openmic-VGG-ML(modeling-baseline)

March 2, 2020

1 OpenMIC-2018 baseline model tutorial

This notebook demonstrates how to replicate a simplified version of the baseline modeling experiment in (Humphrey, Durand, and McFee, 2018).

First, make sure you download the dataset!

We'll load in the pre-computed VGGish features and labels, and fit a RandomForest model for each of the 20 instrument classes using the pre-defined train-test splits provided in the repository.

We'll then evaluate the models we fit, and show how to apply them to new audio signals.

This notebook is not meant to demonstrate state-of-the-art performance on instrument recognition. Rather, we hope that it can serve as a starting point for building your own instrument detectors without too much effort!

[1]:
```
# These dependencies are necessary for loading the data
import json
import os
import numpy as np
import pandas as pd

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Be sure to set this after downloading the dataset!
DATA_ROOT = 'openmic-2018/'

if not os.path.exists(DATA_ROOT):
    raise ValueError('Did you forget to set `DATA_ROOT`?')
```

1.1 Loading the data

The openmic data is provided in a python-friendly format as `openmic-2018.npz`.

You can load it as follows:

[2]:
```
OPENMIC = np.load(os.path.join(DATA_ROOT, 'openmic-2018.npz'),allow_pickle=True)
```

[3]:
```
# What's included?
print(list(OPENMIC.keys()))
```

```
['X', 'Y_true', 'Y_mask', 'sample_key']
```

1.1.1 What's included in the data?

- `X`: 20000 * 10 * 128 array of VGGish features
  - First index (0..19999) corresponds to the sample key
  - Second index (0..9) corresponds to the time within the clip (each time slice is 960 ms long)
  - Third index (0..127) corresponds to the VGGish features at each point in the 10sec clip
  - Example `X[40, 8]` is the 128-dimensional feature vector for the 9th time slice in the 41st example
- `Y_true`: 20000 * 20 array of *true* label probabilities
  - First index corresponds to sample key, as above
  - Second index corresponds to the label class (accordion, . . . , voice)
  - Example: `Y[40, 4]` indicates the confidence that example #41 contains the 5th instrument
- `Y_mask`: 20000 * 20 binary mask values
  - First index corresponds to sample key
  - Second index corresponds to the label class
  - Example: `Y[40, 4]` indicates whether or not we have observations for the 5th instrument for example #41
- `sample_key`: 20000 array of sample key strings
  - Example: `sample_key[40]` is the sample key for example #41

[4]:
```
# It will be easier to use if we make direct variable names for everything
X, Y_true, Y_mask, sample_key = OPENMIC['X'], OPENMIC['Y_true'],␣
↪OPENMIC['Y_mask'], OPENMIC['sample_key']
```

[5]:
```
X.shape
```

[5]:
```
(20000, 10, 128)
```

[6]:
```
# Features for the 9th time slice of 81st example
X[80, 8]
```

[6]:
```
array([192,  30, 176, 126, 208,  85,  84,  95,  69, 234,  99, 118, 166,
       150, 106,  68, 165, 156, 146, 206,  75, 210, 131,  49,  61, 218,
        92, 152, 121, 167,  62, 166, 167, 237,  22, 168, 165, 137, 178,
       132, 196,  96,  54, 166, 169, 132,  59,  27,  46, 123,  89,  47,
        58, 116,  48, 188, 157,  28,  44, 252, 248, 100,  28, 154, 147,
       148, 204, 104,  95,  67, 109, 147, 204, 146, 196, 222,  90, 255,
        94, 171,  53, 133, 202, 152,  35,  55, 231, 255,  62, 227, 168,
       192,  87, 144, 130, 255,   0,   0, 163,  75, 255, 135, 216,  68,
         0, 199,   0, 193, 254, 114,  12, 255,   0,  74, 165,   0, 201,
       246,   0, 127, 211, 218, 164,  57, 238, 176, 158, 255], dtype=int64)
```

[7]:
```
Y_true[40]
```

[7]:
```
array([0.5    , 0.5    , 0.5    , 0.5    , 0.5    , 0.15055, 0.5    ,
       0.5    , 0.5    , 0.5    , 0.5    , 0.5    , 0.5    , 0.5    ,
       0.5    , 0.5    , 0.5    , 0.5    , 0.5    , 0.5    ])
```

[8]:
```
Y_mask[40]
```

[8]:
```
array([False, False, False, False, False,  True, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False])
```

[9]:
```
sample_key.shape
```

[9]:
```
(20000,)
```

[10]:
```
sample_key[40]
```

[10]:
```
'000385_249600'
```

1.1.2 Load the class map

For convenience, we provide a simple JSON object that maps class indices to names.

