Machine Learning HW5 Report

學號：B05901111　系級：電機三　姓名：陳建成

1. (1%) 試說明 hw5_best.sh 攻擊的方法，包括使用的 proxy model、方法、參數等。此方法和 FGSM 的差異為何？如何影響你的結果？請完整討論。(依內容完整度給分)

最後我繳交上去的最高成績仍然是用 FGSM 達成的，proxy model是Keras的ResNet50，參數則是使用Keras本身pretrained的參數。在這之前，我分別使用Keras試過MI-FGSM、PyTorch嘗試FGSM、MI-FGSM即DeepFool，但是攻擊的結果要不然是L-inf. Norm過大（超過strong但小於simple）才能達到 0.880，要不然就是直接跑不出來。除了最後一次是ResNet50當proxy model，其他都是先用VGG16當proxy model測試code結果的可行性。以下分別作討論：
（以下提到的每個proxy model都是用那個套件提供的pretrained當參數，例如Keras就是keras_application提供的，PyTorch就是torchvision提供的）

(1) keras_application VGG16 ← Keras FGSM
這個是最一開始嘗試的，但一開始是連early baseline都過不了的，後來發現狀況是出在攻擊後的後處理上，以及跑model發現愈跑愈慢的狀況。到最後兩天又徹底放棄PyTorch時才發現可以用`K.clear_session()`來加速處理。（原本是用Keras，後來改用TF Keras不知道為什麼有改善）
**Result： Success Rate = 0.880 L-Infinity = 22.8850**

(2) torchvision VGG16 ← PyTorch DeepFool
我本來主要是想拿這個當Best的（還特別去找了論文），然後原先也是想用Keras實作，但是Keras的速度真的讓人受不了，所以我直接用PyTorch搭了一個。然而不知道是怎樣中間的w的norm會變成0，就算加上小小的ε結果也還是會爆掉。所以只能把跑出來的結果放在這邊，總之是失敗了。
**Result： Success Rate = 0.000 L-Infinity = 1.5150**

(3) torchvision VGG16 ← PyTorch FGSM
因為原本Keras跑得實在太慢了，所以我決定直接先改用PyTorch。（那個慢的狀況是前面一分鐘10張，後面一小時3張這樣……）於是我先嘗試用基本的FGSM卻發現不是很理想……
而且那時候local端自己估計正確率時，把6個model代進去似乎都有0.8以上的成功率。但後來才知道我不應該拿labels.csv當ground truth（用attack前的image去predict就得到接近server端的結果了）
**Result： Success Rate = 0.265 L-Infinity = 5.0000**

(4) torchvision VGG16 ← PyTorch MI-FGSM

既然已經要回去重新試FGSM了，於是就連MI-FGSM也試了一遍
**Result： Success Rate = 0.300 L-Infinity = 5.0000**

(5) TF keras_application ResNet50 ← TF Keras FGSM **「Best」**
因為後來在比對labels.csv與predict出來的結果後，發現PyTorch ResNet50的結果與csv完全一致，因此估計可能是ResNet50（那時候，而且後來還有聽到一些風聲），所以最後用來reproduce跟當best的都用它來當proxy
**Result： Success Rate = 0.500 L-Infinity = 4.9500**

(6) TF keras_application ResNet50 ← TF Keras MI-FGSM
因為我一直想嘗試DeepFool卻無法成功跑出有意義的數據，因此聽說同學用MI-FGSM遠比DeepFool效果好，於是回去試momentum-based iterative FGSM。
**Result： Success Rate = 0.500 L-Infinity = 5.0000**

以上是我嘗試過的所有組合，然而似乎跟個人coding能力有相當大的關係，造成前面一直嘗試不出可以 succ. rate跟 L-inf norm都過strong的結果。於是最後也只好選擇比較好過的L-inf過strong而已，而目前DeepFool我認為是實作出錯了才造成結果極差，但目前一直找不出原因。所以最後以FGSM當作best上傳。

