**108-2 Deep Learning Homework 1 - 0853412吳宛儒**

1. **Deep Neural Network for Classiﬁcation**
2. You should decide the following hyperparameters: number of hidden layers, number of hidden units, learning rate, number of iterations and mini-batch size. You have to show your (a) learning curve, (b) training error rate and (c) test error rate in the report. You should design the network architecture by yourself.

以下針對各種超參數比較與設定的過程作敘述：

*number of hidden layers* = 2, 3

在選擇隱藏層的數量的時候，除了比較不同層數的效果以外，也把其中hidden units的數量納入參考。由以下結果可以推測，2層hidden layers的準確度在training和testing會稍微比3層hidden layers表現得還好，而在Loss的部分則是在3層的較多hidden nodes的情況下有最低的Loss。考量到weights在初始化時的隨機性，以及以下幾種測試結果，推測在2層的時候使用[512,256]會有較好的效果，其Testing acc雖不是測試中最高，但差距也相對微小，因此後面的實驗決定使用2層hidden layers。

|  |  |  |  |
| --- | --- | --- | --- |
| Number of hidden layers | Training acc rate | Testing  acc rate | Loss |
| 2 [64,32] | 0.9908 | **0.90**18 | 0.001522982912986206 |
| 3 [64,32,16] | 0.9936 | 0.8985 | 0.0011779387895317244 |
| 2 [512,256] | **0.9999** | 0.8720 | **0.0001107692271499326** |
| 3 [512,256,128] | **0.9999** | 0.8510 | 0.00015172037504256403 |

*number of hidden units* = [512,256], [256,128], [128,64], [64,32], [32,16]

在hidden units的選擇上，嘗試了以下幾種組合，而hidden layers的數量經過前面的實驗，決定固定在2個layers。另外，各層units的數量偏好選取2的倍數，期望效果會較好。

|  |  |  |  |
| --- | --- | --- | --- |
| Number of hidden units | Training acc rate | Testing  acc rate | Loss |
| [512,256] | **0.9999** | 0.8720 | **0.0001107692271499326** |
| [256,128] | 0.9998 | 0.8899 | 0.00024855158543373324 |
| [128,64] | 0.9977 | **0.9075** | 0.0006880426410740182 |
| [64,32] | 0.9921 | 0.8989 | 0.0014515771669385946 |
| [32,16] | 0.9753 | 0.8978 | 0.0031453040882808802 |

經過上述實驗各種不同的hidden units數量組合，發現[512,256]平均表現較好，因此後面也會繼續使用兩層layers、[512,256]的組合。雖然參數量較大會使得訓練較為緩慢，但目前的組合數量都還在可接受的範圍內。

*learning rate* = 0.01, 0.001, 0.099

在設定的時候初始以0.01作為learning rate，在其他參數固定的情況下，調整learning rate發現0.099目前有最好的效果。

|  |  |  |  |
| --- | --- | --- | --- |
| Learning rate | Training  acc rate | Testing  acc rate | Loss |
| 0.01 | 0.9914 | 0.8378 | 0.0019462965487882546 |
| 0.001 | 0.7988 | 0.7213 | 0.02477862782140357 |
| 0.099 | **0.9999** | 0.8720 | **0.0001107692271499326** |

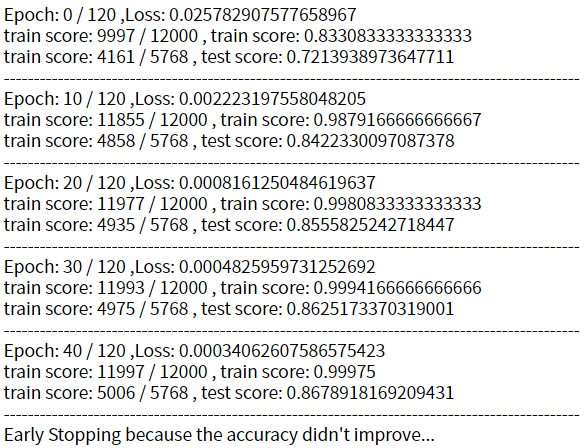
實驗結果可以看出，設定成0.099會有比較好的效果，而在0.01和0.001的結果上就有非常明顯的差異，最後微調至0.099得到相對較好的模型效果。

