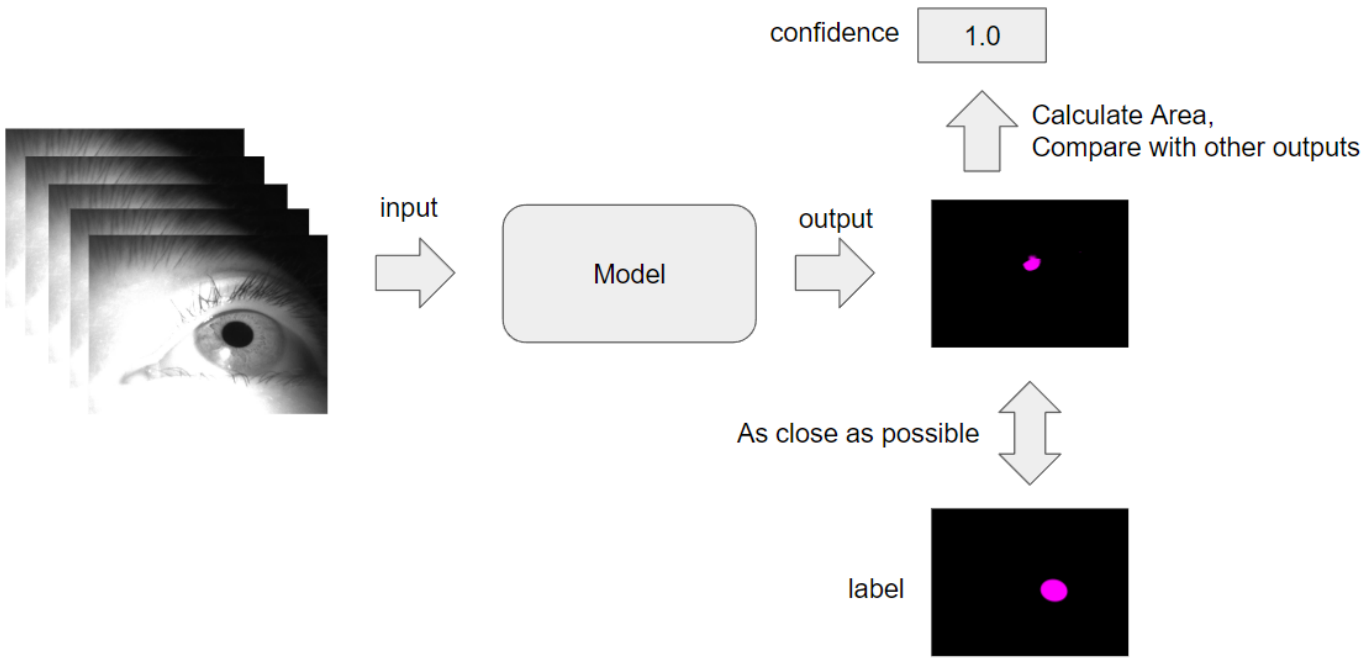
Gazin Final Project Report — Pupil Tracking

Pipeline

1. Concat 5 Images as input data
2. Model training
3. Supervised learning with labels
4. Predict Pupil masks
5. Generate confidence value(by comparing predicted pupil area with others)



Pupil Tracking Method:

我們認為此期末專題可以拆成下列兩個子問題，第一是預測瞳孔在圖片中的位置，
第二為confidence value的預測，也就是我們認為該瞳孔存在該圖片的機率。
而此二問題一個在預測位置，一個為二元分類，是兩個不同類型的子問題，因此一開始我們決定用兩個不同的模型進行預測與學習。

此外，由於本組運算資源比較稀缺，因此我們做了比較多傳統方法(非ML)的實驗來增進結果，就沒有去嘗試很多更深、更複雜的神經網路架構。

1.Pupil Image Generator

為了產生圖片，我們決定訓練一個輸入輸出皆為圖片的Deep Learning Model來解決問題。
我們第一時間想到的是AutoEncoder的架構，因為它的輸入輸出也是相同大小的圖片，所以只要讓模型改成預測瞳孔的label，輸入輸出就對上了，可以用來解看看這個問題。
AutoEncoder的架構為一個編碼器(encoder)直接接到一個解碼器(decoder)。其中encoder可以擷出取圖片中的特徵(feature)，所以我們一開始有試過把encoder的輸出直接接到另一個二元分類器(binary classifier)用來預測confidence，這樣就能將兩個模型一起train，節省時間與參數。雖然因為未解決的overfit的問題，我們最終沒有使用這個binary classifier來預測confidence，但AutoEncoder的架構就保留了下來。

