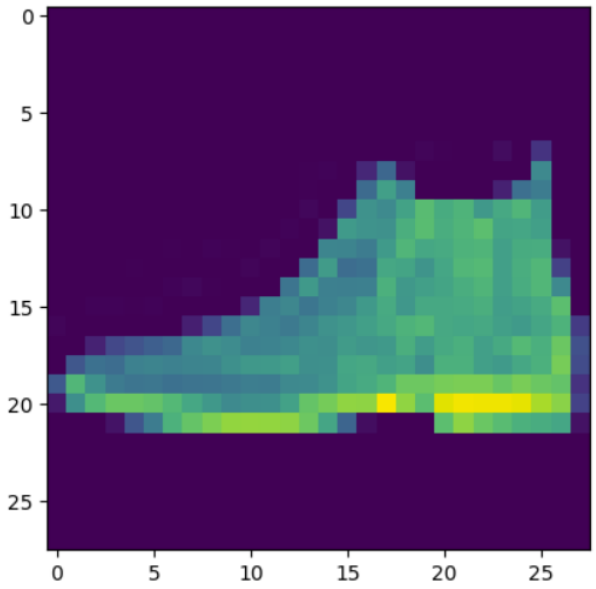
```
reconstructed_data = autoencoder.predict(x_test_noisy)
```

```
313/313 [==============================] - 1s 3ms/step
```

Visualize the original data

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

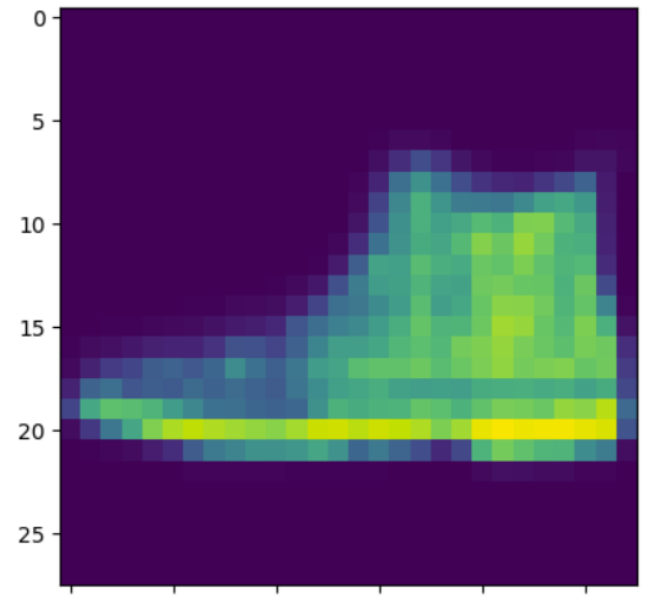
```
<matplotlib.image.AxesImage at 0x7f0b6a70a760>
```



Visualize reconstructed data

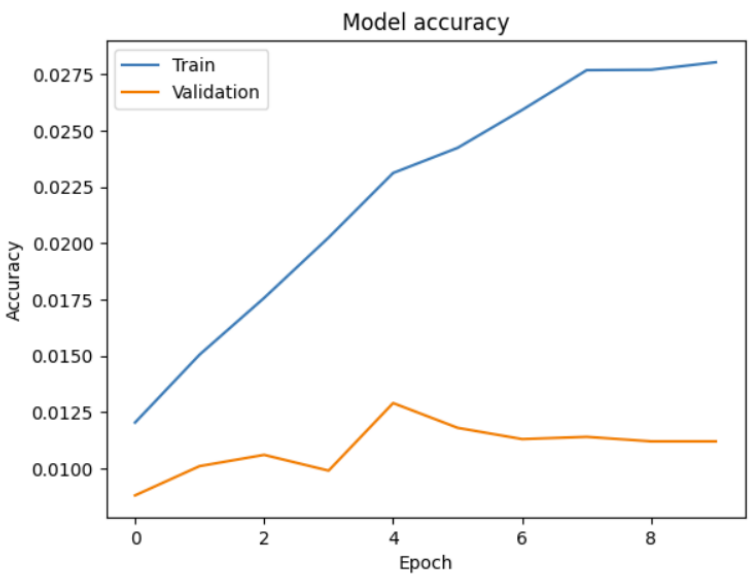
```
plt.imshow(reconstructed_data[0].reshape(28, 28))
```

```
<matplotlib.image.AxesImage at 0x7f0b6a60c730>
```



Accuracy and Loss plots

```
# Plot the training and validation accuracy over epochs
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



```
# Plot the training and validation loss over epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```