6th place solution for Freesound Audio Tagging 2019 Competition

Description of the solution: https://link.medium.com/Kv5kyHjcIX

How to use

- Install fastai and librosa:

```
conda install -c pytorch -c fastai fastai
conda install librosa
```

- Clone the repository:

```
git clone https://github.com/mnpinto/audiotagging2019.git
```

- Download the competition data from kaggle (https://www.kaggle.com/c/freesound-audio-tagging-2019/data) to `audiotagging2019/data/` folder and unzip the `test.zip`, `train_curated.zip` and `train_noisy.zip` to folders `test`, `train_curated` and `train_noisy`.
- On `audiotagging2019` folder run:

```
python run.py --n_epochs 1 --max_processors 8
```

If successful the script will create `train_curated_png` and `train_noisy_png` folders with the Mel spectrograms corresponding to all audio clips and train the model for `1` epochs using the default arguments. The `max_processors` argument will set how many processors to use to this preprocessing step. After the training is complete a folder `models` will be created and a weights file `stage-1.pth` will be saved their. Finally a submission file will be generated with the default name `submission.csv`.

If you find any errors let me know by creating an Issue, the code has not yet been tested on fastai versions after 1.0.51.

Arguments

name	type	default	description
`--path`	`str`	`data`	path to data folder
`--working_path`	`str`	`.`	path to working folder where model weights and outputs will be saved
`--base_dim`	`int`	`128`	size to crop the images on the horizontal axis before rescaling with `SZ`
`--SZ`	`int`	`128`	images will be rescaled to

name	type	default	description
			SZxSZ
--BS	int	64	batch size
--lr	float	0.01	maximum learning rate for one_cycle_learning
--n_epochs	int	80	number of epochs to train the model
--epoch_size	int	1000	number of episodes (with batch size BS each) in each epoch
--f2cl	int	1	train only on samples with F2 score (with threshold of 0.2) less than f2cl
--fold_number	int	0	KFold cross-validation fold number: (0,1,2,3,4) or -1 to train with all data
--loss_name	str	BCELoss	loss function to use, options are BCELoss and FocalLoss
--csv_name	str	submission.csv	name of csv file to save with test predictions
--model	str	models.xresnet18	can be a fastai model as the default or xresnet{18,34,50}ssa to use simple self-attention
--weights_file	str	stage-1	name of file to save the weights
--load_weights	str		provide the name of weights file (e.g., stage-1) to load before training
--max_processors	int	8	number cpu threads to use for converting wav files to png
--force	bool	False	if set to True the pngs will be recomputed for noisy and curated train datasets

Replicating my top scoring solution

Important! This code has not yet been tested. I ran all experiments on Kaggle kernels and refactored the code to create this repository. After the final results of the competition are available, late submissions will be allowed so I will then test the code to check if anything is missing.

My top scoring solution with a score of **0.742 on public LB** and **0.75421 on private LB** (final results pending...) is the average of the following 6 runs:

```
python run.py --model xresnet18ssa --base_dim 128 --SZ 256 --fold_number
              --n_epochs 80 --loss_name FocalLoss --weights_file model1
```

```
python run.py --model xresnet34ssa --base_dim 128 --SZ 256 --fold_number
              --n_epochs 60 --loss_name FocalLoss --weights_file model2

python run.py --model xresnet18ssa --base_dim 128 --SZ 256 --fold_number
              --n_epochs 80 --weights_file model3 --csv_name submission3

python run.py --model xresnet34ssa --base_dim 128 --SZ 256 --fold_number
              --n_epochs 60 --weights_file model4 --csv_name submission4

python run.py --model models.xresnet34 --base_dim 128 --SZ 256 --fold_nu
              --n_epochs 90 --loss_name FocalLoss --weights_file model5

python run.py --model models.xresnet50 --base_dim 128 --SZ 256 --fold_nu
              --n_epochs 65 --weights_file model6_0

python run.py --model models.xresnet50 --base_dim 128 --SZ 256 --fold_nu
              --n_epochs 65 --load_weights model6_0 --weights_file model
```

The penultimate run, generating `model6_0` weights is not used for the ensemble, is just to generate the weights that are used to the last identical run. If you are running locally, try a single run with more epochs, the `2x65` epochs is just to accommodate for the `9h` run-time limit of Kaggle kernels.