這邊我們在做training時刪掉了沒有label上沒有瞳孔的圖片，只留有label的圖片進行training，希望模型能專注在標出圖片中的瞳孔。

2.Confidence Value Generator:

前面提到，我們一開始產生confidnence value時一樣是用deep learning的方式去產生結果，但我們在該方法的表現不佳，且輸出多不是單純的0或1，在結果預測上會有較多誤差，也因此棄用該模型。
我們認為可以直接分析瞳孔標註模型所預測的瞳孔圖片，最後採用頗為直觀的方法如下:
Step 1:

```
def get_avg(data_path, root):
  data_len = len(data_path)
  area = np.zeros(shape=data_len)
  for i in range(data_len):
    img = cv2.imread(root + str(i) + ".png")[:, :, 0]
    area[i] = np.sum(img>128) * 255
  avg_area = np.sum(area) / data_len
  for i in range(data_len):
    if area[i] < avg_area * 0.5:
      area[i] = 0
  return area, avg_area
```

由於會有閉眼、瞳孔面積過小的圖片，該類圖片無法辨別瞳孔位置。因此我們將model產生的圖片每個sequence去算出平均的瞳孔面積，接著將小於0.5*avg_area的圖片area直接設為0回傳，也就是認定為"沒有瞳孔"的圖片。
Step 2:

```
def filter(area, avg_area):
  window_len = 5
  area_len = area.shape[0]
  result = np.zeros(shape = area_len)
  for i in range(window_len//2):
    if(area[i] > avg_area):
      result[i] = 1.0
    if(area[-i-1] > avg_area):
      result[-i-1] = 1.0
  for i in range(window_len // 2, area_len - window_len // 2):
    window = area[i-window_len//2:i+window_len//2+1]
    local_max = np.max(window)
    if local_max > 0:
      if(area[i] > local_max * 0.5):
        result[i] = 1.0
  return result
```

第二，由於我們預測出來的圖片經過觀察，發現會有缺漏，也就是低估瞳孔average的面積(即使瞳孔面積在整個sequence中都保持不變，上一步算出的average還是可能被一些閉眼的圖片給拉低)，因此在第二步進行嚴格的篩選，將先前瞳孔>average*0.5的圖片，再做一次分類，以避免太容易將圖片標為1。
作法：將某張圖及其前後各2張，共5張瞳孔面積的最大值當作local的瞳孔大小，如果大於此面積的0.5倍，就將confidence value標為1，反之就標為0。

Step 3:

```
def post_process(result, txt_path):
  result_len = result.shape[0]
  ans = np.zeros(shape=result.shape)
  window_len = 5
  copy_data = np.zeros(shape = result_len+2*(window_len//2))
  for i in range(result_len):
    copy_data[i+window_len//2] = result[i]
  for i in range(window_len//2):
    w_l = window_len//2 - i
    copy_data[i] = copy_data[i+w_l*2]
    copy_data[-i-1] = copy_data[(-i-1)-w_l*2]

  result_len = copy_data.shape[0]

  for i in range(window_len//2, result_len-window_len//2):
    window = copy_data[i-window_len//2:i+window_len//2+1]
    result[i-window_len//2] = np.median(window)

  with open(txt_path, 'w') as f:
    for cnt, item in enumerate(result):
      f.write(str(item)+"\n")
```

最後這步則是抓出某張圖及其前後各兩張，共5張圖的confidence value，並取出其中位數填入，選定前後5張值的中位數來代表該圖片。會想這麼做是因為多數圖片中眼睛的動作有時間上的連續性，也就是睜眼會繼續睜眼，眨眼的瞬間也連續，而閉眼也會連續，因此若能參酌鄰近圖片的confidence value來修正輸出，就能避免在一串閉眼圖中突然出現出現1，或在一串睜眼圖中輸出0。

