109502009 吳尚明 作業三

A. 程式簡介(main.py)

1. 初始化:

```
class Controller(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.setWindowTitle('Project3')
        self.ui.graph_box.setEnabled(False) #還未選取資料集，故設False
        self.ui.train_btn.setEnabled(False) #還未選取資料集，故設False
        self.epoch = 1
        self.b = 0 #底
        self.h = 0 #高
        self.size = 0 #圖片大小
        self.train_path = '' #訓練資料路徑
        self.test_path = '' #測試資料路徑
        self.train_data = None #訓練資料
        self.test_data = None #測試資料
        self.ui.train_btn.clicked.connect(self.Train) #觸發訓練事件的按鈕
        self.ui.choose_box.currentIndexChanged.connect(self.choose_box) #選擇資料集
        self.ui.graph_box.currentIndexChanged.connect(self.change_graph) #選擇第幾張圖片
        self.ui.epoch_box.valueChanged.connect(self.change_epoch) #選擇epoch
```

2. 依照 hopfield 步驟開始訓練:

```
def Train(self):
    #####清空圖片選擇欄以及三個圖片區塊####
    self.ui.graph_box.setEnabled(True)
    self.ui.graph_box.clear()
    self.ui.train_table.clearContents()
    self.ui.test_table.clearContents()
    self.ui.test_recall_table.clearContents()


    #####取得資料集####
    self.training_data = self.read_file(self.train_path)
    self.testing_data = self.read_file(self.test_path)


    ####依照資料集裡的圖片數新增欄位####
    for i in range(self.training_data.shape[0]):
        self.ui.graph_box.addItem("Graph "+str(i+1))

    self.recall = [] #初始化回想結果的矩陣
    w_matrix = np.zeros((self.size, self.size))#創立權重矩陣
```

```
#####curr_graph為已經扁平化的圖片陣列，依照公式，矩陣相乘後，累加到權重矩陣#####
for curr_graph in self.training_data:
    w_matrix = np.add(w_matrix,curr_graph.T @ curr_graph)

for i in range(self.size):
    w_matrix[i][i] = 0 #斜的那排要為0
θ = w_matrix.sum(axis=0).reshape((-1, 1)) #每一行相加合併後，再轉換成列，求得θ


for curr_graph in self.testing_data:
    x = np.array(curr_graph).copy().T
    for e in range(self.epoch):
        y = w_matrix @ x - θ #依照公式求得預測值
        for i in range(self.size):#判斷是否通過激活函數
            if y[i] == 0:
                y[i] = x[i]
            elif y[i] > 0:
                y[i] = 1
            else:
                y[i] = -1
        x = y.copy()
    self.recall.append(x.copy().T)#將預測結果增加到recall
```

3. 選擇資料集欄位連接到的 function: 取得欄位資訊以及觸發相關的 ui 條件

```
def choose_box(self,index):
    if index==0:#選擇Basic
        self.train_path='./Basic_Training.txt'
        self.test_path='./Basic_Testing.txt'

    else:#選擇Bonus
        self.train_path='./Bonus_Training.txt'
        self.test_path='./Bonus_Testing.txt'
    self.ui.train_btn.setEnabled(True)
    self.ui.graph_box.setEnabled(False)
```

4. 選擇 epoch 欄位連接到的 function: 取得欄位資訊以及觸發相關的 ui 條件

```
def change_epoch(self,index):
    self.epoch=index #取得epoch
    #清空圖片選擇欄以及三個圖片區
    self.ui.graph_box.clear()
    self.ui.graph_box.setEnabled(False)
    self.ui.train_table.clearContents()
    self.ui.test_table.clearContents()
    self.ui.test_recall_table.clearContents()
```

5. 取得選擇欄位的資訊(哪張圖)後，顯示出結果: 將一維陣列重新轉換為二維，並依照數值塗上 table_item 的顏色

```python
def change_graph(self,index):

    ###創建黑色與白色兩個區塊###
    Black = QtGui.QBrush(QtGui.QColor(0, 0, 0))
    Black.setStyle(QtCore.Qt.SolidPattern)
    White = QtGui.QBrush(QtGui.QColor(255, 255, 255))
    White.setStyle(QtCore.Qt.SolidPattern)

    ###將一行的圖片陣列轉換成圖片矩陣並開始填入表格顏色
    graph = self.training_data[index].reshape((self.h, self.b))
    for i in range(self.h):
        for j in range(self.b):
            Table_Item = QtWidgets.QTableWidgetItem()
            if graph[i][j] == 1:
                Table_Item.setBackground(Black)
            else:
                Table_Item.setBackground(White)
            self.ui.train_table.setItem(i, j, Table_Item)
```

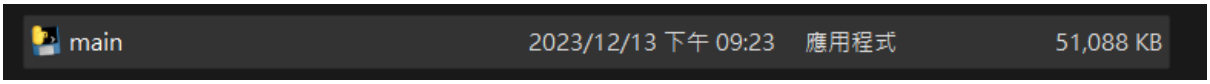
其餘兩個 table 也是一樣的操作

6. 讀取檔案: 一行一行讀，過濾掉換行符號，並依照遇到的空字串次數計算圖片數量，並依照第一張圖的行數以及列數取得底和高，最後做 reshape.(graph_num,1,self.size)的處理，為了之後矩陣的轉置。

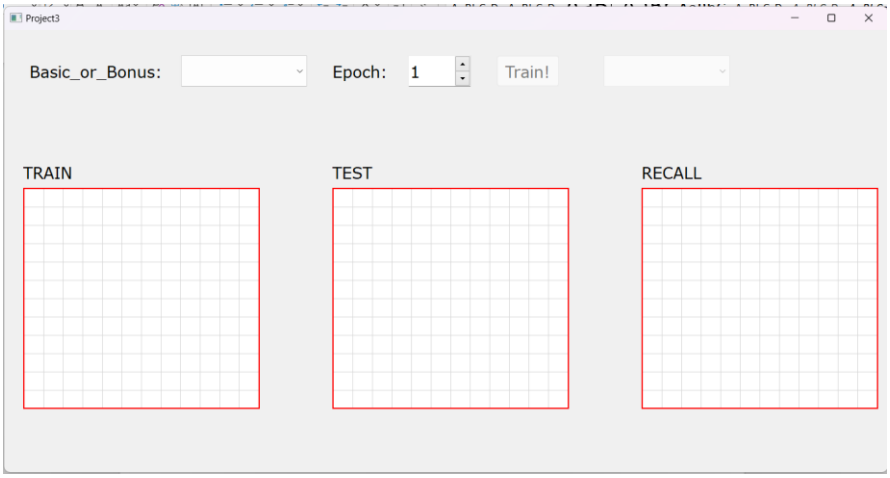
```python
def read_file(self, path: str) -> list:
    self.h = 0
    graph_num = 1
    data = np.array([])
    with open(path) as file:
        file = file.readlines()
    self.b = len(file[0]) - 1
    for line in file:
        if line == file[-1]:
            pass
        else:
            line = line[:-1]
        if line == '':
            graph_num += 1
            continue
        if graph_num == 1:
            self.h += 1
        list_append=[]
        for i in list(line):
            if i =='1':
                list_append.append(1)
            else:
                list_append.append(-1)
        data = np.append(data, list_append)
    self.size = self.h * self.b
    data_list=data.reshape((graph_num, 1, self.size))
    return data list
```

B. 程式執行說明: (UI.py)

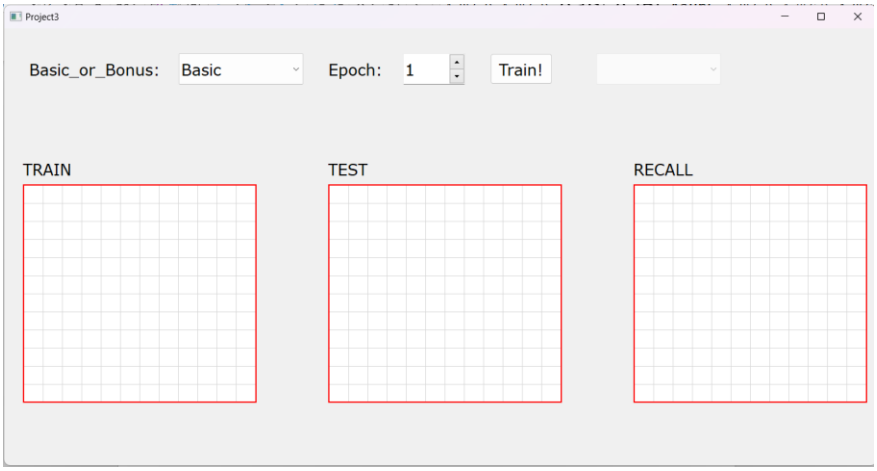
1. 點擊執行檔後，等一下下，UI 就會跳出:



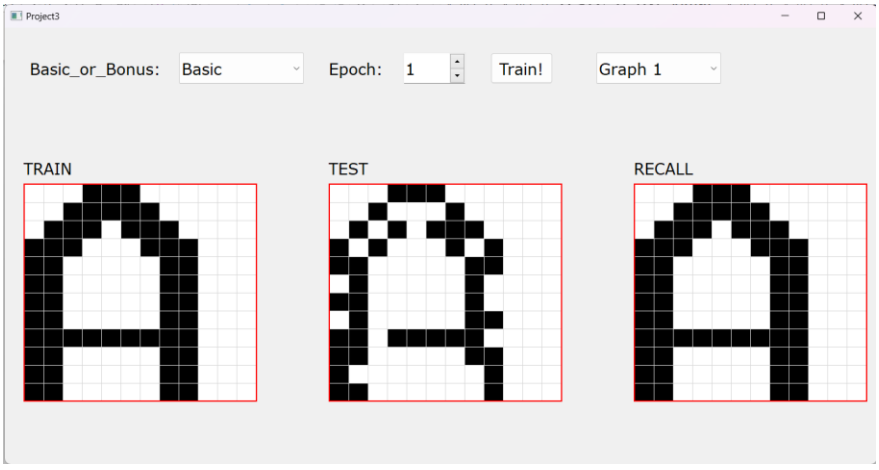
2. UI 最初的畫面: 此時只有左上兩個欄位可以啟動



3. 從左上角欄位選取要使用 Basic 或 Bonus 的資料集: 選取後，Train 按鈕得以啟動

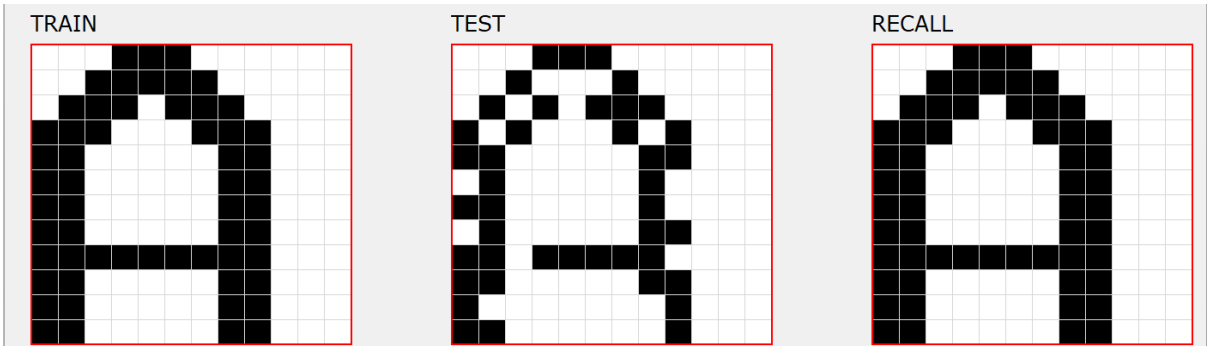


4. 按下 Train!按鈕後，從右上角的圖片選擇欄選擇想要顯示的圖片 ，結果如下圖，分別將訓練和測試資料輸出以及最右邊的回想結果輸出:

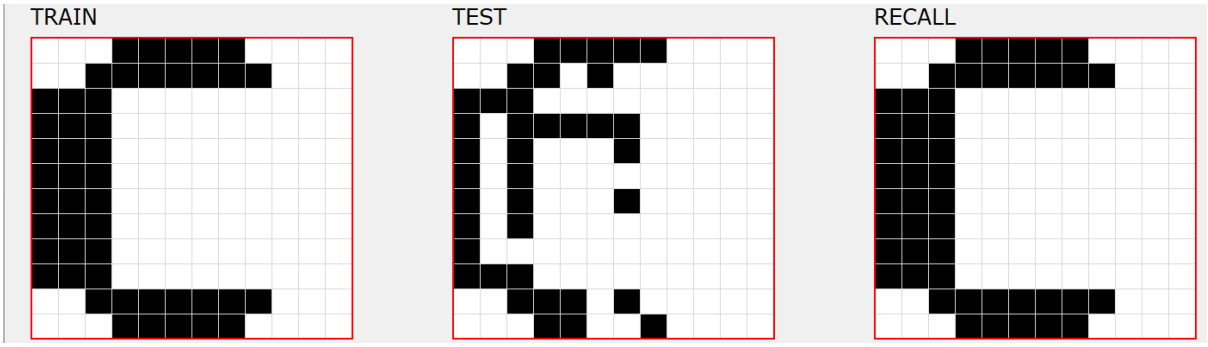


C. (Basic)實驗結果以及分析: 實驗結果為不管使用多少 epoch，都能成功回想

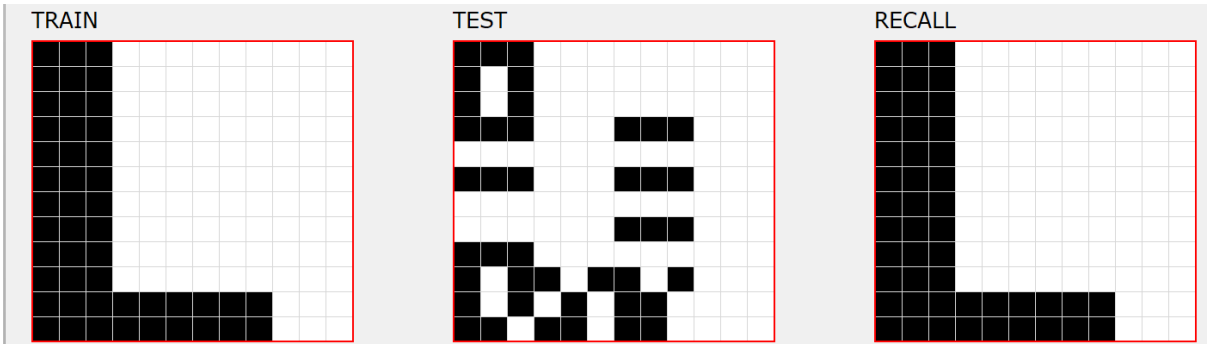
1. Graph 1:



2. Graph 2:



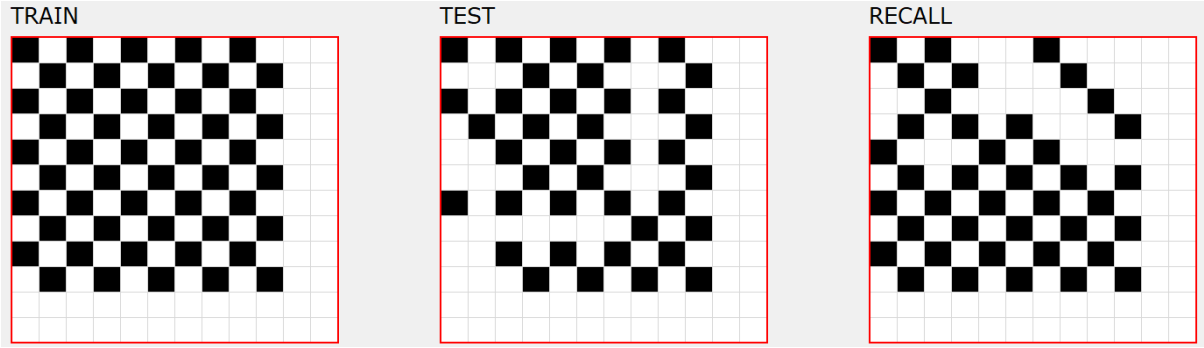
3. Graph 3:



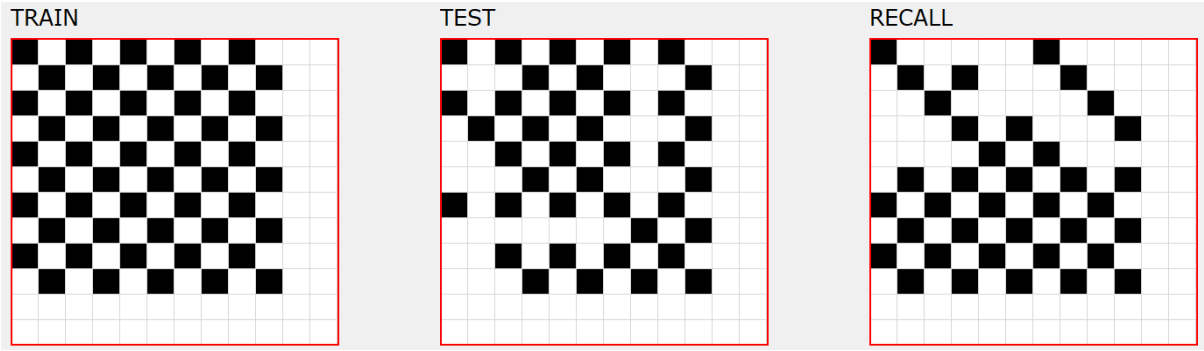
分析及討論:可以看到訓練的圖像有明顯的差異，且破損的圖像可以明顯對應到特定的訓練模式，故模型回想得相當成功，且不論我網路的跌代次數多高都不影響準確性

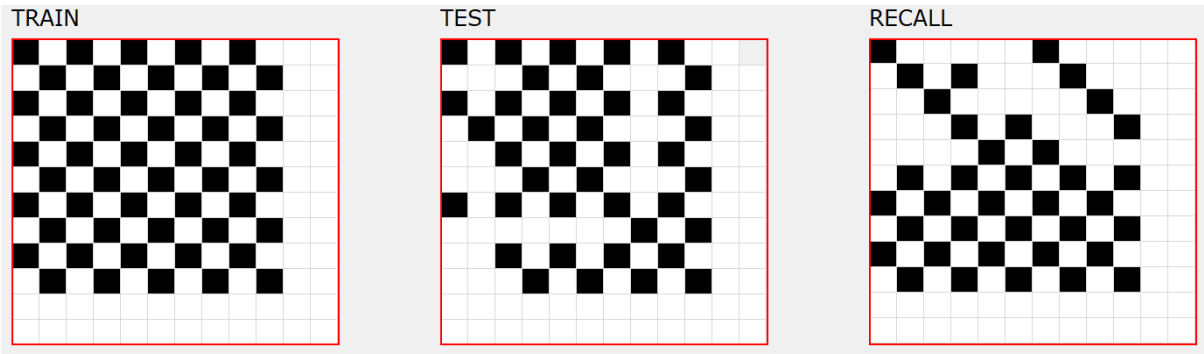
D. (Bonus)實驗結果以及分析: 最好的結果為 epoch:1，為了方便比較這邊放上 epoch:1&3&10 的結果，最上方的圖為 epoch:1，中間的圖為 epoch:3，最下方的圖為 epoch:10

1. Graph 1:

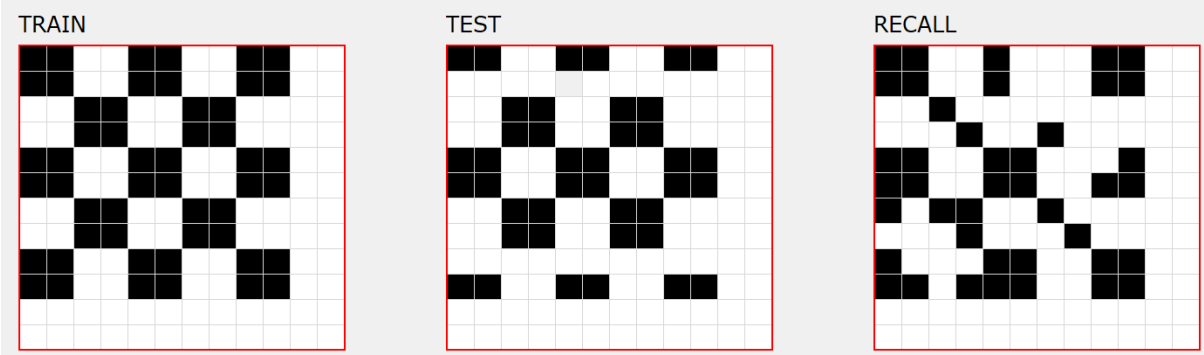


分析以及討論:有填補左下方的空缺但右上方卻損毀的更嚴重，可能的原因有: 1. 相似的訓練模式太多(有許多不一樣的棋盤格分布訓練圖)，因此我們放入得破損的圖對應到許多不同的訓練模式，故無法回想初訓練集中對應的圖片 2.在訓練集中，左下方的密集程度高於右上方 3.Graph 7 為彼此互補的棋盤格圖樣，一定程度增加了回想的困難

