NAGATA Shigemi Laboratory
"Research on Automatic Concrete Compaction Judgment System Based on Deep Learning"

Manual for the Judgment System Based on Semantic Segmentation

Author : Osada Masashi (Bachelor, Graduation in March 2023)

- Index -

- Introduction -

Summary

This is a description of a set of codes related to the learning and evaluation of a judgment system based on Semantic Segmentation.

Working Directory

My working directory is `/media/nagalab/SSD1.7TB/nagalab/osada_ws/concrete_compaction/` . And the directory on learning is `/media/nagalab/SSD1.7TB/nagalab/osada_ws/concrete_compaction/KerasFramework-master_tf2/train/` .

The above directories can be navigated to with the following commands.

```
$ cdws
```

Commands

Ctrl + Alt + T opens a Terminal.

Navigate to working directory.

```
$ cdws
```

Free memory every 10 seconds.

```
$ ~/interval_rm_cache
```

Launch into the Docker environment.

```
$ sh ~/launch.sh 0
```

Researh Outline

Please see the 2023 Proceedings(令和４年度予稿集, Reiwa 4th Yokosyu).

– Building an Env for Docker –

⋆ Env : environment

Directory configuration

- Any directory
    - Dockerfile
    - entrypoint.sh
    - build_docker.sh
    - launch.sh

Execution Procedure

Build a Docker environment. Run only the first time, even if you have changed the Dockerfile.

```
$ sh build_docker.sh <tag name>
```

Launch into the Docker environment.

```
$ sh launch.sh <GPU number>
```

For example, `sh launch.sh 0` would use the 0th GPU. If `sh launch.sh all` , all GPUs are used.

Using Multi-GPU

By the way, to make it multi-GPU, you need to enter Docker as `sh launch.sh all` and then in your Python program, you must write the following.

```
from tensorflow.distribute import MirroredStrategy

strategy = MirroredStrategy()

with strategy.scope():
    model = Define a your model

    ## If you want to set up the original loss function or Optimizer
settings.
    ## Define them in this with statement.

with strategy.scope():

    ## Start learning.
    model.fit()
```

Codes

>> entrypoint.sh

```
#!/bin/bash

USER_ID=${LOCAL_UID:-9001}
GROUP_ID=${LOCAL_GID:-9001}

echo "Starting with UID : $USER_ID, GID: $GROUP_ID"
useradd -u $USER_ID -o -m user
groupmod -o -g $GROUP_ID user
export HOME=/home/user

echo "alias make='python /workspace/train/make_learning_plan.py'" >>
/home/user/.bashrc
echo "alias learn='sh /workspace/train/learning_plan.sh'" >>
/home/user/.bashrc
echo "alias plan='more /workspace/train/learning_plan.sh'" >>
/home/user/.bashrc
echo "export TF_CPP_MIN_LOG_LEVEL=2" >> /home/user/.bashrc

. /home/user/.bashrc

exec /usr/sbin/gosu user "$@"
```

>> Dockerfile

```
FROM nvcr.io/nvidia/tensorflow:21.09-tf2-py3

RUN apt-get update && apt-get upgrade -y && apt-get -y install gosu vim
graphviz v4l-utils ffmpeg
COPY entrypoint.sh /usr/local/bin/entrypoint.sh
RUN chmod +x /usr/local/bin/entrypoint.sh
ENTRYPOINT ["/usr/local/bin/entrypoint.sh"]

USER root

RUN apt-get install -y libgl1-mesa-dev
RUN pip install --upgrade pip

RUN pip install numpy==1.19.2
RUN pip install opencv-python
RUN pip install matplotlib
RUN pip install colorama
RUN pip install pillow
RUN pip install scikit-learn
RUN pip install pandas
RUN pip install pyyaml
RUN pip install argparse
RUN pip install pydot
RUN pip install tqdm
RUN pip install line-bot-sdk
RUN pip install sklearn
RUN pip install umap-learn
RUN pip install scikit-image
RUN pip install openpyxl
RUN pip install tensorflow_model_optimization
RUN pip install tensorflow==2.10.0
RUN pip install git+https://github.com/tensorflow/examples.git
```

>> build_docker.sh

```
#!/bin/bash

if [ $# -ne 1 ]; then
    docker images
    echo;
    echo "タグ名の入力が必要" 1>&2
    exit 1
fi

cd /media/nagalab/SSD1.7TB/nagalab/osada_ws/docker
docker image build -t osada:$1 .
```