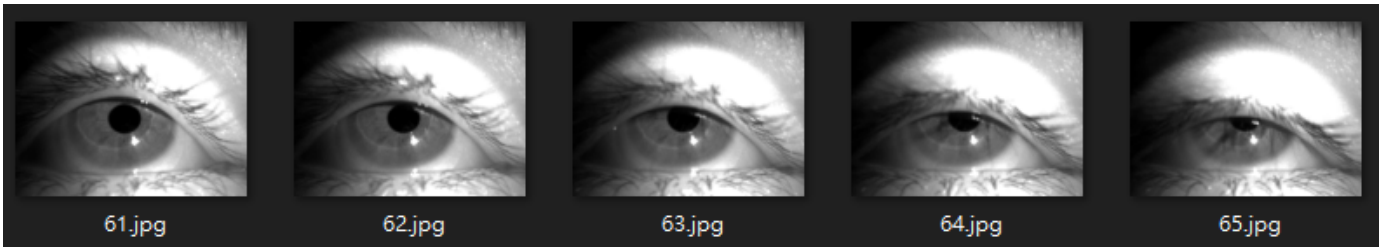
Experiment:

Step1.Oversample the 0-labeled images

雖然我們最終沒有使用NN based binary classifier來預測confidence，但我們一開始還是在改善其表現上做了一點努力。
在訓練binary classifier時，由於在training data中有資料不平衡的狀況，也就是沒有瞳孔的資料太少，大部分都是有瞳孔的資料。這可能導致我們的模型只學到有瞳孔的資料，進而在結果中常常會判斷錯誤(模型會特別喜歡輸出1，也就是有瞳孔)。經過資料分析後，發現有瞳孔的資料：沒瞳孔的資料大約是10:1。因此在訓練中，我們對沒瞳孔的資料進行oversample取10次，希望能讓模型學到沒有瞳孔的資料。在增加此步驟後，模型偏好輸出1的情況獲得了改善。

Step2.Combine 5 images as model input

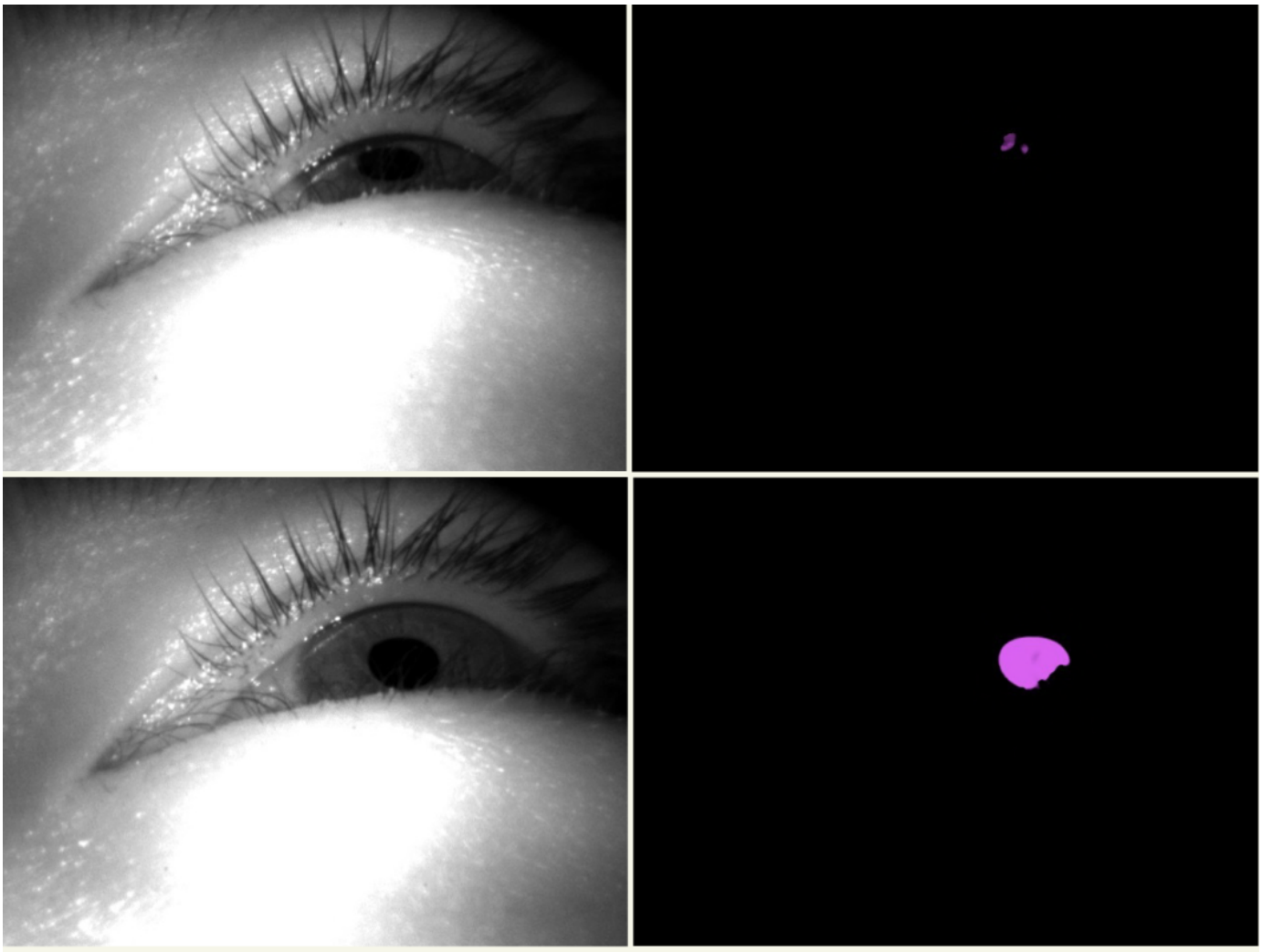
原先對單張圖進行學習並輸出瞳孔位置時，若遇到眨眼，在瞳孔面積接近一半的情況下，模型容易有異常輸出，沒辦法做準確的預測(因為模型的確是沒有被眼皮遮蓋住的瞳孔的資訊，只能"腦補")。而我們認為眨眼是連續動作，經過對資料的觀察與分析，發現約三張圖可以完成眨眼，因此在前後各多取兩張圖，共五張圖做為我們判斷中間那張圖瞳孔位置的依據，讓模型的輸出更健全。



