ML HW3 Report B03901056 孫凡耕

0. Preliminaries:

a) data are preprocessed by dividing 255 so the value of all data is between 0 and 1
b) all Convolution2D layers has filter size 3x3
c) the border mode of all convolution layers is ‘same’
d) all MaxPooling2D and UpSampling have size equivalent to (2, 2)

1. Supervised Learning:

I used the example code of Keras for training cifar10 as my initial model. Then I tried to change the structure of the model to yield better result. The best result I got using supervised learning has an accuracy of about 0.61 after 100 epochs.
The corresponding model is:

```
#first layer
Convolution2D(256, W_regularizer=l2(0.0005))      LeakyReLU(alpha=0.5)
Convolution2D(128, W_regularizer=l2(0.0005))      LeakyReLU(alpha=0.4)
MaxPooling2D()                                    Dropout(0.2)
#second layer
Convolution2D(32, W_regularizer=l2(0.001))        LeakyReLU(alpha=0.3)
Convolution2D(32, W_regularizer=l2(0.001))        LeakyReLU(alpha=0.2)
MaxPooling2D()                                    Dropout(0.3)
#third layer
Convolution2D(16, W_regularizer=l2(0.0008))       LeakyReLU(alpha=0.1)
Convolution2D(16, W_regularizer=l2(0.0009))       LeakyReLU(alpha=0.1)
MaxPooling2D()                                    Dropout(0.35)
#feed-forward
Flatten()
Dense(256, W_regularizer=l2(0.001)) LeakyReLU(alpha=0.05) Dropout(0.4)
Dense(52, W_regularizer=l2(0.002))  LeakyReLU(alpha=0.05) Dropout(0.5)
#output
Dense(10)     Activation('softmax')
#compiling parameters
compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

2. Semi-Supervised Learning – Self-Training:

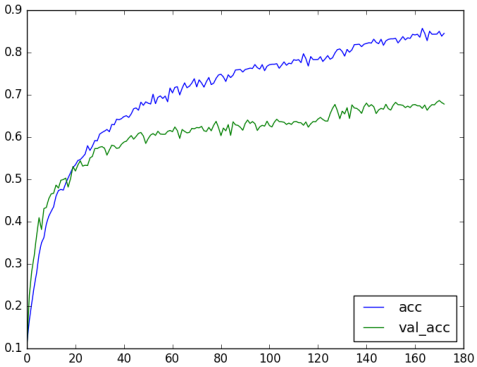
The same model in supervised learning is used again in self-training. However, after training for 100 epochs, I will tried to add more data from the unlabed set if the prediction of the highest probability is over 0.96. Then, the model will be trained for another 25 epochs. The model will be trained in five prediction-add_data-training cycle before eventually converges and stops training.

3. Semi-Supervised Learning – Autoencoder:

The encoding part of my autoencoder consists of three layers of convolution2D, with 64, 32, 16 numbers of 3x3 filters. So the size of the encoded representation is (16, 4, 4). I uses LeakyReLU as the activation function. After every activation function, I applied maxpooling2D. The decoding part is the reverse of the encoding part but the weight is not shared. The alpha value of the LeakyReLU in the decoding part are smaller than those in the encoding part. At last, there is a 3x3 convolution2D with 3 filters and sigmoid function(so the output will be between 0 and 1). The autoencoder are trained with all data(including unlabel and label but not test) with optimizer rmsporp and mean squared error for 50 epochs.

After training the autoencoder, I am able to get the representation of all 50000 images. Then, I used the LabelPropagation class from sklearn.semi_supervised to label all images according to the 5000 labelled images. My LabelPropagation class uses k-nearest neighbors algorithm with k = 3 and the max iterations = 1. After all images are labelled, I used the same model as in the supervised training to train on all 50000 data.

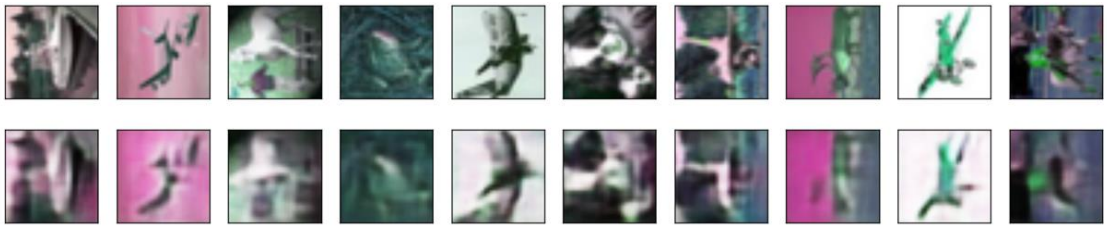
3. Compare and Analyze:



From the graph on the left hand side, we can see that the validation accuracy only goes up a little bit after self-training (after 125 epochs) the validation accuracy goes up a little bit. This is mainly due to the enlargement of dataset. I believe self-training can perform better if the weight of the originally labelled data and newly labelled data have different weight. Also, using a larger model could possibly performs better since there are much more data.

For the autoencoder part, there is some problem with my code. It seems that sklearn.LabelPropagation will change the original labelled data, although setting the clamping factor alpha to 1.0. Thus, while the testing accuracy can easily approaches 0.9, the prediction of testing data will all turn out to be class 0 (i.e. acc=0.1). I haven't have enough time to figure out the problem. However, I can show that my autoencoder is actually working but it's very hard to obtain a sparsely distributed representation.

After about 50 epochs, the decoded images with some degree of loss compared to the original images are shown below (first row: original; second row: decoded):



To show that the distribution of the representations are not well distributed, I used TSNE from sklearn.manifold to reduce the dimension from 16x4x4 to 2D and visualize it for the 5000 originally labelled data.

We can see that the representations are chaotically distributed, thus the labelpropagation stage will introduce a lot of incorrectly labelled data, which cause the overall performace to plunge.