Citing this repository

```
@misc{mnpinto2019audio,
  author = {Pinto, M. M.},
  title = {6th place solution for Freesound Audio Tagging 2019 Competiti
  year = {2019},
  publisher = {GitHub},
  journal = {GitHub repository},
  howpublished = {\url{https://github.com/mnpinto/audiotagging2019}}
}
```

MIT License

Copyright (c) 2019 Miguel Mota Pinto

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.

```
import fastai
from fastai.vision import *
from torch.utils.data.dataloader import default_collate
from torch.utils.data import Sampler, SequentialSampler, RandomSampler
import sklearn


# Modification to ImageDataBunch to allow to give a list of custome samplers.
class ImageDataBunch(ImageDataBunch):
    @classmethod
    def create(cls, train_ds:Dataset, valid_ds:Dataset, test_ds:Optional[Dataset]=None, path:PathOrStr='.', bs:int=64,
               val_bs:int=None, num_workers:int=defaults.cpus, dl_tfms:Optional[Collection[Callable]]=None,
               device:torch.device=None, collate_fn:Callable=data_collate, no_check:bool=False, samplers=None, **dl_kwargs)->'DataBunch':
        "Create a `DataBunch` from `train_ds`, `valid_ds` and maybe `test_ds` with a batch size of `bs` and optionally a list of samplers."
        datasets = cls._init_ds(train_ds, valid_ds, test_ds)
```

```
        val_bs = ifnone(val_bs, bs)
        if samplers is None: samplers = [RandomSampler] + 3*[Seq
uentialSampler]
        dls = [DataLoader(d, b, sampler=s(d, bs=b), num_workers=
num_workers, **dl_kwargs) for d,b,s in
               zip(datasets, (bs,val_bs,val_bs,val_bs), samplers
) if d is not None]
        return cls(*dls, path=path, device=device, dl_tfms=dl_tf
ms, collate_fn=collate_fn, no_check=no_check)


class ImageList(ImageList):
    _bunch = ImageDataBunch


class SequentialSampler(SequentialSampler):
    def __init__(self, data_source, **kwargs):
        self.data_source = data_source


class RandomSampler(RandomSampler):
    def __init__(self, data_source, replacement=False, num_sampl
es=None, **kwargs):
        self.data_source = data_source
        self.replacement = replacement
        self.num_samples = num_samples


class FixedLenRandomSampler(RandomSampler):
    """Sample epochs with a fixed length"""
    def __init__(self, data_source, bs, epoch_size, *args, **kwa
rgs):
        super().__init__(data_source)
        self.epoch_size = epoch_size*bs

    def __iter__(self):
        return iter(np.random.choice(range(len(self.data_source)
), size=len(self), replace=True).tolist())

    def __len__(self):
        return self.epoch_size


def load_image(fn:PathOrStr, div:bool=True, convert_mode:str='RG
B', cls:type=Image,
        after_open:Callable=None)->Image:
    "Return `Image` cropped and resized."
    with warnings.catch_warnings():
        warnings.simplefilter("ignore", UserWarning) # EXIF warn
ing from TiffPlugin
        if Path(fn).parent.name == 'train':
            ind = train_df.loc[Path(fn).name, 'ind']
            x = X_train[ind]
        else:
            ind = test_df.loc[Path(fn).name, 'ind']
            x = X_test[ind]
```

```
        _, time_dim = x.shape
        if time_dim - base_dim > 0:
            crop_x = np.random.randint(0, time_dim - base_dim)
            x = x[:, crop_x:crop_x+base_dim]
        x = PIL.Image.fromarray(x).resize((SZ,SZ)).convert(conve
rt_mode)
    if after_open: x = after_open(x)
    x = pil2tensor(x,np.float32)
    if div: x.div_(255)
    return cls(x)


def load_image_tta(fn:PathOrStr, div:bool=True, convert_mode:str
='RGB', cls:type=Image,
        after_open:Callable=None, flip=False, vert=False, step=1
28)->Image:
    with warnings.catch_warnings():
        warnings.simplefilter("ignore", UserWarning) # EXIF warn
ing from TiffPlugin
        if Path(fn).parent.name == 'train':
            ind = train_df.loc[Path(fn).name, 'ind']
            x = X_train[ind]
        else:
            ind = test_df.loc[Path(fn).name, 'ind']
            x = X_test[ind]
        if flip: x = np.fliplr(x)
        if vert: x = np.flipud(x)
        _, time_dim = x.shape
        xb = []
        for n in range(0, max(1, time_dim-base_dim), step):
            x0 = PIL.Image.fromarray(x[:,n:n+base_dim]).resize((
SZ,SZ)).convert(convert_mode)
            if after_open: x0 = after_open(x0)
            x0 = pil2tensor(x0, np.float32)
            if div: x0.div_(255)