>> launch.sh

```
#!/bin/bash

workspace="/media/nagalab/SSD1.7TB/nagalab/osada_ws"
dataset="/media/nagalab/SSD1.7TB/nagalab/Dataset"

if [ $# -ne 1 ]; then
    nvidia-smi
    echo;
    echo "GPUの番号も入力して" 1>&2
    exit 1
fi

xhost +

docker container run \
--rm --gpus "device=$1" -it \
-v $workspace/concrete_compaction/:/workspace/osada_ws \
-v $workspace/concrete_compaction/KerasFramework-
master_tf2/train:/workspace/train \
-v
$workspace/concrete_compaction/text_dataset/ngc_docker:/workspace/mesh_data
set \
-v $workspace/LuminanceData/:/workspace/luminance \
-v $workspace/semanticSegmentation/:/workspace/semanticSegmentation \
-v $workspace/fullframe/:/workspace/fullframe \
-v $workspace/visualization/:/workspace/visualization \
-v $workspace/hidden_layer/:/workspace/hidden \
-v $workspace/explain/:/workspace/explain \
-v $dataset/CompactionVideo/:/workspace/video \
-v $workspace/mesh_encoder_result/:/workspace/mesh_encoder_result \
-v $workspace/cpp/:/workspace/cpp \
-v $workspace/result_y/:/workspace/result_y \
-v $dataset:/workspace/Dataset \
-v /home/nagalab/:/workspace/home_nagalab \
-v /tmp/.X11-unix:/tmp/.X11-unix:rw \
-e DISPLAY=$DISPLAY \
-e LOCAL_UID=$(id -u $USER) -e LOCAL_GID=$(id -g $USER) \
osada:20220707 bash
```

Notes on using Docker

Docker environments will remain unless you delete them.

```
$ docker images
```

The above command will show you the currently stored Docker environments. You can delete the ones you don't need with the command below.

```
$ docker rmi <ID of the Docker you want to delete>.
```

If you accumulate too many, the worst thing that can happen is that your PC stops working. It is quite a hassle to fix.

Be careful not to erase Docker environments that are dependencies. Just in case, there should be a warning before deleting the environment with dependencies, so that you cannot delete it easily.

Also, if you overwrite a Docker environment with the same tag name, the previous environment will remain with the name `<none>`. Please delete it. Sometimes you will end up with a dependency with `<none>` Docker, but that is no longer an option. Please do not touch it.

- How to learning some models -

Example of the simplest execution

First, define the learning you wish to turn. The definition of the learning is written in Python code. The simplest example is shown below.

The workning directory is located in `/media/nagalab/SSD1.7TB/nagalab/osada_ws/concrete_compaction/KerasFramework-master_tf2/train/` .

The file name should be `make_learning_plan.py` .

```
from build_learning_plan import *


## Create a new ShellScript
Shell.new_file(Train.SH_TEXT)

## The array to store study plans
plan = []

## Draw a cat ASCII art in ShellScript
```

```
plan.append(CatInBox())


## k-partition cross-validation(k=5)
for fold in range(1, 6):

    plan.append(Indention())

    ## Train
    plan.append(Train(
        epochs=50,
        batch_size=8
        fold=fold,
        network_name="eunet",
        save_id="concrete",
        loss="categorical_crossentropy",
        optimizer="adam",
        norm="batch_norm",
    ))

    ## Test
    plan.append(Test(
        fold=fold,
        network_name="eunet",
        load_id="eunet_concrete",
    ))


## Save to ShellScript
for p in plan:
    p.output_shellscript(Train.SH_TEXT, p.params)
```

In the example above.

- Number of epochs is 50
- Batch size is 8
- The deep learning model is an E-UNet
- The name of the model is "eunet_concrete".
- The loss function is CrossEntropy
- Optimizer is Adam
- The regularization function is BatchNormalization

For the actual learning, after entering the Docker environment, the learning can be performed with the following commands.

```
$ make
$ learn
```

By the way, you can see the ShellScript created by typing the command `plan` .

Detailed Learning Settings

By changing the arguments passed to the Train() and Test() classes mentioned above, various types of learning can be configured.

