學號：B05901038　　系級：電機四　　姓名：陳元芃

1. **請說明你實作的 CNN 模型(best model)，其模型架構、訓練參數量和準確率為何？(1%)**

(a) 模型架構：

```
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        # torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)
        # torch.nn.MaxPool2d(kernel_size, stride, padding)
        # input 維度 [3, 128, 128]
        self.cnn = nn.Sequential(
            nn.Conv2d(3, 64, 3, 1, 1),  # [64, 128, 128]
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),      # [64, 64, 64]

            nn.Conv2d(64, 128, 3, 1, 1), # [128, 64, 64]
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),      # [128, 32, 32]

            nn.Conv2d(128, 256, 3, 1, 1), # [256, 32, 32]
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Conv2d(256, 256, 3, 1, 1), # [256, 32, 32]
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),      # [256, 16, 16]
            nn.Conv2d(256, 512, 3, 1, 1), # [512, 16, 16]
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.Conv2d(512, 512, 3, 1, 1), # [512, 16, 16]
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),       # [512, 8, 8]

            nn.Conv2d(512, 512, 3, 1, 1), # [512, 8, 8]
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.Conv2d(512, 512, 3, 1, 1), # [512, 8, 8]
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),       # [512, 4, 4]
        )
        self.fc = nn.Sequential(
            nn.Linear(512*4*4, 1024),
            nn.Dropout(0.5),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.Dropout(0.5),
            nn.ReLU(),
            nn.Linear(512, 11),
        )
```

```
    def forward(self, x):
        out = self.cnn(x)
        out = out.view(out.size()[0], -1)
        return self.fc(out)
```

此 model 架構是根據助教的 sample code 再做修改。助教原本的 model 架構有五批的「Conv2D->MaxPool2d」結構，我將後三批當中的 Conv2D 都重複兩次，output filter 數則分別都維持前一層 input 的 filter 數(分別為 256, 512, 512)。此外，於最後一層以外的 Linear 層後都緊接上一層 Dropout Layer，其中 dropout rate 都設為 0.5。

```
train_transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.RandomHorizontalFlip(), # 隨機將圖片水平翻轉
    transforms.RandomRotation(15), # 隨機旋轉圖片
    transforms.ToTensor(), # 將圖片轉成 Tensor，並把數值 normalize 到 [0,1] (data nor
])
test_transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.ToTensor(),
])
```

Input 的圖片則是有經過和 sample code 相同的 data augmentation 和 data normalization。(於 5.會詳述)

```
loss = nn.CrossEntropyLoss() # 因為是 classification task，所以 loss 使用 CrossEntropyLoss
optimizer = torch.optim.Adam(model.parameters(), lr=0.001) # optimizer 使用 Adam
num_epoch = 75
```

Loss 使用 cross entropy loss。optimizer 使用 Adam，其中 learning rate=0.001。epoch 則設為 75 個((c)會詳述)。

(b) 參數量：18,146,059

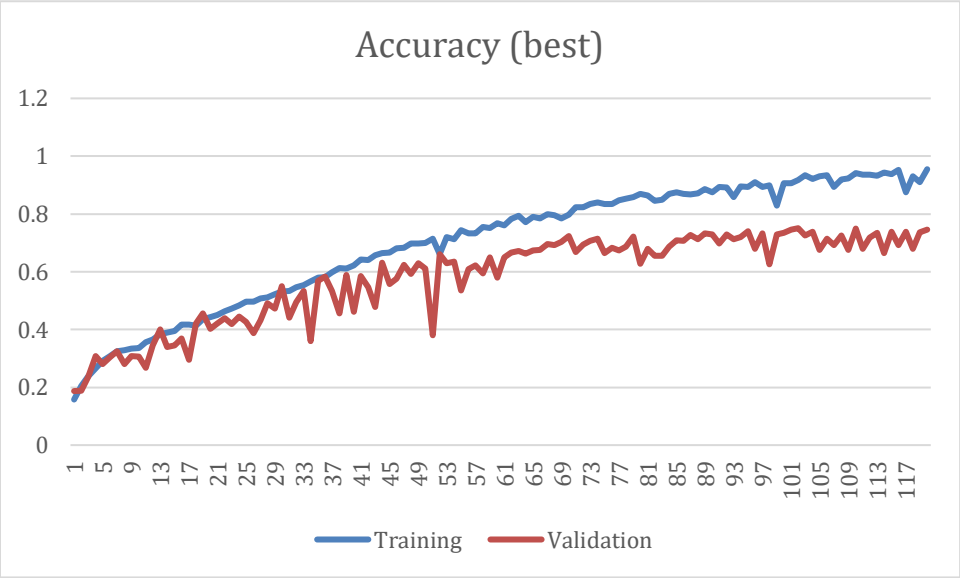
```
pytorch_total_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
```

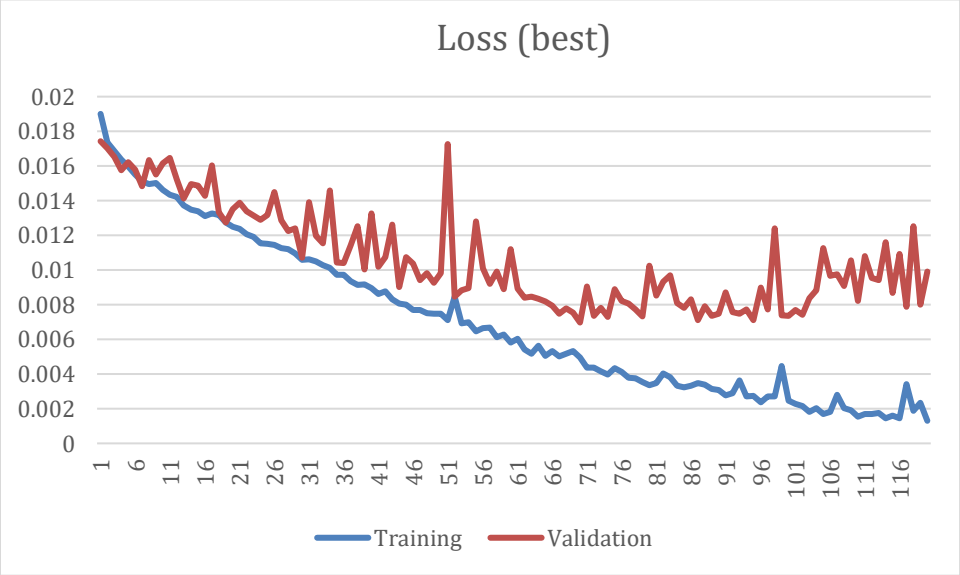
可由以上指令得出。

(c) 準確率：

將 model 在 training set 上 train 過 120 個 epoch 後，accuracy 和 loss

分別對 epoch 作圖如下：





由 validation set 上的 accuracy 和 loss 可以看出 model 的確在 75 個 epoch 左右時即收斂。其中 model 在 75 個 epoch 後，validation accuracy=0.724198。

而將此 model structure 在 training set 和 validation set 上一起訓練 75 個 epoch 後的 model，於 Kaggle 上則是有 0.81171 的準確率。推測是因為 validation set 本身的 distribution 就和 training set 有差異，然而

testing set 和 training+validation set 的 distribution 則是較相近的，因此相同的 model structure 在 training set 上和 training+validation set 上 train 出的結果才會有將近 0.1 左右的準確率誤差。

2. 請實作與第一題接近的參數量，但 CNN 深度（CNN 層數）減半的模型，並說明其模型架構、訓練參數量和準確率為何？(1%)

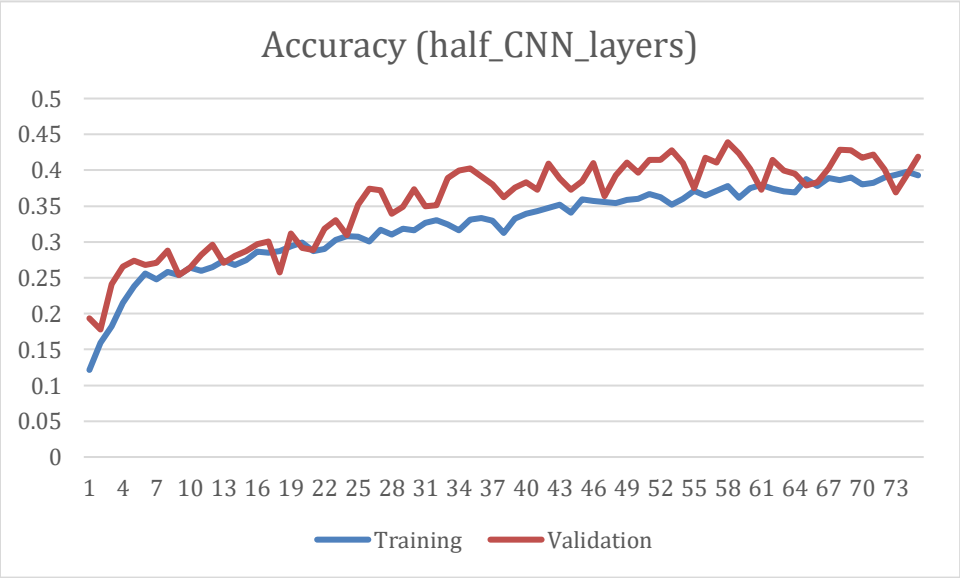
```
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        # torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)
        # torch.nn.MaxPool2d(kernel_size, stride, padding)
        # input 維度 [3, 128, 128]
        self.cnn = nn.Sequential(
            nn.Conv2d(3, 64, 3, 1, 1),  # [64, 128, 128]
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),      # [64, 64, 64]

            nn.Conv2d(64, 128, 3, 1, 1), # [128, 64, 64]
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),      # [128, 32, 32]

            nn.Conv2d(128, 256, 3, 1, 1), # [256, 32, 32]
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Conv2d(256, 256, 3, 1, 1), # [256, 32, 32]
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),      # [256, 16, 16]
        )
        self.fc = nn.Sequential(
            nn.Linear(256*16*16, 262),
            nn.Dropout(0.5),
            nn.ReLU(),
            nn.Linear(262, 48),
            nn.Dropout(0.5),
            nn.ReLU(),
            nn.Linear(48, 11),
        )

    def forward(self, x):
        out = self.cnn(x)
        out = out.view(out.size()[0], -1)
        return self.fc(out)
```

原本的 model 有 8 層 convolution，因此這邊只使用 4 層，其中第 3 和第 4

層連續作用之後才會接到 maxpooling。此外為維持參數量相近，fully connected layer 的 dimension 也稍作調整。原本為「4096->1024->512->11」，此處因為少了兩次的 Maxpooling，所以即使 filter 數較少，flatten 後的維度仍高於 4096 不少，因此第一層的 fully connected layer 只用了 262 維，第二層則是 48，即「65535->262->48->11」。Data augmentation、data normalizaion、optimizer、learning rate 和 epoch 數目等都維持和 1. 相同。