            x0 = normalize(x0, mean=tensor([0.2932, 0.2932, 0.29
32]), std=tensor([0.2556, 0.2556, 0.2556]))
            xb.append(x0[None])
        xb = torch.cat(xb, dim=0)
    return xb


class ImageListMemory(ImageList):
    """ImageList that load images from memory using load_image f
unction"""
    def __init__(self, *args, convert_mode='L', after_open:Calla
ble=None, **kwargs):
        super().__init__(*args, **kwargs)
        self.convert_mode,self.after_open = convert_mode,after_o
pen
        self.copy_new.append('convert_mode')
        self.c,self.sizes = 1,{}

    def open(self, fn):
        "Open image in `fn`, subclass and overwrite for custom b
ehavior."
```

```
        return load_image(fn, convert_mode=self.convert_mode, af
ter_open=self.after_open)


def _cutout(x, n_holes:uniform_int=1, length:uniform_int=40):
    "Cut out `n_holes` number of rectangular bands of size `leng
th` in image at random locations."
    h,w = x.shape[1:]
    for n in range(n_holes):
        h_y = np.random.randint(0, h)
        h_x = np.random.randint(0, w)
        y1 = int(np.clip(h_y - length / 2, 0, h))
        y2 = int(np.clip(h_y + length / 2, 0, h))
        x1 = int(np.clip(h_x - length / 2, 0, w))
        x2 = int(np.clip(h_x + length / 2, 0, w))
        x[:, y1:y2, :] = 0
        x[:, :, x1:x2] = 0
    return x
cutout2 = TfmPixel(_cutout, )


class BCELoss(nn.Module):
    def __init__(self, reduce=False):
        super().__init__()
        self.reduce = reduce

    def forward(self, logit, target):
        target = target.float()
        loss = nn.BCEWithLogitsLoss()(logit, target)
        if len(loss.size())==2:
            loss = loss.sum(dim=1)
        if not self.reduce:
            return loss
        else:
            return loss.mean()


# Adapted from https://www.kaggle.com/c/human-protein-atlas-imag
e-classification/discussion/78109
class FocalLoss(nn.Module):
    def __init__(self, gamma=2, reduce=False):
        super().__init__()
        self.gamma = gamma
        self.reduce = reduce

    def forward(self, logit, target):
        target = target.float()
        max_val = (-logit).clamp(min=0)
        loss = logit - logit * target + max_val + \
               ((-max_val).exp() + (-logit - max_val).exp()).log
()

        invprobs = F.logsigmoid(-logit * (target * 2.0 - 1.0))
        loss = (invprobs * self.gamma).exp() * loss
        if len(loss.size())==2:
            loss = loss.sum(dim=1)
```

```
        if not self.reduce:
            return loss
        else:
            return loss.mean()


def fbeta2(y_pred:Tensor, y_true:Tensor, thresh:float=0.2, beta:
float=2, eps:float=1e-9, sigmoid:bool=True)->Rank0Tensor:
    "Computes the f_beta between `preds` and `targets`"
    beta2 = beta ** 2
    if sigmoid: y_pred = y_pred.sigmoid()
    y_pred = (y_pred>thresh).float()
    y_true = y_true.float()
    TP = (y_pred*y_true).sum(dim=1)
    prec = TP/(y_pred.sum(dim=1)+eps)
    rec = TP/(y_true.sum(dim=1)+eps)
    res = (prec*rec)/(prec*beta2+rec+eps)*(1+beta2)
    return res


class MixupBCELoss(BCELoss):
    def forward(self, x, y):
        if isinstance(y, dict):
            y0, y1, a = y['y0'], y['y1'], y['a']
            loss = a*super().forward(x, y0) + (1-a)*super().forw
ard(x, y1)

            if f2cl is not None:
                # Removing samples with F2 score equal to f2cl
                fbs = fbeta2(x, y0*a.view(-1,1)+(1-a.view(-1,1))
*y1)
                loss = loss[(fbs<f2cl).byte()]
        else:
            loss = super().forward(x, y)
        return 100*loss.mean()


class MixupFocalLoss(FocalLoss):
    def forward(self, x, y):
        if isinstance(y, dict):
            y0, y1, a = y['y0'], y['y1'], y['a']
            loss = a*super().forward(x, y0) + (1-a)*super().forw
ard(x, y1)

            if f2cl is not None:
                # Removing samples with F2 score equal to f2cl
                fbs = fbeta2(x, y0*a.view(-1,1)+(1-a.view(-1,1))
*y1)
                loss = loss[(fbs<f2cl).byte()]
        else:
            loss = super().forward(x, y)
        return loss.mean()


# Calculate the overall lwlrap using sklearn.metrics function.
def lwlrap(scores, truth):
```

```
    """Calculate the overall lwlrap using sklearn.metrics.lrap."
""
    # sklearn doesn't correctly apply weighting to samples with
no labels, so just skip them.
    scores = scores.detach().cpu().numpy()
