HW3 Report r05921012 吳宗澤

1. Supervised learning

I am using CNN as my supervised learning method.

```
model = Sequential()
model.add(Convolution2D(32,3,3,input_shape=(3,32,32),activation='relu',dim_ordering='th',border_mode='same'))
model.add(Dropout(0.2))
model.add(Convolution2D(32, 3, 3,dim_ordering='th',activation='relu',border_mode='same'))
model.add(MaxPooling2D(pool_size=(2, 2),dim_ordering='th'))
model.add(Convolution2D(64, 3, 3,dim_ordering='th',activation='relu',border_mode='same'))
model.add(Dropout(0.2))
model.add(Convolution2D(64, 3, 3,dim_ordering='th',activation='relu',border_mode='same'))
model.add(MaxPooling2D(pool_size=(2, 2),dim_ordering='th'))
model.add(Convolution2D(128, 3, 3,dim_ordering='th',activation='relu',border_mode='same'))
model.add(Dropout(0.2))
model.add(Convolution2D(128, 3, 3,dim_ordering='th',activation='relu',border_mode='same'))
model.add(MaxPooling2D(pool_size=(2, 2),dim_ordering='th'))

model.add(Flatten())
model.add(Dropout(0.25))
model.add(Dense(1024))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(10))
model.add(Activation('softmax'))
```

There are 2 (32,3,3) Convolution2D layer,(2,2) MaxPooling ,2 (64,3,3) Convolution2D layer,(2,2) MaxPooling, 2 (128,3,3) Convolution2D layer,(2,2) MaxPooling, Flatten ,and then 3 Dense layers with 1024, 512,and10. I find a lot of information say that the filter number in the Convolution layers should be bigger and bigger, so I try this kind of architecture. In addition, the activation function I use is relu because of the reason that it can help the weights update faster in our network.

However, the performance of supervised learning isn't good enough. The accuracy in the public set is 0.6~0.63.

2. Semi-supervised learning

First, I use the same architecture like the supervised learning. After 50 epochs, I start predict 3000 unlabeled data and then add the unlabeled data into my next training data. I will repeat it with 15 times to predict all the unlabeled data.

However, it doesn't improve my performance comparing to the supervised learning. Why I am wondering about is can I really get a better performance by using a not good accuracy model to predict the unlabeled data.

The accuracy is almost the same as the accuracy in the supervised learning. In the code, although I have two output models that are model.json and model.hdf5 ,the training and testing command are still the same.

```
    print 'finish_epoch'
    unlabel_test = model.predict(all_unlabel)
    print 'finish_predict'
    for j in range (i*3000,(i+1)*3000):
        if np.amax(unlabel_test[j]) >=0.8 and check_list[j] == False:
            x_train=np.vstack((x_train,all_unlabel[j].reshape((1,3,32,32))))
            all_label_y_train=np.append(all_label_y_train,np.argmax(unlabel_test[j]))
            check_list[j] = True
        if j%500==0:
            print j
    if i ==14:
        y_train = np_utils.to_categorical(all_label_y_train, 10)
        model.fit(x_train,y_train,batch_size=100,nb_epoch=50)


model.save_weights(argv[1]+'.hdf5')
archi = model.to_json()
file = open(argv[1]+'.json','w')
file.write(archi)
```

3. Semi-supervised learning

I use a CNN autoencoder to extract out the feature from the labeled data, and the layers in the encoder part are (32,3,3) ,(16,3,3),and (8,3,3). The layers in decoder are the opposite way. Comparing the input images and the output images, they seems similar after training for 50 epochs. Moreover, I use the function K-means in the sklearn library to do the clustering, and predict the unlabeled data by it. After clustering, I use them as the pre-trained data for the CNN.

However, when the testing data are used in the encoder and throw into the CNN, I get a really bad performance at the public set that is only 0.07~0.1.

In my opinion, it is very strange to compress a 3D data to a 1D data and analyze it. Because we didn’t do any background subtraction and the color filters, it will have a lot of noise inside the data. Thus, 1D data make a really difficult situation for the CNN to learn. I think the most important thing to classify them is the texture of it, and the colors are easily to give the noise of the classifying.

Additionally, there are two models, and one is for encoder, another for CNN model. The command is still the same one.

```
x = Convolution2D(32, 3, 3, activation='relu', dim_ordering='th',border_mode='same')(input_img)
x = MaxPooling2D((2, 2), border_mode='same', dim_ordering='th')(x)
x = Convolution2D(16, 3, 3, activation='relu', dim_ordering='th',border_mode='same')(x)
x = MaxPooling2D((2, 2), border_mode='same', dim_ordering='th')(x)
x = Convolution2D(8, 3, 3, activation='relu', dim_ordering='th',border_mode='same')(x)
encoded = MaxPooling2D((2, 2), border_mode='same', dim_ordering='th')(x)

# at this point the representation is (8, 4, 4) i.e. 128-dimensional

x = Convolution2D(8, 3, 3, activation='relu', dim_ordering='th',border_mode='same')(encoded)
x = UpSampling2D((2, 2), dim_ordering='th')(x)
x = Convolution2D(16, 3, 3, activation='relu', dim_ordering='th',border_mode='same')(x)
x = UpSampling2D((2, 2), dim_ordering='th')(x)
x = Convolution2D(32, 3, 3, activation='relu', dim_ordering='th',border_mode='same')(x)
x = UpSampling2D((2, 2), dim_ordering='th')(x)

decoded = Convolution2D(3, 3, 3, activation='sigmoid', dim_ordering='th',border_mode='same')(x)
autoencoder = Model(input_img, decoded)
autoencoder.summary()
autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.fit(x_train, x_train,nb_epoch=50,batch_size=100,shuffle=True)
autoencoder.save_weights('auto.hdf5')
auto = autoencoder.to_json()
file = open('auto.json','w')
file.write(auto)
file.close()
```

```
encoder = Model(input_img,encoded)
encoder.load_weights('auto.hdf5',by_name=True)
encoder.compile(optimizer='adam', loss='mse')
en_label = encoder.predict(x_train)
encoder.save('model1.h5')
print en_label.shape
kmeans = KMeans(n_clusters=10, random_state=0).fit(en_label.reshape((5000,128)))
la_train = kmeans.labels_
en_unlabel = encoder.predict(x_unlabel)
la_unlabel = kmeans.predict(en_unlabel.reshape(45000,128))
# after cluster
en_label = np.vstack((en_label,en_unlabel))
la_train = np.hstack((la_train,la_unlabel))

#CNN
model = Sequential()
model.add(Convolution2D(32,3,3,input_shape=(8,4,4),activation='relu',dim_ordering='th',border_mode='same'))
model.add(Dropout(0.2))
model.add(Convolution2D(32, 3, 3,dim_ordering='th',activation='relu',border_mode='same'))
model.add(MaxPooling2D(pool_size=(2, 2),dim_ordering='th'))
model.add(Convolution2D(64, 3, 3,dim_ordering='th',activation='relu',border_mode='same'))
model.add(Dropout(0.2))
model.add(Convolution2D(64, 3, 3,dim_ordering='th',activation='relu',border_mode='same'))
model.add(MaxPooling2D(pool_size=(2, 2),dim_ordering='th'))

model.add(Flatten())
model.add(Dropout(0.25))
model.add(Dense(1024))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(10))
model.add(Activation('softmax'))
```


4. Compare and analyze your results

Supervised:60~63%, self-training:60%,and autoencoder being the CNN's pre-train data : 10%. Although I spend lots of time thinking how to solve the bad performance, I don't have GPU to do the debugging and testing it efficiently. However, I mentioned and analyzed the result above.