這邊簡述一下DeepFool的方式。簡言之是藉由在高維空間中的會被判斷為某一個class的凸多面體（就是某一些區域的點會被判斷為正確的class）找出移動某一個norm會使得此點脫離此區域但是移動距離（論文中使用L-2 norm比較）最小的perturbation 加在原圖上，重複此過程直到判斷為其他class為止。

2. (1%) 請列出 hw5_fgsm.sh 和 hw5_best.sh 的結果 (使用的 proxy model、success rate、L-inf. norm)。

hw5_fgsm.sh: Proxy model: ResNet50 (keras),
Success Rate = 0.500, L-Infinity = 4.9500

hw5_best.sh: Proxy model: ResNet50 (keras),
Success Rate = 0.500, L-Infinity = 4.9500

3. (1%) 請嘗試不同的 proxy model，依照你的實作的結果來看，背後的 black box 最有可能為哪一個模型？請說明你的觀察和理由。

依照labels.csv，目前唯一觀察到完全吻合的預測，只有torchvision ResNet50的

模型，由此model甚至（在deadline後才用MI-FGSM）得到了

Success Rate = 0.850, L-Infinity = 5.0000

這樣快可以過strong的結果，但是因為best還是用Keras實作的，因此這裡以Keras的model互相比較。

Proxy model: ResNet50 (keras),

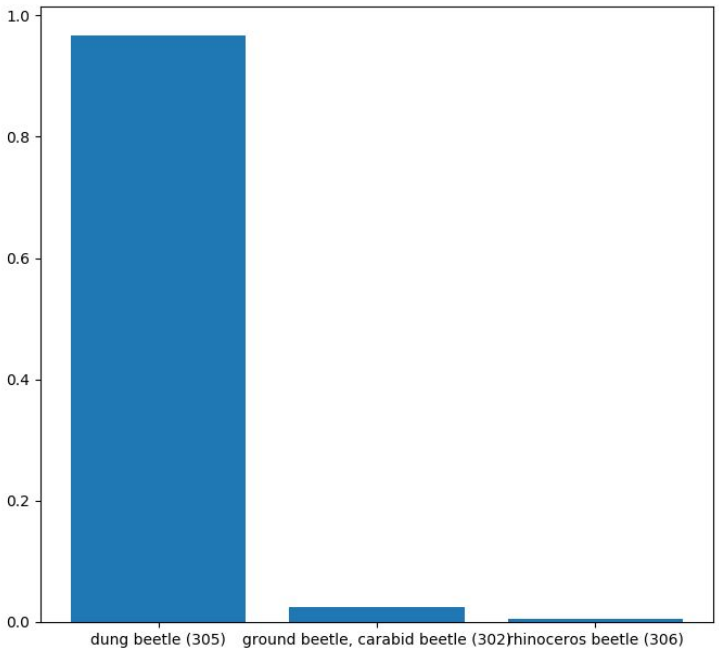
Success Rate = 0.500, L-Infinity = 4.9500

Proxy model: VGG16 (keras),

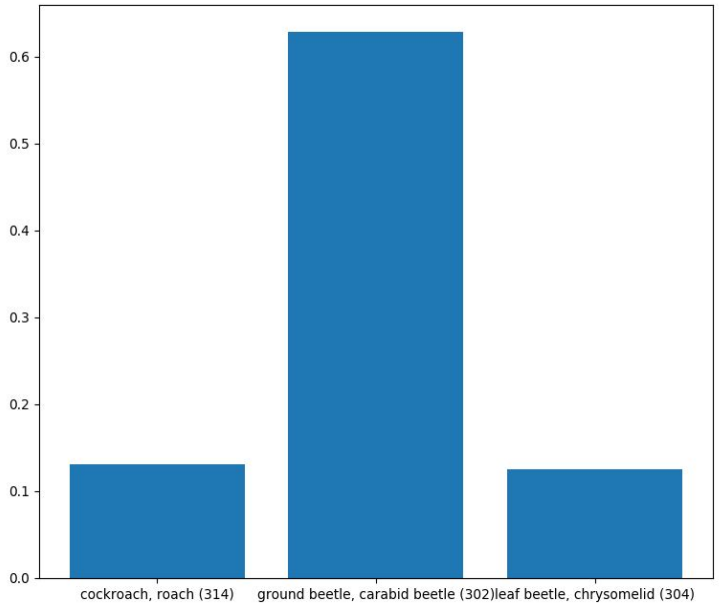
Success Rate = 0.495, L-Infinity = 4.9750

