model_noise_injection_ensemble

December 2, 2020

1 Noise injection into models, ensemble

Inject noise into the model's weights to create an augmented version. There are two scenarios were such operation could be useful: - Inject noise into the model to verify its robustness agaist unpredicted noise. - Model could be augmented N times, forming an ensemble. Each model makes its own predictions, which are then aggregated to conclude a final prediction.

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

[2]:
```
import os

import tensorflow as tf

from scripts import evaluate_model, prepare_data, artifacts_reporter,␣
↪train_model
```

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

[3]:
```
DEST_PATH = 'example_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 original model

The function `trian_model.train` executed the trainig procedure. Trained model will be stored under `experiment_dest_path` folder path. For details about all arguments, please refer to the documentation of the `train_model.train` function (located in `scripts/train_model`).

[5]:
```
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=[],
                  noise_sets=[])
```

```
_________________________________________________________________
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: 1.8423 - acc: 0.2884 - val_loss: 1.3107 - val_acc: 0.5111
Epoch 2/200
 - 1s - loss: 1.0250 - acc: 0.5793 - val_loss: 0.8917 - val_acc: 0.6800
Epoch 3/200
 - 1s - loss: 0.7752 - acc: 0.6790 - val_loss: 0.7486 - val_acc: 0.6800
Epoch 4/200
 - 1s - loss: 0.6684 - acc: 0.7081 - val_loss: 0.6481 - val_acc: 0.7156
Epoch 5/200
 - 1s - loss: 0.6096 - acc: 0.7373 - val_loss: 0.5814 - val_acc: 0.7689
Epoch 6/200
 - 1s - loss: 0.5590 - acc: 0.7565 - val_loss: 0.6010 - val_acc: 0.7333
Epoch 7/200
 - 1s - loss: 0.6426 - acc: 0.7160 - val_loss: 0.5996 - val_acc: 0.7689
Epoch 8/200
 - 1s - loss: 0.5781 - acc: 0.7486 - val_loss: 0.6293 - val_acc: 0.7200
Epoch 9/200
 - 1s - loss: 0.5217 - acc: 0.7699 - val_loss: 0.5584 - val_acc: 0.7556
Epoch 10/200
 - 1s - loss: 0.4973 - acc: 0.7694 - val_loss: 0.5960 - val_acc: 0.6800
Epoch 11/200
 - 1s - loss: 0.5231 - acc: 0.7551 - val_loss: 0.4989 - val_acc: 0.7822
Epoch 12/200
 - 1s - loss: 0.4882 - acc: 0.7921 - val_loss: 0.6526 - val_acc: 0.7600
Epoch 13/200
 - 1s - loss: 0.4948 - acc: 0.7921 - val_loss: 0.4600 - val_acc: 0.7867
Epoch 14/200
 - 1s - loss: 0.4596 - acc: 0.8020 - val_loss: 0.5171 - val_acc: 0.7556
Epoch 15/200
 - 1s - loss: 0.4426 - acc: 0.8020 - val_loss: 0.4853 - val_acc: 0.7911
```

