深度學習模型調校

吳庭育

tyw@mail.npust.edu.tw

識別出模型的過度擬合問題

識別出模型的過度擬合問題

- 過度擬合(Overfitting)是指模型對於訓練資料集的分類或預測有很高的準確度，但是對於測試資料集的準確度就很差，表示模型對於訓練資料集的資料有過度擬合的問題。

為什麼模型會產生過度擬合

- 先前章節已經說明隨著訓練迴圈次數的增加(即訓練週期增加)，神經網路會因為過多的訓練而過度學習，造成神經網路建立的預測模型缺乏「泛化性」(Generalization)，這就是過度擬合(Overfitting)。

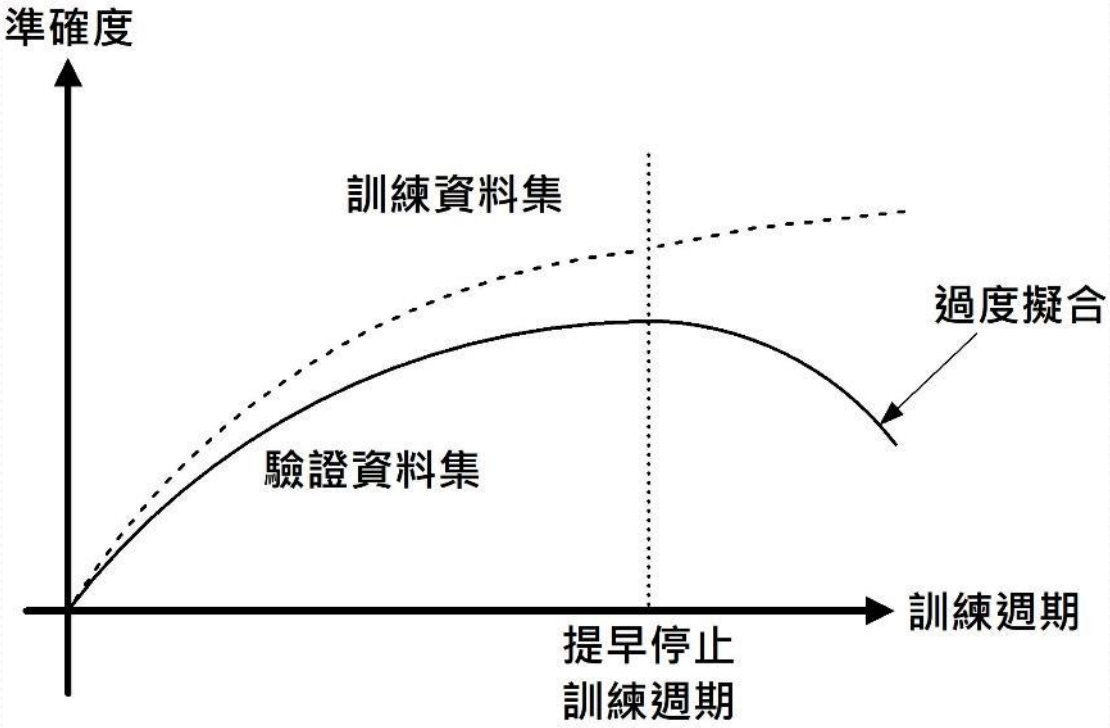
- 在實務上，我們訓練模型時常常會遇到過度擬合的問題，其背後隱藏的原因是我們真正需要的模型比我們訓練出來的模型要簡單，也就是說我們訓練出來的模型太複雜了，記住了太多訓練資料集的雜訊，所以對於沒有看過的資料，錯誤率就會大幅上升。

最佳化(Optimization)與泛化性(Generalization)的差異

- 最佳化(Optimization)
  - 最佳化是在找出能最小化訓練資料損失的模型參數，即找出模型的最佳權重。
- 泛化性(Generalization)
  - 模型對於未知且從沒有看過的資料也能有很好的預測性。

如何識別出模型過度擬合的問題

- 可以從模型的訓練和驗證準確度與損失圖表的趨勢，識別出是否有過度擬合的問題。以訓練和驗證準確度圖表來說，訓練資料集在反覆學習後，損失會逐漸下降；準確度會上升。



如何識別出模型過度擬合的問題

1. 驗證資料集和訓練資料集相同，都是準確度上升；損失逐漸下降，並且逐步接近訓練準確度的哪一條線，這是我們最希望的訓練成果。

2. 驗證資料集的準確度不會上升；損失也不太會下降，和準確度的哪一條線一直維持一定差距，如果之間的差距過大，表示模型過度擬合相當嚴重。

3. 驗證資料集的準確度不但沒有上升，反而是下降，這是訓練次數太多造成的過度擬合，我們可以提早停止訓練週期(Early Stopping Epoch)。

避免過度擬合

避免過度擬合的方式

- 增加訓練資料集的資料量
- 使用 資料增強技術(Data Augmentation)
- 減少模型的複雜度
- 使用Dropout層
- 提早停止訓練週期(Early Stopping Epoch)
- L1和L2常規化(L1 and L2 Regularization)

增加訓練資料集的資料量

- 最簡單避免過度擬合的方法是增加訓練資料，事實上，增加訓練資料的目的是在增加訓練資料的多樣性，因為多樣性的訓練資料可以避免模型過度擬合。

- 例如：建立模型分類貓和狗的圖片，如果訓練資料的圖片只有大型狗，當模型預測一隻屬於小型狗的博美犬，就可能被誤判成是貓，所以，儘可能增加小型狗和不同種類的訓練資料，讓訓練資料更多樣性，如此就可以減少誤判，避免模型過度擬合大型狗，以為狗只有大型狗。

使用資料增強技術(Data Augmentation)

- 如果實際增加訓練資料有困難，我們可以使用資料增強(Data Augmentation)技術，以現有的訓練資料集為範本，使用剪裁、旋轉、縮放、位移和翻轉等方式來增加額外的訓練資料。

- 例如：現有的狗圖片大多是面向左方，我們可以使用資料增強，增加水平翻轉成面向右方的狗圖片，馬上增加訓練資料集的資料量。

減少模型的複雜度

- 一般來說，過度擬合表示我們訓練出來的模型太複雜，所以減少模型的複雜度，讓模型結構變的更簡單可以避免過度擬合。
  - 從模型中刪除一些隱藏層的神經層。
  - 在神經層減少神經元數。

使用Dropout層

- Dropout層可以隨機忽略神經層的神經元集合，也就是隨機將權重歸零，在模型增加Dropout層也是一種避免過度擬合的好方法：
  - 在模型增加更多的Dropout層。
  - 在Dropout層增加權重歸零的比例。
    - 例如：從0.5增加至0.75。

提早停止訓練週期(Early Stopping Epoch)

- 如果是因為訓練週期太多所造成的過度擬合，我們除了手動減少訓練週期外，也可以使用Keras API的EarlyStopping，當準確度不再提升時，提早停止訓練週期。

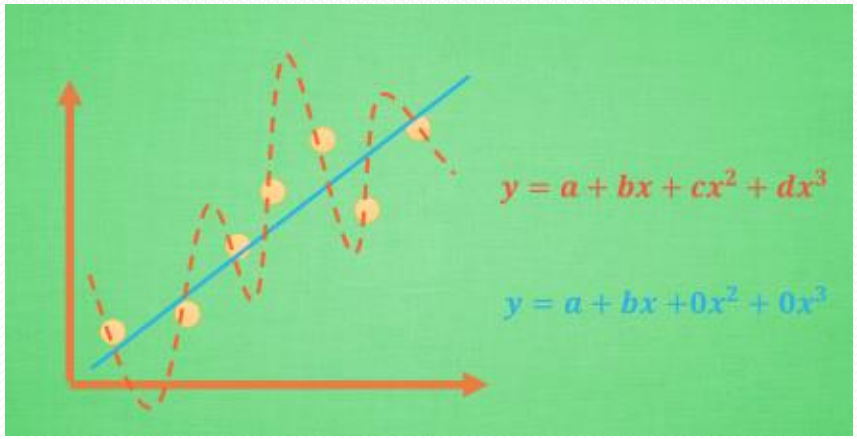
L1和L2常規化 (L1 and L2 Regularization)

- 過度擬合(overfitting)發生時，有可能是因為我們訓練的假設模型本身就過於複雜，所以我們是否能讓複雜的假設模型退回至比較簡單的假設模型呢？這個退回去的方法就是正規化(Regularization)

- L1和L2常規化(L1 and L2 Regularization)是一種權重衰減(Weight Decay)觀念，也就是懲罰權重，因為當模型產生過度擬合時，權重往往也會變的特別大，為了避免權重變的太大，我們可以在計算預測值和真實目標值的損失時，加上一個懲罰項。

L1和L2常規化 (L1 and L2 Regularization)

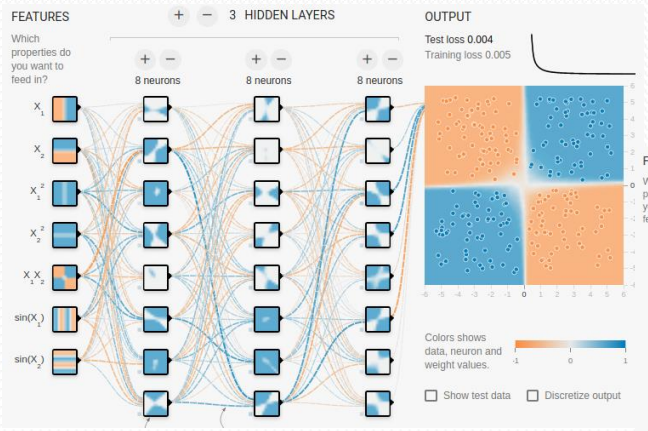
- 如果模型過擬合，那麼這個方程就可能是一個比較複雜的非線性方程，正是因為這裡 X^2和X^3和使得這條虛線能夠被彎來彎去，所以整個模型就會特別努力地去學習作用在X^2和X^3上的c和d參數，但是我們期望模型要學到的卻是這條藍色的曲線，因為它能更有效地概括數據。



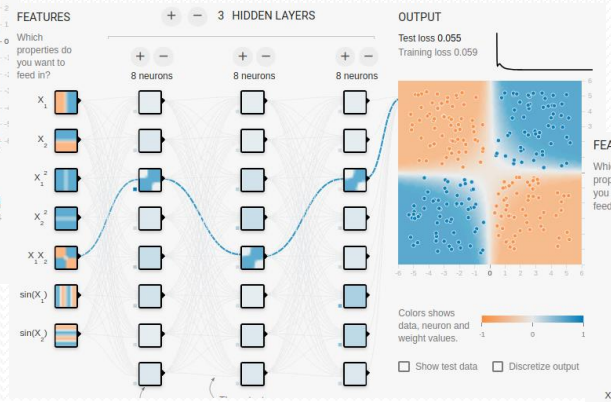
  - 第1個：就是減少特徵，把他的權重變成0，這叫L1正規化。
  - 第2個：就是減少特徵權重差異，讓某些特徵的權重不要太突出，這叫L2正規化。

L1和L2常規化 (L1 and L2 Regularization)

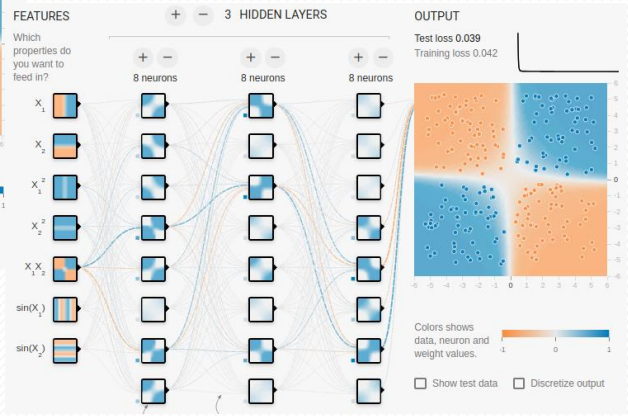
- L1正規化：有可能導致零權重，因刪除更多特徵而使模型稀疏。
- L2正規化：會對更大的權重值造成更大的影響，將使權重值保持較小。



沒有正規化的情況，
每個權重與神經元都處於非常活躍狀態



L1：剩下X1X2的特徵與權重。



L2：每個權重與神經元都處於活動狀態，但是非常虛弱

L1正規化使用其中一個特徵而將某些拋棄，而L2正規化將同時保留特徵並使權重值保持較小。因此，使用L1，您可以得到一個較小的模型，但預測性可能較低。

避免低度擬合(Underfitting)

增加模型的複雜度

- 我們需要增加模型的複雜度來避免低度擬合(Underfitting)，這是因為我們的訓練資料太複雜，但是模型太簡單，以至於根本沒有能力學會這些資料。
- 我們有多種方法來增加模型的複雜度：
  - 增加模型的神經層數。
  - 在每一層神經層增加神經元數。
  - 使用不同的神經層種類。

減少Dropout層

- 當模型產生低度擬合時，我們需要刪除過多的Dropout層，或是降低Dropout層隨機歸零的比例。
- 例如：50%歸零的Dropout層如果造成低度擬合，我們可以降低成30%；不行，再降低成20%；再不行，根本就直接刪除Dropout層。

在樣本資料增加更多的特徵數

- 對於訓練資料集的樣本資料，我們可以增加更多特徵數來避免低度擬合，更多的特徵數可以幫助模型更容易進行分類，例如：使用高度和寬度的尺寸特徵來進行分類，如果再加上額外的色彩特徵，就可以幫助模型分類的更好。
  - 例如準備建立預測股票價格的模型，原來只有收盤價的特徵，如果模型低度擬合，我們可以增加開盤價、最高價、最低價和成交量等額外特徵，幫助模型能夠更容易的預測股價。

加速神經網路的訓練
選擇優化器

認識優化器(Optimizer)

- 優化器(Optimizer)的功能是更新神經網路的權重來讓損失函數的誤差值最小化，以便找出神經網路的最佳權重。在作法上，優化器是使用反向傳播計算出每一層權重需要分擔損失的梯度後，使用梯度下降法更新神經網路每一個神經層的權重來最小化損失函數，其基本公式如下所示：

  W1 = W0 - 學習率 * 梯度

- 上述公式是將權重W0更新至W1，為了加速神經網路的訓練，我們可以從學習率(Learning Rate)的步伐大小和使用多少資料量計算梯度來改進優化器的效能。另一種方式是增加一些參數，例如：動量：

  W1 = W0 - (學習率 * 梯度 + 動量)

Keras優化器的超參數：SGD

- Keras最基本的優化器是SGD，Keras的SGD就是指MBGD (Mini-Batch Gradient Descent)，在每一次的迭代是使用批次數的樣本數量來計算梯度，而且對每一個參數都是使用相同的學習率來進行更新。

- SGD的問題是如果學習率太小，收斂速度會很慢，而且很容易找到局部最佳解，而不是全域最佳解。在Python程式建立優化器需要匯入Optimizers，如下所示：

```
from keras import optimizers
opt_sgd = optimizers.SGD(lr=0.01, momentum=0.0,
decay=0.0)
```

- 上述程式碼建立SGD物件，其參數是優化器的超參數 (Hyperparameters)。

動量(Momentum)

- 動量是源於物理學的慣性，同一個方向會加速；更改方向會減速，其公式如下所示：
  - **$V_1 = -lr * 梯度 + V_0 * momentum$**
  - **$W_1 = W_0 + V_1$**
- 上述V_1是這一次的更新量；V_0是上一次的更新量，lr是前述學習率的超參數，momentum是前述動量的超參數，此時有兩種情況，如下所示：
  - 當梯度方向和上一次更新量的方向相同，可以從上一次更新量得到加速作用。
  - 當梯度方向和上一次更新量的方向相反，可以從上一次更新量得到減速作用。

學習率衰減係數(Learning Rate Decay)

- 學習率衰減係數是指學習率會隨著每一次的參數更新而逐漸減少，換句話說，剛開始的訓練使用大步伐，隨著訓練增加，愈來愈接近最小值時，學習率的步伐也會愈變愈小：

$$lr_1 = lr_0 * 1.0 / (1.0 + decay * 更新次數)$$

自適應性學習率(Adaptive Learning Rates)

- Keras的Adagrad、Adadelta、RMSprop和Adam都是一種自適應性學習率的優化器。「自適應性」(Adaptive)是指依目前的條件或環境，可以自動依據條件及環境來自動進行調整，以便達到更好的適應性。

```python
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import optimizers

# 指定亂數種子
seed = 10
np.random.seed(seed)
# 載入資料集
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()
# 因為是固定範圍, 所以執行正規化, 從 0-255 至 0-1
X_train = X_train.astype("float32") / 255
X_test = X_test.astype("float32") / 255
# One-hot編碼
Y_train = to_categorical(Y_train)
Y_test = to_categorical(Y_test)
# 定義模型
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), padding="same",
                 input_shape=X_train.shape[1:], activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, kernel_size=(3, 3), padding="same",
                 activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation="softmax"))
model.summary()      # 顯示模型摘要資訊
# 編譯模型
opt_sgd = optimizers.SGD(lr=0.05,decay=1e-6, momentum=.09)
model.compile(loss="categorical_crossentropy", optimizer=opt_sgd,
              metrics=["accuracy"])
# 訓練模型
history = model.fit(X_train, Y_train, validation_split=0.2,
                    epochs=40, batch_size=128, verbose=2)
```

```python
# 評估模型
print("\nTesting ...")
loss, accuracy = model.evaluate(X_test, Y_test)
print("測試資料集的準確度 = {:.2f}".format(accuracy))
# 顯示圖表來分析模型的訓練過程
import matplotlib.pyplot as plt
# 顯示訓練和驗證損失
loss = history.history["loss"]
epochs = range(1, len(loss)+1)
val_loss = history.history["val_loss"]
plt.plot(epochs, loss, "bo-", label="Training Loss")
plt.plot(epochs, val_loss, "ro--", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
# 顯示訓練和驗證準確度
acc = history.history["accuracy"]
epochs = range(1, len(acc)+1)
val_acc = history.history["val_accuracy"]
plt.plot(epochs, acc, "bo-", label="Training Accuracy")
plt.plot(epochs, val_acc, "ro--", label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

使用自訂的Keras優化器- 使用SGD優化器

- Python程式是使用自訂參數的SGD優化器

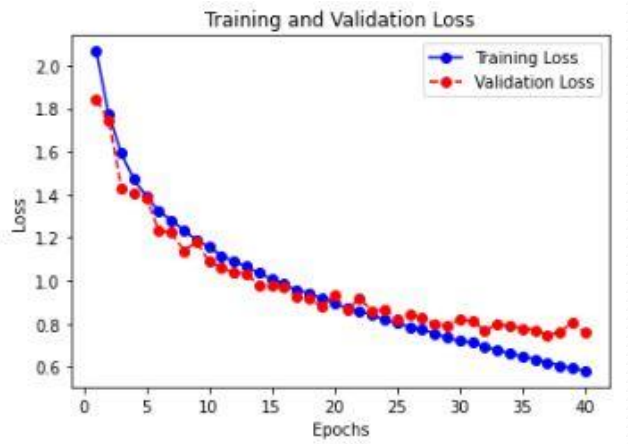
```
1 import numpy as np
2 from keras.datasets import cifar10
3 from keras.models import Sequential
4 from keras.layers import Dense
5 from keras.layers import Flatten
6 from keras.layers import Conv2D
7 from keras.layers import MaxPooling2D
8 from keras.layers import Dropout
9 from tensorflow.keras.utils import to_categorical
10 from tensorflow.keras import optimizers
```

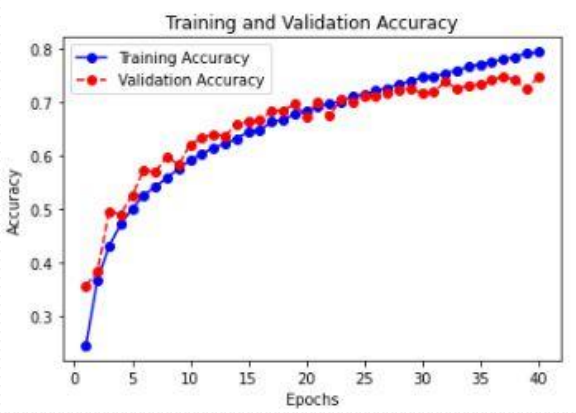
```
38 # 編譯模型
39 opt_sgd = optimizers.SGD(lr=0.05,decay=1e-6, momentum=.09)
40 model.compile(loss="categorical_crossentropy", optimizer=opt_sgd,
41                 metrics=["accuracy"])
```

```
Testing ...
313/313 [==============================] - 1s 3ms/step - loss: 0.7642 - accuracy: 0.7396
測試資料集的準確度 = 0.74
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
dropout (Dropout)	(None, 16, 16, 32)	0
conv2d_1 (Conv2D)	(None, 16, 16, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 512)	2097664
dropout_2 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 10)	5130

```
Total params: 2,122,186
Trainable params: 2,122,186
Non-trainable params: 0
```





```python
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import optimizers

# 指定亂數種子
seed = 10
np.random.seed(seed)
# 載入資料集
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()
# 因為是固定範圍, 所以執行正規化, 從 0-255 至 0-1
X_train = X_train.astype("float32") / 255
X_test = X_test.astype("float32") / 255
# One-hot編碼
Y_train = to_categorical(Y_train)
Y_test = to_categorical(Y_test)
# 定義模型
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), padding="same",
                 input_shape=X_train.shape[1:], activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, kernel_size=(3, 3), padding="same",
                 activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation="softmax"))
model.summary()      # 顯示模型摘要資訊
# 編譯模型
opt_adam = optimizers.Adam(lr=0.001,decay=1e-6)
model.compile(loss="categorical_crossentropy", optimizer=opt_adam,
              metrics=["accuracy"])
# 訓練模型
history = model.fit(X_train, Y_train, validation_split=0.2,
                    epochs=13, batch_size=128, verbose=2)
```

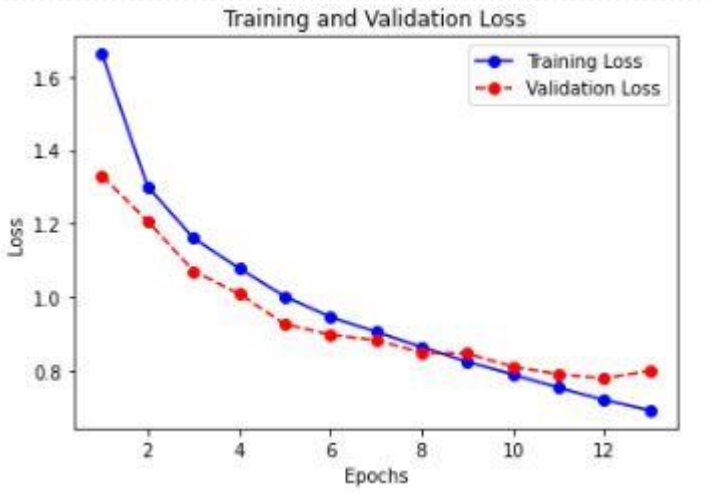
```python
# 評估模型
print("\nTesting ...")
loss, accuracy = model.evaluate(X_test, Y_test)
print("測試資料集的準確度 = {:.2f}".format(accuracy))
# 顯示圖表來分析模型的訓練過程
import matplotlib.pyplot as plt
# 顯示訓練和驗證損失
loss = history.history["loss"]
epochs = range(1, len(loss)+1)
val_loss = history.history["val_loss"]
plt.plot(epochs, loss, "bo-", label="Training Loss")
plt.plot(epochs, val_loss, "ro--", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
# 顯示訓練和驗證準確度
acc = history.history["accuracy"]
epochs = range(1, len(acc)+1)
val_acc = history.history["val_accuracy"]
plt.plot(epochs, acc, "bo-", label="Training Accuracy")
plt.plot(epochs, val_acc, "ro--", label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

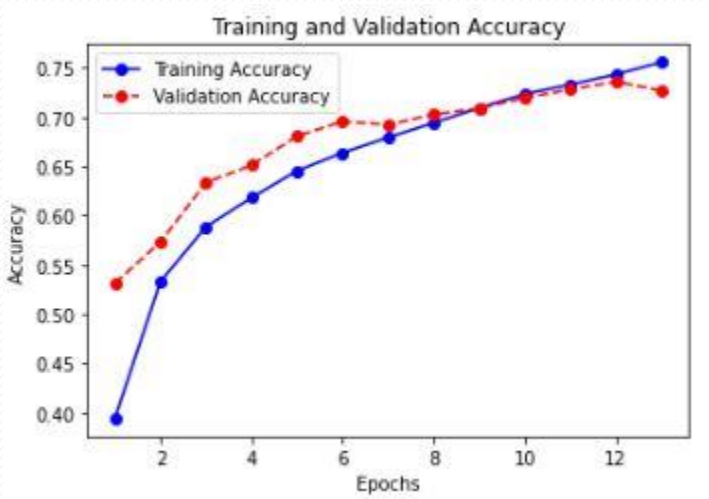
使用自訂的Keras優化器-使用Adam優化器

- Python程式是使用自訂參數的Adam優化器

```
38 #  編譯模型
39 opt_adam  =  optimizers.Adam(lr=0.001,decay=1e-6)
40 model.compile(loss="categorical_crossentropy",  optimizer=opt_adam,
41                            metrics=["accuracy"])
```

```
Testing ...
313/313 [==============================] - 1s 3ms/step - loss: 0.8005 - accuracy: 0.7192
測試資料集的準確度 = 0.72
```





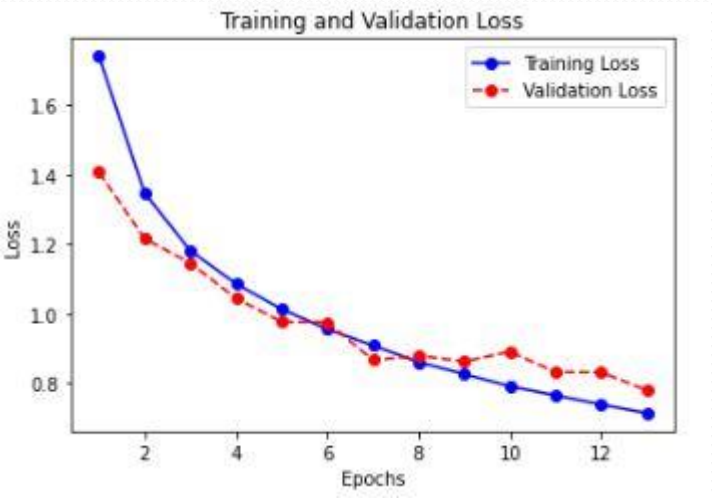
```python
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import optimizers

# 指定亂數種子
seed = 10
np.random.seed(seed)
# 載入資料集
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()
# 因為是固定範圍, 所以執行正規化, 從 0-255 至 0-1
X_train = X_train.astype("float32") / 255
X_test = X_test.astype("float32") / 255
# One-hot編碼
Y_train = to_categorical(Y_train)
Y_test = to_categorical(Y_test)
# 定義模型
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), padding="same",
                 input_shape=X_train.shape[1:], activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, kernel_size=(3, 3), padding="same",
                 activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation="softmax"))
model.summary()      # 顯示模型摘要資訊
# 編譯模型
opt_rms = optimizers.RMSprop(lr=0.001,decay=1e-6)
model.compile(loss="categorical_crossentropy", optimizer=opt_rms,
              metrics=["accuracy"])
# 訓練模型
history = model.fit(X_train, Y_train, validation_split=0.2,
                    epochs=13, batch_size=128, verbose=2)
```

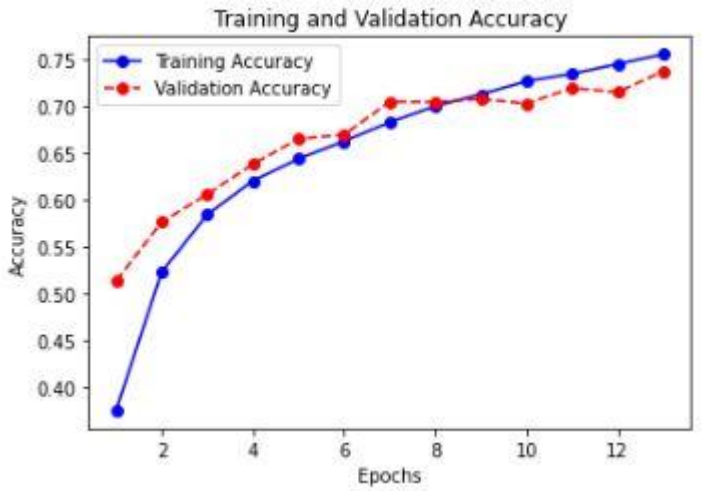
```python
# 評估模型
print("\nTesting ...")
loss, accuracy = model.evaluate(X_test, Y_test)
print("測試資料集的準確度 = {:.2f}".format(accuracy))
# 顯示圖表來分析模型的訓練過程
import matplotlib.pyplot as plt
# 顯示訓練和驗證損失
loss = history.history["loss"]
epochs = range(1, len(loss)+1)
val_loss = history.history["val_loss"]
plt.plot(epochs, loss, "bo-", label="Training Loss")
plt.plot(epochs, val_loss, "ro--", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
# 顯示訓練和驗證準確度
acc = history.history["accuracy"]
epochs = range(1, len(acc)+1)
val_acc = history.history["val_accuracy"]
plt.plot(epochs, acc, "bo-", label="Training Accuracy")
plt.plot(epochs, val_acc, "ro--", label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

使用自訂的Keras優化器-使用RMSprop優化器

- Python程式是使用自訂參數的RMSprop優化器

```
Testing ...
313/313 [==============================] - 1s 3ms/step - loss: 0.7969 - accuracy: 0.7303
測試資料集的準確度 = 0.73
```





End !