Implementing and Evaluating Anchor-based Plain Net for Mobile Image Super-Resolution

Michael Makarenko, Adam Bewick Mulvihill

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Implementing and Evaluating Anchor-based Plain Net for Mobile Image Super-Resolution

Michael Makarenko

Adam Bewick Mulvihill

Instructor: Dr. Subrahmanyam Murala

2023-12-21
A project report submitted in partial fulfilment of the requirements for course CS7GV1 (computer vision)

Declaration

I hereby declare that this project report is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar.

I have completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write.

I consent / do not consent to the examiner retaining a copy of the thesis beyond the examining period, should they so wish (EU GDPR May 2018).

I agree that this thesis will not be publicly available but will be available to TCD staff and students in the University's open access institutional repository on the Trinity domain only, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement. **Please consult with your supervisor on this last item before agreeing and delete if you do not consent.**

Signed:

Michael Makarenko

Adam Mulvihill

Date:

21/12/2023

1. Abstract

The report below presents the investigations, implementation, and findings made throughout the work undertaken by Michael Makarenko and Adam Bewick Mulvihill as part of a 2023 Computer Vision project graded under the CS7GV1 module under the instruction of Dr. Subrahmanyam Murala at Trinity College Dublin. We implemented and evaluated a Single-Image Super Resolution neural network model described in a 2021 paper on high-end modern consumer PC and mobile hardware.

2. Introduction

The paper '*Anchor-based Plain Net for Mobile Image Super-Resolution*' by Du Z. et al.[1] is a computer vision paper focused on developing a network for Super Resolution that is realistically deployable on mobile devices. Through a literature review, the paper gives a brief history of solutions to the problem of Super Resolution and solutions to speed up the networks, and highlights the lack of any solutions suitable for a mobile device. They go on to suggest a network that uses residual learning, and propose Anchor-Based Residual Learning as a method of achieving accurate results with fast performance. In this project, we sought to recreate and understand the network described in the paper, including following the training methods used by the authors of the paper.

3. Related work (Literature review)

Single Image Super Resolution, or SISR, is one of the long standing problems for computer vision. This involves taking a low resolution image and trying to reconstruct a high resolution version. The main issue associated with this is that multiple high resolution images can map to a single identical low resolution image, and trying to narrow the result down to a single high resolution image is difficult to do.

The rise of deep learning has allowed the creation of new methods for SISR using Convolutional Neural Networks, or CNNs, and have achieved dramatic improvements over traditional methods such as those based on interpolation and image statistics, such as ESPCN[2] and EDSR[3]. However, these solutions have their own issues. These networks have high computational loads due to their complexity and depth, with some having up to twenty layers. This meant that despite the improvements these solutions brought, they are very slow to run and train without very powerful hardware.

With the increase in demand for realistic solutions, the focus shifted to making CNNs that are lighter and faster. The solutions to this can be split into explicit and implicit, with explicit solutions using simpler operations to reduce network complexity, while implicit solutions attempt to use intermediate features and information to reduce computational cost. The former explicit solutions generally result in accuracy loss or extra overhead, whereas implicit solutions result in both lower computational costs and better results overall. Examples of two implicit strategies would be hierarchical feature fusion and attention mechanism. While these work well for desktops, these are not so effective for

mobile hardware. Mobile hardware is quite different from a desktop and less powerful, with the previously mentioned methods causing slow RAM access and large time overhead respectively.

One method for increasing the performance and reducing the memory use of a network is quantization. This involves converting a continuous and infinite set of values to a discrete and finite set. Full-integer quantization for example can result in a network that is 3 times faster than a network using floating point numbers. However, there are issues with applying these to SISR solutions, as it will usually result in a significant accuracy drop. This is mainly due to previously mentioned solutions removing batch normalization layers as they result in blurred outputs that often contain artifacts. However, the removal of these layers also results in a high dynamic quantization range. There are few works that focus on solving this problem, which is what this paper focuses on.