    truth = truth.detach().cpu().numpy()
    sample_weight = np.sum(truth > 0, axis=1)
    nonzero_weight_sample_indices = np.flatnonzero(sample_weight
 > 0)
    overall_lwlrap = sklearn.metrics.label_ranking_average_preci
sion_score(
      truth[nonzero_weight_sample_indices, :] > 0,
      scores[nonzero_weight_sample_indices, :],
      sample_weight=sample_weight[nonzero_weight_sample_indices]
)
    return tensor(overall_lwlrap)


class AudioMixup(LearnerCallback):
    def __init__(self, learn):
        super().__init__(learn)

    def on_batch_begin(self, last_input, last_target, train, **k
wargs):
        if train:
            bs = last_input.size()[0]
            lambd = np.random.uniform(0, 0.5, bs)
            shuffle = torch.randperm(last_target.size(0)).to(las
t_input.device)
            x1, y1 = last_input[shuffle], last_target[shuffle]
            a = tensor(lambd).float().view(-1, 1, 1, 1).to(last_
input.device)
            last_input = a*last_input + (1-a)*x1
            last_target = {'y0':last_target, 'y1':y1, 'a':a.view
(-1)}
            return {'last_input': last_input, 'last_target': las
t_target}


def get_preds_tta(learn, valid=True, flip=False, vert=False):
    with torch.no_grad():
        preds0 = []
        N = len(learn.data.valid_ds) if valid else len(learn.dat
a.test_ds)
        for i in progress_bar(range(N), total=N):
            if valid:
                xb = load_image_tta(learn.data.valid_ds.items[i]
, flip=flip, vert=vert, step=base_dim)
            else:
                xb = load_image_tta(learn.data.test_ds.items[i],
 flip=flip, vert=vert, step=base_dim)
            out = learn.model(xb.cuda())
            out = out.sigmoid().max(0)[0]
            preds0.append(out[None].cpu())
        preds0 = torch.cat(preds0, dim=0)
        return preds0
```

```
def print_scores(name, preds, ys):
    print(f'{name} | F2={fbeta(preds, ys).item():.4f}; LWL={lwlr
ap(preds, ys).item():.4f}')from fastai.vision import *
import librosa, librosa.display

# Based on: https://www.kaggle.com/daisukelab/cnn-2d-basic-solut
ion-powered-by-fast-ai

conf = {'sr': 44100, 'duration': 2, 'fmin': 20, 'n_mels': 128}
conf['hop_length'] = 347*conf['duration']
conf['fmax'], conf['n_fft'] = conf['sr'] // 2, conf['n_mels'] *
20
conf['samples'] = conf['sr'] * conf['duration']

def read_audio(conf, pathname, trim_long_data):
    y, sr = librosa.load(pathname, sr=conf['sr'])
    # trim silence
    if 0 < len(y): # workaround: 0 length causes error
        y, _ = librosa.effects.trim(y) # trim, top_db=default(60
)
    # make it unified length to conf.samples
    if len(y) > conf['samples']: # long enough
        if trim_long_data:
            y = y[0:0+conf['samples']]
    else: # pad blank
        padding = conf['samples'] - len(y) # add padding at both
 ends
        offset = padding // 2
        y = np.pad(y, (offset, conf['samples'] - len(y) - offset
), 'constant')
    return y

def audio_to_melspectrogram(conf, audio):
    conf2 = conf.copy()
    del conf2['duration'], conf2['samples']
    spectrogram = librosa.feature.melspectrogram(audio, **conf2)
    spectrogram = librosa.power_to_db(spectrogram)
    spectrogram = spectrogram.astype(np.float32)
    return spectrogram

def show_melspectrogram(conf, mels, title='Log-frequency power s
pectrogram'):
    librosa.display.specshow(mels, x_axis='time', y_axis='mel',
sr=conf['sr'], hop_length=conf['hop_length'], fmin=conf['fmin'],
 fmax=conf['fmax'])
    plt.colorbar(format='%+2.0f dB')
    plt.title(title)
    plt.show()

def read_as_melspectrogram(conf, pathname, trim_long_data, debug
_display=False):
    x = read_audio(conf, pathname, trim_long_data)
    mels = audio_to_melspectrogram(conf, x)
    if debug_display:
```

```
        IPython.display.display(IPython.display.Audio(x, rate=co
nf['sr']))
        show_melspectrogram(conf, mels)
    return mels

def mono_to_color(X, mean=None, std=None, norm_max=None, norm_mi
n=None, eps=1e-6):
    # Standardize
    mean = mean or X.mean()
    std = std or X.std()
    Xstd = (X - mean) / (std + eps)
    _min, _max = Xstd.min(), Xstd.max()
    norm_max = norm_max or _max
    norm_min = norm_min or _min
    if (_max - _min) > eps:
        # Normalize to [0, 255]
        V = Xstd
        V[V < norm_min] = norm_min
        V[V > norm_max] = norm_max
        V = 255 * (V - norm_min) / (norm_max - norm_min)
        V = V.astype(np.uint8)
