main

TensorFlow_Tutorials / TensorFlow_2.x / Convolutional_Autoencoder.ipynb

hy-23 Created using Colaboratory History

1 contributor

353 lines (353 sloc) 48.1 KB

Open in Colab

In [1]:

```
import numpy as np
import tensorflow as tf
import tensorflow.keras as k
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, UpSampling2D
```

In [2]:

```
# seed values
np.random.seed(111)
tf.random.set_seed(111)

# hyperparameters
batch_size = 128
max_epochs = 50
filters = [32, 32, 16]
```

In [3]:

```
# download dataset
(x_train, _), (x_test, _) = k.datasets.mnist.load_data()

# process dataset
x_train = x_train / 255.
x_test = x_test / 255.

x_train = x_train.astype(np.float32)
x_test = x_test.astype(np.float32)

print("shape of x_train is {}".format(x_train.shape))
x_train = np.reshape(x_train, (*(x_train.shape), 1)) # * operator dereferences tuple.
print("shape of x_train is {}".format(x_train.shape))

print("shape of x_test is {}".format(x_test.shape))
x_test = np.reshape(x_test, (*(x_test.shape), 1)) # * operator dereferences tuple.
print("shape of x_train is {}".format(x_test.shape))

noise = np.random.normal(loc=0.0, scale=1.0, size = x_train.shape)
x_train_noisy = x_train + noise

noise = np.random.normal(loc=0.0, scale=1.0, size = x_test.shape)
x_test_noisy = x_test + noise

x_train_noisy = np.clip(x_train_noisy, 0.0, 1.0)
x_test_noisy = np.clip(x_test_noisy, 0.0, 1.0)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
shape of x_train is (60000, 28, 28)
shape of x_train is (60000, 28, 28, 1)
shape of x_test is (10000, 28, 28)
shape of x_train is (10000, 28, 28, 1)
```

Display outputs before and after adding noise.

In [4]:

```
number = 10
plt.figure(figsize=(20,4))
for index in range(number):
  # display original
  ax = plt.subplot(2, number, index+1)
  plt.imshow(x_test[index].reshape(28, 28), cmap='gray')
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

    # display original
  ax = plt.subplot(2, number, number+index+1)
  plt.imshow(x_test_noisy[index].reshape(28, 28), cmap='gray')
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

plt.show()
```

Autoencoder Network

In [5]:

```
# Encoder Network
class Encoder(k.layers.Layer):
  def __init__(self, filters, in_shape):
    #print("In __init__ of Encoder.")
    super(Encoder, self).__init__()
    self.conv1 = Conv2D(filters=filters[0], kernel_size=3, strides=1,
                        activation = 'relu', padding='same',
                        input_shape = in_shape)
    self.conv2 = Conv2D(filters=filters[1], kernel_size=3, strides=1,
                        activation='relu', padding='same')
    self.conv3 = Conv2D(filters=filters[2], kernel_size=3, strides=1,
                        activation='relu', padding='same')
    self.pool = MaxPooling2D(padding='same')

  def call(self, input_features):
    x = self.conv1(input_features)
    x = self.pool(x)
    x = self.conv2(x)
    x = self.pool(x)
    x = self.conv3(x)
    x = self.pool(x)
    return x

# Decoder Network
class Decoder(k.layers.Layer):
  def __init__(self, filters):
    #print("In __init__ of Decoder")
    super(Decoder, self).__init__()
    self.conv1 = Conv2D(filters=filters[2], kernel_size=3, strides=1,
                        activation='relu', padding='same')
    self.conv2 = Conv2D(filters=filters[1], kernel_size=3, strides=1,
                        activation='relu', padding='same')
    self.conv3 = Conv2D(filters=filters[0], kernel_size=3, strides=1,
                        activation='relu', padding='valid')
    self.conv4 = Conv2D(filters = 1, kernel_size=3, strides=1,
                        activation='softmax', padding='same')
    self.upsample = UpSampling2D(size=(2,2))

  def call(self, encoded_features):
    x = self.conv1(encoded_features)
    x = self.upsample(x)
    x = self.conv2(x)
    x = self.upsample(x)
    x = self.conv3(x)
    x = self.upsample(x)
    x = self.conv4(x)
    return x

# Autoencoder Network
class Autoencoder(k.Model):
  def __init__(self, filters, in_shape):
    super(Autoencoder, self).__init__()
    self.encoder = Encoder(filters, in_shape)
    self.decoder = Decoder(filters)

  def call(self, input_features):
    encode = self.encoder(input_features)
    decode = self.decoder(encode)
    return decode
```

In [6]:

```
model = Autoencoder(filters=filters, in_shape=x_train.shape[1:])
```

In [7]:

```
model.compile(loss='binary_crossentropy', optimizer='adam')

# model.summary()
# ValueError: This model has not yet been built. Build the model first by
# calling `build()` or by calling the model on a batch of data.
```

In [8]:

```
loss = model.fit(x=x_train_noisy, y=x_train, batch_size=batch_size,
                 epochs=25, verbose = 0)
```

In [ ]:

```
loss = model.fit(x=x_train_noisy, y=x_train, batch_size=batch_size,
                 epochs=25, verbose = 0)
```

In [ ]:

```
plt.plot(range(max_epochs), loss.history['loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```

In [ ]:

```
number = 10
plt.figure(figsize=(20,4))
for index in range(number):
  # display original
  ax = plt.subplot(2, number, index+1)
  plt.imshow(x_test_noisy[index].reshape(28, 28), cmap='gray')
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # display reconstruction
  ax = plt.subplot(2, number, index+number+1)
  #plt.imshow(model(x_test_noisy)[index].reshape(28, 28), cmap='gray')
  plt.imshow(tf.reshape(model(x_test_noisy)[index], (28, 28)), cmap='gray')
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```

In [ ]: