```
datagen = ImageDataGenerator(
    featurewise_center=False,  # set inp
    samplewise_center=False,  # set each
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,  # apply ZCA wh
    rotation_range=0,  # randomly rotate
    width_shift_range=0.1,  # randomly sh
    height_shift_range=0.1,  # randomly
    horizontal_flip=True,  # randomly fl
    vertical_flip=False)  # randomly fli
```

1. supervised learning

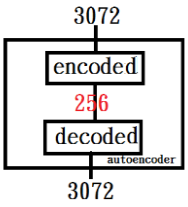
這裡首先是嘗試做 keras 提供的 cifar10 CNN 架構範例：Convolution2D→ Convolution2D→MaxPooling2D→Convolution2D →Convolution2D→MaxPooling2D→Flatten→Dense(512)→Dense(10)，過程中三次 dropout 的比例依序為 0.25、0.25 與 0.5，Activation 皆為 relu(最後一次是 sigmoid)，另外 loss 是用 categorical_crossentropy 計算，optimizer 是用 SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)，metrics=['accuracy']，因為 train set 較小所以也有使用 data augmentation 做資料的前處理(設定如右上圖)。

這個 model 我覺得蠻適合用來測試 kaggle 的 acc 和自己做 validation set 的 acc 變化是否一致，因為它架構簡單、train 的速度蠻快，又可以了解若當之後在更複雜的模型上較難做 validation set 去判斷 acc 時(因為一 train 就要 train 好幾個小時)、是否可以相信 kaggle public set 的分數。而根據自己嘗試 train 了多種不同 epoch 和 batch_size 的模型(各種不同 epoch 和 batch_size 的組合的 performance 在第 4 題呈現)預測出的結果在 public set 的 acc 和自己做 validation set(在這裡是隨機從 5000 個 labeled data 中切 500 個出來當 validation set)的 acc 非常相近!雖然 validation set 的 acc 在 train 的過程中有時會變化很大(相鄰 epoch 有時 acc 可相差達 0.05 且上下來回跳的頻率很高，不像 train set 的 acc 變化方向較明確或較穩定)，但是我發現 validation set 的 acc 與 kaggle public set 的 acc 通常相差不到 0.02，所以我覺得這次應該 public 與 private set 的 acc 不會差太多，因此在後面較複雜的模型上便以 kaggle public set 的 acc 當作 performance 考量而不另外切 validation set。

2. semi-supervised learning: cluster by autoencoder

a.方法：這題的分類方式主要是先藉由將 60000 個 examples (包括 train set 中的 5000 個 labeled examples、45000 個 unlabel examples 與 10000 個 test set 中的 unlabel examples，每個 example 有 3072 個 features，可視其為一個 1*3072 的 vector)降維使其可以用更少的 features 表示(作業中是做成 256 個 features，1*256 的 vector)，再利用這些較少的新 features 對 45000 個 unlabel examples 進行分類，再將分類效果較好的 unlabel examples 加進 train set(一開始的 train set 只有 5000 個 label examples)，再利用新的 train set 去 train 一個 model，最後再用這個 model 預測 test set 的 10000 個 examples。

b. Autoencoder：Autoencoder 的架構由一層 encoded 和一層 decoded 所組成：encoded 輸出的每個 data 的長度(output_dim)是 256、輸入的每個 data 的長度是 3072，使用的 activation 為 relu；而 decoded 輸出的每個 data 的長度(output_dim)是 3072、輸入的每個 data 的長度是 256，使用的 activation 為 sigmoid(因為 decoded 是 model 的最後一層)。Autoencoder 在 learning 過程中使用 binary crossentropy 計算 loss 並搭配 Adadelta 調整前進速度，epoch 設 1000、batch size 設 20。因為這裡只需要拿 train 好的 model 去預測 labeled 與 unlabel examples 的 256 codes 是什麼，所以可以單獨把 encoded 從 train 好的 Autoencoder 拿出來轉成 model 的形式(encoder = Model(input=Input(shape=(3072,)), output=encoded))，然後拿來預測 examples 的 codes。



c. Cluster：將 5000 筆 labeled examples 以 Autoencoder 產生的 256 個 codes 為 features 拿去 train k-means、random forest(預設產生十棵 trees)和 K Neighbors Classifier(# of neighbors=3)三種不同的分類模型(input data 大小為 5000*256)，再用這三個分類模型個別預測 45000 unlabel examples，若某 unlabel example 被這三種模型預測的類別皆相同，則會被加進 train set。依此條件，可以額外在 train set(for next model)中加入 1011 個原為 unlabel 之 examples。

d. Model：這裡使用的 model 為 keras 提供的 cifar10 CNN 架構範例(如第一題的敘述)。此 model 使用 cluster 產生的 6011 個 examples 當作 train data，每個 data 使用原本的 3072 features，而每個 data 的 label 有些(1011 個 unlabel examples)是在 cluster 預測的結果或有些是原本就有的。當 epoch=1000、batch_size=50 時，kaggle 的 acc 達 0.64480，而在同樣的 model 架構下，若 train data 只有原來的 5000 個 labeled examples(也就是 supervised learning)，kaggle 的 acc 高達 0.71560，前面第 1 題提到在此 model 下做的 validation 的 acc 與 kaggle 的 acc 差不多，所以可以猜測此分類效果並不太好，很有可能是因為 Autoencoder 產生的 represent code 還不足以清楚地代表這些 train data(也許 epoch 不夠高)，所以在分類上出現如此偏頗的錯誤(都已經用 3 種不同分類模型加強驗證了)。

3. semi-supervised learning：self-learning

這裡實作的方法是用前面提到的 keras 提供的 cifar10 CNN 架構範例做修改，因為有在網路上查到一些做 image 分類的實作方法都使用 LeakyReLU 且 layer 數更多(試過當多加上一層 model.add(Dense(512))時 train set 的進步程度其實不明顯，但若改為 model.add(Dense(1024))則 train set 的 acc 上升速度會變快)，即：架構範例：Convolution2D→ Convolution2D→ MaxPooling2D→Convolution2D→Convolution2D→MaxPooling2D→Flatten→Dense(1024)→Dense(512)→Dense(10)。另外 optimizer 改為 keras 官方提供的 Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)的話 train set 的 acc 上升速度會變快。另外這個改良版的架構的 validation set 的 acc 上升速度很明顯較原版架構快、且 validation set 的 acc 也與 kaggle public acc 差不多，因此可以用較少的 epoch 來提升預測精確度(也就是較省時間)。

這題的 semi-supervised learning 就是基於這個被修改過的架構反覆做 train the train set→predict unlabel datas→add part of unlabel datas into train set，其中在衡量是否要將 unlabel datas 加進 train set 是依據 model 預測的 class 的機率大小所決定，當機率超過一定值的 data 便會被選為下一次 train model 的 data，而這個定值若設的越高則每次加進 train 的 data 就會相對較少，但是也就相對較穩定不易 overfitting，作業中我使用定值 0.97。而中止條件我是設定當剩下的 unlabel data(也就是還沒被加進 train set 的 data)數量少於 15000 個(也就是 train set 之 data 數超過 35000 時)時或這次只加進了不到 1000 筆的 datas 到 train set 中的話，就停止再繼續 train model 去 predict unlabel datas，改為用這個包含一些原為 unlabel datas 的新 train set 去 train model 然後再預測 10000 個 test datas。

一開始 train 5000 筆 labeled datas 的 model 使用的 epoch 為 800、batch_size 為 100，若將此 model 拿來預測 45000 筆 unlabel datas，則可加進約 26000 筆 unlabel datas，若將此 model 拿來預測 test set(其實就是 supervised learning)，在 kaggle 的 public acc 為 0.75660；第二次 train 這約 31000 筆 datas 的 model 使用的 epoch 為 400、batch_size 為 300，預測剩餘的 unlabel data 後可再加進約 8000 筆 datas，若拿此 train 了這約 31000 筆 datas 的 model 去預測 test set，則在 kaggle 的 public acc 為 0.78900 (train set acc=0.93)；因為 train set 的 data 數已經 39000 所以會被中斷繼續 train model 去 predict unlabel datas，而拿這 train 這 39000 datas 的 model(使用的 epoch 為 400、batch_size 為 300)去預測 test set，在 kaggle 的 public acc 為 0.79900。由此約略可看出 self learning 的 performance 較 supervised learning 好，且加進來的 data 數越多效果更明顯。

4. 分析與比較

supervised learning	batch_size/ epoch	keras 提供的 cifar10 CNN 架構範例			第三題用的改良版		
		Kaggle public acc	Kaggle private acc	Train set acc	Kaggle public acc	Kaggle private acc	Train set acc
	100/500	0.71720	0.72460	0.92	0.75160	0.75940	0.85
	100/800	0.72340	0.72360	0.95	0.75660	0.75880	0.91

上表是分析兩種架構的差異，接著來比較以這兩種為基底作 semi-supervised learning 的 performance：當以 keras 提供的 cifar10 CNN 架構範例為基底做 semi-supervised learning，若一開始的 epoch=1000、batch_size=100，則在第一次 train 完 5000 筆 labeled datas 之 model 可以預測出約 30000 筆的 unlabed datas(這些被預測的 class 的機率皆大於 0.9)，若將這 train 完 5000 筆 data 的 model 也拿去預測 test set(其實就是 supervised learning)可得到 kaggle public set acc=0.73140。若再以 epoch=300、batch_size=200 去 train 這有約 35000 筆 data 的 train set 所產生的 model 去預測剩餘的 unlabel data 則又可再加入約 5000 筆 datas。若將這 train 完 35000 筆 data 的 model 也拿去預測 test set 可得到 kaggle public set acc=0.74840。最後若再以 epoch=800、batch_size=200 去 train 這有約 39000 筆 data 的 train set 所產生的 model 去預測 test set 可得到 kaggle public set acc=0.75980。與第三點的結果比較可以發現改良版的效果不論是在 self-learning 的哪個階段所產生的 model 都具有較佳的預測效果。而 cluster by autoencoder 的效果較 self-learning 差很多，後者甚至是不用到 unlabel data 就可以贏過前者，很有可是因為其 train 的次數不夠久導致其壓縮效果不好(represent code 的代表性不那麼充足)。