[11]:
```
with open(os.path.join(DATA_ROOT, 'class-map.json'), 'r') as f:
    class_map = json.load(f)
```

[12]:
```
class_map
```

[12]:
```
{'accordion': 0,
 'banjo': 1,
 'bass': 2,
 'cello': 3,
 'clarinet': 4,
 'cymbals': 5,
 'drums': 6,
 'flute': 7,
 'guitar': 8,
 'mallet_percussion': 9,
 'mandolin': 10,
 'organ': 11,
 'piano': 12,
 'saxophone': 13,
 'synthesizer': 14,
 'trombone': 15,
 'trumpet': 16,
 'ukulele': 17,
 'violin': 18,
 'voice': 19}
```

1.2 Loading the train-test splits

OpenMIC-2018 comes with a pre-defined train-test split. Great care was taken to ensure that this split is approximately balanced and artists are not represented in both sides of the split, so please use it!

This is done by sample key, not row number, so you will need to go through the `sample_key` array to slice the data.

[13]:
```python
# Let's split the data into the training and test set
# We use squeeze=True here to return a single array for each, rather than a␣
↪full DataFrame

split_train = pd.read_csv(os.path.join(DATA_ROOT, 'partitions/split01_train.
↪csv'),
                          header=None, squeeze=True)
split_test = pd.read_csv(os.path.join(DATA_ROOT, 'partitions/split01_test.csv'),
                         header=None, squeeze=True)
```

[14]:
```python
# These two tables contain the sample keys for training and testing examples
# Let's see the keys for the first five training example
split_train.head(5)
```

[14]:
```
0      000046_3840
1    000135_483840
2    000139_119040
3    000141_153600
4     000144_30720
Name: 0, dtype: object
```

[15]:
```python
# How many train and test examples do we have?  About 75%/25%
print('# Train: {},  # Test: {}'.format(len(split_train), len(split_test)))
```

```
# Train: 14915,  # Test: 5085
```

These sample key maps are easier to use as sets, so let's make them sets!

[16]:
```python
train_set = set(split_train)
test_set = set(split_test)
```

1.2.1 Split the data

Now that we have the sample keys for the training and testing examples, we need to partition the data arrays (`X`, `Y_true`, `Y_mask`).

This is a little delicate to get right.

[17]:
```python
# These loops go through all sample keys, and save their row numbers
# to either idx_train or idx_test
#
# This will be useful in the next step for slicing the array data
idx_train, idx_test = [], []
```

```python
for idx, n in enumerate(sample_key):
    if n in train_set:
        idx_train.append(idx)
    elif n in test_set:
        idx_test.append(idx)
    else:
        # This should never happen, but better safe than sorry.
        raise RuntimeError('Unknown sample key={}! Abort!'.
 ↪format(sample_key[n]))

# Finally, cast the idx_* arrays to numpy structures
idx_train = np.asarray(idx_train)
idx_test = np.asarray(idx_test)
```

[18]:
```python
# Finally, we use the split indices to partition the features, labels, and␣
 ↪masks
X_train = X[idx_train]
X_test = X[idx_test]

Y_true_train = Y_true[idx_train]
Y_true_test = Y_true[idx_test]

Y_mask_train = Y_mask[idx_train]
Y_mask_test = Y_mask[idx_test]
```

[19]:
```python
# Print out the sliced shapes as a sanity check
print(X_train.shape)
print(X_test.shape)
```

```
(14915, 10, 128)
(5085, 10, 128)
```

2 Fit the models

Now, we'll iterate over all the instrument classes, and fit a separate `RandomForest` model for each one.