    else:
        # Just zero
        V = np.zeros_like(Xstd, dtype=np.uint8)
    return V

def convert_wav_to_image(df, fold, source):
    X = []
    for i, row in progress_bar(df.iterrows(), total=len(df)):
        x = read_as_melspectrogram(conf, source/str(row.fname),
trim_long_data=False)
        x_color = mono_to_color(x)
        X.append(x_color)
    return X

def convert_wav_to_png(row):
    fname = row[1][0]
    x = read_as_melspectrogram(conf, path_source/str(fname), tri
m_long_data=False)
    x_color = mono_to_color(x)
    fsave = path_save/f'{fname[:-4]}.png'
    PIL.Image.fromarray(x_color).save(fsave)from fastai.script i
mport *
from fastai.vision import *
from fastai.callbacks import *
from fastai.distributed import *
from fastprogress import fastprogress
from torchvision.models import *
from fastai.vision.models.xresnet import *

__all__= ['XResNetssa', 'xresnet18ssa', 'xresnet34ssa', 'xresnet
50ssa', 'xresnet101ssa', 'xresnet152ssa']

# XResnet with Simple Self Attention taken from: https://github.
com/sdoria/SimpleSelfAttention
```

```
def noop(x): return x

class Flatten(nn.Module):
    def forward(self, x): return x.view(x.size(0), -1)

def conv(ni, nf, ks=3, stride=1, bias=False):
    return nn.Conv2d(ni, nf, kernel_size=ks, stride=stride, padd
ing=ks//2, bias=bias)

act_fn = nn.ReLU(inplace=True)

def init_cnn(m):
    if getattr(m, 'bias', None) is not None: nn.init.constant_(m
.bias, 0)
    if isinstance(m, (nn.Conv2d,nn.Linear)): nn.init.kaiming_nor
mal_(m.weight)
    for l in m.children(): init_cnn(l)

def conv_layer(ni, nf, ks=3, stride=1, zero_bn=False, act=True):
    bn = nn.BatchNorm2d(nf)
    nn.init.constant_(bn.weight, 0. if zero_bn else 1.)
    layers = [conv(ni, nf, ks, stride=stride), bn]
    if act: layers.append(act_fn)
    return nn.Sequential(*layers)

def conv1d(ni:int, no:int, ks:int=1, stride:int=1, padding:int=0
, bias:bool=False):
    "Create and initialize a `nn.Conv1d` layer with spectral nor
malization."
    conv = nn.Conv1d(ni, no, ks, stride=stride, padding=padding,
 bias=bias)
    nn.init.kaiming_normal_(conv.weight)
    if bias: conv.bias.data.zero_()
    return spectral_norm(conv)

class SimpleSelfAttention(nn.Module):

    def __init__(self, n_in:int, ks=1):#, n_out:int):
        super().__init__()
        self.conv = conv1d(n_in, n_in, ks, padding=ks//2, bias=F
alse)
        self.gamma = nn.Parameter(tensor([0.]))

    def forward(self,x):
        size = x.size()
        x = x.view(*size[:2],-1)
        o = torch.bmm(x.permute(0,2,1).contiguous(),self.conv(x)
)
        o = self.gamma * torch.bmm(x,o) + x
        return o.view(*size).contiguous()

def conv(ni, nf, ks=3, stride=1, bias=False):
    return nn.Conv2d(ni, nf, kernel_size=ks, stride=stride, padd
ing=ks//2, bias=bias)

def conv_layer(ni, nf, ks=3, stride=1, zero_bn=False, act=True):
```

```
    bn = nn.BatchNorm2d(nf)
    nn.init.constant_(bn.weight, 0. if zero_bn else 1.)
    layers = [conv(ni, nf, ks, stride=stride), bn]
    if act: layers.append(act_fn)
    return nn.Sequential(*layers)

class ResBlock(nn.Module):
    def __init__(self, expansion, ni, nh, stride=1,sa=False):
        super().__init__()


        nf,ni = nh*expansion,ni*expansion
        layers  = [conv_layer(ni, nh, 3, stride=stride),
                   conv_layer(nh, nf, 3, zero_bn=True, act=False
)
        ] if expansion == 1 else [
                   conv_layer(ni, nh, 1),
                   conv_layer(nh, nh, 3, stride=stride),

                   conv_layer(nh, nf, 1, zero_bn=True, act=False
)

        ]

        self.sa = SimpleSelfAttention(nf,ks=1) if sa else noop

        self.convs = nn.Sequential(*layers)
        self.idconv = noop if ni==nf else conv_layer(ni, nf, 1,
act=False)
        self.pool = noop if stride==1 else nn.AvgPool2d(2, ceil_
mode=True)

    def forward(self, x):


        return act_fn(self.sa(self.convs(x)) + self.idconv(self.
pool(x)))

class XResNetssa(nn.Sequential):
    @classmethod
    def create(cls, expansion, layers, c_in=3, c_out=1000):
        nfs = [c_in, (c_in+1)*8, 64, 64]
        stem = [conv_layer(nfs[i], nfs[i+1], stride=2 if i==0 el
se 1)
            for i in range(3)]

        nfs = [64//expansion,64,128,256,512]
        res_layers = [cls._make_layer(expansion, nfs[i], nfs[i+1
],
                                      n_blocks=l, stride=1 if i=
