SESR: Super-Efficient Super-Resolution

Thomas Christie
thwc3

Viktor Mirjanic
vvm22

Abstract

Super-resolution is an active research area, which is dominated by neural network models. The ability to perform real-time super-resolution on constrained hardware, such as mobile phones, remains a challenge. In this project we re-implemented SESR [17], the state-of-the-art super-resolution model that achieved top scores in both accuracy and computation cost, which was published in the MLSys 2022 Conference. We also extended the model by applying pruning, which we found enabled us to reach *higher* accuracies with *fewer* iterations of training. Our code is available on https://github.com/Vlam1r/sesr-jax, with key branches detailed in this report.

1 Introduction

Super-resolution is a computer vision problem with the goal of restoring detail to low-resolution sources. The case in which the source is a single low-resolution (LR) image is known as Single Image Super-Resolution, or SISR. This problem can be solved efficiently with classical algorithms such as bicubic or bilinear interpolation [15], but they have poor performance. On the other hand, the best SISR models, such as FSRCNN [4] and TPSR [16], use complex architectures like deep CNNs or GANs.

As 4K and larger displays are becoming ubiquitous, the need for more efficient upscaling models has arisen. In particular, recent research has sought to achieve the performance capabilities of the best SISR models, whilst being efficient enough to run on hardware found in devices such as TVs and laptops. This becomes even more challenging when trying to perform inference in real-time; for displays playing video at the standard 30 frames per second, this requires inference to take place in $\sim$ 33ms.

2 SESR

Super-Efficient Super-Resolution (SESR) [17] is a new model designed with efficiency in mind. It offers a new method for improved picture quality at speeds suitable for real-time upscaling. The key to achieving these results is the use of collapsible blocks.

2.1 Collapsible Blocks

Collapsible blocks are a compromise between using many convolution layers to increase accuracy, and using shallow models to maximise speed. Each collapsible block consists of several consecutive convolutional layers without intermediate activations. This use of several different layers, in spite of the lack of activations between them, is known as linear overparameterisation [8], and has been demonstrated to enable neural networks to be trained more quickly.

Submitted project report for L46 (Principles of Machine Learning) course

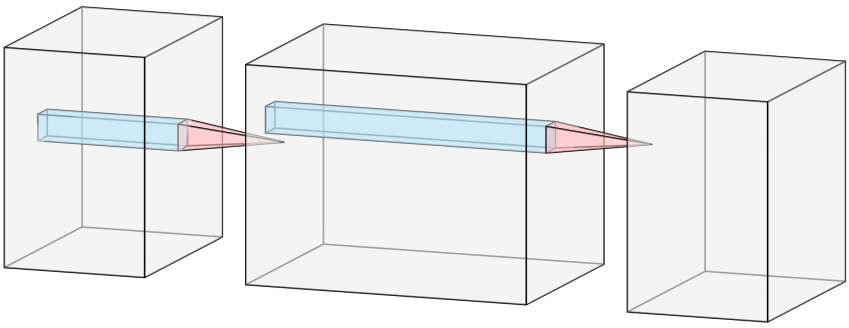

Figure 1: *Collapsible block with two consecutive convolutional layers without intermediate activations. The inner representation with a large number of channels will disappear after collapsing.*

The input and output of the collapsible block have the same shape with just a few channels. The intermediate layers, however, have a much larger channel count. Having no intermediate activations means that the blocks can be analytically collapsed (hence the name) into a single convolutional layer.

2.2 Model Overview

SESR features many residual connections, most of which are placed in a way that lets them be analytically collapsed as well. Having residual connections increases the amount of intermediate results that need to be stored, so collapsing those connections reduces the memory requirements of the model.

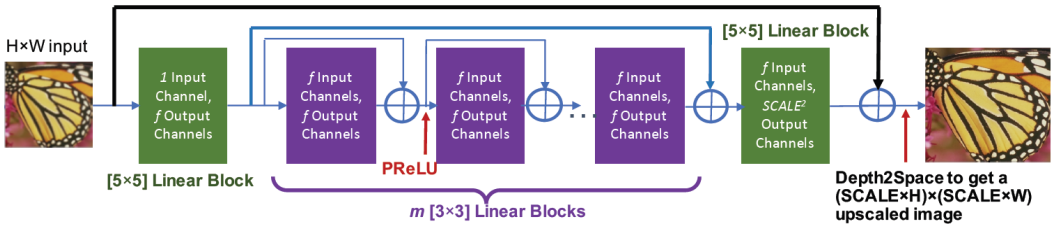

Figure 2: *Overview of the entire model. Green and purple squares represent collapsible (linear) blocks. Figure taken from [17].*

