

Unsupervised Content-based Image Retrieval on SOFM

指導教授：闕志達
b03901166 謝宜展
b03901109 陳緯哲
b03901098 王建翔

動機

隨著機器學習技術的成熟，圖片的辨識與分類變得更加容易，但大多數的辨識還是必須仰賴Supervised Neural Network，工程師必須不斷的告訴電腦這樣的答案對或錯，提供NN更新權重的標準；但資料庫越來越大，Supervised的方式不僅效率低落，也浪費人力，因此我們希望能利用Unsupervised的方式來進行圖像的處理。

我們希望類神經網路能直接依據圖片本身的內容作出分類，取代人力一張圖片一張圖片進行Label的過程，以Google的圖片搜尋為例，我們可以直接輸入一張Google資料庫內所沒有的圖片，即使我們沒有為這張圖片進行Label，搜尋引擎依舊可以判讀圖片的內容，並為我們找出相似的圖片，這樣的圖片搜尋方法被稱為Content-based Image Retrieval。

以下將介紹我們在這學期中，運用SOFM與各種Feature Extraction方法，進行Content-based Image Retrieval的方法與結果。

Self Organized Feature Mapping(SOFM)

- 原理

SOFM的概念是大腦在處理資訊時，處理相同資訊的神經會聚集在一起，因此會形成一個一個聚落(Cluster)，而其延伸在資料處理時，我們可以將N維的資料映射到一個二維平面，進行降維，再觀察聚集在一起的資料，是否有相同特性。
進行SOFM的流程如下：

1. 對每個node的權重進行初始化
2. 從training data中隨機選一筆資料
3. 找出一個Node(BMU, Best Matching Unit)，其與這筆資料之間有著最小的Euclidean Distance

$$Dist = \sqrt{\sum_{i=0}^{i=n} (V_i - W_i)^2}$$

V為該筆training data，W為node之權重

4. 計算更新半徑：

$$\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\lambda}\right)$$

σ_0 為mapping的大小，在 $t = 0$ 時，更新半徑會涵蓋整個map

λ 為 time constant

5. 調整該半徑內所有node的權重，使其更接近input vector，越靠近BMU的node其調整幅度更大，更新公式為：

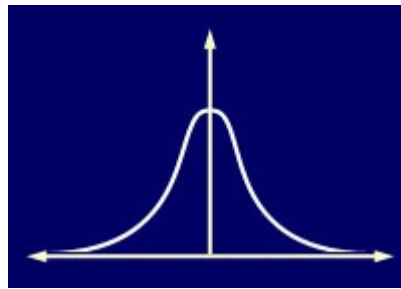
$$W(t+1) = W(t) + \Theta(t)L(t)(V(t) - W(t))$$

$$L(t) = L_0 \exp\left(-\frac{t}{\lambda}\right)$$

$$\Theta(t) = \exp\left(-\frac{dist^2}{2\sigma^2(t)}\right) \quad t = 1, 2, 3, \dots$$

$L(t)$ 為 Learning rate，會隨著時間而降低

$\Theta(t)$ 為隨著與BMU之間的Euclidean Distance改變，其更新幅度的大小，如下圖

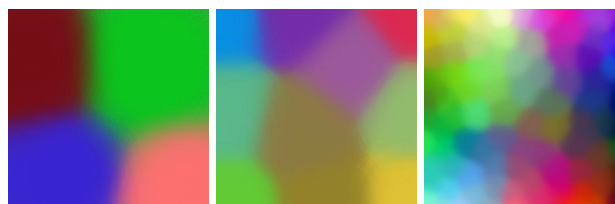


6. 重複 2. 直到跑完所有epoch
 7. 每個node會將離自己最近的64張training data存入自身
 8. 比較每個node，若任兩個Node中，有50個以上相同的data，則這兩個Node會被分類到同一個cluster，依照上述過程，將所有Node進行分類
 9. 輸入test data，找出離他最近的node，輸出該node所屬的cluster中所有的圖片
- 實作