=0 else 2, sa = True if i in[len(layers)-4] else False)
                  for i,l in enumerate(layers)]
        res = cls(
            *stem,
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1),
            *res_layers,
```

```
            nn.AdaptiveAvgPool2d(1), Flatten(),
            nn.Linear(nfs[-1]*expansion, c_out),
        )
        init_cnn(res)
        return res

    @staticmethod
    def _make_layer(expansion, ni, nf, n_blocks, stride, sa = False):
        return nn.Sequential(
            *[ResBlock(expansion, ni if i==0 else nf, nf, stride if i==0 else 1, sa if i in [n_blocks -1] else False)
              for i in range(n_blocks)])

def xresnet18ssa(**kwargs): return XResNetssa.create(1, [2, 2, 2, 2], **kwargs)
def xresnet34ssa(**kwargs): return XResNetssa.create(1, [3, 4, 6, 3], **kwargs)
def xresnet50ssa(**kwargs): return XResNetssa.create(4, [3, 4, 6, 3], **kwargs)
def xresnet101ssa(**kwargs): return XResNetssa.create(4, [3, 4, 23, 3], **kwargs)
def xresnet152ssa(**kwargs): return XResNetssa.create(4, [3, 8, 36, 3], **kwargs)import utils, preprocessing
from utils import *
from preprocessing import *
from models import *
from sklearn.model_selection import KFold
import argparse

def main(path=None, model=None, base_dim=None, SZ=None, BS=None, lr=None,
         n_epochs=None, epoch_size=None, f2cl=None, fold_number=None,
         loss_name=None, csv_name=None, weights_file=None, working_path=None,
         max_processors=None, load_weights=None, force=None):
    utils.base_dim = base_dim
    utils.SZ = SZ
    utils.f2cl = f2cl

    if loss_name == 'BCELoss':
        loss_func = MixupBCELoss()
    elif loss_name == 'FocalLoss':
        loss_func = MixupFocalLoss()
    else: raise NotImplementedError('Choose BCELoss or FocalLoss for the loss_name.')

    # Processing curated train dataset
    if not (path/'train_curated_png').is_dir() or force:
        print('\nComputing mel spectrograms for the curated train dataset and saving as .png:')
        train_df = pd.read_csv(path/'train_curated.csv')
        preprocessing.path_source = path/'train_curated'
        preprocessing.path_save = path/'train_curated_png'
```

```
        preprocessing.path_save.mkdir(exist_ok=True)
        with ThreadPoolExecutor(max_processors) as e:
            list(progress_bar(e.map(convert_wav_to_png, list(tra
in_df.iterrows())), total=len(train_df)))

    # Processing noisy train dataset
    if not (path/'train_noisy_png').is_dir() or force:
        print('\nComputing mel spectrograms for the noisy train
dataset and saving as .png:')
        train_df = pd.read_csv(path/'train_noisy.csv')
        preprocessing.path_source = path/'train_noisy'
        preprocessing.path_save = path/'train_noisy_png'
        preprocessing.path_save.mkdir(exist_ok=True)
        with ThreadPoolExecutor(max_processors) as e:
            list(progress_bar(e.map(convert_wav_to_png, list(tra
in_df.iterrows())), total=len(train_df)))

    # Processing test data
    print('\nComputing mel spectrograms for the test dataset:')
    test_df = pd.read_csv(path/'sample_submission.csv')
    X_test = convert_wav_to_image(test_df, fold='test', source=p
ath/'test')
    test_df['ind'] = test_df.index
    test_df.set_index('fname', inplace=True)
    test_df['fname'] = test_df.index

    # Load indices of noisy data to use
    good_noisy = pd.read_csv(path/'good_idx.csv').idx.values

    # Create train dataframe and list of arrays
    print('\n\nLoading train data:')
    train_df = pd.read_csv(path/'train_curated.csv')
    train_df.loc[:, 'fname'] = [f[:-4] for f in train_df.fname]
    train_noisy_df = pd.read_csv(path/'train_noisy.csv').iloc[go
od_noisy]
    train_noisy_df.loc[:, 'fname'] = [f[:-4] for f in train_nois
y_df.fname]
    X_train_curated = [np.array(PIL.Image.open(path/f'train_cura
ted_png/{fn}.png')) for fn in progress_bar(train_df.fname)]
    X_train_noisy = [np.array(PIL.Image.open(path/f'train_noisy_
png/{fn}.png')) for fn in progress_bar(train_noisy_df.fname)]
    train_df = pd.concat((train_df, train_noisy_df)).reset_index
(drop=True)
    train_df['ind'] = train_df.index
    train_df.set_index('fname', inplace=True)
    train_df['fname'] = train_df.index
    X_train = [*X_train_curated, *X_train_noisy]


    # Flipped images and labels
    for o in progress_bar(X_train.copy()):
        X_train.append(np.fliplr(o))
    train_df_flip = train_df.copy()