For each instrument, the steps are as follows:

1. Find the subset of training (and testing) data that have been annotated for the current instrument
2. Simplify the features to have one observation point per clip, instead of one point per time slice within each clip
3. Initialize a classifier
4. Fit the classifier to the training data
5. Evaluate the classifier on the test data and print a report

[20]:
```python
# This dictionary will include the classifiers for each model
models = dict()

# We'll iterate over all istrument classes, and fit a model for each one
# After training, we'll print a classification report for each instrument
for instrument in class_map:

    # Map the instrument name to its column number
    inst_num = class_map[instrument]

    # Step 1: sub-sample the data

    # First, we need to select down to the data for which we have annotations
    # This is what the mask arrays are for
    train_inst = Y_mask_train[:, inst_num]
    test_inst = Y_mask_test[:, inst_num]

    # Here, we're using the Y_mask_train array to slice out only the training␣
 ↪examples
    # for which we have annotations for the given class
    X_train_inst = X_train[train_inst]

    # Step 3: simplify the data by averaging over time

    # Let's arrange the data for a sklearn Random Forest model
    # Instead of having time-varying features, we'll summarize each track by␣
 ↪its mean feature vector over time
    X_train_inst_sklearn = np.mean(X_train_inst, axis=1)

    # Again, we slice the labels to the annotated examples
    # We thresold the label likelihoods at 0.5 to get binary labels
    Y_true_train_inst = Y_true_train[train_inst, inst_num] >= 0.5


    # Repeat the above slicing and dicing but for the test set
    X_test_inst = X_test[test_inst]
    X_test_inst_sklearn = np.mean(X_test_inst, axis=1)
    Y_true_test_inst = Y_true_test[test_inst, inst_num] >= 0.5

    # Step 3.
    # Initialize a new classifier
    clf = RandomForestClassifier(max_depth=8, n_estimators=100, random_state=0)

    # Step 4.
    clf.fit(X_train_inst_sklearn, Y_true_train_inst)

    # Step 5.
```

```python
    # Finally, we'll evaluate the model on both train and test
    Y_pred_train = clf.predict(X_train_inst_sklearn)
    Y_pred_test = clf.predict(X_test_inst_sklearn)

    print('-' * 52)
    print(instrument)
    print('\tTRAIN')
    print(classification_report(Y_true_train_inst, Y_pred_train))
    print(Y_true_train_inst[3])
    print(Y_pred_train[3])
    print('\tTEST')
    print(classification_report(Y_true_test_inst, Y_pred_test))

    print(Y_true_test_inst.shape)
    print(Y_pred_test.shape)

    # Store the classifier in our dictionary
    models[instrument] = clf
```

```
----------------------------------------------------
accordion
        TRAIN
              precision    recall  f1-score   support

       False       0.96      1.00      0.98      1159
        True       1.00      0.88      0.94       374

    accuracy                           0.97      1533
   macro avg       0.98      0.94      0.96      1533
weighted avg       0.97      0.97      0.97      1533

True
False
        TEST
              precision    recall  f1-score   support

       False       0.84      0.97      0.90       423
        True       0.77      0.32      0.45       115

    accuracy                           0.83       538
   macro avg       0.81      0.65      0.68       538
weighted avg       0.83      0.83      0.81       538

(538,)
(538,)
----------------------------------------------------
banjo
```

```
        TRAIN
              precision    recall  f1-score   support

       False       0.98      0.98      0.98      1148
        True       0.97      0.97      0.97       592

    accuracy                           0.98      1740
   macro avg       0.98      0.97      0.98      1740
weighted avg       0.98      0.98      0.98      1740

False
False
        TEST
              precision    recall  f1-score   support

       False       0.82      0.90      0.86       338
        True       0.68      0.52      0.59       140

    accuracy                           0.79       478
   macro avg       0.75      0.71      0.72       478
weighted avg       0.78      0.79      0.78       478

(478,)
(478,)
----------------------------------------------------
bass
        TRAIN
              precision    recall  f1-score   support

       False       0.97      0.99      0.98      1010
        True       0.96      0.93      0.95       415

    accuracy                           0.97      1425
   macro avg       0.97      0.96      0.96      1425
weighted avg       0.97      0.97      0.97      1425

False
False
        TEST
              precision    recall  f1-score   support

       False       0.82      0.96      0.89       329
        True       0.83      0.49      0.61       134

    accuracy                           0.82       463
   macro avg       0.83      0.72      0.75       463
weighted avg       0.82      0.82      0.81       463
```

```
(463,)
(463,)
----------------------------------------------------
cello
        TRAIN
              precision    recall  f1-score   support

       False       0.99      0.96      0.97       866
        True       0.95      0.98      0.96       598

    accuracy                           0.97      1464
   macro avg       0.97      0.97      0.97      1464
weighted avg       0.97      0.97      0.97      1464

False
False
        TEST
              precision    recall  f1-score   support

       False       0.80      0.83      0.81       259
        True       0.79      0.76      0.78       226

    accuracy                           0.80       485
   macro avg       0.80      0.79      0.79       485
weighted avg       0.80      0.80      0.80       485

(485,)
(485,)
----------------------------------------------------
clarinet
        TRAIN
              precision    recall  f1-score   support

       False       0.92      1.00      0.96      1349
        True       1.00      0.70      0.82       396