The paper authored by Du Z. et al.[1] starts by looking at the latency associated with different types of nodes/layers to figure out which types can be used realistically for their model. The results showed a limited number of nodes were usable. As a result, there was a need to look at the relationship between architecture design and Int8 quantization. The main difficulty as previously described is that many high resolution images can map to the same low resolution images, so the main idea is to create lower standard deviation weights and activations. Two simple ways to achieve this goal include adding a Batch Normalization layer or using residual learning. Since Batch Normalization is integrated into Residual Learning, adding a dedicated layer will just increase memory overhead. Residual Learning can be further subdivided into image-space and feature-space. While feature-space outperforms image-space in floating-point networks, Du Z. et al.[1] argue that an image-space implementation would be better for Int8 quantization as it forces the whole network to learn small residuals. However, both methods of image-space residual learning, using nearest neighbor interpolation and bilinear interpolation, incur severe time costs that are not satisfactory for realistic demand, mainly due to the need for floating point calculations in coordinate mapping. As such, they propose the Anchor-Based Residual Learning (ABRL), which works by directly repeating every pixel in Low Resolution space 9 times, and these are used as anchors for every pixel in High Resolution. This can be realised with a single one channel concatenation layer and a single addition layer thanks to the use of a unique pixel shuffle layer.

4. Proposed work

The first part of the implementation is the network that will be trained. The input to the network is a low resolution image that we want to up-scale using super-resolution. The input first has each pixel repeated 9 times including all channels, and the result is concatenated together, which will allow it to be used in a later layer of the model. The first layer of the network is a convolution layer using 28 3x3 kernels and ReLU activation. This first layer is for shallow feature extraction. This is followed by 5 convolution layers for deep feature extraction, all using 28 3x3 kernels and ReLU activation. This is followed by a single convolution layer used to transfer the features from the previous layers to the high resolution image space. The number of kernels used by this layer depends on both the scale

factor and number of output channels of the high resolution image, and the layer also does not use an activation function. After this layer, the ABRL is applied by adding the residuals to the output of the last convolution layer. Lastly, this is followed by a pixel shuffle layer, and after that a clip node which restricts the values of the output to the range 0-255.

The model is initially trained for 1000 epochs using batches of 16 low resolution images with a size of 64 x 64. The loss function used is the mean absolute error between the outputs of the network and the high resolution versions of the inputs, and the initial learning rate used is 0.001. The learning rate is halved every 200 epochs. At the end of each epoch a callback is used to calculate the PSNR of the model. If the calculated PSNR is the highest recorded so far, the model is saved and the new PSNR is recorded. After the initial training, quantization aware training is used. The model is first set to emulate quantization, and then trained for 200 epochs. The initial learning rate is set to 0.0001 and halves every 50 epochs.

5. Experimental Discussion

We used a GitHub repository to host our implementation and Git as our version control system, which enabled effective collaborative work. Conda environments were set up in the form of .yaml files containing all necessary packages/libraries and their respective versions, including the correct Python and Pip versions, in order to ensure consistency and compatibility across machines when running the code; this has the added advantage of easing first-time setup. All necessary instructions were written in the repository's README.md markdown file.

As far as overall software architecture design goes, we decided to implement as much as possible in one standalone Python script, srmodel.py. We did however find it necessary to separate the code for pre-processing and preparing the dataset into a separate data.py script which is invoked by importing and calling a Data() object for the training and validation datasets respectively. The machine learning library used was Tensorflow for Python.

We used the DIV2K Bicubic x3 dataset[1] to train a model that is eventually stored in a mobile-optimised .tflite file that can be evaluated on the AI Benchmark[2] app on an Android mobile device.

There are ultimately three steps in the process and ultimately three ways of running the code. Running srmodel.py without arguments performs the initial 1000-epoch training; running the script with the `-q` argument performs the 200-epoch Quantization Aware Training given the previously trained model; finally, the `-g` argument generates the .tflite file by quantizing the QAT model and converting it to a mobile optimised INT8 model. This .tflite is then evaluated on the same machine to provide a final average PSNR score over the 100-image validation set. As such, our resulting figures come from the initial training, the QA training, and the final quantized model evaluation both on PC and on mobile. We

[1] https://data.vision.ee.ethz.ch/cvl/DIV2K/
[2] https://play.google.com/store/apps/details?id=org.benchmark.demo

recorded our training results using TensorboardX, printing to the terminal, and using the output from AI Benchmark. The AI Benchmark tests upscaling from 640x360 to 1920x1080.

As training the models takes several hours on even one of the most powerful consumer PC hardware (our test machine used an Nvidia Geforce RTX 4080 GPU), we took care to implement an auto-resumption feature for initial and QA training, whereby if an incomplete model and its respective log file(s) are present, a state.pkl file containing the last best-recorded state of the model and the latest epoch will be read and loaded. This state file is written and saved along with the model files themselves after the completion of every epoch. Since training performance is of utmost importance, we took great care to ensure our implementation was as efficient as possible. To this end, we had the .png files from the dataset converted to .pt files before commencing training to speed up file load times; we also found through experimentation that a `model.train()` parameter of `workers=8` is ideal to take advantage of the parallel processing capabilities of our test machine.