One important feature of the collapsible blocks is that their input and output dimensions are the same. This means that the model can have a varying amount of them. This is a model parameter, and can be tuned to the capabilities of the target hardware and the desired performance. Values typically range from 3 to 11. We follow the same naming convention as [17], and name models according to the number of collapsible blocks (i.e. an M3 model contains 3 collapsible blocks). Another model parameter is the number of channels in each intermediate step, and is typically chosen to be 16 [17].

The model output keeps the same width and height as input, but has 4 channels instead of 1. As the model upscales images by a factor of 2, the total pixel count needs to be quadrupled, and so channels are interpreted as quarters of each original pixel.

3 Implementation

3.1 JAX

We decided to re-implemenent the SESR model from scratch to help with reproducibility. The original paper already had code in TensorFlow, so we decided to write ours in a different framework. We picked JAX [10], as it is a novel framework, and we wanted to gain experience with it.

JAX is a framework created by Google for ML research, and therefore has many benefits compared to other frameworks. It re-implements NumPy [14] with XLA, a domain-specific compiler for linear algebra acceleration. This means that JAX offers both a familiar interface and competitive performance

out of the box. Other features of JAX that we relied on include just-in-time compilation to further speed up CPU code, and robust determinism when using randomness to increase reproducibility of our work.

We also used Haiku [13], a low level framework for neural networks built on top of JAX by DeepMind.

3.2 Model Structure

When implementing the model, we decided to rely on the original TensorFlow implementation as little as possible, and instead use ideas from the paper directly. We implemented two separate models, `SESR`, which used expanded weights, and `SESR_Collapsed` which used collapsed weights. While training, weights would be stored in expanded form for back propagation. However, for each forward pass the weights would be collapsed, and fed into `SESR_Collapsed`. This meant that we never really needed to execute the expanded `SESR` model; neither in the training loop nor afterwards. However, having it gave us more flexibility and allowed us to easily compare the two models. Specifically, we used the expanded model to compute the gains of all optimisations that we implemented into the collapsed version.

3.3 Collapsing Algorithm

In JAX, the model parameters are exposed directly as a dictionary, and therefore manipulating them is easy. The collapsing procedure is based on this simple identity (informal):

$$A \star W_1 \star W_2 = A \star (I \star W_1 \star W_2)$$

W_1 and W_2 are expanded weights of two convolutional layers in the collapsible block, while $I \star W_1 \star W_2$ are collapsed weights of a single convolutional layer that replaces them. Therefore, the effect of two sequential convolutions can be precomputed and then applied directly. For preprocessing, we reused the very collapsible block that we are collapsing. The effect of applying the identity matrix to the collapsible block produces the collapsed weights.

Residual connections can be similarly collapsed into a single convolution:

$$A \star W + A = A \star (W + I)$$

3.4 Issues and Re-implementation

We implemented the collapsing algorithm to generate collapsed weights from the expanded ones. Then we noticed a discrepancy between the outputs of the two models – the collapsed model was producing completely different results. We traced the discrepancy to the use of bias terms in the convolutions. Namely, the collapsing algorithm didn't handle them properly.

We consulted the authors' original implementation to discover that their collapsing strategy was different from ours. Our algorithm was taking the weights from the expanded model and producing weights for the collapsed one, at least when there were no bias terms. The authors' algorithm was also producing weights for the collapsed model. However, its inputs couldn't be used as the weights of the expanded model. We reimplemented their version in JAX as well, so that we could compare it with ours. Then we discovered that it didn't correctly handle bias terms either, and this was in fact an open issue on the authors' GitHub repository [1].

We could have avoided the problem by not including biases at all, but we didn't want to do that as we suspected that it would have an impact on accuracy. Furthermore, we wanted to improve our collapsing version as it was more modular. With it, both expanded and collapsed models can be learned in a single training run, while the authors' version needs to train the expanded and collapsed models separately.

