深度視覺期末專題

學號：B093022023
系級：機電系 113
姓名：林宥辰

目錄

一、資料增量

先將原始 anchor 資料進行隨機資料增量(data augmentation)，模擬下游的影像情形，其順序如圖一所示。每個處理過程會有 1/2 的機率被執行，以此確保資料增量的隨機度。

(一) 深度改變：使用高斯亂數(常態分佈)選定一個深度，將每個像素值進行平移，若超出 0 或是 65535 就強壓在這個範圍之內。

(二) 深度傾斜：使用高斯亂數(常態分佈)選定一個角度，用三維的旋轉矩陣和 openCV 的 warpPerspective 函數產生立體旋轉的效果。其中 angle 的單位爲弧度。因爲需要進行三維的立體旋轉，所以用 A1 映射矩陣將二維照片轉換至三維空間，用 RY 旋轉後再映射回二維。

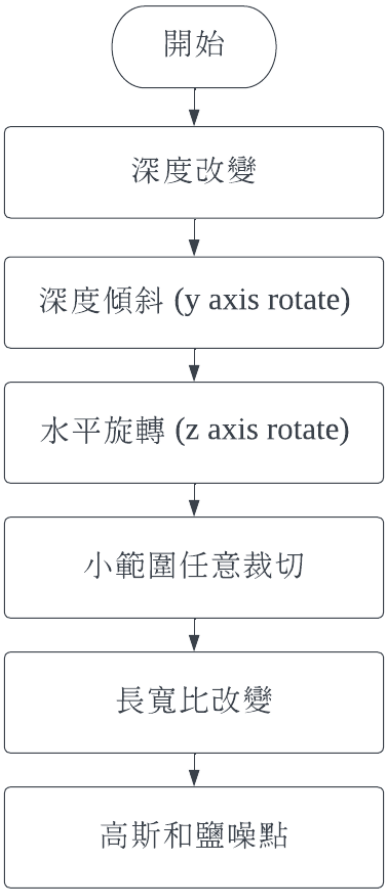
```
def rotate_y_axis(img, file, b=15):
  angle = normal_rand()*b
  angle = angle*np.pi/180
  w = img.shape[1]
  h = img.shape[0]

  A1 = np.array([[1, 0, -w/2],
                 [0, 1, -h/2],
                 [0, 0, 1],
                 [0, 0, 1]])

  A2 = np.array([ [1, 0, w/2, 0],
                 [0, 1, h/2, 0],
                 [0, 0, 1, 0]])

  RY = np.array([[np.cos(angle), 0, -np.sin(angle), 0],
                [0, 1, 0, 0],
                [np.sin(angle), 0, np.cos(angle), 0],
                [0, 0, 0, 1]])

  M = np.dot(A2, np.dot(RY, A1))

  img = cv2.warpPerspective(img, M, (img.shape[1], img.shape[0]))
  # img = clip(img, file, process=rotate_y_axis.__name__, )

  return img
```

(三) 水平旋轉：使用高斯亂數(常態分佈)選定一個角度，用 OpenCV 的 getRotationMatrix2D 來產生一個旋轉矩陣，並且用 warpAffine 旋轉照片。