*number of iterations* = 120, 1000

由上面幾次的實驗選定超參數以後，模型效果可以穩定在120epochs內達到不錯的效果(上述實驗皆在120 epochs下完成)。另外，當epochs數增多以後，考量在此處設定了簡易的提早結束方法，類似early stopping之概念，規則訂為：

1. 當training的accuracy超過十次沒有提升，提早結束training。
2. 當training的loss超過十次沒有下降，提早結束training。

最後為了避免混淆，僅先使用第一種規則，在訓練中會出現例如：

等情況，中斷訓練。

*mini-batch size* = 8,16,32

設定不同的batch size，一樣選取2的倍數，期望效果會較好。

|  |  |  |  |
| --- | --- | --- | --- |
| Size of mini-batch | Training acc rate | Testing  acc rate | Loss |
| 8 | 0.999 | 0.8923 | 0.00048224379848218715 |
| 16 | 0.9998 | 0.8796 | 0.0003805761410311823 |
| 32 | **0.9999** | 0.8720 | **0.0001107692271499326** |

考量到選用後面64,128的大小時，一次訓練會花上太多時間，因此只有比較到32。可以看到當batch size為32的時候效果相對好，即使testing accuracy有下降的趨勢，整體來說在batch size為32的時候Loss還是會是最佳的。

除了上述設定以外，也加入一小段比較：儲存最佳準確度時的weights，若此次epoch之準確度沒有比最佳準確度更高的話，將它的weights改成最佳準確度時的weights，以提升模型效果。

最終設定：

number of hidden layers = 2

number of hidden units = [512,128]

learning rate = 0.099

number of iterations = 120

mini-batch size = 32

最終結果：

|  |  |  |
| --- | --- | --- |
| Learning Curve | Training error rate | Testing error rate |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\1AA35291.tmp | C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\5E4F03C7.tmp | C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\5C26686D.tmp |
| 0.0002345907169628365 | 0.9998333333333334 | 0.8859223300970874 |

1. Please perform zero and random initializations for the model weights and compare the corresponding error rates. Are there any diﬀerence between two initializations? Please discuss in the report.

Initialize w randomly:

(w: random; b: zeros)

|  |  |  |
| --- | --- | --- |
| Learning Curve | Training error rate | Testing error rate |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\1AA35291.tmp | C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\5E4F03C7.tmp | C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\5C26686D.tmp |
| 0.0002345907169628365 | **0.99**98333333333334 | **0.88**59223300970874 |

Initialize w with zeros:

(w: zeros; b: zeros)

|  |  |  |
| --- | --- | --- |
| Learning Curve | Training error rate | Testing error rate |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\D11B4E09.tmp |  | C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\45D38203.tmp |
| 0.031424696895915316 | **0.56**24166666666667 | **0.55**32246879334258 |

可以看出來當weights初始化成0的時候，效果降低了很多，可見random帶有的隨機性，可以讓模型有比較好也較快的收斂效果。另外也將bias試試看不同的初始化效果，由於weights隨機初始化會有比較好的效果，因此在bias的初始化比較上，僅會使用隨機初始化的weights，而不再使用初始化為0的weights。最後發現bias初始化為0效果會比random初始化好很多。

1. Design your network architecture with the layer of 2 nodes before the output layer. (1) Plot the distributions of latent features at diﬀerent training stages. For example, you may show the results when running at 20th and 80th learning epochs. (2) Please discuss the evolution of latent features at diﬀerent training stage.

