UNDERSTANDING OPPORTUNITIES FOR EFFICIENCY IN SINGLE-IMAGE SUPER RESOLUTION NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

A successful application of convolutional architectures is to increase the resolution of single low-resolution images – a vision task called super-resolution (SR). Naturally, SR is of value to resource constrained devices like mobile phones, electronic photograph frames and hoe televisions to enhance image quality. However, SR demands perhaps the most extreme amounts of memory and compute operations of any mainstream vision task known today. And this in-turn prevents SR from being deployed to devices that require them. In this paper, we perform one of the only systematic studies of system resource efficiency for SR, within the context of a variety of architectural and low-precision approaches originally developed for discriminative neural networks. We present a rich set of insights, representative SR architectures and efficiency trade-offs; for example, highly compact SR models suitable for DSPs and FPGAs – along with SR models suitable for smartphones that are 3x smaller than those of comparable quality found in the literature. Collectively, we believe these results provides the foundation for further research into the little explored area of resource efficiency for SR.

1 INTRODUCTION

Rapid progress has been made in the development of convolutional networks such as Kim et al. (2015); Dong et al. (2015); Lim et al. (2017) that are capable of taking as input a low-resolution image, and producing as output an image with a significant increase in resolution (2x/4x/8x) – yet with a quality much beyond that of conventional methods like bicubic interpolation. This vision task referred to as super-resolution (SR) and has many potential applications in devices with limited memory and compute capacity. The fundamental problem however is that the networks that perform SR are some of the most resource intensive networks currently known. For example, RCAN by Zhang et al. (2018) has more than 2,000 layers – and so dwarfs more familiar networks for tasks like object recognition that are at most hundreds of layers deep. Furthermore, because such SR seeks to produce images of large dimensions – its requires an architectures do not perform pooling operations to scale down matrices; rather the opposite is done in the form of expensive up-scaling modules performed as the network progresses deeper. As a result, current SR models are completely unable to execute locally on constrained devices (televisions, cameras, phones, electronic picture frames).

The challenge of the system resource requirements for deep learning models for tasks other than SR (such as, speech recognition and scene understanding) have been carefully studied in works such as Zhang et al. (2017); Howard et al. (2017). In fact remarkable reductions in memory and compute have been achieved with gains of 50x in efficiency being found to be possible – with a wide variety of methods being developed grounded in primarily architecture-level changes, as well as techniques grounded in the use of low precision and quantized model parameters. But how these efficiency methods behave when applied within SR have yet have been studied in significant depth, with very few results available in the literature. Extrapolating from prior results for other tasks is problematic given predominantly existing studies are applied to discriminative tasks with substantially different architectures and operation. Even worse, as SR are routinely 4x to 10x larger than architectures for other vision tasks techniques to reduce resources have to be applied even more intensely than normal, and therefore produce potentially stronger side-effects to model quality.

In this paper, we detail the most systematic study performed thus far that seeks to bridge current understanding in SR and known approaches for scaling down the consumption of system resources

by deep models. By examining in controlled micro-architectures the impact on SR quality when performing individual techniques (such as 8-bit quantization, 1x1 convolutions, group convolutions, ternary operators), we produce key insights as to which are most effective and how they may complement each other towards larger efficiency gains. We demonstrate the value of such insights by devising representative SR models, each with important resource trade-offs that fit key device deployment scenarios. For instance, when considering compute cost we identify an SR solution with up-scaling quality comparable to Tai et al. (2017) yet requires 6x fewer operations. Alternatively, if concerned more about model size we devise a SR model that is comparable to Ahn et al. (2018) in quality by only 2.5x smaller. Even more impressive, Ahn et al. (2018) is the most efficient form of SR we discovered in the existing literature. In sum, the core scientific value of this work is found in a comprehensive mixture of empirical results that are synthesized into best practices – and tested in the form of various SR models each with valuable absolute and relative performance gains over existing models previously described. This paper offers a foundation that will accelerate the ability of other researchers can investigate this key challenge of performing high-efficiency SR suitable for a variety of constrained compute platforms.

2 SYSTEMATIC STUDY OF LOW-RESOURCE SUPER RESOLUTION NETWORKS

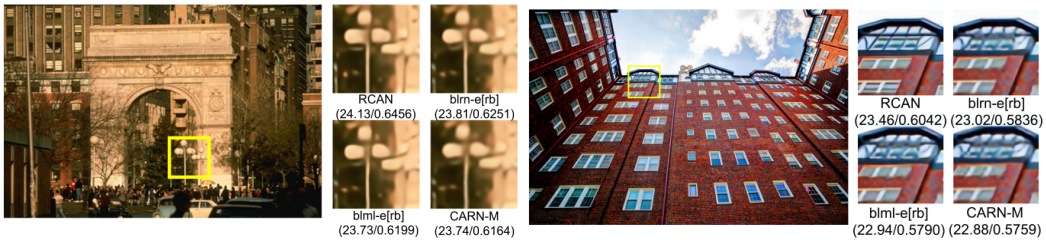
The key step in our work is to build understanding towards building resource-efficient architectures for super-resolution. While there is a lot of understanding of how these efficiency-saving techniques work in classification problems, there is a lack of experimental studies and systematic approaches to understand their practicality in super-resolution. To our knowledge, this is the first systematic study of wide range efficiency methods on super-resolution. The key aim of this paper is to understand and outline a few design choices through empirical observations from a series of devised experiments over the next two sections for both conventional and non-conventional compute.