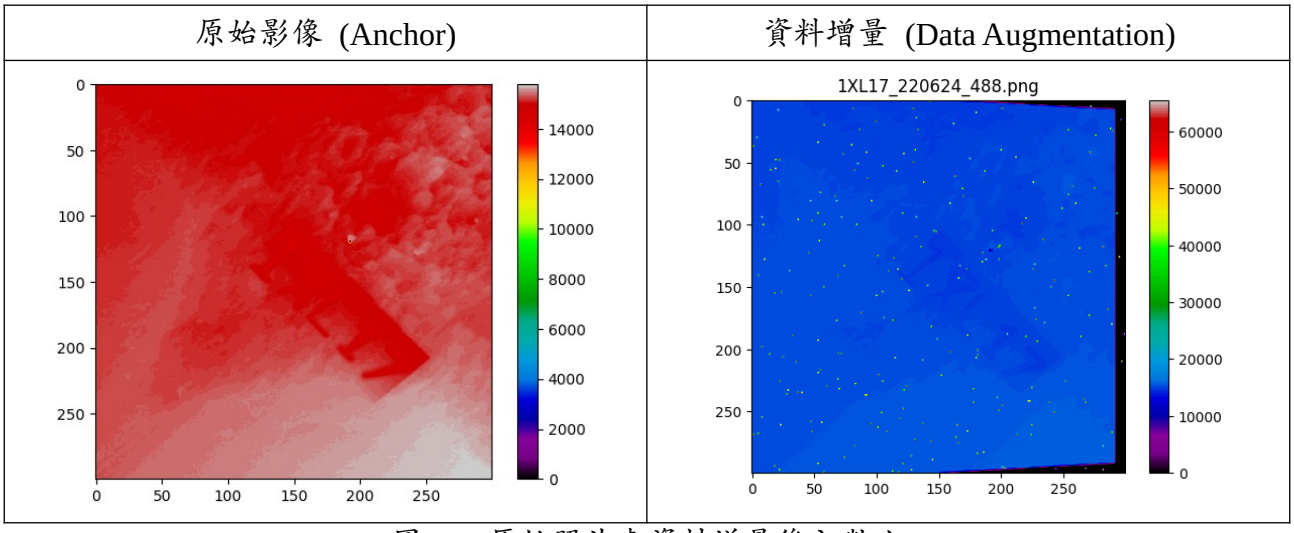
(四) 小範圍任意裁切：使用高斯亂數(常態分佈)決定裁切大小(dx, dy)，並且決定起始點(x,y)，裁切出照片(x:x+dx, y:y+dy)。如果發生問題(如 dx, dy = 0)就跳過此步驟不裁切。