(示意圖，此為預測63.jpg的瞳孔位置時的輸入，這5張圖會被concat起來變成一張5個channel的圖片丟進模型，這樣模型就有機會能根據61.jpg來得知完整的瞳孔形狀)

Step3.Increase the training data with eyelash on it

與Oversample相同的道理，由結果發現，當有睫毛擋到瞳孔時，我們輸出的瞳孔圖常會有缺陷，如圖

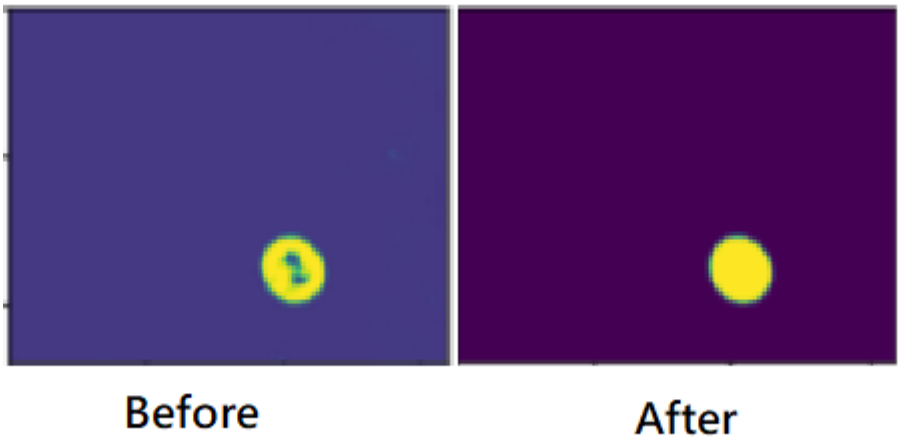


推測是因為有睫毛擋到瞳孔的資料太少，因此我們對有睫毛的資料進行data augmentation，而為了不讓瞳孔移出圖片外，我們只進行horizontal flip, vertical flip以及rotate，以盡量保持資料的正確性。然而，此效果不彰，對於模型的預測結果沒有太大的改善。

Step4.Adjust CNN Kernel Size

一開始在訓練標記瞳孔的模型時，我們發現當瞳孔大小比較大時，模型標記的瞳孔會容易是中空的。針對這種洞我們則是將模型某一層的的kernel size調大，因為我們推測kernel size不夠大，或通過層數不夠多可能導致瞳孔中間無法被判斷為瞳孔，進而產生洞。(瞳孔中心附近為一片黑色，沒有足夠的feature來確認其為瞳孔而不是背景，因此模型很可能是先做edge detection找到瞳孔邊緣後再往內填色，而前述方法就是為了讓模型能將edge的資訊橫向傳遞給距離更遠的