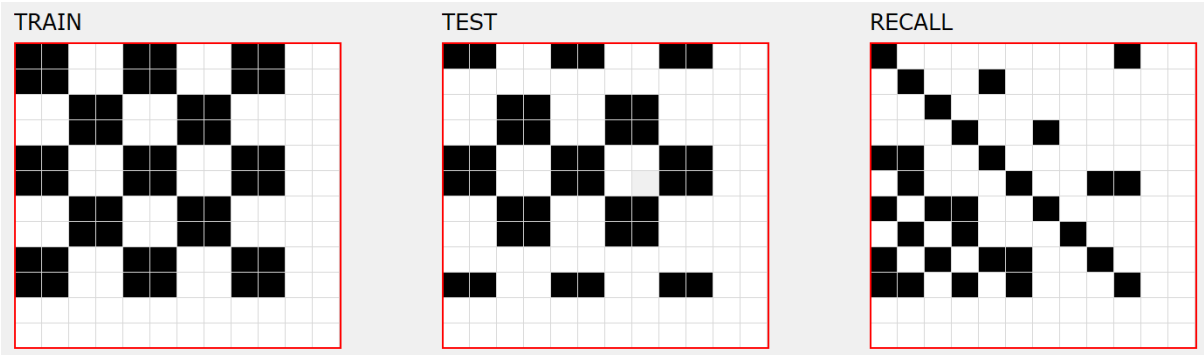


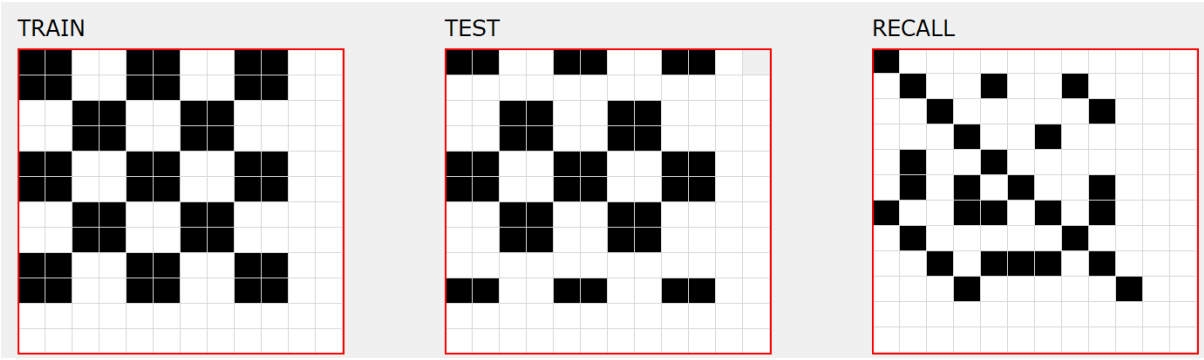


2. Graph 2:

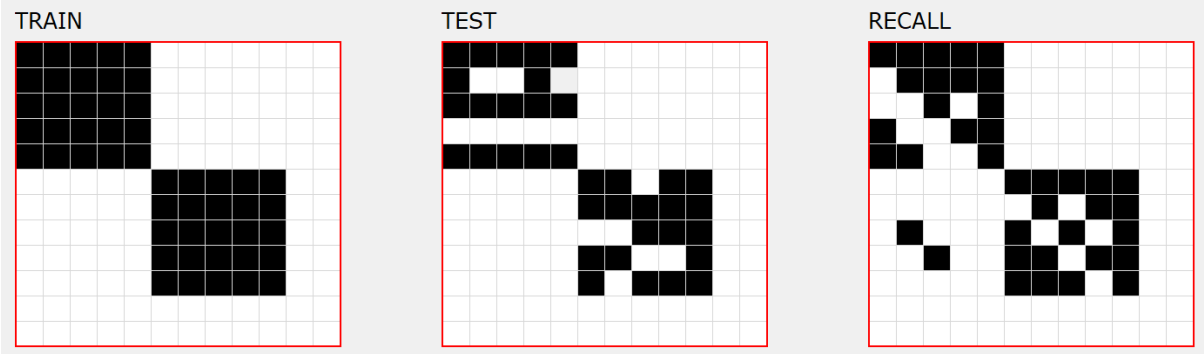


分析以及討論:有填補四個角的空缺但中間卻損毀了，可能的原因有: 1.在訓練集中，有大量的棋盤格圖片，一定程度的增加了回想的困難，從回想結果可以看到中間那部分的圖案雖有破損，但仍舊有保留棋盤格的線條狀



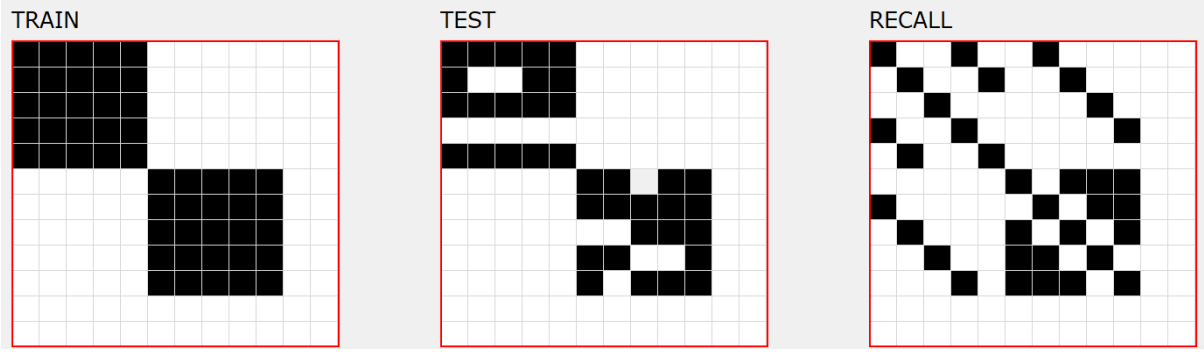


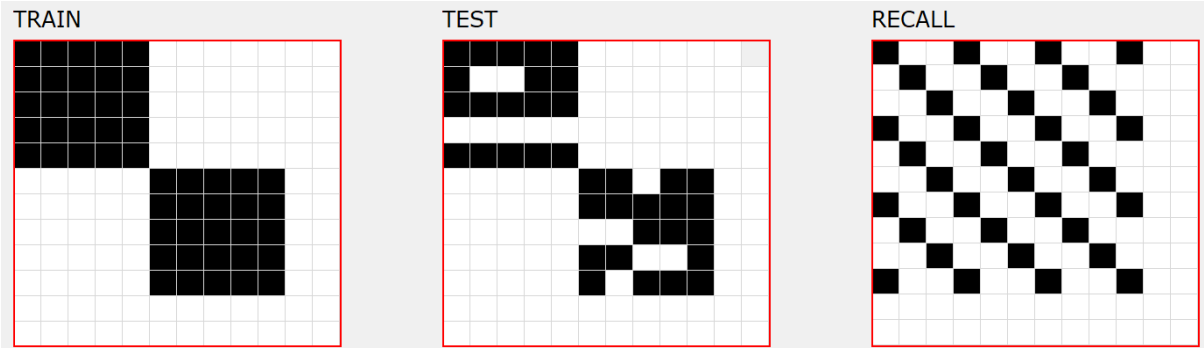
3. Graph 3:



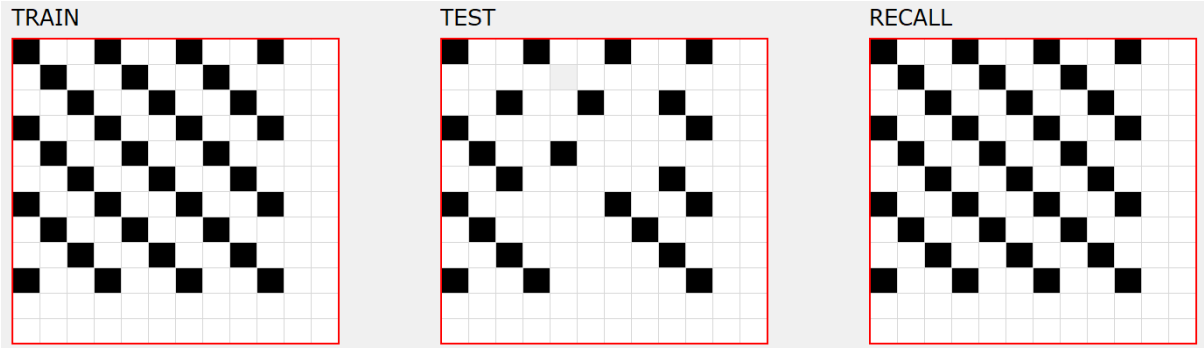
分析以及討論:有少部分的區塊有需補到，但仍舊有更多受損的片段，可能的原因有: 1.在訓練集中，有大量的棋盤格圖片，一定程度的增加了回想的困難，從回想