(五) 長寬比改變：因爲剛剛經過了隨意裁切，所以此步驟只要將照片強制縮放成 300px*300px 即可。此外，這個過程一定會被執行，以確保影像輸出的大小無誤。

(六) 高斯和鹽噪點：用 skimage 工具產生高斯雜訊，並且產生一個隨機布林矩陣當作鹽噪點。



圖一、資料增量架構圖



圖二、原始照片與資料增量後之對比

每個影像會生成數張增量影像，並且儲存在 ./myaug/ 當中。用原始影像和增量影像的所有組合生成 dataframe ，並且給予相對應的 label。

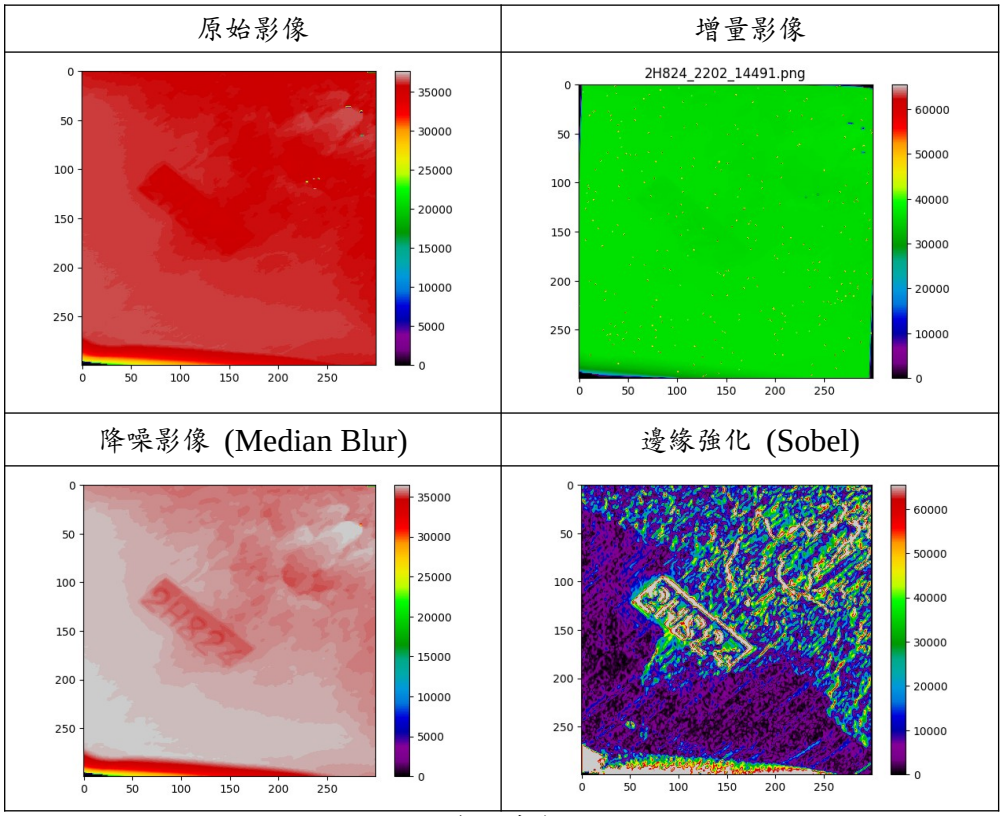
	Anchor	img	label
1037	3MB30_2111_8178.png	3MB30_2111_8178_2.png	1
29060	3MB30_2111_8176.png	3MB30_2111_8176_4.png	1
16491	2H824_2202_13419.png	2H824_2202_13419_3.png	1
24536	2H824_2202_14477.png	2H824_2202_14477_2.png	1
8899	2H824_2202_14494.png	2H824_2202_14494_2.png	1
31854	2H824_2202_14488.png	2H824_2202_14488_1.png	1
31003	3MB30_2111_8183.png	3MB30_2111_8183_4.png	1
18663	1XR21_2202_14622.png	1XR21_2202_14622_1.png	1
5153	2H824_2202_14493.png	2H824_2202_14493_1.png	1
19906	2H824_2202_14486.png	2H824_2202_14486_1.png	1

圖三、 train_dataframe (節錄)

二、資料(前)處理

針對影像進行降造處理以及邊緣強化，提升神經網路的辨識度。爲了消去鹽噪點，使用 cv2.medianBlur 進行中間值模糊，並用 Sobel edge detection 來強化邊緣。

```
def sobel( img ):

  x = cv2.Sobel(img, cv2.CV_16S, 1, 0, ksize=3)
  y = cv2.Sobel(img, cv2.CV_16S, 0, 1, ksize=3)

  # to uint8
  absX = cv2.convertScaleAbs(x)
  absY = cv2.convertScaleAbs(y)

  # merge x and y
  dst = cv2.addWeighted(absX, 0.5, absX,0.5,0)

  return (dst/255.*65535.).astype(np.uint16)
```



圖四、降噪前後之對比

```python
def data_preproccessing(img):
  # [300, 300]
  # img type must be np.uint16

  img = cv2.medianBlur(img,3)
  img = sobel(img)

  # [300, 300]
  return img
```

```python
def __getitem__(self, idx):
  row = self.df.iloc[idx]
  # A_img = torch.from_numpy(cv2.imread(DATA_DIR + row.Anchor, -1).astype(np.int32)).unsqueeze(2).permute(2, 0, 1) / 65535.0
  A_img = cv2.imread(DATA_DIR + row.Anchor, -1).astype(np.uint16)
  # [300, 300]
  A_img = data_preproccessing(A_img)
  A_img = torch.from_numpy(A_img.astype(np.int32)).unsqueeze(2).permute(2, 0, 1) / 65535.0


  Img = cv2.imread(DATA_AUG_DIR + row.img, -1).astype(np.uint16)
  # [300, 300]
  Img = data_preproccessing(Img)
  Img = torch.from_numpy(Img.astype(np.int32)).unsqueeze(2).permute(2, 0, 1) / 65535.0
  label = torch.from_numpy(np.array([row.label]).astype(np.float32))

  # P_img = torch.from_numpy(cv2.imread(DATA_DIR + row.Positive, -1).astype(np.int32)).unsqueeze(2).permute(2, 0, 1) / 65535.0
  # N_img = torch.from_numpy(cv2.imread(DATA_DIR + row.Negative, -1).astype(np.int32)).unsqueeze(2).permute(2, 0, 1) / 65535.0
  return A_img, Img, label
```

