data_noise_injection

December 2, 2020

1 Noise injection into data

This example presents how the the noise can be injected into any part of the dataset: train, test and validation. There are three types of noise implemented:

- Gaussian
- Impulsive
- Shot

There are a few parameters which indicate how a given noise behaves:

- *pa* - Fraction of noisy pixels, the number of affected samples is calculated by: floor(n_samples * pa).
- *pb* - Fraction of noisy bands. When established the number of samples that undergo noise injection, for each sample the: floor(n_bands * pb) bands are affected.
- *bc* - Boolean indicating whether the indexes of affected bands, are constant for each sample. When set to: False, different bands can be augmented with noise for each pixel.
- *mean* - Gaussian noise parameter, the mean of the normal distribution.
- *std* - Gaussian noise parameter, standard deviation of the normal distribution.
- *pw* - Impulsive noise parameter, ratio of whitened pixels for the affected set of samples.

[1]:
```
import os
import sys
sys.path.append(os.path.dirname(os.getcwd()))
```

[ ]:
```
import os
import shutil
import re

import clize
import mlflow
import tensorflow as tf
from clize.parameters import multi

from scripts import evaluate_model, prepare_data, artifacts_reporter,␣
↪train_model
from ml_intuition.enums import Splits, Experiment
from ml_intuition.data.io import load_processed_h5
from ml_intuition.data.utils import get_mlflow_artifacts_path, parse_train_size
```

```
from ml_intuition.data.loggers import log_params_to_mlflow, log_tags_to_mlflow
```

Specify path to the `.npy` dataset and ground truth, as well as the output path to store all the artifacts.

[3]:
```
DEST_PATH = 'data_noise_injection_results'
DATA_FILE_PATH = os.path.join(os.path.dirname(os.getcwd()), 'datasets/pavia/
 ↪pavia.npy')
GT_FILE_PAT = os.path.join(os.path.dirname(os.getcwd()), 'datasets/pavia/
 ↪pavia_gt.npy')
experiment_dest_path = os.path.join(DEST_PATH, 'experiment_0')
os.makedirs(experiment_dest_path, exist_ok=True)
```

2 Prepare the data

To fit into the the pipeline, the data has to be preprocessed. It is achieved by the `prepare_data.main` function. It accepts a path to a `.npy` file with the original cube as well as the corresponding ground truth. In this example, we randomly extract 250 samples from each class (balanced scenario), use 10% of them as validation set, and extract only spectral information of a pixel. The returned object is a dictionary with three keys: `train`, `test` and `val`. Each of them contains an additional dictionary with `data` and `labels` keys, holding corresponding `numpy.ndarray` objects with the data. For more details about the parameters, refer to the documentation of `prepare_data.main` function (located in `scripts/prepare_data`).

[4]:
```
data = prepare_data.main(data_file_path=DATA_FILE_PATH,
                         ground_truth_path=GT_FILE_PAT,
                         output_path=None,
                         train_size=250,
                         val_size=0.1,
                         stratified=True,
                         background_label=0,
                         channels_idx=2,
                         neighborhood_size=None,
                         save_data=False,
                         seed=0)
```

3 Train the model with nosiy training set

The function `trian_model.train` executed the trainig procedure. In order to inject noise into the training set, provide `noise` with a name of the noise type, `noise_sets` with the set you would like to augment, and `noise_params` with the noise parameters. Trained model will be stored under `experiment_dest_path` folder path.