Essentially, models built for both classification and super-resolution have adopted similar architectures such as residual blocks He et al. (2015), attention mechanisms Zhang et al. (2018), and many more. Additionally, models constructed for both problems have been getting deeper as increasing layers has been shown to increase performance. One main difference between the two is that models built for classification problems focus on building blocks for spatial downsampling while models built for super-resolution problems focus on spatial upsampling. Therefore, we aim to understand the methods and techniques, that are previously successful on downsampling models, on upsampling models.

According to the recently proposed super-resolution architectures (See Table 3), recent models differ in a small change of two common metrics: PSNR (dB) and SSIM. Although these changes seem negligible, they make a big difference at the higher scaling factors. Metrics that measure the memory and efficiency of the models include the number of parameters and the number of multiply-adds (Mult-Adds) and these dictate on which platform these models can run on. However, these metrics alone do not reflect the trade-off between performance and efficiency and between performance and memory. Therefore, we introduce two new metric that measures the number of Giga Mult-Adds saved and the number of parameters saved for every 0.01dB PSNR loss in the test sets: Set5 Bevilacqua et al. (2012), Set14 Yang et al. (2010), B100 Martin et al. (2001), and Urban100 Huang et al. (2015), of the compressed model against the uncompressed model. All Mult-Adds are calculated by upscaling to a 720p image.

We decide to use RCAN Zhang et al. (2018) as our baseline model as it proves to be the state-of-the-art and has the best evaluation metric at this time of writing. We take its simplest building block and build a shallower network and use that as a basis for exploring the use of a variety of techniques (see Table 1).

Implementation Details: We train our models in section 3 and section 4.1 in the same manner as that of EDSR Lim et al. (2017), we train our models using 48×48 RGB patches of LR images from the DIV2K dataset Timofte et al. (2017). We augment the training data with random horizontal flips and 90 degree rotations and pre-process them by subtracting the mean RGB value of the DIV2K dataset. Our model is trained using the ADAM Kingma & Ba (2014) optimizer with hyper-parameters $\beta_1 = 0.9, \beta_2 = 0.999$, and $\epsilon = 10^{-8}$. The mini-batch size is 16, learning rate begins with $1e-4$ and is halved at 200 epochs, and the model is trained for 300 epochs using L1 loss. We train x2 models from scratch and use them to train x3 and x4 models.



(a) *148089* from B100 (b) *img034* from Urban100

Figure 1: Visual comparison on x4 scale datasets

3 Efficient Network Architectures for Super Resolution

We begin our evaluation by conducting a series of experiments while focussing on three aspects of SR efficiency.

3.1 Effects of various resource-efficient techniques

Objective: The main purpose of this paper is to improve the execution efficiency of SR models, while keeping the quality of the reconstructed image as high as possible. Towards this goal, we first develop an initial understanding of the trade-off solution by replacing and modifying the 3x3 convolution layer blocks in the baseline model (see below). Especially we adopt various optimizing techniques from previous attempts on building resource-efficient neural network architectures.

Model Architecture Optimization: In the following we summarize the optimization techniques we explore in the paper to gain overall runtime efficiency of SR models. Especially, we explore the use of techniques such as bottleneck design, group convolutions, and channel shuffling. In our experiments, we replace residual block (RB) in each RCAB layer with an optimization unit as indicated above. Interestingly, we found that replacing each 3x3 convolution layer with the optimization unit leads to a higher evaluation metric in generated SR images while still improve efficiency of the modified model. In general, we take the non-downsampling unit from resource-efficient architectures and remove all batch normalisation layers as they were previously shown to reduce performance and increase GPU memory usage Lim et al. (2017). We replace each 3x3 convolution layer in all RGs in our baseline model. We calculate our trade-off metrics based on the baseline model.

bl: Our baseline model from RCAN Zhang et al. (2018). We reduce the number of residual groups (RG) from 10 to 2 and the number of residual channel attention block (RCAB) in each RG from 20 to 5. Making the network shallower and small in parameters allow us to clearly understand each architectural changes as opposed to having a deep network which may cause other effects and interplay.

blrn(r): We adopt the residual bottleneck design from ResNet He et al. (2015) with a reduction factor of r. Specifically, a 1x1 convolution is used to compress information among channels by a reduction factor, resulting in a cheaper 3x3 convolution. Another 1x1 convolution is then used to recover the dimension of the output channel and a skip connection is used to pass on information that may have been lost.

blrxn(r,g): We replace the 3x3 convolution in *blrn* to a 3x3 grouped convolution, forming a block that is similar to that of ResNeXt Xie et al. (2016) with an additional group size of g. Computation cost is further reduced by the use of grouped convolutions, first introduced in Krizhevsky et al. (2012). Although grouping convolutions increases efficiency, output channels are derived from a subset of input channels, causing a degradation in performance. Therefore, 1x1 convolution layers are used to mitigate the drawback.

blm1: In order to further improve efficiency of the 3x3 grouped convolution, we can maximise the group size , forming a convolution that is known as depthwise convolution. Following this idea, we adopt the MobileNet v1 Howard et al. (2017) unit which uses a 3x3 depthwise convolution, followed by a 1x1 convolution, also known as a pointwise convolution.