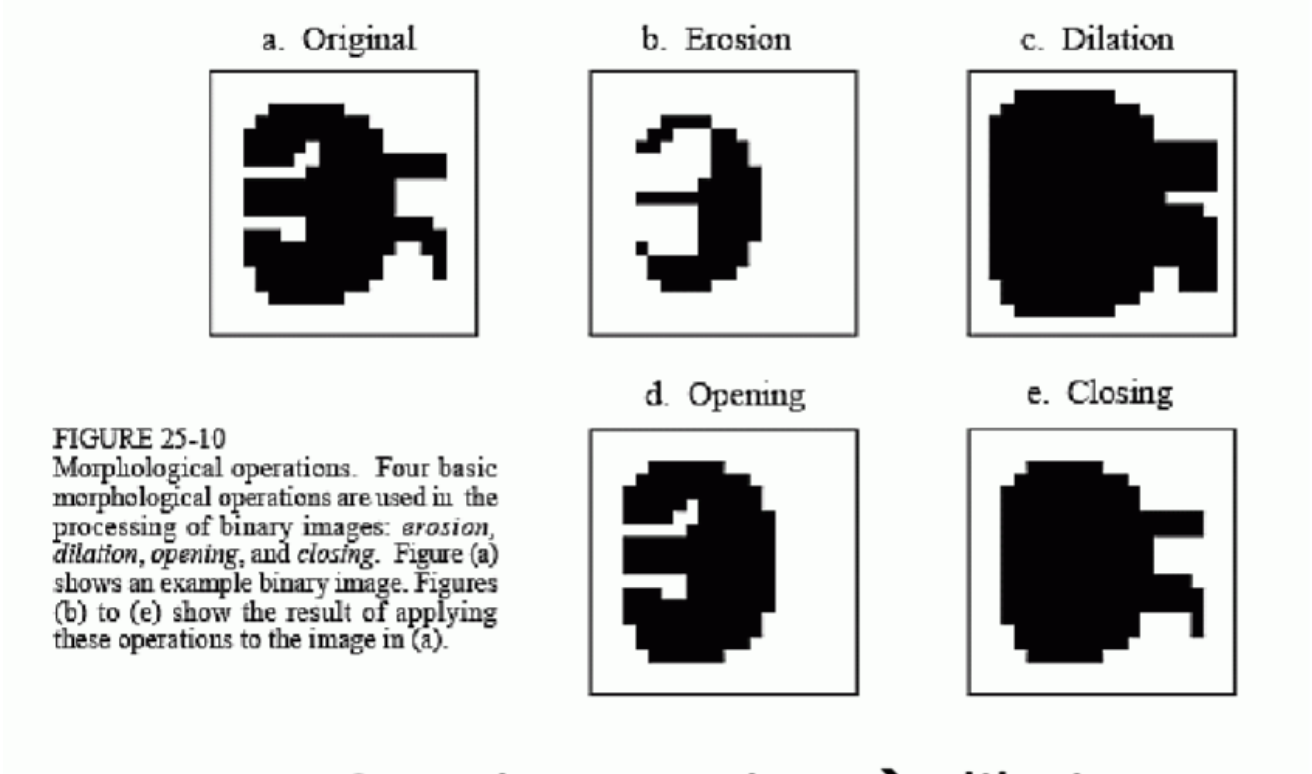
pixel)
經過此調整後，預測瞳孔的圖有效獲得改善。



Step5.Morphological Operation & Weighted Median Filter(WMF)

我們發現，模型生出來的圖有些只能將瞳孔邊緣標記出來，中間會有空洞，無法完整表示瞳孔的位置，且在處理睫毛遮蓋到瞳孔的圖片時，該現象會特別明顯，因此針對此問題我們嘗試下列兩項方法：

(1)Morphological Operation:

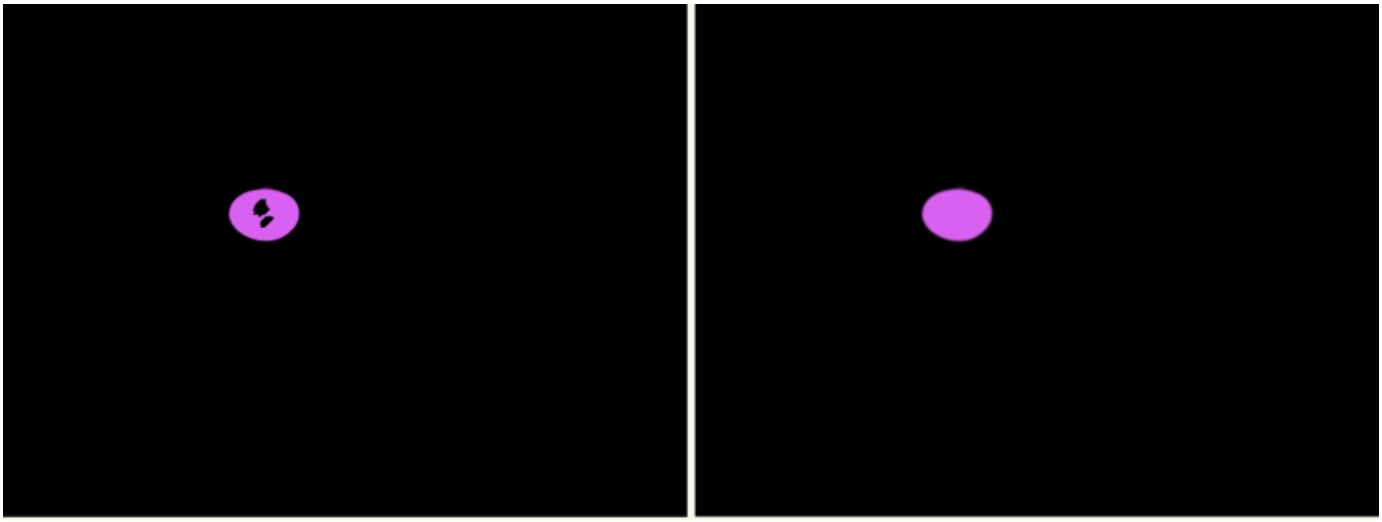


- Opening: erosion → dilation
- Closing: dilation → erosion

(出自簡韶逸老師上課投影片Lec2)
從label圖片中可發現，除了瞳孔以外，其他部分皆為黑色，也就是pixel value為0，而瞳孔為粉紅色，其pixel value大於0。
因此為了將黑色的孔洞補成粉紅色，我們先進行Dilation，也就是透過max pooling以該kernel中的最大值來表示該kernel所有成員的值。