[5]:
```python
train_model.train(model_name='model_2d',
                    kernel_size=5,
                    n_kernels=200,
                    n_layers=1,
                    dest_path=experiment_dest_path,
                    data=data,
                    sample_size=103,
                    n_classes=9,
                    lr=0.001,
                    batch_size=128,
                    epochs=200,
                    verbose=2,
                    shuffle=True,
                    patience=15,
                    noise=['gaussian'],
                    noise_sets=['train'],
                    noise_params="{\"mean\": 0, \"std\": 1, \"pa\": 0.1, \"pb\":
 ↪ 1}")
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 99, 1, 200)        1200
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 1, 200)        200200
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 1, 200)        200200
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 5, 1, 200)         200200
_________________________________________________________________
flatten (Flatten)            (None, 1000)              0
_________________________________________________________________
dense (Dense)                (None, 200)               200200
_________________________________________________________________
dense_1 (Dense)              (None, 128)               25728
_________________________________________________________________
dense_2 (Dense)              (None, 9)                 1161
=================================================================
Total params: 828,889
Trainable params: 828,889
Non-trainable params: 0
_________________________________________________________________
Train on 2025 samples, validate on 225 samples
Epoch 1/200
 - 2s - loss: 1.8776 - acc: 0.2390 - val_loss: 1.3069 - val_acc: 0.5244
Epoch 2/200
 - 1s - loss: 1.2942 - acc: 0.5096 - val_loss: 1.0817 - val_acc: 0.6489
```

```
Epoch 3/200
 - 1s - loss: 1.0488 - acc: 0.5773 - val_loss: 0.8458 - val_acc: 0.6844
Epoch 4/200
 - 1s - loss: 0.8571 - acc: 0.6622 - val_loss: 0.7634 - val_acc: 0.7289
Epoch 5/200
 - 1s - loss: 0.7798 - acc: 0.6820 - val_loss: 0.6838 - val_acc: 0.6356
Epoch 6/200
 - 1s - loss: 0.6999 - acc: 0.7057 - val_loss: 0.6314 - val_acc: 0.7511
Epoch 7/200
 - 1s - loss: 0.6292 - acc: 0.7160 - val_loss: 0.6730 - val_acc: 0.6622
Epoch 8/200
 - 1s - loss: 0.6421 - acc: 0.7279 - val_loss: 0.6066 - val_acc: 0.7511
Epoch 9/200
 - 1s - loss: 0.5706 - acc: 0.7462 - val_loss: 0.5435 - val_acc: 0.7689
Epoch 10/200
 - 1s - loss: 0.4847 - acc: 0.7921 - val_loss: 0.5501 - val_acc: 0.7956
Epoch 11/200
 - 1s - loss: 0.4860 - acc: 0.7951 - val_loss: 0.6575 - val_acc: 0.7111
Epoch 12/200
 - 1s - loss: 0.4998 - acc: 0.7881 - val_loss: 0.5474 - val_acc: 0.7422
Epoch 13/200
 - 1s - loss: 0.4379 - acc: 0.8143 - val_loss: 0.5357 - val_acc: 0.7378
Epoch 14/200
 - 1s - loss: 0.4815 - acc: 0.7896 - val_loss: 0.5660 - val_acc: 0.7644
Epoch 15/200
 - 1s - loss: 0.4478 - acc: 0.8089 - val_loss: 0.4934 - val_acc: 0.7733
Epoch 16/200
 - 1s - loss: 0.4256 - acc: 0.8089 - val_loss: 0.5716 - val_acc: 0.7422
Epoch 17/200
 - 1s - loss: 0.4027 - acc: 0.8267 - val_loss: 0.5396 - val_acc: 0.7822
