```
1 Makefile
2 Dockerfile
3 make_folds.py
4 train_folds.py
5 predict_folds.py
6 ensemble_pipeline.sh
7 kernel_template.py
8 train_stacking.py
9 stacking_val_predict.py
10 stacking_random_search.py
11 stacking_predict.py
12 stacking_kernel_template.py
13 random_search.py
14 make_fol
15 build_kernel.py
====src:
16
    101 after_train_folds.py
     27 blend_kernel_template.py
     74 blend_predict.py
     70 build_kernel.py
     74 corrections.json
      0 data
     43 Dockerfile
      1 empty.txt
     57 ensemble_pipeline.sh
     31 kernel_template.py
     20 LICENSE
     21 Makefile
     27 make_folds.py
   8745 noisy_corrections.json
    151 predict_folds.py
    126 random_search.py
      0 readme_images
    188 README.md
      0 src
     26 stacking_kernel_template.py
    124 stacking_predict.py
    126 stacking_random_search.py
     99 stacking_val_predict.py
    146 train_folds.py
    109 train_stacking.py
  10386 total
======

1 Makefile
```

```
NAME=argus-freesound

.PHONY: all build stop run

all: stop build run

build:
	docker build -t $(NAME) .

stop:
	-docker stop $(NAME)
	-docker rm $(NAME)

run:
	nvidia-docker run --rm -it \
		--net=host \
		--ipc=host \
		-v $(shell pwd):/workdir \
		--name=$(NAME) \
		$(NAME) \
		bash
======
2 Dockerfile
2 Dockerfile

FROM nvidia/cuda:10.0-cudnn7-devel-ubuntu18.04

RUN apt-get update &&\
    apt-get -y install build-essential yasm nasm cmake
unzip git wget \
    sysstat libtcmalloc-minimal4 pkgconf autoconf
libtool \
    python3 python3-pip python3-dev python3-setuptools \
    libsm6 libxext6 libxrender1 &&\
    ln -s /usr/bin/python3 /usr/bin/python &&\
    ln -s /usr/bin/pip3 /usr/bin/pip &&\
    apt-get clean &&\
    apt-get autoremove &&\
    rm -rf /var/lib/apt/lists/* &&\
    rm -rf /var/cache/apt/archives/*

RUN pip3 install --no-cache-dir numpy==1.16.2

# Install PyTorch
RUN pip3 install https://download.pytorch.org/whl/cu100/
```

```
torch-1.0.1.post2-cp36-cp36m-linux_x86_64.whl &&\
    pip3 install torchvision==0.2.2 &&\
    rm -rf ~/.cache/pip

# Install python ML packages
RUN pip3 install --no-cache-dir \
    opencv-python==3.4.2.17 \
    scipy==1.2.1 \
    matplotlib==3.0.3 \
    pandas==0.24.1 \
    jupyter==1.0.0 \
    scikit-learn==0.20.2 \
    scikit-image==0.14.2 \
    librosa==0.6.3 \
    pytorch-argus==0.0.8

RUN git clone https://github.com/NVIDIA/apex &&\
    cd apex &&\
    git checkout 855808f &&\
    pip install -v --no-cache-dir --global-option="--
cpp_ext" --global-option="--cuda_ext" . &&\
    cd .. && rm -rf apex

ENV PYTHONPATH $PYTHONPATH:/workdir
ENV TORCH_HOME=/workdir/data/.torch

WORKDIR /workdir
======
3 make_folds.py
import random
import numpy as np
import pandas as pd

from sklearn.model_selection import KFold

from src import config


if __name__ == '__main__':
    random_state = 42

    random.seed(random_state)
    np.random.seed(random_state)

    train_curated_df =
pd.read_csv(config.train_curated_csv_path)
```

```
    train_curated_df['fold'] = -1
    file_paths = train_curated_df.fname.apply(lambda x:
config.train_curated_dir / x)
    train_curated_df['file_path'] = file_paths

    kf = KFold(n_splits=config.n_folds,
random_state=random_state, shuffle=True)

    for fold, (_, val_index) in
enumerate(kf.split(train_curated_df)):
        train_curated_df.iloc[val_index, 2] = fold

    train_curated_df.to_csv(config.train_folds_path,
index=False)
    print(f"Train folds saved to
'{config.train_folds_path}'")
======
4 train_folds.py
import json
import argparse

from argus.callbacks import MonitorCheckpoint, \
    EarlyStopping, LoggingToFile, ReduceLROnPlateau

from torch.utils.data import DataLoader

from src.datasets import FreesoundDataset,
FreesoundNoisyDataset, RandomDataset
from src.datasets import get_corrected_noisy_data,
FreesoundCorrectedNoisyDataset
from src.mixers import RandomMixer, AddMixer,
SigmoidConcatMixer, UseMixerWithProb
from src.transforms import get_transforms
from src.argus_models import FreesoundModel
from src.utils import load_noisy_data, load_folds_data
from src import config


parser = argparse.ArgumentParser()
parser.add_argument('--experiment', required=True,
type=str)
args = parser.parse_args()

BATCH_SIZE = 128
CROP_SIZE = 256
DATASET_SIZE = 128 * 256
```

```
NOISY_PROB = 0.01
CORR_NOISY_PROB = 0.42
MIXER_PROB = 0.8
WRAP_PAD_PROB = 0.5
CORRECTIONS = True
if config.kernel:
    NUM_WORKERS = 2
else:
    NUM_WORKERS = 8
SAVE_DIR = config.experiments_dir / args.experiment
PARAMS = {
    'nn_module': ('AuxSkipAttention', {
        'num_classes': len(config.classes),