由此結果便可以證實背後的proxy model應該跟ResNet50比較接近。

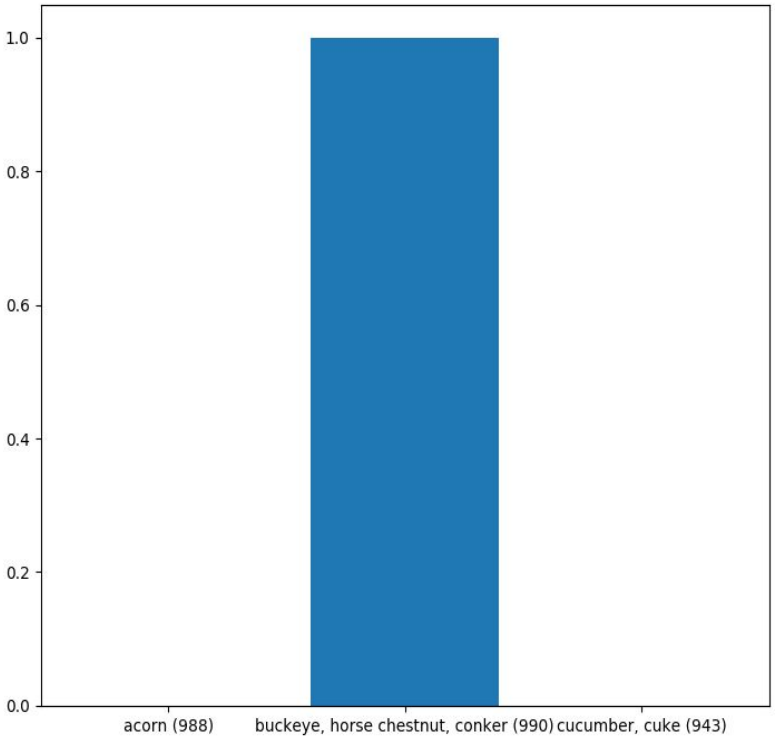
4. (1%) 請以 hw5_best.sh 的方法，visualize 任意三張圖片攻擊前後的機率圖 (分別取前三高的機率)。

