CV Final Report—Recognize AR

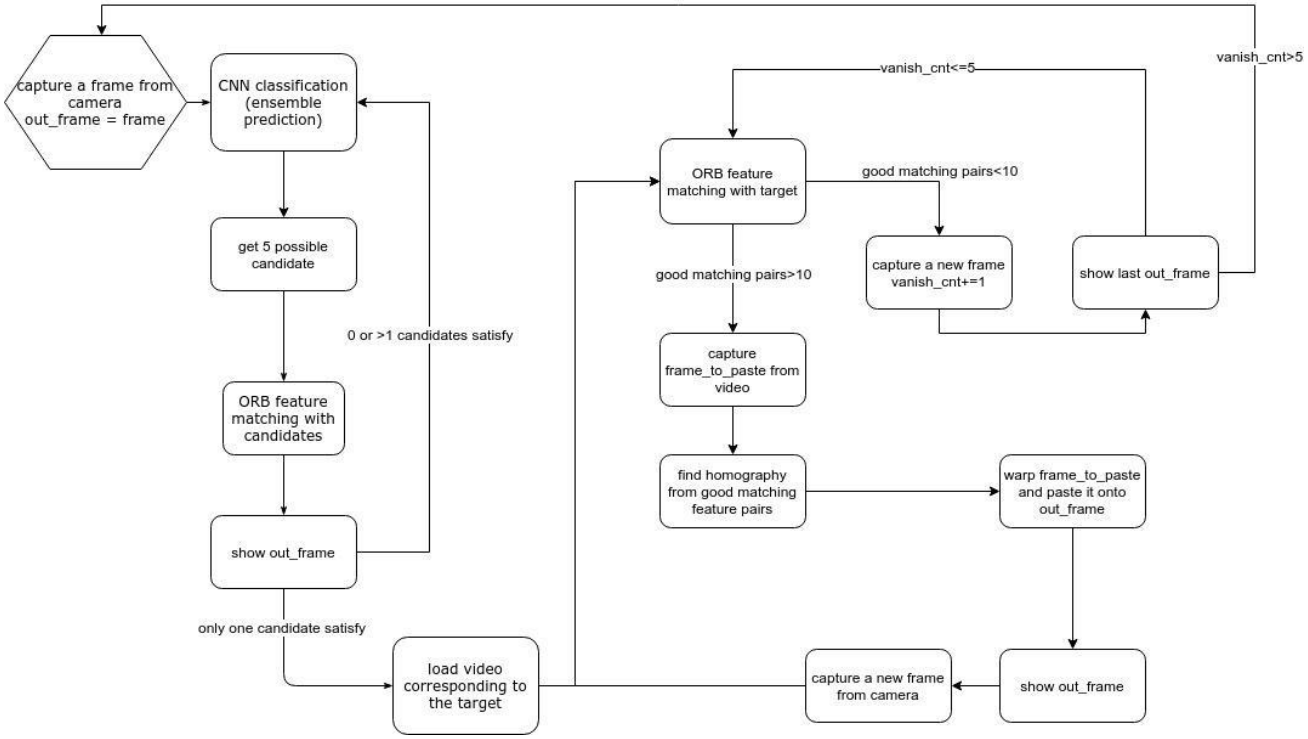
何榮晟、黃聖喻、劉雨東、黃齡萱

Source code and models:
https://www.dropbox.com/s/zhjn05epm1ogvln/final_complete.zip?dl=0

前言:

AR 是最近幾年非常火熱的題材，它在生活中的應用除了很常見的娛樂(譬如說 PokemonGo)之外，也可以用來當作教育、醫療、藝術創作的媒介。一般來說，要在環境中找到指定的目標並且根據此目標貼上對應的 AR 模型，通常都是會事先讓應用程式知道我們所指定的目標是什麼。然而，若一個使用者在資料庫當中放入非常多的目標種類(幾百幾千種)，那麼，每次要找到自己要追蹤的目標為何，甚至是在過程中可能會切換不同的目標，都將花費許多時間。因此，我們希望設計出一款可以自動辨別資料庫中平面(2D)目標的 AR 程式。