[1] https://github.com/ARM-software/sesr/issues/14

Algorithm 1 Collapse Convolutions With Proper Bias Generation

1: **procedure** COLLAPSE_LB($W_1, b_1, W_2, b_2, N_{in}, kernel\_dim$)
2: $\Delta \leftarrow$ eye(N_{in})
3: $\Delta \leftarrow$ expand_dims(expand_dims(Δ, 1), 1)
4: $pad \leftarrow kernel\_dim/2$
5: $\Delta \leftarrow$ pad($\Delta, [0, pad, pad, 0]$)
`// Finished preparing identity matrix for input`
6: $W_C \leftarrow$ Conv2D($\Delta; W_1, b_1$)
7: $W_C \leftarrow$ Conv2D($W_C; W_2, b_2$)
8: $W_C \leftarrow$ flip($W_C, (1, 2)$)
9: $W_C \leftarrow$ transpose($W_C, (1, 2, 0, 3)$)
10: $b_C \leftarrow$ reshape($b_1, (1, 1, 1, -1)$)
11: $b_C \leftarrow$ Conv2D($b_C; W_2, b_2$)
12: $b_C \leftarrow$ flatten(b_C)
13: **return** W_C, b_C
14: **end procedure**

Eventually, we revised the collapsing algorithm to work correctly with biases. We present our algorithm as Algorithm 1. Lines 10-12 are our addition to the original model. This algorithm is based on the following equality:

$$(A \star W_1 + b_1) \star W_2 + b_2 = A \star \underbrace{(I \star W_1 \star W_2)}_{W_C} + \underbrace{\texttt{expand}(b_1) \star W_2 + b_2}_{b_C}$$

This is why we provide 2 branches in our code-base. The `new-sesr` branch contains a JAX re-implementation in the style of the authors' TensorFlow code, while `master` contains our original modular implementation with fixes to work properly with biases.

3.5 Optimisations

We added multiple optimisations to improve training time. Similarly to [17], we collapsed weights in the training loop and used the faster collapsed model for the forward pass. We kept the expanded weights to use them for backpropagation.

The expanded model has $2m + 4$ consecutive convolutional layers. On the other hand, the collapsed model has only $m + 2$ convolutions, and the collapsing procedure requires executing $2m + 4$ convolutions on a single input. However, this is a one-time cost, and these convolutions can be parallelised as well. As this optimisation nearly halved the number of convolutions, we expected that it would give nearly a 2x speedup.

We also extensively profiled our training loop, and JIT-compiled as much of the CPU code as possible.

3.6 Data Processing

We followed the same data processing steps as in the original paper, for accurate comparison of our obtained results. We used the Div2K [7] dataset, which is comprised of 800 training images and 100 test images. The data consists of low resolution image inputs and corresponding upscaled targets. We focused on the task of $\times 2$ (e.g. 1080p to 4K) super-resolution, but changing to the task of $\times 4$ super-resolution would only require minor changes to the final layers of our model, as noted in [17].

The images in the dataset are defined in the RGB colour space. However, the SESR paper [17] notes that within the super-resolution literature, the standard practise is to convert images from the RGB colour space to the YCbCr colour space, and only use the Y channel for super-resolution [5]. In the YCbCr colour space, the Y channel represents luminance, whilst the Cb and Cr channels represent the intensity of the blue and red components of the image relative to green respectively. The human eye is more sensitive to changes in luminance compared to hue [6], and so it makes sense to use the model to upsample only the Y channel for efficiency. Metrics reported in this report correspond to the Y channel only, as is done in the SESR paper [17].

When reconstructing images for a qualitative analysis, we used the trained model to perform super-resolution on the Y channel, and then used bicubic interpolation [15] for upsampling the Cb and Cr channels, before converting back from YCbCr colour space to RGB.

We used the same parameters as the original SESR paper for training. The SESR models were trained for 300 epochs in total, using the AMSGrad optimiser [2] (favoured over ADAM [2] due to its convergence guarantees in cases where ADAM may fail to converge) with a learning rate of 5×10^{-4}. In terms of data augmentation, we took 64 random crops of size 64×64 from each image in the dataset, as done in [17], and used a batch size of 32. We used the mean absolute error between the predicted super-resolution image and the target image as the loss function. For evaluating the performance of the network we used the de facto super-resolution metric, peak signal-to-noise ratio (PSNR), on the Y channel of the produced image. Using mse to denote the mean squared error of the pixel values in model's output image, PSNR is defined as:

$$\text{PSNR} = 20 \times \log_{10}(\frac{1}{\sqrt{mse}}) \tag{1}$$

4 Pruning

Pruning methods have existed for a long time (e.g. [1]), and are grounded in the idea that neural networks often contain parameters which aren't necessary, and can be set to zero without harming the performance of the model. Different criteria can be used to decide which weights to prune, but in practise even simple *magnitude pruning*, whereby the weights with the smallest absolute values are removed, has been shown to work well (e.g. [9].

