COMP6258 REPRODUCIBILITY REPORT SHUFFLEMIXER

Jos-Elliot Jeapes & Samson Yebio
{jej1g19,sy1c20}@soton.ac.uk

ABSTRACT

This report evaluates the reproducibility of Sun et al. (2022). This was done by attempting to reimplement the model from the information given in the paper, and looking at the output of this new model as well as a pretrained model. The paper is found to be highly reproducible, with the new implementation producing similar results.

1 INTRODUCTION

This reproducibility report focuses on the paper 'ShuffleMixer: An Efficient ConvNet for Image Super-Resolution by' Sun et al. (2022). It applies techniques from Ma et al. (2018) to image upsampling, to produce high-quality upsampled images efficiently. The paper describes a set of models: two families (tiny and standard) that can have different upsample factors (2x, 4x). This paper focuses on the most impressive, the standard 4x model.

1.1 TARGET QUESTIONS

The target questions of this report are:

- Does the model perform as expected when presented with new data?
- Can a model with the same architecture and performance be trained:
  1. Using just the paper and references as a guide?
  2. With reference to the open-source code?

In answer to the above, (1) would show stronger reproducibility than (2). Reproducible aspects of the paper not covered by this report include:

- Complexity and performance compared to competing models,
- Comparison of how model parameters effect performance,
- Effectiveness of the novel aspects of the model.

The first seems outside the scope of this report; it would also be testing the reproducibility of other papers. The others both require multiple training runs of a large model (and so a very large amount of time), and would be uninteresting if the target questions don't show reproducibility.

1.2 METHODOLOGY

Two separate methodologies are used, one for both of the target questions. The first is an attempted reimplementation of the model using just the literature as guidance. The second is running the pretrained model on new data, and comparing the output to the expected results. The reimplemented model is also included in this second comparison, and checked against the original code.

2 REIMPLEMENTATION

The original paper includes a breakdown of the model, fig. 1, showing the model structure at different levels of detail.

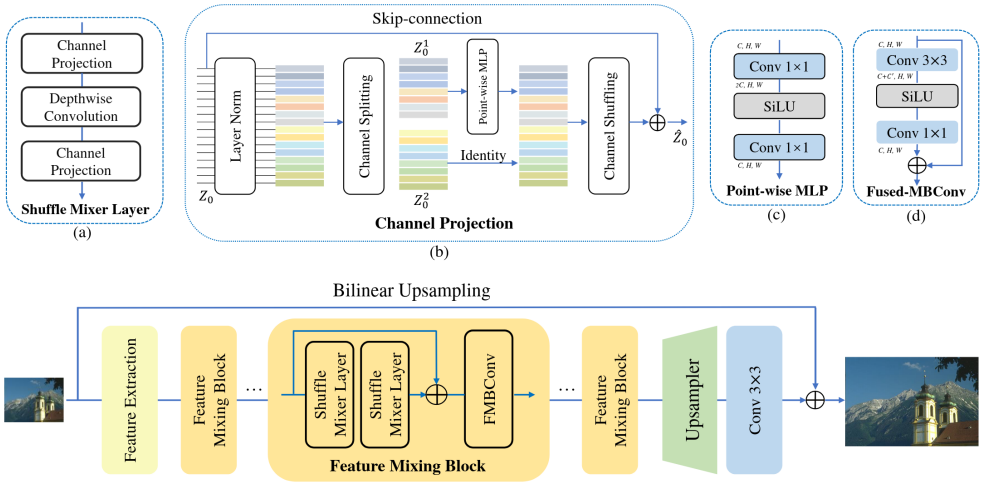

Figure 1: Block diagram showing the structure of ShuffleMixer, from Sun et al. (2022).

This figure was used to organize the reimplementation; each block is written as a PyTorch module. For example, the *Shuffle Mixer Layer* is shown to include two *Channel Projection* blocks with *Depthwise Convolution* in between. In a separate section of the figure, the 'Channel Projection' block is broken down. So, the `ShuffleMixerLayer` module includes two `ChannelProjection` modules.

This makes it easy to quickly confirm that the reimplemented architecture is as described in the paper. A number of details are spread out in the text instead, but fig. 1 is complete enough that this is easy to follow. This includes:

- The *Feature Extraction* block is a single 3×3 convolution layer,
- 64×64 images are used for input during training,
- The *Feature Mixing Block* is repeated five times,
- The *Upsampler* is 'only a convolutional layer of size 1×1 and a pixel shuffling layer'.

