臺北市立大學資訊科學系

期末專題計畫書

指導教授：蔡俊明 博士

**使用 TensorFlow 和 Keras 進行 Python 圖像識別**Python Image Recognition with TensorFlow and Keras

研究生：阿力

中華民國一百一一年十四月十九日

目錄

[第一節 介紹 3](#_Toc122899569)

[第二節 參考方法 5](#_Toc122899570)

[第三節 實驗影像 8](#_Toc122899571)

[第四節 結果 10](#_Toc122899572)

[第四節 參考文獻 11](#_Toc122899573)

# 第一節 介紹

## 1-1背景

TensorFlow 和 Keras 最常見的用途之一是圖像識別和分類。 如果您想學習如何使用 Keras 進行圖像分類或識別，本文將教您如何操作。

## 1-2 動機

圖像分類是指計算機視覺根據圖像的視覺內容對圖像進行分類的過程。 例如，可以設計圖像分類算法來確定圖像是否包含貓或狗。 儘管目標檢測對人類來說是微不足道的，但可靠的圖像分類仍然是計算機視覺應用中的一個挑戰。

## 1-3目的

在本篇文章上主要的目的是將學習如何在 Python 中使用 Tensorflow 對 CIFAR-10 數據集（由飛機、狗、貓和其他 7 個對象組成）中的圖像進行成功分類。

第二節 參考方法

1. 提取特徵使用過濾器

A picture containing cargo container, LEGO

Description automatically generated with low confidence

神經網路的第一層接受圖像中的所有圖元。 在所有數據輸入網路之後，圖像會經過不同的濾波器處理，這些濾波器形成對圖像不同部分的理解。 這就是特徵提取，它創建了“特徵圖”。 通過「捲積層」來從圖像中提取特徵，捲積簡單地形成圖像部分的表示。 正是從這個卷積概念中，我們得到了「卷積神經網路」 （Convolutional Neural Network， CNN） 這個術語，它是最常用於圖像分類和識別的一種神經網路類型。 如果你想可視化特徵圖的創建過程，想像一下在黑暗的房間中將手電筒筒照向圖像的過程。 當你在圖像上滑動光束時，你會瞭解到圖像的特徵。 濾波器是網路用於形成圖像表示的工具，在這個比喻中，手電筒筒筒的光就是濾波器。 你手電筒筒筒的光束寬度決定了你一次查看的圖像片段的大小，神經網路也有類似的參數 - 濾波器的大小。 濾波器的大小決定了一次檢查多少個圖元。 在CNN中常用的濾波器大小為3，它涵蓋了高度和寬度，因此濾波器檢查的是一個3 x 3的像素區域。

A picture containing screenshot, rectangle, LEGO, cube

Description automatically generated

虽然滤波器的大小涵盖了滤波器的高度和宽度，但滤波器的深度也需要指定。但是，二维图像如何具有深度呢？

问题在于数字图像以高度、宽度和某个RGB值表示，该值确定像素的颜色，因此"深度"表示图像具有的颜色通道数量。灰度图像（非彩色）只有1个颜色通道，而彩色图像具有3个通道的深度。

这意味着对于应用于彩色图像的大小为3的滤波器，该滤波器的最终尺寸将是3 x 3 x 3。对于每个由该滤波器覆盖的像素，网络将滤波器的值与像素本身的值相乘，以获得该像素的数值表示。然后，该过程对整个图像执行，以获得完整的表示。滤波器根据称为"步幅"的参数在图像的其余部分移动，该参数确定滤波器在计算其当前位置的值后应移动多少个像素。CNN的常规步幅大小为2。所有这些计算的最终结果是特征图。通常，该过程使用多个滤波器执行，这有助于保留图像的复杂性。

1. 啟動函數

在創建圖像的特徵圖之後，表示圖像的值會通過啟動函數或啟動層。 啟動函數接收這些值，這些值由卷積層產生，並以線性形式表示（即僅是一組數位），並增加它們的非線性性，因為圖像本身是非線性的。

為了實現這個目標，通常使用的啟動函數是修正線性單元（ReLU）。

3. 合併層

在啟動之後，數據通過合併層進行處理。 合併層對圖像進行「簡化」：它提取表示圖像的信息並進行壓縮。 合併的過程通過池化操作使網路更加靈活，能夠更好地基於相關函數識別物件和圖像。

