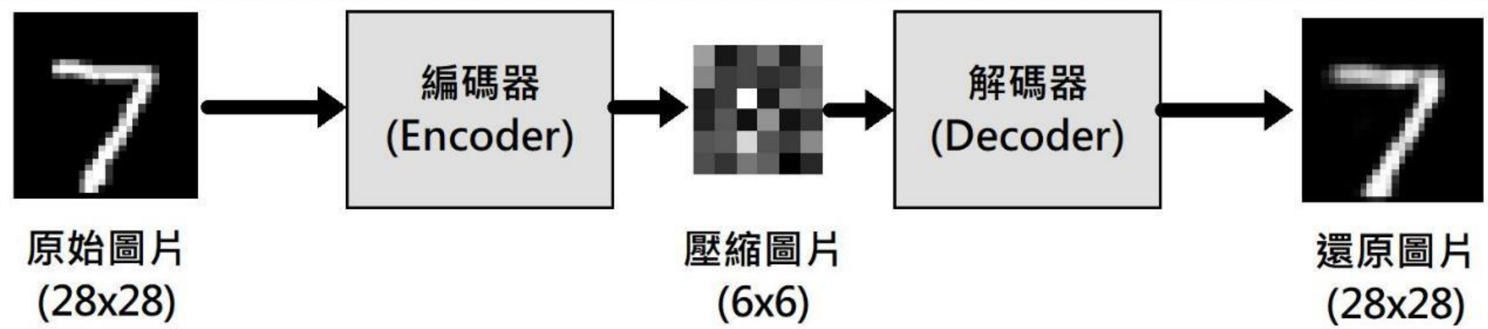
自編碼器(Autoencoder)

吳庭育
tyw@mail.npust.edu.tw

Cifar-10彩色圖片 資料集

- 自編碼器(Autoencoder)是一種實現編碼和解碼的神經網路，一種資料壓縮演算法，可以將原始資料透過編碼器(Encoder)的神經網路進行壓縮，和使用解碼器(Decoder)的神經網路還原成原始資料。



自編碼器的特色

- 只適用特定資料(Data-specific)
  - 自編碼器只適用與訓練資料集相似的資料壓縮。
- 資料損失
  - 自編碼器壓縮和還原會有資料損失，還原資料不會和原始資料完全相同。
- 非監督是學習
  - 自編碼器是自行從資料學習，因為訓練資料集就是和自己比較損失來進行學習。

- 自編碼器常用來減少資料集的維度，但仍然可以保留資料集的主要特徵，就是『主成分分析』(Principal Component Analysis, PCA)，就是降維(Dimensionality Reduction)的特徵擷取

Keras的Functional API

Keras神經層物件就是一個函式

- 在Keras建立的神經層物件可以當成函式來呼叫，也就是將各神經層視為是一個函式：

```
a = Input(shape=(32,))
b = Dense(32, activation="relu")(a)
```

- 上述程式碼先建立Input輸入層物件，傳回值是張量a(這就是輸入層輸入神經網路的特徵資料)，然後建立Dense物件，可以將Dense物件視為函式呼叫，函式的參數是此層神經層的輸入張量a，傳回值是此層神經層的輸出張量b，然後，我們可以建立Model模型：

```
model = Model(inputs=a, outputs=b)
```

- Model()的inputs參數是輸入模型的張量: outputs參數是輸出張量。
- 如果是多輸入和多輸出模型，可以使用清單來指定輸入和輸出張量：

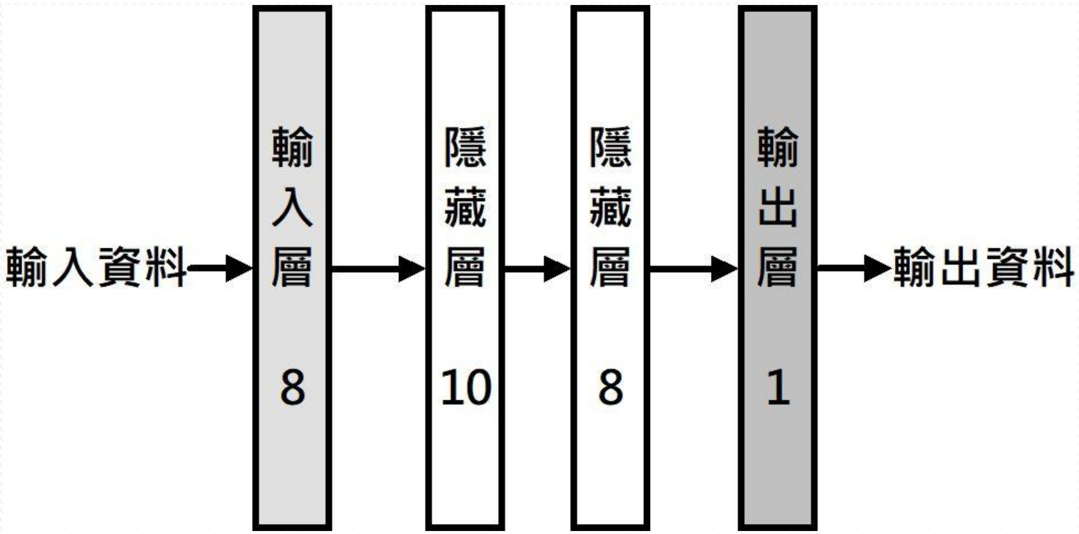
```
model = Model(inputs=[a1, a2], outputs=[b1, b2, b3])
```

