Spring 2024: CS5720 Neural Networks & Deep Learning - ICP-8

Sai Deva Pranay Kumar Rao Guddity (700745063)

Programming elements:

- Basics of Autoencoders
- Role of Autoencoders in unsupervised learning
- Types of Autoencoders
- Use case: Simple autoencoder-Reconstructing the existing image, which will contain most important
- features of the image
- Use case: Stacked autoencoder

In class programming:

- Add one more hidden layer to autoencoder
- Do the prediction on the test data and then visualize one of the reconstructed version of that test data.
- Also, visualize the same test data before reconstruction using Matplotlib
- Repeat the question 2 on the denoisening autoencoder
- plot loss and accuracy using the history object

Code:

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

encoding_dim = 32  # 32 floats -> compression factor 24.5, assuming the input is 784 floats

input_img = Input(shape=(784,))
encoded = Dense(encoding_dim, activation='relu')(input_img)
decoded = Dense(784, activation='sigmoid')(encoded)
autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adadelta', 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:])))
autoencoder.fit(x_train, x_train,
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

Output:

```
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 [==============================] - 6s 19ms/step - loss: 0.6937 - accuracy: 0.0011 - val_loss: 0.6936 - val_accuracy: 0.0014
Epoch 2/5
235/235 [==============================] - 3s 12ms/step - loss: 0.6935 - accuracy: 0.0011 - val_loss: 0.6935 - val_accuracy: 0.0014
Epoch 3/5
235/235 [==============================] - 2s 9ms/step - loss: 0.6934 - accuracy: 0.0011 - val_loss: 0.6933 - val_accuracy: 0.0015
Epoch 4/5
235/235 [==============================] - 2s 8ms/step - loss: 0.6932 - accuracy: 0.0011 - val_loss: 0.6931 - val_accuracy: 0.0015
Epoch 5/5
235/235 [==============================] - 2s 8ms/step - loss: 0.6931 - accuracy: 0.0011 - val_loss: 0.6930 - val_accuracy: 0.0015
<keras.src.callbacks.History at 0x7bb314340ee0>
```

Code:

```
from keras.layers import Input, Dense
from keras.models import Model
encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats
input_img = Input(shape=(784,))

encoded1 = Dense(128, activation='relu')(input_img)
encoded2 = Dense(encoding_dim, activation='relu')(encoded1)

decoded1 = Dense(128, activation='relu')(encoded2)
decoded2 = Dense(784, activation='sigmoid')(decoded1)
autoencoder = Model(input_img, decoded2)
```

```
encoder = Model(input_img, encoded2)
encoded_input = Input(shape=(encoding_dim,))
decoder_layer1 = autoencoder.layers[-2]
decoder_layer2 = autoencoder.layers[-1]
decoder = Model(encoded_input, decoder_layer2(decoder_layer1(encoded_input)))
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy',metrics ='accuracy')

from keras.datasets import mnist, fashion_mnist
import numpy as np
(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:])))
autoencoder.fit(x_train, x_train,
 epochs=5,
 batch_size=256,
shuffle=True,
validation_data=(x_test, x_test))
```

Output:

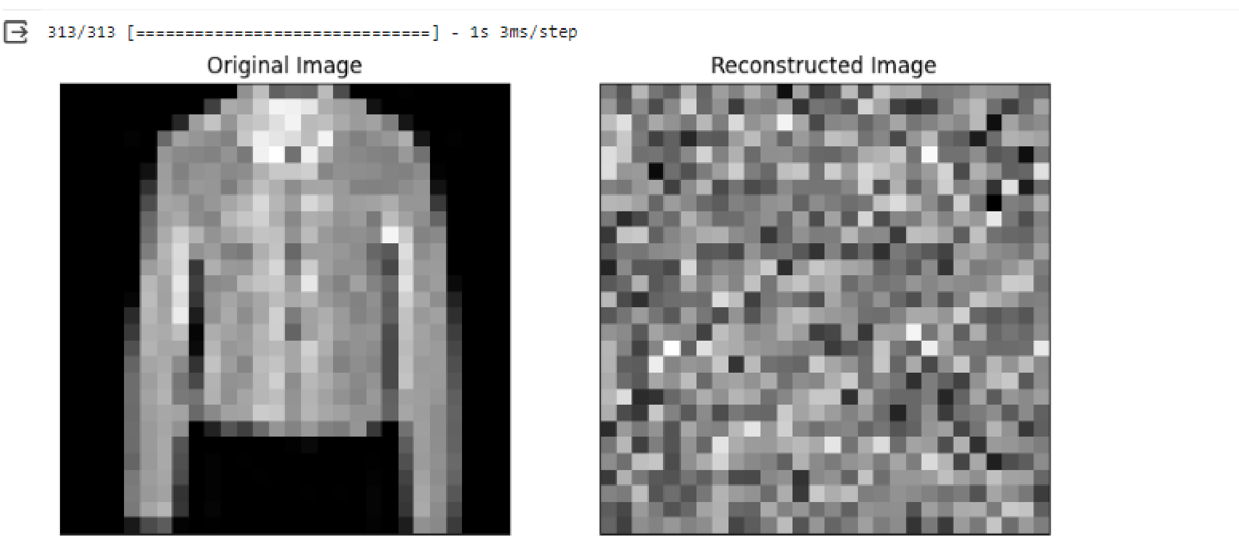
```
Epoch 1/5
235/235 [==============================] - 3s 10ms/step - loss: 0.6930 - accuracy: 0.0016 - val_loss: 0.6929 - val_accuracy: 0.0018
Epoch 2/5
235/235 [==============================] - 2s 9ms/step - loss: 0.6928 - accuracy: 0.0015 - val_loss: 0.6927 - val_accuracy: 0.0016
Epoch 3/5
235/235 [==============================] - 2s 10ms/step - loss: 0.6927 - accuracy: 0.0015 - val_loss: 0.6926 - val_accuracy: 0.0015
Epoch 4/5
235/235 [==============================] - 3s 11ms/step - loss: 0.6925 - accuracy: 0.0015 - val_loss: 0.6924 - val_accuracy: 0.0015
Epoch 5/5
235/235 [==============================] - 2s 9ms/step - loss: 0.6923 - accuracy: 0.0015 - val_loss: 0.6922 - val_accuracy: 0.0016
<keras.src.callbacks.History at 0x7bb30c2a9900>
```

Code:

```
import matplotlib.pyplot as plt
reconstructed_imgs = autoencoder.predict(x_test)
```

```
n = 10  # index of the image to be plotted
plt.figure(figsize=(10, 5))
ax = plt.subplot(1, 2, 1)
plt.imshow(x_test[n].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
ax.set_title("Original Image")
ax = plt.subplot(1, 2, 2)
plt.imshow(reconstructed_imgs[n].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
ax.set_title("Reconstructed Image")
plt.show()
```

Output:



Code:

```
from keras.layers import Input, Dense
from keras.models import Model

encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats
input_img = Input(shape=(784,))
encoded = Dense(encoding_dim, activation='relu')(input_img)
decoded = Dense(784, activation='sigmoid')(encoded)
autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy',metrics ='accuracy')
from keras.datasets import fashion_mnist
import numpy as np
(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:])))
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)
autoencoder.fit(x_train_noisy, x_train,
        epochs=10,
        batch_size=256,
        shuffle=True,
        validation_data=(x_test_noisy, x_test_noisy))
```

Output:

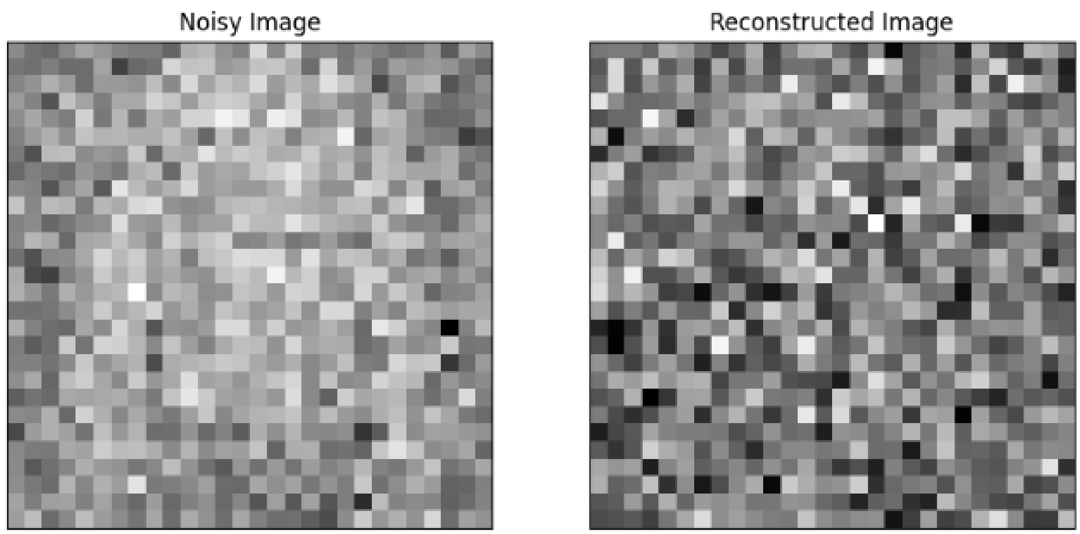
```
Epoch 1/10
235/235 [==============================] - 2s 7ms/step - loss: 0.6959 - accuracy: 0.0017 - val_loss: 0.6957 - val_accuracy: 0.0014
Epoch 2/10
235/235 [==============================] - 2s 8ms/step - loss: 0.6957 - accuracy: 0.0017 - val_loss: 0.6955 - val_accuracy: 0.0013
Epoch 3/10
235/235 [==============================] - 1s 6ms/step - loss: 0.6954 - accuracy: 0.0016 - val_loss: 0.6953 - val_accuracy: 0.0013
Epoch 4/10
235/235 [==============================] - 1s 6ms/step - loss: 0.6952 - accuracy: 0.0016 - val_loss: 0.6951 - val_accuracy: 0.0013
Epoch 5/10
235/235 [==============================] - 1s 6ms/step - loss: 0.6950 - accuracy: 0.0017 - val_loss: 0.6949 - val_accuracy: 0.0013
Epoch 6/10
235/235 [==============================] - 2s 9ms/step - loss: 0.6948 - accuracy: 0.0017 - val_loss: 0.6947 - val_accuracy: 0.0013
Epoch 7/10
235/235 [==============================] - 1s 5ms/step - loss: 0.6946 - accuracy: 0.0017 - val_loss: 0.6945 - val_accuracy: 0.0013
Epoch 8/10
235/235 [==============================] - 1s 6ms/step - loss: 0.6944 - accuracy: 0.0017 - val_loss: 0.6943 - val_accuracy: 0.0013
Epoch 9/10
235/235 [==============================] - 2s 7ms/step - loss: 0.6942 - accuracy: 0.0017 - val_loss: 0.6941 - val_accuracy: 0.0012
Epoch 10/10
235/235 [==============================] - 1s 6ms/step - loss: 0.6940 - accuracy: 0.0017 - val_loss: 0.6939 - val_accuracy: 0.0012
<keras.src.callbacks.History at 0x7bb30f5a8250>
```

<u>Code:</u>

```
import matplotlib.pyplot as plt


reconstructed_imgs = autoencoder.predict(x_test_noisy)
n = 10  # index of the image to be plotted
plt.figure(figsize=(10, 5))


ax = plt.subplot(1, 2, 1)
plt.imshow(x_test_noisy[n].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
ax.set_title("Noisy Image")
ax = plt.subplot(1, 2, 2)
plt.imshow(reconstructed_imgs[n].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
ax.set_title("Reconstructed Image")


plt.show()
```

Output:

```
313/313 [==============================] - 1s 2ms/step
```



Code:

```
import matplotlib.pyplot as plt
history = autoencoder.fit(x_train_noisy, x_train,
            epochs=10,
            batch_size=256,
            shuffle=True,
            validation_data=(x_test_noisy, x_test_noisy))
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
```

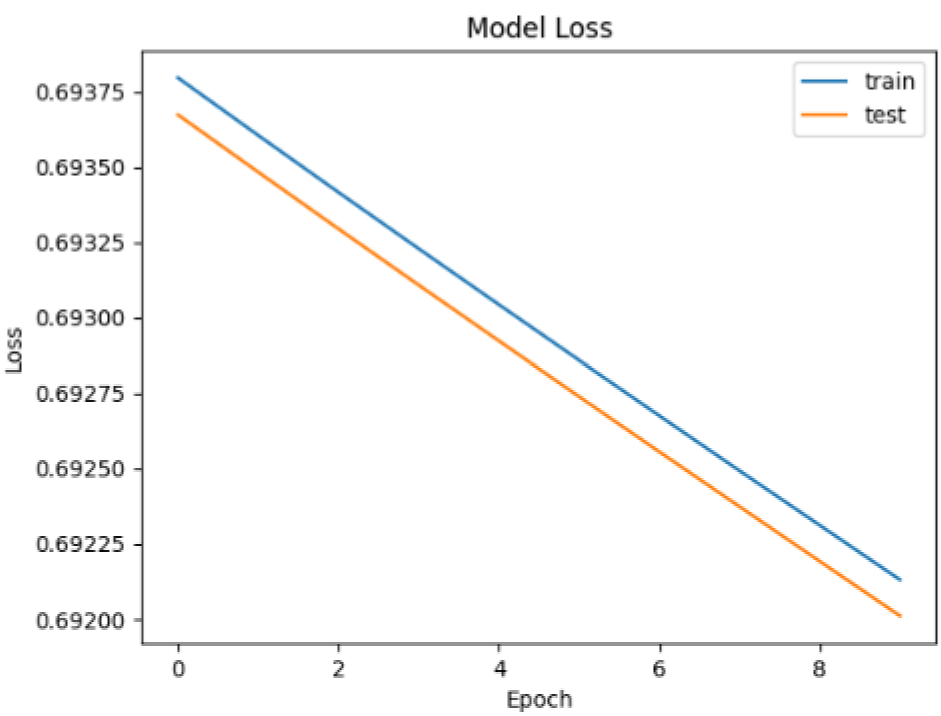
plt.xlabel('Epoch')

