

# Summer 2024: CS5720

## Neural Networks and Deep Learning - ICP-5

Github Link : <https://github.com/gouthamthogaru/ICP-5>

### 1. Add one more hidden layer to autoencoder

CODE :

```
+ Code + Text Connecting Gemini ^
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist, fashion_mnist
import numpy as np

# this is the size of our encoded representations
encoding_dim = 32
# 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)

# Adding an additional hidden layer
hidden_layer_dim = 64
hidden_layer = Dense(hidden_layer_dim, activation='relu')(encoded)

# "decoded" is the lossy reconstruction of the input, now connected to the hidden layer instead of 'encoded'
decoded = Dense(784, activation='sigmoid')(hidden_layer)

# 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')

# Load and prepare the data
(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:])))
```

### Explanation :

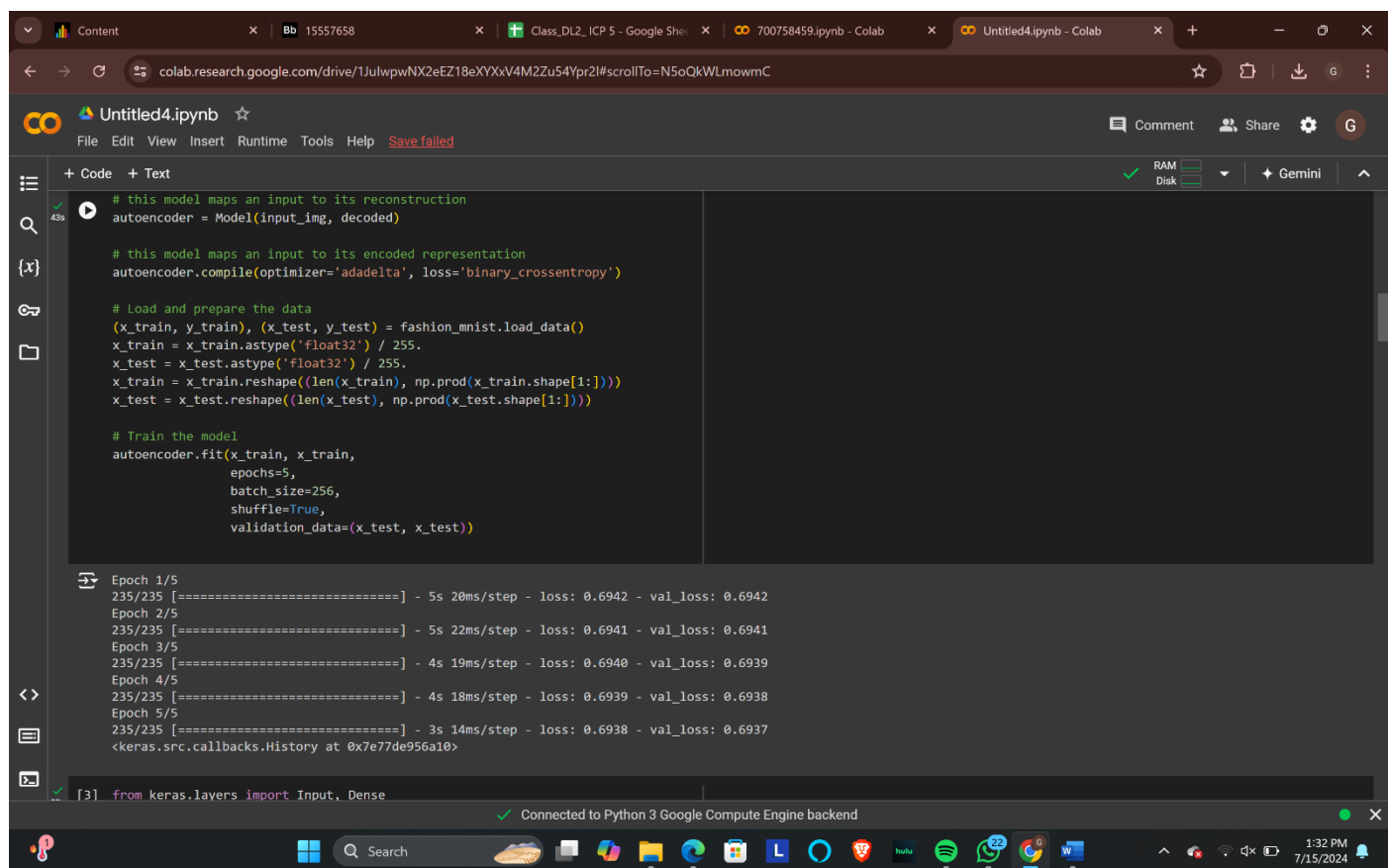
1. Autoencoder Architecture: The network is structured with an input layer, an encoded layer to compress the input, an additional hidden layer for potentially capturing more complex patterns, and a decoded layer to reconstruct the input from its encoded representation.
2. Encoding Dimension: The encoded layer compresses the input to 32 floating point numbers, significantly reducing the dimensionality from the original 784 floats (assuming the input images are 28x28 pixels, flattened to 784 floats for a fully connected network).
3. Hidden Layer: An additional hidden layer with 64 units is introduced after the encoding layer. This can help in learning more complex representations and aid in the decoding process. The activation function for both the encoded layer and the hidden layer is ReLU (Rectified Linear Unit), which introduces non-linearity into the model.
4. Loss Function and Optimizer: The model uses binary crossentropy as the loss function, which is common for reconstruction tasks, and the 'adadelta' optimizer, which is an adaptive learning rate method.

