Denoising_Autoencoders_and_Deep_Neural_Networks

April 8, 2020

1 Denoising Autoencoder

[0]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Flatten, Dropout, Dense, Conv2D,␣
↪MaxPooling2D, UpSampling2D
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist
from matplotlib import pyplot as plt
```

1.0.1 Let's define the Denoising Autoencoder.

[2]:
```
# Input layer and add noise
input_img = Input(shape=(28, 28, 1))  # adapt this if using `channels_first`␣
↪image data format
noisy_img = Dropout(0.2)(input_img)

# Encoder layers
x = Conv2D(32, (3, 3), activation='relu', padding='same')(noisy_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (8, 8, 16)
encoding_dim = (8, 8, 16)

# Decoder layers
x = Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu')(x)
x = UpSampling2D((2, 2))(x)
```

```python
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

# Build and compile the model
autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer=tf.keras.optimizers.Adam(),
                    loss=tf.keras.losses.BinaryCrossentropy(),
                    metrics=["accuracy"])
autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
dropout (Dropout)            (None, 28, 28, 1)         0
_________________________________________________________________
conv2d (Conv2D)              (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 16)        4624
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 16)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 16)          2320
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 16)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 16)          2320
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 8, 8, 16)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 16)          2320
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 32)        4640
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 1)         289
=================================================================
Total params: 16,833
Trainable params: 16,833
Non-trainable params: 0
_________________________________________________________________
```

1.0.2 Now define the Encoder by selecting the appropiate layers of the Autoencoder.

[0]:
```
# this model maps an input to its encoded representation
input_img = Input(shape=(28, 28, 1))
x = input_img

# retrieve the layers of the encoder model
encoder_layers = autoencoder.layers[2:8]
for layer in encoder_layers:
  x = layer(x)

encoded = x

# create the encoder model
encoder = Model(input_img, encoded)
```

1.0.3 Same thing for the Decoder.

[0]:
```
# this model maps an encoded input to its decoded image
encoded_input = Input(shape=encoding_dim)
x = encoded_input

# retrieve the layers of the encoder model
decoder_layers = autoencoder.layers[8:]
for layer in decoder_layers:
  x = layer(x)

decoded = x

# create the encoder model
decoder = Model(encoded_input, x)
```

1.0.4 Read the MNIST dataset and normalize values.

[0]:
```
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))  # adapt this if using␣
↪`channels_first` image data format
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))  # adapt this if using␣
↪`channels_first` image data format
```

1.0.5 Train the Denoising Autoencoder.

[6]:
```
autoencoder.fit(x_train, x_train,
                epochs=10,
                batch_size=128,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 1/10
469/469 [==============================] - 3s 6ms/step - loss: 0.1781 -
accuracy: 0.8013 - val_loss: 0.1265 - val_accuracy: 0.8049
Epoch 2/10
469/469 [==============================] - 3s 6ms/step - loss: 0.1147 -
accuracy: 0.8095 - val_loss: 0.1106 - val_accuracy: 0.8106
Epoch 3/10
469/469 [==============================] - 3s 6ms/step - loss: 0.1051 -
accuracy: 0.8112 - val_loss: 0.1059 - val_accuracy: 0.8119
Epoch 4/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0998 -
accuracy: 0.8121 - val_loss: 0.0992 - val_accuracy: 0.8117
Epoch 5/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0964 -
accuracy: 0.8126 - val_loss: 0.1000 - val_accuracy: 0.8127
Epoch 6/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0939 -
accuracy: 0.8130 - val_loss: 0.0951 - val_accuracy: 0.8128
Epoch 7/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0921 -
accuracy: 0.8133 - val_loss: 0.0983 - val_accuracy: 0.8131
Epoch 8/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0906 -
accuracy: 0.8135 - val_loss: 0.0958 - val_accuracy: 0.8132
Epoch 9/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0893 -
accuracy: 0.8137 - val_loss: 0.0922 - val_accuracy: 0.8132
Epoch 10/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0883 -
accuracy: 0.8138 - val_loss: 0.0917 - val_accuracy: 0.8134
```

[6]:
```
<tensorflow.python.keras.callbacks.History at 0x7f506029e080>
```

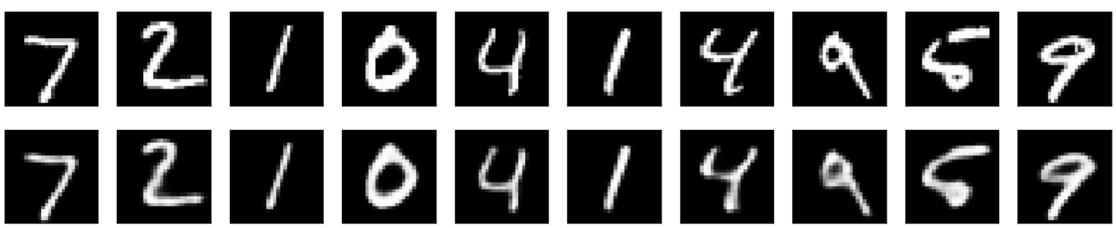
1.0.6 Show input images and the Denoising Autoencoder's output.

[7]:
```
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)