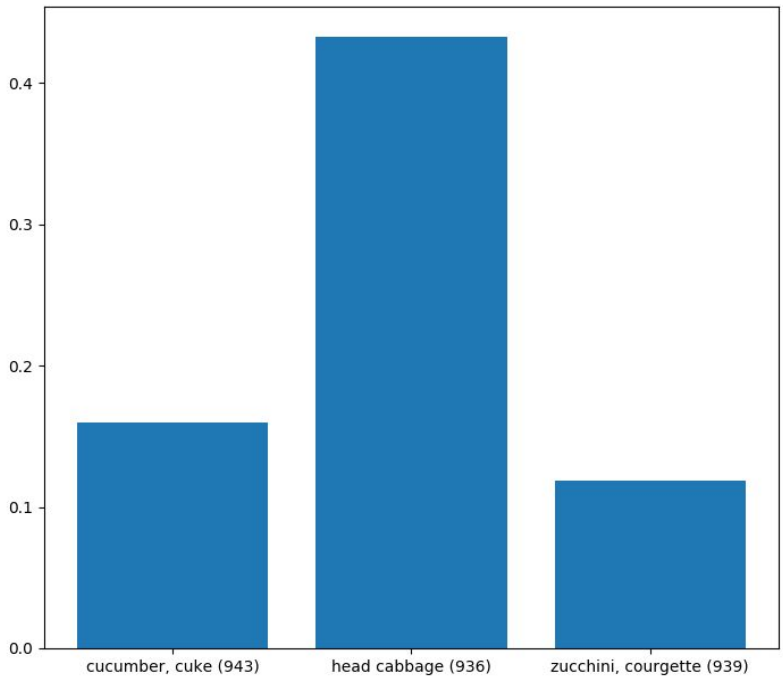



000.png :
dung beetle (305) 96.73%
→ ground beetle, carabid beetle (302) 62.85%

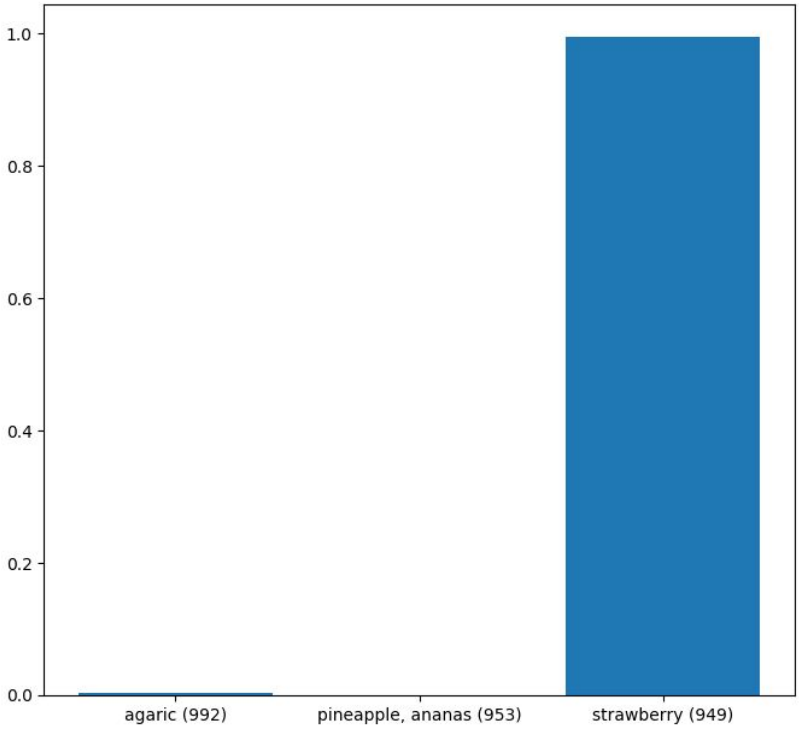


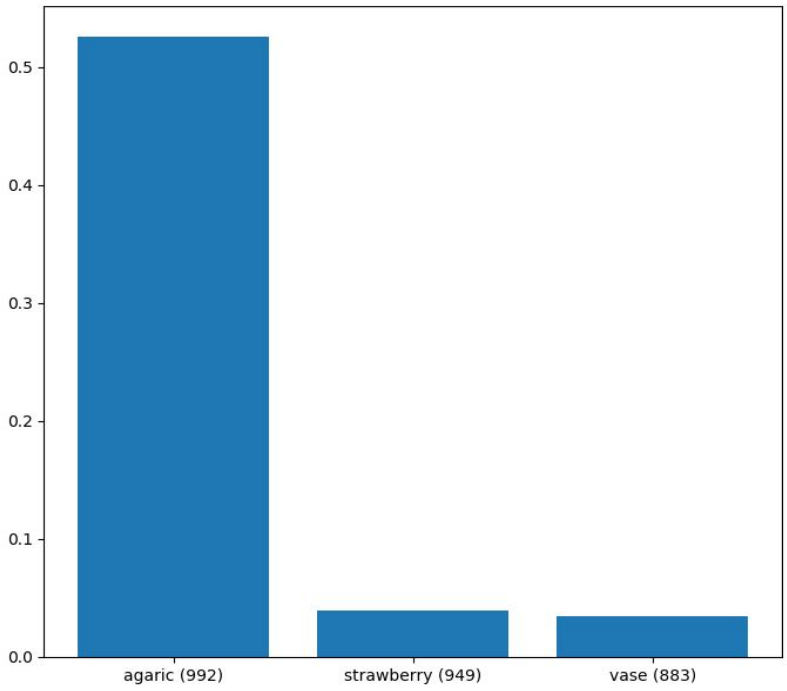
005.png : buckeye, horse chestnut, conker (990) 99.97%
→ head cabbage (936) 43.27%