    accuracy                           0.93      1745
   macro avg       0.96      0.85      0.89      1745
weighted avg       0.94      0.93      0.93      1745

False
False
        TEST
              precision    recall  f1-score   support

       False       0.80      0.99      0.88       503
        True       0.71      0.09      0.16       137
```

```
    accuracy                           0.80       640
   macro avg       0.75      0.54      0.52       640
weighted avg       0.78      0.80      0.73       640

(640,)
(640,)
----------------------------------------------------
cymbals
        TRAIN
              precision    recall  f1-score   support

       False       1.00      0.90      0.95       485
        True       0.94      1.00      0.97       814

    accuracy                           0.96      1299
   macro avg       0.97      0.95      0.96      1299
weighted avg       0.97      0.96      0.96      1299

True
True
        TEST
              precision    recall  f1-score   support

       False       0.95      0.85      0.90       139
        True       0.93      0.98      0.96       297

    accuracy                           0.94       436
   macro avg       0.94      0.91      0.93       436
weighted avg       0.94      0.94      0.94       436

(436,)
(436,)
----------------------------------------------------
drums
        TRAIN
              precision    recall  f1-score   support

       False       1.00      0.95      0.98       495
        True       0.97      1.00      0.99       828

    accuracy                           0.98      1323
   macro avg       0.99      0.98      0.98      1323
weighted avg       0.98      0.98      0.98      1323

True
True
        TEST
              precision    recall  f1-score   support
```

```
       False       0.93      0.79      0.86       146
        True       0.90      0.97      0.93       278

    accuracy                           0.91       424
   macro avg       0.91      0.88      0.89       424
weighted avg       0.91      0.91      0.91       424

(424,)
(424,)
----------------------------------------------------
flute
        TRAIN
              precision    recall  f1-score   support

       False       0.97      0.99      0.98      1050
        True       0.98      0.94      0.96       472

    accuracy                           0.98      1522
   macro avg       0.98      0.97      0.97      1522
weighted avg       0.98      0.98      0.98      1522

False
False
        TEST
              precision    recall  f1-score   support

       False       0.76      0.91      0.83       387
        True       0.65      0.37      0.47       175

    accuracy                           0.74       562
   macro avg       0.71      0.64      0.65       562
weighted avg       0.73      0.74      0.72       562

(562,)
(562,)
----------------------------------------------------
guitar
        TRAIN
              precision    recall  f1-score   support

       False       1.00      0.95      0.98       362
        True       0.98      1.00      0.99       852

    accuracy                           0.99      1214
   macro avg       0.99      0.98      0.98      1214
weighted avg       0.99      0.99      0.99      1214
```

```
True
True
        TEST
              precision    recall  f1-score   support

       False       0.97      0.97      0.97       150
        True       0.98      0.98      0.98       286

    accuracy                           0.98       436
   macro avg       0.97      0.97      0.97       436
weighted avg       0.98      0.98      0.98       436

(436,)
(436,)
----------------------------------------------------
mallet_percussion
        TRAIN
              precision    recall  f1-score   support

       False       1.00      0.95      0.97       802
        True       0.93      1.00      0.96       522

    accuracy                           0.97      1324
   macro avg       0.96      0.97      0.97      1324
weighted avg       0.97      0.97      0.97      1324

True
True
        TEST
              precision    recall  f1-score   support

       False       0.77      0.84      0.81       267
        True       0.78      0.69      0.73       211

    accuracy                           0.77       478
   macro avg       0.77      0.76      0.77       478
weighted avg       0.77      0.77      0.77       478

(478,)
(478,)
----------------------------------------------------
mandolin
        TRAIN
              precision    recall  f1-score   support

       False       0.97      0.96      0.97      1185
        True       0.93      0.95      0.94       652
```

```
    accuracy                           0.96      1837
   macro avg       0.95      0.96      0.95      1837
weighted avg       0.96      0.96      0.96      1837

False
False
        TEST
              precision    recall  f1-score   support

       False       0.81      0.83      0.82       434
        True       0.59      0.57      0.58       193

    accuracy                           0.75       627
   macro avg       0.70      0.70      0.70       627
weighted avg       0.75      0.75      0.75       627

(627,)
(627,)
----------------------------------------------------
organ
        TRAIN
              precision    recall  f1-score   support

       False       0.97      1.00      0.98       977
        True       1.00      0.93      0.96       482

    accuracy                           0.98      1459
   macro avg       0.98      0.96      0.97      1459
weighted avg       0.98      0.98      0.98      1459

False
False
        TEST
              precision    recall  f1-score   support

       False       0.76      0.95      0.85       310
        True       0.67      0.25      0.36       121

    accuracy                           0.75       431
   macro avg       0.72      0.60      0.60       431
weighted avg       0.74      0.75      0.71       431

(431,)
(431,)
----------------------------------------------------
piano
        TRAIN
              precision    recall  f1-score   support
```

```
