Reproducibility Companion Paper: Norm-in-Norm Loss with Faster Convergence and Better Performance for Image Quality Assessment

Dingquan Li
dingquanli@pku.edu.cn
Peking University & Peng Cheng Lab.
China

Tingting Jiang*
ttjiang@pku.edu.cn
Peking University
China

Ming Jiang
ming-jiang@pku.edu.cn
Peking University
China

Vajira Lasantha Thambawita
vajira@simula.no
SimulaMet
Norway

Haoliang Wang
hawang@adobe.com
Adobe Research
USA

ABSTRACT

This companion paper supports the experimental replication of the paper "Norm-in-Norm Loss with Faster Convergence and Better Performance for Image Quality Assessment" presented at ACM Multimedia 2020. We provide the software package for replicating the implementation of the "Norm-in-Norm" loss and the corresponding "LinearityIQA" model used in the original paper. This paper contains the guidelines to reproduce all the experimental results of the original paper.

CCS CONCEPTS

• **Computing methodologies** → **Image processing**; **Supervised learning by regression**; *Image representations*; • **General and reference** → *Measurement*; *Metrics*.

KEYWORDS

IQA; faster convergence; loss; normalization; reproducibility

ACM Reference Format:
Dingquan Li, Tingting Jiang, Ming Jiang, Vajira Lasantha Thambawita, and Haoliang Wang. 2021. Reproducibility Companion Paper: Norm-in-Norm Loss with Faster Convergence and Better Performance for Image Quality Assessment. In *Proceedings of the 29th ACM Int'l Conference on Multimedia (MM '21), Oct. 20–24, 2021, Virtual Event, China.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3474085.3477937

1 SOFTWARE PACKAGE DESCRIPTION

The software package is available at: https://github.com/lidq92/LinearityIQA. It is written in Python language with the PyTorch framework [4]. The program is executed in command line and it supports customization by optional parameters. The KonIQ-10k [2] and CLIVE [1] datasets can be downloaded from the official links or an alternative link provided by us. The model weights pre-trained on KonIQ-10k train set can be downloaded in https://drive.google.com/drive/folders/1S5XljhTyKsFkRclhijXBM6SrcD5Tov6Z, though it can also be re-trained with the provided source code.

The descriptions of files and folders in this software package are as follows.

- **README.md**: It is the documentation of this repository.
- **data/**: It contains the information (**e.g.**, MOS values) of KonIQ-10k and CLIVE datasets (**KonIQ-10kinfo.mat** and **CLIVEinfo.mat**), as well as a test image (**1000.JPG**).
- **requirements.txt**: It is the collections of package requirements.
- **IQAdataset.py**: It is used to prepare the IQA dataset for data loading.
- **IQAmodel.py**: It is to define the IQA model, *i.e.*, **"LinearityIQA"**.
- **IQAloss.py**: It is used to define the loss function for IQA model training. This file contains the implementation of **"Norm-in-Nrom" loss**.
- **IQAperformance.py**: It is used to calculate the performance metrics of the IQA model.
- **modified_ignite_engine.py**: It is a naive modification of ignite engine to support automatic mixed precision (amp).
- **main.py**: The main file for training the IQA model.
- **test_dataset.py**: It is used for testing the performance of a trained IQA model on a dataset.
- **test_demo.py**: It is a test demo using the pre-trained IQA model.
- **results_in_the_paper/**: This folder contains the reproduced results and procedures for reproducing the tables and figures in the original paper.

2 INSTALLATION

The software package can be downloaded with this command:

```
git clone https://github.com/lidq92/LinearityIQA.git
```

*Corresponding Author

MM '21, October 20–24, 2021, Virtual Event, China.

ACM ISBN 978-1-4503-8651-7/21/10…$15.00
https://doi.org/10.1145/3474085.3477937

2.1 System Environment

Our experiment was tested with Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz CPU, 128GB RAM and NVIDIA GeForce RTX 2080 Ti. The operating system is Ubuntu 16.04.1 LTS. The following dependencies is required for reproducing the experiments:

- **Python**: tested with version 3.6.9. It can be installed with anaconda distribution following this command:

```
conda create -n reproducibleresearch pip python=3.6
```

- **CUDA Toolkit and cuDNN**: tested with CUDA 10.0 and cuDNN 7.6.2. The installation can refer to official guide.
- **Python Dependencies**: The python packages used in the experiment. Except for the apex package, they can all be automatically installed with the command:

```
source activate reproducibleresearch
pip install -r requirements.txt
```

And the apex package (**version 0.1**) is installed from its official GitHub link.

```
git clone https://github.com/NVIDIA/apex.git
cd apex
pip install -v --no-cache-dir --global-option="--cpp_ext" --global-option="--cuda_ext" ./ > install.log
cd .. && rm -rf apex
```

3 REPRODUCED EXPERIMENT

After installing the software package, we can now conduct the reproduced experiments.