原本使用在output layer前加上2 nodes的隱藏層，但是發現這樣training效果會大幅下降，因此除此之外也用了PCA以及tSNE降維的方式，取output layer前的隱藏層之latent features作為觀察。另外，在討論區有同學向助教詢問確認以後，是以test data在其中幾次epoch作為比較與紀錄。(程式每10個epochs會記錄一次latent features)

在最後一層前加上2 nodes的隱藏層：

|  |  |  |
| --- | --- | --- |
| Hidden units | 10th epoch | 110th epoch |
| [16,2] | C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\75C91F6F.tmp | C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\B7435A71.tmp |
| [32,16,2] | C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\1FC8B6A7.tmp | C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\F89C51E9.tmp |

Hidden nodes數量為[16,2]或[32,16,2]，可以看到在[32,16,2]這樣相對於[16,2]較多層的網路中，經過110個epochs以後，latent features有更分開的效果。

降維方法：

在實作過程中，發現較高維度的hidden nodes在降維上除了時間較久(tSNE較有感覺)，在二維圖中呈現也比較沒辦法明顯呈現evolution的過程，有可能是因為256維壓至2維差距過大使得沒辦法有效投影。因此在這邊選擇使用[20,15]作為hidden nodes之數量，發現這樣投影在二維會比較清楚，所花時間也會比較短。實作時以[20,15]作為hidden nodes之數量，其他設定皆以上述最終設定為主。

PCA降維(15至2維)

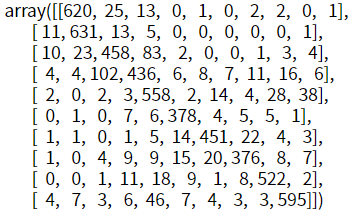
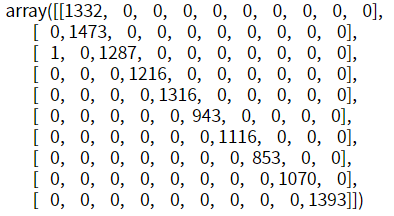
|  |  |
| --- | --- |
| 10th epoch | 110th epoch |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\39EB553.tmp | C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\C6E57A17.tmp |

tSNE降維(15至2維)

|  |  |
| --- | --- |
| 10th epoch | 110th epoch |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\26B334DF.tmp | C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\844C3861.tmp |

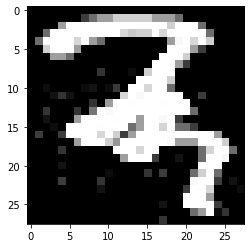
由PCA和tSNE的結果可以看出來，PCA在經過110個epochs以後，雖然點沒有很外散，但是可以看到原本在中間的淺藍色群或其他在一開始混雜的群有像四處分散的趨勢，像一個圓。在tSNE降維的結果可以明顯看出來在第10epoch中尚未分清楚的類別，在110epoch已經明顯分開，且同群的點有集中的趨勢。

1. Please list your confusion matrix and discuss about your results.



左表代表總共分錯1個training samples。右表代表總共分錯120個testing samples。除了斜對角正確答案以外，最大的數字，也就是最常被分錯的為分錯102次的真實為3、預測為2的類別。

以下是將真實label為3、預測類別為2的samples印出來：



1. **Convolutional Neural Network for Image Recognition**
2. Please describe in details how to preprocess images because of the diﬀerent resolution images and various bounding boxes region in Medical Masks dataset and explain why. You have to submit your preprocessing code.

第一種方法：

因為不同圖片上戴口罩的部分不一樣，因此透過pillow中crop((left, bottom, right, top))的方法，將圖片裁剪成需要的樣子。而對於裁減以後圖片大小不一的問題，則是在網路最初始的地方使用PyTorch中的AdaptiveAvgPool2d將圖片統一變成32\*32的大小。

第二種方法：

因為第一種方法是在網路前面才做修改，因此batch size只能為1，否則會有input不同維度的問題。因此常是另種方法，在PyTorch ImageFolder中的transform參數設定裡，加入transform.Resize((32,32))，將image大小轉成統一的32\*32大小，如此便可以在後面的dataloader設定非1的batchsize，可訓練較快。

1. Please implement a CNN for image recognition by using Medical Masks dataset. You need to design the network architecture, describe your network architecture and analyze the eﬀect of diﬀerent settings including stride size and ﬁlter size. Plot the learning curve and the accuracy rate of training and test data.