Epoch 18/200
 - 1s - loss: 0.3885 - acc: 0.8306 - val_loss: 0.4510 - val_acc: 0.7778
Epoch 19/200
 - 1s - loss: 0.3645 - acc: 0.8425 - val_loss: 0.5031 - val_acc: 0.7822
Epoch 20/200
 - 1s - loss: 0.3714 - acc: 0.8375 - val_loss: 0.4682 - val_acc: 0.7956
Epoch 21/200
 - 1s - loss: 0.3608 - acc: 0.8469 - val_loss: 0.4599 - val_acc: 0.8444
Epoch 22/200
 - 1s - loss: 0.3477 - acc: 0.8523 - val_loss: 0.5151 - val_acc: 0.7733
Epoch 23/200
 - 1s - loss: 0.3597 - acc: 0.8494 - val_loss: 0.5178 - val_acc: 0.7867
Epoch 24/200
 - 1s - loss: 0.3520 - acc: 0.8528 - val_loss: 0.4771 - val_acc: 0.7911
Epoch 25/200
 - 1s - loss: 0.3149 - acc: 0.8647 - val_loss: 0.4903 - val_acc: 0.8222
Epoch 26/200
 - 1s - loss: 0.3499 - acc: 0.8390 - val_loss: 0.4844 - val_acc: 0.8267
```

```
Epoch 27/200
 - 1s - loss: 0.3270 - acc: 0.8578 - val_loss: 0.4659 - val_acc: 0.8222
Epoch 28/200
 - 1s - loss: 0.3169 - acc: 0.8667 - val_loss: 0.4782 - val_acc: 0.8267
Epoch 29/200
 - 1s - loss: 0.3308 - acc: 0.8598 - val_loss: 0.4581 - val_acc: 0.7867
Epoch 30/200
 - 1s - loss: 0.2958 - acc: 0.8770 - val_loss: 0.3715 - val_acc: 0.7956
Epoch 31/200
 - 1s - loss: 0.2990 - acc: 0.8741 - val_loss: 0.4265 - val_acc: 0.8222
Epoch 32/200
 - 1s - loss: 0.2855 - acc: 0.8770 - val_loss: 0.4218 - val_acc: 0.7867
Epoch 33/200
 - 1s - loss: 0.2750 - acc: 0.8849 - val_loss: 0.4143 - val_acc: 0.8133
Epoch 34/200
 - 1s - loss: 0.2695 - acc: 0.8844 - val_loss: 0.3566 - val_acc: 0.8667
Epoch 35/200
 - 1s - loss: 0.2514 - acc: 0.8963 - val_loss: 0.4516 - val_acc: 0.8044
Epoch 36/200
 - 1s - loss: 0.2801 - acc: 0.8879 - val_loss: 0.3582 - val_acc: 0.8267
Epoch 37/200
 - 1s - loss: 0.3015 - acc: 0.8770 - val_loss: 0.3937 - val_acc: 0.8044
Epoch 38/200
 - 1s - loss: 0.2869 - acc: 0.8721 - val_loss: 0.3548 - val_acc: 0.8222
Epoch 39/200
 - 1s - loss: 0.2530 - acc: 0.9027 - val_loss: 0.3383 - val_acc: 0.8444
Epoch 40/200
 - 1s - loss: 0.2437 - acc: 0.8993 - val_loss: 0.4022 - val_acc: 0.8400
Epoch 41/200
 - 1s - loss: 0.2446 - acc: 0.8914 - val_loss: 0.3633 - val_acc: 0.8311
Epoch 42/200
 - 1s - loss: 0.2558 - acc: 0.8953 - val_loss: 0.3692 - val_acc: 0.8311
Epoch 43/200
 - 1s - loss: 0.2503 - acc: 0.8899 - val_loss: 0.3771 - val_acc: 0.8711
Epoch 44/200
 - 1s - loss: 0.2648 - acc: 0.8943 - val_loss: 0.3894 - val_acc: 0.8444
Epoch 45/200
 - 1s - loss: 0.2792 - acc: 0.8894 - val_loss: 0.4229 - val_acc: 0.8178
Epoch 46/200
 - 1s - loss: 0.2484 - acc: 0.8973 - val_loss: 0.4699 - val_acc: 0.8667
Epoch 47/200
 - 1s - loss: 0.2580 - acc: 0.8983 - val_loss: 0.3728 - val_acc: 0.8444
Epoch 48/200
 - 1s - loss: 0.2448 - acc: 0.9032 - val_loss: 0.3569 - val_acc: 0.8800
Epoch 49/200
 - 1s - loss: 0.2584 - acc: 0.8998 - val_loss: 0.4528 - val_acc: 0.8533
Epoch 50/200
 - 1s - loss: 0.2524 - acc: 0.9037 - val_loss: 0.3271 - val_acc: 0.8444
```

```
Epoch 51/200
 - 1s - loss: 0.2131 - acc: 0.9081 - val_loss: 0.3332 - val_acc: 0.8400
Epoch 52/200
 - 1s - loss: 0.2238 - acc: 0.9141 - val_loss: 0.2969 - val_acc: 0.8667
Epoch 53/200
 - 1s - loss: 0.1947 - acc: 0.9170 - val_loss: 0.2855 - val_acc: 0.8622
Epoch 54/200
 - 1s - loss: 0.2272 - acc: 0.9081 - val_loss: 0.3061 - val_acc: 0.8533
Epoch 55/200
 - 1s - loss: 0.2521 - acc: 0.8869 - val_loss: 0.3415 - val_acc: 0.8667
Epoch 56/200
 - 1s - loss: 0.2072 - acc: 0.9195 - val_loss: 0.4266 - val_acc: 0.8356
Epoch 57/200
 - 1s - loss: 0.1892 - acc: 0.9254 - val_loss: 0.3007 - val_acc: 0.8800
Epoch 58/200
 - 1s - loss: 0.2119 - acc: 0.9146 - val_loss: 0.4083 - val_acc: 0.8444
Epoch 59/200
 - 1s - loss: 0.2266 - acc: 0.9091 - val_loss: 0.3553 - val_acc: 0.8756
Epoch 60/200
 - 1s - loss: 0.2671 - acc: 0.8968 - val_loss: 0.3355 - val_acc: 0.8533
Epoch 61/200
 - 1s - loss: 0.2135 - acc: 0.9131 - val_loss: 0.3507 - val_acc: 0.8578
Epoch 62/200
 - 1s - loss: 0.2060 - acc: 0.9116 - val_loss: 0.4814 - val_acc: 0.8222
Epoch 63/200
 - 1s - loss: 0.2261 - acc: 0.9052 - val_loss: 0.3173 - val_acc: 0.8711
Epoch 64/200
 - 1s - loss: 0.1788 - acc: 0.9323 - val_loss: 0.2993 - val_acc: 0.8578
Epoch 65/200
 - 1s - loss: 0.1958 - acc: 0.9195 - val_loss: 0.2669 - val_acc: 0.8889
Epoch 66/200
 - 1s - loss: 0.1958 - acc: 0.9210 - val_loss: 0.2668 - val_acc: 0.8844
Epoch 67/200
 - 1s - loss: 0.1871 - acc: 0.9264 - val_loss: 0.3078 - val_acc: 0.8800
Epoch 68/200
 - 1s - loss: 0.1821 - acc: 0.9269 - val_loss: 0.3319 - val_acc: 0.8578
Epoch 69/200
 - 1s - loss: 0.1805 - acc: 0.9264 - val_loss: 0.2229 - val_acc: 0.8933
Epoch 70/200
 - 1s - loss: 0.1865 - acc: 0.9254 - val_loss: 0.3069 - val_acc: 0.8578
Epoch 71/200
 - 1s - loss: 0.1863 - acc: 0.9294 - val_loss: 0.3000 - val_acc: 0.8800
Epoch 72/200
 - 1s - loss: 0.1917 - acc: 0.9131 - val_loss: 0.3670 - val_acc: 0.8578
Epoch 73/200
 - 1s - loss: 0.1760 - acc: 0.9299 - val_loss: 0.2915 - val_acc: 0.8889
Epoch 74/200
 - 1s - loss: 0.1978 - acc: 0.9141 - val_loss: 0.3211 - val_acc: 0.8756
```

```
Epoch 75/200
 - 1s - loss: 0.1822 - acc: 0.9200 - val_loss: 0.3128 - val_acc: 0.8489
Epoch 76/200
 - 1s - loss: 0.1593 - acc: 0.9368 - val_loss: 0.2444 - val_acc: 0.8844
Epoch 77/200
 - 1s - loss: 0.1726 - acc: 0.9348 - val_loss: 0.2547 - val_acc: 0.8978
Epoch 78/200
 - 1s - loss: 0.1633 - acc: 0.9333 - val_loss: 0.4057 - val_acc: 0.8667
Epoch 79/200
 - 1s - loss: 0.1973 - acc: 0.9249 - val_loss: 0.3414 - val_acc: 0.8800
Epoch 80/200
 - 1s - loss: 0.2940 - acc: 0.9047 - val_loss: 0.2978 - val_acc: 0.8800
Epoch 81/200
 - 1s - loss: 0.2596 - acc: 0.9032 - val_loss: 0.2688 - val_acc: 0.8756
Epoch 82/200
 - 1s - loss: 0.2201 - acc: 0.9131 - val_loss: 0.4018 - val_acc: 0.8444
Epoch 83/200
 - 1s - loss: 0.2567 - acc: 0.8978 - val_loss: 0.3062 - val_acc: 0.8711
Epoch 84/200
 - 1s - loss: 0.1985 - acc: 0.9235 - val_loss: 0.2886 - val_acc: 0.8756
```

4 Evaluate the model

Evaluate the model, calculating all metrics. All arfticats will be stored under provided `experiment_dest_path`. In this step, it is also possible to inject nosie into the `test` set, similarly to the previous function call.

[ ]:
```
evaluate_model.evaluate(
    model_path=os.path.join(experiment_dest_path, 'model_2d'),
    data=data,
    dest_path=experiment_dest_path,
    n_classes=9,
    batch_size=1024,
    noise=['gaussian'],
    noise_sets=['test'],
    noise_params="{\"mean\": 0, \"std\": 1, \"pa\": 0.1, \"pb\": 1}")
tf.keras.backend.clear_session()
```