然而這樣的做法會將部份瞳孔外的區域認定為瞳孔，因此再對外圍進行Erosion，也就是min pooling將邊界值認定為黑色。

如此一來，雖然有可能損失部分瞳孔邊界的銳利度，但能將中間大量的孔洞填補起來。

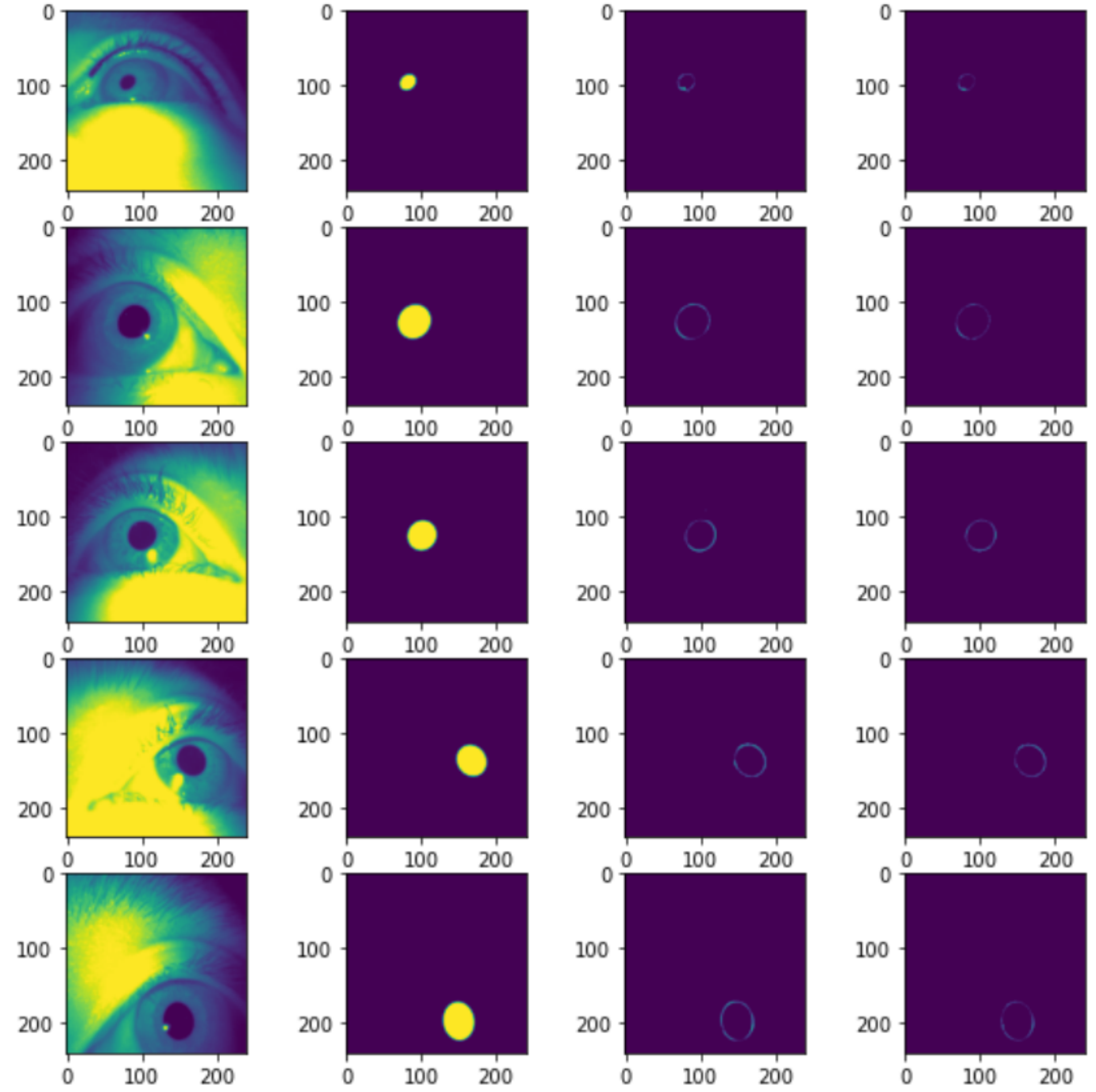


before　　after

(2)Weighted median filter

傳統的median filter可以被拿來補一些小洞，或消除照片中局部的noise。而weighted median filter不同於傳統的median filter直接取周圍所有值的median，WMF會考慮到周圍pixel的顏色，給顏色相近者較大的權重，來解決sharp edge會被median filter模糊的問題。

不僅不會模糊，WMF甚至也有機會能使edge更加清晰，達到sharpen edge的作用，如下圖所示。



(column1：input data,
column2 : labels,
column3 : the L1 difference from labels before WMF,
column4 : the L1 difference from labels after WMF)
由圖可得知，column3在進行WMF前與labels的差距，明顯比column4進行WMF之後的圖來得大，因此WMF對於edge sharpening有很大的幫助。

Data	Percentage error before WMF	Percentage error After WMF
1	10.6%	6.8%
2	3.1%	1.9%
3	3.6%	2.8%
4	3.7%	2.4%
5	3.6%	2.3%

Codalab Results(S5)

Method	Accuracy
Pupil Generator Model + Confidence Value Model(NN) + step1~2	0.767419
Pupil Generator Model + Confidence Value Model(NN) + step1~3	0.786316
Above + data augmentation	0.837472
Confidence Value Generator by above algorithm(Rule-based)	0.885683

Review

1.沒有進行額外model的嘗試:

我們在產生瞳孔的模型僅使用過基礎的autoencoder架構(簡單的CNN)，而沒有再額外嘗試諸如DenseNet, Resnext, VGG等等經典的model交互比較，而由於我們上述Rule-based confidence value generator的表現很大程度取決於我們model所生出來的結果(標出來的瞳孔面積)，因此我們認為沒有選擇效果較強且更為適切的模型是其中一個關鍵問題。
由於本次期末專題使用的圖片較大張，資訊較多，壓縮圖片又怕會失去很多資訊導致邊緣判斷失準，解析度降低，因此都用原圖進行訓練。而deep learning的方法很吃GPU設備，只有使用Colab的我們常常訓練模型都要訓練20、30個小時，中間也可能被中斷，被硬體給限制住能嘗試的模型種類。