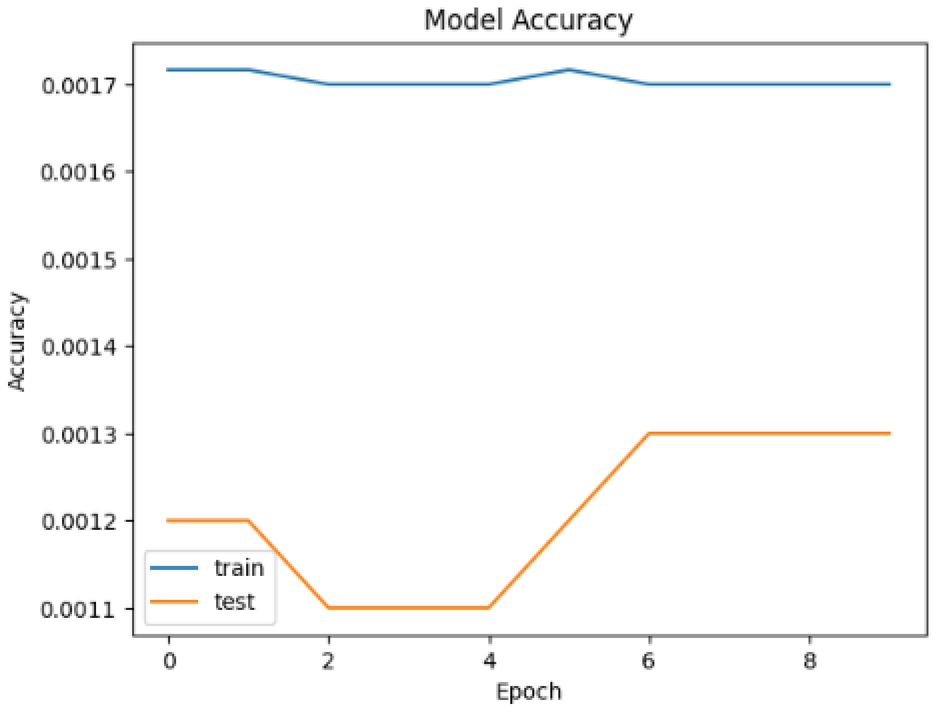
plt.legend()

plt.show()

<u>Output:</u>

```
Epoch 1/10
235/235 [==============================] - 2s 8ms/step - loss: 0.6938 - accuracy: 0.0017 - val_loss: 0.6937 - val_accuracy: 0.0012
Epoch 2/10
235/235 [==============================] - 1s 6ms/step - loss: 0.6936 - accuracy: 0.0017 - val_loss: 0.6935 - val_accuracy: 0.0012
Epoch 3/10
235/235 [==============================] - 1s 6ms/step - loss: 0.6934 - accuracy: 0.0017 - val_loss: 0.6933 - val_accuracy: 0.0011
Epoch 4/10
235/235 [==============================] - 1s 6ms/step - loss: 0.6932 - accuracy: 0.0017 - val_loss: 0.6931 - val_accuracy: 0.0011
Epoch 5/10
235/235 [==============================] - 1s 6ms/step - loss: 0.6930 - accuracy: 0.0017 - val_loss: 0.6929 - val_accuracy: 0.0011
Epoch 6/10
235/235 [==============================] - 1s 6ms/step - loss: 0.6929 - accuracy: 0.0017 - val_loss: 0.6927 - val_accuracy: 0.0012
Epoch 7/10
235/235 [==============================] - 1s 6ms/step - loss: 0.6927 - accuracy: 0.0017 - val_loss: 0.6926 - val_accuracy: 0.0013
Epoch 8/10
235/235 [==============================] - 1s 6ms/step - loss: 0.6925 - accuracy: 0.0017 - val_loss: 0.6924 - val_accuracy: 0.0013
Epoch 9/10
235/235 [==============================] - 3s 15ms/step - loss: 0.6923 - accuracy: 0.0017 - val_loss: 0.6922 - val_accuracy: 0.0013
Epoch 10/10
235/235 [==============================] - 2s 7ms/step - loss: 0.6921 - accuracy: 0.0017 - val_loss: 0.6920 - val_accuracy: 0.0013
```





Model Accuracy
0.0017
0.0016
0.0015
0.0014
0.0013
0.0012
0.0011
Accuracy
train
test
0
2
4
6
8
Epoch