bleff(r): We can further approximate the 3x3 depthwise convolution by using a 1x3 and a 3x1 depthwise convolution, a technique that is used in EffNet Freeman et al. (2018). We adopt the unit from EffNet by removing the pooling layers.

bls1(r, g): Besides grouping 3x3 convolutions, 1x1 convolution can also be grouped. In order to improve the information flow among channels in 3x3 and 1x1 grouped convolutions, channel shuffling can be used. In order to test the effects of channel shuffling, we adopt the ShuffleNet v1 Zhang et al. (2017) unit.

blclc(g1, g2): Channel shuffle is also used in Clcnet Zhang (2017) to further improve efficiency of *blm1*. In order to maximise efficiency from our adoption of ClcNet units, we follow the group size guidelines recommended by the authors for both the group sizes of the 3x3 (*g1*) and 1x1 (*g2*) grouped convolution.

bls2: Apart from using grouped convolutions, the authors of ShuffleNet v2 Ma et al. (2018) proposed splitting the flow into two, which is termed as channel splitting, and performing convolution on only half of the input channels in each unit at each pass. Channel shuffle is then used to enable information flow between both branches.

blm2(e): Inverted residuals can be used to enable skip connections directly on the bottleneck layers. Therefore, we adopt the MobileNet v2 Sandler et al. (2018) unit in our experiments

Table 1: Quantitative results of applying resource-efficient techniques. **bold**/*italics* indicates best/second-best trade-off.

Scale	Model	Params (K)	Mult-Adds (G)	Set5 PSNR/SSIM	Set14 PSNR/SSIM	B100 PSNR/SSIM	Urban100 PSNR/SSIM	Mult-Add Saved (G)	Params Saved (K)
2×	bl	1006	231.2	37.86/0.9600	33.39/0.9159	32.06/0.8982	31.74/0.9248	-	-
	blrn(r=2)	464	106.5	37.61/0.9591	33.18/0.9137	31.90/0.8961	31.09/0.9173	**0.9819**	**4.27**
	blm1	265	60.7	37.45/0.9586	33.00/0.9122	31.79/0.8948	30.65/0.9128	*0.7894*	*3.43*
	blrxn(r=2)	305	69.9	37.45/0.9585	33.01/0.9123	31.80/0.8948	30.71/0.9131	0.7755	3.37
	blrn(r=4)	258	59	37.42/0.9583	32.96/0.9120	31.76/0.8943	30.66/0.9123	0.7653	3.32
	blclc	232	52.9	37.40/0.9581	32.94/0.9118	31.73/0.8939	30.53/0.9108	0.7278	3.16
	bls2	211	48.4	37.37/0.9580	32.96/0.9117	31.71/0.8936	30.49/0.9102	0.7254	3.15
	bleff(r=2)	257	58.7	37.33/0.9580	32.94/0.9114	31.71/0.8936	30.41/0.9098	0.6485	2.82
	blm2(e=2)	561	128.9	37.43/0.9586	33.08/0.9130	31.81/0.8949	30.77/0.9138	0.5219	2.27
	bls1(r=2)	188	42.9	37.06/0.9568	32.74/0.9096	31.52/0.8912	29.94/0.9031	0.4968	2.16
	blm2(e=3)	763	175.4	37.56/0.9589	33.11/0.9133	31.86/0.8954	30.93/0.9155	0.3509	1.53

Results. Our results in Table 1 show that techniques that have a better trade-off between memory and performance will have a better trade-off between efficiency and performance as well [1]. The use of bottlenecks alone (*blrn*) result in the best trade-offs.

The use of 3x3 grouped convolutions results in the next best trade-offs. For groups, we use a group size of 4 unless the authors in the respective papers recommended otherwise. The use of grouped convolutions requires additional tuning of group size in order to give the best trade-off as shown in Ahn et al. (2018). In general, a smaller group size leads to a higher evaluation metric. Given that we use small group sizes among the units that include group convolution, *blm1* performs well without the need to tune the group size.

Reducing the number of features to accommodate the adoption of MobileNet v2 severely impact the performance which we omit from the table. We speculate that doing so would result in insufficient features at the upsampling layer to fully capture the SR image representation. Thus, we use the same number of feature maps as our bottleneck. Although the use of inverted residuals in our experiments seem worse off, it may perform better on models that use a larger feature size like EDSR Lim et al. (2017) or models that use multiple smaller upsampling layers like LapSRN Lai et al. (2017).

Lastly, the use of 1x1 grouped convolution or channel splitting with channel shuffling further reduces the evaluation metric. Although doing so can drastically reduce size, the trade-off does not seem to justify its advantages. Therefore, we recommend using bottlenecks for building resource-efficient SR architectures. If the budget for memory and efficiency is tight, we recommend the use of 3x3 grouped convolutions together with pointwise convolutions instead.

[1] Results for 3x and 4x upscaling show similar evaluation, memory, and efficiency trade-offs

Table 2: Quantitative results of applying techniques on different parts of the model. **bold**/*italics* indicates best/second-best trade-off.

