NEURAL NETWORK & DEEP LEARNING
ASSIGNMENT 8

Name : SRAVYA REDDY PILLI
Student ID : 700747154

Git hub Link: https://github.com/09sravyareddy/NNDL-ICP9

Video link: https://drive.google.com/file/d/1xXI6NnNW9xtuTiJPJT8goSDnoG_TwcmX/view?usp=sharing

ICP-9.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:58 PM

+ Code + Text

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

# 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')
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,
```

+ Code + Text

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

```
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 [==============================] - 9s 30ms/step - loss: 0.6949 - val_loss: 0.6948
Epoch 2/5
235/235 [==============================] - 4s 19ms/step - loss: 0.6947 - val_loss: 0.6946
Epoch 3/5
235/235 [==============================] - 3s 14ms/step - loss: 0.6945 - val_loss: 0.6944
Epoch 4/5
235/235 [==============================] - 3s 15ms/step - loss: 0.6943 - val_loss: 0.6942
Epoch 5/5
235/235 [==============================] - 2s 11ms/step - loss: 0.6941 - val_loss: 0.6940
<keras.src.callbacks.History at 0x7a843dd3e6e0>
```

Added a new hidden layer to the encoder and the decoder

+ Code + Text

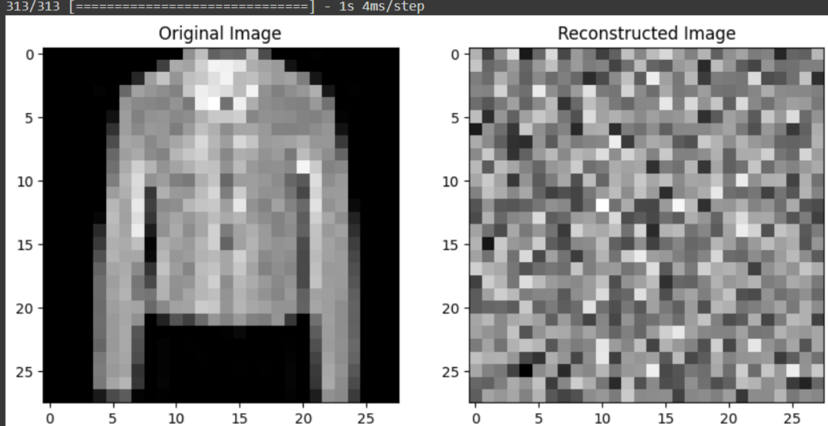
```
from keras.layers import Input, Dense
from keras.models import Model

# Define input shape
input_shape = (784,)

# Define encoding dimensions
encoding_dim1 = 64
encoding_dim2 = 32

# Define input layer
input_img = Input(shape=input_shape)

encoded1 = Dense(encoding_dim1, activation='relu')(input_img)
encoded2 = Dense(encoding_dim2, activation='relu')(encoded1)
decoded1 = Dense(encoding_dim1, activation='relu')(encoded2)
decoded2 = Dense(input_shape[0], activation='sigmoid')(decoded1)
autoencoder = Model(input_img, decoded2)
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:])))
```

+ Code + Text

```python
# Train model
history = autoencoder.fit(x_train, x_train,
                          epochs=20,
                          batch_size=256,
                          shuffle=True,
                          validation_data=(x_test, x_test))

# Predict on test data
decoded_imgs = autoencoder.predict(x_test)

# Visualize reconstructed image and original image
import matplotlib.pyplot as plt
# Choose an index of a test image to visualize
idx = 10