The implementation of the *Depthwise Convolution* was based on the definition given in Liu et al. (2022), which is that depthwise convolution is a group convolution with a number of groups equal to the number of input channels.

2.1 Implementation Issues

A number of issues were encountered while reimplementing ShuffleMixer. They are listed here, along with how they were handled.

Channel Shuffling Implementation The model uses a channel shuffling layer, from Zhang et al. (2017). PyTorch includes an implementation of channel shuffling, but it doesn't current support auto-differentiation. There is an open issue discussing this on the PyTorch repository (Sun-WeiZhen, 2021). A different implementation was taken from Kuangliu (2019).

Channel Splitting Ratio For the channel splitting operation, no ratio was explicitly given in the paper. In fig. 1, and in the original definition of the operation (Ma et al., 2018), an equal split is used and so this was chosen.

Missing Parameters & Ambiguity The number of features extracted by the feature extraction layer and the depthwise convolution kernel size are not clearly given in the paper. The following

quote is extracted from a section of the paper describing the difference between ShuffleMixer and ShuffleMixerTiny:

> The number of channels and convolution kernel sizes is 64 and 7×7 pixels for the ShuffleMixer model and 32 and 3×3 pixels for the ShuffleMixer-Tiny model

Which can be mapped on to the aforementioned missing parameters. However, this is rather ambiguous; there are many convolutional layers in the model that the above values could be assigned to. Their meaning had to be inferred.

Padding A sanity check of the structure was done with TorchInfo and unexpected shapes were found. This was a padding issue, and quickly resolved.

FFT Parameters The paper gives a loss function that includes the Fast Fourier Transform, but no parameters are given. As the data is two-dimensional, PyTorch's `fft2` was used with default arguments.

Datasets The dataset the original was trained on is DF2K, which is a combination of the high resolution images from DIV2K (Agustsson & Timofte, 2017) and Flickr2K. No reference could be found for Flickr2K, the reference for it in the original paper (Lim et al., 2017) only mentions DIV2K. There is an issue for this on GitHub, wjuny (2017), but the links given as a solution no longer work. A copy of the dataset was eventually found at Ddlee (2019).

2.2 Comparison with Original

Before training the new implementation, considering some of the issues above, it made sense to sanity check against the original code. Major differences would indicate that the model cannot be implemented just from the literature.

Unfortunately, their implementation does not perfectly follow the block structure from fig. 1. This isn't surprising as a number of the conceptual blocks are very simple and can be merged without decreasing code readability. An extreme example is `FeatureExtraction`, which does not really need to be a separate module as it contains just one layer.

This did make comparing the models slightly more difficult. As an example, the *Layer Norm* and concatenation are split from the *Channel Projection* equivalent in the original code, and happen directly inside their `SMLayer`. However, it seems that the overall architectures are identical.

The parameters mentioned in section 2.1 can all be confirmed as correct. This is the splitting ratio, the depthwise kernel size and number of features. That the loss function uses `fft2` can also be confirmed.

There are some differences, however. Their code also supports x3 scaling, which isn't mentioned the paper. This doesn't affect the x4 model this report examines.

The only affecting difference is that their *Upsampler* finishes with a SiLU layer, counter to the quote given earlier in section 2. This change was carried over to the reimplementation.

2.3 Training

The original model was trained on a V100, with batches of 64. Neither Iridis 5's Lyceum cluster nor the authors' personal desktops could handle batches of 64, 64×64 images along with their 256×256 expected values. Through experimentation, the batch size was reduced to 16.

The training required loading very large images and cropping random 256×256 sections. The 256×256 images are augmented by rotation and flipping, before being downsampled to 64×64.

Using a PyTorch `ImageFolder` data loader, followed the necessary transformations, resulted in extremely slow training. A preprocessing script was written to 4x downsample the original dataset, and a custom data loader was written to load those images and perform the same transformations on both. This halved the training time.

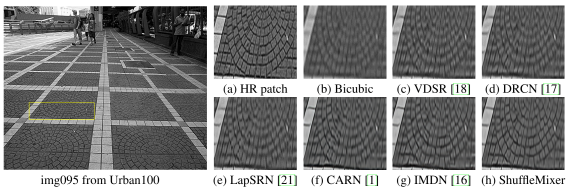

Figure 2: Examples output generated by ShuffleMixer and competitors, from Sun et al. (2022).