    train_df_flip.loc[:, 'labels'] = [','.join([f'{o}_flip' for
o in a.split(',')]) for a in train_df_flip.labels.values]
    train_df_flip.loc[:, 'ind'] = train_df_flip.ind + train_df_f
```

```
lip.ind.max() + 1
    train_df_flip.loc[:, 'fname'] = [f'{o}_flip' for o in train_
df_flip.fname]
    train_df_flip = train_df_flip.set_index('fname', drop=False)
    train_df = pd.concat((train_df, train_df_flip))

    # Vertical flip
    for o in progress_bar(X_train.copy()):
        X_train.append(np.flipud(o))
    train_df_flip_vert = train_df.copy()
    train_df_flip_vert.loc[:, 'labels'] = [','.join([f'{o}_vert'
 for o in a.split(',')]) for a in train_df_flip_vert.labels.valu
es]
    train_df_flip_vert.loc[:, 'ind'] = train_df_flip_vert.ind +
train_df_flip_vert.ind.max() + 1
    train_df_flip_vert.loc[:, 'fname'] = [f'{o}_vert' for o in t
rain_df_flip_vert.fname]
    train_df_flip_vert = train_df_flip_vert.set_index('fname', d
rop=False)
    train_df = pd.concat((train_df, train_df_flip_vert))

    utils.train_df = train_df
    utils.test_df = test_df
    utils.X_train = X_train
    utils.X_test = X_test

    if fold_number is not None:
        kf = KFold(n_splits=5, shuffle=True, random_state=534)

        # Validation indices
        valid_idx = list(kf.split(list(range(len(train_df)//4)))
)[fold_number][1]
        valid_idx_flip = valid_idx +      1*len(train_df)//4
        valid_idx_vert = valid_idx +      2*len(train_df)//4
        valid_idx_flip_vert = valid_idx + 3*len(train_df)//4
        valid_idx = [*valid_idx, *valid_idx_flip, *valid_idx_ver
t, *valid_idx_flip_vert]

    # List of augmentations
    tfms = get_transforms(do_flip=False, max_rotate=0, max_zoom=
1.5, max_warp=0,
                          xtra_tfms=[cutout2(n_holes=(1, 4), len
gth=(5, 20), p=0.75)])

    # ImageLists
    train = ImageListMemory.from_df(train_df, path=path, cols='f
name', folder='train')
    test = ImageListMemory.from_df(test_df, path=path, cols='fna
me', folder='test')

    # Custom samplers
    train_sampler = partial(FixedLenRandomSampler, epoch_size=ep
och_size)
    samplers = [train_sampler, SequentialSampler, SequentialSamp
ler, SequentialSampler]
```

```
    # Create databunch
    if fold_number is None:
        train_split = train.split_none()
    else:
        train_split = train.split_by_idx(valid_idx)

    data = (train_split.label_from_df(cols='labels', label_delim
=',')
            .add_test(test)
            .transform(tfms, size=SZ)
            .databunch(samplers=samplers, path=working_path, bs=
BS)
            .normalize([tensor([0.2932, 0.2932, 0.2932]), tensor
([0.2556, 0.2556, 0.2556])]))

    # Create learner
    mod_name = inspect.getmodule(model).__name__
    if 'fastai.vision.models' in  mod_name or 'torchvision.model
s' in mod_name:
        learn = cnn_learner(data, model, pretrained=False, loss_
func=loss_func, metrics=[fbeta, lwlrap],
                            callback_fns=[AudioMixup])
    else:
        learn = Learner(data, model(c_out=data.c), loss_func=los
s_func, metrics=[fbeta, lwlrap],
                            callback_fns=[AudioMixup])
    learn.clip_grad = 1

    if load_weights is not None:
        print(f'\n\nLoading {load_weights}.pth weights.')
        learn.load(load_weights)

    # Train
    print('\nTraning the model:')
    learn.fit_one_cycle(n_epochs, slice(lr))
    learn.save(weights_file)
    print(f'\nModel weights save to {working_path/"models"/weigh
ts_file}.pth.')

    normal = tensor(['_flip' not in o and '_vert' not in o for o
 in learn.data.classes]).long()
    flip = tensor(['_flip' in o and '_vert' not in o for o in le
arn.data.classes]).long()
    vert = tensor(['_vert' in o and '_flip' not in o for o in le
arn.data.classes]).long()
    flip_vert = tensor(['_flip' in o and '_vert' in o for o in l
earn.data.classes]).long()
    assert sum(normal) == 80
    assert sum(flip) == 80
    assert sum(vert) == 80
    assert sum(flip_vert) == 80

    # Validate only if fold number is not None, otherwise all da
ta has been used to train
    if fold_number is not None:
        print('\nComputing validation scores without TTA:')
```

```
        preds0, ys0 = learn.get_preds(ds_type=DatasetType.Valid)
        S = len(preds0)//4
        preds_normal, ys_normal = preds0[:1*S, normal.byte()], y
s0[:1*S, normal.byte()]
        preds_flip, ys_flip = preds0[1*S:2*S, flip.byte()], ys0[
