```
1 run.sh
2 utils.py
3 pretrain.py
4 train.py
5 predict.py
6 ensemble.py
----------
1 run.sh


#!/usr/bin/env bash


python utils.py
python pretrain.py
python train.py
python predict.py
python ensemble.py
----------
2 utils.py

import numpy as np
from tqdm import tqdm
import pandas as pd
from keras.utils.data_utils import Sequence
import librosa
from keras.preprocessing.sequence import pad_sequences
from config import *
import multiprocessing as mp
import pickle
from models import cnn_model
from sklearn.preprocessing import StandardScaler
from collections import defaultdict, Counter
import scipy

class FreeSound(Sequence):
    def __init__(self,X,Gfeat,Y,cfg,mode,epoch):

        self.X, self.Gfeat, self.Y, self.cfg =
X,Gfeat,Y,cfg
        self.bs = cfg.bs
        self.mode = mode
        self.ids = list(range(len(self.X)))
        self.epoch = epoch

        self.aug = None
        if mode == 'train':
            self.get_offset = np.random.randint
            np.random.shuffle(self.ids)

        elif mode == 'pred1':
            self.get_offset = lambda x: 0
        elif mode == 'pred2':
            self.get_offset = lambda x: int(x/2)
        elif mode == 'pred3':
            self.get_offset = lambda x: x
        else:
            raise RuntimeError("error")


    def __len__(self):
        return (len(self.X)+self.bs-1) // self.bs


    def __getitem__(self,idx):

        batch_idx = self.ids[idx*self.bs:(idx+1)*self.bs]
        batch_x = {
            'audio':[],
            'other':[],
            'global_feat':self.Gfeat[batch_idx],
        }
        for i in batch_idx:
            audio_sample = self.X[i]

            feature = [audio_sample.shape[0] / 441000]
            batch_x['other'].append(feature)

            max_offset = audio_sample.shape[0] -
self.cfg.maxlen
            data = self.get_sample(audio_sample,
max_offset)

            batch_x['audio'].append(data)

        batch_y = np.array(self.Y[batch_idx])
        batch_x = {k: np.array(v) for k, v in
batch_x.items()}

        if self.mode == 'train':
            batch_y = self.cfg.lm * (1-batch_y) + (1 -
self.cfg.lm) * batch_y
```

```
        if self.mode == 'train' and np.random.rand() <
self.cfg.mixup_prob and self.epoch <
self.cfg.milestones[0]:
            batch_idx =
np.random.permutation(list(range(len(batch_idx))))
            rate = self.cfg.x1_rate

            batch_x['audio'] = rate * batch_x['audio'] +
(1-rate) * batch_x['audio'][batch_idx]
            batch_y = rate * batch_y + (1-rate) *
batch_y[batch_idx]


        batch_x['y'] = batch_y
        return batch_x, None

    def augment(self,data):
        # if self.mode == 'train' and self.epoch <
self.cfg.milestones[0] and np.random.rand() < 0.5:
        #     mask_len = int(data.shape[0] * 0.02)
        #     s = np.random.randint(0,data.shape[0]-
mask_len)
        #     data[s:s+mask_len] = 0
        return data

    def get_sample(self,data,max_offset):
        if max_offset > 0:
            offset = self.get_offset(max_offset)
            data = data[offset:(self.cfg.maxlen +
offset)]
            if self.mode == 'train':
                data = self.augment(data)

        elif max_offset < 0:
            max_offset = -max_offset
            offset = self.get_offset(max_offset)
            if self.mode == 'train':
                data = self.augment(data)
            if len(data.shape) == 1:
                data = np.pad(data, ((offset, max_offset
- offset)), "constant")
            else:
                data = np.pad(data, ((offset, max_offset
- offset),(0,0),(0,0)), "constant")
        return data
    def on_epoch_end(self):
        if self.mode == 'train':
            np.random.shuffle(self.ids)


def get_global_feat(x,num_steps):
    stride = len(x)/num_steps
    ts = []
    for s in range(num_steps):
        i = s * stride
        wl = max(0,int(i - stride/2))
        wr = int(i + 1.5*stride)
        local_x = x[wl:wr]
        percent_feat = np.percentile(local_x, [0, 1, 25,
30, 50, 60, 75, 99, 100]).tolist()
        range_feat = local_x.max()-local_x.min()