        'base_size': 64,
        'dropout': 0.4,
        'ratio': 16,
        'kernel_size': 7,
        'last_filters': 8,
        'last_fc': 4
    }),
    'loss': ('OnlyNoisyLSoftLoss', {
        'beta': 0.7,
        'noisy_weight': 0.5,
        'curated_weight': 0.5
    }),
    'optimizer': ('Adam', {'lr': 0.0009}),
    'device': 'cuda',
    'aux': {
        'weights': [1.0, 0.4, 0.2, 0.1]
    },
    'amp': {
        'opt_level': 'O2',
        'keep_batchnorm_fp32': True,
        'loss_scale': "dynamic"
    }
}


def train_fold(save_dir, train_folds, val_folds,
               folds_data, noisy_data,
corrected_noisy_data):
    train_transfrom = get_transforms(train=True,
                                     size=CROP_SIZE,

wrap_pad_prob=WRAP_PAD_PROB,
                                     resize_scale=(0.8,
1.0),
                                            resize_ratio=(1.7,
2.3),
                                            resize_prob=0.33,
                                            spec_num_mask=2,

spec_freq_masking=0.15,

spec_time_masking=0.20,
                                            spec_prob=0.5)

    mixer = RandomMixer([
        SigmoidConcatMixer(sigmoid_range=(3, 12)),
        AddMixer(alpha_dist='uniform')
    ], p=[0.6, 0.4])
    mixer = UseMixerWithProb(mixer, prob=MIXER_PROB)

    curated_dataset = FreesoundDataset(folds_data,
train_folds,

transform=train_transfrom,
                                              mixer=mixer)
    noisy_dataset = FreesoundNoisyDataset(noisy_data,

transform=train_transfrom,
                                                 mixer=mixer)
    corr_noisy_dataset =
FreesoundCorrectedNoisyDataset(corrected_noisy_data,

transform=train_transfrom,

mixer=mixer)
    dataset_probs = [NOISY_PROB, CORR_NOISY_PROB, 1 -
NOISY_PROB - CORR_NOISY_PROB]
    print("Dataset probs", dataset_probs)
    print("Dataset lens", len(noisy_dataset),
len(corr_noisy_dataset), len(curated_dataset))
    train_dataset = RandomDataset([noisy_dataset,
corr_noisy_dataset, curated_dataset],
                                  p=dataset_probs,
                                  size=DATASET_SIZE)

    val_dataset = FreesoundDataset(folds_data, val_folds,
                                   get_transforms(False,
CROP_SIZE))
    train_loader = DataLoader(train_dataset,
batch_size=BATCH_SIZE,
                              shuffle=True,
drop_last=True,
                              num_workers=NUM_WORKERS)
    val_loader = DataLoader(val_dataset,
batch_size=BATCH_SIZE * 2,
                            shuffle=False,
num_workers=NUM_WORKERS)

    model = FreesoundModel(PARAMS)

    callbacks = [
        MonitorCheckpoint(save_dir,
monitor='val_lwlrap', max_saves=1),
        ReduceLROnPlateau(monitor='val_lwlrap',
patience=6, factor=0.6, min_lr=1e-8),
        EarlyStopping(monitor='val_lwlrap', patience=18),
        LoggingToFile(save_dir / 'log.txt'),
    ]

    model.fit(train_loader,
              val_loader=val_loader,
              max_epochs=700,
              callbacks=callbacks,
              metrics=['multi_accuracy', 'lwlrap'])


if __name__ == "__main__":
    if not SAVE_DIR.exists():
        SAVE_DIR.mkdir(parents=True, exist_ok=True)
    else:
        print(f"Folder {SAVE_DIR} already exists.")

    with open(SAVE_DIR / 'source.py', 'w') as outfile:
        outfile.write(open(__file__).read())

    print("Model params", PARAMS)
    with open(SAVE_DIR / 'params.json', 'w') as outfile:
        json.dump(PARAMS, outfile)

    folds_data =
load_folds_data(use_corrections=CORRECTIONS)
    noisy_data = load_noisy_data()
    corrected_noisy_data = get_corrected_noisy_data()

    for fold in config.folds:
        val_folds = [fold]
        train_folds = list(set(config.folds) -
set(val_folds))
        save_fold_dir = SAVE_DIR / f'fold_{fold}'
        print(f"Val folds: {val_folds}, Train folds:
{train_folds}")
        print(f"Fold save dir {save_fold_dir}")
        train_fold(save_fold_dir, train_folds, val_folds,
                   folds_data, noisy_data,
corrected_noisy_data)
======
5 predict_folds.py
import json
import argparse
import numpy as np
import pandas as pd

from src.predictor import Predictor
from src.audio import read_as_melspectrogram
from src.transforms import get_transforms
from src.metrics import LwlrapBase
from src.utils import get_best_model_path,
gmean_preds_blend
from src.datasets import get_test_data
from src import config


parser = argparse.ArgumentParser()
parser.add_argument('--experiment', required=True,
type=str)