我們試圖將SOFM以視覺化的方式表現出來，除了說明其降維的功效外，也能觀察其將node進行分類並依據種類而聚集的狀態。

首先設定map為 500 x 500 個 node，每個node的weight皆為隨機的三維變數，接著使用我們事先寫好的training data(0到1之間)進行training。

train完後將node的weight放大成0到255間，並以RGB的方式，使每一個Node代表一個Pixel，將所有Node輸出成一張pixel 500 x 500的圖片，如下圖：從左而右分別為4筆、10筆、100筆input data。



- 複雜度

依此設計來講時間複雜度為 $O(nf)$
， n 為node的數量， f 為feature的數量

下面的設計中node數皆為400，feature數則盡量接近1000

參考網站：[Kohonen's Self Organizing Feature Maps](#)

[1]Dian Pratiwi,"The Use of Self Organizing Map Method and Feature Selection in Image Database Classification System."2012.

[2]YangKun,ZhuHong,PanYing-jie,"Human Face Detection Based on SOFM NeuralNetwork."2006.

Feature Extraction之方法與結果

Color Histogram

- 原理

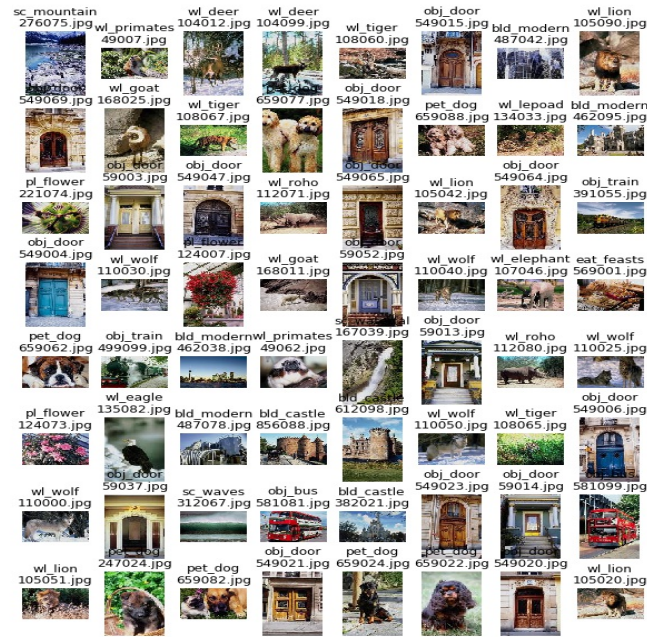
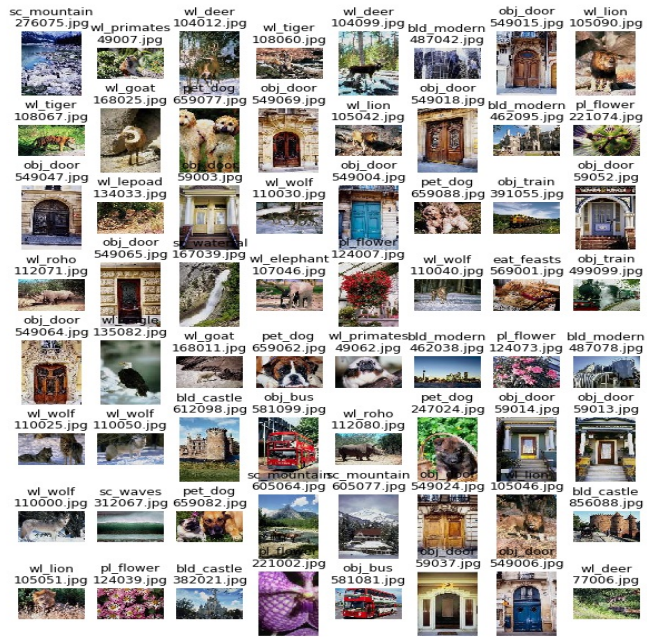
Color Histogram是在許多Content Based Image Retrieval System中被廣泛採用的feature。它所描述的是不同色彩在整幅圖像中所佔的比例，但是對於每一個顏色在該圖片的相對位置卻不夠重視。Color Histogram特別適於描述那些難以進行自動分割的圖像。