參數量：18,146,161

準確率：在 training set 上 train 75 個 epoch 後，validation accuracy=0.419242。



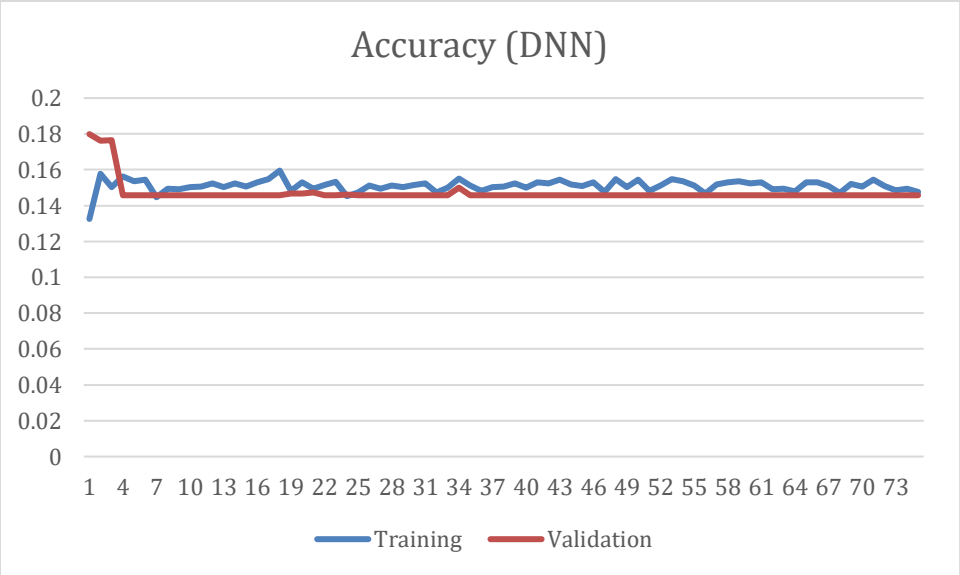
3. 請實作與第一題接近的參數量，簡單的 DNN 模型，同時也說明其模型架構、訓練參數和準確率為何？(1%)

```
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        self.fc = nn.Sequential(
            nn.Linear(3*128*128, 368),
            nn.Dropout(0.5),
            nn.ReLU(),
            nn.Linear(368, 151),
            nn.Dropout(0.5),
            nn.ReLU(),
            nn.Linear(151, 11),
        )

    def forward(self, x):
        #out = self.cnn(x)
        out = x.view(x.size()[0], -1)
        return self.fc(out)
```

與 1.和 2.當中的 fully connected 架構一樣是三層，其 dimension 分別為「49152->368->151->11」。

參數量：18,145,695

準確率：在 training set 上 train 75 個 epoch 後，validation accuracy= 0.145773。



4. 請說明由 1 ~ 3 題的實驗中你觀察到了什麼？(1%)

   可觀察到 CNN 確實在圖像相關的 task 上有好處，在相同的參數量下，可以更有效率的利用參數，extract 出更多有用的 feature。1.的 CNN 層數有 8 層，準確率在 validation set 上可達到 70%以上；2.只有 4 層 CNN，可能因此 extract 不出太細節的 feature，導致準確率只有 40%左右；3.直接使用 DNN，可看出幾乎是 train 不太起來，在 5 個 epoch 以後左右 validation accuracy 便不再變動，此外 accuracy 0.145773 只比 random(0.0909)好一點而已。

5. 請嘗試 data normalization 及 data augmentation，說明實作方法並且說明實行前後對準確率有什麼樣的影響？(1%)

如 1.中所述，

```
train_transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.RandomHorizontalFlip(), # 隨機將圖片水平翻轉
    transforms.RandomRotation(15), # 隨機旋轉圖片
    transforms.ToTensor(), # 將圖片轉成 Tensor，並把數值 normalize 到 [0,1] (data nor
])
test_transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.ToTensor(),
])
```