Network Architecture

AdaptiveAvgPool2d(32,32) → 第一種resize方法才會用到

Conv2d=2層

MaxPool2d=2層，各別接在convolution層後，2\*2

Epochs=10

Learning rate=0.001

以下會針對stride size, filter size, 以及optimizer做比較。

*Stride size*=1,2

|  |  |  |  |
| --- | --- | --- | --- |
| Stride size | Training acc rate | Testing acc rate | Loss |
| 1 | 0.950  [0.93 0.97 0.27] | **0.934**  **[0.93 0.99 0.27]** | 0.158 |
| 2 | **0.957**  **[0.95 0.98 0.44]** | 0.820  [0.40 1.00 0.18] | **0.128** |

嘗試了兩種stride size，發現兩者效果在training accuracy跟loss上並沒有太大差別，但在testing上很stride=1效果比stride=2好很多，因此後面的實驗還是先以stride=1為主。

*Flter size*=3\*3, 5\*5

|  |  |  |  |
| --- | --- | --- | --- |
| Filter size | Training acc rate | Testing acc rate | Loss |
| 3\*3 | **0.950**  **[0.93 0.97 0.27]** | **0.934**  **[0.93 0.99 0.27]** | **0.158** |
| 5\*5 | 0.940  [0.90 0.97 0.27] | **0.934**  **[0.97 0.97 0.36]** | 0.186 |

在filter size大小比較上，比較了3\*3和5\*5兩種，發現平均來看filter size=3\*3會有比較好的表現。

*Optimizer*=SGD, Adam

|  |  |  |  |
| --- | --- | --- | --- |
| Optimizer | Training acc rate | Testing acc rate | Loss |
| SGD | **0.952**  **[0.93 0.98 0.29]** | 0.926  [0.91 0.97 0.41] | **0.145** |
| Adam | 0.950  [0.93 0.97 0.27] | **0.934**  **[0.93 0.99 0.27]** | 0.158 |

嘗試了兩種不同的optimizer，發現SGD和Adam兩者差別沒有很大。

最終設定：

Optimizer=SGD

Learning rate=0.001

Stride size=1

Filter size=3\*3

Epochs=10

Batch size=16

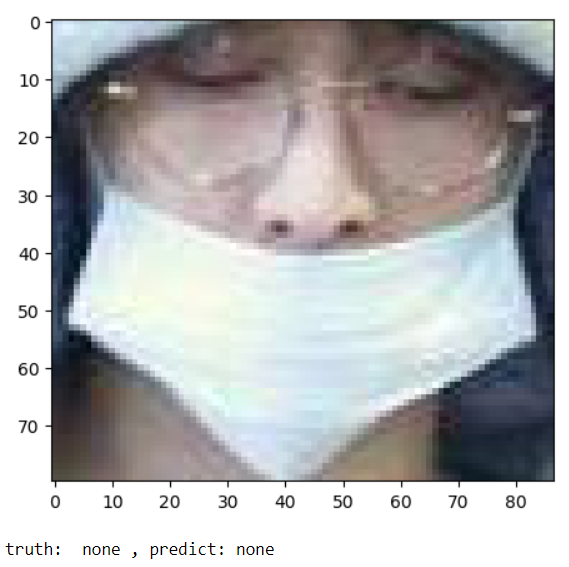
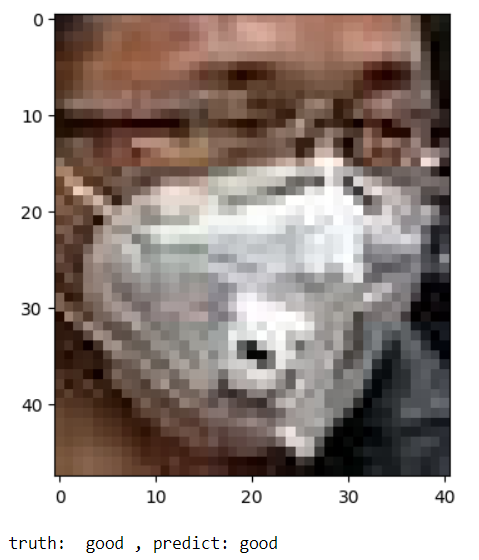
兩層convolution，搭配兩層max pooling