2.Training Epoch不夠多

我們推測瞳孔的圖會沒標好的另一原因在於訓練的epoch不夠多，model可能還沒train至完全收斂我們就開始test，也因此我們的結果不甚理想。

3.使用了Rule-based的方法來做confidence prediction

我們最後用了rule-based的方法來產生confidence value，並且在S5上得到了還不錯的結果。但前面也提過，這種方法的表現會嚴重取決於瞳孔標註模型標出的瞳孔面積；此外由於我們是人工來制定判別規則，對於一些我們想像之外的圖片分布可能就比較無法適應。比如說我們設計的演算法中有用到"同一個sequence中各圖之間瞳孔大小的關係"這個資訊；而在我們看得到的dataset中，label為0的圖片量從未超過sequence中圖片總數的一半，但在hidden set中未必是如此，如果出現諸如此類的狀況，我們設計的演算法就可能會出各種奇怪的問題。這邊我們認為，瞳孔的存在與否應該可以直接由瞳孔的形狀、與相鄰其他瞳孔的可見面積等等local的資訊就推得，未必需要看到整個sequence這麼多，而NN理應能把這件事做得很好，所以如果我們改回使用NN來做confidence prediction，並想辦法解決overfit的話，應該能有更好的表現。

4.沒有實作太多Data augmentation

因為擔心把瞳孔移出圖片外，我們在做binary classifier時沒有疊太多augment。也可能是這個原因，最後跑出來的confidence model在S5上的表現不理想。後來我們覺得其實還有很多色彩上的augmentation是可以嘗試的，不用擔心ground truth位移的問題，這方面是我們的盲點。

5.沒有嘗試更多的post process

我們在最後產生confidence value時沒有使用deep learning中常使用的技巧–ensemble，且也沒對生出來的瞳孔的圖做更多圖像處理會用的技巧，我們事後認為若這些操作可能會對表現有所提升，比起單一模型可以有較robust的結果。

Conclusion:

在Pupil Image Generator的任務當中，為了加快運算速度，我們選擇較輕量的深度學習模型，搭配傳統後處理的演算法來補足模型上的不足，但也許因為模型較小，導致部分輸出的圖像不夠完整，尤其是較為noisy的input，就算經過後處理還是無法還原瞳孔位置。

而在Confidence Value Generator任務中，我們利用瞳孔面積的變化設計出二元分類演算法，有著非常快的運算速度，但缺點是效果的強弱須取決於Pupil Image Generator，若圖片輸出的結果不佳，則會使Confidence Value也表現的差強人意。

根據上述分析與結論，我們在報告的最後提出可以改進演算法的方式以及我們所遺漏的方法與嘗試，在未來碰到相關的task時，也可以將以上所述的方法加以實踐，獲得比現在更好的結果。

Future Work (聽完報告後寫的):

藉由以上的討論以及課堂的報告，我們也學到了許多技巧，如果要繼續這個題目的話我們會往以下的方向來嘗試。

1. prepocessing:

前處理使用Gamma correction可以讓瞳孔更加明顯，幫助後續的演算法辨認。另外也可以嘗試不同edge detection的方式，在前處理就獲得一些特徵，幫助模型訓練。最後Ganzin的人員有提到，S8的hidden set連camera的角度都是不同的。對於這樣的資料，做random-perspective應該也是不錯的方法。

2. postprocessing:

在後處理中嘗試使用find contour搭配ellipse fitting或Binarization的方式，提升瞳孔的完整性。(cv2.ellipse_fitting應該算是滿重要的postprocessing方法，我們在這之前不知道這個函數的存在真的是比較可惜)

3.Deep learning model:

嘗試使用DeepLabV3、RIT、U-Net或其他專門用在segmentation的model
今天最後一組報告使用到了YOLO先初步找出瞳孔的位置，再進行瞳孔標註的方法我們也覺得很有趣，感覺這麼做之後，瞳孔標註模型就可以很大程度的減少label unbalance的問題，聽起來也是很棒的嘗試。

報告真的很精彩，讓我們學到了很多東西，想在最後感謝一下老師跟助教，謝謝你們花時間與心力舉辦這個期末專題。

Reference:

1.AutoEncoder:

(1)https://medium.com/ml-note/autoencoder-一-認識與理解-725854ab25e8 (https://medium.com/ml-note/autoencoder-%E4%B8%80-%E8%AA%8D%E8%AD%98%E8%88%87%E7%90%86%E8%A7%A3-725854ab25e8)

(2)https://jason-chen-1992.weebly.com/home/-autoencoder (https://jason-chen-1992.weebly.com/home/-autoencoder)

(3)https://blog.keras.io/building-autoencoders-in-keras.html (https://blog.keras.io/building-autoencoders-in-keras.html)