ts.append([np.mean(local_x),np.std(local_x),range_feat]
+percent_feat)
    ts = np.array(ts)
    assert ts.shape == (128,12),(len(x),ts.shape)
    return ts


def worker_cgf(file_path):
    result = []
    for path in tqdm(file_path):
        data, _ = librosa.load(path, 44100)
        result.append(get_global_feat(data,
num_steps=128))
    return result


def create_global_feat():

    df = pd.concat([pd.read_csv(f'../input/
train_curated.csv'),pd.read_csv('../input/
train_noisy.csv',usecols=['fname','labels'])])
    df = df.reset_index(drop=True)
    file_path = train_dir + df['fname']

    workers = mp.cpu_count() // 2
    pool = mp.Pool(workers)
    results = []
    ave_task = (len(file_path) + workers - 1) // workers
    for i in range(workers):
        res = pool.apply_async(worker_cgf,
                               args=(file_path[i *
ave_task:(i + 1) * ave_task],))
        results.append(res)
    pool.close()
    pool.join()

    results = np.concatenate([res.get() for res in
results],axis=0)
    print(results.shape)
    np.save('../input/gfeat', np.array(results))

    df = pd.read_csv(f'../input/sample_pred.csv')

    file_path = train_dir + df['fname']

    workers = mp.cpu_count() // 2
    pool = mp.Pool(workers)
    results = []
    ave_task = (len(file_path) + workers - 1) // workers
    for i in range(workers):
        res = pool.apply_async(worker_cgf,
                               args=(file_path[i *
ave_task:(i + 1) * ave_task],))
        results.append(res)
    pool.close()
    pool.join()

    results = np.concatenate([res.get() for res in
results], axis=0)
    print(results.shape)
    np.save('../input/te_gfeat', np.array(results))

def split_and_label(rows_labels):
    row_labels_list = []
    for row in rows_labels:
        row_labels = row.split(',')
        labels_array = np.zeros((n_classes))
        for label in row_labels:
            index = label2i[label]
            labels_array[index] = 1
        row_labels_list.append(labels_array)
    return np.array(row_labels_list)
```

```
if __name__ == '__main__':

    create_global_feat()
----------

3 pretrain.py


from tqdm import tqdm
from sklearn.metrics import
label_ranking_average_precision_score
from utils import *
from config import *


def main(cfg,get_model):

    if True: # load data
        df = pd.read_csv(f'../input/train_noisy.csv')
        y = split_and_label(df['labels'].values)
        x = train_dir + df['fname'].values
        x = [librosa.load(path, 44100)[0] for path in
tqdm(x)]
        x = [librosa.effects.trim(data)[0] for data in
tqdm(x)]

        gfeat = np.load('../input/gfeat.npy')[-len(x):]


        df = pd.read_csv(f'../input/train_curated.csv')
        val_y = split_and_label(df['labels'].values)
        val_x = train_dir + df['fname'].values
        val_x = [librosa.load(path, 44100)[0] for path
in tqdm(val_x)]
        val_x = [librosa.effects.trim(data)[0] for data
in tqdm(val_x)]
        val_gfeat = np.load('../input/gfeat.npy')
[:len(val_x)]

    print(cfg)

    if True: # init
        K.clear_session()
        model = get_model(cfg)
        best_score = -np.inf
    for epoch in range(35):

        if epoch in cfg.milestones:
            K.set_value(model.optimizer.lr,
K.get_value(model.optimizer.lr) * cfg.gamma)

        tr_loader = FreeSound(x, gfeat, y, cfg, 'train',
epoch)
        val_loaders = [FreeSound(val_x, val_gfeat,
val_y, cfg, f'pred{i+1}', epoch) for i in range(3)]

        model.fit_generator(
            tr_loader,
            steps_per_epoch=len(tr_loader),
            verbose=0,
            workers=6
        )
        val_pred = [model.predict_generator(vl,
workers=4) for vl in val_loaders]
        ave_val_pred = np.average(val_pred, axis=0)
        score =
label_ranking_average_precision_score(val_y,
ave_val_pred)

        if epoch >= 28 and score > best_score:
            best_score = score
            model.save_weights(f"../model/{cfg.name}
pretrainedbest.h5")

        if epoch >= 28:
            model.save_weights(f"../model/{cfg.name}
pretrained{epoch}.h5")
            print(f'{epoch} score {score},  best
{best_score}...')