args = parser.parse_args()


EXPERIMENT_DIR = config.experiments_dir / args.experiment
PREDICTION_DIR = config.predictions_dir / args.experiment
DEVICE = 'cuda'
CROP_SIZE = 256
BATCH_SIZE = 16


def pred_val_fold(predictor, fold):
    fold_prediction_dir = PREDICTION_DIR /
f'fold_{fold}' / 'val'
    fold_prediction_dir.mkdir(parents=True,
exist_ok=True)
```

```
    train_folds_df = pd.read_csv(config.train_folds_path)
    train_folds_df = train_folds_df[train_folds_df.fold
== fold]

    fname_lst = []
    pred_lst = []
    for i, row in train_folds_df.iterrows():
        image = read_as_melspectrogram(row.file_path)
        pred = predictor.predict(image)

        pred_path = fold_prediction_dir /
f'{row.fname}.npy'
        np.save(pred_path, pred)

        pred = pred.mean(axis=0)
        pred_lst.append(pred)
        fname_lst.append(row.fname)

    preds = np.stack(pred_lst, axis=0)
    probs_df = pd.DataFrame(data=preds,
                            index=fname_lst,
                            columns=config.classes)
    probs_df.index.name = 'fname'
    probs_df.to_csv(fold_prediction_dir / 'probs.csv')


def pred_test_fold(predictor, fold, test_data):
    fold_prediction_dir = PREDICTION_DIR /
f'fold_{fold}' / 'test'
    fold_prediction_dir.mkdir(parents=True,
exist_ok=True)

    fname_lst, images_lst = test_data
    pred_lst = []
    for fname, image in zip(fname_lst, images_lst):
        pred = predictor.predict(image)

        pred_path = fold_prediction_dir / f'{fname}.npy'
        np.save(pred_path, pred)

        pred = pred.mean(axis=0)
        pred_lst.append(pred)

    preds = np.stack(pred_lst, axis=0)
    subm_df = pd.DataFrame(data=preds,
                           index=fname_lst,
                           columns=config.classes)
    subm_df.index.name = 'fname'
    subm_df.to_csv(fold_prediction_dir / 'probs.csv')


def blend_test_predictions():
    probs_df_lst = []
    for fold in config.folds:
        fold_probs_path = PREDICTION_DIR /
f'fold_{fold}' / 'test' / 'probs.csv'
        probs_df = pd.read_csv(fold_probs_path)
        probs_df.set_index('fname', inplace=True)
        probs_df_lst.append(probs_df)

    blend_df = gmean_preds_blend(probs_df_lst)

    if config.kernel:
        blend_df.to_csv('submission.csv')
    else:
        blend_df.to_csv(PREDICTION_DIR / 'probs.csv')


def calc_lwlrap_on_val():
    probs_df_lst = []
    for fold in config.folds:
        fold_probs_path = PREDICTION_DIR /
f'fold_{fold}' / 'val' / 'probs.csv'
        probs_df = pd.read_csv(fold_probs_path)
        probs_df.set_index('fname', inplace=True)
        probs_df_lst.append(probs_df)

    probs_df = pd.concat(probs_df_lst, axis=0)
    train_curated_df =
pd.read_csv(config.train_curated_csv_path)

    lwlrap = LwlrapBase(config.classes)
    for i, row in train_curated_df.iterrows():
        target = np.zeros(len(config.classes))
        for label in row.labels.split(','):
            target[config.class2index[label]] = 1.

        pred = probs_df.loc[row.fname].values
        lwlrap.accumulate(target[np.newaxis],
pred[np.newaxis])

    result = {
        'overall_lwlrap': lwlrap.overall_lwlrap(),
        'per_class_lwlrap': {cls: lwl for cls, lwl in
zip(config.classes,

lwlrap.per_class_lwlrap())}
    }
    print(result)
    with open(PREDICTION_DIR / 'val_lwlrap.json', 'w')
as file:
        json.dump(result, file, indent=2)


if __name__ == "__main__":
    transforms = get_transforms(False, CROP_SIZE)
    test_data = get_test_data()

    for fold in config.folds:
        print("Predict fold", fold)
        fold_dir = EXPERIMENT_DIR / f'fold_{fold}'
        model_path = get_best_model_path(fold_dir)
        print("Model path", model_path)
        predictor = Predictor(model_path, transforms,
                              BATCH_SIZE,
                              (config.audio.n_mels,
CROP_SIZE),
                              (config.audio.n_mels,
CROP_SIZE//4),
                              device=DEVICE)

        if not config.kernel:
            print("Val predict")
            pred_val_fold(predictor, fold)

        print("Test predict")
        pred_test_fold(predictor, fold, test_data)

    print("Blend folds predictions")
    blend_test_predictions()

    if not config.kernel:
        print("Calculate lwlrap metric on cv")
        calc_lwlrap_on_val()
======
6 ensemble_pipeline.py
#!/usr/bin/env bash
set -e
```

```
NAME="argus-freesound"
DOCKER_OPTIONS="--rm -it --ipc=host -v $(pwd):/workdir --
name=${NAME} ${NAME}"

git checkout master
docker build -t ${NAME} .

