Preferred Networks Intern Screening 2019 Coding Task for Chainer (Computer Vision) Field

- This problem is about the computer vision using machine learning.
- The problem #2 assumes the knowledge of Chainer.

Changelog

- 2019-04-19 : Initial version

Notice

- You can use OpenCV or any other UIs like web browser for #1.
- Please use chainer for (https://chainer.org/) #2
- Please tackle the task by yourself. Do not share or discuss this coding task with anyone including other applicants. Especially, do not upload your solution and/or problem description to public repository of GitHub during screening period. If we find any evidence of leakage, the applicant will be disqualified. If one applicant allows another applicant to copy answers, both applicants will be disqualified.
- We expect you to spend up to two days for this task. You can submit your work without solving all of the problems. Please do your best without neglecting your coursework.

Things to submit

- Submit the source code along with the building instructions and execution results for both problems #1 and #2.
- Submit a pdf file of screen shots for #1 and disscussion for #2.

Evaluation

It is desirable that your submission satisfies the following. (These are not mandatory. Your submission does not need to satisfy all of them if you are too busy.)

- The source code is readable for other programmers.
- There is an appropriate amount of unit tests and/or verification code to ensure the source code is correct.
- It is easy to reproduce the experimental results.
- The submitted report is concise and easy to follow.

During the interview after the first screening process, we may ask some questions on your source code.

How to submit

- Create a zip archive with all of the submission materials and submit it on this form. Due date is Tuesday, May 7th, 2019, 12:00 JST.

Inquiry

- If you have any question regarding this problem description, please contact us at intern2019-admin@preferred.jp. An updated version will be shared with all applicants if any changes are made. Note that we cannot comment on the approach or give hints to the solution.

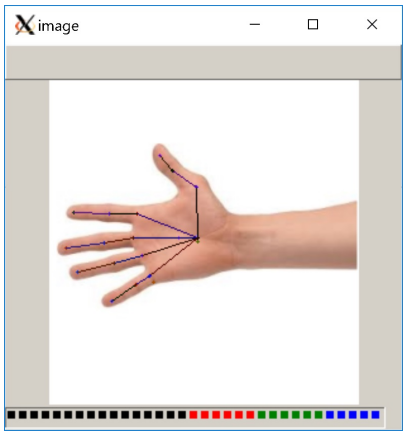
Task description

Problem 1: Make an annotation tool for hand pose detection.

- You can use OpenCV or any other UIs like web browser.
- Attach screenshots for GUI implementations.

Pose Detection is one of the major open problems of machine learning. OpenPose is a popular example.

In this section, let us suppose a task of pose detection of a hand. At first, you need to prepare an annotation tool that makes a dataset consisting of images of hands and true positions of key-points.



1. Prepare images of hands (you can either download them from external websites or take photos of your own) and make a program to display them.

```
Assign 'n' key for loading next image.
```

2. Implement a function that displays a point at clicked place. These points are the key-points.
3. Assign 's' key for saving a set positions to a file. Define the file format as you like.
4. Implement the function that reads and displays a set of key-points required for hand pose detection. The edges connecting the key-points should also be displayed.
5. To detect each finger separately, each key-point should belong to different class. Reimplement the tool so that it can store class information for each finger.
6. Propose some additional features that might be useful for this tool. Implementations are welcomed.

Problem 2: Try to train detection model using CNN.

In this problem, all implementation should use Chainer.

1. Make a dataset class that reads data from annotation tool created in #1, converts key-points to heat maps, and then returns heat maps along with original images.

```
- Each layer of a heat map should represent one class; the output shape of a heat map is
in [n_class x H x W].
- See also https://docs.chainer.org/en/stable/reference/datasets.html
```

2. Pretrained image detector network is often used as a feature extractor for other tasks. Make a model that predicts heat map from an image of a hand using pretrained VGG features.

```
- Note that input image size is 224x224px and output image size is 56x56px.
- VGG16:
https://docs.chainer.org/en/stable/reference/generated/chainer.links.VGG16Layers.html
(You can choose which features to be used.)
```

3. Since most parts of a heat map has a value of 0, the loss decreases even if predictor returns 0 for everywhere.

```
One of the methods that could be used to avoid this problem is called weighted loss
function. Implement such loss function.
- Example: weighted mean_abusolute_error that multiplies weight: 1 for values larger than
0, and weight: 0.01 otherwise (pixels with 0).
```

4. For more stable traing of key-points prediction, "gradual refinement" could be used.

```
For example, the first stage predicts key-points from the image features, and the second
stage predicts key-points from the image features and the output from the first stage,
and so on. Finally, the loss is calculated using each output.
Make a network of 3 stages that predicts outputs 1 to 3, and train it using the dataset
made in #1 and the loss function made in 3.
- The acurracy is not concerned.
- Submission of trained models is not necessary.
```

5. Display the prediction result of the trained model on the tool made in #1. (you can use image used to train)

```
- You need to transform heat map to positions of points.
- Example: getting positions of points with the maximum value for each class after
applying gaussian filter.
```

6. To improve the accuracy of result, pick up some methods and disscuss about it.

```
- The expected answer should not be as simple as just making dataset large.
- Example: the method to obtain more robust training result.
```