使用Functional API定義神經網路模型

```python
import numpy as np
import pandas as pd
from keras.models import Model
from keras.layers import Input, Dense

np.random.seed(10)  # 指定亂數種子
# 載入糖尿病資料集
df = pd.read_csv("./diabetes.csv")
dataset = df.values
np.random.shuffle(dataset)  # 使用亂數打亂資料
# 分割成特徵資料和標籤資料
X = dataset[:, 0:8]
Y = dataset[:, 8]
# 定義模型 - 使用 Function API
inputs = Input(shape=(8,))
hidden1 = Dense(10, activation="relu")(inputs)
hidden2 = Dense(8, activation="relu")(hidden1)
outputs = Dense(1, activation="sigmoid")(hidden2)
model = Model(inputs=inputs, outputs=outputs)
model.summary()  # 顯示模型摘要資訊
# 編譯模型
model.compile(loss="binary_crossentropy", optimizer="sgd",
              metrics=["accuracy"])
# 訓練模型
model.fit(X, Y, epochs=15, batch_size=10)
# 評估模型
loss, accuracy = model.evaluate(X, Y)
print("準確度 = {:.2f}".format(accuracy))
```

使用Functional API定義神經網路模型

- 將之前糖尿病模型的Sequential模型改用Functional API來建立Model模型(只有定義模型部分不同，其他部分完全相同)，這是一個四層的深度神經網路。



Functional API定義網路模型

建立input層

input層 的傳回值當
作hidden1的輸入

```python
# 定義模型 - 使用 Function API
inputs = Input(shape=(8,))
hidden1 = Dense(10, activation="relu")(inputs)
hidden2 = Dense(8, activation="relu")(hidden1)
outputs = Dense(1, activation="sigmoid")(hidden2)
model = Model(inputs=inputs, outputs=outputs)
model.summary()     # 顯示模型摘要資訊
```

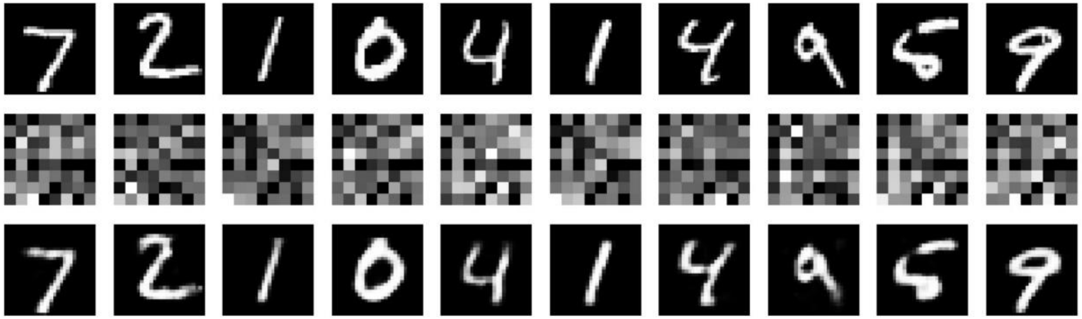
參數inputs 是模型的輸入張量(input層的傳回值)
參數outputs是模型的輸出張量(output層的傳回值)

使用MLP建立自編碼器(AE)

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

# 指定亂數種子
seed = 7
np.random.seed(seed)
# 載入資料集
(X_train, _), (X_test, _) = mnist.load_data()
# 轉換成 28*28 = 784 的向量
X_train = X_train.reshape(X_train.shape[0], 28*28).astype("float32")
X_test = X_test.reshape(X_test.shape[0], 28*28).astype("float32")
# 因為是固定範圍, 所以執行正規化, 從 0-255 至 0-1
X_train = X_train / 255
X_test = X_test / 255
# 定義 autoencoder 模型
input_img = Input(shape=(784,))
x = Dense(128, activation="relu")(input_img)
encoded = Dense(64, activation="relu")(x)
x = Dense(128, activation="relu")(encoded)
decoded = Dense(784, activation="sigmoid")(x)
autoencoder = Model(input_img, decoded)
autoencoder.summary()    # 顯示模型摘要資訊
# 定義 encoder 模型
encoder = Model(input_img, encoded)
encoder.summary()        # 顯示模型摘要資訊
# 定義 decoder 模型
decoder_input = Input(shape=(64,))
decoder_layer = autoencoder.layers[-2](decoder_input)
decoder_layer = autoencoder.layers[-1](decoder_layer)
decoder = Model(decoder_input, decoder_layer)
decoder.summary()        # 顯示模型摘要資訊
# 編譯模型
autoencoder.compile(loss="binary_crossentropy", optimizer="adam",
                    metrics=["accuracy"])
# 訓練模型
autoencoder.fit(X_train, X_train, validation_data=(X_test, X_test),
                epochs=10, batch_size=256, shuffle=True, verbose=2)
# 壓縮圖片
encoded_imgs = encoder.predict(X_test)
# 解壓縮圖片
decoded_imgs = decoder.predict(encoded_imgs)
# 顯示原始, 壓縮和還原圖片
import matplotlib.pyplot as plt

n = 10   # 顯示幾個數字
plt.figure(figsize=(20, 6))
for i in range(n):
    # 原始圖片
    ax = plt.subplot(3, n, i + 1)
    ax.imshow(X_test[i].reshape(28, 28), cmap="gray")
    ax.axis("off")
    # 壓縮圖片
    ax = plt.subplot(3, n, i + 1 + n)
    ax.imshow(encoded_imgs[i].reshape(8, 8), cmap="gray")
    ax.axis("off")
    # 還原圖片
    ax = plt.subplot(3, n, i + 1 + 2*n)
    ax.imshow(decoded_imgs[i].reshape(28, 28), cmap="gray")
    ax.axis("off")
plt.show()
```

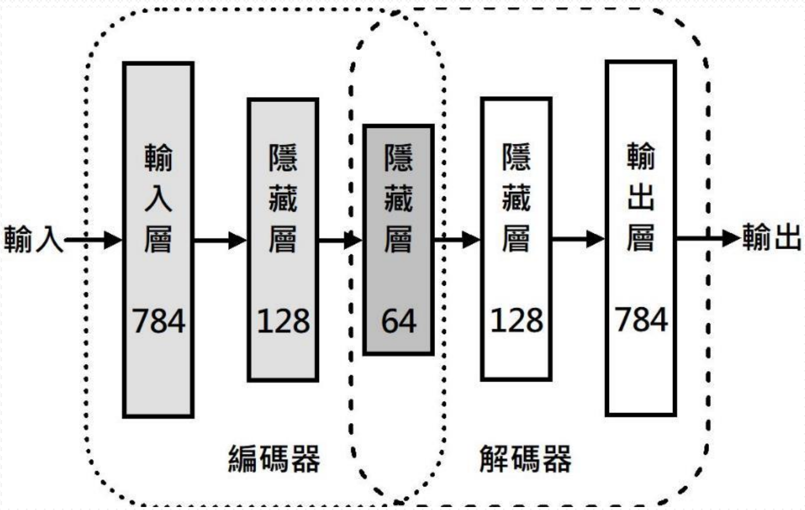
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 784)]             0

 dense_3 (Dense)             (None, 128)               100480

 dense_4 (Dense)             (None, 64)                8256

 dense_5 (Dense)             (None, 128)               8320

 dense_6 (Dense)             (None, 784)               101136

=================================================================
Total params: 218,192
Trainable params: 218,192
Non-trainable params: 0
_________________________________________________________________
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 784)]             0

 dense_3 (Dense)             (None, 128)               100480

 dense_4 (Dense)             (None, 64)                8256

=================================================================
Total params: 108,736
Trainable params: 108,736
Non-trainable params: 0
_________________________________________________________________
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 64)]              0

 dense_5 (Dense)             (None, 128)               8320

 dense_6 (Dense)             (None, 784)               101136

=================================================================
Total params: 109,456
Trainable params: 109,456
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/10
235/235 - 4s - loss: 0.2208 - accuracy: 0.0106 - val_loss: 0.1424 - val_accuracy: 0.0135 - 4s/epoch - 16ms/step
Epoch 2/10
235/235 - 4s - loss: 0.1256 - accuracy: 0.0113 - val_loss: 0.1120 - val_accuracy: 0.0098 - 4s/epoch - 15ms/step
Epoch 3/10
235/235 - 6s - loss: 0.1078 - accuracy: 0.0114 - val_loss: 0.1019 - val_accuracy: 0.0116 - 6s/epoch - 24ms/step
Epoch 4/10
235/235 - 6s - loss: 0.0994 - accuracy: 0.0116 - val_loss: 0.0951 - val_accuracy: 0.0128 - 6s/epoch - 24ms/step
Epoch 5/10
235/235 - 6s - loss: 0.0943 - accuracy: 0.0123 - val_loss: 0.0911 - val_accuracy: 0.0138 - 6s/epoch - 26ms/step
Epoch 6/10
235/235 - 6s - loss: 0.0910 - accuracy: 0.0126 - val_loss: 0.0886 - val_accuracy: 0.0135 - 6s/epoch - 26ms/step
Epoch 7/10
235/235 - 4s - loss: 0.0889 - accuracy: 0.0123 - val_loss: 0.0869 - val_accuracy: 0.0146 - 4s/epoch - 17ms/step
Epoch 8/10
235/235 - 3s - loss: 0.0872 - accuracy: 0.0121 - val_loss: 0.0854 - val_accuracy: 0.0125 - 3s/epoch - 14ms/step
Epoch 9/10
235/235 - 3s - loss: 0.0858 - accuracy: 0.0126 - val_loss: 0.0842 - val_accuracy: 0.0119 - 3s/epoch - 14ms/step
Epoch 10/10
235/235 - 3s - loss: 0.0846 - accuracy: 0.0125 - val_loss: 0.0832 - val_accuracy: 0.0129 - 3s/epoch - 14ms/step
```



使用MLP建立自編碼器(AE)

- Keras並不能使用Sequential模型建立自編碼器，因為需要重組自編碼器的神經層，來建立編碼器和解碼器模型，所以是使用Functional API來建立自編碼器，首先使用MLP建立自編碼器。



- 前半段是編碼器，每一層的神經元數都比上一層少。
- 後半段是解碼器，每一層的神經元數都比上一層多。
- 中間的隱藏層為重疊，並且前後的神經層是對稱

載入資料

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

- 匯入Functional API 的Model和 Input，然後載入資料集。因為是非監督式學習，只需要訓練和測試資料集的特徵資料，不需要標籤資料，故使用『_』變數來代替。

```
#  載入資料集
(X_train,  _),  (X_test,  _)  =  mnist.load_data()
```

步驟一：資料預處理

- 使用MLP打造自編碼器，用來壓縮和解壓縮MNIST手寫辨識資料集的圖片，在載入資料集後，需要執行資料預處理，將特徵資料轉換成28*28 = 784的向量：

```
# 轉換成 28*28 = 784 的向量
X_train = X_train.reshape(X_train.shape[0], 28*28).astype("float32")
X_test = X_test.reshape(X_test.shape[0], 28*28).astype("float32")
```

- 因為灰階值是固定範圍0~255，所以執行正規化從0~255轉換成0~1：

```
# 因為是固定範圍, 所以執行正規化, 從 0-255 至 0-1
X_train = X_train / 255
X_test = X_test / 255
```

步驟二：定義模型

- 完成資料載入和資料預處理，可以定義自編碼器的神經網路模型，首先定義自編碼器(AE)模型及建立Model物件：

```
# 定義 autoencoder 模型
input_img = Input(shape=(784,))
x = Dense(128, activation="relu")(input_img)
encoded = Dense(64, activation="relu")(x)
x = Dense(128, activation="relu")(encoded)
decoded = Dense(784, activation="sigmoid")(x)
autoencoder = Model(input_img, decoded)
autoencoder.summary()    # 顯示模型摘要資訊
```

對稱

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_8 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_9 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_10 (Dense)             (None, 784)               101136
=================================================================
Total params: 218,192
Trainable params: 218,192
Non-trainable params: 0
_________________________________________________________________
```

步驟二：定義模型

- 然後建立編碼器模型，這就是自編碼器(AE)模型的前半段：

```
# 定義 encoder 模型
encoder = Model(input_img, encoded)
encoder.summary()          # 顯示模型摘要資訊
```

- 上述程式碼建立Model物件，第1個參數是輸入張量input_img，第2個參數是輸出張量encoded，其模型摘要資訊如下所示：

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_8 (Dense)              (None, 64)                8256
=================================================================
Total params: 108,736
Trainable params: 108,736
Non-trainable params: 0
_________________________________________________________________
```

步驟二：定義模型

- 最後是解碼器模型，除了使用自編碼器(AE)模型的後半段外，還需要新增Input輸入層(形狀是編碼器模型的輸出層)：

```
# 定義 decoder 模型
decoder_input = Input(shape=(64,))
decoder_layer = autoencoder.layers[-2](decoder_input)
decoder_layer = autoencoder.layers[-1](decoder_layer)
decoder = Model(decoder_input, decoder_layer)
decoder.summary()        # 顯示模型摘要資訊
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 64)                0
_________________________________________________________________
dense_9 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_10 (Dense)             (None, 784)               101136
=================================================================
Total params: 109,456
Trainable params: 109,456
Non-trainable params: 0
_________________________________________________________________
```

步驟三：編譯模型

- 定義好模型後，需要編譯模型來轉換成低階TensorFlow計算圖：

```
# 編譯模型
autoencoder.compile(loss="binary_crossentropy", optimizer="adam",
                    metrics=["accuracy"])
```

- 上述compile()函式的損失函數是binary_crossentropy，優化器是adam，評估標準是accuracy準確度。

步驟四：訓練模型

- 在成功編譯模型後，就可以開始訓練模型，fit()中的第一個參數是X_train 訓練資料集，第二個參數也是X_train (標籤資料<自己>)，並且使用validation_data參數指定檢驗資料集是測試資料集。Shuffle參數值為True是打亂資料：

```
# 訓練模型
autoencoder.fit(X_train, X_train, validation_data=(X_test, X_test),
                epochs=10, batch_size=256, shuffle=True, verbose=2)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
 - 6s - loss: 0.2224 - acc: 0.7899 - val_loss: 0.1414 - val_acc: 0.8067
Epoch 2/10
 - 6s - loss: 0.1253 - acc: 0.8099 - val_loss: 0.1113 - val_acc: 0.8116
Epoch 3/10
 - 6s - loss: 0.1069 - acc: 0.8127 - val_loss: 0.1008 - val_acc: 0.8125
Epoch 4/10
 - 6s - loss: 0.0995 - acc: 0.8135 - val_loss: 0.0957 - val_acc: 0.8131
Epoch 5/10
 - 6s - loss: 0.0946 - acc: 0.8140 - val_loss: 0.0913 - val_acc: 0.8132
Epoch 6/10
 - 6s - loss: 0.0912 - acc: 0.8143 - val_loss: 0.0887 - val_acc: 0.8134
Epoch 7/10
 - 6s - loss: 0.0888 - acc: 0.8145 - val_loss: 0.0868 - val_acc: 0.8136
Epoch 8/10
 - 6s - loss: 0.0870 - acc: 0.8146 - val_loss: 0.0853 - val_acc: 0.8138
Epoch 9/10
 - 6s - loss: 0.0855 - acc: 0.8147 - val_loss: 0.0839 - val_acc: 0.8138
Epoch 10/10
 - 6s - loss: 0.0842 - acc: 0.8148 - val_loss: 0.0828 - val_acc: 0.8138
```

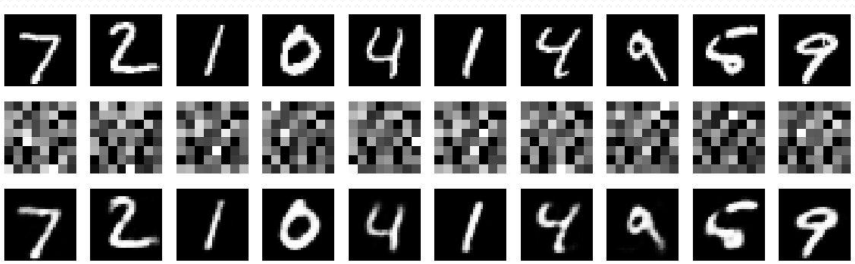
步驟五：使用自編碼器來編碼和解碼手寫數字圖片

- 當使用訓練資料集成功訓練模型後，我們可以使用encoder編碼器模型來編碼輸入資料的手寫圖片，也就是壓縮圖片
- 程式碼使用encoder模型的predict()函式壓縮X_test測試資料集的圖片，可以傳回編碼壓縮後的圖片資料，然後使用decoder解碼器模型來解壓縮圖片，即解碼圖片：

```
#  壓縮圖片
encoded_imgs = encoder.predict(X_test)
#  解壓縮圖片
decoded_imgs = decoder.predict(encoded_imgs)
```

步驟五：使用自編碼器來編碼和解碼手寫數字圖片

```python
# 顯示原始, 壓縮和還原圖片
import matplotlib.pyplot as plt

n = 10  # 顯示幾個數字
plt.figure(figsize=(20, 6))
for i in range(n):
    # 原始圖片
    ax = plt.subplot(3, n, i + 1)
    ax.imshow(X_test[i].reshape(28, 28), cmap="gray")
    ax.axis("off")
    # 壓縮圖片
    ax = plt.subplot(3, n, i + 1 + n)
    ax.imshow(encoded_imgs[i].reshape(8, 8), cmap="gray")
    ax.axis("off")
    # 還原圖片
    ax = plt.subplot(3, n, i + 1 + 2*n)
    ax.imshow(decoded_imgs[i].reshape(28, 28), cmap="gray")
    ax.axis("off")
plt.show()
```



使用CNN建立自編碼器(CAE)

```python
import numpy as np
from keras.datasets import mnist
from keras.models import Model
from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D

# 指定亂數種子
seed = 7
np.random.seed(seed)
# 載入資料集
(X_train, _), (X_test, _) = mnist.load_data()
# 轉換成 4D 張量
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype("float32")
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype("float32")
# 因為是固定範圍, 所以執行正規化, 從 0-255 至 0-1
X_train = X_train / 255
X_test = X_test / 255
# 定義 autoencoder 模型
input_img = Input(shape=(28,28,1))
x = Conv2D(16, (3,3), activation="relu", padding="same")(input_img)
x = MaxPooling2D((2,2), padding="same")(x)
x = Conv2D(8, (3,3), activation="relu", padding="same")(x)
x = MaxPooling2D((2,2), padding="same")(x)
x = Conv2D(8, (3,3), activation="relu", padding="same")(x)
encoded = MaxPooling2D((2,2), padding="same")(x)
x = Conv2D(8, (3,3), activation="relu", padding="same")(encoded)
x = UpSampling2D((2,2))(x)
x = Conv2D(8, (3,3), activation="relu", padding="same")(x)
x = UpSampling2D((2,2))(x)
x = Conv2D(16, (3,3), activation="relu")(x)
x = UpSampling2D((2,2))(x)
decoded = Conv2D(1, (3, 3), activation="sigmoid", padding="same")(x)
autoencoder = Model(input_img, decoded)
autoencoder.summary()    # 顯示模型摘要資訊
# 定義 encoder 模型
encoder = Model(input_img, encoded)
encoder.summary()    # 顯示模型摘要資訊
# 定義 decoder 模型
decoder_input = Input(shape=(4,4,8))
decoder_layer = autoencoder.layers[-7](decoder_input)
decoder_layer = autoencoder.layers[-6](decoder_layer)
decoder_layer = autoencoder.layers[-5](decoder_layer)
decoder_layer = autoencoder.layers[-4](decoder_layer)
decoder_layer = autoencoder.layers[-3](decoder_layer)
decoder_layer = autoencoder.layers[-2](decoder_layer)
decoder_layer = autoencoder.layers[-1](decoder_layer)
decoder = Model(decoder_input, decoder_layer)
decoder.summary()    # 顯示模型摘要資訊
# 編譯模型
autoencoder.compile(loss="binary_crossentropy", optimizer="adam",
                    metrics=["accuracy"])
# 訓練模型
autoencoder.fit(X_train, X_train, validation_data=(X_test, X_test),
                epochs=10, batch_size=128, shuffle=True, verbose=2)
# 壓縮圖片
encoded_imgs = encoder.predict(X_test)
# 解壓縮圖片
decoded_imgs = decoder.predict(encoded_imgs)
# 顯示原始, 壓縮和還原圖片
import matplotlib.pyplot as plt

n = 10    # 顯示幾個數字
plt.figure(figsize=(20, 8))
for i in range(n):
    # 原始圖片
    ax = plt.subplot(3, n, i + 1)
    ax.imshow(X_test[i].reshape(28, 28), cmap="gray")
    ax.axis("off")
    # 壓縮圖片
    ax = plt.subplot(3, n, i + 1 + n)
    ax.imshow(encoded_imgs[i].reshape(4, 4*8).T, cmap="gray")
    ax.axis("off")
    # 還原圖片
    ax = plt.subplot(3, n, i + 1 + 2*n)
    ax.imshow(decoded_imgs[i].reshape(28, 28), cmap="gray")
    ax.axis("off")
plt.show()
```

使用CNN建立自編碼器(CAE)

- 使用CNN建立自編碼器，簡稱CAE，CNN自編碼器的結構：
  - 前半段編碼器：3組Conv2D和MaxPooling2D神經層。
  - 後半段解碼器：3組Conv2D和UpSampling2D神經層。
- 上述MaxPooling2D最大池化層會壓縮圖片，UpSampling2D是對應MaxPooling2D來還原圖片。

```
# 定義 autoencoder 模型
input_img = Input(shape=(28,28,1))
x = Conv2D(16, (3,3), activation="relu", padding="same")(input_img)
x = MaxPooling2D((2,2), padding="same")(x)
x = Conv2D(8, (3,3), activation="relu", padding="same")(x)
x = MaxPooling2D((2,2), padding="same")(x)
x = Conv2D(8, (3,3), activation="relu", padding="same")(x)
encoded = MaxPooling2D((2,2), padding="same")(x)
x = Conv2D(8, (3,3), activation="relu", padding="same")(encoded)
x = UpSampling2D((2,2))(x)
x = Conv2D(8, (3,3), activation="relu", padding="same")(x)
x = UpSampling2D((2,2))(x)
x = Conv2D(16, (3,3), activation="relu")(x)
x = UpSampling2D((2,2))(x)
decoded = Conv2D(1, (3, 3), activation="sigmoid", padding="same")(x)
autoencoder = Model(input_img, decoded)
autoencoder.summary()        # 顯示模型摘要資訊
```

使用CNN建立自編碼器(CAE)

- 上面前後神經層的形狀幾乎是對稱，因為池化運算縮小2倍，上升取樣運算放大2倍
- 池化運算: 28/2 → 14/2→ 7/2 → 4
- 上升取樣運算: 4*2→8*2→ 16*2→32

- 最後輸出32 不是原來的28
- 在conv2d_6的 Conv2D層沒有使用padding="same" 參數，目的是將尺寸調整成14，14*2=28，
最後輸出28(14*2)

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 16)        160
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 8)         1160
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 8)           0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 8)           584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 8)           0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 4, 8)           584
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 8, 8, 8)           0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 8)           584
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 16, 16, 8)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 14, 14, 16)        1168
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 28, 28, 16)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 1)         145
=================================================================
Total params: 4,385
Trainable params: 4,385
Non-trainable params: 0
_________________________________________________________________
```

定義自編碼器(AE)模型

- 然後建立編碼器模型，這就是自編碼器(AE)模型的前半段：