n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    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 reconstruction
    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()
```

```
WARNING:tensorflow:Model was constructed with shape (None, 8, 8, 16) for input
Tensor("input_3:0", shape=(None, 8, 8, 16), dtype=float32), but it was called on
an input with incompatible shape (None, 4, 4, 16).
```



2 Classifier

2.0.1 Set the encoder's layers to non-trainable except the last.

[0]:
```
for layer in encoder.layers[:-2]:
  layer.trainable = False
```

2.0.2 Set up the classifier layers.

[9]:
```
flattening_layer = tf.keras.layers.Flatten()
dense_layer = tf.keras.layers.Dense(64, activation="relu")
prediction_layer = tf.keras.layers.Dense(10)
model = tf.keras.Sequential([encoder, flattening_layer, dense_layer,␣
↪prediction_layer])

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss=tf.keras.losses.
↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=["accuracy"])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
model_1 (Model)              (None, 4, 4, 16)          7264
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 64)                16448
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 24,362
Trainable params: 19,418
Non-trainable params: 4,944
_________________________________________________________________
```

2.0.3 Train the model.

[10]:
```
model.fit(x_train, y_train,
          epochs=10,
          batch_size=128,
          shuffle=True,
          validation_data=(x_test, y_test))
```

```
Epoch 1/10
469/469 [==============================] - 2s 3ms/step - loss: 0.5469 -
accuracy: 0.8370 - val_loss: 0.2028 - val_accuracy: 0.9388
Epoch 2/10
469/469 [==============================] - 1s 3ms/step - loss: 0.1792 -
accuracy: 0.9455 - val_loss: 0.1407 - val_accuracy: 0.9551
Epoch 3/10
```

```
469/469 [==============================] - 1s 3ms/step - loss: 0.1311 -
accuracy: 0.9602 - val_loss: 0.1067 - val_accuracy: 0.9657
Epoch 4/10
469/469 [==============================] - 1s 3ms/step - loss: 0.1065 -
accuracy: 0.9668 - val_loss: 0.0896 - val_accuracy: 0.9702
Epoch 5/10
469/469 [==============================] - 1s 3ms/step - loss: 0.0918 -
accuracy: 0.9717 - val_loss: 0.0812 - val_accuracy: 0.9736
Epoch 6/10
469/469 [==============================] - 1s 3ms/step - loss: 0.0807 -
accuracy: 0.9755 - val_loss: 0.0683 - val_accuracy: 0.9771
Epoch 7/10
469/469 [==============================] - 1s 3ms/step - loss: 0.0723 -
accuracy: 0.9778 - val_loss: 0.0705 - val_accuracy: 0.9758
Epoch 8/10
469/469 [==============================] - 1s 3ms/step - loss: 0.0667 -
accuracy: 0.9794 - val_loss: 0.0701 - val_accuracy: 0.9766
Epoch 9/10
469/469 [==============================] - 1s 3ms/step - loss: 0.0622 -
accuracy: 0.9808 - val_loss: 0.0669 - val_accuracy: 0.9761
Epoch 10/10
469/469 [==============================] - 1s 3ms/step - loss: 0.0576 -
accuracy: 0.9822 - val_loss: 0.0549 - val_accuracy: 0.9810
```

```
[10]: <tensorflow.python.keras.callbacks.History at 0x7f506007f160>
```

The classifier achieves an accuracy of >98% in 10 epochs. It is comparable to the accuracy of training a neural network of fully connected layers with ~100,000 parameters. Whereas here we trained ~20,000 parameters and took advantage of a pretrained convolutional autoencoder.