RandomHorizontalFlip()可將圖片隨機水平翻轉，RandomRotation(15)則可將圖片隨機旋轉-15°、15°之間的一個角度。

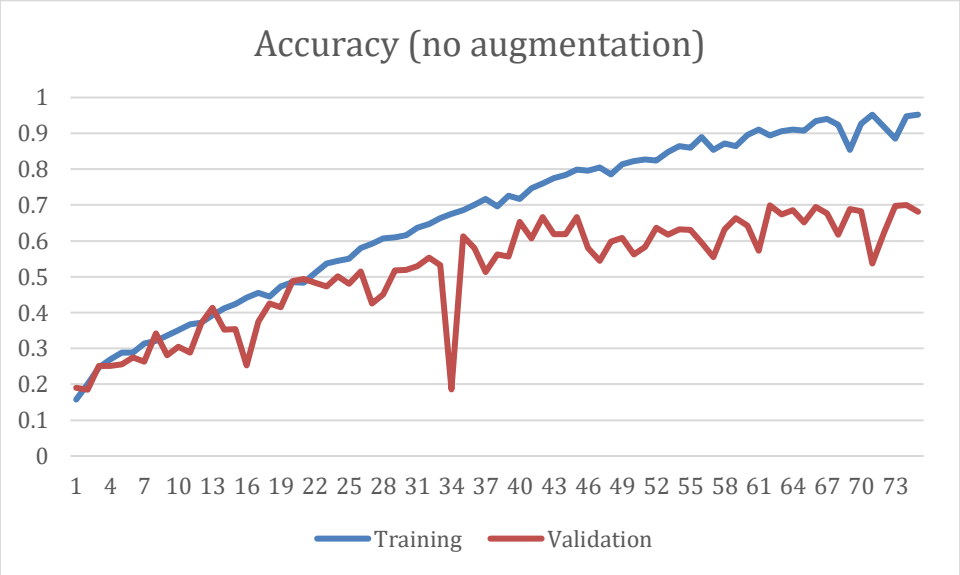
ToTensor()可將圖片 normalize 到 0~1 之間，可於 source code 中看到其其實只是直接將圖片的值全部除以 255。。

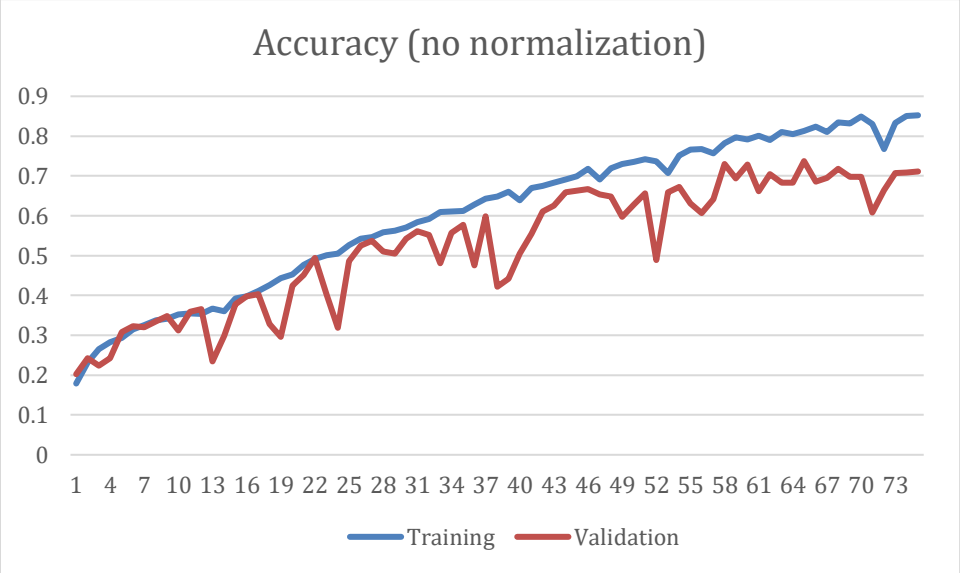
```
if isinstance(pic, np.ndarray):
    # handle numpy array
    if pic.ndim == 2:
        pic = pic[:, :, None]

    img = torch.from_numpy(pic.transpose((2, 0, 1)))
    # backward compatibility
    if isinstance(img, torch.ByteTensor):
        return img.float().div(255)
    else:
        return img
```