# Build kernel
git checkout ddbe02ae88b6bd05c1b9726d2fd30c38854be4fd
nvidia-docker run ${DOCKER_OPTIONS} python
build_kernel.py

# Make folds split
nvidia-docker run ${DOCKER_OPTIONS} python make_folds.py

# Experiment auxiliary_016
git checkout 31156c79e470ffacc494ba846aef3bd80faf0d10
nvidia-docker run ${DOCKER_OPTIONS} python
train_folds.py --experiment auxiliary_016

# Experiment auxiliary_019
git checkout 9639288b9240e7e45db497feb7593f05a4f463d1
nvidia-docker run ${DOCKER_OPTIONS} python
train_folds.py --experiment auxiliary_019

# Experiment corr_noisy_003
git checkout 1fb2eea443d99df4538420fa42daf098c94322c2
nvidia-docker run ${DOCKER_OPTIONS} python
train_folds.py --experiment corr_noisy_003

# Experiment corr_noisy_004
git checkout db945ac11df559e0e1c0a2be464faf46122f1bef
nvidia-docker run ${DOCKER_OPTIONS} python
train_folds.py --experiment corr_noisy_004

# Experiment corr_noisy_007
git checkout bdb9150146ad8d500b4e19fa6b9fe98111fb28b0
nvidia-docker run ${DOCKER_OPTIONS} python
train_folds.py --experiment corr_noisy_007

# Experiment corrections_002
git checkout 05a7aee7c50148677735531bdddf32902b468bea
nvidia-docker run ${DOCKER_OPTIONS} python
train_folds.py --experiment corrections_002

# Experiment corrections_003
```

```
git checkout 24a4f20ffc284d22b38bbabfe510ed194f62e496
nvidia-docker run ${DOCKER_OPTIONS} python
train_folds.py --experiment corrections_003


# Experiment stacking_008_fcnet_43040
git checkout 1e1c265fc6e45c103d8d741c1bdcc5959f71348d
nvidia-docker run ${DOCKER_OPTIONS} python
train_stacking.py

# Stacking train stacking_008_fcnet_45041
git checkout bc48f8a17ac4452ee3f2a3d18fd7caa31f812b27
nvidia-docker run ${DOCKER_OPTIONS} python
train_stacking.py

# Stacking train stacking_008_fcnet_50013
git checkout 493908aeaff4b0e1df8298003b10af1cf56e6b3c
nvidia-docker run ${DOCKER_OPTIONS} python
train_stacking.py

git checkout master
======
7 kernel_template.py

import gzip
import base64
import os
from pathlib import Path
from typing import Dict

EXPERIMENT_NAME = 'corr_noisy_007'
KERNEL_MODE = "predict"  # "train" or "predict"

# this is base64 encoded source code
file_data: Dict = {file_data}


for path, encoded in file_data.items():
    print(path)
    path = Path(path)
    path.parent.mkdir(parents=True, exist_ok=True)

path.write_bytes(gzip.decompress(base64.b64decode(encoded)))


def run(command):
    os.system('export PYTHONPATH=${PYTHONPATH}:/kaggle/
working && '
              f'export MODE={KERNEL_MODE} && ' + command)


run('python make_folds.py')
if KERNEL_MODE == "train":
    run(f'python train_folds.py --experiment
{EXPERIMENT_NAME}')
else:
    run(f'python predict_folds.py --experiment
{EXPERIMENT_NAME}')
run('rm -rf argus src')
======
8train_stacking.py
import json

from argus.callbacks import MonitorCheckpoint, \
    EarlyStopping, LoggingToFile, ReduceLROnPlateau

from torch.utils.data import DataLoader

from src.stacking.datasets import get_out_of_folds_data,
StackingDataset
from src.stacking.transforms import get_transforms
from src.stacking.argus_models import StackingModel
from src import config


STACKING_EXPERIMENT = "stacking_008_fcnet_50013"

EXPERIMENTS = [
    'auxiliary_016',
    'auxiliary_019',
    'corr_noisy_003',
    'corr_noisy_004',
    'corr_noisy_007',
    'corrections_002',
    'corrections_003'
]
RS_PARAMS = {"base_size": 512, "reduction_scale": 1,
"p_dropout": 0.1662788540244386, "lr":
2.5814932060476834e-05,
             "patience": 7, "factor":
0.5537460438294733, "batch_size": 128}
BATCH_SIZE = RS_PARAMS['batch_size']
```

```
DATASET_SIZE = 128 * 256
CORRECTIONS = True
if config.kernel:
    NUM_WORKERS = 2
else:
    NUM_WORKERS = 8
SAVE_DIR = config.experiments_dir / STACKING_EXPERIMENT
PARAMS = {
    'nn_module': ('FCNet', {
        'in_channels': len(config.classes) *
len(EXPERIMENTS),
        'num_classes': len(config.classes),
        'base_size': RS_PARAMS['base_size'],
        'reduction_scale': RS_PARAMS['reduction_scale'],
        'p_dropout': RS_PARAMS['p_dropout']
    }),
    'loss': 'BCEWithLogitsLoss',
    'optimizer': ('Adam', {'lr': RS_PARAMS['lr']}),
    'device': 'cuda',
}


def train_fold(save_dir, train_folds, val_folds,
folds_data):
    train_dataset = StackingDataset(folds_data,
train_folds,
                                    get_transforms(True),
                                    DATASET_SIZE)
    val_dataset = StackingDataset(folds_data, val_folds,
                                  get_transforms(False))

    train_loader = DataLoader(train_dataset,
batch_size=BATCH_SIZE,
                              shuffle=True,
drop_last=True,
                              num_workers=NUM_WORKERS)
    val_loader = DataLoader(val_dataset,
batch_size=BATCH_SIZE * 2,
                            shuffle=False,
num_workers=NUM_WORKERS)

    model = StackingModel(PARAMS)

    callbacks = [
        MonitorCheckpoint(save_dir,
monitor='val_lwlrap', max_saves=1),
        ReduceLROnPlateau(monitor='val_lwlrap',
                          patience=RS_PARAMS['patience'],
                          factor=RS_PARAMS['factor'],
                          min_lr=1e-8),
        EarlyStopping(monitor='val_lwlrap', patience=30),
        LoggingToFile(save_dir / 'log.txt'),
    ]

    model.fit(train_loader,
              val_loader=val_loader,
              max_epochs=700,
              callbacks=callbacks,
              metrics=['multi_accuracy', 'lwlrap'])


if __name__ == "__main__":
    if not SAVE_DIR.exists():
        SAVE_DIR.mkdir(parents=True, exist_ok=True)
    else:
        print(f"Folder {SAVE_DIR} already exists.")