3.1 Downloading Datasets

The datasets KonIQ-10k (10073 images) and CLIVE (1162 images) used in the paper can be downloaded from the official links http://database.mmsp-kn.de/koniq-10k-database.html and https://live.ece.utexas.edu/research/ChallengeDB/index.html (or an alternative link https://drive.google.com/drive/folders/1E4nl3Y8yUDaKH2r0BjEQ6cFwdmMZaQQh?usp=sharing).
And then decompress the files by following the bash commands provided in **README.md**. We make soft links for the datasets to 'root-of-the-repository/KonIQ-10k' and 'root-of-the-repository/CLIVE', respectively.

3.2 Parameters and Their Default Values

One can use the command **python main.py –help** to have a look at the parameters and their default values. We list some important parameters here.

- **learning_rate**: learning rate, default 1e-4.
- **batch_size**: batch size, default 8.
- **ft_lr_ratio**: fine-tuned rate, *i.e.*, the ratio between the learning rate of the backbone's parameters and of the other parameters, default 0.1.
- **architecture**: the architecture, default resnext101_32x8d.
- **use_bn_end**: the option to use batch normalization at the end of the output, default False.
- **loss_type**: the loss function, *e.g.*, mae for MAE, mse for MSE, and norm-in-norm for "Norm-in-Norm" loss. default value is norm-in-norm.
- **p**: the hyper-parameter p in the "Norm-in-Norm" loss, default 1.
- **q**: the hyper-parameter p in the "Norm-in-Norm" loss, default 2.
- **dataset**: the considered dataset, default KonIQ-10k.
- **resize**: the option to set the image resized, default False.
- **test_during_training**: the option to test the performance of the test set in order to draw the test curve, default False.

For the **test_dataset.py**, there is one more important parameter, *i.e.*, **trained_model_file**, which is used for setting the path to the trained model file.

3.3 Reproducing Procedure

3.3.1 Reproducing Figure 1. To reproduce the results of Figure 1, we can set the parameter **loss_type** to different values including mae, mse and norm-in-norm. The results can be obtained by executing the following commands.

```
python main.py --dataset KonIQ-10k --resize --loss_type mae --test_during_training
python main.py --dataset KonIQ-10k --resize --loss_type mse --test_during_training
python main.py --dataset KonIQ-10k --resize --loss_type norm-in-norm --test_during_training
```

Listing 1: Commands for obtaining results of Figure 1

It would take about **930 seconds per epoch** for the model with the default resnext101_32x8d architecture during the training. After completing the above commands, we open TensorBoard visualization with **tensorboard --logdir=runs --port=6006**. Then, we can download the performance *metric* (**SROCC/PLCC/RMSE**) values in each *stage* (**train/val/test**) and each *loss* (**mae/mse/norm-in-norm**) from TensorBoard visualization, which are saved as the following format:

```
'loss={}-{}_KonIQ-10k_{}.csv'.format(loss, stage, metric)
```

These files are saved in results_in_the_paper/csv/. Finally, we can plot Figure 1 based on these results.

```
cd results_in_the_paper
python loss_performance_curves.py
cd ..
```

Listing 2: Commands for plotting Figure 1

3.3.2 Reproducing Figure 4. To reproduce the results of Figure 4, we can choose the option **use_bn_end** to set it be True. The results can be obtained by executing the following commands.

```
# python main.py --resize --loss_type norm-in-norm --test_during_training # done before
# python main.py --resize --loss_type mse --test_during_training # done before
python main.py --resize --loss_type mse --test_during_training --use_bn_end
```

Listing 3: Commands for obtaining results of Figure 4

After completing the above commands, we can download the performance *metric* (**SROCC/PLCC**) values in the val stage and for each *loss* (**Norm-in-Norm/MSE/bnMSE**) from TensorBoard visualization, which are saved as the following format:

```
'{}-{}.csv'.format(loss, metric)
```

These files are saved in results_in_the_paper/csv/. Finally, we can plot Figure 4 based on these results.

```
cd results_in_the_paper
python bnMSE.py
cd ..
```