在 dataloader 中調用影像處理的函式，便可以針對每張影像進行處理。

三、模型架構

由於要進行的是 contrastive learning，我們要一次餵兩個影像給模型，由模型將影像轉換爲特徵向量。若兩張影像屬於同類，希望讓這兩張影像的距離縮短，反之增長。

(一) 模型：使用數層 CNN layer 進行 feature extraction，之後再平坦化成特徵向量，進而比對相似度。完整架構圖請參考附錄。

(二) Loss Function：使用 contrastive loss，其方程式如下。

$$\mathbf{D}^2 = \sum (embsB - embsA)^2$$

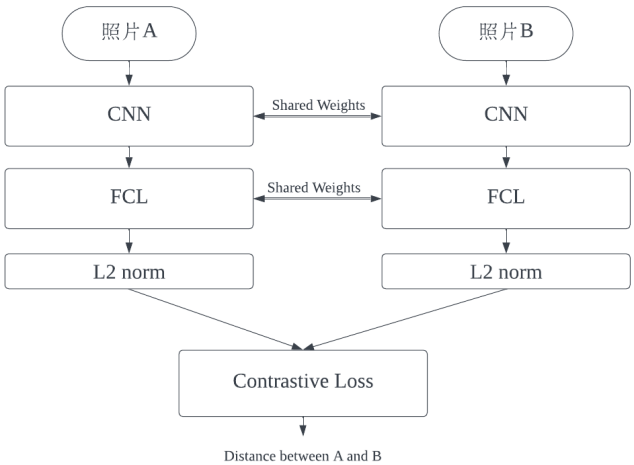
$$\mathbf{S} = \begin{cases} 0 & \text{if two images are different} \\ 1 & \text{otherwise} \end{cases}$$

$$L = \frac{1}{n} \sum \{\mathbf{SD^2} + (1 - \mathbf{S})[max((m - \mathbf{D}), 0)]^2\}$$

其中，embsA, embsB 分別爲上游影像(anchor)及任意一張下游影像(positive 或 negative)的特徵向量， S 即爲 Label 。

Loss function 可分爲前後兩個部份，前項的 loss 和距離呈正相關，而後項呈負相關。

如果兩張照片不一樣，則 loss function 中前項爲 0，所以距離越近，loss 會越大。反之若 S=1，距離越遠，loss 會越大。和我們想要得到的結果相同。



圖五、簡易架構圖

四、訓練成果及分析

使用 learning rate = 0.0001，weight decay = 0.0005，batch size =12，epoch = 10。資料爲上游原始影像以及各 4 張增量影像的所有組合，共 55696 組資料。使用其中 44556 組當作測試資料，11140 組當作驗證資料。其中：

- Loss 爲 每個 batch 的 contrastive 平均值：

$$\mathbf{D}^2 = \sum (embsB - embsA)^2$$

$$\mathbf{S} = \begin{cases} 0 & \text{if two images are different} \\ 1 & \text{otherwise} \end{cases}$$

$$L = \frac{1}{n} \sum \{\mathbf{SD^2} + (1-\mathbf{S})[max((m-\mathbf{D}), 0)]^2\}$$

