Kaggle Freesound Audio Tagging 2019 2nd place code

Usage

- Download the datasets and place them in the input folder.
- Unzip the train_curated.zip and train_noisy.zip, then put all the audio clips into audio_train.
- sh run.sh

requirements

tensorflow_gpu==1.11.0 numpy==1.14.2 tqdm==4.22.0 librosa==0.6.3 scipy==1.0.0 iterative_stratification==0.1.6 Keras==2.1.5 pandas==0.24.2 scikit_learn==0.21.2

Hardware

- 64GB of RAM
- 1 tesla P100

Solution

single model CV: 0.89763

ensemble CV: 0.9108

feature engineering

- log mel (441,64) (time,mels)
- global feature (128,12) (Split the clip evenly, and create 12 features for each frame. local cv +0.005)
- length

```
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,
        range_feat = local_x.max()-local_x.min()
        ts.append([np.mean(local_x),np.std(local_x),range_feat]+percent_f
    ts = np.array(ts)
    assert ts.shape == (128,12),(len(x),ts.shape)
    return ts
```

prepocess

- audio clips are first trimmed of leading and trailing silence
- random select a 5s clip from audio clip

model

For details, please refer to code/models.py *Melspectrogram Layer(code from kapre,We use it to search the hyperparameter of log mel end2end)* Our main model is a 9-layer CNN. In this competition, we consider that the two axes of the log mel feature have different physical meanings, so the max pooling and average pooling in the model are replaced by one axis using max pooling and the other axis using average pooling. (Our local cv gain a lot from it, but the exact number is forgotten). *global pooling: pixelshuffle + max pooling in time axes + ave pooling in mel axes.* se block (several of our models use se block) *highway + 1*1 conv (several of our models use se block) * label smoothing

```
# log mel layer
x_mel = Melspectrogram(n_dft=1024, n_hop=cfg.stride, input_shape=(1, K.in
                            # n_hop -> stride   n_dft kernel_size
                            padding='same', sr=44100, n_mels=64,
                            power_melgram=2, return_decibel_melgram=True,
                            trainable_fb=False, trainable_kernel=False,
                            image_data_format='channels_last', trainable=F


# pooling mode
x = AveragePooling2D(pool_size=(pool_size1,1), padding='same', strides=(s
x = MaxPool2D(pool_size=(1,pool_size2), padding='same', strides=(1,stride


# model head
def pixelShuffle(x):
    _,h,w,c = K.int_shape(x)
    bs = K.shape(x)[0]
    assert w%2==0
    x = K.reshape(x,(bs,h,w//2,c*2))

    # assert h % 2 == 0
    # x = K.permute_dimensions(x,(0,2,1,3))
    # x = K.reshape(x,(bs,w//2,h//2,c*4))
    # x = K.permute_dimensions(x,(0,2,1,3))
    return x
x = Lambda(pixelShuffle)(x)
x = Lambda(lambda x: K.max(x, axis=1))(x)
x = Lambda(lambda x: K.mean(x, axis=1))(x)
```

data augmentation

- mixup (local cv +0.002, lb +0.008)
- random select 5s clip + random padding
- 3TTA

pretrain

- train a model only on train_noisy as pretrained model

ensemble

For details, please refer to code/ensemble.py * We use nn for stacking, which uses localconnect1D to learn the ensemble weights of each class, then use fully connect to learn about label correlation, using some initialization and weight constraint tricks.

```
def stacker(cfg,n):
    def kinit(shape, name=None):
        value = np.zeros(shape)
        value[:, -1] = 1
        return K.variable(value, name=name)


    x_in = Input((80,n))
    x = x_in
    # x = Lambda(lambda x: 1.5*x)(x)
    x = LocallyConnected1D(1,1,kernel_initializer=kinit,kernel_constraint
    x = Flatten()(x)
    x = Dense(80, use_bias=False, kernel_initializer=Identity(1))(x)
    x = Lambda(lambda x: (x - 1.6))(x)
    x = Activation('tanh')(x)
    x = Lambda(lambda x:(x+1)*0.5)(x)

    model = Model(inputs=x_in, outputs=x)
    model.compile(
        loss='binary_crossentropy',
        optimizer=Nadam(lr=cfg.lr),
    )
    return model
```