One particularly interesting result within the recent pruning literature is the *lottery ticket hypothesis* [9]. This paper found that if, after pruning, unpruned weights had their values reset to the random values they were initialised with, then the resulting pruned network was easier to train from scratch than if the weights were randomly reinitialised. Furthermore, these pruned networks would often go on to obtain higher accuracies than the original unpruned model, whilst requiring less iterations of training to achieve these performance levels.

Subsequent work [12] found that *rewinding* unpruned weights to their values from a few iterations into training $(0.1 - 7\%)$, rather than *resetting* them to their randomly initialised values could result in improved performance on more challenging tasks, with more complex architectures. We decided to implement this pruning strategy in the hope of speeding up the training process, as well as potentially improving the PSNR value obtained by the model. A summary of the algorithm from [12] which we used can be found below. Note that a pruned model is defined as having parameters θ and a binary mask M, such that the pruned model parameters are formed from the element-wise product of the parameters and the mask, $\theta \odot M$.

Algorithm 2 Iterative Magnitude Pruning with Rewind

1: Initialise model with parameters θ_0 and mask $M = 1^{|\theta_0|}$.
2: Train the network for k iterations to obtain parameters θ_k.
3: Train the network for $N - k$ further iterations to obtain parameters θ_N.
4: Prune the smallest $p\%$ ($smallest\_p$) unpruned parameters from θ_N such that if $\theta_N[i] \in smallest\_p$ then $M[i] = 0$.
5: If you wish to prune more parameters, rewind the model parameters to θ_k and return to step 3 with the updated mask M. Otherwise terminate with the final parameters θ and mask M.

The SESR model has the additional complication of having a collapsed set of parameters and an expanded set of parameters. We generated a mask for the collapsed set of parameters, since these parameters are used in the forward pass of the network. In practise, we found that pruning was surprisingly effective, and lead to the model reaching a higher PSNR score with less iterations of training, as is reported in our evaluation. The implementation of pruning can be found on the `pruning` branch of the repository.

[2] https://optax.readthedocs.io/en/latest/api.html#optax.amsgrad

5 Evaluation

5.1 Comparison to SESR Reported Performance

For the main comparison to the author's original implementation, we trained an M3 model for 300 epochs using the same hyper-parameters as reported in the SESR paper [17]. Our model achieved a PSNR value of 34.915, whilst the authors reported a value of 35.03. We were pleased to obtain such similar performance, but are curious as to how the extra 0.115 PSNR could be achieved. There are two key areas which we suspect could be responsible for this discrepancy in performance:

1. **Collapsing Convolutions** - As noted in section 3.4, we found that the authors' code didn't handle biases correctly when collapsing blocks. The model we trained used the revised algorithm we present in 1, and as such this could lead to a difference in performance.

2. **Data Augmentation** - In the SESR paper, the authors stated that for each image in the training set, they took 64 random crops. As such, we followed the same approach. However, upon inspection of their code, we found that they also would randomly flip and rotate these images, which wasn't explicitly referenced in their paper. Perhaps adding these additional transformations could improve performance, but it is hard to say, as it wouldn't actually increase the size of the training set.

5.2 Pruning

In order to evaluate the effectiveness of the pruning strategy detailed in algorithm 2, we pruned an M3 model for 10 iterations in total, pruning 20% of the remaining parameters at each iteration. After pruning, we would rewind the unpruned parameters of the network back to their values from 5 epochs into training, and train the model for a further 50 epochs. Note that the original SESR paper trained its models for 300 epochs, but we only trained for 50 epochs at a time in this experiment as we found that the majority of performance was achieved within the first epochs of training, and even with 50 epochs of training this experiment took around 3hrs 40mins on an Nvidia A100 GPU on Google Colab Pro.