6. Application of the Proposed Method

We provide the GitHub repository containing the source code and all associated documentation to public access under the BSD 3-Clause "New" or "Revised" License.[3]

Please see the below figures for our documented results of the initial training (Figure 1), QA training (Figure 2), quantized .tflite model PSNR evaluation (Figure 3), and quantized .tflite model mobile inference time evaluation (Figure 4).

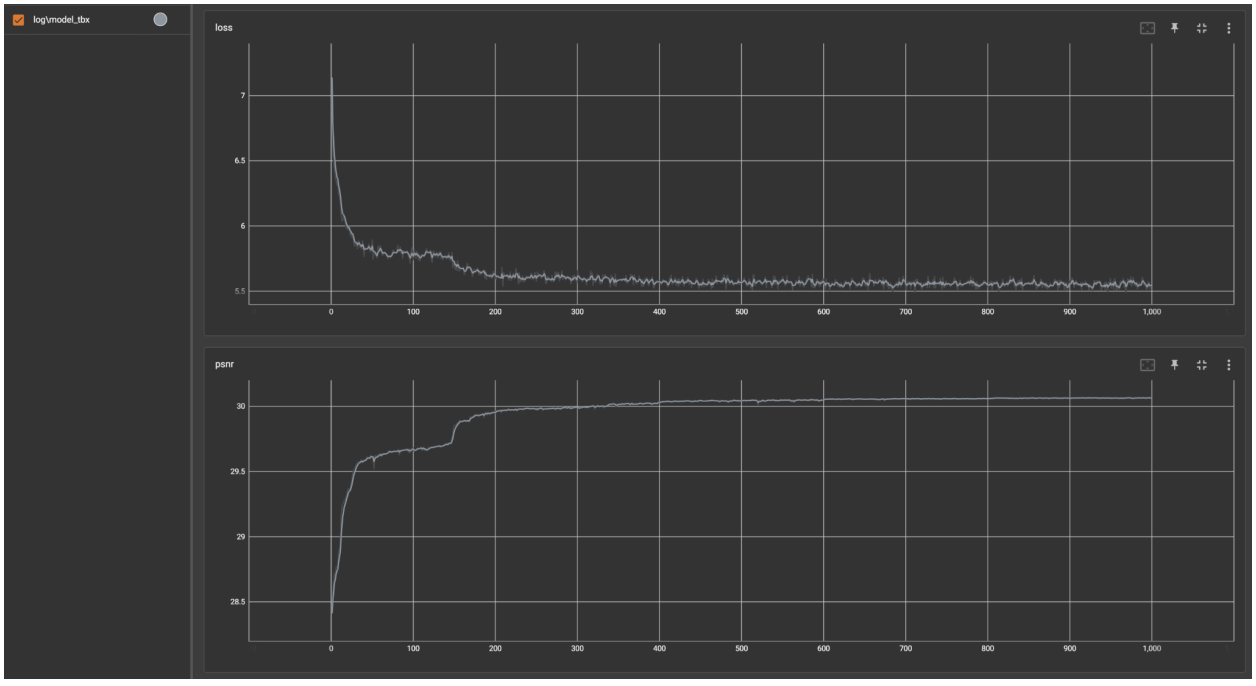

Figure 1: Loss and PSNR graphed for initial training using Tensorboard

[3] https://github.com/A-Mulvihill/TCD-Computer-Vision-Project-2023

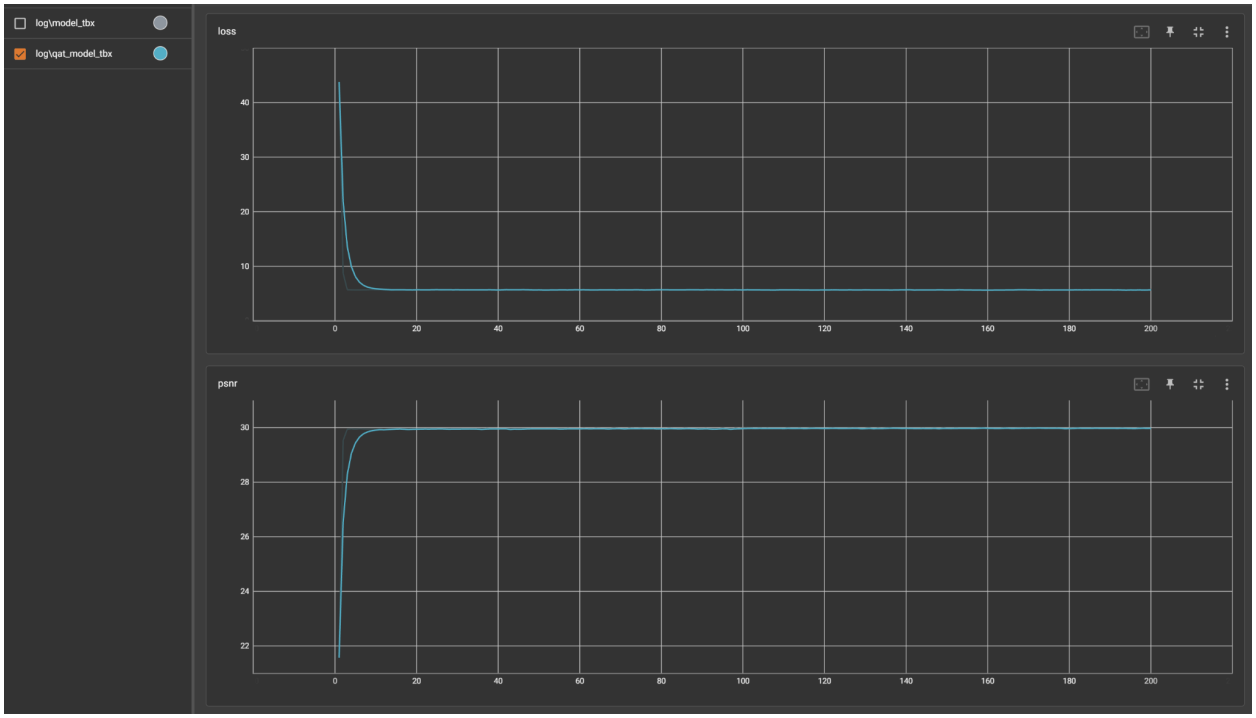

Figure 2: Loss and PSNR graphed for quantization-aware training using Tensorboard

```
❯ python srmodel.py -g
Evaluating using image 898
Evaluating using image 899
Evaluating using image 900
Average PSNR: 29.982325452146384
```

Figure 3: Final Average PSNR score output for .tflite model evaluation

Figure 4: Three sets of inference time results running the .tflite model for 100 iterations in INT8 mode in AI Benchmark on a Qualcomm Snapdragon 888 equipped Android 13 mobile device (Asus ROG Phone 5) using three methods – CPU, GPU, and Qualcomm QNN HTP AI Accelerator.

7. Conclusion

We can observe that our implementation achieved a PSNR score measuring between the two figures in the paper authored by Du Z. et al.[1] for ABPN: 30.15 and 29.87 in the authors' test and in the MAI2021 SISR Challenge results respectively. We therefore conclude that our implemented model's correctness is within a reasonable margin of error and accurately reflects the theory in the paper.