if __name__ == '__main__':
    from models import *

    cfg = Config(
        duration=5,
        name='v1mix',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.7,
        lm=0.01,
        pool_mode=('max', 'avemax1'),
        x1_rate=0.7,
        milestones=(8,12,16),
        get_backbone=get_conv_backbone
    )
    main(cfg, cnn_model)

    cfg = Config(
        duration=5,
        name='model_MSC_se_r4_1.0_10fold',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.7,
        lm=0.01,
        pool_mode=('max', 'avemax1'),
        x1_rate=0.7,
        milestones=(8, 12, 16),
        get_backbone=model_se_MSC,
        w_ratio=1,
    )
    main(cfg, cnn_model)

    cfg = Config(
        duration=5,
        name='model_MSC_se_r4_2.0_10fold',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.7,
        lm=0.01,
        pool_mode=('max', 'avemax1'),
        x1_rate=0.7,
        milestones=(8, 12, 16),
        get_backbone=model_se_MSC,
        w_ratio=2.0,
    )
    main(cfg, cnn_model)
    cfg = Config(
        duration=5,
        name='model_se_r4_1.5_10fold',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.7,
        lm=0.01,
        pool_mode=('max', 'avemax1'),
        x1_rate=0.7,
        milestones=(8, 12, 16),
        get_backbone=model_se_MSC,
        w_ratio=1.5,
    )
    main(cfg, cnn_model)

    cfg = Config(
        duration=5,
        name='se',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.7,
        lm=0.01,
        pool_mode=('max', 'avemax1'),
        x1_rate=0.7,
        milestones=(8, 12, 16),
        get_backbone=get_se_backbone
    )
    main(cfg, cnn_model)
```

```
----------
4. train.py
import tensorflow as tf
import keras.backend.tensorflow_backend as KTF
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
sess = tf.Session(config=config)
```

```
KTF.set_session(sess)
from sklearn.metrics import
label_ranking_average_precision_score
from sklearn.model_selection import StratifiedKFold
from utils import *
from config import *
from iterstrat.ml_stratifiers import
MultilabelStratifiedKFold
from models import *
import pickle
import multiprocessing as mlp
# seed = 3921
# random.seed(seed)
# os.environ['PYTHONHASHSEED'] = f'{seed}'
# np.random.seed(seed)

def worker_prepocess(file_path):

    result = []
    for path in tqdm(file_path):
        data = librosa.load(path, 44100)[0]
        data = librosa.effects.trim(data)[0]

        result.append(data)
    return result

def prepocess_para(file_path):

    workers = mp.cpu_count() // 2
    pool = mp.Pool(workers)
    results = []
    ave_task = (len(file_path) + workers - 1) // workers
    for i in range(workers):
        res = pool.apply_async(worker_prepocess,
                               args=(file_path[i *
ave_task:(i + 1) * ave_task],))
        results.append(res)
    pool.close()
    pool.join()

    dataset = []
    for res in results:
        dataset += res.get()
    return dataset
```

```
def main(cfg,get_model):

    if True: # load data
        df = pd.read_csv(f'../input/train_curated.csv')
        y = split_and_label(df['labels'].values)
        x = train_dir + df['fname'].values
        # # x = prepocess_para(x)

        x = [librosa.load(path, 44100)[0] for path in
tqdm(x)]
        x = [librosa.effects.trim(data)[0] for data in
tqdm(x)]
        # with open('../input/tr_logmel.pkl', 'rb') as f:
        #     x = pickle.load(f)
        gfeat = np.load('../input/gfeat.npy')[:len(y)]



    print(cfg)
    mskfold = MultilabelStratifiedKFold(cfg.n_folds,
shuffle=False, random_state=66666)
    folds = list(mskfold.split(x,y))[::-1]
    # te_folds = list(mskfold.split(te_x,
(te_y>0.5).astype(int)))

    oofp = np.zeros_like(y)
    for fold, (tr_idx, val_idx) in enumerate(folds):
        if fold not in cfg.folds:
            continue
        print("Beginning fold {}".format(fold + 1))

        if True: # init
            K.clear_session()
            model = get_model(cfg)
            best_epoch = 0
            best_score = -1

        for epoch in range(40):
            if epoch >=35 and epoch - best_epoch > 10:
                break

            if epoch in cfg.milestones:

K.set_value(model.optimizer.lr,K.get_value(model.optimizer.lr)
* cfg.gamma)
```

```
            tr_x, tr_y, tr_gfeat = [x[i] for i in 
tr_idx], y[tr_idx], gfeat[tr_idx]
            val_x, val_y, val_gfeat = [x[i] for i in 
val_idx], y[val_idx], gfeat[val_idx]

            tr_loader = FreeSound(tr_x, tr_gfeat, tr_y, 
cfg, 'train',epoch)
            val_loaders = [FreeSound(val_x, val_gfeat, 
val_y, cfg, f'pred{i+1}',epoch) for i in range(3)]

            model.fit_generator(
                tr_loader,
                steps_per_epoch=len(tr_loader),
                verbose=0,
                workers=6
            )
            val_pred = 
[model.predict_generator(vl,workers=4) for vl in 
val_loaders]
            ave_val_pred = np.average(val_pred,axis=0)
            score = 
label_ranking_average_precision_score(val_y,ave_val_pred)

            if score > best_score:
                best_score = score
                best_epoch = epoch
                oofp[val_idx] = ave_val_pred
                model.save_weights(f"../model/{cfg.name}
{fold}.h5")
            print(f'{epoch} score {score} ,  best 
{best_score}...')

