Machine Learning

Homework #3 - Semi-supervised Learning - Image Classification

B02901080 電機四 董皓文

Supervised Learning - CNN

The CNN I used is with 2 pairs of convolution layer and max pooling layer, which followed by a flatten layer and a fully connected feedforward network. And the loss function is '**categorical cross entropy**', which is used in multi class classification. I choose **AdaDelta** as my optimizer.

And then I read the input data by Pickle, to whom a **shuffle** is performed. This prevents validation split from getting all the validation data in the same categories. Since I use **dropout** as my regularization method, the loss and accuracy during the training process may fluctuate a lot. Thus, early stopping doesn't work. So I use **ModelCheckPoint** to save my best model (lowest val_acc), as shown followed.

```
checkpoint = ModelCheckpoint(weight_saved_path, monitor='val_acc',
       verbose=1, save_best_only=True, mode='max')
fitting = model.fit( x_train, y_train, batch_size=100,
       nb_epoch=200, validation_split=0.3, callbacks=[checkpoint])
```

Semi-supervised Learning I – CNN with self-training

I use **the same CNN model** to perform **self-training** as my first approach of semi-supervised learning. **Inside each iteration when performing self-training**, we first make probabilistic predictions of our unlabeled data according to our last trained model.

```
prob_predictions = model.predict_proba( x_train_unlabelled )
```

Then, we **collect part of the unlabeled data into our training data** to preform **self-training**. A preset **threshold** is used to filter good enough prediction to be collected. That is, among all unlabeled data, only those having a prediction of enough confidence is collected into the training data. The following codes conduct the abovementioned operations. Variable `collected_unlabel_index` collects the index of unlabeled data with a confidence higher than the threshold.

```
th = 0.99
collected_unlabel_argmax = np.array([])
collected_unlabel_index = np.array([], dtype=int)
prob_predictions_max = np.amax(prob_predictions, axis= 1).reshape(-1)
prob_predictions_argmax = np.argmax(prob_predictions, axis= 1).reshape(-1)
for j in range( x_train_unlabelled.shape[0] ):
    if  prob_predictions_max[j] > th:
        collected_unlabel_index = np.append(collected_unlabel_index, j)
```

Then the collected unlabeled data is added into `x_train` and the predicted labels is added into `y_labels`, which will be passed to `np_utils.to_categorical` to generate `y_train`. After added to training data, the collected unlabeled is then be removed from the unlabeled data set, which prevents an unlabeled data be chosen again and again in each iteration.

Semi-supervised Learning II – K-Means clustering initialized by autoencoder

In this section, I use **K-Means clustering** as my classifier. In order to reduce the dimension of the input image so that we can use K-Means clustering, an **autoencoder** is used to extract features of the input images. Also, to perform semi-supervised learning, I **use the centroid of the encoded labeled data to initialize of K-Means clustering**.

First, I train a **CNN autoencoder** to encode input images. Then we take all the images in labeled, unlabeled, test set as our training data to train the autoencoder. To initialize K-Means clustering by labeled data, we need to find the centroid of the encoded labeled data, which can be done by averaging values in each feature dimension. And then we perform K-Means clustering on the test data. Note that the test data should also be encoded to use K-Means clustering.

```
encoded_x_test = encoder.predict(x_test)
encoded_x_test = encoded_x_test.reshape( test_size, -1 )
print(encoded_x_test.shape)
clustering = KMeans( n_clusters=10, init=initial_centroids, n_init=1,
       max_iter=300, tol=0.0001,  precompute_distances='auto',
       verbose=0, random_state=None, copy_x=True, n_jobs=1)
predictions = clustering.fit_predict(encoded_x_test)
```

Comparison

Supervised Learning vs Semi-Supervised Learning

Self-training DO WORK! The accuracy improves from 55% of supervised CNN learning to 62% of semi-supervised CNN with self-learning. By adding most confident unlabeled data into training set can teach the model a lot more.

CNN with self-training vs K-Means clustering

K-Means clustering is basically an unsupervised learning algorithms. Although we initialize the centroids by labeled data, most information hiding in the labeled data has not been used, thus resulting in a bad performance. In the other hand, the first approach, CNN with self-training, use the unlabeled data better than K-Means clustering do. It did extract important features from the image, instead of only extracting the label (K-Means clustering do so).

Overall Comparison

	supervised CNN	CNN with self-learning	K-Means clustering initialized by labeled data
Accuracy on validation set	55%	**62%**	X
Kaggle public score	0.58	**0.60**	0.19