Scale	Model [Changes]	Params (K)	Mult-Adds (G)	Set5 PSNR/SSIM	Set14 PSNR/SSIM	B100 PSNR/SSIM	Urban100 PSNR/SSIM	Mult-Add Saved (G)	Params Saved (K)
2×	bl-e	1006	265.3	37.92/0.9602	33.43/0.9162	32.09/0.8988	31.83/0.9256	-	-
	blrn-e(r=2)[rb]	535	156.8	37.75/0.9596	33.30/0.9153	32.00/0.8973	31.48/0.9218	**1.4662**	**6.3649**
	blm1-e[rb]	363	117	37.65/0.9592	33.19/0.9143	31.92/0.8964	31.13/0.9181	*1.0746*	*4.6594*
	blm1-e[rg]	265	94.7	37.59/0.9590	33.12/0.9132	31.87/0.8959	30.91/0.9158	0.9584	4.1629
	blrn-e(r=2)[rg]	464	140.5	37.64/0.9592	33.20/0.9142	31.93/0.8967	31.18/0.9185	0.9455	4.1061
	blrn-e(r=2)[rg+u]	370	97.1	37.56/0.9589	33.10/0.9131	31.86/0.8954	30.99/0.9164	0.9557	3.6136
	blm1-e[rg+u]	137	35.4	37.35/0.9580	32.94/0.9117	31.72/0.8939	30.45/0.9103	0.8181	3.0925
3×	bl-e	1080	156.3	34.35/0.9269	30.29/0.8415	29.06/0.8046	28.13/0.8521	-	-
	blrn-e(r=2)[rb]	609	108.1	34.19/0.9257	30.18/0.8394	29.00/0.8026	27.89/0.8469	**0.8456**	**8.2632**
	blm1-e[rb]	437	90.5	34.09/0.9249	30.13/0.8379	28.95/0.8015	27.69/0.8419	0.6784	*6.6289*
	blrn-e(r=2)[rg+u]	397	57.6	33.97/0.9236	30.04/0.8362	28.89/0.7997	27.50/0.8376	*0.6902*	4.7762
	blrn-e(r=2)[rg]	538	100.9	34.13/0.9251	30.14/0.8380	28.97/0.8016	27.72/0.8421	0.6368	6.2299
	blm1-e[rg]	339	80.6	34.08/0.9244	30.03/0.8356	28.92/0.8005	27.55/0.8386	0.6056	5.928
	blm1-e[rg+u]	146	21.4	33.73/0.9221	29.83/0.8320	28.78/0.7970	27.10/0.8283	0.5644	3.9079
4×	bl-e	1154	135.5	32.08/0.8942	28.58/0.7815	27.56/0.7360	26.16/0.7872	-	-
	blrn-e(r=2)[rb]	683	108.3	32.10/0.8938	28.51/0.7795	27.51/0.7340	25.95/0.7808	**0.8774**	**15.1935**
	blm1-e[rb]	511	98.4	31.98/0.8921	28.45/0.7778	27.47/0.7328	25.80/0.7754	0.5456	*9.4559*
	blrn-e(r=2)[rg+u]	424	50	31.84/0.8898	28.34/0.7750	27.38/0.7295	25.52/0.7673	*0.6577*	5.6154
	blm1-e[rg]	413	92.8	32.02/0.8921	28.44/0.7765	27.44/0.7310	25.67/0.7715	0.5272	9.1481
	blrn-e(r=2)[rg]	612	104.3	32.02/0.8925	28.47/0.7779	27.48/0.7323	25.79/0.7755	0.5032	8.7419
	blm1-e[rg+u]	156	18.7	31.47/0.8847	28.12/0.7697	27.26/0.7252	25.16/0.7538	0.4928	4.211

3.2 Effects of Architectural Layers between the Input and Output Layer

Experimental Aim. Given our results shown in section 3.1, we apply the best techniques, which are *blrn* and *blm1*, on different parts of the model to further understand the extent of their applicability.

Model Architecture. Our preliminary experiments with applying some of these techniques on the first and last convolution layer led to worse trade-offs. Therefore, we apply our techniques on other parts of the model. Additionally, since some techniques use skip connections and require the number of input and output channels to be the same, we replace the sub-pixel convolution upsampling layer to the enhanced upscaling module (EUM) as proposed in EUSR Kim & Lee (2018). Including EUMs essentially adds more residual modules, leading to an increase in evaluation but increasing both the memory and operation cost. In order to maintain the same memory cost as that of the sub-pixel convolution layer or even decrease the memory cost for x3 scaling, we use recursion, forming the enhanced recursive upscaling module (ERUM). The number of ERUMs is the same as the upsampling factor and each ERUM recurses twice or thrice for x2, x4 or x3 upscaling respectively before the outputs are concatenated and shuffled as shown in figure 2. We refer to experiments that use ERUMs for upsampling with a postfix *-e*. We calculate our trade-off metrics based on *bl-e*. We modify all 3x3 convolution layers as such:

rb: Changes are made in the RB in each RCAB.

rg: Changes are made in each RG, therefore including those in *rb*. Experiments in section 3.1 are done in this setting.

rg+u: Changes are made in both *rg* as well as the ERUMs in the upsampling portion.

