Machine Learning HW3 Report

R05944013 網媒一 高滿馨

Problem1. Supervised Learning

使用四層CNN的架構，將全部的label data都放進CNN裡做training，取得最後的output。試過許多參數後，最後使用adam algorithm來做optimize，並把Relu改成Leaky Relu。另外也有使用keras內建的ImageDataGenerator來增加data的數量。

```
model = initKerasModel()
X_train, Y_train = loadTrainData()

model.add(Convolution2D(32, 3, 3, border_mode='same', input_shape= ( channelNumber, height, width ) , dim_ordering = "th" ) )
model.add(LeakyReLU(alpha=.01))
model.add(Convolution2D(32, 3, 3, input_shape= ( channelNumber, height, width ) , dim_ordering = "th" ) )
model.add(LeakyReLU(alpha=.01))
model.add(MaxPooling2D(pool_size=(2, 2), dim_ordering = "th" ) )
model.add(Dropout(0.25))

model.add(Convolution2D(64, 3, 3, border_mode='same', dim_ordering = "th" ) )
model.add(LeakyReLU(alpha=.01))
model.add(Convolution2D(64, 3, 3, dim_ordering = "th" ) )
model.add(LeakyReLU(alpha=.01))
model.add(MaxPooling2D( pool_size=(2, 2), dim_ordering = "th" ) )
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(LeakyReLU(alpha=.01))
model.add(Dropout(0.5))
model.add(Dense(n_classes))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

X_train = X_train.astype('float32')
```

Supervised Learning

Performance Analyze:

一開始有使用過另外一個架構，是3~4層的Conv Layer，都使用相同的filter size

filter size	train acc	kaggle score
32	98.82	0.52080
64	99.5	0.49920

後來改用Keras 所提供的CNN架構後，有去測試不同的batch size的performance(train acc)，經過比較後，最後都使用128當batch size。

batch size	100 epoch	200 epoch	rotation 5 degree and 200 epoch
32 (default)	0.7774	0.8016	0.7978
128	0.7204	0.8784	0.8586
256		0.8398	0.7958

batch size	100 epoch	200 epoch	rotation 5 degree and 200 epoch
512		0.6952	0.6350

Problem2. Semi-Supervised learning1 - selfTraining

方法：先用label data做supervised learning 建一個model，再將unlabel data讀進來，用先前建好的model做predict，並把predict出來的結果當作這筆data的label。因為unlabel data總共有45000筆資料，所以將這筆資料分成3份，predict完15000筆資料後，就先把這些資料都加進training data後，再繼續train model，並用這個model去predict剩下的unlabel data，來得到最終的結果。

```
def selfTraining( model, mode, X_train, Y_train ):

    data = pickle.load( open( unlabelDataFileName, 'rb' ) )
    if mode == 0 :
        #retrain model after adding 15000 datas            #mode 0 -> 將所有的unlabel data 都放進train data
        for i in range( 0, n_ulabel, 15000 ):
            addData = zeros( shape = ( 15000, 3072 ) )
            #predict data
            for x in range( i, i+15000 ):
                addData[x-i] = data[x]
            addData = addData.reshape( 15000, channelNumber, height, width )
            addData /= 255
            result = model.predict_classes( addData )
            result = np_utils.to_categorical( result )

            X_train, Y_train = updateTrainData( X_train, Y_train, addData, result, 15000 )
    elif mode == 1 :
        #retrain model after adding 5000 datas            #mode 1 -> 有條件地把unlabel data 都放進train data
        threshold = 0.7
        addData = zeros( shape = ( 15000, channelNumber, height, width ) )
        addLabel = zeros( shape = ( 15000, n_classes ) )
        addDataCount = 0
        for i in range( n_ulabel ):
            tmp = zeros( shape = ( 1, 3072 ) )
            tmp[ 0, : ] = data[i]
            tmp = tmp.reshape( 1, channelNumber, height, width )
            tmp /= 255
            result = model.predict_proba( tmp )

            maxValue = 0
            maxIdx = 0
            for x in range( n_classes ):
                if result[ 0, x] > maxValue:
                    maxValue = result[ 0, x]
                    maxIdx = x

            if maxValue > threshold:    #最大機率要大於threshold才把unlabel data放進train data
                addData[ addDataCount ] = tmp
                addLabel[ addDataCount, maxIdx ] = 1
                addDataCount += 1

            if addDataCount == 15000:
                X_train, Y_train = updateTrainData( X_train, Y_train, addData, result, 15000 )
                addDataCount = 0
                addData = zeros( shape = ( 15000, channelNumber, height, width ) )
                addLabel = zeros( shape = ( 15000, n_classes ) )

        subAddData = zeros( shape = ( addDataCount, channelNumber, height, width ) )
        subAddData = addData[ : addDataCount ]
        X_train, Y_train = updateTrainData( X_train, Y_train, subAddData, result, addDataCount )

    return model
```