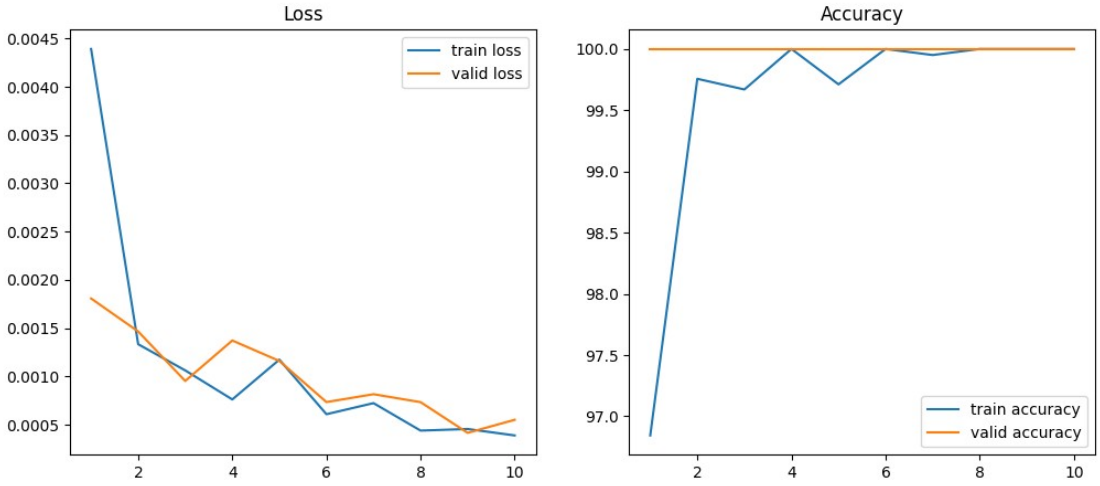
- Accuracy 爲每個 batch 中，positive 距離小於 negative 距離的機率：

$$Accuracy = P(D_A < D_B \mid label_A = 1 \wedge label_B = 0)$$

參考圖六曲線，準確率從 97%提升到 100%，驗證準確率則是維持在 100%，說明資料集有可能過於簡單，參考圖六。

然而，用該模型進行預測時(使用提供的 ./testdata/ 資料預測)，結果也呈獻 100% ，並無明顯的下降趨勢，故初步判斷該模型的**效度較高**。

此外，從測資結果來看， negative pair 的距離幾乎都維持在 1 以上，而 positive pair 則比較浮動，模型**信度較低**，但由於資料量較少，所以進一步用假說檢定檢測。



圖六、訓練曲線圖

```
Current Time:  2023-05-15 12:01:15.82908112:01:15
Number of test cases:  3
Test accuracy: 100.0%
```

圖七、簡單測資結果

case1	case2	case3
'pos.png': 0.38 'neg1.png': 1.35 'neg2.png': 1.08 'neg3.png': 1.46	'pos.png': 0.53 'neg1.png': 1.18 'neg2.png': 0.71 'neg3.png': 0.90 'neg4.png': 1.24	'pos.png': 0.76 'neg1.png': 1.30

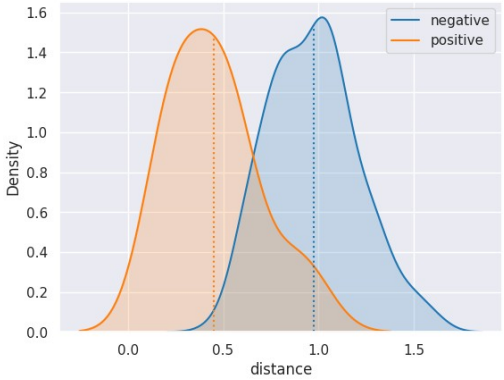
表一、簡單測資結果

五、模型評估

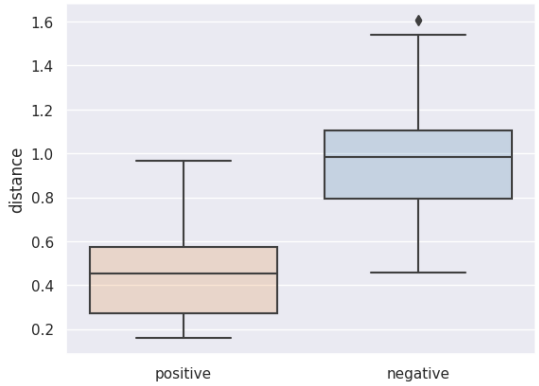
欲了解距離 D 的大小是否與影像爲 positive 或 negative 有關，以及其顯著度。使用簡單測資的預測結果進行以下假說檢定：