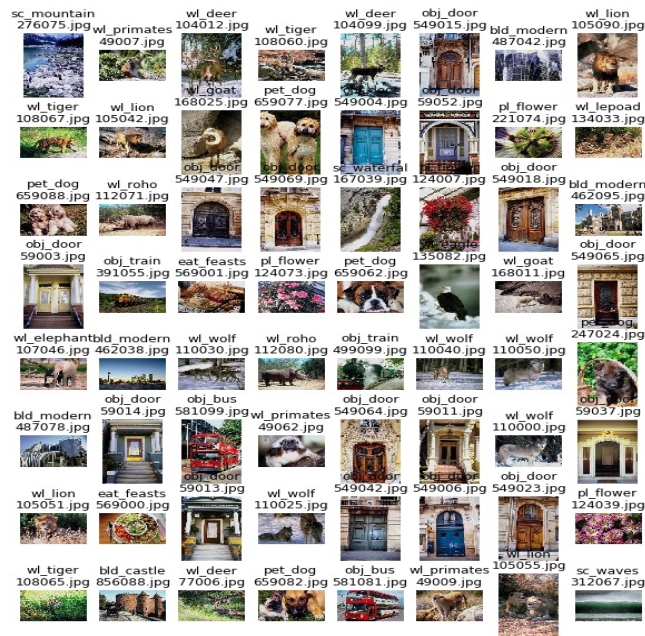
- 方法

先將圖片轉成一個三維的資訊，分別為長、寬、以及RGB，然後對每一格根據RGB的值算出一個0~255之間的值，即可求出Color Histogram。

- 結果

前方大小為120x80的原圖，後方為經過我們系統得到的最相近的64張圖





• 討論

由以上的結果可以看到，把histogram作為主要的feature取法是不夠具有代表性的，因為只考慮到顏色0~255的數值且無法有效的以此作為判斷圖片內物體的依據，所以我們改尋找其他能夠有如圖片內物體的edge的feature取法。

• Reference

Dhanraj R. Dhotre,G. R. Bamnote,"Multilevel Haar Wavelet Transform and Histogram Usage in Content Based Image Retrieval System."2017.

Speeded Up Robust Features (SURF)

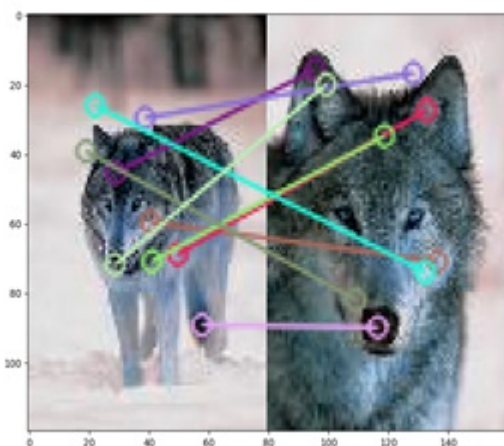
SURF是一種圖像辨識與描述的算法，他使用了海森矩陣的行列式值作特徵點偵測並用積分圖加速運算；SURF 的描述子基於 2D 離散小波變換 響應並且有效地利用了積分圖。在 computer vision 的領域中，可用於物件辨識與3D構件，如下圖，他可以找出圖片的特徵點，當圖片被扭曲時，該特徵點依舊能保持原有的特性，從而將兩張圖片進行比對，如下圖。



也因為SURF能夠辨認出圖片當中的特徵點，而Opencv有支援SURF的運算，因此我們想利用這些特徵點當作圖片的feature，每張圖片取16個特徵點，每個特徵點為64維，再將他放進SOFM的架構內進行training。