當我們觀察圖像時，通常我們關注的不是全部資訊（例如圖像的背景），而只關注我們感興趣的特徵，比如人、動物等。

同樣地，CNN中的合併層會剔除圖像中的無關部分，只保留其認為相關的部分，具體取決於指定的合併層大小。

由於網路需要在關注圖像的最重要部分時做出決策，計算過程假設它只會學習真正代表所研究對象本質的圖像部分。 這有助於防止“過擬合”，即網络過於完美地學習了訓練示例的所有方面，無法將新數據推廣，因為它會考慮到無關的差異。

A picture containing text, screenshot, diagram, number

Description automatically generated

有各種方式可以合併數值，但最常用的是最大池化。 最大池化意味著在一個濾波器（圖像片段）內選擇最大的圖元值。 這樣做可以過濾掉3/4的資訊，前提是使用2 x 2大小的濾波器。

選擇圖元的最大值是為了考慮可能的圖像扭曲，並減少參數數量（圖像尺寸），以控制過擬合。 還有其他合併原則，如平均合併或總和合併，但它們不太常用，因為最大池化提供了更高的準確性。

1. 壓縮

我們的CNN的最後幾層是全連接層，它們要求數據以向量的形式表示以進行進一步處理。 因此，數據需要被「壓縮」到一個長向量或按順序排列的數位列中。

在這個過程中，將原始數據的維度降低，並將其重新排列為一維向量。 這使得數據可以被輸入到全連接層進行處理和分類。 壓縮操作將圖像的空間維度轉換為線性結構，以便網路可以更好地理解和利用數據的關聯性。

第三節 實驗影像

首先，我們需要一個用於訓練的數據集。 在這個例子中，我們將使用著名的CIFAR-10數據集。 CIFAR-10是一個大型數據集，包含超過60，000個圖像，代表著10個不同的物體類別，例如貓、飛機和汽車等。

A picture containing screenshot, collage

Description automatically generated

首先，我們需要導入所需的庫。 在進行實際操作時，您將看到具體的導入方式，但現在只需要知道我們將使用Numpy和與Keras相關的各種模組。

A screenshot of a computer

Description automatically generated with low confidence

準備資料

1. from keras.datasets import cifar10

現在讓我們載入數據集。 我們可以通過指定要載入資料的變數，然後使用load\_data（）函數來完成這個任務：

1. # loading in the data
2. (X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()

為了對數據進行歸一化，我們可以簡單地將圖像值除以255。 為此，我們首先需要將數據轉換為浮點數格式，因為目前它們是整數。

1.# normalize the inputs from 0-255 to between 0 and 1 by dividing by 255

1. X\_train = X\_train.astype('float32')
2. X\_test = X\_test.astype('float32')
3. X\_train = X\_train / 255.0
4. X\_test = X\_test / 255.0

為了準備數據供網路使用，下一步是將其轉換為二進位編碼。

1.# one hot encode outputs

1. y\_train = np\_utils.to\_categorical(y\_train)
2. y\_test = np\_utils.to\_categorical(y\_test)
3. class\_num = y\_test.shape[1]

設計模式

我們已經到達了CNN模型的設計階段。 首先，我們需要確定我們希望在模型中使用的格式。 Keras有幾種不同的格式（計劃）可用於構建模型，但最常用的是Sequential（順序

模型）- 因此我們從Keras中導入它。

創造模式

1. model = Sequential()

我們模型的第一層是捲積層。 它將接收輸入數據並通過捲積濾波器對其進行處理。

1.model.add(Conv2D(32, (3, 3), input\_shape=X\_train.shape[1:], padding='same'))

1. model.add(Activation('relu'))

如前所述，ReLU是最常用的啟動函數，而填充我們將定義為padding = 'same'。

您可以根據自己的喜好調整卷積層的數量，但每個捲積層都會增加計算成本。 請注意，當添加捲積層時，通常會增加過濾器的數量，以便模型可以學習更複雜的表示。

1.model.add(Dense(class\_num))

2.model.add(Activation('softmax'))