    print('lrap: 
',label_ranking_average_precision_score(y,oofp))
        # best_threshold, best_score, raw_score = 
threshold_search(Y, oofp)
        # print(f'th {best_threshold}, val raw_score 
{raw_score}, val best score:{best_score}')

if __name__ == '__main__':
    from models import *

    cfg = Config(
        duration=5,
        name='v1mix',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.6,
        lm=0.01,
        pool_mode=('max', 'avemax1'),
        x1_rate=0.7,
        n_folds=10,
        get_backbone=get_conv_backbone,
        pretrained='../model/v1mixpretrainedbest.h5',
    )
    main(cfg, cnn_model)

    cfg = Config(
        duration=5,
        name='max3exam',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.6,
        lm=0.01,
        pool_mode=('max', 'avemax3'),
        x1_rate=0.7,
        n_folds=10,
        get_backbone=get_conv_backbone,
        pretrained='../model/v1mixpretrainedbest.h5',
    )
    main(cfg, cnn_model)

    cfg = Config(
        duration=5,
        name='model_MSC_se_r4_1.0_10fold',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.6,
        lm=0.01,
        pool_mode=('max', 'avemax1'),
        x1_rate=0.7,
        n_folds=10,
        get_backbone=model_se_MSC,
        w_ratio=1,
        pretrained='../model/
```

```
model_MSC_se_r4_1.0_10foldpretrainedbest.h5',
    )
    main(cfg, cnn_model)

    cfg = Config(
        duration=5,
        name='model_MSC_se_r4_2.0_10fold',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.6,
        lm=0.01,
        pool_mode=('max', 'avemax1'),
        x1_rate=0.7,
        n_folds=10,
        get_backbone=model_se_MSC,
        w_ratio=2.0,
        pretrained='../model/
model_MSC_se_r4_2.0_10foldpretrainedbest.h5',
    )
    main(cfg, cnn_model)

    cfg = Config(
        duration=5,
        name='model_se_r4_1.5_10fold',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.6,
        lm=0.01,
        pool_mode=('max', 'avemax1'),
        x1_rate=0.7,
        n_folds=10,
        get_backbone=model_se_MSC,
        w_ratio=1.5,
        pretrained='../model/
model_se_r4_1.5_10foldpretrainedbest.h5',
    )
    main(cfg, cnn_model)

    cfg = Config(
        duration=5,
        name='se',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.6,
        lm=0.01,
        pool_mode=('max', 'avemax1'),
        x1_rate=0.7,
        n_folds=10,
        get_backbone=get_se_backbone,
        pretrained='../model/sepretrainedbest.h5',
    )
    main(cfg, cnn_model)




----------
5 predict.py


import pandas as pd
from utils import *
from iterstrat.ml_stratifiers import
MultilabelStratifiedKFold
import keras.backend as K
from sklearn.metrics import
label_ranking_average_precision_score
from tqdm import tqdm
from models import *

def get_oofp(cfg, get_model):
    if True: # load data
        df = pd.read_csv(f'../input/train_curated.csv')
        y = split_and_label(df['labels'].values)
        x = train_dir + df['fname'].values
        # # x = prepocess_para(x)

        x = [librosa.load(path, 44100)[0] for path in
tqdm(x)]
        x = [librosa.effects.trim(data)[0] for data in
tqdm(x)]
        # with open('../input/tr_logmel.pkl', 'rb') as f:
        #     x = pickle.load(f)
        gfeat = np.load('../input/gfeat.npy')[:len(y)]

    mskfold = MultilabelStratifiedKFold(cfg.n_folds,
shuffle=False, random_state=66666)
    folds = list(mskfold.split(x, y))
    # te_folds = list(mskfold.split(te_x,
(te_y>0.5).astype(int)))

    oofp = np.zeros_like(y)
    model = get_model(cfg)
    for fold, (tr_idx, val_idx) in
tqdm(enumerate(folds)):

        if True: # init
            model.load_weights(f"../model/{cfg.name}
{fold}.h5")

        val_x, val_y, val_gfeat = [x[i] for i in
val_idx], y[val_idx], gfeat[val_idx]
        val_loaders = [FreeSound(val_x, val_gfeat,
val_y, cfg, f'pred{i + 1}', 40) for i in range(3)]

        val_pred = [model.predict_generator(vl,
workers=4) for vl in val_loaders]
        ave_val_pred = np.average(val_pred, axis=0)
        oofp[val_idx] = ave_val_pred

    print(label_ranking_average_precision_score(y,oofp))

    np.save(f'../output/{cfg.name}oof',oofp)

def predict_test(cfg,get_model):
    test = pd.read_csv('../input/sample_submission.csv')
    x = [librosa.load(path, 44100)[0] for path in
tqdm('../input/audio_test/' + test['fname'].values)]
    Gfeat = np.array([get_global_feat(data, 128) for
data in tqdm(x)])
    x = [librosa.effects.trim(data)[0] for data in
tqdm(x)]

    y =
test[test.columns[1:].tolist()].values.astype(float)
    model = get_model(cfg)
    for fold in range(cfg.n_folds):
        val_loaders = [FreeSound(x, Gfeat, y, cfg,
f'pred{i + 1}',40) for i in range(3)]
        model.load_weights(f"../model/{cfg.name}
{fold}.h5")
        y += np.average([model.predict_generator(vl,
workers=4, verbose=1) for vl in val_loaders], axis=0)
    y /= cfg.n_folds

    np.save(f'../output/{cfg.name}pred',y)



if __name__ == '__main__':

    cfg = Config(
        duration=5,
        name='v1mix',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.6,
        lm=0.01,
        pool_mode=('max', 'avemax1'),
        n_folds=10,
        get_backbone=get_conv_backbone,
    )
    get_oofp(cfg, cnn_model)
    predict_test(cfg, cnn_model)

    cfg = Config(
        duration=5,
        name='max3exam',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.6,
        lm=0.01,
        pool_mode=('max', 'avemax3'),
        x1_rate=0.7,
        n_folds=10,
        get_backbone=get_conv_backbone,
    )
    get_oofp(cfg, cnn_model)
    predict_test(cfg, cnn_model)

    cfg = Config(
        duration=5,
        name='model_MSC_se_r4_1.0_10fold',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.6,
        lm=0.01,
        pool_mode=('max', 'avemax1'),
        x1_rate=0.7,
        n_folds=10,
        get_backbone=model_se_MSC,
        w_ratio=1,
    )
    get_oofp(cfg, cnn_model)
    predict_test(cfg, cnn_model)

    cfg = Config(
        duration=5,
        name='model_MSC_se_r4_2.0_10fold',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.6,
        lm=0.01,
        pool_mode=('max', 'avemax1'),
        x1_rate=0.7,
        n_folds=10,
        get_backbone=model_se_MSC,
        w_ratio=2.0,
    )
    get_oofp(cfg, cnn_model)
    predict_test(cfg, cnn_model)

    cfg = Config(
        duration=5,
        name='model_se_r4_1.5_10fold',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.6,
        lm=0.01,
        pool_mode=('max', 'avemax1'),
        x1_rate=0.7,
        n_folds=10,
        get_backbone=model_se_MSC,
        w_ratio=1.5,
    )
    get_oofp(cfg, cnn_model)
    predict_test(cfg, cnn_model)

    cfg = Config(
        duration=5,
        name='se',
        lr=0.0005,
        batch_size=32,
        rnn_unit=128,
        momentum=0.85,
        mixup_prob=0.6,
        lm=0.01,
        pool_mode=('max', 'avemax3'),
        x1_rate=0.7,
        n_folds=10,
        get_backbone=get_se_backbone,
    )
    get_oofp(cfg, cnn_model)
    predict_test(cfg, cnn_model)


----------
6 ensemble.py

from utils import *
from sklearn.metrics import
label_ranking_average_precision_score
from iterstrat.ml_stratifiers import
MultilabelStratifiedKFold
from models import stacker
from keras import backend as K





def stacking(cfg,files):

    print(list(files.keys()))
    ave_oof, ave_pred = average(cfg,files,True)
    tr_oof_files = [np.load(f'../output/{name}oof.npy')
[:,:,np.newaxis] for name in files.keys()] +
[ave_oof[:,:,np.newaxis]]
    tr_oof = np.concatenate(tr_oof_files,axis=-1)
    test_files = [np.load(f'../output/{name}pred.npy')
[:,:,np.newaxis] for name in files.keys()] +
[ave_pred[:,:,np.newaxis]]
    test_pred = np.concatenate(test_files,axis=-1)
    df = pd.read_csv(f'../input/train_curated.csv')
    y = split_and_label(df['labels'].values)


    mskfold = MultilabelStratifiedKFold(cfg.n_folds,
shuffle=False, random_state=66666)
    folds = list(mskfold.split(y, y))

    predictions = np.zeros_like(test_pred)[:,:,0]
    oof = np.zeros_like((y))
    for fold, (tr_idx, val_idx) in enumerate(folds):
        print('fold ',fold)
        if True:  # init
            K.clear_session()
            model = stacker(cfg,tr_oof.shape[2])
            best_epoch = 0
            best_score = -1

        for epoch in range(1000):
            if epoch - best_epoch > 15:
                break


            tr_x, tr_y = tr_oof[tr_idx], y[tr_idx]
            val_x, val_y = tr_oof[val_idx], y[val_idx]

            val_pred = model.predict(val_x)

            score =
label_ranking_average_precision_score(val_y, val_pred)

            if score > best_score:
                best_score = score
                best_epoch = epoch
                oof[val_idx] = val_pred
                model.save_weights(f"../model/
stacker{cfg.name}{fold}.h5")

            model.fit(x=tr_x, y=tr_y, batch_size=cfg.bs,
verbose=0)
            print(f'{epoch} score {score} ,  best
{best_score}...')

        model.load_weights(f"../model/stacker{cfg.name}
{fold}.h5")
        predictions += model.predict(test_pred)

    print('lrap: ',
label_ranking_average_precision_score(y, oof))
    predictions /= cfg.n_folds
    print(label_ranking_average_precision_score(y,oof))
    test = pd.read_csv('../input/sample_submission.csv')
    test.loc[:, test.columns[1:].tolist()] = predictions
    test.to_csv('submission.csv', index=False)

def average(cfg,files,return_pred = False):
    df = pd.read_csv(f'../input/train_curated.csv')
    y = split_and_label(df['labels'].values)

    result = 0
    oof = 0
    all_w = 0
    for name,w in files.items():
        oof += w * np.load(f'../output/{name}oof.npy')
        print(name,'lrap
',label_ranking_average_precision_score(y,np.load(f'../
output/{name}oof.npy')))
        result += w * np.load(f'../output/{name}
pred.npy')
        all_w += w

    oof /= all_w
    result /= all_w
    print(label_ranking_average_precision_score(y,oof))
    if return_pred:
        return oof,result
    test = pd.read_csv('../input/sample_submission.csv')
    test.loc[:, test.columns[1:].tolist()] = result
    test.to_csv('../submissions/submission.csv',
index=False)
    # print(test)



if __name__ == '__main__':

    cfg = Config(n_folds=10,lr = 0.0001, batch_size=40)
    # stacking(cfg,{
    #     'model_MSC_se_r4_1.0_10fold_withpretrain_e28_':
1.0,
    #     'max3exam':2.1,
    #     'v1mix':2.4,
    #     'model_MSC_se_r4_2.0_10fold_withpretrain_e28_':
1.0,
    #     # 'model_se_r4_1.5_10fold_withpretrain_e28_':
1.0,
    #     'se_':1,
    #     # 'concat_v1':0,
    #     'se_concat':1,
    #
    # })

    # stacking(cfg, {
    #
'model_MSC_se_r4_1.0_10fold_withpretrain_e28_': 1.0,
    #     'max3exam': 1.9,
    #     'v1mix': 2.1,
    #
'model_MSC_se_r4_2.0_10fold_withpretrain_e28_': 1.0,
    #     'model_se_r4_1.5_10fold_withpretrain_e28_':1.0,
    #     'se_': 0,
    # })

    stacking(cfg, {
        'model_MSC_se_r4_1.0_10fold': 1.0,
        'max3exam': 1.9,
        'v1mix': 2.1,
        'model_MSC_se_r4_2.0_10fold': 1.0,
        'model_se_r4_1.5_10fold': 1.0,
        'se_': 0,
    })

----------
```