實作方法：



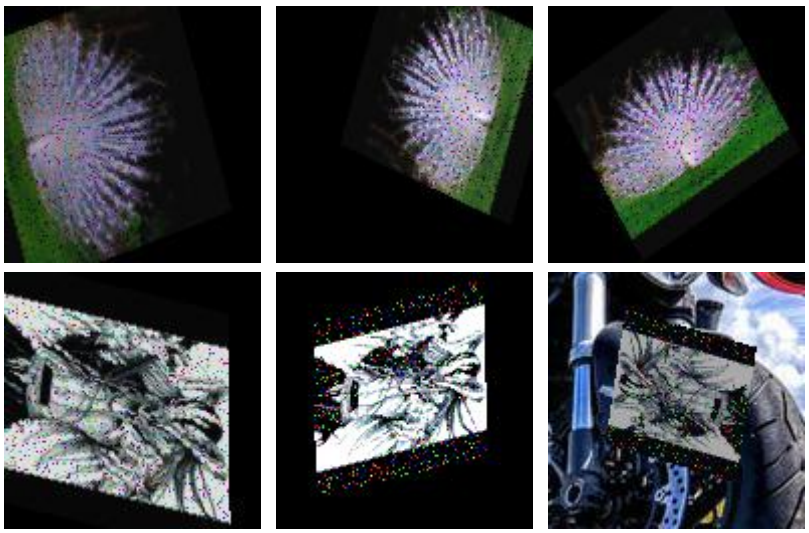
我們程式的執行流程如上圖。主要區分成辨識以及追蹤目標並貼上 AR 影片這兩個部分。辨識的部分，由於目標種類很多，由於我們的目標是實時執行，由於一個 frame 的執行時間有所限制，因此難以進行所有種類的一一比對。於是我們想到了使用深度學習的架構先做一個分類器（ResNet18 架構），將攝影機取得的影像輸入分類器，分類器會輸出五個可能的目標種類(top 5 targets)，接著我們再利用 ORB 特徵匹配的方式一一比對，若是剛好只有一種目標符合我們所指定的匹配條件(匹配點數>某閾值)，則我們相信它就是我們所要找的目標。若沒有符合的對象或是超過一個符合對象同時出現，則重新抓取攝影機影像並重新進行辨識。由於這個方式可以將需要比對的對象從整個資料庫縮小到五個

可能性，執行起來的速度就可以達到實時辨識的目標。

確定目標之後，接下來的工作就是 AR 必須要有的：持續追蹤目標並且貼上對應的模型。我們在這個部分仍舊是沿用 ORB 的特徵匹配，在每一張攝影機捕捉到的畫面中，尋找和先前找到的目標圖像間的特徵匹配，並且利用作業三練習過的 homography 技巧把對應的模型轉到符合目標的角度，並貼上給定的模型（這裡的模型都是 2D 影片，所以可以用 homography 貼上）。

至於如何讓程式懂得更換目標，我們在程式中加上了消失閾值的判斷，當連續 3~5 張攝影機抓取的畫面都無法與前面給的目標匹配，我們便相信目標應該是消失或是到達無法辨識的地步了，這個時候便會返回到最開始的分類器，重新辨識畫面中有什麼目標，這樣就可以做到更換目標的目的。

遭遇的問題：



從一開始試圖做分類器的時候就出現了相當的挑戰，由於對於每一個目標種類來說，唯一的資料就是該目標的原圖，因此只有這樣是不足以做深度學習的訓練的。為了解決這個問題，我們把每一張目標的原圖經過旋轉、加噪、改變色調、縮放、平移並貼到事先準備的背景圖裡，模擬出目標在環境中出現的樣子，如上圖，然後藉此方法把一張目標圖隨機產生成 16 張訓練資料，再加上原圖，總共 17 張。