5. Data Preparation: The Fashion MNIST dataset is loaded, normalized to have pixel values between 0 and 1 (by dividing by 255), and reshaped to fit the model's input requirements. The dataset consists of 28x28 pixel grayscale images of clothing items, which are flattened to 784 dimensions to match the input layer of the model.

6. Model Training: The autoencoder is trained on the Fashion MNIST dataset for 5 epochs with a batch size of 256, using both the training and validation datasets. Here, the model learns to compress (encode) the input data and then reconstruct (decode) it as closely as possible to the original input.

7. Purpose and Application: Autoencoders like this one are used for dimensionality reduction, feature learning, and denoising images. By learning to reconstruct the input data from a compressed representation, the model can discover important features and patterns in the data.

## Output :



```
# 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')

# Load and prepare the data
(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:])))

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

```
Epoch 1/5
235/235 [=====] - 5s 20ms/step - loss: 0.6942 - val_loss: 0.6942
Epoch 2/5
235/235 [=====] - 5s 22ms/step - loss: 0.6941 - val_loss: 0.6941
Epoch 3/5
235/235 [=====] - 4s 19ms/step - loss: 0.6940 - val_loss: 0.6939
Epoch 4/5
235/235 [=====] - 4s 18ms/step - loss: 0.6939 - val_loss: 0.6938
Epoch 5/5
235/235 [=====] - 3s 14ms/step - loss: 0.6938 - val_loss: 0.6937
<keras.src.callbacks.History at 0x7e77de956a10>
```

```
[3]: from keras.layers import Input, Dense
```

Connected to Python 3 Google Compute Engine backend

2.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**

**Code :**

```
+ Code + Text
23s
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

# Define the model architecture
encoding_dim = 32
hidden_layer_dim = 64

input_img = Input(shape=(784,))
encoded = Dense(encoding_dim, activation='relu')(input_img)
hidden_layer = Dense(hidden_layer_dim, activation='relu')(encoded) # Additional hidden layer
decoded = Dense(784, activation='sigmoid')(hidden_layer)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')