Listing 4: Commands for plotting Figure 4

3.3.3 Reproducing Figure 5. To reproduce the results of Figure 5, we can vary the parameters (**lr/bs/ft_lr_ratio**). The learning rate (**lr**) is chosen from 1e-3, 1e-4, and 1e-5. The batch size (**bs**) varies from 4, 8, and 16. And the "fine-tuned rate" (**ft_lr_ratio**) is selected from 0, 0.01, 0.1, and 1. The results can be obtained by executing the following commands.

```
## base exp
python main.py --resize -lr 1e-4 -bs 8 --ft_lr_ratio 0.1
    -arch resnet50
# other lr
python main.py --resize -lr 1e-3 -arch resnet50
python main.py --resize -lr 1e-5 -arch resnet50
# other bs
python main.py --resize -bs 4 -arch resnet50
python main.py --resize -bs 16 -arch resnet50
# other ft_lr_ratio
python main.py --resize --ft_lr_ratio 0 -arch resnet50
python main.py --resize --ft_lr_ratio 0.01 -arch resnet50
python main.py --resize --ft_lr_ratio 1 -arch resnet50
```

Listing 5: Commands for obtaining results of Figure 5

After completing the above commands, we can download the PLCC values in the val stage for the norm-in-norm loss and for each *parameter* (**lr/bs/ft_lr_ratio**) *value* from TensorBoard visualization, which are saved as the following format:

```
'loss=norm-in-norm-{}={}-val_KonIQ-10k_PLCC.csv'
.format(parameter, value)
```

These files are saved in results_in_the_paper/csv/. Finally, we can plot Figure 5 based on these results.

```
cd results_in_the_paper
python lr_performance_plot.py
python batch_size_performance_plot.py
python ft_lr_ratio_performance_plot.py
cd ..
```

Listing 6: Commands for plotting Figure 5

3.3.4 Reproducing Figures 6 & 7. To reproduce the results of Figures 6 & 7, we can vary the parameter **arch** to set different backbone architectures. The results can be obtained by executing the following commands.

```
## Training on KonIQ-10k train set
python main.py --resize -arch resnet18
python main.py --resize -arch resnet34
# python main.py --resize -arch resnet50 # done before
# python main.py --resize -arch resnext101_32x8d # done
    before
## Testing on KonIQ-10k test set and CLIVE using
    test_dataset.py
python test_dataset.py --resize -arch resnet18
python test_dataset.py --resize -arch resnet34
python test_dataset.py --resize -arch resnet50
python test_dataset.py --resize -arch resnext101_32x8d
python test_dataset.py --resize -arch resnet18 --dataset
    CLIVE
python test_dataset.py --resize -arch resnet34 --dataset
    CLIVE
python test_dataset.py --resize -arch resnet50 --dataset
    CLIVE
python test_dataset.py --resize -arch resnext101_32x8d --
    dataset CLIVE
```

Listing 7: Commands for obtaining results of Figures 6 & 7

After completing the above commands, we obtain the test PLCC values on KonIQ-10k test set and SROCC values on CLIVE for each *arch* (**ResNet-18/ResNet-34/ResNet-50/ResNeXt-101**) from the printed screen, which are filled into backbone_performance_plot.py in the directory 'results_in_the_paper/'. When *arch* corresponds to **ResNeXt-101**, we move the corresponding saved results (in the directory 'results/') to results_in_the_paper/npy/ and rename them with the following format:

```
'{}-{}.npy'.format(dataset, loss)
```

Where *dataset* can be **KonIQ-10k** or **CLIVE**, and *loss* can be **MAE**, **MSE**, or **Norm-in-Norm**. Finally, Figures 6 & 7 are plotted based on the following commands.

```
cd results_in_the_paper
python backbone_performance_plot.py # plot Figure 6
python scatter_plots.py # plot Figure 7
cd ..
```

Listing 8: Commands for plotting Figures 6 & 7

3.3.5 Reproducing Table 1. To reproduce the results of Table 1, we can vary the values of **p**, **q**, and **arch**. The results are printed to the screen by executing the following commands.

```
python main.py --resize -arch resnet18 --p 1 --q 1
python main.py --resize -arch resnet18 --p 1 --q 2
python main.py --resize -arch resnet18 --p 2 --q 1
python main.py --resize -arch resnet18 --p 2 --q 2
python main.py --resize -arch resnet34 --p 1 --q 1
python main.py --resize -arch resnet34 --p 1 --q 2
python main.py --resize -arch resnet34 --p 2 --q 1
python main.py --resize -arch resnet34 --p 2 --q 2
python main.py --resize -arch resnet50 --p 1 --q 1
python main.py --resize -arch resnet50 --p 1 --q 2
python main.py --resize -arch resnet50 --p 2 --q 1
python main.py --resize -arch resnet50 --p 2 --q 2
python main.py --resize --p 1 --q 1
python main.py --resize --p 1 --q 2
python main.py --resize --p 2 --q 1
python main.py --resize --p 2 --q 2
```