```
# 定義 encoder 模型
encoder = Model(input_img, encoded)
encoder.summary()     # 顯示模型摘要資訊
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 16)        160
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 8)         1160
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 8)           0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 8)           584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 8)           0
=================================================================
Total params: 1,904
Trainable params: 1,904
Non-trainable params: 0
_________________________________________________________________
```

定義自編碼器(AE)模型

- 首先新增解碼器模型的Input輸入層，然後使用Model物件的layers屬性取出autoencoder模型最後7層神經層，就可以建立解碼器的Model模型，其模型摘要資訊如下所示：

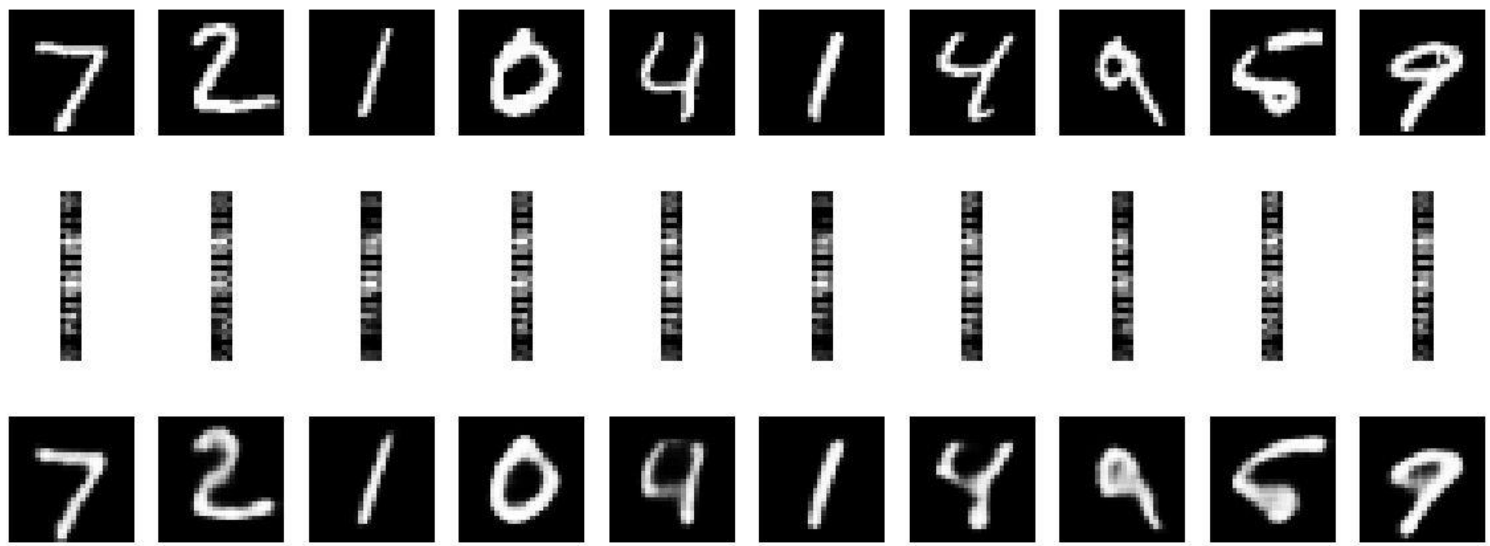
```
# 定義 decoder 模型
decoder_input = Input(shape=(4,4,8))
decoder_layer = autoencoder.layers[-7](decoder_input)
decoder_layer = autoencoder.layers[-6](decoder_layer)
decoder_layer = autoencoder.layers[-5](decoder_layer)
decoder_layer = autoencoder.layers[-4](decoder_layer)
decoder_layer = autoencoder.layers[-3](decoder_layer)
decoder_layer = autoencoder.layers[-2](decoder_layer)
decoder_layer = autoencoder.layers[-1](decoder_layer)
decoder = Model(decoder_input, decoder_layer)
decoder.summary()      # 顯示模型摘要資訊
```

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 4, 4, 8)	0
conv2d_4 (Conv2D)	(None, 4, 4, 8)	584
up_sampling2d_1 (UpSampling2	(None, 8, 8, 8)	0
conv2d_5 (Conv2D)	(None, 8, 8, 8)	584
up_sampling2d_2 (UpSampling2	(None, 16, 16, 8)	0
conv2d_6 (Conv2D)	(None, 14, 14, 16)	1168
up_sampling2d_3 (UpSampling2	(None, 28, 28, 16)	0
conv2d_7 (Conv2D)	(None, 28, 28, 1)	145

Total params: 2,481
Trainable params: 2,481
Non-trainable params: 0

定義自編碼器(AE)模型

- 當使用訓練資料集成功訓練模型後，我們可以使用自編碼器來編碼和解碼手寫數字圖片和使用Matplotlib繪出前10張原始圖片、壓縮圖片和最後的還原圖片：



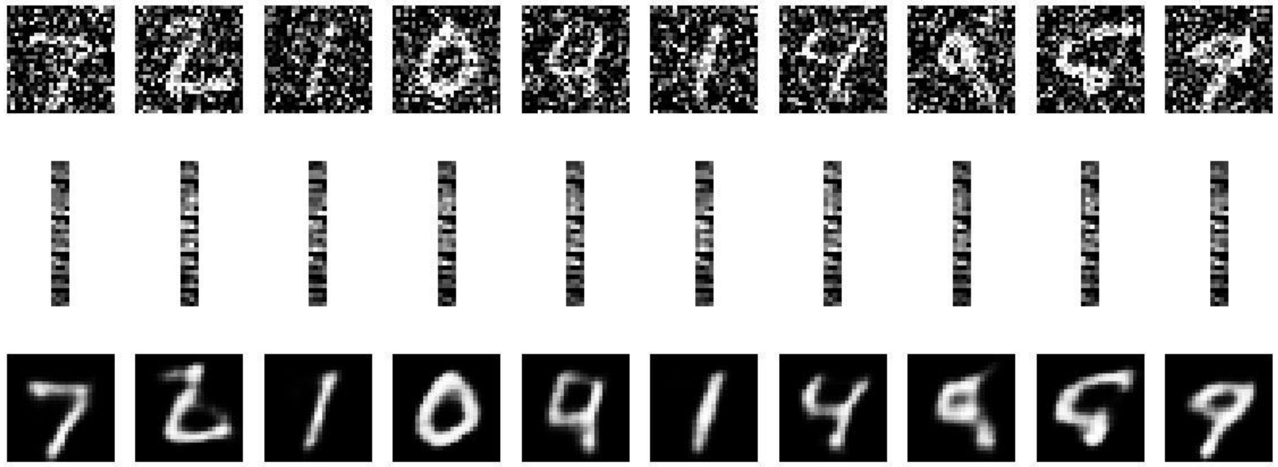
使用CNN自編碼器去除圖片雜訊

```python
import numpy as np
from keras.datasets import mnist
from keras.models import Model
from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D

# 指定亂數種子
seed = 7
np.random.seed(seed)
# 載入資料集
(X_train, _), (X_test, _) = mnist.load_data()
# 轉換成 4D 張量
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype("float32")
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype("float32")
# 因為是固定範圍, 所以執行正規化, 從 0-255 至 0-1
X_train = X_train / 255
X_test = X_test / 255
# 替圖片製造雜訊
nf = 0.5
size_train = X_train.shape
X_train_noisy = X_train+nf*np.random.normal(loc=0.0,
                                            scale=1.0,size=size_train)
X_train_noisy = np.clip(X_train_noisy, 0., 1.)
size_test = X_test.shape
X_test_noisy = X_test+nf*np.random.normal(loc=0.0,
                                          scale=1.0,size=size_test)
X_test_noisy = np.clip(X_test_noisy, 0., 1.)
# 定義 autoencoder 模型
input_img = Input(shape=(28,28,1))
x = Conv2D(16, (3,3), activation="relu", padding="same")(input_img)
x = MaxPooling2D((2,2), padding="same")(x)
x = Conv2D(8, (3,3), activation="relu", padding="same")(x)
x = MaxPooling2D((2,2), padding="same")(x)
x = Conv2D(8, (3,3), activation="relu", padding="same")(x)
encoded = MaxPooling2D((2,2), padding="same")(x)
x = Conv2D(8, (3,3), activation="relu", padding="same")(encoded)
x = UpSampling2D((2,2))(x)
x = Conv2D(8, (3,3), activation="relu", padding="same")(x)
x = UpSampling2D((2,2))(x)
x = Conv2D(16, (3,3), activation="relu")(x)
x = UpSampling2D((2,2))(x)
decoded = Conv2D(1, (3, 3), activation="sigmoid", padding="same")(x)
autoencoder = Model(input_img, decoded)
autoencoder.summary()    # 顯示模型摘要資訊
```