最后，softmax激活函数选择具有最高概率的神经元作为其输出值，假设图像属于该类别。

現在，我們已經設計好了要使用的模型，只剩下編譯它。 讓我們指定訓練的時代（epoch）數量以及要使用的優化器。

1. epochs = 25
2. optimizer = 'adam'
3. model.compile(loss='categorical\_crossentropy', optimizer=optimizer, metrics=['accuracy'])
4. print(model.summary())

Print 會給我們些資訊:

1. Results:
3. Layer (type) Output Shape Param #
4. =================================================================
5. conv2d\_1 (Conv2D) (None, 32, 32, 32) 896
6. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
7. activation\_1 (Activation) (None, 32, 32, 32) 0
8. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
9. dropout\_1 (Dropout) (None, 32, 32, 32) 0
10. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
11. batch\_normalization\_1 (Batch (None, 32, 32, 32) 128
12. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
13. conv2d\_2 (Conv2D) (None, 32, 32, 64) 18496
14. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
15. activation\_2 (Activation) (None, 32, 32, 64) 0
16. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
17. max\_pooling2d\_1 (MaxPooling2 (None, 16, 16, 64) 0
18. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
19. dropout\_2 (Dropout) (None, 16, 16, 64) 0
20. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
21. batch\_normalization\_2 (Batch (None, 16, 16, 64) 256
22. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
23. conv2d\_3 (Conv2D) (None, 16, 16, 64) 36928
24. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
25. activation\_3 (Activation) (None, 16, 16, 64) 0
26. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
27. max\_pooling2d\_2 (MaxPooling2 (None, 8, 8, 64) 0
28. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
29. dropout\_3 (Dropout) (None, 8, 8, 64) 0
30. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
31. batch\_normalization\_3 (Batch (None, 8, 8, 64) 256
32. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
33. conv2d\_4 (Conv2D) (None, 8, 8, 128) 73856
34. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
35. activation\_4 (Activation) (None, 8, 8, 128) 0
36. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
37. dropout\_4 (Dropout) (None, 8, 8, 128) 0
38. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
39. batch\_normalization\_4 (Batch (None, 8, 8, 128) 512
40. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
41. flatten\_1 (Flatten) (None, 8192) 0
42. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
43. dropout\_5 (Dropout) (None, 8192) 0
44. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
45. dense\_1 (Dense) (None, 256) 2097408
46. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
47. activation\_5 (Activation) (None, 256) 0
48. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
49. dropout\_6 (Dropout) (None, 256) 0
50. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
51. batch\_normalization\_5 (Batch (None, 256) 1024
52. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
53. dense\_2 (Dense) (None, 128) 32896
54. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
55. activation\_6 (Activation) (None, 128) 0
56. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
57. dropout\_7 (Dropout) (None, 128) 0
58. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
59. batch\_normalization\_6 (Batch (None, 128) 512
60. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
61. dense\_3 (Dense) (None, 10) 1290
62. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
63. activation\_7 (Activation) (None, 10) 0
64. =================================================================
65. Total params: 2,264,458
66. Trainable params: 2,263,114
67. Non-trainable params: 1,344

現在我們可以開始練我們的模式, 為了練它咱們使用 fit() 函數:

1. numpy.random.seed(seed)
2. model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=epochs, batch\_size=64)
3. Epoch 1/25
5. 64/50000 [..............................] - ETA: 16:57 - loss: 3.1479 - acc: 0.0938
6. 128/50000 [..............................] - ETA: 10:12 - loss: 3.0212 - acc: 0.0938
7. 192/50000 [..............................] - ETA: 7:57 - loss: 2.9781 - acc: 0.1250
8. 256/50000 [..............................] - ETA: 6:48 - loss: 2.8830 - acc: 0.1484
9. 320/50000 [..............................] - ETA: 6:07 - loss: 2.8878 - acc: 0.1469
10. 384/50000 [..............................] - ETA: 5:40 - loss: 2.8732 - acc: 0.1458
11. 448/50000 [..............................] - ETA: 5:20 - loss: 2.8842 - acc: 0.1406
13. ...
14. ...
15. ...
17. 49664/50000 [============================>.] - ETA: 1s - loss: 1.5160 - acc: 0.4611
18. 49728/50000 [============================>.] - ETA: 1s - loss: 1.5157 - acc: 0.4612
19. 49792/50000 [============================>.] - ETA: 1s - loss: 1.5153 - acc: 0.4614
20. 49856/50000 [============================>.] - ETA: 0s - loss: 1.5147 - acc: 0.4615
21. 49920/50000 [============================>.] - ETA: 0s - loss: 1.5144 - acc: 0.4617
22. 49984/50000 [============================>.] - ETA: 0s - loss: 1.5141 - acc: 0.4617
23. 50000/50000 [==============================] - 262s 5ms/step - loss: 1.5140 - acc: 0.4618 - val\_loss: 1.0715 - val\_acc: 0.6195
25. End of Epoch 1