    with open(SAVE_DIR / 'source.py', 'w') as outfile:
        outfile.write(open(__file__).read())

    print("Model params", PARAMS)
    with open(SAVE_DIR / 'params.json', 'w') as outfile:
        json.dump(PARAMS, outfile)

    if CORRECTIONS:
        with open(config.corrections_json_path) as file:
            corrections = json.load(file)
        print("Corrections:", corrections)
    else:
        corrections = None

    folds_data = get_out_of_folds_data(EXPERIMENTS,
corrections)

    for fold in config.folds:
        val_folds = [fold]
        train_folds = list(set(config.folds) -
set(val_folds))
        save_fold_dir = SAVE_DIR / f'fold_{fold}'
        print(f"Val folds: {val_folds}, Train folds:
{train_folds}")
        print(f"Fold save dir {save_fold_dir}")
        train_fold(save_fold_dir, train_folds,
val_folds, folds_data)
======
9 stacking_val_predict.py

import json
import numpy as np
import pandas as pd

from src.stacking.datasets import load_fname_probs
from src.stacking.predictor import StackPredictor
from src.metrics import LwlrapBase
from src.utils import get_best_model_path
from src import config


STACKING_EXPERIMENT = "stacking_008_fcnet_50013"

EXPERIMENTS = [
    'auxiliary_016',
    'auxiliary_019',
    'corr_noisy_003',
    'corr_noisy_004',
    'corr_noisy_007',
    'corrections_002',
    'corrections_003'
]

EXPERIMENT_DIR = config.experiments_dir /
STACKING_EXPERIMENT
PREDICTION_DIR = config.predictions_dir /
STACKING_EXPERIMENT
DEVICE = 'cuda'
BATCH_SIZE = 256


def pred_val_fold(predictor, fold):
    fold_prediction_dir = PREDICTION_DIR /
f'fold_{fold}' / 'val'
    fold_prediction_dir.mkdir(parents=True,
exist_ok=True)

    train_folds_df = pd.read_csv(config.train_folds_path)
    train_folds_df = train_folds_df[train_folds_df.fold
== fold]
```

```
    fname_lst = []
    probs_lst = []
    for i, row in train_folds_df.iterrows():
        probs = load_fname_probs(EXPERIMENTS, fold,
row.fname)

        probs_lst.append(probs.mean(axis=0))
        fname_lst.append(row.fname)

    stack_probs = np.stack(probs_lst, axis=0)
    preds = predictor.predict(stack_probs)

    probs_df = pd.DataFrame(data=list(preds),
                            index=fname_lst,
                            columns=config.classes)
    probs_df.index.name = 'fname'
    probs_df.to_csv(fold_prediction_dir / 'probs.csv')


def calc_lwlrap_on_val():
    probs_df_lst = []
    for fold in config.folds:
        fold_probs_path = PREDICTION_DIR /
f'fold_{fold}' / 'val' / 'probs.csv'
        probs_df = pd.read_csv(fold_probs_path)
        probs_df.set_index('fname', inplace=True)
        probs_df_lst.append(probs_df)

    probs_df = pd.concat(probs_df_lst, axis=0)
    train_curated_df =
pd.read_csv(config.train_curated_csv_path)

    lwlrap = LwlrapBase(config.classes)
    for i, row in train_curated_df.iterrows():
        target = np.zeros(len(config.classes))
        for label in row.labels.split(','):
            target[config.class2index[label]] = 1.