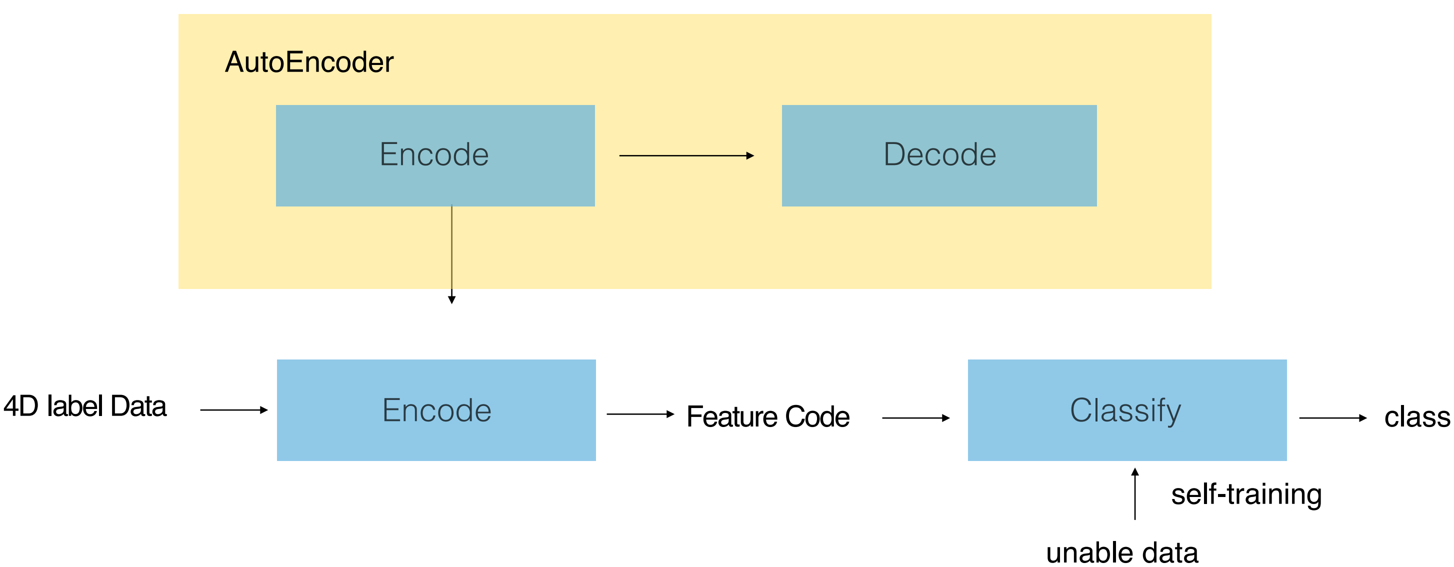
```python
def updateTrainData( X_train, Y_train, addData, result, n ):

    #update train data
    oriTrainData = (X_train.shape)[0]
    newX_train = zeros( shape = ( oriTrainData + n , channelNumber, height, width ) )
    newY_train = zeros( shape = ( oriTrainData + n, n_classes ) )
    newX_train[ : oriTrainData, : , : , : ] = X_train
    newX_train[ oriTrainData : oriTrainData + n, : , : , : ] = addData
    newY_train[ : oriTrainData, : ] = Y_train
    newY_train[ oriTrainData : oriTrainData + n, :  ] = result

    X_train = zeros( shape = ( (newX_train.shape)[0], channelNumber, height, width ) )
    Y_train = zeros( shape = ( (newX_train.shape)[0], n_classes ) )

    #shuffle data
    index_shuf = range( (newX_train.shape)[0] )
    random.shuffle( index_shuf )
    for i in range( 0, len( index_shuf )):
        X_train[i] = newX_train[index_shuf[i]]
        Y_train[i] = newY_train[index_shuf[i]]

    model.fit( X_train, Y_train, nb_epoch= 50, batch_size = 32, validation_split = 0.1, shuffle = True )
                                        #用新的train data( 原本的data + 加上label後的unlabel data) 再train一個model
    return X_train, Y_train
```