- 虛無假設 H0：距離 D 的大小和 label 無關
- 對立假設 H1：距離 D 的大小和 label 有關(且 positive<negative)

因爲是非連續、獨立的兩組小樣本，選用 Mann–Whitney U test 進行假說檢定。最終結果得到 p_value = *8e-9* < *.001*，說明有足夠證據拒絕需無假設，故**假說成立**。



圖八、positive 及 negative 距離的密度分佈



圖九、positive 及 negative 距離分佈級距

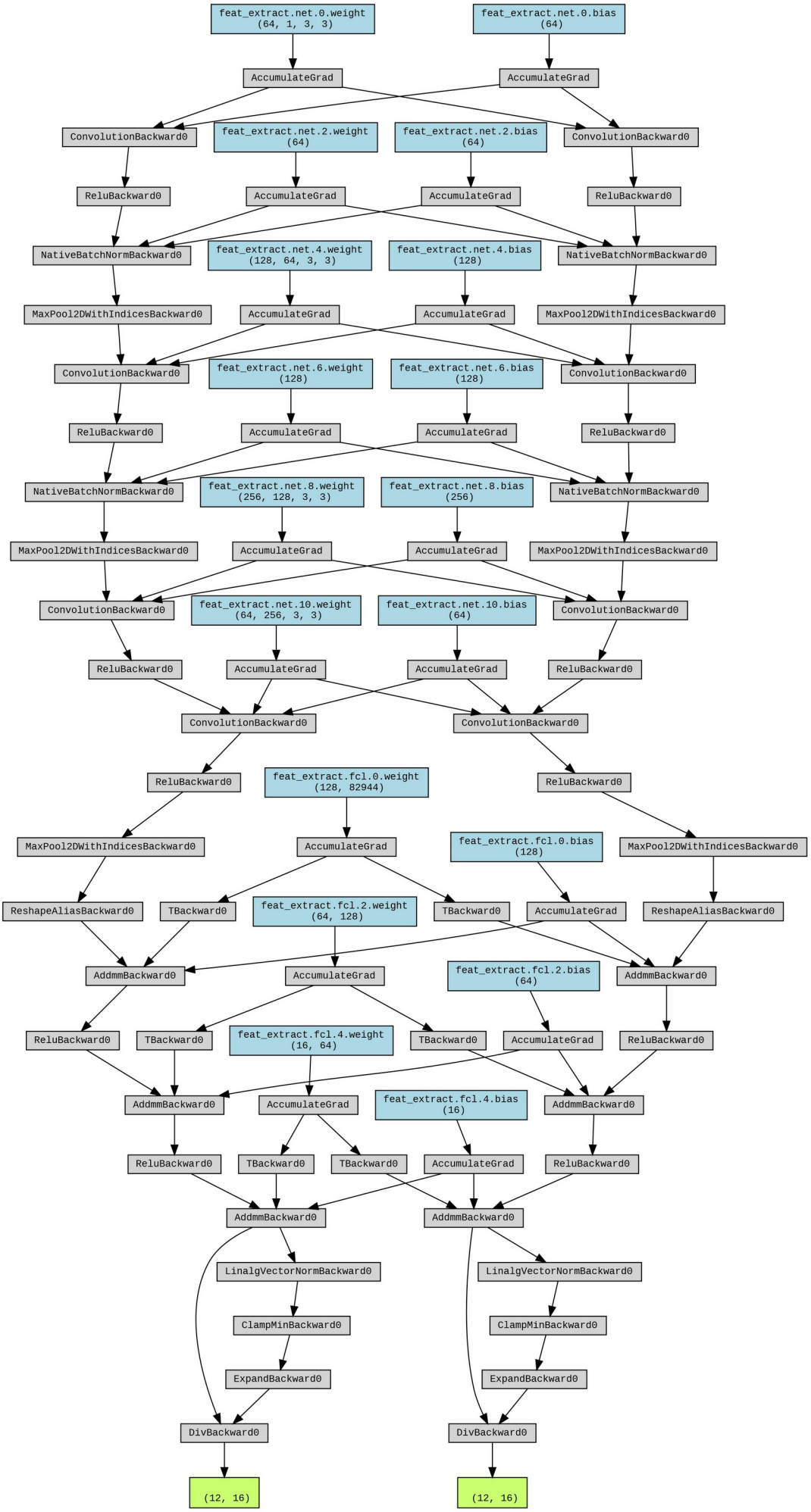
另外，從 positive 及 negative 影像的距離密度分佈曲線來看，兩曲線的交集面積較大，距離在 0.5~1 時，模型有較大的機率會誤判，當距離落在 positive 影像的第三四分位距(Q3)前和 negative 影像的第一四分位距(Q1)後，模型會有較高的信心水準。