        pred = probs_df.loc[row.fname].values
        lwlrap.accumulate(target[np.newaxis],
pred[np.newaxis])

    result = {
        'overall_lwlrap': lwlrap.overall_lwlrap(),
        'per_class_lwlrap': {cls: lwl for cls, lwl in
zip(config.classes,
lwlrap.per_class_lwlrap())}
    }
    print(result)
    with open(PREDICTION_DIR / 'val_lwlrap.json', 'w')
as file:
        json.dump(result, file, indent=2)


if __name__ == "__main__":
    for fold in config.folds:
        print("Predict fold", fold)
        fold_dir = EXPERIMENT_DIR / f'fold_{fold}'
        model_path = get_best_model_path(fold_dir)
        print("Model path", model_path)
        predictor = StackPredictor(model_path,
                                   BATCH_SIZE,
                                   device=DEVICE)

        print("Val predict")
        pred_val_fold(predictor, fold)

    print("Calculate lwlrap metric on cv")
    calc_lwlrap_on_val()
======
10 stacking_random_search.py
10 stacking_random_search.py
10 stacking_random_search.py
10 stacking_random_search.py
10 stacking_random_search.py
10 stacking_random_search.py
10 stacking_random_search.py
10 stacking_random_search.py
10 stacking_random_search.py
10 stacking_random_search.py

import json
import time
import torch
import random
import numpy as np
from pprint import pprint

from argus.callbacks import MonitorCheckpoint, \
    EarlyStopping, LoggingToFile, ReduceLROnPlateau
```

```
from torch.utils.data import DataLoader

from src.stacking.datasets import get_out_of_folds_data,
StackingDataset
from src.stacking.transforms import get_transforms
from src.stacking.argus_models import StackingModel
from src import config

EXPERIMENT_NAME = 'fcnet_stacking_rs_004'
START_FROM = 0
EXPERIMENTS = [
    'auxiliary_007',
    'auxiliary_010',
    'auxiliary_012',
    'auxiliary_014'
]
DATASET_SIZE = 128 * 256
CORRECTIONS = True
if config.kernel:
    NUM_WORKERS = 2
else:
    NUM_WORKERS = 4
SAVE_DIR = config.experiments_dir / EXPERIMENT_NAME


def train_folds(save_dir, folds_data):
    random_params = {
        'base_size': int(np.random.choice([64, 128, 256,
512])),
        'reduction_scale': int(np.random.choice([2, 4,
8, 16])),
        'p_dropout': float(np.random.uniform(0.0, 0.5)),
        'lr': float(np.random.uniform(0.0001, 0.00001)),
        'patience': int(np.random.randint(3, 12)),
        'factor': float(np.random.uniform(0.5, 0.8)),
        'batch_size': int(np.random.choice([32, 64,
128])),
    }
    pprint(random_params)

    save_dir.mkdir(parents=True, exist_ok=True)
    with open(save_dir / 'random_params.json', 'w') as
outfile:
        json.dump(random_params, outfile)

    params = {
        'nn_module': ('FCNet', {
            'in_channels': len(config.classes) *
len(EXPERIMENTS),
            'num_classes': len(config.classes),
            'base_size': random_params['base_size'],
            'reduction_scale':
random_params['reduction_scale'],
            'p_dropout': random_params['p_dropout']
        }),
        'loss': 'BCEWithLogitsLoss',
        'optimizer': ('Adam', {'lr':
random_params['lr']}),
        'device': 'cuda',
    }

    for fold in config.folds:
        val_folds = [fold]
        train_folds = list(set(config.folds) -
set(val_folds))
        save_fold_dir = save_dir / f'fold_{fold}'
        print(f"Val folds: {val_folds}, Train folds:
{train_folds}")
        print(f"Fold save dir {save_fold_dir}")

        train_dataset = StackingDataset(folds_data,
train_folds,

get_transforms(True),
                                        DATASET_SIZE)
        val_dataset = StackingDataset(folds_data,
val_folds,

get_transforms(False))

        train_loader = DataLoader(train_dataset,

batch_size=random_params['batch_size'],
                                  shuffle=True,
drop_last=True,

num_workers=NUM_WORKERS)
        val_loader = DataLoader(val_dataset,

batch_size=random_params['batch_size'] * 2,
                                shuffle=False,
num_workers=NUM_WORKERS)
```

```
        model = StackingModel(params)

        callbacks = [
            MonitorCheckpoint(save_fold_dir,
monitor='val_lwlrap', max_saves=1),
            ReduceLROnPlateau(monitor='val_lwlrap',

patience=random_params['patience'],

factor=random_params['factor'],
                              min_lr=1e-8),
            EarlyStopping(monitor='val_lwlrap',
patience=20),
            LoggingToFile(save_fold_dir / 'log.txt'),
        ]

        model.fit(train_loader,
                  val_loader=val_loader,
                  max_epochs=300,
                  callbacks=callbacks,
                  metrics=['multi_accuracy', 'lwlrap'])


if __name__ == "__main__":
    SAVE_DIR.mkdir(parents=True, exist_ok=True)
    with open(SAVE_DIR / 'source.py', 'w') as outfile:
        outfile.write(open(__file__).read())

    if CORRECTIONS:
        with open(config.corrections_json_path) as file:
            corrections = json.load(file)
        print("Corrections:", corrections)
    else:
        corrections = None

    folds_data = get_out_of_folds_data(EXPERIMENTS,
corrections)

    for num in range(START_FROM, 10000):
        np.random.seed(num)
        random.seed(num)

        save_dir = SAVE_DIR / f'{num:04}'
        train_folds(save_dir, folds_data)
        time.sleep(5.0)
        torch.cuda.empty_cache()
        time.sleep(5.0)
======
11 stacking_predict.py
import numpy as np
import pandas as pd
from scipy.stats.mstats import gmean

from src.predictor import Predictor
from src.transforms import get_transforms
from src.utils import get_best_model_path
from src.datasets import get_test_data
from src import config

from src.stacking.predictor import StackPredictor


NAME = "stacking_008"

EXPERIMENTS = [
    'auxiliary_016',
    'auxiliary_019',
    'corr_noisy_003',
    'corr_noisy_004',
    'corr_noisy_007',
    'corrections_002',
    'corrections_003'
]

STACKING_EXPERIMENTS = [
    'stacking_008_fcnet_43040',
    'stacking_008_fcnet_45041',
    'stacking_008_fcnet_50013'
]

DEVICE = 'cuda'
CROP_SIZE = 256
BATCH_SIZE = 16
STACK_BATCH_SIZE = 256
TILE_STEP = 2


def pred_test(predictor, images_lst):
    pred_lst = []
    for image in images_lst:
        pred = predictor.predict(image)
        pred = pred.mean(axis=0)
        pred_lst.append(pred)

    preds = np.stack(pred_lst, axis=0)
    return preds


def experiment_pred(experiment_dir, images_lst):
    print(f"Start predict: {experiment_dir}")
    transforms = get_transforms(False, CROP_SIZE)

    pred_lst = []
    for fold in config.folds:
        print("Predict fold", fold)
        fold_dir = experiment_dir / f'fold_{fold}'
        model_path = get_best_model_path(fold_dir)
        print("Model path", model_path)
        predictor = Predictor(model_path, transforms,
                              BATCH_SIZE,
                              (config.audio.n_mels,
CROP_SIZE),
                              (config.audio.n_mels,
CROP_SIZE//TILE_STEP),
                              device=DEVICE)

        pred = pred_test(predictor, images_lst)
        pred_lst.append(pred)

    preds = gmean(pred_lst, axis=0)
    return preds


def stacking_pred(experiment_dir, stack_probs):
    print(f"Start predict: {experiment_dir}")

    pred_lst = []
    for fold in config.folds:
        print("Predict fold", fold)
        fold_dir = experiment_dir / f'fold_{fold}'
        model_path = get_best_model_path(fold_dir)
        print("Model path", model_path)
        predictor = StackPredictor(model_path,
STACK_BATCH_SIZE,
                                   device=DEVICE)
        pred = predictor.predict(stack_probs)
        pred_lst.append(pred)

    preds = gmean(pred_lst, axis=0)
    return preds


if __name__ == "__main__":
    print("Name", NAME)
    print("Experiments", EXPERIMENTS)
    print("Stacking experiments", STACKING_EXPERIMENTS)
    print("Device", DEVICE)
    print("Crop size", CROP_SIZE)
    print("Batch size", BATCH_SIZE)
    print("Stacking batch size", STACK_BATCH_SIZE)
    print("Tile step", TILE_STEP)

    fname_lst, images_lst = get_test_data()

    exp_pred_lst = []
    for experiment in EXPERIMENTS:
        experiment_dir = config.experiments_dir / 
experiment
        exp_pred = experiment_pred(experiment_dir, 
images_lst)
        exp_pred_lst.append(exp_pred)

    stack_probs = np.concatenate(exp_pred_lst, axis=1)

    stack_pred_lst = []
    for experiment in STACKING_EXPERIMENTS:
        experiment_dir = config.experiments_dir / 
experiment
        stack_pred = stacking_pred(experiment_dir, 
stack_probs)
        stack_pred_lst.append(stack_pred)

    stack_pred = gmean(exp_pred_lst + stack_pred_lst, 
axis=0)

    stack_pred_df = pd.DataFrame(data=stack_pred,
                                 index=fname_lst,
                                 columns=config.classes)
    stack_pred_df.index.name = 'fname'
    stack_pred_df.to_csv('submission.csv')
======
12 stacking_kernel_template.py
```

```
import gzip
import base64
import os
from pathlib import Path
from typing import Dict

KERNEL_MODE = "predict"

# this is base64 encoded source code
file_data: Dict = {file_data}


for path, encoded in file_data.items():
    print(path)
    path = Path(path)
    path.parent.mkdir(parents=True, exist_ok=True)

path.write_bytes(gzip.decompress(base64.b64decode(encoded)))


def run(command):
    os.system('export PYTHONPATH=${PYTHONPATH}:/kaggle/
working && '
              f'export MODE={KERNEL_MODE} && ' + command)


run('python stacking_predict.py')
run('rm -rf argus src')
======
13random_search.py

import torch
import numpy as np
import random
import json
import time
from pprint import pprint

from argus.callbacks import MonitorCheckpoint, \
    EarlyStopping, LoggingToFile, ReduceLROnPlateau

from torch.utils.data import DataLoader

from src.datasets import FreesoundDataset,
CombinedDataset, FreesoundNoisyDataset
from src.transforms import get_transforms
from src.argus_models import FreesoundModel
from src.utils import load_folds_data, load_noisy_data
from src import config


EXPERIMENT_NAME = 'noisy_lsoft_rs_002'
VAL_FOLDS = [0]
TRAIN_FOLDS = [1, 2, 3, 4]
BATCH_SIZE = 128
CROP_SIZE = 128
DATASET_SIZE = 128 * 256
if config.kernel:
    NUM_WORKERS = 2
else:
    NUM_WORKERS = 8
SAVE_DIR = config.experiments_dir / EXPERIMENT_NAME
START_FROM = 0


def train_experiment(folds_data, noisy_data, num):
    experiment_dir = SAVE_DIR / f'{num:04}'
    np.random.seed(num)
    random.seed(num)

    random_params = {
        'p_dropout': float(np.random.uniform(0.1, 0.3)),
        'batch_size': int(np.random.choice([128])),
        'lr': float(np.random.choice([0.001, 0.0006,
0.0003])),
        'add_prob': float(np.random.uniform(0.0, 1.0)),
        'noisy_prob': float(np.random.uniform(0.0, 1.0)),
        'lsoft_beta': float(np.random.uniform(0.2, 0.8)),
        'noisy_weight': float(np.random.uniform(0.3,
0.7)),
        'patience': int(np.random.randint(2, 10)),
        'factor': float(np.random.uniform(0.5, 0.8))
    }
    pprint(random_params)

    params = {
        'nn_module': ('SimpleKaggle', {
            'num_classes': len(config.classes),
            'dropout': random_params['p_dropout'],
            'base_size': 64
        }),
        'loss': ('OnlyNoisyLSoftLoss', {
            'beta': random_params['lsoft_beta'],
            'noisy_weight':
random_params['noisy_weight'],
            'curated_weight': 1 -
random_params['noisy_weight']
        }),
        'optimizer': ('Adam', {'lr':
random_params['lr']}),
        'device': 'cuda',
        'amp': {
            'opt_level': 'O2',
            'keep_batchnorm_fp32': True,
            'loss_scale': "dynamic"
        }
    }
    pprint(params)
    try:
        train_transfrom = get_transforms(True, CROP_SIZE)
        curated_dataset = FreesoundDataset(folds_data,
TRAIN_FOLDS,

transform=train_transfrom,

add_prob=random_params['add_prob'])
        noisy_dataset = FreesoundNoisyDataset(noisy_data,

transform=train_transfrom)
        train_dataset = CombinedDataset(noisy_dataset,
curated_dataset,

noisy_prob=random_params['noisy_prob'],

size=DATASET_SIZE)

        val_dataset = FreesoundDataset(folds_data,
VAL_FOLDS,

get_transforms(False, CROP_SIZE))
        train_loader = DataLoader(train_dataset,
batch_size=random_params['batch_size'],
                                  shuffle=True,
drop_last=True,

num_workers=NUM_WORKERS)
        val_loader = DataLoader(val_dataset,
batch_size=random_params['batch_size'] * 2,
                                shuffle=False,
num_workers=NUM_WORKERS)

        model = FreesoundModel(params)

        callbacks = [
            MonitorCheckpoint(experiment_dir,
monitor='val_lwlrap', max_saves=1),
            ReduceLROnPlateau(monitor='val_lwlrap',

patience=random_params['patience'],

factor=random_params['factor'],
                              min_lr=1e-8),
            EarlyStopping(monitor='val_lwlrap',
patience=20),
            LoggingToFile(experiment_dir / 'log.txt'),
        ]

        with open(experiment_dir / 'random_params.json',
'w') as outfile:
            json.dump(random_params, outfile)

        model.fit(train_loader,
                  val_loader=val_loader,
                  max_epochs=100,
                  callbacks=callbacks,
                  metrics=['multi_accuracy', 'lwlrap'])
    except KeyboardInterrupt as e:
        raise e
    except BaseException as e:
        print(f"Exception '{e}' with random params
'{random_params}'")


if __name__ == "__main__":
    print("Start load train data")
    noisy_data = load_noisy_data()
    folds_data = load_folds_data()

    for i in range(START_FROM, 10000):
        train_experiment(folds_data, noisy_data, i)
        time.sleep(5.0)
        torch.cuda.empty_cache()
        time.sleep(5.0)
======
14 make_fol

import random
import numpy as np
import pandas as pd

from sklearn.model_selection import KFold

from src import config


if __name__ == '__main__':
    random_state = 42

    random.seed(random_state)
    np.random.seed(random_state)

    train_curated_df = 
pd.read_csv(config.train_curated_csv_path)
    train_curated_df['fold'] = -1
    file_paths = train_curated_df.fname.apply(lambda x: 
config.train_curated_dir / x)
    train_curated_df['file_path'] = file_paths

    kf = KFold(n_splits=config.n_folds, 
random_state=random_state, shuffle=True)

    for fold, (_, val_index) in 
enumerate(kf.split(train_curated_df)):
        train_curated_df.iloc[val_index, 2] = fold

    train_curated_df.to_csv(config.train_folds_path, 
index=False)
    print(f"Train folds saved to 
'{config.train_folds_path}'")
======
15 build_kernel.py

#!/usr/bin/env python3
# Kaggle script build system template: https://
github.com/lopuhin/kaggle-script-template
import os
import base64
import gzip
from pathlib import Path
```

```
IGNORE_LIST = ["data", "build"]

PACKAGES = [
    'https://github.com/lRomul/argus.git'
]


def encode_file(path: Path) -> str:
    compressed = gzip.compress(path.read_bytes(),
compresslevel=9)
    return base64.b64encode(compressed).decode('utf-8')


def check_ignore(path: Path, ignore_list):
    if not path.is_file():
        return False
    for ignore in ignore_list:
        if str(path).startswith(ignore):
            return False
    return True


def clone_package(git_url):
    name = Path(git_url).stem
    os.system('mkdir -p tmp')
    os.system(f'rm -rf tmp/{name}')
    os.system(f'cd tmp && git clone {git_url}')
    os.system(f'cp -R tmp/{name}/{name} .')
    os.system(f'rm -rf tmp/{name}')


def build_script(ignore_list, packages,
template_name='kernel_template.py'):
    to_encode = []

    for path in Path('.').glob('**/*.py'):
        if check_ignore(path, ignore_list + packages):
            to_encode.append(path)

    for package in packages:
        clone_package(package)
        package_name = Path(package).stem
        for path in Path(package_name).glob('**/*'):
            if check_ignore(path, ignore_list):
                to_encode.append(path)
    file_data = {str(path): encode_file(path) for path
in to_encode}
    print("Encoded python files:")
    for path in file_data:
        print(path)
    template = Path(template_name).read_text('utf8')
    (Path('kernel') / template_name).write_text(
        template.replace('{file_data}', str(file_data)),
        encoding='utf8')


if __name__ == '__main__':
    os.system('rm -rf kernel && mkdir kernel')
    build_script(IGNORE_LIST, PACKAGES,
                 template_name='kernel_template.py')
    build_script(IGNORE_LIST, PACKAGES,

template_name='blend_kernel_template.py')
    build_script(IGNORE_LIST, PACKAGES,

template_name='stacking_kernel_template.py')
======
```