While we were unable to acquire a Qualcomm Snapdragon 820 equipped device for testing mobile inference times, we achieved results on the Snapdragon 888 chipset measuring ~25% faster on the CPU, ~30% faster on the GPU, and ~273% faster on the Neural Net Accelerator, although it is worth noting that the latter differs significantly in implementation as the Snapdragon 820 uses the Android NNAPI while the 888 uses the Qualcomm QNN HTP. We therefore conclude that our implementation scales well on newer and more performant mobile hardware, particularly when it comes to dedicated AI accelerators.

We recommend further testing on a variety of mobile chipsets as an area for future work.

8. References

[1]: Du, Z., Liu, J., Tang, J., & Wu, G. (2021). *Anchor-based Plain Net for Mobile Image Super-Resolution*. Cornell University arXiv.org Electrical Engineering and Systems Science. [Online]. DOI: https://doi.org/10.48550/arXiv.2105.09750

[2]: Shi, W., Caballero, J., Husz´ar, F., Totz, J., & Wang, Z. (2016). *Real-time single image and video superresolution using an efficient sub-pixel convolutional neural network*. Cornell University arXiv.org Computer Science. [Online].
DOI: https://doi.org/10.48550/arXiv.1609.05158

[3]: Lim, B., Son, S., Kim, H., Nah, S., & Lee, K.M. (2017). *Enhanced deep residual networks for single image super-resolution*. Cornell University arXiv.org Computer Science. [Online].
DOI: https://doi.org/10.48550/arXiv.1707.02921