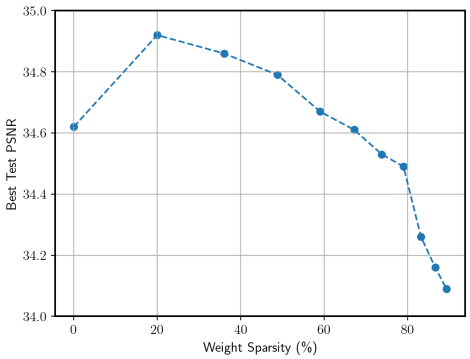

Figure 3: *Sparsity of M3 model and the best test PSNR achieved within 50 epochs of training the model. Pruning enables the model to achieve higher PSNR scores within 50 epochs of training; the best score of 34.92 is achieved by the model with 20% of its parameters pruned. It was only after pruning more than 67.2% of the parameters that the PSNR score dropped below that obtained by the unpruned model.*

Figure 3 shows the best PSNR score achieved by the pruned M3 models as the sparsity increased. For reference, the authors report a PSNR score of 35.03 for their unpruned M3 model trained for 300 epochs, 6× the number of epochs used to train the models in this experiment. Interestingly the best PSNR score was achieved by the model which had 20% of its parameters pruned, and it was only after pruning more than 67.2% of parameters that the PSNR score dropped below that obtained by the unpruned model. The combination of pruning and rewinding unpruned parameters to their values from 5 epochs into training seemed to enable the model to achieve a *higher* PSNR score with *fewer* iterations of training, as shown in figure 4.

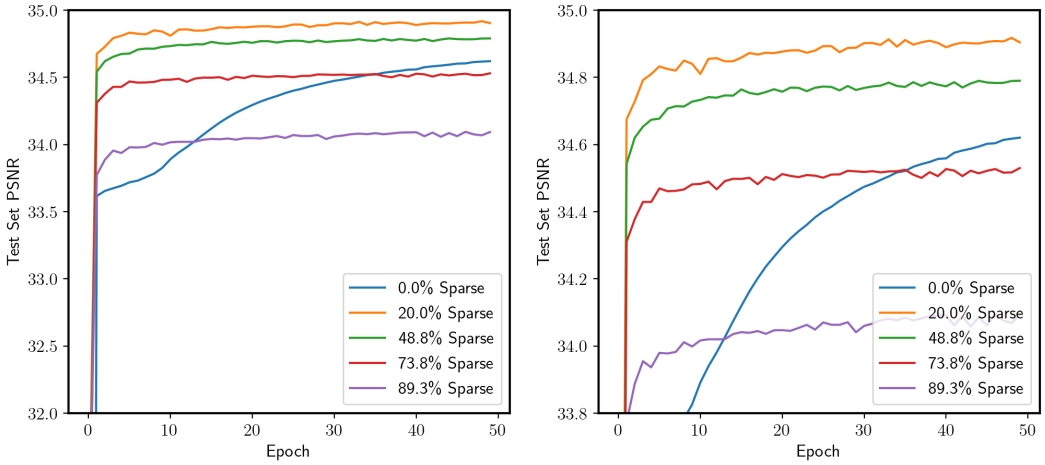

Figure 4: ***Left**: Progression of test PSNR achieved by models of varying sparsity over 50 training epochs. **Right**: Same plot, but zoomed in. Pruned models achieve higher PSNR scores with fewer iterations of training.*