然而，即便如此，要訓練出足夠好的分類器（前五高分的結果中存在正確答案），這樣仍是不夠的，而且我們還發現每次重新訓練一個分類器的準確率都有不同的差異，為了可以讓準確度更穩定的提升，我們使用 ensemble learning 的方式，直接訓練用四種不同權重初始化（weight initialization）的分類器，分別是 Xavier normal, Xavier uniform, Kaiming normal, Kaiming uniform，然後把四種分類器得分數相加之後再找出前五名，至此，我們的分類器就可以給出相對準確很多的結果了。

接著，在追蹤的部分，我們一開始是使用 SIFT 的找出特徵點，也嘗試了 SURF 的方法，並進行特徵匹配來尋找目標在畫面中的位置。然而，這個方法執行的速度實在太慢，於是我們嘗試利用 optical flow 的方法直接追蹤目標的移動。optical flow 的方法雖然可以快速地計算特徵點的下個位置，但是當特徵點在圖中移動量太大時就會追蹤不到，其穩定性比起 SIFT 的特徵匹配來的弱許多，我們推測可能是因為我們的攝影機影片品質不夠好（攝影機本身是定焦，目標很

容易就離開焦距因而變得模糊）。後來也嘗試利用 opencv 內建的 CSRT tracker，分別追蹤目標的四個頂點，建立四個方框，並且將每一張捕捉影像所追蹤到的方框中心點當作新的頂點座標，但是這個 tracker 在鏡頭有旋轉的情況下沒有辦法很精準的框住目標，會稍微偏移一些，因此對於需要精準 tracking 並以其結果來計算邊界點的這個要求沒有辦法達到。最後我們試著回過頭尋找更快速的特徵匹配方式，最後才選用 ORB 來實作。

討論：

優點：

1. 我們的架構雖然有使用到深度學習模型，但是整體在訓練上十分快速，從資料前置處理到四個分類器都訓練完畢只要約五分鐘，對於使用者來說十分友善，可以即時丟入資料後很快取得分類器。
2. 由於在追蹤目標時使用的是特徵匹配，我們的程式在 AR 物件的穩定性表現十分良好，抖動不明顯且很少出現比較大的偏誤，即便我們在測試以及拍 demo 影片時都是使用品質沒那麼好的網路攝影機，目標靠近鏡頭時會變得模糊，仍然有不錯的成效。
3. 可以做到實時的追蹤以及 AR 物件貼圖

缺點：

1. 整體執行的 frame rate 還可以更提升，當前大約 10-20fps
2. 做特徵匹配以及 homography 都是十分耗費運算資源的，我們曾經嘗試多開執行緒來同時辨認以及追蹤兩個以上的目標，但是由於光是執行一個就會占用 60~70%的 cpu，因此以失敗告終，或許還能再優化執行過程。
3. 當目標物在畫面中佔的比例不同時，要將模型以 homography 處裡過後再貼上就會有快慢的差別，因此若是將目標不斷前後移動，就可以發現 frame rate 有可見的改變，無法很一致。

參考文獻：

1. ResNet: https://arxiv.org/pdf/1512.03385.pdf
2. Ensemble learning : https://medium.com/@chih.sheng.huang821/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-ensemble-learning%E4%B9%8Bbagging-boosting%E5%92%8Cadaboost-af031229ebc3
3. Data augmentation: https://medium.com/@thimblot/data-augmentation-boost-your-image-dataset-with-few-lines-of-python-155c2dc1baec
4. SIFT : https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
5. SURF : https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html
6. brute force matcher : https://opencv-python-

tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html

7. ORB : https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_orb/py_orb.html?highlight=orb
8. Tracking :https://pysource.com/2018/06/05/object-tracking-using-homography-opencv-3-4-with-python-3-tutorial-34/
9. Homography :https://docs.opencv.org/3.4.1/d9/dab/tutorial_homography.html
10. opencv tracker :https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/?fbclid=IwAR3Jpg3DQX7Euvo5Do55wPOcCV9AD4LQews56QR1DSJgQMFULRpRKrc-_bU