       False       1.00      0.96      0.98       420
        True       0.98      1.00      0.99       885

    accuracy                           0.99      1305
   macro avg       0.99      0.98      0.99      1305
weighted avg       0.99      0.99      0.99      1305

False
False
        TEST
              precision    recall  f1-score   support

       False       0.96      0.85      0.90       130
        True       0.93      0.98      0.96       285

    accuracy                           0.94       415
   macro avg       0.94      0.91      0.93       415
weighted avg       0.94      0.94      0.94       415

(415,)
(415,)
----------------------------------------------------
saxophone
        TRAIN
              precision    recall  f1-score   support

       False       0.99      0.94      0.97       906
        True       0.94      0.99      0.96       830

    accuracy                           0.96      1736
   macro avg       0.97      0.97      0.96      1736
weighted avg       0.97      0.96      0.96      1736

True
True
        TEST
              precision    recall  f1-score   support

       False       0.85      0.80      0.83       324
        True       0.80      0.86      0.83       305

    accuracy                           0.83       629
   macro avg       0.83      0.83      0.83       629
weighted avg       0.83      0.83      0.83       629

(629,)
(629,)
```

```
----------------------------------------------------
synthesizer
        TRAIN
              precision    recall  f1-score   support

       False       0.99      0.95      0.97       399
        True       0.98      1.00      0.99       823

    accuracy                           0.98      1222
   macro avg       0.99      0.98      0.98      1222
weighted avg       0.98      0.98      0.98      1222

True
True
        TEST
              precision    recall  f1-score   support

       False       0.94      0.90      0.92       112
        True       0.96      0.97      0.97       268

    accuracy                           0.95       380
   macro avg       0.95      0.94      0.94       380
weighted avg       0.95      0.95      0.95       380

(380,)
(380,)
----------------------------------------------------
trombone
        TRAIN
              precision    recall  f1-score   support

       False       0.95      0.98      0.97      1405
        True       0.95      0.89      0.92       635

    accuracy                           0.95      2040
   macro avg       0.95      0.94      0.94      2040
weighted avg       0.95      0.95      0.95      2040

False
False
        TEST
              precision    recall  f1-score   support

       False       0.81      0.92      0.87       492
        True       0.77      0.54      0.63       228

    accuracy                           0.80       720
   macro avg       0.79      0.73      0.75       720
```

```
weighted avg       0.80      0.80      0.79       720

(720,)
(720,)
----------------------------------------------------
trumpet
        TRAIN
              precision    recall  f1-score   support

       False       0.97      0.97      0.97      1303
        True       0.96      0.95      0.95       828

    accuracy                           0.96      2131
   macro avg       0.96      0.96      0.96      2131
weighted avg       0.96      0.96      0.96      2131

True
True
        TEST
              precision    recall  f1-score   support

       False       0.77      0.88      0.82       467
        True       0.78      0.62      0.69       318

    accuracy                           0.78       785
   macro avg       0.78      0.75      0.76       785
weighted avg       0.78      0.78      0.77       785

(785,)
(785,)
----------------------------------------------------
ukulele
        TRAIN
              precision    recall  f1-score   support

       False       0.97      0.98      0.98      1279
        True       0.96      0.93      0.94       556

    accuracy                           0.97      1835
   macro avg       0.96      0.95      0.96      1835
weighted avg       0.97      0.97      0.96      1835

True
False
        TEST
              precision    recall  f1-score   support

       False       0.81      0.88      0.84       408
```

```
        True       0.67      0.54      0.60       182

    accuracy                           0.78       590
   macro avg       0.74      0.71      0.72       590
weighted avg       0.77      0.78      0.77       590

(590,)
(590,)
----------------------------------------------------
violin
        TRAIN
              precision    recall  f1-score   support

       False       1.00      0.88      0.94       623
        True       0.91      1.00      0.95       779

    accuracy                           0.95      1402
   macro avg       0.96      0.94      0.94      1402
weighted avg       0.95      0.95      0.95      1402

False
False
        TEST
              precision    recall  f1-score   support

       False       0.87      0.70      0.78       237
        True       0.84      0.94      0.88       394

    accuracy                           0.85       631
   macro avg       0.85      0.82      0.83       631
weighted avg       0.85      0.85      0.84       631

(631,)
(631,)
----------------------------------------------------
voice
        TRAIN
              precision    recall  f1-score   support

       False       1.00      0.91      0.95       426
        True       0.95      1.00      0.98       764

    accuracy                           0.97      1190
   macro avg       0.97      0.95      0.96      1190
weighted avg       0.97      0.97      0.97      1190

True
True
```

```
        TEST
              precision    recall  f1-score   support

       False       0.94      0.89      0.91       150
        True       0.93      0.96      0.94       224

    accuracy                           0.93       374
   macro avg       0.93      0.92      0.93       374
weighted avg       0.93      0.93      0.93       374

(374,)
(374,)
```