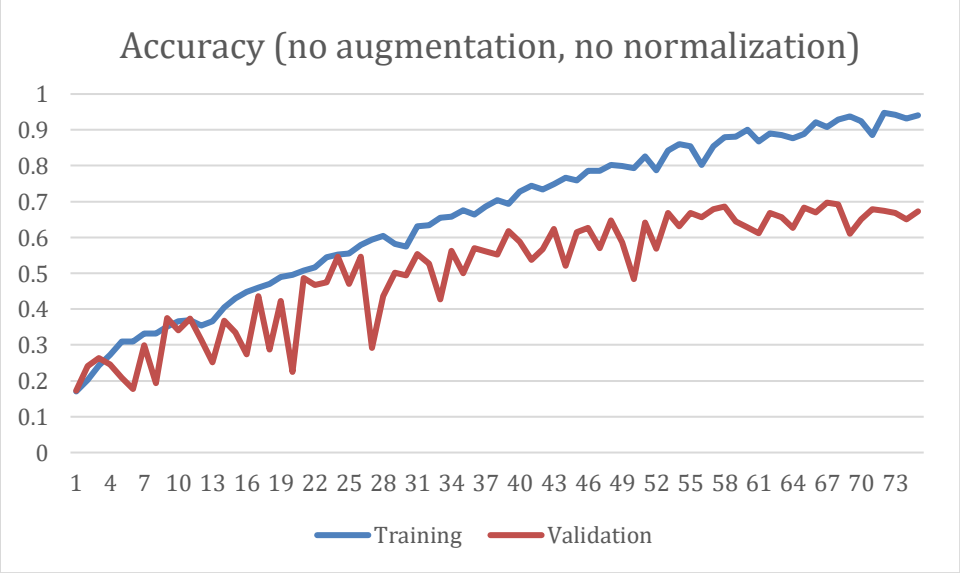
注意到如果不使用 ToTensor()將 PIL Image 轉成 tensor 的話，需要手動轉換。

```
class ImgDataset(Dataset):
    def __init__(self, x, y=None, transform=None):
        self.x = x
        # label is required to be a LongTensor
        self.y = y
        if y is not None:
            self.y = torch.LongTensor(y)
        self.transform = transform
    def __len__(self):
        return len(self.x)
    def __getitem__(self, index):
        X = self.x[index]
        if self.transform is not None:
            X = self.transform(X)
        X = np.array(X)
        X = np.transpose(X, (2,0,1)).astype("float32")
        X = torch.from_numpy(X)
        if self.y is not None:
            Y = self.y[index]
            return X, Y
        else:
            return X
```

準確率：


Accuracy (no augmentation)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73
Training
Validation


Accuracy (no normalization)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73
Training
Validation


Accuracy (no augmentation, no normalization)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73
Training
Validation

Accuracy (epoch=75)	No data augmentation	Data augmentation
No data normalization	0.672886	0.711662
Data normalization	0.681050	0.724198

可見 data augmentation 和 data normalization 都可幫助提升 accuracy，其中 data augmentation 的效果又比 data normalization 顯著。此外，可以從曲線看到，沒有 data augmentation 的話，training accuracy 和 validation accuracy 之間的差距會更明顯，可見 data augmentation 對圖片做一些隨機的變化，真的可有效幫助 model 適應不同的 data，減少 overfit 的程度。

6. 觀察答錯的圖片中，哪些 class 彼此間容易用混？[繪出 confusion matrix 分析](1%)

(confusion matrix 使用套件 seaborn 繪製，然而此部分的 code 並不包含於 reproduce 時所需跑的 code)

此 confusion matrix 乃 1.的 model 在 training set 上 train 75 個 epoch 後，在 validation set 上得出的結果所繪製。

(a) Class 1 (Dairy product) 很容易被誤判成 Class 2 (Dessert)

一個可能的原因是：Dairy product 當中的起司看起來和 Dessert 當中的起司蛋糕極為相像。(左圖：training/1_209.jpg，右圖：training/2_82.jpg)

(b) Class 7 (Rice) 很容易被誤判成 Class 6 (Noodles/Pasta)

可能的原因是：麵和飯主要組成都是澱粉，因此色澤上看起來會比較像；盛盤方式也都是麵/飯為主體佔滿一整個碗盤；此外麵和飯都有線條狀的特徵，即使飯的線條比較短，而麵的比較長，但 model 似乎沒有區別出這一點。(左圖：training/6_423.jpg，右圖：training/7_12.jpg)

(c) Class 3 (Egg) 很容易被誤判成 Class 0 (Bread)

一個可能的原因是：training set 當中，蛋的圖片很多都是將蛋放在麵包上。(左圖：training/3_951.jpg，右圖：training/3_980.jpg)