# Load and prepare 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:])))

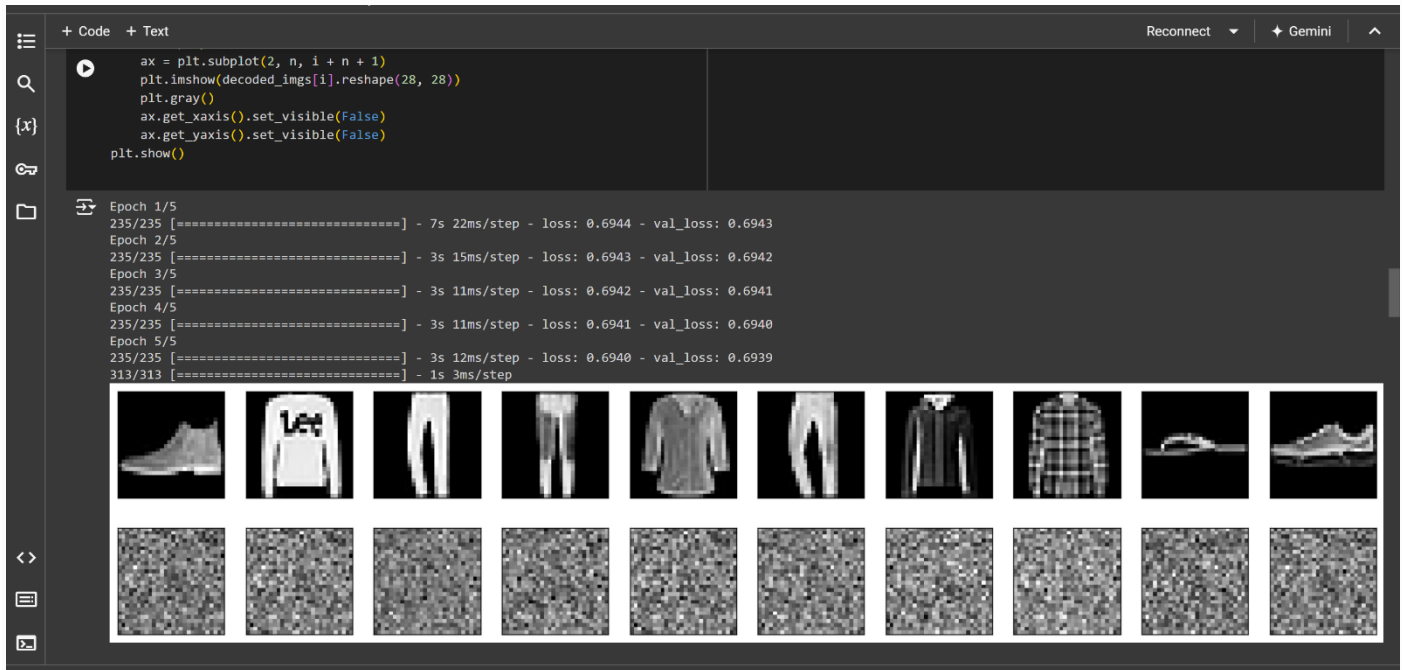
# Train the model
autoencoder.fit(x_train, x_train,
               epochs=5,
               batch_size=256,
               shuffle=True,
               validation_data=(x_test, x_test))

# Predict on the test data
```

**Explanation :**

1. Model Architecture: The model consists of an input layer that accepts flattened 28x28 grayscale images (784 pixels), an encoding layer to compress the input to a 32-dimensional representation, an additional hidden layer with 64 units for more complex feature extraction, and a decoding layer to reconstruct the original image from the compressed form.
2. Compilation: The autoencoder is compiled with the Adadelta optimizer and binary crossentropy loss, a common setup for autoencoders since the task is to reproduce the input image as closely as possible.
3. Data Preparation: The Fashion MNIST dataset, comprising 28x28 grayscale images of clothing items, is loaded, normalized (pixel values scaled between 0 and 1), and reshaped to fit the model.
4. Training: The model is trained on the Fashion MNIST training data for 5 epochs, using a batch size of 256. Validation is performed using the test set.
5. Visualization: After training, the model predicts on the test set, and a comparison between original images and their reconstructed counterparts is visualized for 10 example cases. This step helps to visually assess the model's performance in terms of how well it can reconstruct the input images after compression and decompression.

## Output :



### 3. Repeat the question 2 on the denoising autoencoder

Code :