The remaining bottleneck is loading the images themselves. A potential, unimplemented, solution would be to change the images to bitmaps, so that the cropped sections could be streamed from the disk rather than loading the whole image.

With the bottleneck being the image loading, there was little difference between training on Iridis and the authors' desktops so the training was done locally. The model was trained for 350,000 iterations (taking about 3 days). This is higher than the original implementation's 300,000 iterations, but the number of examples is lower because of the smaller batch size.

3 TESTING

Examples from the datasets Celeb A HQ (Liu et al., 2015) and Kitti (Geiger et al., 2012) are included in figs. 3 and 4 respectively. These are the first images of those datasets, so not handpicked. Neither of the models were trained on these datasets, and the datasets were chosen specifically because their content is very different: human faces, and outdoor scenes

In both cases, there is a gradual improvement from bicubic upsampling, to the reimplemented model, to the pretrained model. This is most noticeable in areas with straight lines and high contrast. For fig. 3 this is the teeth, eyelashes and hair, and for fig. 4 this is the shadows and the buildings. The roof of the building on the right is straight in both of the model outputs, but pixelated for the bicubic upsampling. Important, this improvement can be also seen in the examples shown in the original paper. Figure 2 is one such example, and shows that ShuffleMixer is much better than bicubic upsampling at defining the gaps between pavement tiles.

It makes sense that the pretrained model is better than the reimplemented one as it has seen almost 4x as many examples. It is clear, however, that ShuffleMixer works better than bicubic upsampling on a variety of (new) data, and that so does the reimplemented version.

4 ANALYSIS

Figure 1 does a lot of heavy lifting when it comes to reimplementing, and therefore reproducing, ShuffleMixer. There were a few issues around vague parameters, but the correct information could all be inferred from the paper even when not explicitly stated. The exception being the missing SiLU layer. However, it isn't unreasonable to expect a reader to assume that a convolutional layer uses the same activation function as every other such layer in the model, not a linear one.

The fact that the pretrained model produced output that was superior, in a similar way to fig. 2, to bicubic upsampling indicates that the model handles new images well. The fact that the reimplemented model produced similar output to the pretrained model is a strong indicator of the reproducibility of ShuffleMixer.

Perhaps the final point related to reproducibility is the issue related to obtaining the dataset, the need to download it from a third party. There is no reason that a different dataset couldn't be used however, as long as it had the same variety and was at least the same size.

Overall, the investigation in this report indicates a high level of reproducibility. Both of the target questions given in section 1.1 (and both (1) and so (2) for the second question) can be answered with a strong yes.

REFERENCES

Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

Ddlee. Cv notes, Sep 2019. URL https://cvnote.ddlee.cc/2019/09/22/image-super-resolution-datasets.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

Kuangliu. Kuangliu/pytorch-cifar, 2019. URL https://github.com/kuangliu/pytorch-cifar/blob/49b7aa97b0c12fe0d4054e670403a16b6b834ddd/models/shufflenet.py.

Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. *CoRR*, abs/1707.02921, 2017. URL http://arxiv.org/abs/1707.02921.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s, 2022.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design, 2018.

Long Sun, Jinshan Pan, and Jinhui Tang. Shufflemixer: An efficient convnet for image super-resolution, 2022.

Sun-WeiZhen. Feature: Add derivative for channel shuffle, 2021. URL https://github.com/pytorch/pytorch/issues/67240.

wjuny. Where can i get flickr2k dataset?, 2017. URL https://github.com/limbee/NTIRE2017/issues/25.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices, 2017.

A EXAMPLE IMAGES

Figure 3: Example images from the first image in the Celeb A HQ dataset (CITE). The original image (source: Liu et al. (2015)) is at the bottom, and the image at the top left is it 4x bicubic downsampled. Then, the top-right, mid-left and mid-right are the output of: 4x bicubic upsampling, the reimplemented model, and the pretrained model respectively.

Figure 4: Example images from the first image in the Kitti dataset (CITE). The original image (source: Geiger et al. (2012)) is at the bottom, and the image at the top is it 4x bicubic downsampled. Then, from top to bottom, the remaining images are the output of: 4x bicubic upsampling, the reimplemented model, and the pretrained model.