結果可以看到中間那部分的圖案雖有破損，但仍舊有保留棋盤格的線條狀

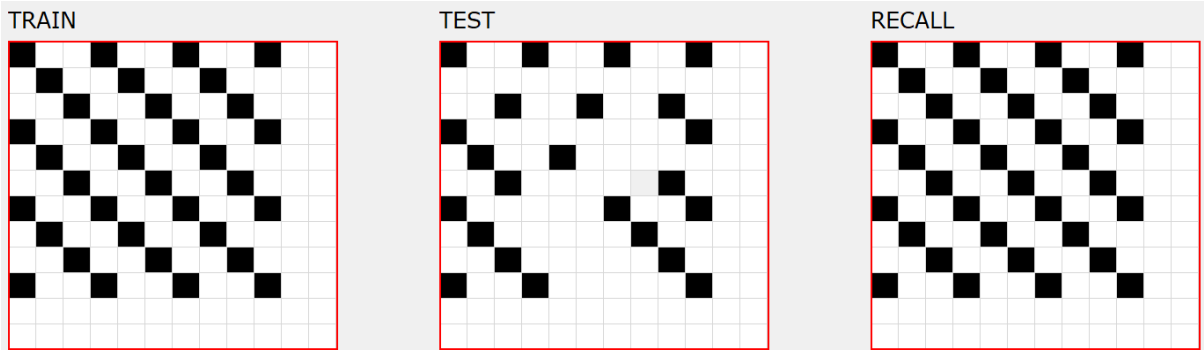


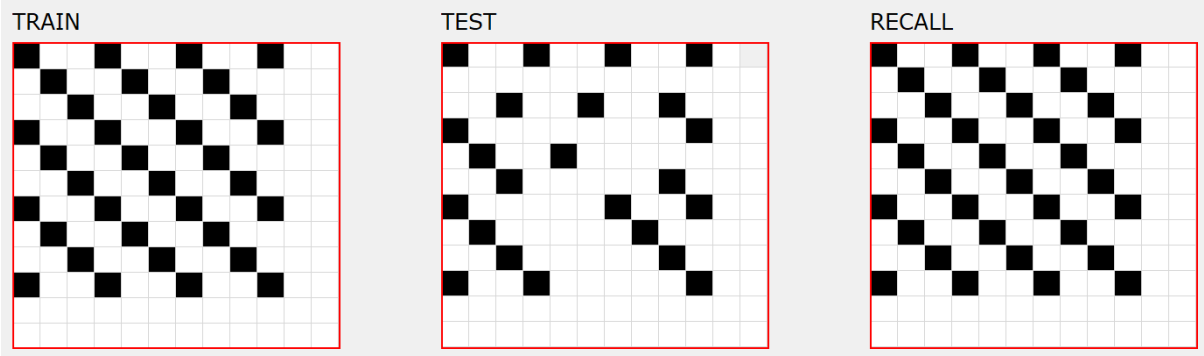


4. Graph 4:

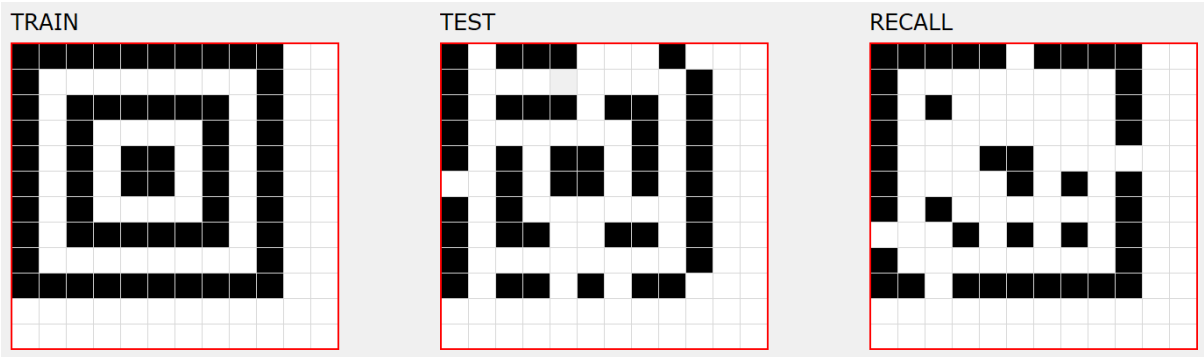


分析以及討論:相當成功，原因可能為，此圖片很明顯為棋盤格圖片的破損狀，但重要的點是黑色方格的位置都是正確的，這一定程度的增加了回想的成功性。

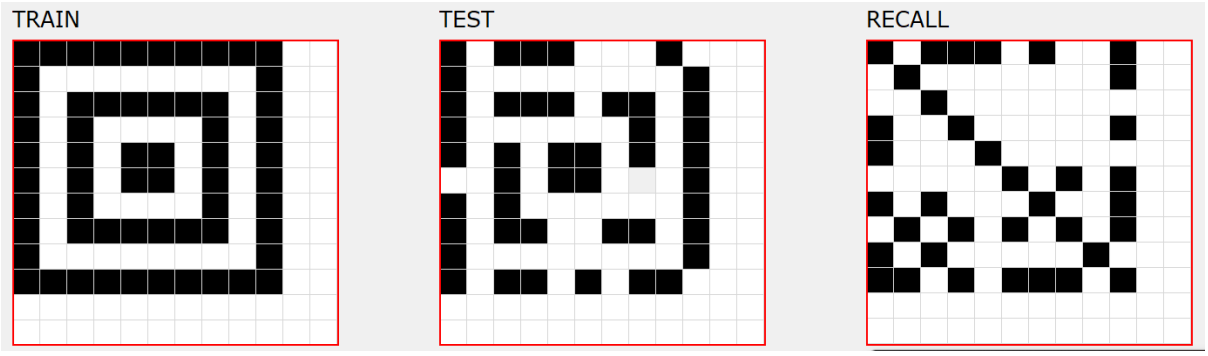


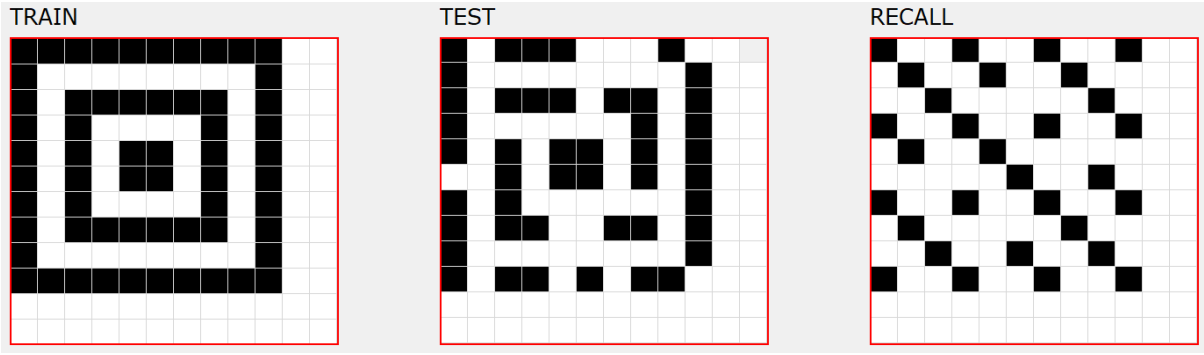


5. Graph 5:

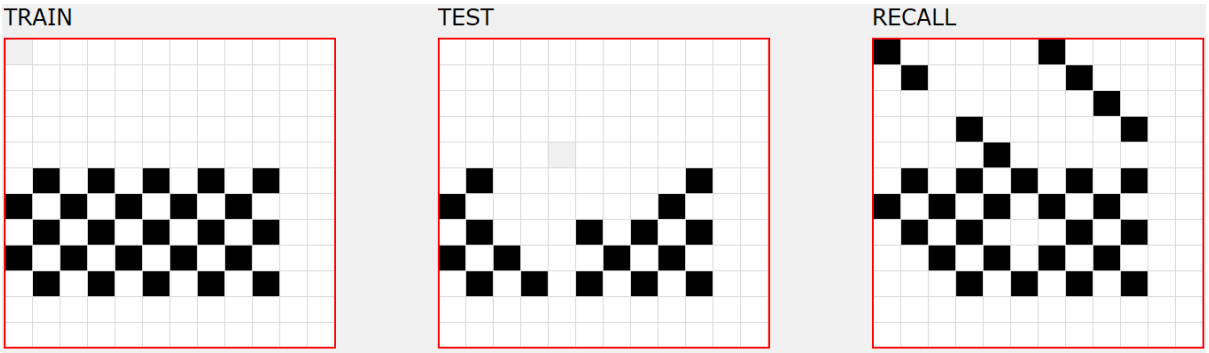


分析以及討論:外框部分的區塊有補到，但中間仍舊有更多受損的片段，可能的原因有: 1. 外框的特徵較為明顯，幫助了模型去回想。

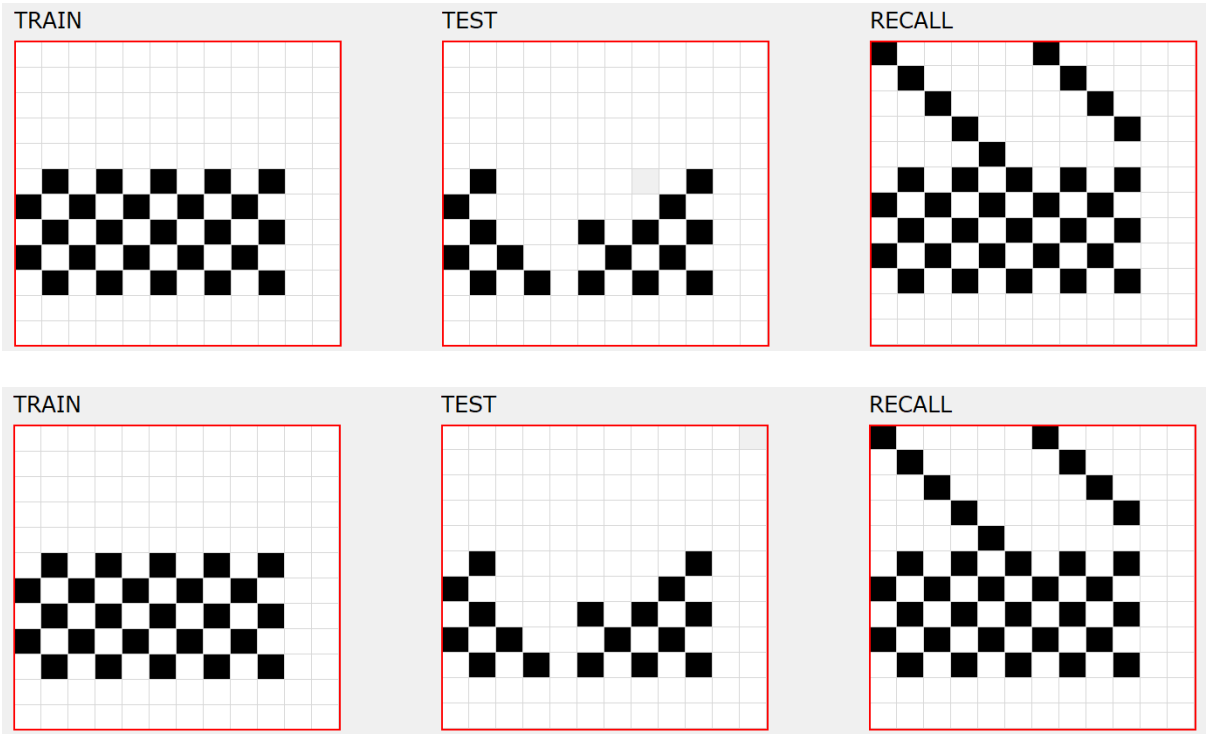




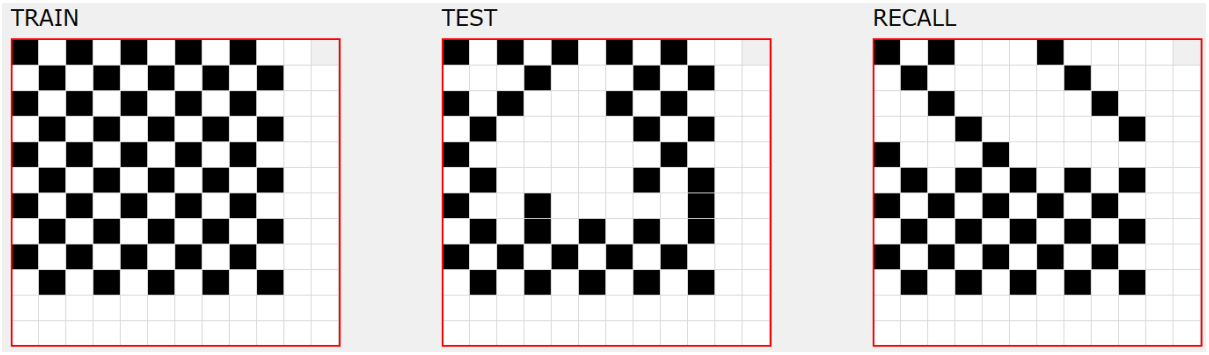
6. Graph 6:



分析以及討論:下半部分的區塊有需補到，但上方仍舊有更多受損的片段，可能的原因有: 1.此破損照對應到的是一半的棋盤格照片，因此下半部的棋盤格回想的成功，但上半部的回想，模型可能與其他棋盤格圖片混淆了

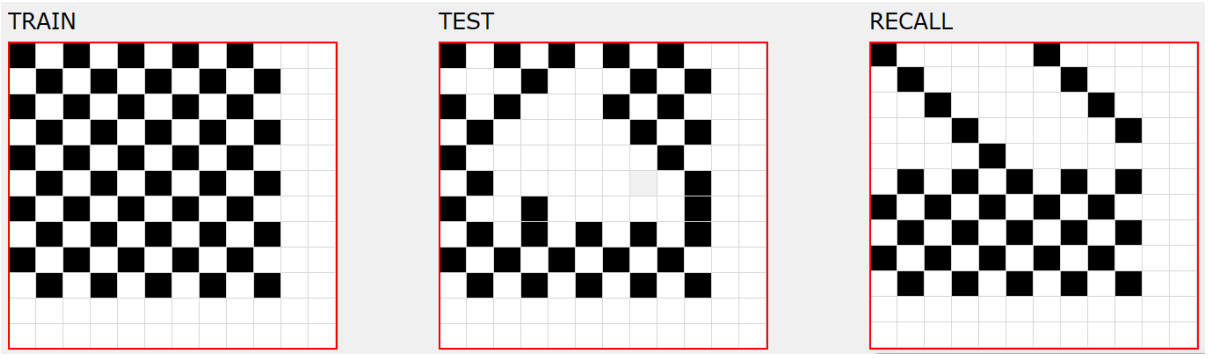


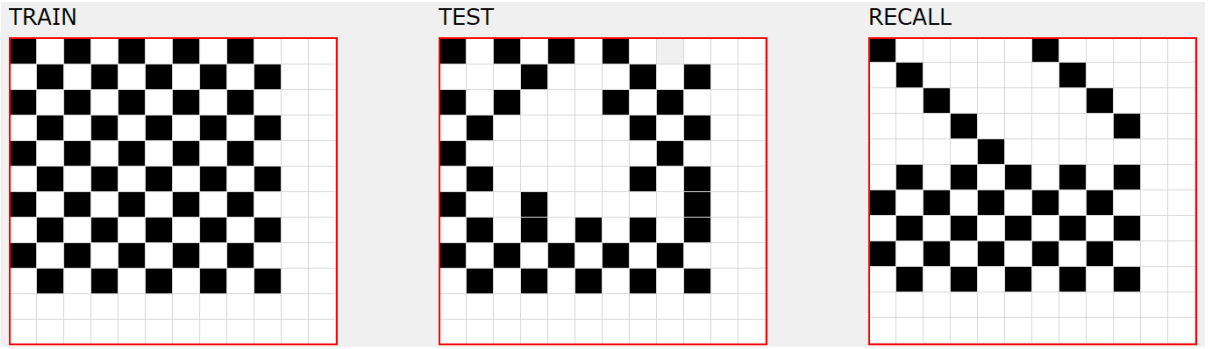
7. Graph 7:



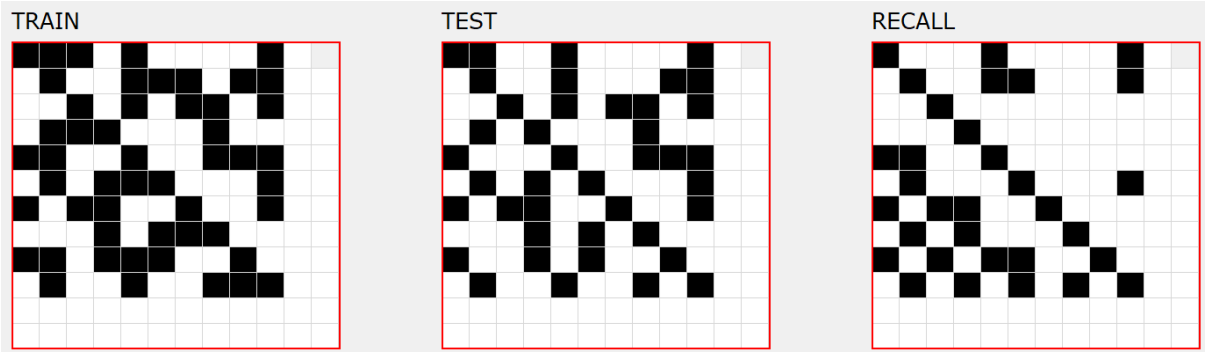
分析以及討論:可以看到回想得不錯，不錯的點在於: 他有將圖片重新回想成棋盤

格的黑白相間，但可惜的是他只修復到中間的斜線，推測是因為中間的斜線，在我們的訓練集中出現率相當高

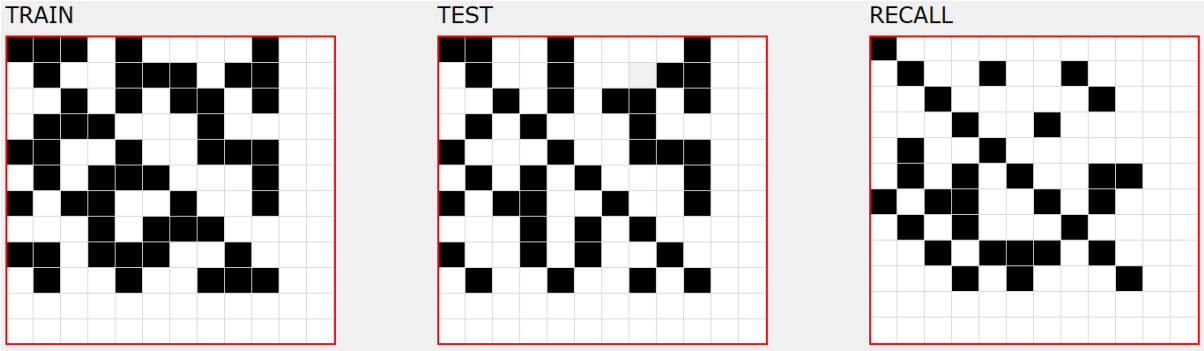


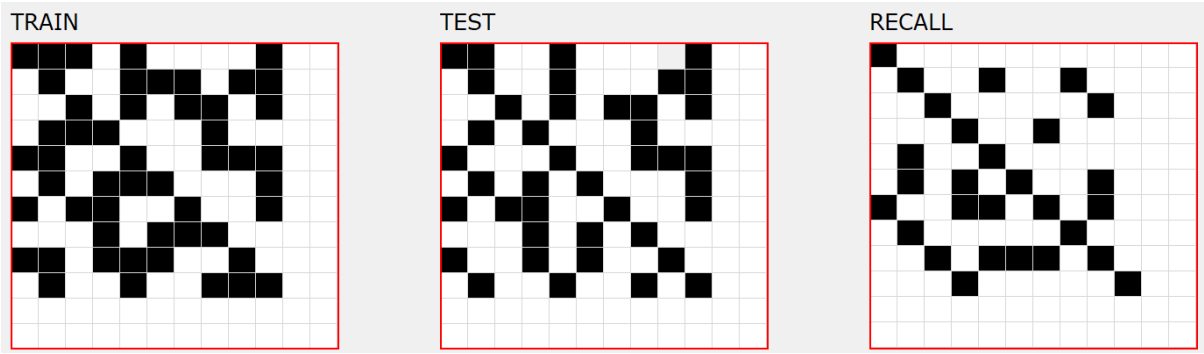


8. Graph 8:

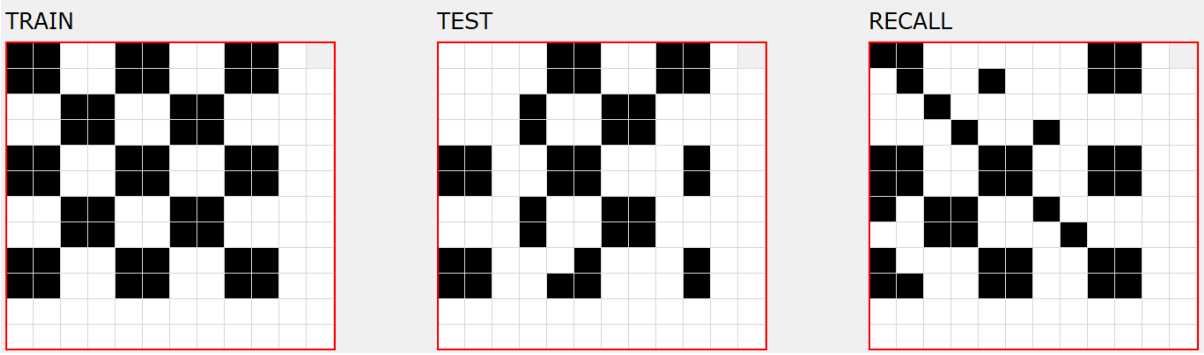


分析以及討論:可以看到回想得不好，右上方該有的特徵都被破壞掉了，推測是因為我們要回想的照片與其他訓練集的照片相比，更沒有規律可言，因此模型無法回想

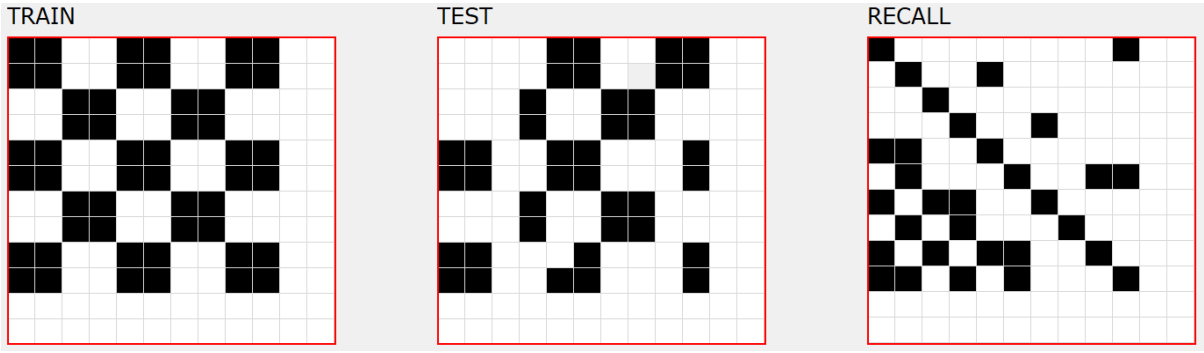


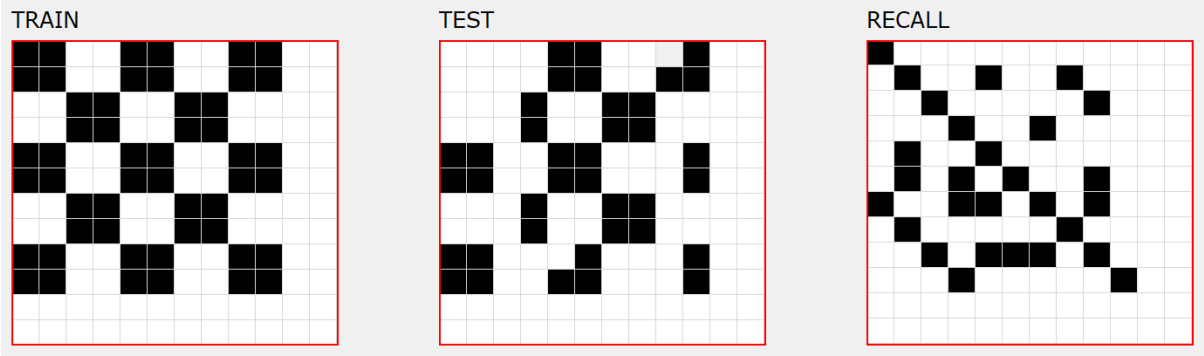


9. Graph 9:

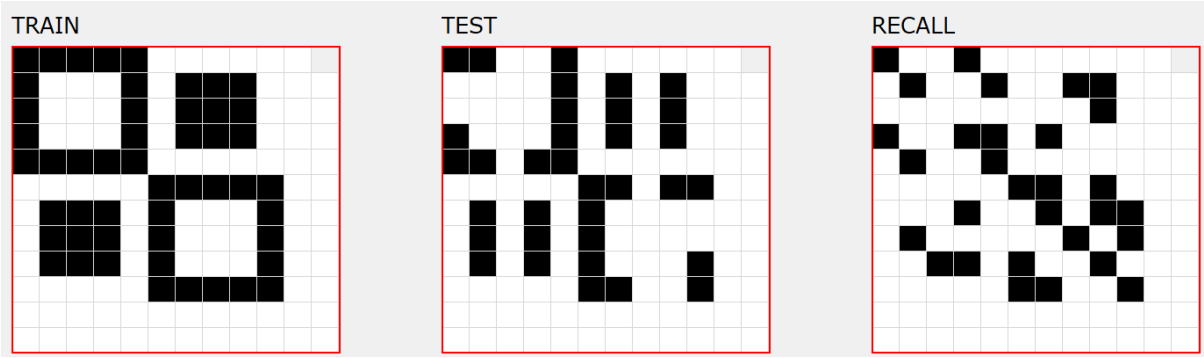


分析以及討論:可以看到回想得不錯，不錯的點在於: 他有將大部分的 2x2 正方形恢復，但可惜點與上述相同，仍舊有一些因棋盤格訓練資料造成的雜訊出現，最明顯的是從左上到右下的斜線，黑色的區塊被破壞成白色

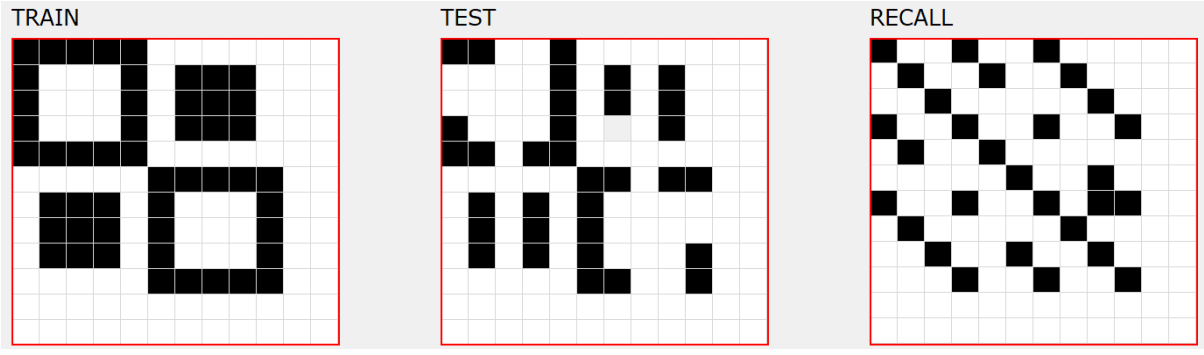


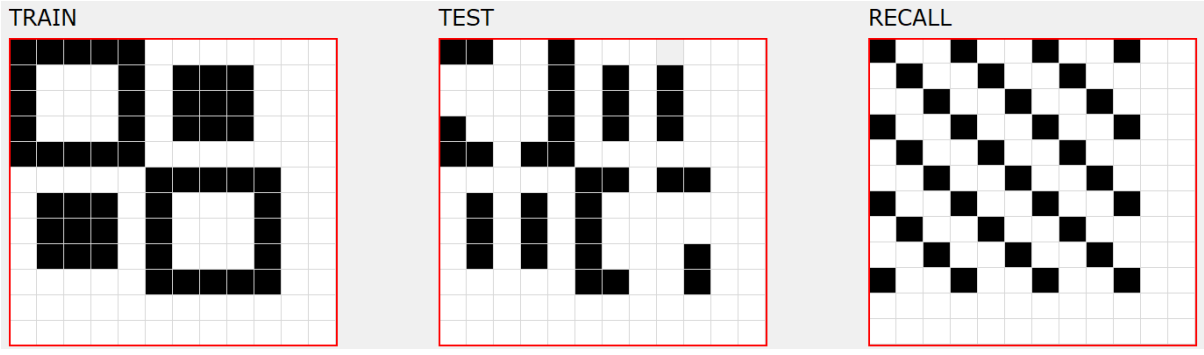


10. Graph 10:

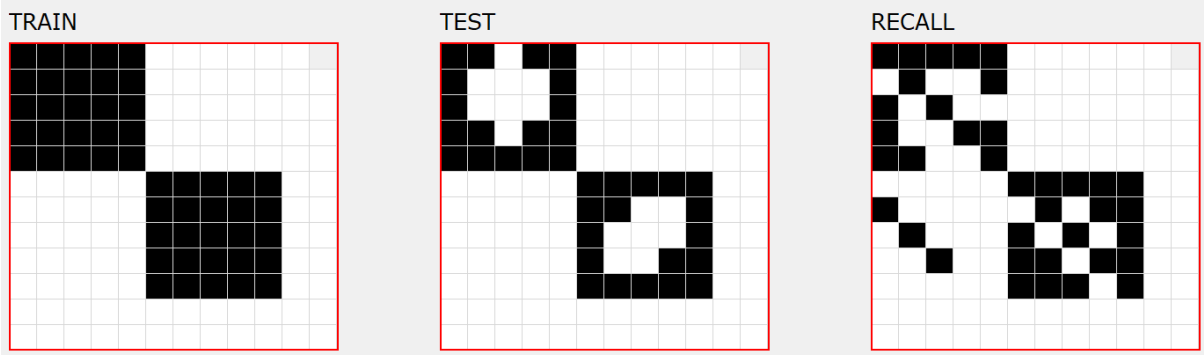


分析以及討論:可以看到回想得不好，推測的原因為 1. 這次要恢復的照片與其他照片相違背太多，比如說左上和右下的正方形為空心，然後右上和左下多了特有的實心正方形 2. 此圖形特別容易受到棋盤格雜訊的影響