# Reshape the test image
test_img = x_test[idx].reshape(28, 28)

# Reshape the reconstructed image
reconstructed_img = decoded_imgs[idx].reshape(28, 28)

# Plot the original and reconstructed images side by side
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(test_img, cmap='gray')
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(reconstructed_img, cmap='gray')
plt.title('Reconstructed Image')
plt.show()
```

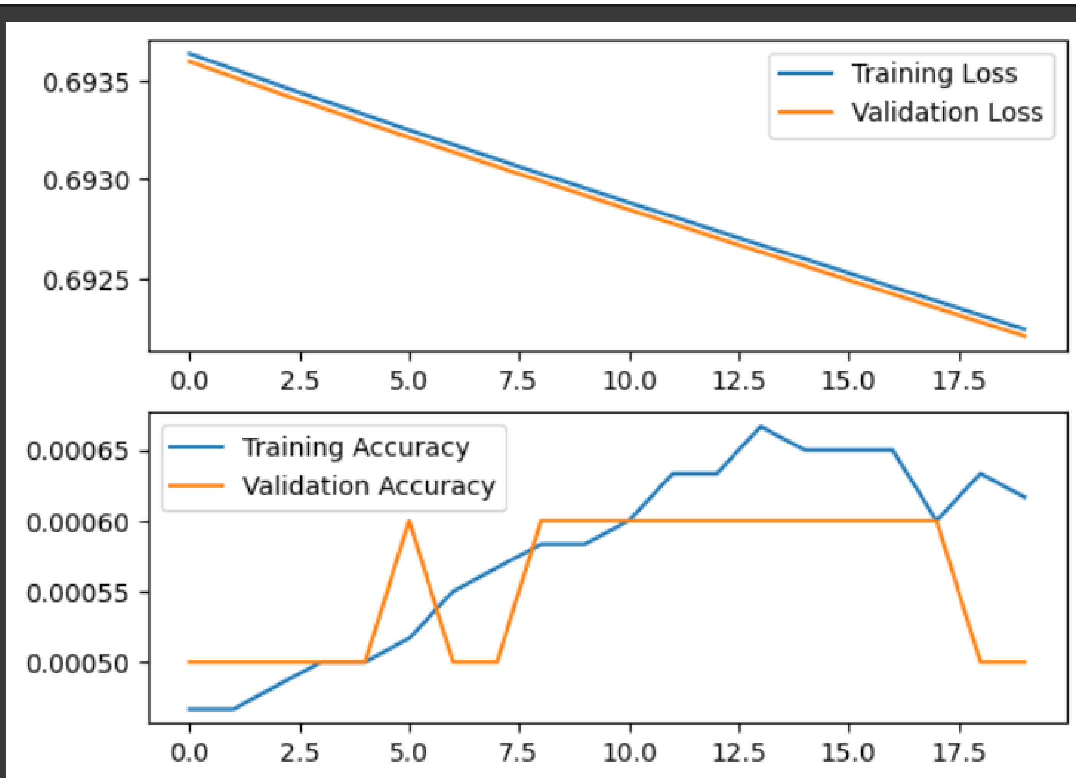
+ Code + Text

```
Epoch 18/20
235/235 [==============================] - 4s 15ms/step - loss: 0.6924 - accuracy: 6.0000e-04 - val_loss: 0.6923 - val_accuracy: 6.0000e-04
Epoch 19/20
235/235 [==============================] - 3s 13ms/step - loss: 0.6923 - accuracy: 6.3333e-04 - val_loss: 0.6923 - val_accuracy: 5.0000e-04
Epoch 20/20
235/235 [==============================] - 3s 13ms/step - loss: 0.6922 - accuracy: 6.1667e-04 - val_loss: 0.6922 - val_accuracy: 5.0000e-04
313/313 [==============================] - 1s 4ms/step
```



Calculated the loss and accuracy of the model using the history object, the compile metrics are set to accuracy

```python
# Plot the loss and accuracy over epochs
plt.subplot(2, 1, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()

plt.show()
```



Adding the noise to denoise autoencoder

+ Code + Text

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

# 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'])

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:])))

#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)
```

+ Code + Text

```python
#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)

history=autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test_noisy, x_test_noisy))
```

```
Epoch 1/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6961 - accuracy: 0.0033 - val_loss: 0.6960 - val_accuracy: 0.0018
Epoch 2/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6958 - accuracy: 0.0034 - val_loss: 0.6958 - val_accuracy: 0.0019
Epoch 3/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6956 - accuracy: 0.0035 - val_loss: 0.6955 - val_accuracy: 0.0019
Epoch 4/10
235/235 [==============================] - 3s 15ms/step - loss: 0.6953 - accuracy: 0.0036 - val_loss: 0.6953 - val_accuracy: 0.0020
Epoch 5/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6951 - accuracy: 0.0036 - val_loss: 0.6951 - val_accuracy: 0.0021
Epoch 6/10
235/235 [==============================] - 4s 17ms/step - loss: 0.6949 - accuracy: 0.0037 - val_loss: 0.6949 - val_accuracy: 0.0021
Epoch 7/10
235/235 [==============================] - 5s 23ms/step - loss: 0.6946 - accuracy: 0.0037 - val_loss: 0.6946 - val_accuracy: 0.0020
Epoch 8/10
235/235 [==============================] - 5s 23ms/step - loss: 0.6944 - accuracy: 0.0038 - val_loss: 0.6944 - val_accuracy: 0.0020
Epoch 9/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6942 - accuracy: 0.0039 - val_loss: 0.6942 - val_accuracy: 0.0021
```