```
Epoch 16/200
 - 1s - loss: 0.4619 - acc: 0.8059 - val_loss: 0.5004 - val_acc: 0.7600
Epoch 17/200
 - 1s - loss: 0.4298 - acc: 0.8247 - val_loss: 0.4557 - val_acc: 0.7956
Epoch 18/200
 - 1s - loss: 0.4340 - acc: 0.8064 - val_loss: 0.4986 - val_acc: 0.7644
Epoch 19/200
 - 1s - loss: 0.4982 - acc: 0.7837 - val_loss: 0.4597 - val_acc: 0.7956
Epoch 20/200
 - 1s - loss: 0.4036 - acc: 0.8291 - val_loss: 0.4257 - val_acc: 0.8178
Epoch 21/200
 - 1s - loss: 0.3990 - acc: 0.8356 - val_loss: 0.4116 - val_acc: 0.8178
Epoch 22/200
 - 1s - loss: 0.3936 - acc: 0.8291 - val_loss: 0.5127 - val_acc: 0.7867
Epoch 23/200
 - 1s - loss: 0.4140 - acc: 0.8277 - val_loss: 0.4421 - val_acc: 0.8000
Epoch 24/200
 - 1s - loss: 0.3751 - acc: 0.8351 - val_loss: 0.4708 - val_acc: 0.8000
Epoch 25/200
 - 1s - loss: 0.3762 - acc: 0.8395 - val_loss: 0.4232 - val_acc: 0.8133
Epoch 26/200
 - 1s - loss: 0.3567 - acc: 0.8459 - val_loss: 0.4257 - val_acc: 0.7867
Epoch 27/200
 - 1s - loss: 0.3998 - acc: 0.8286 - val_loss: 0.4965 - val_acc: 0.8000
Epoch 28/200
 - 1s - loss: 0.3857 - acc: 0.8316 - val_loss: 0.4216 - val_acc: 0.8311
Epoch 29/200
 - 1s - loss: 0.3789 - acc: 0.8380 - val_loss: 0.4744 - val_acc: 0.8133
Epoch 30/200
 - 1s - loss: 0.3431 - acc: 0.8538 - val_loss: 0.4023 - val_acc: 0.8267
Epoch 31/200
 - 1s - loss: 0.3385 - acc: 0.8563 - val_loss: 0.3821 - val_acc: 0.8311
Epoch 32/200
 - 1s - loss: 0.3517 - acc: 0.8474 - val_loss: 0.4698 - val_acc: 0.8489
Epoch 33/200
 - 1s - loss: 0.3662 - acc: 0.8365 - val_loss: 0.3751 - val_acc: 0.8089
Epoch 34/200
 - 1s - loss: 0.3233 - acc: 0.8573 - val_loss: 0.3632 - val_acc: 0.8667
Epoch 35/200
 - 1s - loss: 0.3336 - acc: 0.8607 - val_loss: 0.3810 - val_acc: 0.8311
Epoch 36/200
 - 1s - loss: 0.3139 - acc: 0.8696 - val_loss: 0.3663 - val_acc: 0.8267
Epoch 37/200
 - 1s - loss: 0.2761 - acc: 0.8904 - val_loss: 0.3149 - val_acc: 0.8578
Epoch 38/200
 - 1s - loss: 0.2614 - acc: 0.8854 - val_loss: 0.3199 - val_acc: 0.8578
Epoch 39/200
 - 1s - loss: 0.2708 - acc: 0.8859 - val_loss: 0.3422 - val_acc: 0.8311
```

```
Epoch 40/200
 - 1s - loss: 0.2842 - acc: 0.8815 - val_loss: 0.3099 - val_acc: 0.8489
Epoch 41/200
 - 1s - loss: 0.2766 - acc: 0.8869 - val_loss: 0.3280 - val_acc: 0.8444
Epoch 42/200
 - 1s - loss: 0.2600 - acc: 0.8973 - val_loss: 0.3292 - val_acc: 0.8400
Epoch 43/200
 - 1s - loss: 0.2726 - acc: 0.8770 - val_loss: 0.3750 - val_acc: 0.8267
Epoch 44/200
 - 1s - loss: 0.2696 - acc: 0.8889 - val_loss: 0.3290 - val_acc: 0.8667
Epoch 45/200
 - 1s - loss: 0.2941 - acc: 0.8830 - val_loss: 0.3248 - val_acc: 0.8667
Epoch 46/200
 - 1s - loss: 0.2873 - acc: 0.8844 - val_loss: 0.3973 - val_acc: 0.8533
Epoch 47/200
 - 1s - loss: 0.3035 - acc: 0.8790 - val_loss: 0.3368 - val_acc: 0.8533
Epoch 48/200
 - 1s - loss: 0.2461 - acc: 0.8953 - val_loss: 0.3060 - val_acc: 0.8711
Epoch 49/200
 - 1s - loss: 0.2422 - acc: 0.8919 - val_loss: 0.3549 - val_acc: 0.8711
Epoch 50/200
 - 1s - loss: 0.2303 - acc: 0.9067 - val_loss: 0.2997 - val_acc: 0.8622
Epoch 51/200
 - 1s - loss: 0.2423 - acc: 0.8958 - val_loss: 0.3090 - val_acc: 0.8800
Epoch 52/200
 - 1s - loss: 0.2330 - acc: 0.9027 - val_loss: 0.2848 - val_acc: 0.8756
Epoch 53/200
 - 1s - loss: 0.2117 - acc: 0.9141 - val_loss: 0.2781 - val_acc: 0.8578
Epoch 54/200
 - 1s - loss: 0.2086 - acc: 0.9151 - val_loss: 0.3018 - val_acc: 0.8667
Epoch 55/200
 - 1s - loss: 0.2288 - acc: 0.9086 - val_loss: 0.2856 - val_acc: 0.8711
Epoch 56/200
 - 1s - loss: 0.2319 - acc: 0.9027 - val_loss: 0.3271 - val_acc: 0.8578
Epoch 57/200
 - 1s - loss: 0.2410 - acc: 0.8953 - val_loss: 0.3243 - val_acc: 0.8667
Epoch 58/200
 - 1s - loss: 0.2357 - acc: 0.9042 - val_loss: 0.4307 - val_acc: 0.8533
Epoch 59/200
 - 1s - loss: 0.2814 - acc: 0.8869 - val_loss: 0.4320 - val_acc: 0.8533
Epoch 60/200
 - 1s - loss: 0.3153 - acc: 0.8746 - val_loss: 0.3284 - val_acc: 0.8400
Epoch 61/200
 - 1s - loss: 0.2686 - acc: 0.8894 - val_loss: 0.2790 - val_acc: 0.8933
Epoch 62/200
 - 1s - loss: 0.2201 - acc: 0.9086 - val_loss: 0.2818 - val_acc: 0.8889
Epoch 63/200
 - 1s - loss: 0.2118 - acc: 0.9116 - val_loss: 0.2571 - val_acc: 0.8800
```