You can probably see all the arguments by looking at `build_learning_plan.py` , so you can guess what functions can be set from the argument names.

I always use the following learning

- Semantic Segmentation
- Load the learned weights
- The network is CFPNet-M
- Trained as a 4-class classification task
- Use the trained CFPNet-M as the model for illumination correction
- Randomly rotate the training dataset
- Randomly flip the training dataset horizontally and vertically
- Randomly transform the training dataset with illumination

The following is an example of executing.

```python
from build_learning_plan import *
import sys


## In case something goes wrong, you can save the ShellScript as a separate file by adding an extra argument to the make command
args = sys.argv
buf = "" if (len(args) == 1) else f"_{args[1]}"


## ================ config ================

SH_TEXT = f"/workspace/train/learning_plan{buf}.sh"

AE_ID = "cfpnetm_20230107_ssim_mse_dopout-0-25"

NETWORK_NAME = "cfpnetm_4class"

## Pre-Trained
LOAD_ID = "20230205-2_AutoLearning"

SAVE_ID = "20230209_eunetAE_AutoLearning"
```

```
## ========================================


Shell.new_file(SH_TEXT)

## Remove from model save name
rep = WithoutReplace(
    arcface=True,
    cosface=True,
    sphereface=True,
    only_classifier=True,
    fourclass=True,
    after=True,
)

plan = []
plan.append(CatInBox())


## k-partition cross-validation(k=5)
for fold in range(1, 6):

    ## 改行
    plan.append(Indention())

    ## Train
    plan.append(Train(is_autotrain=True,
        auto_train_acc_limit=95,
        fold=fold,
        network_name=rep.network_name(NETWORK_NAME),
        save_path="$fullframe",
        save_id=LOAD_ID,
        is_use_fullframe=True,
        is_use_fresh=False,
        is_load_weight=True,
        load_weight_path=f"
{rep.network_name(NETWORK_NAME)}_4class_{SAVE_ID}_fold{fold}_576x576",
        is_extend_luminance=False,
        is_grayscale=False,
        is_use_BCL=False,
        loss="$cross",
        optimizer="adam",
        is_h5=False,
        is_use_metric=False,
        is_load_fullframe_weight=True,
        output_layer_name="classifier",
        is_fusion_face=False,
        nullification_metric=False,
        dropout_const=0.01,
        label_smoothing=0,
        norm="batch_norm",
        use_attention=False,
        classification="fourclasses",
        multi_losses=False,
        fourclasses_type="rectified",
        eunet_metric_mode="conv_dense",
        eunet_metric_subcontext="default",
        color_type="rgb",
        normalization="default",
        noise_type="includeAE-noise",
        use_AE_input=True,
        AE_model_id=AE_ID,
        is_flip=True,
        flip_list=[0, 0, 1, 1, 2, 3],
        is_rotate=True,
        rotate_degrees=[[0, 360]],
        is_enlarge=False,
        reduce_const=1.,
        learning_rate=1e-3,
        rotate_rate=1.,
        all_in_one=False,
    ))

    # Test
    for fourclasses_test in range(2):
        plan.append(Test(fold=fold,
            network_name=rep.network_name(NETWORK_NAME),
            load_path="$fullframe",
            load_id=f"
{rep.network_name(NETWORK_NAME)}_4class_{LOAD_ID}_fold{fold}",
            size=(576, 576),
            is_use_fullframe=True,
            is_use_fresh=False,
            is_judgement_by_mesh=False,
            is_grayscale=False,
            is_fusion_face=False,
            norm="batch_norm",
            is_use_averageimage=True,
            use_attention=False,
            classification="fourclasses",
            is_quantized=False,
            do_fourclasses_test=fourclasses_test,
            do_threeclasses_test=False,
            multi_losses=False,
            fourclasses_type="rectified",
            is_use_LE=False,
            use_AE_input=True,
            AE_model_id=AE_ID,
        ))


## Save to ShellScript
for p in plan:
    p.output_shellscript(Train.SH_TEXT, p.params)
```

Basically, I think we can perform various studies in the form of modifications to the above code.

If you need a new function and want to add arguments, you can use `SemanticSegmentationTrain.py` and `SemanticSegmentationTest.py`. and then add the arguments to `build_learning_plan.py` and add the arguments to `build_learning_plan.py`.