Plotted the original image and the reconstructed image using the matplotlib library

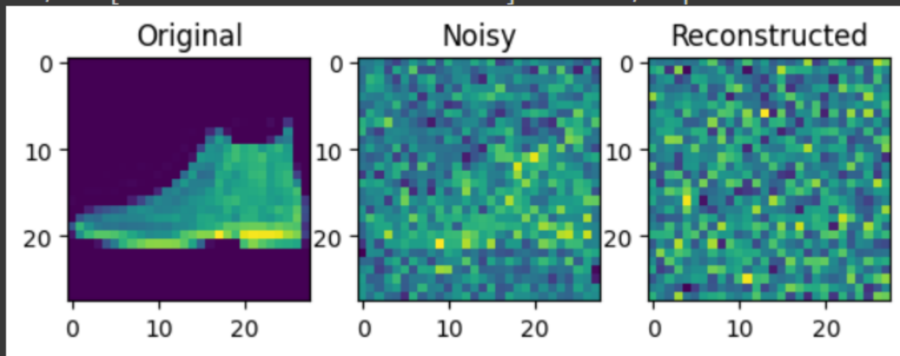
+ Code + Text

```
Epoch 8/10
235/235 [==============================] - 5s 23ms/step - loss: 0.6944 - accuracy: 0.0038 - val_loss: 0.6944 - val_accuracy: 0.0020
Epoch 9/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6942 - accuracy: 0.0039 - val_loss: 0.6942 - val_accuracy: 0.0021
Epoch 10/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6940 - accuracy: 0.0040 - val_loss: 0.6940 - val_accuracy: 0.0020
```

```python
import matplotlib.pyplot as plt

# Get the reconstructed images
reconstructed_imgs = autoencoder.predict(x_test_noisy)

# Select one image to display
img_to_display = 0

# Display the original, noisy, and reconstructed images side by side
plt.subplot(1, 3, 1)
plt.imshow(x_test[img_to_display].reshape(28, 28))
plt.title('Original')

plt.subplot(1, 3, 2)
plt.imshow(x_test_noisy[img_to_display].reshape(28, 28))
plt.title('Noisy')

plt.subplot(1, 3, 3)
plt.imshow(reconstructed_imgs[img_to_display].reshape(28, 28))
plt.title('Reconstructed')

plt.show()
```

+ Code + Text

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

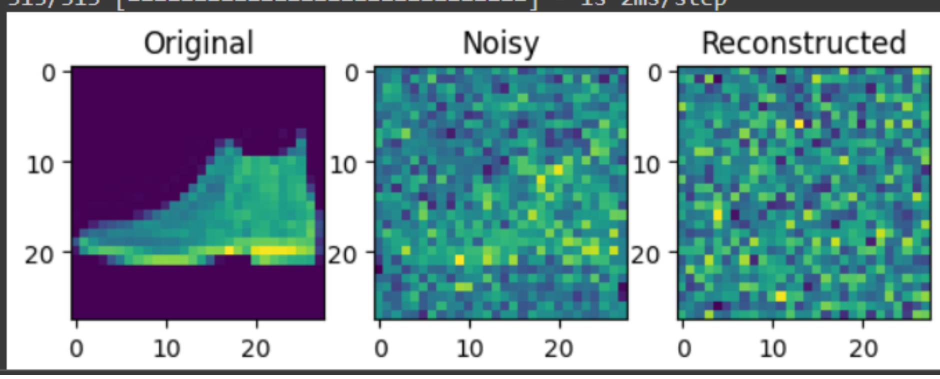



```python
import matplotlib.pyplot as plt

# Get the reconstructed images
reconstructed_imgs = autoencoder.predict(x_test_noisy)

# Select one image to display
img_to_display = 0

# Display the original, noisy, and reconstructed images side by side
plt.subplot(1, 3, 1)
plt.imshow(x_test[img_to_display].reshape(28, 28))
plt.title('Original')

plt.subplot(1, 3, 2)
plt.imshow(x_test_noisy[img_to_display].reshape(28, 28))
plt.title('Noisy')
```