- 結果與討論

而從下圖中，可以看到兩張完全不同的圖片所對應到的特徵點，其實是有所差異的，沒有辦法完全對應起來。



當我們仔細去檢視每個Node當中儲存的圖片，發現圖片幾乎是呈現隨機分布，沒有辦法用人眼判讀其分類是否正確，推究其原因後，我們認為我們對SURF所取出的特徵點不夠了解，無法判斷每個特徵點是否能代表其物件，但我們時間不足，也只能繼續尋找下一個方法。

參考網站：[Introduction to SURF \(Speeded-Up Robust Features\)](#)

Brutal

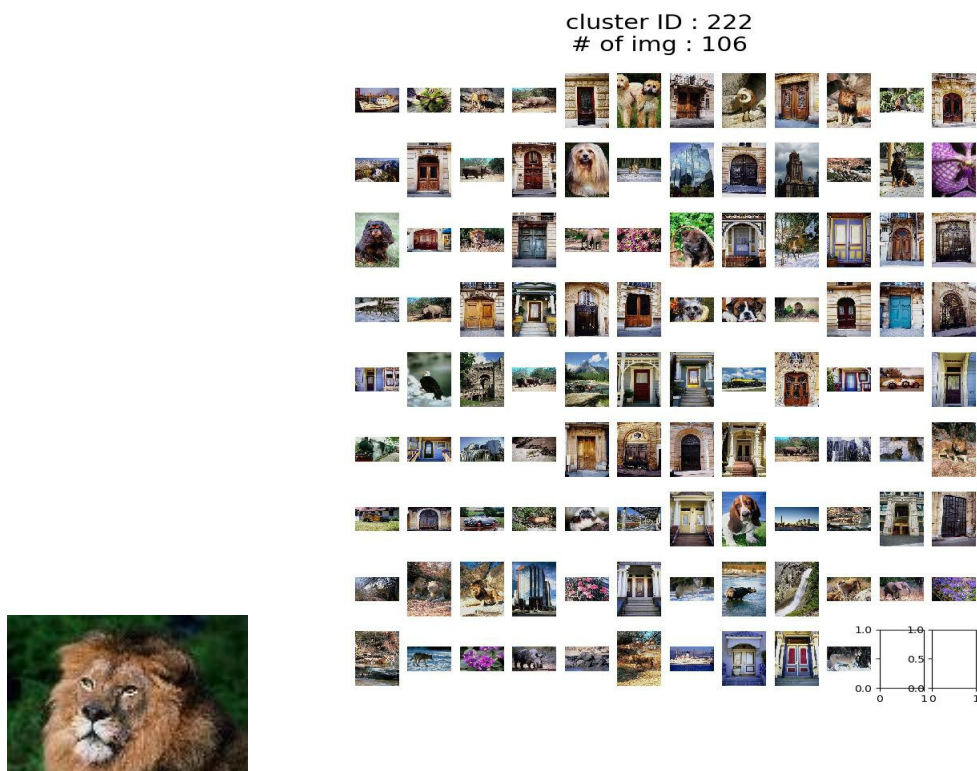
因為SURF的feature無法使用SOFM的方式來做分類，因此我們又另外找方法。這次我們想要直接拿圖片的每個pixel來做為feature，而一個pixel又有RGB三個資訊，因此總共有 $80 * 120 * 3 = 28800$ 個feature。以我們的硬體設備是沒有辦法跑的，因此必須要降維。我們嘗試過PCA和autoencoder兩種方式，最後採用autoencoder，不過兩種降維方式拿來跑SOFM各自產生了不一樣的問題

PCA

在多元統計分析中，PCA是一種降低資料量的技術。主要是用於在feature降維的時候還是能夠保持feature的代表性。這是通過保留低階主成分，忽略高階主成分做到的。這樣低階成分往往能夠保留住數據的最重要方面，PCA不只可用於電腦視覺辨識，亦可用於語音辨識或其他機器學習領域。