5.3 Qualitative Analysis

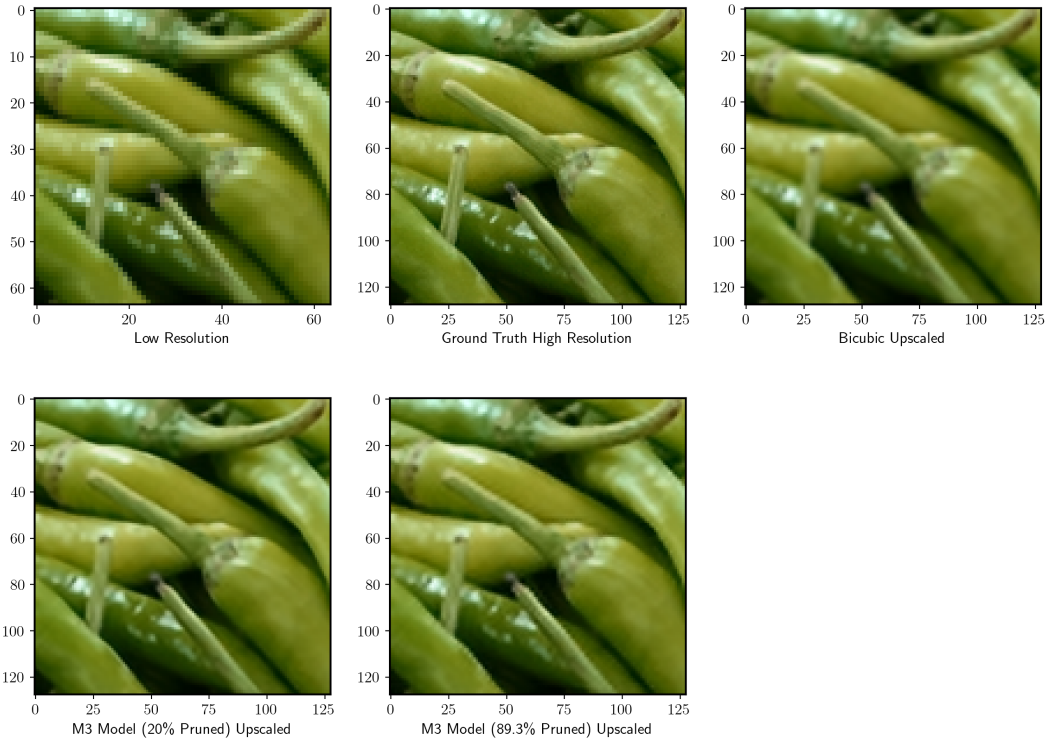

Figure 5: *Comparison of super-resolution techniques performed on a section of an image containing chillis. Note that the low resolution image is the input to the model.*

Figure 5 shows a crop of an image containing chillis, up-scaled using several techniques. The top-left panel shows the low-resolution image which is the input to the model, and the top-middle panel shows the ground truth high resolution image. The M3 model qualitatively generates clearer images than bicubic interpolation, and more examples can be seen in appendix A. Examples can also be

viewed in a Jupyter notebook [3], where the differences may be easier to see. The M3 model with 89.3% of its parameters pruned competes fairly well with the best performing M3 model (with only 20% of its parameters pruned), but a little less clarity is evident around the edge of objects.

5.4 Performance

As one of the goals of the SESR model is high efficiency to support real-time upscaling, we evaluated the performance of our model, with and without our optimisations.

We discovered that, when training, running a forward pass on the collapsed model instead of the expanded one speeds up computation by 1.7x. This matches our analysis above where we argued that collapsed model should be approximately 2 times faster. Other optimisations produced another 2x speedup. Together, these optimisations made our training loop execute faster than that of the original authors. On a single RTX 3060 Laptop GPU, our code completes one epoch without pruning in 50-60s, while the code from [17] needs 60-70s per epoch. At 300 epochs, we saved approximately 1 hour when training. As we have already seen, pruning further reduced the training time by greatly reducing the number of epochs needed to train.

6 Future Work

We were surprised at how much of an impact the pruning had on the model's performance and trainability, and believe this could be a fruitful avenue to pursue further. For instance, whilst we pruned the weights with the smallest magnitude, this can result in sparse convolutional kernels, which often don't translate to significant speedups in inference in hardware. It would be interesting to try using a pruning approach which removes whole filters, such as Fisher pruning [11], rather than individual weights, and see if this could lead to the increase in trainability we witnessed with magnitude pruning, as well as more efficient inference in hardware.

Another technique which would be interesting to implement is knowledge distillation [3]. Due to the architectural similarities, it would be interesting to take a larger M11 model, with its improved PSNR score, and use its predictions as a target for a smaller M3 model during training, in combination with the ground truth. This could further increase the PSNR score of the smaller model, whilst having the advantage of faster inference achieved through the reduced number of collapsible blocks.

Finally, it would be interesting to take the resulting models and evaluate their performance on realistic hardware, such as TVs, to see whether real-time super-resolution would be feasible.

7 Conclusion

To conclude, in this project we have taken a state-of-the-art approach to super-resolution, SESR [17], and re-implemented it in a different framework to that of the authors. Whilst implementing the model, we discovered an issue in their algorithm which caused the biases of the collapsible layers to be handled incorrectly, which we resolved in our implementation. We then investigated whether iterative magnitude pruning introduced by [9] would enable our network to train more quickly and reach a higher accuracy, and found this approach to be surprisingly effective.