1*S:2*S, flip.byte()]
        preds_vert, ys_vert = preds0[2*S:3*S, vert.byte()], ys0[
2*S:3*S, vert.byte()]
        preds_flip_vert, ys_flip_vert = preds0[3*S:, flip_vert.b
yte()], ys0[3*S:, flip_vert.byte()]
        print_scores('Mix   ', preds0, ys0)
        print_scores('Normal', preds_normal, ys_normal)
        print_scores('Flip  ', preds_flip, ys_flip)
        print_scores('Vert  ', preds_vert, ys_vert)
        print_scores('FlVert', preds_flip_vert, ys_flip_vert)
        print_scores('Ensble', preds_normal.sigmoid() + preds_fl
ip.sigmoid() + preds_vert.sigmoid() + preds_flip_vert.sigmoid(),
 ys_normal)

        print('\nComputing validation scores with TTA:')
        preds0 = get_preds_tta(learn)
        S = len(preds0)//4
        preds_normal, ys_normal = preds0[:1*S, normal.byte()], y
s0[:1*S, normal.byte()]
        preds_flip, ys_flip = preds0[1*S:2*S, flip.byte()], ys0[
1*S:2*S, flip.byte()]
        preds_vert, ys_vert = preds0[2*S:3*S, vert.byte()], ys0[
2*S:3*S, vert.byte()]
        preds_flip_vert, ys_flip_vert = preds0[3*S:, flip_vert.b
yte()], ys0[3*S:, flip_vert.byte()]
        print_scores('Mix   ', preds0, ys0)
        print_scores('Normal', preds_normal, ys_normal)
        print_scores('Flip  ', preds_flip, ys_flip)
        print_scores('Vert  ', preds_vert, ys_vert)
        print_scores('FlVert', preds_flip_vert, ys_flip_vert)
        preds_ens = torch.cat((preds_normal[None], preds_flip[No
ne], preds_vert[None], preds_flip_vert[None]), dim=0)
        print_scores('Ensble', preds_ens.mean(0), ys_normal)

    # Compute results for test set and generate submission csv.
    print('\nComputing predictions for test data:')
    _ = learn.get_preds(ds_type=DatasetType.Test)
    preds_normal = get_preds_tta(learn, valid=False)
    preds_flip = get_preds_tta(learn, valid=False, flip=True)
    preds_vert = get_preds_tta(learn, valid=False, vert=True)
    preds_flip_vert = get_preds_tta(learn, valid=False, flip=Tru
e, vert=True)
    preds_normal = preds_normal[:, normal.byte()]
    preds_flip = preds_flip[:, flip.byte()]
    preds_vert = preds_vert[:, vert.byte()]
    preds_flip_vert = preds_flip_vert[:, flip_vert.byte()]
    preds_all = (preds_normal + preds_flip + preds_vert + preds_
flip_vert)/4

    classes = [c for c in learn.data.classes if '_flip' not in c
 and '_vert' not in c]
```

```
    assert len(classes) == 80

    for i, v in enumerate(classes):
        test_df[v] = preds_all[:, i]
    test_df.to_csv(working_path/csv_name, index=False)
    print(f'\n\nPredictions saved to {working_path/csv_name}!')


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    arg = parser.add_argument
    arg('--path', type=str, default='data')
    arg('--working_path', type=str, default='.')
    arg('--base_dim', type=int, default=128)
    arg('--SZ', type=int, default=128)
    arg('--BS', type=int, default=64)
    arg('--lr', type=float, default=1e-2)
    arg('--n_epochs', type=int, default=80)
    arg('--epoch_size', type=int, default=1000)
    arg('--f2cl', type=int, default=1)
    arg('--fold_number', type=int, default=0)
    arg('--loss_name', type=str, default='BCELoss', choices=['BC
ELoss', 'FocalLoss'])
    arg('--csv_name', type=str, default='submission.csv')
    arg('--model', type=str, default='models.xresnet18')
    arg('--load_weights', type=str, default='')
    arg('--weights_file', type=str, default='stage-1')
    arg('--max_processors', type=int, default=8)
    arg('--force', type=bool, default=False)
    args = parser.parse_args()

    path = Path(args.path)
    model = eval(args.model)
    working_path = Path(args.working_path)

    fold_number = args.fold_number
    if fold_number == -1:
        fold_number = None

    load_weights = args.load_weights
    if load_weights == '':
        load_weights = None

    print('\nStarting run using the following configuration:')
    for arg in vars(args):
        print(f'{arg:14s}: {getattr(args, arg)}')

    main(path=path, model=model, working_path=working_path, base
_dim=args.base_dim, SZ=args.SZ, BS=args.BS,
         lr=args.lr, n_epochs=args.n_epochs, epoch_size=args.epo
ch_size, f2cl=args.f2cl, fold_number=fold_number,
         loss_name=args.loss_name, csv_name=args.csv_name, weigh
ts_file=args.weights_file, max_processors=args.max_processors,
         load_weights=load_weights, force=args.force)
```