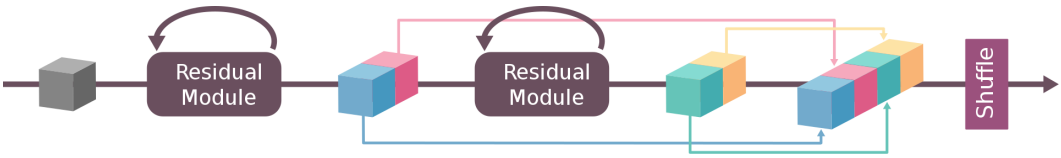


Figure 2: Our proposed ERUM for image upscaling.

Results. Our results in Table 2 validate our findings in section 3.1 that the adoption of bottleneck alone leads to the best trade-offs. Additionally, compressing the model further leads to worse trade-offs for both efficiency and size for the use of grouped convolutions while doing so for the use of bottlenecks result in a worse trade-off between performance and size. Therefore, we recommend

taking gradual steps to compress the model. For instance, we suggest changing small parts of the model apart from the upsampling, first, and last convolutions till a budget is reached. If further compression is needed on the upsampling layer, we suggest to use the bottleneck design instead of grouped convolutions.

3.3 Comparisons with previous SR models

In terms of size and evaluation metric, our best model outperforms all models that have a count of 1,500K parameters and below. By comparing efficiency and evaluation, our best model performs better, slightly smaller, and has roughly x6 less operations than MemNet but has roughly 6x less operations. It is also comparable with the CARN model in the number of operations, trading a slightly worse evaluation metric with 2.5x size reduction. Overall, our best model is better than earlier models such as VDSR and later models such as SRMDNF for 3x and 4x upscaling. Our second and third best models also outperform earlier models in evaluation metric with huge savings in the number of operations for 3x and 4x upscaling.

4 Quantization and low-precision under super resolution

In our next set of experiments, we examine the viability of quantization and the use of extreme low-precision (ternary/binary) as mechanisms to reduce system resource for SR.

4.1 Integer quantization

Objective. With the success of low precision on neural networks on classification problems, we aim to show initial understanding of applying 8-bits integer quantization on our baseline model as described in section 3.1. Moving from 32-bits to 8-bits will result in a 4x reduction in memory and allows support for low-power embedded devices such as IoT devices. Furthermore, arithmetic operations is much faster on specialized and other custom hardware.

Quantization Method. We train the model in full precision and apply the quantization scheme in Tensorflow-Lite for integer-only arithmetic Jacob et al. (2017) and retrain for an additional 5 epochs with the same learning rate.

Results. Our results show that applying quantization lead to a slight evaluation loss in 2x scaling and a slight improvement in 4x scaling. Our results are similar to that of classification Jacob et al. (2017). Furthermore the results show that deep neural networks are robust to noise and perturbations caused by quantization. Therefore, we strongly recommend quantization especially on hardware that can further utilise its benefits.

4.2 Ternary precision

Objective. Quantization has become a standard practice in many respects, we contrast this with the use of a collection of more extreme forms of techniques grounded in reduced parameter precision. In performing such experiments, we present two end points in what is possible to achieve in terms of super resolution efficiency with quantization/low precision methods.

Ternary Architecture. As discussed, to present a counter extreme view to the pedestrian quantization of section 4.1 – we adapt the baseline super resolution architecture used in prior experiments in section 3.1 but modify it structurally by replacing every convolution layer with sum-product convolution layers proposed in StrassenNets Tschannen et al. (2017). These sum-product convolution layers represent a sum-product network (SPN) that is used to approximate a matrix multiplication. Specifically, each convolution layer is replaced with a convolution layer that outputs r feature maps, followed by a element-wise multiplication and a transpose convolution layer. As both the convolution layers hold ternary weights, the number of multiply operations required is determined by the number of element-wise multiplication which is controlled by r. Besides outlining the trade-off of tuning r, we aggressively use group convolutions as have been seen in architectural efficiency works described earlier in section 3.1.

Table 3: We extend the table from Ahn et al. (2018) and add recent models to compare with our best three models (in order) from previous sections.