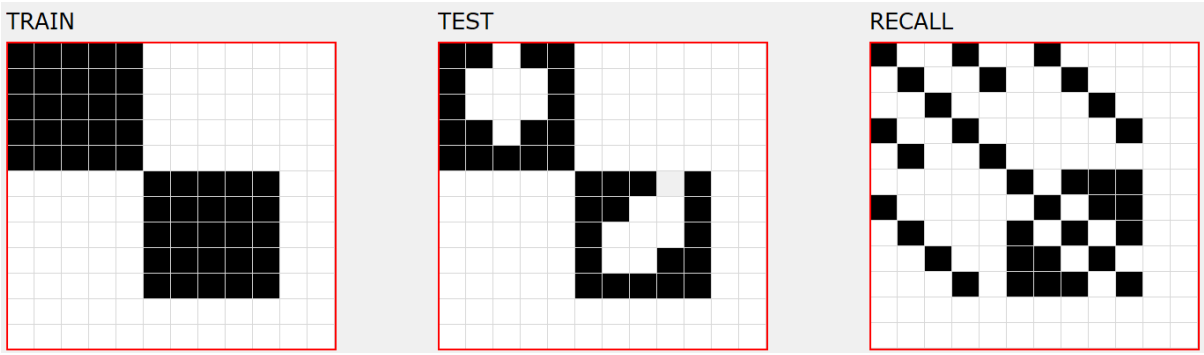


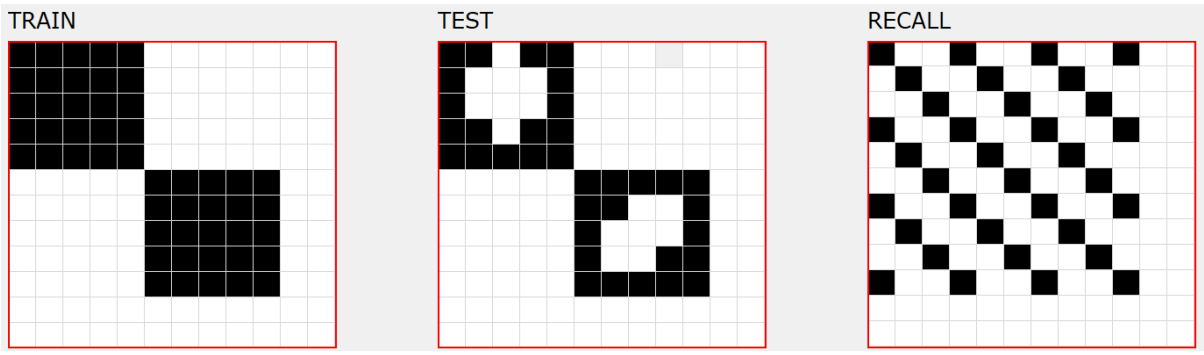


11. Graph 11:

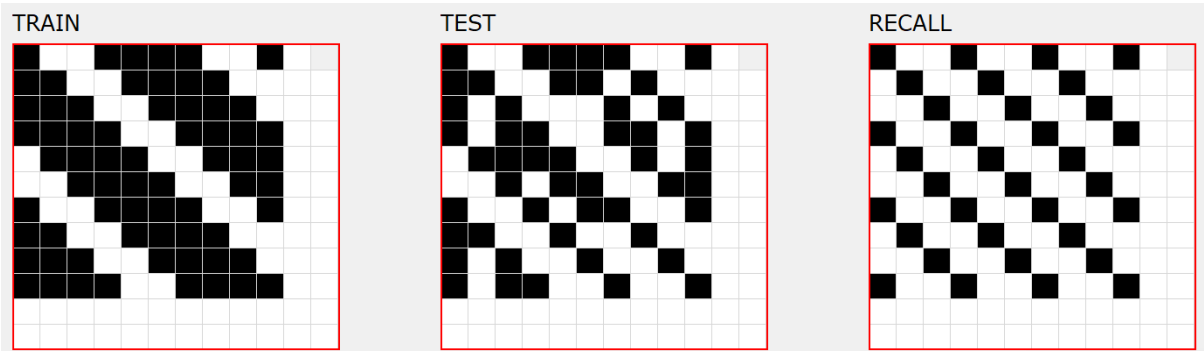


分析以及討論:可以看到回想得不錯，大致的形狀還在，但仍舊有明顯地受到棋盤格圖案的影響，依照棋盤格的排列被破壞了。

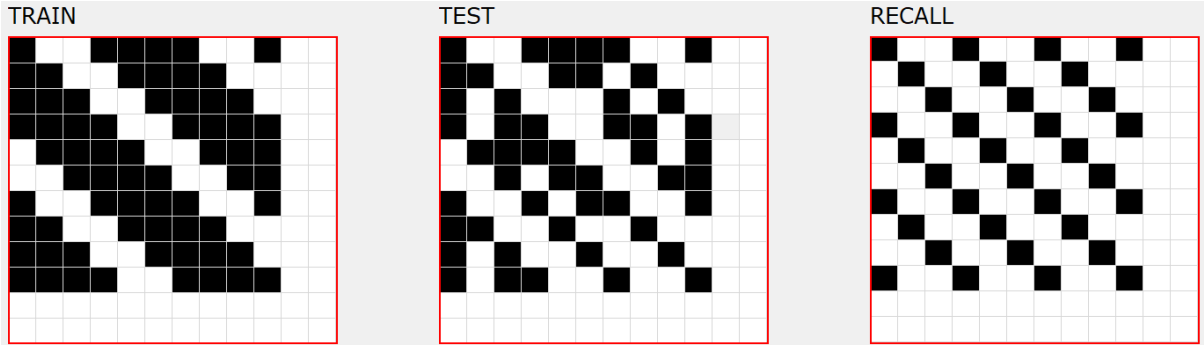


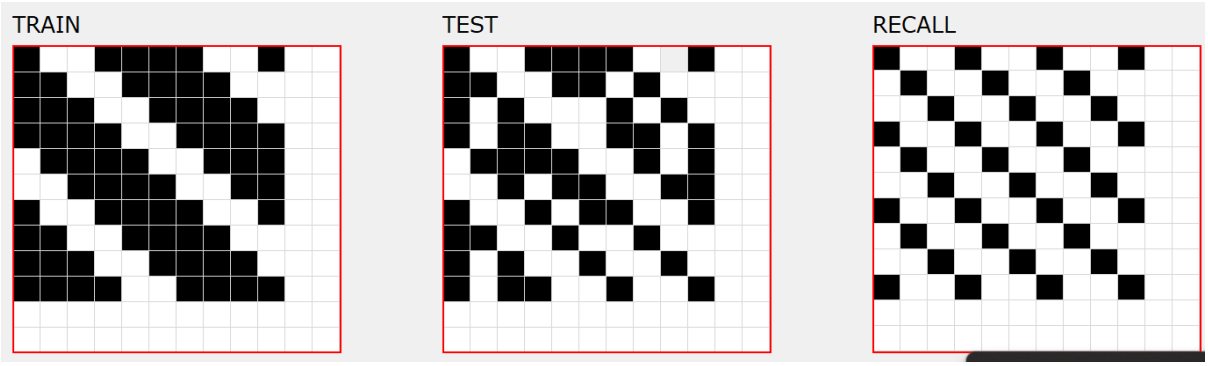


12. Graph 12:

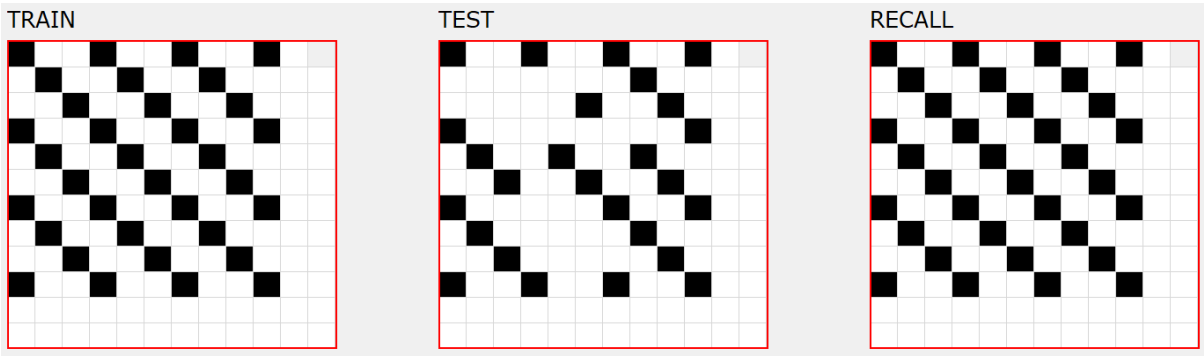


分析以及討論:可以看到回想得很失敗，儘管肉眼看得出訓練資料與測試資料的相同性，但模型在回想時，反倒是回想出了另一張圖片。

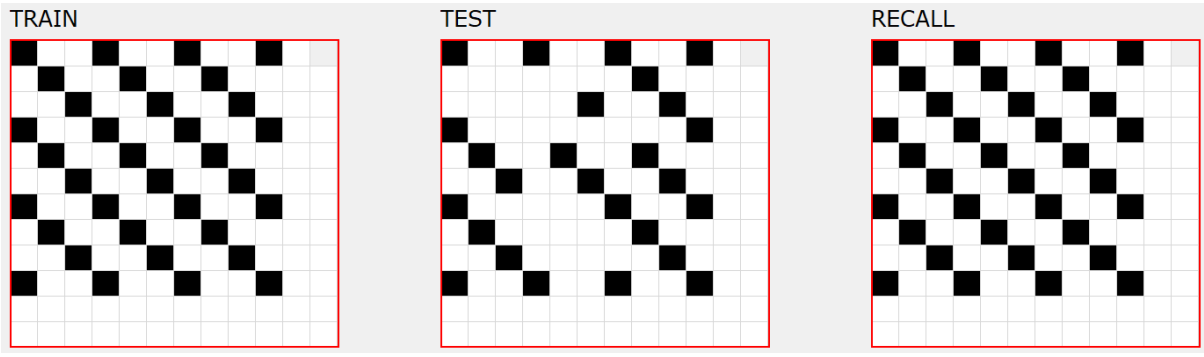


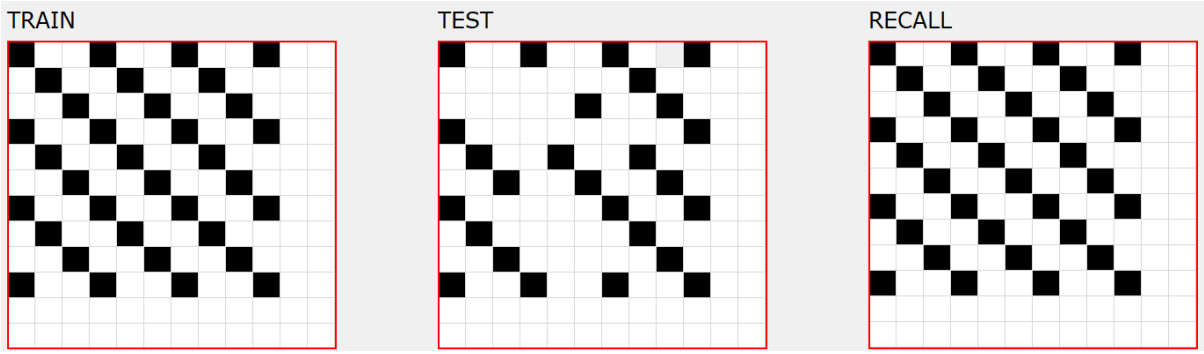


13. Graph 13:

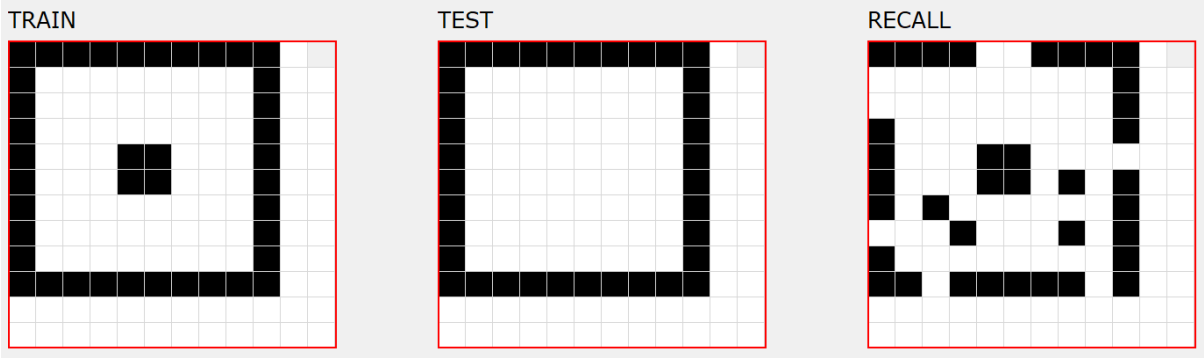


分析以及討論:可以看到回想得很成功， 原因可能為，此圖片很明顯為棋盤格圖片的破損狀，但重要的點是黑色方格的位置都是正確的，這一定程度的增加了回想的成功性。

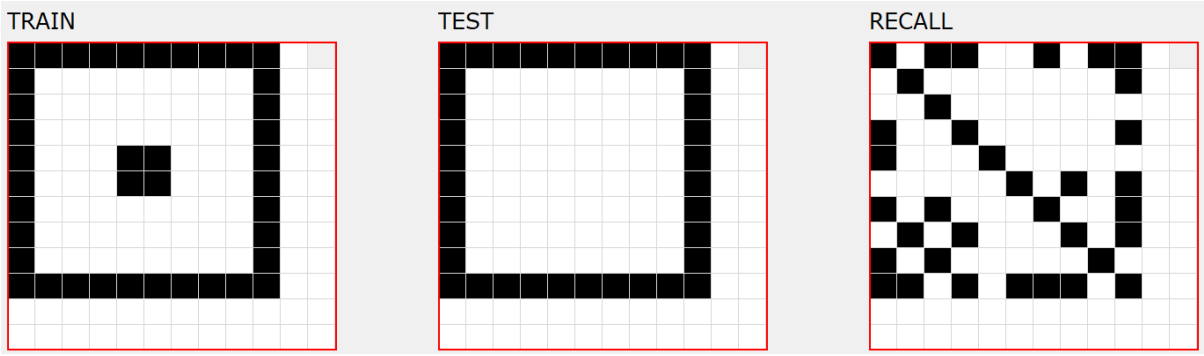


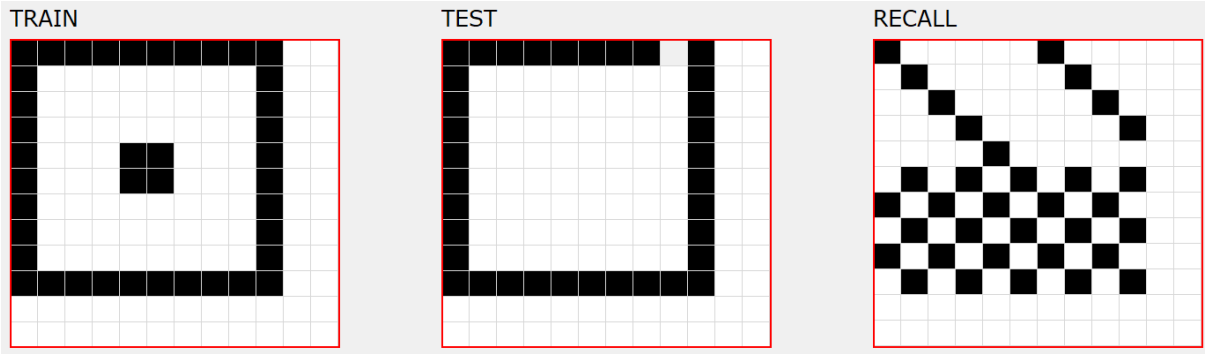


14. Graph 14:

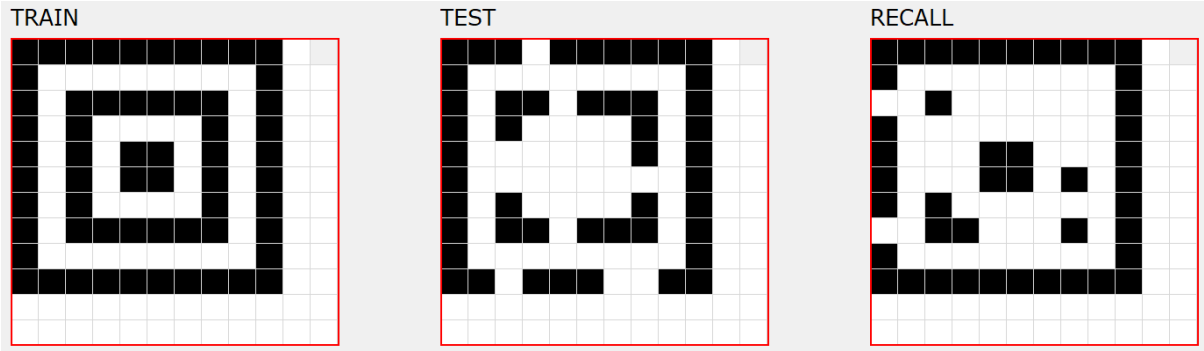


分析以及討論:可以看到回想得很不錯，模型依照毀損照片外框的完整性，回想初了中間那一個方形。

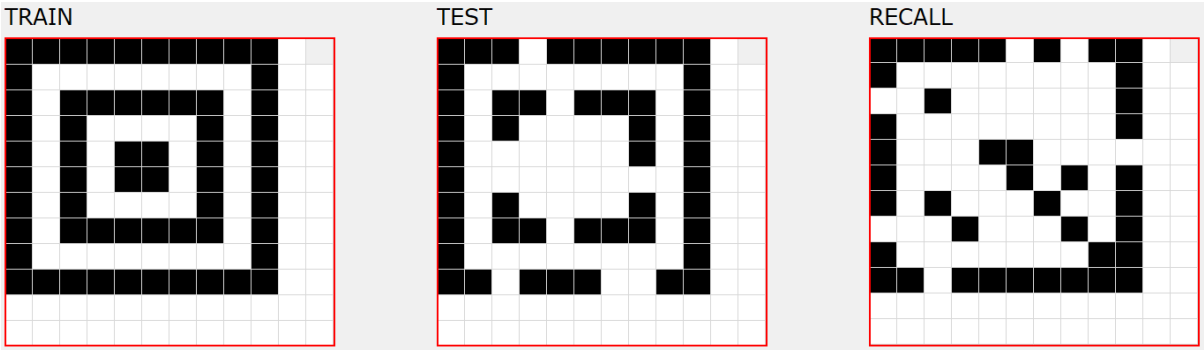


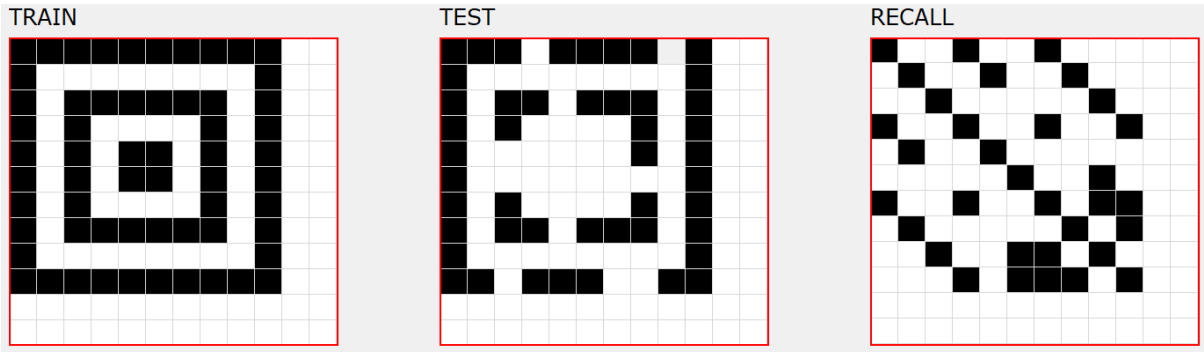