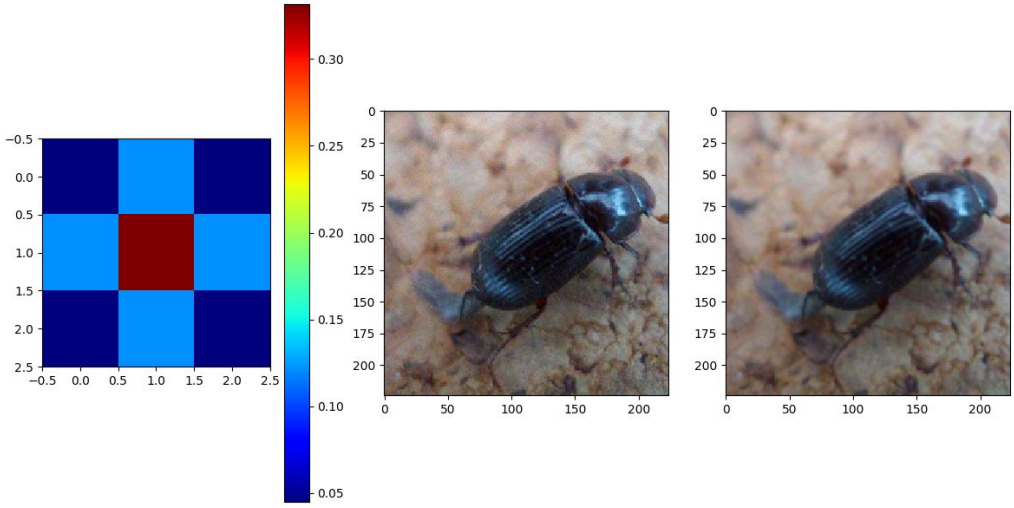
006.png : strawberry (949) 99.55% → agaric (992) 52.56%




5. (1%) 請將你產生出來的 adversarial img，以任一種 smoothing 的方式實作被動防禦 (passive defense)，觀察是否有效降低模型的誤判的比例。請說明你的方法

，附上你"防禦前後"的 success rate，並簡要說明你的觀察。另外也請討論此防禦對原始圖片會有什麼影響。
這次我實作的是Gaussian Filter（參考網站為 https://medium.com/@bob800530/python-gaussian-filter-%E6%A6%82%E5%BF%B5%E8%88%87%E5%AF%A6%E4%BD%9C-676aac52ea17），
然後調整其Gaussian的標準差為 $\sqrt{0.5}$ 時實作出來的防禦，假定server端的模型為torchvision ResNet50 model，並比較原先攻擊前與攻擊後的圖片，結果為
防禦前：Success Rate = 0.500, L-Infinity = 4.9500
防禦後：Success Rate = 0.365, L-Infinity = 4.9750
假定server端的為ResNet50的理由與前面敘述相同，是因為其predict原始圖片結果與labels.csv完全一致，然後攻擊後的圖片比對與server端最為接近。
由此可發現的確有對攻擊的成功率起到降低的作用。
另外我將防禦後攻擊前後的圖片上傳到server得到的結果為
防禦後攻擊前：Success Rate = 0.570, L-Infinity = 79.3800
防禦後攻擊後：Success Rate = 0.085, L-Infinity = 76.7950




原始圖片在經過了smoothing之後，可以看出有變得比較模糊，加上跟原本的圖片不太一樣，所以上傳上去的L-Infinity norm也有變大。
或許是文件的壓縮可能不太明顯，但是右邊大約可以看到有些霧霧的感覺。同時附上做filtering的卷積核作為參考比對。