```
Epoch 64/200
 - 1s - loss: 0.2059 - acc: 0.9160 - val_loss: 0.2581 - val_acc: 0.8711
Epoch 65/200
 - 1s - loss: 0.2022 - acc: 0.9160 - val_loss: 0.2651 - val_acc: 0.8756
Epoch 66/200
 - 1s - loss: 0.2036 - acc: 0.9195 - val_loss: 0.3508 - val_acc: 0.8444
Epoch 67/200
 - 1s - loss: 0.2178 - acc: 0.9126 - val_loss: 0.2374 - val_acc: 0.8889
Epoch 68/200
 - 1s - loss: 0.1904 - acc: 0.9284 - val_loss: 0.2619 - val_acc: 0.8844
Epoch 69/200
 - 1s - loss: 0.2016 - acc: 0.9185 - val_loss: 0.2581 - val_acc: 0.8978
Epoch 70/200
 - 1s - loss: 0.1842 - acc: 0.9269 - val_loss: 0.2634 - val_acc: 0.8622
Epoch 71/200
 - 1s - loss: 0.1835 - acc: 0.9269 - val_loss: 0.2828 - val_acc: 0.8889
Epoch 72/200
 - 1s - loss: 0.2194 - acc: 0.9116 - val_loss: 0.2562 - val_acc: 0.8711
Epoch 73/200
 - 1s - loss: 0.2519 - acc: 0.8983 - val_loss: 0.3043 - val_acc: 0.8489
Epoch 74/200
 - 1s - loss: 0.2196 - acc: 0.9101 - val_loss: 0.2614 - val_acc: 0.8800
Epoch 75/200
 - 1s - loss: 0.1852 - acc: 0.9294 - val_loss: 0.2702 - val_acc: 0.8756
Epoch 76/200
 - 1s - loss: 0.2115 - acc: 0.9072 - val_loss: 0.3366 - val_acc: 0.8533
Epoch 77/200
 - 1s - loss: 0.2007 - acc: 0.9269 - val_loss: 0.2438 - val_acc: 0.8933
Epoch 78/200
 - 1s - loss: 0.2030 - acc: 0.9146 - val_loss: 0.3068 - val_acc: 0.8800
Epoch 79/200
 - 1s - loss: 0.2015 - acc: 0.9121 - val_loss: 0.2778 - val_acc: 0.8800
Epoch 80/200
 - 1s - loss: 0.2222 - acc: 0.9096 - val_loss: 0.2870 - val_acc: 0.8578
Epoch 81/200
 - 1s - loss: 0.1999 - acc: 0.9269 - val_loss: 0.2449 - val_acc: 0.8889
Epoch 82/200
 - 1s - loss: 0.1891 - acc: 0.9235 - val_loss: 0.2787 - val_acc: 0.8889
```

4 Evaluate an ensemble of models

To use an ensemble of augmented models, provide a `use_ensemble` argument to `evaluate_model.evaluate` function. Indicate how many copies should be generated with `ensemble_copies` parameter. Lastly, indicate the voting algorithm. It accepts three values:

- **hard** - uses predicted class labels for majority rule voting.
- **soft** - predicts the class label based on the argmax of the sums of the predicted probabilities.

- **classifier** - Use a classifier which accepts predicted probabilities from all the models as features and return the final prediction. The classifier is trained on the train set predictions. Random forest will be used.

Each layer is modified separately. It is achieved by drawing a random number for each of the parameters of the layer. The number is drawn from the normal distribution with provided mean and standard deviation calculated from the layer's parameters, multiplied by `0.1`.

[ ]:
```
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,
    use_ensemble=True,
    ensemble_copies=4,
    voting='hard',
    noise=[],
    noise_sets=[],
    noise_params="{\"mean\": 0, \"std\": None}")
tf.keras.backend.clear_session()
```