- 作法和結果

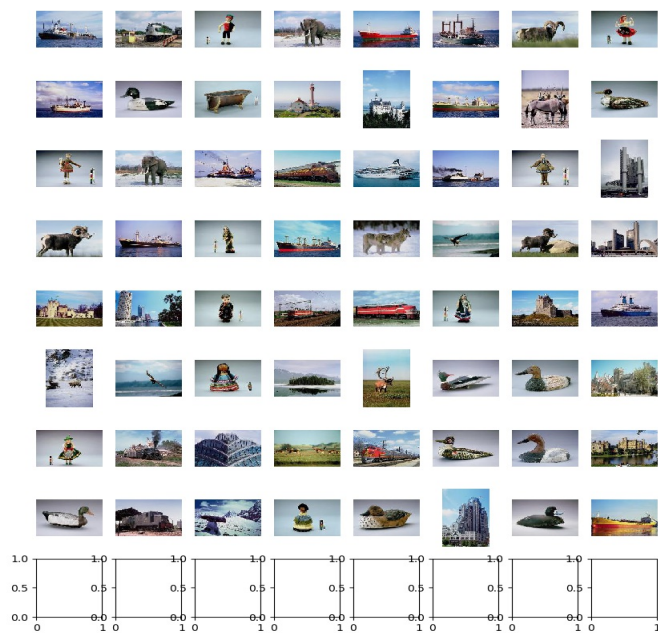
透過呼叫scikit-learn裏頭的PCA套件來將28800維的feature降為1024維，並發現結果卻不如預期，下方為透過PCA把feature降維做出來的結果，前方為原圖、後方為透過我們系統拿出的相近的圖，但是卻發現選出來的圖會被原圖有最高比例的颜色給dominate。



cluster ID : 395
of img : 64



cluster ID : 0
of img : 64



Autoencoder

使用autoencoder的原因是因為比起PCA，autoencoder的降維方式對二維的資料比較有用(畢竟也是用CNN達成)。因此在教授的建議下，我們改使用autoencoder。

首先參考了這個網頁有關使用keras來實作的教學：[Building Autoencoders in Keras](#)。不過他使用的data是MNIST，是黑白的，而我們要encoder的對象是彩色的，因此某些參數可能需要


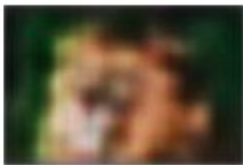








修改，我們最後用了以下的model當作我們的encoder。

```
ACT = 'tanh'
PADDING = 'same'
input_img = Input(shape=(80, 120, 3)) # adapt this if using `channels_first`
x = Conv2D(64, (3, 3), activation=ACT, padding=PADDING)(input_img)
x = MaxPooling2D((2, 2), padding=PADDING)(x)
x = Conv2D(32, (3, 3), activation=ACT, padding=PADDING)(x)
x = MaxPooling2D((2, 2), padding=PADDING)(x)
x = Conv2D(8, (3, 3), activation=ACT, padding=PADDING)(x)
encoded = MaxPooling2D((2, 2), padding=PADDING)(x)
encoder = Model(input_img, encoded)

x = Conv2D(8, (3, 3), activation=ACT, padding=PADDING)(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation=ACT, padding=PADDING)(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation=ACT, padding=PADDING)(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(3, (3, 3), activation='sigmoid', padding=PADDING)(x)
autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='sgd', loss='mse')
```

值得一提的是使用NN做autoencoder的時候CNN的padding要使用same，否則CNN會讓資料數量下降，讓我們encode之後無法decode回來。

另外，之所以要train autoencoder而不是只train encoder就好，是為了看我們的model好不好。而在改了幾次model的架構之後，有了不錯的結果

Original Image	After Autoencoder
	
	
	
	
	