3 Applying the model to new data

In this section, we'll take the models trained above and apply them to audio signals, stored as OGG Vorbis files.

[ ]:
```python
# We need soundfile to load audio data
import soundfile as sf

# And the openmic-vggish preprocessor
import openmic.vggish

# For audio playback
from IPython.display import Audio
```

```
      ␣
↪---------------------------------------------------------------------------

       ModuleNotFoundError                       Traceback (most recent call␣
↪last)

       <ipython-input-21-51bf03974dcb> in <module>
         1 # We need soundfile to load audio data
   ----> 2 import soundfile as sf
         3
         4 # And the openmic-vggish preprocessor
         5 import openmic.vggish


       ModuleNotFoundError: No module named 'soundfile'
```

```
[ ]: # We include a test ogg file in the openmic repository, which we can use here.
     audio, rate = sf.read(os.path.join(DATA_ROOT, 'audio/000/000046_3840.ogg'))

     time_points, features = openmic.vggish.waveform_to_features(audio, rate)
```

```
[ ]: # The time_points array marks the starting time of each observation
     time_points
```

```
[ ]: # The features array includes the vggish feature observations
     features.shape
```

```
[ ]: # Let's listen to the example
     Audio(data=audio.T, rate=rate)
```

```
[ ]: # finally, apply the classifier

     # Average over time to one observation, but keep the number of dimensions the␣
      ↪same
     # The test clip is 10sec long, so this is the same process as in the training␣
      ↪step
     # However, you could also apply the classifier to each frame independently to␣
      ↪get time-varying predictions
     feature_mean = np.mean(features, axis=0, keepdims=True)

     for instrument in models:

         clf = models[instrument]

         print('P[{:18s}=1] = {:.3f}'.format(instrument, clf.
      ↪predict_proba(feature_mean)[0,1]))
```

4 Wrapping up

So the predictions here are definitely not perfect, but they're a good start!

Some things you might want to try out:

1. Instead of averaging features over time, apply the classifiers to each time-step to get a time-varying instrument detector.
2. Play with the parameters of the `RandomForest` model, changing the depth and number of estimators.
3. Run the trained model on your own favorite songs!
4. Train a different model, maybe using different features!
5. Make use of label uncertainties or unlabeled data when training!