Scale	Model	Params (K)	Mult-Adds (G)	Set5 PSNR/SSIM	Set14 PSNR/SSIM	B100 PSNR/SSIM	Urban100 PSNR/SSIM
2×	SRCNN	57	52.7	36.66/0.9542	32.42/0.9063	31.36/0.8879	29.50/0.8946
	FSRCNN	12	6.0	37.00/0.9558	32.63/0.9088	31.53/0.8920	29.88/0.9020
	VDSR	665	612.6	37.53/0.9587	33.03/0.9124	31.90/0.8960	30.76/0.9140
	DRCN	1,774	9,788.7	37.63/0.9588	33.04/0.9118	31.85/0.8942	30.75/0.9133
	CNF	337	311.0	37.66/0.9590	33.38/0.9136	31.91/0.8962	-
	LapSRN	813	29.9	37.52/0.9590	33.08/0.9130	31.80/0.8950	30.41/0.9100
	DRRN	297	6,796.9	37.74/0.9591	33.23/0.9136	32.05/0.8973	31.23/0.9188
	BTSRN	410	207.7	37.75/-	33.20/-	32.05/-	31.63/-
	MemNet	677	623.9	37.78/0.9597	33.28/0.9142	32.08/0.8978	31.31/0.9195
	SelNet	974	225.7	37.89/0.9598	33.61/0.9160	32.08/0.8984	-
	EDSR	40,711	9384.7	38.11/0.9602	33.92/0.9195	32.92/0.9351	32.93/0.9351
	SRMDNF	2218	513.6	37.79/0.9601	33.32/0.9159	32.05/0.8985	31.33/0.9204
	D-DBPN	1,261	158.9	38.09/0.9600	33.85/0.9190	32.27/0.9000	32.55/0.9324
	RDN	22,123	5,096.2	38.24/0.9614	34.01/0.9212	32.34/0.9017	32.89/0.9353
	CARN	1,592	222.8	37.76/0.9590	33.52/0.9166	32.09/0.8978	31.92/0.9256
	CARN-M	412	91.2	37.53/0.9583	33.26/0.9141	31.92/0.8960	31.23/0.9193
	RCAN	15,444	3526.8	38.27/0.9614	34.12/0.9216	32.41/0.9027	33.34/0.9384
	blrn-e(r=2)[rb](ours)	**535**	**156.8**	**37.75/0.9596**	**33.30/0.9153**	**32.00/0.8973**	**31.48/0.9218**
	blm1-e[rb](ours)	**363**	**117**	**37.65/0.9592**	**33.19/0.9143**	**31.92/0.8964**	**31.13/0.9181**
	blm1-e[rg](ours)	**265**	**94.7**	**37.59/0.9590**	**33.12/0.9132**	**31.87/0.8959**	**30.91/0.9158**
3×	SRCNN	57	52.7	32.75/0.9090	29.28/0.8209	28.41/0.7863	26.24/0.7989
	FSRCNN	12	5.0	33.16/0.9140	29.43/0.8242	28.53/0.7910	26.43/0.8080
	VDSR	665	612.6	33.66/0.9213	29.77/0.8314	28.82/0.7976	27.14/0.8279
	DRCN	1,774	9,788.7	33.82/0.9226	29.76/0.8311	28.80/0.7963	27.15/0.8276
	CNF	337	311.0	33.74/0.9226	29.90/0.8322	28.82/0.7980	-
	DRRN	297	6,796.9	34.03/0.9244	29.96/0.8349	28.95/0.8004	27.53/0.8378
	BTSRN	410	176.2	34.03/-	29.90/-	28.97/-	27.75/-
	MemNet	677	623.9	34.09/0.9248	30.00/0.8350	28.96/0.8001	27.56/0.8376
	SelNet	1,159	120.0	34.27/0.9257	30.30/0.8399	28.97/0.8025	-
	EDSR	43,660	4470.8	34.65/0.9289	30.52/0.8462	29.25/0.8093	28.80/0.8653
	SRMDNF	2,956	305.5	34.12/0.9254	30.04/0.8382	28.97/0.8225	27.570.8398
	RDN	22,308	2,281.2	34.71/0.9296	30.57/0.8468	29.26/0.8093	28.80/0.8653
	CARN	1,592	118.8	34.29/0.9255	30.29/0.8407	29.06/0.8034	28.06/0.8493
	CARN-M	412	46.1	33.99/0.9236	30.08/0.8367	28.91/0.8000	27.55/0.8385
	RCAN	15,629	1584.7	34.74/0.9299	30.65/0.8482	29.32/0.8111	29.09/0.8702
	blrn-e(r=2)[rb](ours)	**609**	**108.1**	**34.19/0.9257**	**30.18/0.8394**	**29.00/0.8026**	**27.89/0.8469**
	blm1-e[rb](ours)	**437**	**90.5**	**34.09/0.9249**	**30.13/0.8379**	**28.95/0.8015**	**27.69/0.8419**
	blrn-e(r=2)[rg+u](ours)	**397**	**57.6**	**33.97/0.9236**	**30.04/0.8362**	**28.89/0.7997**	**27.50/0.8376**
4×	SRCNN	57	52.7	30.48/0.8628	27.49/0.7503	26.90/0.7101	24.52/0.7221
	FSRCNN	12	4.6	30.71/0.8657	27.59/0.7535	26.98/0.7150	24.62/0.7280
	VDSR	665	612.6	31.35/0.8838	28.01/0.7674	27.29/0.7251	25.18/0.7524
	DRCN	1,774	9,788.7	31.53/0.8854	28.02/0.7670	27.23/0.7233	25.14/0.7510
	CNF	337	311.0	31.55/0.8856	28.15/0.7680	27.32/0.7253	-
	LapSRN	813	149.4	31.54/0.8850	28.19/0.7720	27.32/0.7280	25.21/0.7560
	DRRN	297	6,796.9	31.68/0.8888	28.21/0.7720	27.38/0.7284	25.44/0.7638
	BTSRN	410	165.2	31.85/-	28.20/-	27.47/-	25.74/-
	MemNet	677	623.9	31.74/0.8893	28.26/0.7723	27.40/0.7281	25.50/0.7630
	SelNet	1,417	83.1	32.00/0.8931	28.49/0.7783	27.44/0.7325	-
	SRDenseNet	2,015	389.9	32.02/0.8934	28.50/0.7782	27.53/0.7337	26.05/0.7819
	EDSR	43,070	2894.5	32.46/0.8968	28.8/0.7876	27.71/0.7420	26.64/0.8033
	SRMDNF	3,988	232.7	31.96/0.8925	28.35/0.7787	27.49/0.7337	25.68/0.7731
	D-DBPN	2,207	79.7	32.47 0.8980	28.82/0.7860	27.72/0.7400	26.38/0.7946
	RDN	22,271	1,309.2	32.47/0.8990	28.81/0.7871	27.72/0.7419	26.61/0.8028
	CARN	1,592	90.9	32.13/0.8937	28.60/0.7806	27.58/0.7349	26.07/0.7837
	CARN-M	412	32.5	31.92/0.8903	28.42/0.7762	27.44/0.7304	25.62/0.7694
	RCAN	15,592	916.9	32.63/0.9002	28.87/0.7889	27.77/0.7436	26.82/0.8087
	blrn-e(r=2)[rb](ours)	**683**	**108.3**	**32.10/0.8938**	**28.51/0.7795**	**27.51/0.7340**	**25.95/0.7808**
	blm1-e[rb](ours)	**511**	**98.4**	**31.98/0.8921**	**28.45/0.7778**	**27.47/0.7328**	**25.80/0.7754**
	blrn-e(r=2)[rg+u](ours)	**424**	**50**	**31.84/0.8898**	**28.34/0.7750**	**27.38/0.7295**	**25.52/0.7673**