```
+ Code + Text Connecting Gemini ^
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

# Define the model architecture
encoding_dim = 32

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')

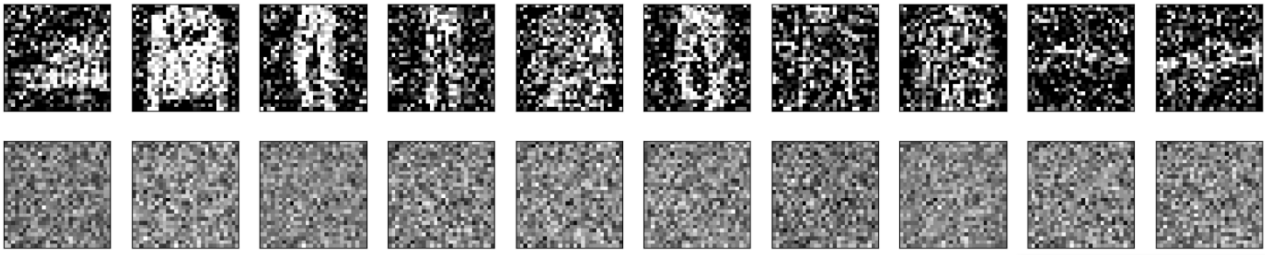
# Load data
(x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

# Introducing 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)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)

# Train the model
autoencoder.fit(x_train_noisy, x_train,
                epochs=20,
                batch_size=256,
                shuffle=True,
```

Output:

```
+ Code + Text Connecting Gemini ^
235/235 [=====] - 2s 11ms/step - loss: 0.6933 - val_loss: 0.6934
Epoch 12/20
235/235 [=====] - 3s 12ms/step - loss: 0.6933 - val_loss: 0.6933
Epoch 13/20
235/235 [=====] - 3s 14ms/step - loss: 0.6932 - val_loss: 0.6931
Epoch 14/20
235/235 [=====] - 2s 10ms/step - loss: 0.6931 - val_loss: 0.6930
Epoch 15/20
235/235 [=====] - 2s 10ms/step - loss: 0.6929 - val_loss: 0.6929
Epoch 16/20
235/235 [=====] - 2s 10ms/step - loss: 0.6928 - val_loss: 0.6928
Epoch 17/20
235/235 [=====] - 3s 11ms/step - loss: 0.6927 - val_loss: 0.6926
Epoch 18/20
235/235 [=====] - 4s 15ms/step - loss: 0.6926 - val_loss: 0.6925
Epoch 19/20
235/235 [=====] - 2s 10ms/step - loss: 0.6924 - val_loss: 0.6924
Epoch 20/20
235/235 [=====] - 3s 11ms/step - loss: 0.6923 - val_loss: 0.6923
313/313 [=====] - 1s 2ms/step
```



## 4. plot loss and accuracy using the history object

Code :

```
+ Code + Text
Reconnect Gemini ^
↑ ↓ ↻ 🗨 ⚙ 📄 🗑 ⋮

from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import fashion_mnist
from keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
from keras.optimizers import Adam

# Load and prepare the Fashion MNIST data
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train = x_train.reshape(-1, 784).astype('float32') / 255
x_test = x_test.reshape(-1, 784).astype('float32') / 255

# Convert labels to one-hot encoding
num_classes = 10
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

# Model architecture
input_img = Input(shape=(784,))
encoded = Dense(128, activation='relu')(input_img)
decoded = Dense(10, activation='softmax')(encoded) # Classification layer

model = Model(input_img, decoded)
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=256,
                    shuffle=True,
```

Output :