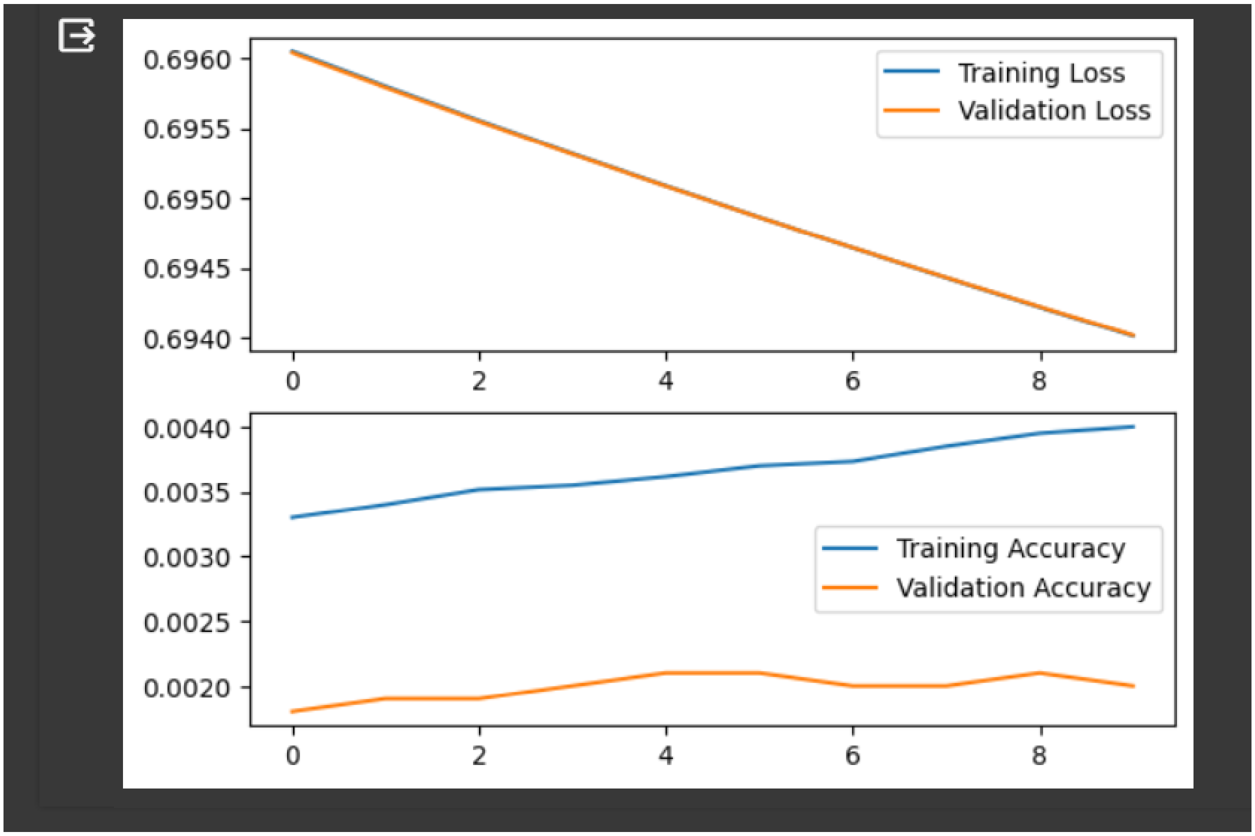
+ Code + Text

```
# Select one image to display
img_to_display = 00

# Display the original, noisy, and reconstructed images side by side
plt.subplot(1, 3, 1)
plt.imshow(x_test[img_to_display].reshape(28, 28))
plt.title('Original')

plt.subplot(1, 3, 2)
plt.imshow(x_test_noisy[img_to_display].reshape(28, 28))
plt.title('Noisy')

plt.subplot(1, 3, 3)
plt.imshow(reconstructed_imgs[img_to_display].reshape(28, 28))
plt.title('Reconstructed')

plt.show()
```

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




```
# Plot the loss and accuracy over epochs
plt.subplot(2, 1, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()

plt.show()
```


Training Loss
Validation Loss
0.6960
0.6955
0.6950
0.6945
0.6940
0
2
4
6
8
Training Accuracy
Validation Accuracy
0.0040
0.0035
0.0030
0.0025
0.0020
0
2
4
6
8