Results. Similar to section 4.1, the results in table 5 are similar to that of applying it to classification problems Tschannen et al. (2017). Specifically, the higher the width of the hidden layer of the SPN, r, the better the evaluation metric at a cost of additional multiplications and additions. When $r = 6c_out$, we achieve an evaluation score that is close to the uncompressed model for 2x upscaling

Table 4: Quantitative results of applying 8-bit integer quantization in TF-Lite.

Scale	Model	Params (K)	Mult-Adds (G)	Set5 PSNR/SSIM	Set14 PSNR/SSIM	B100 PSNR/SSIM	Urban100 PSNR/SSIM
2×	bl	961	231.2	37.82/0.9599	33.36/0.9160	32.05/0.8981	31.67/0.9237
	bl_q			37.68/0.9582	33.34/0.9146	32.01/0.8966	31.64/0.9226
3×	bl	1191	122.4	34.20/0.9257	30.15/0.8392	29.01/0.8028	27.91/0.8467
	bl_q			34.17/0.9259	30.13/0.8393	29.00/0.8030	27.92/0.8474
4×	bl	1154	93	32.01/0.8927	28.41/0.7793	27.49/0.7337	25.87/0.7792
	bl_q			31.99/0.8922	28.43/0.7794	27.51/0.7337	25.88/0.7790

and suffer a slight drop for 3x and 4x upscaling. Any further attempts to increase r do not improve evaluation metric.

As proposed by the authors Tschannen et al. (2017), we use group convolutions to reduce the number of additions. We take a step further and experiment with a wide range of groups as well. We found that the reduced number of additions do not justify the evaluation drop and that the use of a lower r is much better in terms of evaluation metric and the number of multiplication reduced than the use of groups. Additionally, since multipliers are more costly and take up more area on chip than adders, we suggest lowering r instead of using grouped convolutions.

Table 5: Quantitative results of applying ternary-weighted SP convolutions. c_out refers to the number of output channels in the SP convolution layer.

Scale	Model	r	Reduction in Mult (%)	Reduction in Add (%)	Set5 PSNR/SSIM	Set14 PSNR/SSIM	B100 PSNR/SSIM	Urban100 PSNR/SSIM
2×	bl	-	-	-	37.86/0.9600	33.39/0.9159	32.06/0.8982	31.74/0.9248
	ST-bl	c_out	99.82	-15.96	37.59/0.9588	33.14/0.9138	31.88/0.8959	31.02/0.9171
		2c_out	99.64	-132.1	37.73/0.9595	33.30/0.9152	31.98/0.8973	31.37/0.9210
		4c_out	99.28	-364.38	37.81/0.9598	33.31/0.9151	32.03/0.8978	31.59/0.9229
		6c_out	98.92	-596.65	37.85/0.9600	33.41/0.9162	32.06/0.8981	31.68/0.9240
3×	bl	-	-	-	34.24/0.9260	30.26/0.8405	29.03/0.8033	27.96/0.8479
	ST-bl	c_out	99.81	-35.58	33.79/0.9215	29.92/0.8340	28.82/0.7983	27.25/0.8318
		2c_out	99.64	-171.33	33.99/0.9236	30.07/0.8365	28.91/0.8005	27.54/0.8389
		4c_out	99.28	-442.83	34.16/0.9250	30.11/0.8380	28.97/0.8021	27.73/0.8435
		6c_out	98.92	-714.33	34.17/0.9253	30.15/0.8383	28.99/0.8023	27.83/0.8454
4×	bl	-	-	-	32.06/0.8930	28.49/0.7787	27.50/0.7337	25.87/0.7788
	ST-bl	c_out	99.81	-26.04	31.46/0.8829	28.15/0.7709	27.27/0.7265	25.24/0.7566
		2c_out	99.64	-152.24	31.72/0.8871	28.31/0.7743	27.37/0.7296	25.49/0.7662
		4c_out	99.28	-404.65	31.90/0.8901	28.40/0.7770	27.45/0.7319	25.67/0.7723
		6c_out	98.93	-657.06	31.95/0.8907	28.43/0.7777	27.46/0.7324	25.77/0.7752