[3] https://github.com/Vlam1r/sesr-jax/blob/pruning/data_visualisation.ipynb

References

[1] Yann LeCun, John Denker, and Sara Solla. "Optimal brain damage". In: *Advances in neural information processing systems* 2 (1989).

[2] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[3] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* 2.7 (2015).

[4] Chao Dong, Chen Change Loy, and Xiaoou Tang. *Accelerating the Super-Resolution Convolutional Neural Network*. arXiv:1608.00367 [cs]. Aug. 2016. DOI: 10.48550/arXiv.1608.00367. URL: http://arxiv.org/abs/1608.00367 (visited on 01/13/2023).

[5] Chao Dong, Chen Change Loy, and Xiaoou Tang. "Accelerating the super-resolution convolutional neural network". In: *European conference on computer vision*. Springer. 2016, pp. 391–407.

[6] Neethu John et al. "Analysis of various color space models on effective single image super resolution". In: *Intelligent Systems Technologies and Applications*. Springer, 2016, pp. 529–540.

[7] Eirikur Agustsson and Radu Timofte. "Ntire 2017 challenge on single image super-resolution: Dataset and study". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2017, pp. 126–135.

[8] Sanjeev Arora, Nadav Cohen, and Elad Hazan. "On the optimization of deep networks: Implicit acceleration by overparameterization". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 244–253.

[9] Jonathan Frankle and Michael Carbin. "The lottery ticket hypothesis: Finding sparse, trainable neural networks". In: *arXiv preprint arXiv:1803.03635* (2018).

[10] Roy Frostig, Matthew James Johnson, and Chris Leary. "Compiling machine learning programs via high-level tracing". In: *Systems for Machine Learning* 4.9 (2018).

[11] Lucas Theis et al. "Faster gaze prediction with dense networks and fisher pruning". In: *arXiv preprint arXiv:1801.05787* (2018).

[12] Jonathan Frankle et al. "Stabilizing the lottery ticket hypothesis". In: *arXiv preprint arXiv:1903.01611* (2019).

[13] Igor Babuschkin et al. *The DeepMind JAX Ecosystem*. 2020. URL: http://github.com/deepmind.

[14] Charles R Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (2020), pp. 357–362.

[15] Donya Khaledyan et al. "Low-Cost Implementation of Bilinear and Bicubic Image Interpolation for Real-Time Image Super-Resolution". In: *2020 IEEE Global Humanitarian Technology Conference (GHTC)*. IEEE. 2020, pp. 1–5.

[16] Royson Lee et al. *Journey Towards Tiny Perceptual Super-Resolution*. arXiv:2007.04356 [cs, eess]. July 2020. DOI: 10.48550/arXiv.2007.04356. URL: http://arxiv.org/abs/2007.04356 (visited on 01/13/2023).

[17] Kartikeya Bhardwaj et al. "Collapsible linear blocks for super-efficient super resolution". In: *Proceedings of Machine Learning and Systems* 4 (2022), pp. 529–547.

A Further Super-Resolution Examples

Figure 6: *Image of a stall selling chillis, with the crop taken for super-resolution shown in red.*