```python
# 定義 encoder 模型
encoder = Model(input_img, encoded)
encoder.summary()    # 顯示模型摘要資訊
# 定義 decoder 模型
decoder_input = Input(shape=(4,4,8))
decoder_layer = autoencoder.layers[-7](decoder_input)
decoder_layer = autoencoder.layers[-6](decoder_layer)
decoder_layer = autoencoder.layers[-5](decoder_layer)
decoder_layer = autoencoder.layers[-4](decoder_layer)
decoder_layer = autoencoder.layers[-3](decoder_layer)
decoder_layer = autoencoder.layers[-2](decoder_layer)
decoder_layer = autoencoder.layers[-1](decoder_layer)
decoder = Model(decoder_input, decoder_layer)
decoder.summary()    # 顯示模型摘要資訊
# 編譯模型
autoencoder.compile(loss="binary_crossentropy", optimizer="adam",
                    metrics=["accuracy"])
# 訓練模型
autoencoder.fit(X_train_noisy, X_train,
                validation_data=(X_test_noisy, X_test),
                epochs=10, batch_size=128, shuffle=True, verbose=2)
# 壓縮圖片
encoded_imgs = encoder.predict(X_test_noisy)
# 解壓縮圖片
decoded_imgs = decoder.predict(encoded_imgs)
# 顯示雜訊圖片, 壓縮圖片和還原圖片
import matplotlib.pyplot as plt

n = 10   # 顯示幾個數字
plt.figure(figsize=(20, 8))
for i in range(n):
    # 雜訊圖片
    ax = plt.subplot(3, n, i + 1)
    ax.imshow(X_test_noisy[i].reshape(28, 28), cmap="gray")
    ax.axis("off")
    # 壓縮圖片
    ax = plt.subplot(3, n, i + 1 + n)
    ax.imshow(encoded_imgs[i].reshape(4, 4*8).T, cmap="gray")
    ax.axis("off")
    # 還原圖片
    ax = plt.subplot(3, n, i + 1 + 2*n)
    ax.imshow(decoded_imgs[i].reshape(28, 28), cmap="gray")
    ax.axis("off")
plt.show()
```

定義自編碼器(AE)模型

- CNN自編碼器只需使用有雜訊的圖片來進行訓練，就可以用來去除圖片上的雜訊，只是改用有雜訊圖片來進行訓練，以便使用CNN自編碼器來去除圖片上的雜訊。



使用CNN自編碼器去除圖片雜訊

```
# 替圖片製造雜訊
nf = 0.5
size_train = X_train.shape
X_train_noisy = X_train+nf*np.random.normal(loc=0.0,
                                            scale=1.0,size=size_train)
X_train_noisy = np.clip(X_train_noisy, 0., 1.)
size_test = X_test.shape
X_test_noisy = X_test+nf*np.random.normal(loc=0.0,
                                          scale=1.0,size=size_test)
X_test_noisy = np.clip(X_test_noisy, 0., 1.)
```

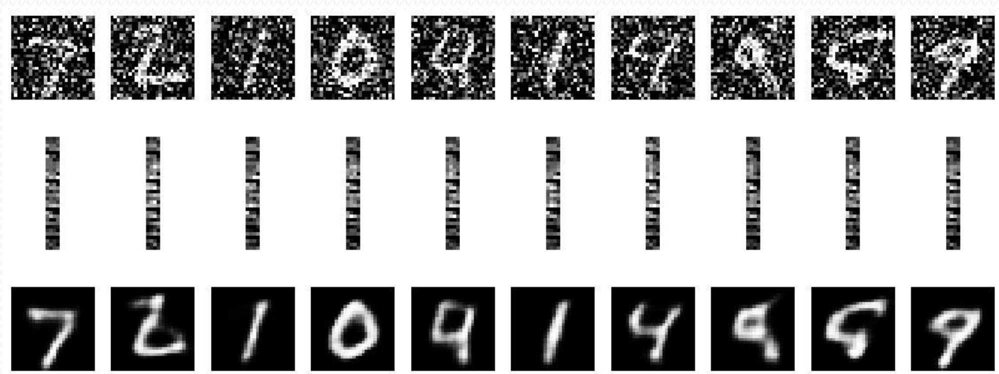
使用CNN自編碼器去除圖片雜訊

- Python程式改用X_train_noisy有雜訊圖片來訓練模型：

```python
# 訓練模型
autoencoder.fit(X_train_noisy, X_train,
                validation_data=(X_test_noisy, X_test),
                epochs=10, batch_size=128, shuffle=True, verbose=2)
```

- 上述fit()函式的第1個參數是X_train_noisy，在完成訓練後，我們可以壓縮圖片和解壓縮圖片：

```python
# 壓縮圖片
encoded_imgs = encoder.predict(X_test_noisy)
# 解壓縮圖片
decoded_imgs = decoder.predict(encoded_imgs)
```



End ！