上方的表格中，前三張圖片是不在training資料裡面的，因此可以確定我們train的model是有用的，可以在一定的程度下把圖片encode再decode回相似的原圖

結果

利用autoencoder我們把原本整張圖的 $80 \times 120 \times 3 = 28800$ 個 features 降到 $10 \times 15 \times 8 = 1200$ 個，有效強化了train的效率。然而實作出來的結果甚至比PCA還要糟糕。

我們似乎與做HNN那組一樣，出現了fixpoint的問題。第一次輸出圖片之後發現所有的node幾乎都在圖片是盤子的附近，也就是說100個node全部都是一個cluster，不管丟什麼圖片test都只會跑出盤子。而把所有的盤子刪掉之後，fixpoint變成門的圖片。改了不少參數之後仍然有這個問題存在，因此我們推測encode之後的feature會讓我們的sofm算法的node特別趨近於某些圖片，無法把這些node分開。

但因為encoder的model是train出來的，因此我們無法得知圖片與weight之間的關係，所以這部份實在沒有辦法debug

程式執行環境與指令

- 環境：Python 3.6

- Database : [CoralDB](#)

在使用該Database時，我們刪除了一些如星球、面具、紋路等不常見的圖片，讓我們的过程與結果能更加貼近現實。

- 使用方式
- 取feature :

執行 `extraction.py`，依據想取的方式去改動`main()`中使用的function，各functions的功能如下：

```
load_and_turn_gray() # 輸出灰階的color histogram (feature數:256)
load_and_turn_RGB() # 輸出灰階以及RGB的color histogram (feature數:256*4=1024)
load_brutal() #把所有pixel的資訊extend成一條長長的list (只有一維)
               #格式如下: [[R],[G],[B]] feature數為80*120*3=28800
load_CNN() : #直接輸出圖片的架構(80*120*3的三維陣列，提供autoencoder train)
```

- `python3 final.py [command] command`部分請見以下指令

```
auto_train: train autoencoder model
auto_weight: 使用train好的model encode 所有圖片並輸出成.txt檔
auto_test: decode 圖片 (用來檢視model的效果如何)
train: train SOFM model
test: Input test data
```

結論

我們在學期中時就已經寫出了SOFM，但取feature的方法卻一直沒有著落，因此我們花了半個學期，希望能找個一個兼具效率與準確度的Feature Extraction。

從Color Histogram開始，node會被顏色主導，無法有效辨識物體；SURF在比對圖片的扭曲上有著很好的效果，但拿來比對不同圖片時就完全沒有效果；若直接將圖片每個畫素的RGB當作feature，可能會有不錯的效果，但我們的硬體設備實在不足以支撐這樣的運算量；為了解決運算量的問題，我們使用PCA的降維技術，希望在壓低運算量的同時，也能保持資料不失真，但結果仍然是被原圖的顏色主導，無法準確找出我們需要的主題；而在老師的建議後，我們將PCA改成Autoencoder，利用Autoencoder model將圖片encoder取得feature後在使用SOFM進行train。但儘管Autoencoder model將圖片decode後，其失真率十分的小，Image Retrieval的效果卻十分的糟糕。

Unsupervised 的 Image-Retrieval System如果不使用CNN的話，實在很難做出來。畢竟同樣的一個物體，可能會有不同的形狀以及顏色，因此單靠顏色或是邊界偵測效果非常有限。我認為比較好的方法還是透過辨識將照片中的物體辨識出來，再label上去，但這就不是我們這個學期的主要目標了。

經歷了一學期的嘗試，我們認知到自己在computer vision上還不夠熟稔，因此不斷在Feature Extraction的部分遇到難題，也就是不斷的"Try and Error"，儘管充滿實驗精神，卻依舊無法直搗問題核心，算是令人比較沮喪的地方，但我們學習到了許多取Feature的方法，也認識到computer vision的重要性，這也是這學期除了學到了許多Neural Network的知識外，最大的收穫吧。