15. Graph 15:



分析以及討論:可以看到回想得不錯，雖然中間的方形得到了修復，但內層的框框受到了毀損，可以推測模型對於外層的框框以及中間的方形更為熟悉。





E. 延伸討論與比較

1. Basic 與 Bonus 成效不同:
    1. Bonus 有著比 Basic 更為複雜的訓練圖片，更別提這些訓練圖片彼此之間也存在著局部的相似性，因此 Bonus 的模型，更容易混淆。
2. Bonus 在 epoch 增加時的回想成效顯著降低:
    1. 我們可以看到 epoch 從 1 到 3 時，回想的圖片已經有明顯的破損
    2. 甚至於將 epoch 設為 10 後，除了 Graph 1,4,8,13 之外，其餘的全部被回想呈棋盤狀，由此可觀察出訓練集中存在的棋盤狀圖片，嚴重的混淆了我們的模型，最明顯的是有許多我們肉眼可見的方框或方塊圖騰都被破壞了
    3. 但反之，graph 1,4,8,13，並未受到 epoch 數增加的影響
3. Graph3&11 修復結果:

    1. 這兩者的破損狀況不同，但從回想結果再再證明了棋盤格圖片的影響，因那些斜線狀都有受到恢復，破壞的部分也是斜線狀
4. Graph5&14&15 修復結果:
    1. 從結果可以看出中間的方格，以及最外圈的方框是模型比較熟悉的
5. Graph3&11 修復結果:
    1. 這兩者的破損狀況不同，但從回想結果再再證明了棋盤格圖片的影響，因那些斜線狀都有受到恢復，破壞的部分也是斜線狀
6. 總結可能造成成效不佳的因素: 1. 不足的訓練樣本 2. 特徵選擇 3. 數據不平衡 4. Epoch 數過多 5. 模型能力有限

F. 加分項:

1. 我讓使用者可以去改變 epoch，進而獲得更全面的觀察，也因此在報告中多了很大的篇幅在討論 epoch 的影響。
2. 我花了許多時間在思索如何呈現結果，最後學習 table 的使用方法，也以清晰好懂且一鍵式的方式，一次呈現出了三張相關的圖。
3. Bonus 的資料集成果也相當不錯。