Listing 9: Commands for obtaining results of Table 1

3.3.6 Reproducing Table 2. To reproduce the results of Table 2, we can vary the values of **p**, **q**, and **arch**. The results are printed to the screen by executing the following commands.

```
## Training
python main.py --resize # done before
python main.py --resize --alpha 1 0.1
## Testing
python test_dataset.py --dataset KonIQ-10k --resize
python test_dataset.py --dataset KonIQ-10k --resize --
    alpha 1 0.1
python test_dataset.py --dataset CLIVE --resize
python test_dataset.py --dataset CLIVE --resize --alpha 1
    0.1
```

Listing 10: Commands for obtaining results of Table 2

3.3.7 Reproducing Supplemental Results. In this part, we show how to reproducing the supplemental results.

For reproducing Figures A1 & A2, uncomment line 96 of IQAloss.py to print the value of $\hat{b}$ and plot the figures with results_in_the_paper/bhat.py.

```
python main.py --resize > bhat.log 2>&1 &
cd results_in_the_paper
python bhat.py
cd ..
```

Listing 11: Commands for reproducing Figures A1 & A2

For reproducing Table A1, change the optimizer **Adam** to **SGD** or **Adadelta** and other settings are same as the settings for Figure 5(a). For reproducing Table A2, set *-arch* to **alexnet** or **vgg16** and other settings are same as the settings for Figure 6. At last, set *-arch* to **resnet18** and *dataset* to **CLIVE**. Run experiments from exp_id=0 to exp_id=9, and the paired t-test can be conducted based on PLCC values over these ten runs.

3.4 Experimental Summary

We have set a fixed seed 19920517 and discarded randomness as much as possible for reproducible experiments. Based on the reproducing procedure, with the same system environment, we can exactly reproduce all the reported results in the original paper [3]. We also tested the software in different system environments, and a minor difference in the system environment may cause a minor change in the reproduced results. However, the main observations in the original paper are still validated.

4 REVIEWING PROCESS

In the review process, the experimental setup was initiated by reviewers according to the instructions given in the GitHub repository. Then, reviewers performed most of the experiments from the training to the prediction steps to reproduce the results presented in the paper. In addition to training from scratch, pre-generated checkpoints were evaluated by reproducing the results presented in the tables using the checkpoints.

Overall, the software package has clear installation instructions and works as expected. During the review process, several minor issues were found, including availability of data sources used by the software, clarification of version of certain library used by the software, paths of the dataset, improvements to the argument parsing and repository structure. Suggestions were provided to the authors and the issues have been addressed in the revised version.

In conclusion, reviewers and authors worked together for this companion paper. The revised code now enables other researchers to reproduce the experimental results in the original paper as well as build their own solutions on top of it.

5 CONCLUSION

In this paper, we documented the replication of the original paper entitled "Norm-in-Norm Loss with Faster Convergence and Better Performance for Image Quality Assessment". According to the experiments, the results are corresponding to the observations in the original paper. It is reproduced that the Norm-in-Norm loss can consistently facilitate faster convergence and better performance than traditional MAE/MSE loss over different network architectures, datasets, and optimizers due to its embedded normalization operation.

ACKNOWLEDGMENTS

This work was partially supported by National Science Foundation of China under Grants 61572042 and 11961141007 and Sino-German Center for Research Promotion under Grant M-0187. We also acknowledge High-Performance Computing Platform of Peking University for providing computational resources.

REFERENCES

[1] Deepti Ghadiyaram and Alan C. Bovik. 2016. Massive online crowdsourced study of subjective and objective picture quality. *IEEE Transactions on Image Processing* 25, 1 (2016), 372–387.

[2] Vlad Hosu, Hanhe Lin, Tamas Sziranyi, and Dietmar Saupe. 2020. KonIQ-10k: An ecologically valid database for deep learning of blind image quality assessment. *IEEE Transactions on Image Processing* 29 (2020), 4041–4056.

[3] Dingquan Li, Tingting Jiang, and Ming Jiang. 2020. Norm-in-norm loss with faster convergence and better performance for image quality assessment. In *ACM International Conference on Multimedia*. 789–797.

[4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. 8024–8035.