The code for learning has been built in stages over the course of a year, so the design may be somewhat inconsistent. Still, we are quite careful about the amount of time calculation, so we think it will work efficiently.

"make", "learn" command definitions

The definitions of the `make` and `learn` commands are as follows

```
alias make='python /workspace/train/make_learning_plan.py'
alias learn='sh /workspace/train/learning_plan.sh'
```

`/workspace/` is the working directory you specify when building your Docker environment.

`/workspace/train/` is the working directory on my PC `/media/nagalab/SSD1.7TB/nagalab/osada_ws/concrete_compaction/KerasFramework-master_tf2/train/` , so please read accordingly.

Others

>> Dataset

A dataset consists of images and text with their path names. As you can actually see when you look into the dataset folder, there is a large amount of text with different names. Among them, `??_rectified.txt` is the latest one, so please use it. If none of the above apply

>> Line Bot

First, "Line" is a chat application commonly used in Japan.

We have a Line Bot that sends the results to your own Line at the end of the study/test, but you cannot use it because you have deleted the information you have set up. If you want to use it, please register yourself to the Line API and then write the necessary information in `~/line_information.txt` .

>> Application of branch pruning to CNNs

In `MyPruning.py` , we have implemented the original CNN lightweight framework that extends branch pruning to be used in CNNs. Just going through this part could be one research topic.

```
from MyPruning import MyPruning

model_before = # Pre-Trained model
model_after = # Model with smaller filter size than the model above

model = MyPruning.prune_tuning(model_before=model_before,
model_after=model_after, reduce="ssim")
```

With the above code, we can copy the weights of `model_before` to `model_after` while pruning branches.

The mechanism is described in my project report, but I'll explain it in a moment.

Branch pruning is one of the lightening techniques in machine learning. Usually refers to a method to reduce the number of parameters by removing weights with small values from the weights of all coupled layers. On the other hand, convolution operation is a computation on two rectangular matrices, each of which is a rectangular matrix, so branch and prune is not applicable. (More precisely, the computation between two matrices of shapes that can be computed by Fast Discrete Fourier Transform.) So we did something similar to branch-and-prune by removing the entire kernel (filter) of the convolutional layer.

Compute the SSIM for all streets in the convolution kernel that the convolution layer has. where SSIM is the structural similarity between images. Deletes the convolution kernel with a higher rank when its mutual SSIMs are sorted in order of magnitude. In other words, only one of the kernels that extracts similar image features will be removed. If we were to represent it in pseudo code (Python3 Like), it would look like the following. (You should implement the function by yourself.)

```
model_before = # Pre-Trained model
model_after = # Model with smaller filter size than the model above
```

```
for layer_be, layer_af in zip(model_before.layers, model_after.layers):
    w = layer_be.weights

    if ("conv" in layer_be.name):

        ## Calculate mutual SSIM
        ssim_val = calc_co_ssim(w)

        ## Sort kernel numbers in order of decreasing mutual SSIM
        include_img = argsort(sum(ssim_val, axis=0))
[:layer_af.weights.shape[-1]]

        ## Save only selected kernels
        w_af = [i for i in range(size) if i in include_img]

    else:
        w_af = copy(w)

    layer_af.set_weights(w_af)
```

– Codes overview –

Directory configuration

- osada_ws/
  - concrete_compaction/KerasFramework-master_tf2/
    - train/
      - SemanticSegmentationTrain.py
      - SemanticSegmentationTest.py
      - SemSegLight.py
      - MyUtils.py
      - luminance_function.py
      - Optimizer.py
      - Normalization.py
      - my_loss_function.py
      - train_autoencoder.py
      - MyPruning.py
      - make_learning_plan.py
      - build_learning_plan.py
      - learning_plan.sh
  - fullframe/
    - result/

        - 540x540/
        - autoencoder/

Perhaps you can guess the function from the file name. Basically, `SemanticSegmentationTrain.py` , `SemanticSegmentationTest.py` , `make_learning_plan.py` , `MyUtils.py` , `MyUtils.py` , and `MyUtils.py`. I think you will have to tweak above.

The weights of the deep learning model are stored in `fullframe/540x540/` . The name "540x540" is a remnant of the 540x540 image size used to be used for training. It no longer has any particular meaning.

The model for illumination correction is stored in `fullframe/autoencoder/` .