|  |  |
| --- | --- |
| Learning Curve (Loss) | Accuracy |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\34CED80C.tmp | C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\88EA55FA.tmp |
| Loss = 0.137 | Train : 0.953, test : 0.929 |

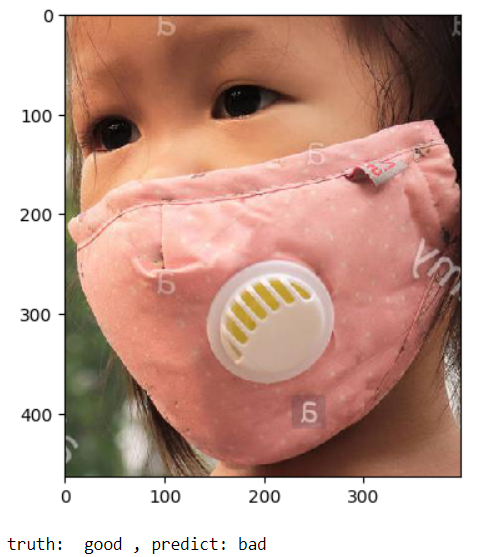
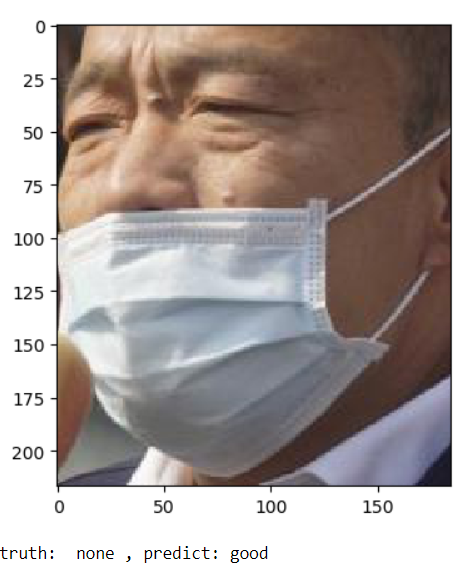
1. Show some examples of classiﬁcation result, *list your accuracy of each classes for both training and test data*, and answer the following questions:

Some examples of classiﬁcation result

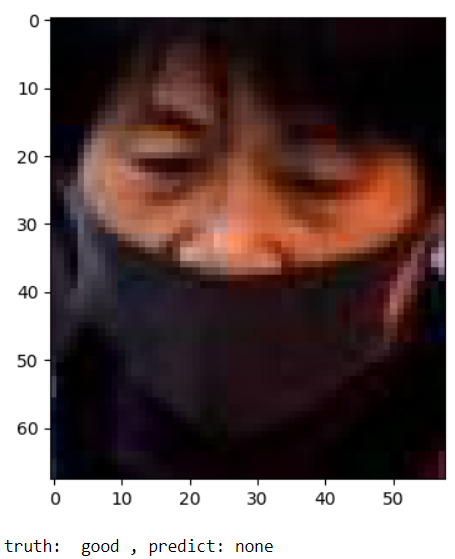
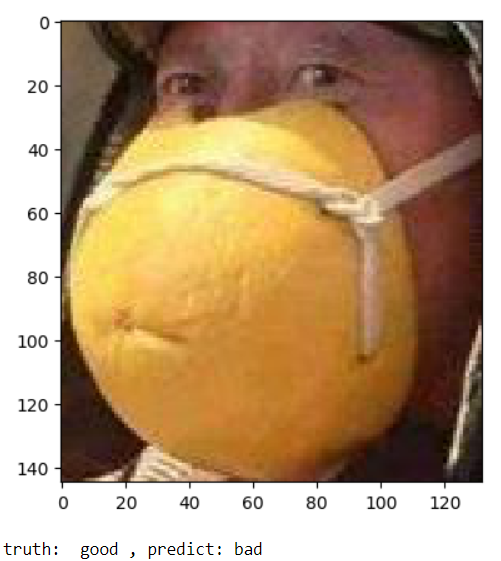
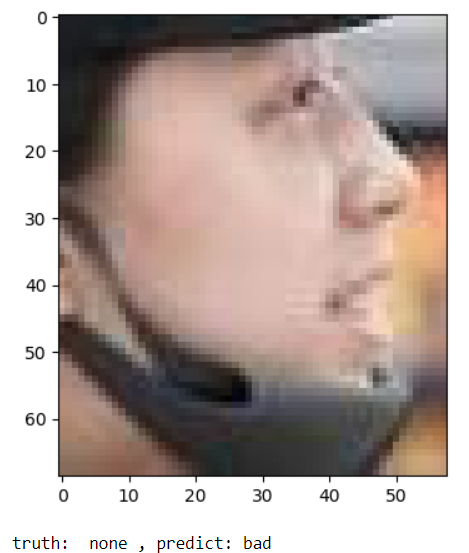
分類成功的案例：

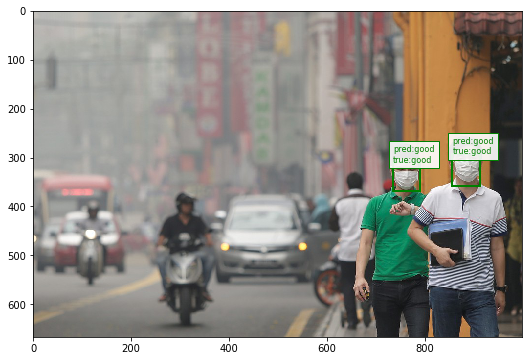


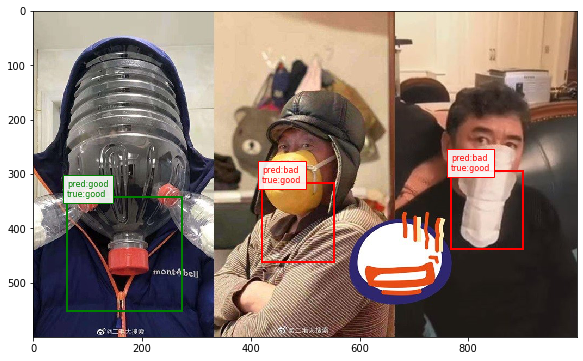
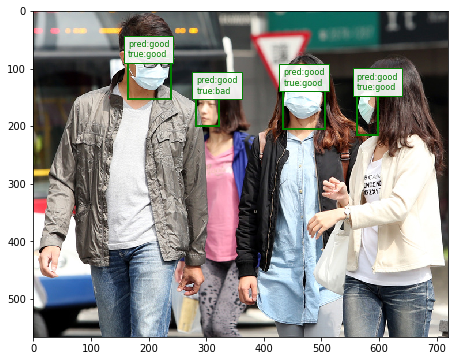
Training中分類失敗的案例：



Testing中分類失敗的案例：

在失敗的案例中，可以看出大多屬於較難分辨的例子：例如口罩其實有遮住口鼻，但鼻子輪廓也很明顯，或是口罩顏色並非藍綠白色，會讓模型以為並沒有戴口罩。另外也有解析度非常差(原圖很小)的例子，也會讓分類難度提高許多。另外，也試著畫出像是作業說明檔案上的圖片，如下：