發動上面程式的一部分會給我們這些:

1. Epoch 1/25
3. 64/50000 [..............................] - ETA: 16:57 - loss: 3.1479 - acc: 0.0938
4. 128/50000 [..............................] - ETA: 10:12 - loss: 3.0212 - acc: 0.0938
5. 192/50000 [..............................] - ETA: 7:57 - loss: 2.9781 - acc: 0.1250
6. 256/50000 [..............................] - ETA: 6:48 - loss: 2.8830 - acc: 0.1484
7. 320/50000 [..............................] - ETA: 6:07 - loss: 2.8878 - acc: 0.1469
8. 384/50000 [..............................] - ETA: 5:40 - loss: 2.8732 - acc: 0.1458
9. 448/50000 [..............................] - ETA: 5:20 - loss: 2.8842 - acc: 0.1406
11. ...
12. ...
13. ...
15. 49664/50000 [============================>.] - ETA: 1s - loss: 1.5160 - acc: 0.4611
16. 49728/50000 [============================>.] - ETA: 1s - loss: 1.5157 - acc: 0.4612
17. 49792/50000 [============================>.] - ETA: 1s - loss: 1.5153 - acc: 0.4614
18. 49856/50000 [============================>.] - ETA: 0s - loss: 1.5147 - acc: 0.4615
19. 49920/50000 [============================>.] - ETA: 0s - loss: 1.5144 - acc: 0.4617
20. 49984/50000 [============================>.] - ETA: 0s - loss: 1.5141 - acc: 0.4617
21. 50000/50000 [==============================] - 262s 5ms/step - loss: 1.5140 - acc: 0.4618 - val\_loss: 1.0715 - val\_acc: 0.6195
23. End of Epoch 1

請注意，在大多數情況下，您需要有一個與測試集不同的驗證集，因此您需要指定用作驗證集的訓練數據的比例。 在這種情況下，我們只是將測試數據傳遞給驗證集，以確保測試數據被保留，並且沒有用於訓練。 在這個例子中，我們只有測試數據，以使事情更簡單。

現在我們可以判斷程式的工作, 只要導致 model.evaluate():

1. # Model evaluation
2. scores = model.evaluate(X\_test, y\_test, verbose=0)
3. print("Accuracy: %.2f%%" % (scores[1]\*100))

這就是我們的結果:

1. Accuracy: 83.01%

將神經網絡應用於識別驗證碼的實際應用

(CNN）來解決實際日常問題的應用，理論和實驗研究的結果合理地指向了對於圖像識別和分類的實際應用。在識別和分類圖像領域中，最為重要的任務之一是解決驗證碼（CAPTCHA）的問題，特別是當今最流行的 Google ReCaptcha v2。

第三節 結果

當您在Keras中實現了您的第一個圖像識別網絡後，探索模型並觀察其參數如何影響效能是很有價值的。

這將為您提供對於不同模型參數的最佳值的直觀理解。 同時，您應該研究不同的參數和超參數選擇，並在改進您的模型時進行實驗。 在您掌握了這些技巧後，您可以嘗試在其他數據集上實現自己的圖像分類器。

第四節 參考文獻

1. <https://waksoft.susu.ru/2021/04/03/kak-sozdat-klassifikator-izobrazhenij-na-python-s-pomoshhyu-tensorflow-2-i-keras/> - 使用TensorFlow 2和Keras创建Python的图像分类器的方法
2. <https://evileg.com/ru/post/619/-> 使用 TensorFlow 和 Keras 進行 Python 圖像識別Python Image Recognition with TensorFlow and Keras
3. <https://pythonbook.site/python/image-recognition-tensorflow?ysclid=liafkpa9ct378876087> – 使用AI(Tensorflow) 來識別圖像