5 BEST PRACTICES FOR EFFICIENT SUPER-RESOLUTION

Through our experiments, we come up with a list of recommendations to build resource-efficient super-resolution:

- The bottleneck design in ResNet He et al. (2015) is the best technique for the best performance, memory, efficiency trade-off. If further compression is needed, tuning group sizes in 3x3 grouped convolutions or simply using depthwise convolutions followed by pointwise convolutions is ideal. If efficiency on conventional hardware is the topmost priority, we recommend the use of channel splitting and shuffling like units in ShuffleNet v2 Ma et al. (2018).
- The fewer resource-efficient architecture changes applied, the better the trade-off. Therefore, we recommend a mixture of convolution and resource-efficient units unless further compression is needed. Avoid architecture changes on the first and the last convolution layers.
- We strongly recommend using binary/ternary quantization if the hardware supports it. However, if evaluation metric is critical, we suggest lowering the precision instead.

REFERENCES

Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. Fast, accurate, and, lightweight super-resolution with cascading residual network. *CoRR*, abs/1803.08664, 2018. URL http://arxiv.org/abs/1803.08664.

Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie line Alberi Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In *Proceedings of the British Machine Vision Conference*, pp. 135.1–135.10. BMVA Press, 2012. ISBN 1-901725-46-4. doi: http://dx.doi.org/10.5244/C.26.135.

Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *CoRR*, abs/1501.00092, 2015. URL http://arxiv.org/abs/1501.00092.

Ido Freeman, Lutz Roese-Koerner, and Anton Kummert. Effnet: An efficient structure for convolutional neural networks. *CoRR*, abs/1801.06434, 2018. URL http://arxiv.org/abs/1801.06434.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861.

J. Huang, A. Singh, and N. Ahuja. Single image super-resolution from transformed self-exemplars. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5197–5206, June 2015. doi: 10.1109/CVPR.2015.7299156.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *CoRR*, abs/1712.05877, 2017. URL http://arxiv.org/abs/1712.05877.

Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. *CoRR*, abs/1511.04587, 2015. URL http://arxiv.org/abs/1511.04587.

Jun-Hyuk Kim and Jong-Seok Lee. Deep residual network with enhanced upscaling module for super-resolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Deep laplacian pyramid networks for fast and accurate super-resolution. *CoRR*, abs/1704.03915, 2017. URL http://arxiv.org/abs/1704.03915.

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.

N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. *ArXiv e-prints*, July 2018.

D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pp. 416–423 vol.2, July 2001. doi: 10.1109/ICCV.2001.937655.

Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. URL http://arxiv.org/abs/1801.04381.

Ying Tai, Jian Yang, Xiaoming Liu, and Chunyan Xu. Memnet: A persistent memory network for image restoration. *CoRR*, abs/1708.02209, 2017. URL http://arxiv.org/abs/1708.02209.

R. Timofte, E. Agustsson, L. V. Gool, M. Yang, L. Zhang, B. Lim, S. Son, H. Kim, S. Nah, K. M. Lee, X. Wang, Y. Tian, K. Yu, Y. Zhang, S. Wu, C. Dong, L. Lin, Y. Qiao, C. C. Loy, W. Bae, J. Yoo, Y. Han, J. C. Ye, J. Choi, M. Kim, Y. Fan, J. Yu, W. Han, D. Liu, H. Yu, Z. Wang, H. Shi, X. Wang, T. S. Huang, Y. Chen, K. Zhang, W. Zuo, Z. Tang, L. Luo, S. Li, M. Fu, L. Cao, W. Heng, G. Bui, T. Le, Y. Duan, D. Tao, R. Wang, X. Lin, J. Pang, J. Xu, Y. Zhao, X. Xu, J. Pan, D. Sun, Y. Zhang, X. Song, Y. Dai, X. Qin, X. Huynh, T. Guo, H. S. Mousavi, T. H. Vu, V. Monga, C. Cruz, K. Egiazarian, V. Katkovnik, R. Mehta, A. K. Jain, A. Agarwalla, C. V. S. Praveen, R. Zhou, H. Wen, C. Zhu, Z. Xia, Z. Wang, and Q. Guo. Ntire 2017 challenge on single image super-resolution: Methods and results. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1110–1121, July 2017. doi: 10.1109/CVPRW.2017.149.

Michael Tschannen, Aran Khanna, and Anima Anandkumar. Strassennets: Deep learning with a multiplication budget. *CoRR*, abs/1712.03942, 2017. URL http://arxiv.org/abs/1712.03942.

Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016. URL http://arxiv.org/abs/1611.05431.

Jianchao Yang, John Wright, Thomas S. Huang, and Yi Ma. Image super-resolution via sparse representation. *Trans. Img. Proc.*, 19(11):2861–2873, November 2010. ISSN 1057-7149. doi: 10.1109/TIP.2010.2050625. URL http://dx.doi.org/10.1109/TIP.2010.2050625.

Dong-Qing Zhang. clcnet: Improving the efficiency of convolutional neural network using channel local convolutions. *CoRR*, abs/1712.06145, 2017. URL http://arxiv.org/abs/1712.06145.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017. URL http://arxiv.org/abs/1707.01083.

Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu. Image Super-Resolution Using Very Deep Residual Channel Attention Networks. *ArXiv e-prints*, July 2018.