以testing data為資料，分別印出每張圖中true以及predict的label為何。框線的顏色以predict結果來區分。為了有比較效果，這些圖片是經過兩次epoch訓練以後的模型結果，因此效果比較沒那麼好。也可以看到在初期的時候，許多未戴口罩的會被分類為有戴的，許多不是戴一般顏色口罩的則會被誤認為沒有戴。另外，在這部分畫圖的時候，發現會有位置偏移的情況，因此在實際繪圖的時候，top的值減去60，讓圖片的框(rectangle)會在相對較正確的位置上。

Accuracy of each classes

|  |  |  |
| --- | --- | --- |
| Class | Train Accuracy | Test Accuracy |
| 0 bad | 100.0% | 93.3% |
| 1 good | 97.4% | 98.2% |
| 2 none | **38.5%** | **27.3%** |

(1) Which class has the worst classiﬁcation result and why?

None分得最差，因為從資料數量就可以看出來，None標籤的資料數量最少，因此模型在None特徵學習上可能不夠多，也學不好。

(2) How to solve this problem? (explain and do some experiment to compare the result)

可以增加None的樣本(over-sampling)或是減少另外兩類的樣本(under-sampling)，為了達成這樣的目的，使用了PyTorch中的WeightedRandomSampler來改變model sample data時的權重。做法為：首先先取得每個class的權重，有三個類別，而其權重以各class資料量之倒數得到，因此類別資料量越大，其sample之權重會越低。

Trainset之sample weights為[0.0017301, 0.00035137, 0.00961538]

Testset 之sample weights為[0.01123596, 0.00353357, 0.04545455]

接著，將設定好的sampler做為dataloader之參數即可。

以下是設定完以後重新訓練10個epochs的訓練結果：

**整體：**

|  |  |
| --- | --- |
| Learning Curve (Loss) | Accuracy |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\3F32E051.tmp | C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\5A049787.tmp |
| Loss = 0.024 | Train : 0.863, test : 0.766 |

**各類別：**

|  |  |  |
| --- | --- | --- |
| Class | Train Accuracy | Test Accuracy |
| 0 bad | 100.0%→94.3% | 93.3%→94.6% |
| 1 good | 97.4%→85.2% | 98.2%→78.2% |
| 2 none | **38.5%→79.5%** | **27.3%→63.0%** |

從訓練結果可以看出，純粹改變sample的權重，網路架構不變的話，有可能會使得整體的效果(不論是loss或是準確度上)會表現較差，但這都可以再透過調整網路架構與參數來提升整體效果。另外也看到了我們原本想要解決的「因imbalanced dataset而使得資料量少的類別訓練效果差」的問題，在增加sampling的權重以後效果有所提升，而且效果蠻顯著的，在表格上以粗體表示。

(3) Do some discussion about your results.

在口罩影像分類的任務上，可以從兩個角度來看模型訓練：首先是訓練結果的分析，如上述，其實可以很直覺地瞭解到，當口罩並非常見的顏色或是形狀時，模型較難以分辨類別，容易誤判為沒有戴口罩，而若是有正確地戴口罩但是鼻子輪廓相對明顯者，也會容易被分類到none的類別，而且加上解析度與其他訓練參數考量，模型在分類任務上難度也會相對提升。另一個角度是imbalanced dataset，從類別數量來看，可以很明顯看出各類別的資料量非常不平均，雖然good和bad也有將近六倍數量之差距，但基本兩層Conv2D仍可以輕易地將它們訓練至90%以上的準確度，不過none這個類別不然，none的類別資料量非常少且不容易判別(相較於有戴與沒戴，有戴與沒戴較容易分辨)，若直接將資料數量原封不動進行訓練，none的類別的準確度是非常低的，因此在這裡使用了帶有權重的sampler，將各類別被sample到的機率/權重和其各自的數量呈反比，使得各類別數量能夠較為平均，而實驗結果也顯示，這樣的手法讓none這個類別的準確度大大地提升。