綜合密度分佈及級距分佈的結果來看，雖然模型顯著度相當高，但信度仍然較低，與初步推論信度較低的結論相符。

最後使用正式測資，準確率爲：

```
Current Time:  2023-05-22 11:17:29.39436811:17:29
Number of test cases:  15
Test accuracy: 97.82609%
```

附錄、模型架構圖


feat_extract.net.0.weight
(64, 1, 3, 3)
feat_extract.net.0.bias
(64)
AccumulateGrad
AccumulateGrad
ConvolutionBackward0
feat_extract.net.2.weight
(64)
feat_extract.net.2.bias
(64)
ConvolutionBackward0
ReluBackward0
AccumulateGrad
AccumulateGrad
ReluBackward0
NativeBatchNormBackward0
feat_extract.net.4.weight
(128, 64, 3, 3)
feat_extract.net.4.bias
(128)
NativeBatchNormBackward0
MaxPool2DWithIndicesBackward0
AccumulateGrad
AccumulateGrad
MaxPool2DWithIndicesBackward0
ConvolutionBackward0
feat_extract.net.6.weight
(128)
feat_extract.net.6.bias
(128)
ConvolutionBackward0
ReluBackward0
AccumulateGrad
AccumulateGrad
ReluBackward0
NativeBatchNormBackward0
feat_extract.net.8.weight
(256, 128, 3, 3)
feat_extract.net.8.bias
(256)
NativeBatchNormBackward0
MaxPool2DWithIndicesBackward0
AccumulateGrad
AccumulateGrad
MaxPool2DWithIndicesBackward0
ConvolutionBackward0
feat_extract.net.10.weight
(64, 256, 3, 3)
feat_extract.net.10.bias
(64)
ConvolutionBackward0
ReluBackward0
AccumulateGrad
AccumulateGrad
ReluBackward0
ConvolutionBackward0
ConvolutionBackward0
ReluBackward0
feat_extract.fcl.0.weight
(128, 82944)
ReluBackward0
MaxPool2DWithIndicesBackward0
AccumulateGrad
feat_extract.fcl.0.bias
(128)
MaxPool2DWithIndicesBackward0
ReshapeAliasBackward0
TBackward0
feat_extract.fcl.2.weight
(64, 128)
TBackward0
AccumulateGrad
ReshapeAliasBackward0
AddmmBackward0
AccumulateGrad
feat_extract.fcl.2.bias
(64)
AddmmBackward0
ReluBackward0
TBackward0
feat_extract.fcl.4.weight
(16, 64)
TBackward0
AccumulateGrad
ReluBackward0
AddmmBackward0
AccumulateGrad
feat_extract.fcl.4.bias
(16)
AddmmBackward0
ReluBackward0
TBackward0
TBackward0
AccumulateGrad
ReluBackward0
AddmmBackward0
AddmmBackward0
LinalgVectorNormBackward0
LinalgVectorNormBackward0
ClampMinBackward0
ClampMinBackward0
ExpandBackward0
ExpandBackward0
DivBackward0
DivBackward0
(12, 16)
(12, 16)