Performance

把unlabel data全部都加進去的效果會比supervisied的效果好，kaggle score會高大約0.02(Kaggle score: 0.60 -> 0.62)，但有條件地把unlabel data加進去，效果則會變得很差。

Problem3. Semi-Supervised learning2 - autoEncoder + selfTraining

方法：先建立一個autoencoder的model(encode: 兩層conv, decode: 兩層conv)。並把encoded的model取出來，把label data當作input，取出1024維的feature code。再用CNN建另外一個classify的model，把feature code當做input，原本的label當做output，來學習feature會對應到哪個class。再把unlabel data都變成feature code，用self-training的方法，來train這個classify的model。



```python
def constructAutoEncoderModel():

    input_img = Input( shape = ( channelNumber, height, width ) )

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

    ## dimension( 16 * 8 * 8 )

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

    decoded = Convolution2D(3, 3, 3, activation='sigmoid', border_mode='same', dim_ordering="th" )(x)

    autoencoder = Model( input_img, decoded )
    autoencoder.compile( optimizer='adadelta', loss = 'binary_crossentropy', metrics = ['accuracy'] )

    return input_img, encoded, decoded, autoencoder
```

autoencoder

```python
def codeClassify( code, Y_train ):

    print( code.shape )
    #ode = code.reshape( n_train, 1024 ) # origin code shpae = ( 5000, 16, 8, 8 )

    classModel = Sequential()

    classModel.add(Convolution2D(32, 3, 3, border_mode='same', input_shape= ( 16, 8, 8 ) , dim_ordering = "th" ) )
    classModel.add(Activation('relu'))
    classModel.add(Convolution2D(32, 3, 3, dim_ordering = "th" ) )
    classModel.add(Activation('relu'))
    classModel.add(MaxPooling2D(pool_size=(2, 2), dim_ordering = "th" ) )
    classModel.add(Dropout(0.25))
    classModel.add(Flatten())

    classModel.add(Dense( 512, activation='relu'))
    classModel.add(Dropout(0.5))
    classModel.add(Dense( n_classes, activation='softmax') )

    # Compile model
    classModel.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    classModel.fit( code, Y_train, nb_epoch = 100, batch_size = 128, shuffle = True )

    return classModel, code
```

codeClassify model

```python
def codeClassifySelfTraining( classModel, encodedModel, X_train, Y_train ):

    data = pickle.load( open( unlabelDataFileName, 'rb' ) )

    for i in range( 0, n_ulabel, 15000 ):
        addData = zeros( shape = ( 15000, 3072 ) )

        #predict data
        curr = i
        for x in range( curr, curr+15000 ):
            addData[x-curr] = data[x]

        addData = addData.reshape( 15000, channelNumber, height, width )
        addData /= 255

        encodedAddData = encodedModel.predict( addData )

        predictResult = classModel.predict_classes( encodedAddData )
        predictResult = np_utils.to_categorical( predictResult, nb_classes = n_classes )

        X_train, Y_train = updateTrainData( X_train, Y_train, encodedAddData, predictResult, 15000, classModel )

    return classModel
```

codeClassifymodel Self-Training

*最後會輸出兩個model(encoded model及classify model)

Performance
auto encoder train loss: 0.56
classModel train acc: 0.82
Kaggle Score: 0.53

Problem4. Compare and Analyze results

目前三個方法試下來，是self-training的semi-supervised效果最好(方法一)。不過發現self-training的semi-supervised要好的前提是，model0的準確度要夠好，否則semi-supervised只會越train越差。另外，在增加unlabel data的部分，有試過兩種方法，一種是全加，另一種是去看predict結果的機率分佈，假如機率最高的那個class機率沒有大於一個threshold，就判定為不準確，不會加進training data，不過可能因為原本的model準確度沒有很高，因此如果還設一個threshold的話，整個model就會overfitting得很嚴重，最後testing的效果會很差。

另外，在方法二的部分，可能因為一開始auto-encoder沒有建得很好，所以導致最後的performance比不上方法一。在這邊原本想要用KMeans去做clustering，不過因為一開始model沒建好，沒有把不同的class分得很開，所以導致最後clustering的結果不好，因此最後改用self-training去學feature code。

另外，發現好像不是所有目前state-of-art的架構都適合這樣的資料，supervisied的部分，有試過VGG16，不過可能參數太多了，然後input image的維度太小，所以train不起來，acc只會在0.09和0.1之間徘徊。