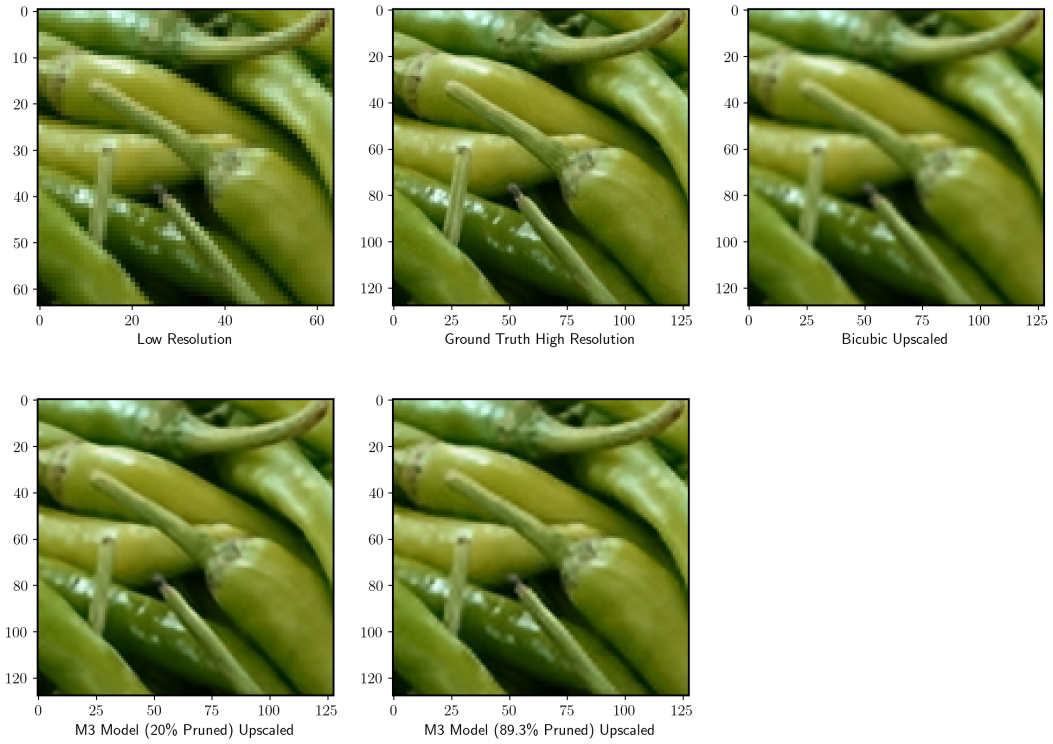

Figure 7: *Comparison of super-resolution techniques performed on the cropped section of chillis from figure 6*

Figure 8: *Image of a sculpture, with the crop taken for super-resolution shown in red.*

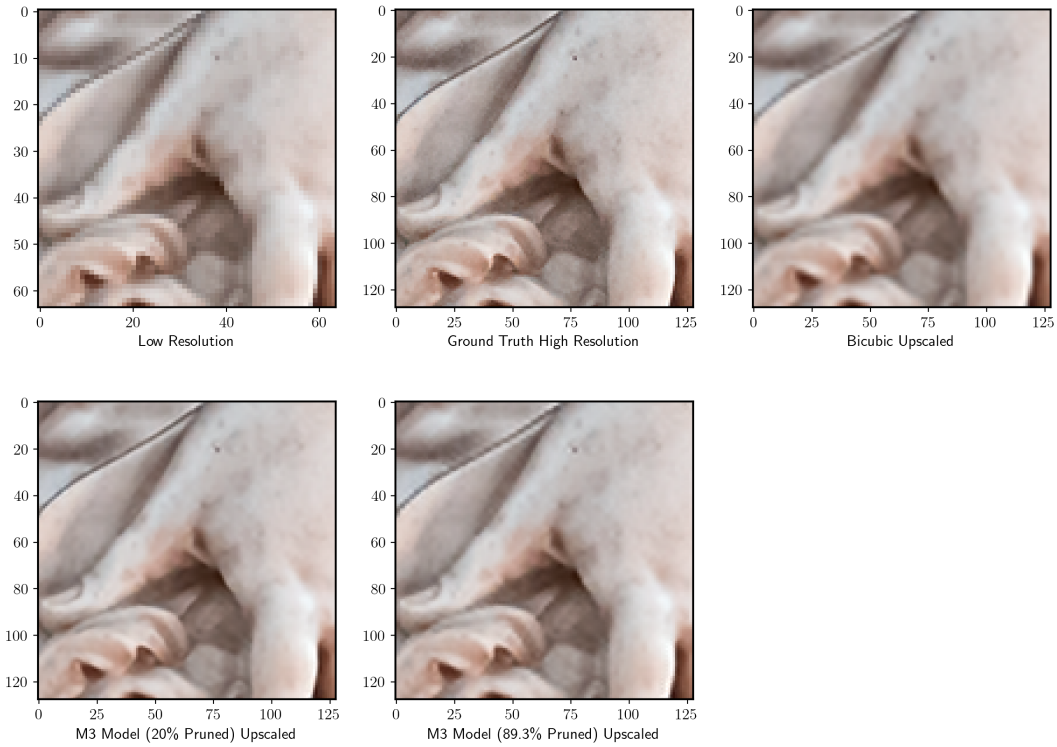

Figure 9: *Comparison of super-resolution techniques performed on the cropped section of the sculpture from figure 8*