Single Image Super Resolution on iPhone

Myles Robinson mylesr
Nick Chermak nchermak
Vedant Bhasin vedantb
Bharathi Sridhar bsridha2

Abstract

Efficient inference of machine learning models in mobile environments is critical to balance performance, accuracy and resource consumption. This project explores various model compression techniques, including quantization and pruning, to optimize the SwinIR image enhancement architecture for deployment on an iPhone. The SwinIR model, a state-of-the-art transformer-based architecture, is adapted to run on-device with a focus on achieving efficient inference while maintaining high-quality outputs. Strategies such as post-training quantization, mixed precision, and structured pruning are implemented to reduce the model's inference latency and memory footprint. Experimental results show that these techniques enable significant reductions in model size and latency with varying degrees of performance degradation, demonstrating the viability of deploying advanced image enhancement solutions on resource-constrained mobile devices. This report provides some insight into the trade-offs between model compression and accuracy, offering some advances in on-device image-based models.

1 Introduction

1.1 Motivation and Significance

In today's world, mobile device cameras have made remarkable strides in image quality and detail. As a result, bulky cameras are becoming a thing of the past, with more people choosing their phones to capture everyday moments. However, despite the advancements in mobile photography, users still encounter limitations that lead to less-than-ideal photos.

Imagine a scenario where a photo is taken, but an unwanted object, like a car in the background, detracts from the intended subject. Or perhaps, someone wishes the city skyline they captured during the day appeared as it would at night, with all the city lights and reflections. Maybe a person wants a picture of a lake at sunset, but only managed to take one midday. Similarly, many users may struggle with their camera's complex settings, resulting in overexposed images or poorly focused subjects.

Current capabilities and solutions exist but not without their own limitations. Modern mobile devices offer impressive tools for image enhancement, such as Google Pixel's Magic Editor or Apple iPhone's ProRAW; these features often rely on cloud processing. This raises concerns about privacy, as personal photos are transmitted over the internet which can also lead to delays in processing time, especially in places where connectivity is less than ideal. Additionally, companies' AI enhancement tools sometimes result in over processed and unnatural images while also reducing manual control and limiting users ability to tailor their photos.

Our solution seeks to mitigate these issues by developing an app that allows users to conduct single-image super-resolution (SISR) on their photos with professional level adjustments, while preserving user privacy and operating fully on-device using a machine learning model. By doing so, users could increase the resolution of poorly focus images or recover images they took on poor camera settings via mobile without relying on cloud.

1.2 Summary of Contributions

We used various compression approaches such as various quantization and pruning methods. Specifically, we experimented with static, dynamic, and mixed precision quantization, as well as structured attention head pruning and unstructured pruning. The key results were that static and dynamic quantization under perform, while mixed precision compute and FP16 storage work well with Apple's GPU and Neural Engine. Additionally, pruning heads results in better image quality than pruning

neurons per layer or unstructured pruning. Larger models seem more sensitive to quantization potentially because SISR and image reconstruction tasks rely on precise, high-fidelity computations and diverse weight distributions to accurately capture fine-grained details and textures.

2 Task Definition and Problem Setup

2.1 Task Definition

Our task is to conduct single-image super-resolution on images that a user wishes to change due low resolution issues. The image augmentation pipeline will take in images as user inputs and subsequently output super resolution enhanced images back to the user. The pipeline is dynamic with respect to the input image format whether they are in RAW or HEIF formats, both common image formats on Apple OS's. The returned image will be in HEIF or RAW, whichever corresponds to the format of the input.

The Pillow, pyheif, and rawpy python libraries have functions to retrieve the RGB values from HEIF or RAW image formats. Using the tools in these libraries, we can feed the RGB values into the model to augment the images.

Solving this on-device will allow resolution recovery of photos ensuring user privacy and creative control without relying on cloud processing.

2.2 Data Description and Metrics

For the purposes of evaluating our model with different compression techniques we will make use of a subset of DIV2K [1]. The dataset consists of 1000 2K resolution images divided into 800 images for training, 100 images for validation, 100 images for testing. Starting from 800 high definition high resolution images they obtain corresponding low resolution images and provide both high and low resolution images for 2, 3, and 4 downscaling factors.

To evaluate the performance, we chose a number of metrics to verify our results such as number of parameters, Memory, PSNR value, Latency, and percent sparsity. The number of parameters and memory size helped us compare various compressed models with our baseline. We could see how large our compressed model was and how much memory they used compared to our baseline to understand what factor we were able to shrink our model and maintain accuracy. PSNR value is a metric that measures the image quality of the restructured image. This helped us verify that the model was still outputting the same or similar enhanced images. Latency was used to capture the responsiveness and efficiency of the model. Finally, sparsity was used to compare different compression trials and see how much of the model could be removed while maintaining accuracy. Finally, the specific technique helped compare the previous described metrics across baseline, attention head pruning or unstructured pruning techniques.

2.3 Hardware Constraints

As the goal of this task focuses on on-device SISR, it is appropriate that we target a device commonly used to take photos. Today, most people take images with their phones, so this project targets the iPhone 16 Pro Max, Apple's newest and most advanced phone to-date. The hardware specifications for this device are shown in Table 1.

Table 1: iPhone 16 Pro Max Hardware Specifications

Specification	Description
Chip	A18 Pro
Processor	Hexa-core (2 perf. + 4 eff.)
GPU	6-core
RAM	8GB
Storage	256 GB

Our pipeline doesn't include any training, all compression techniques are carried out in PyTorch, the model is then exported to a `mlpackage` format to be deployed on iOS.

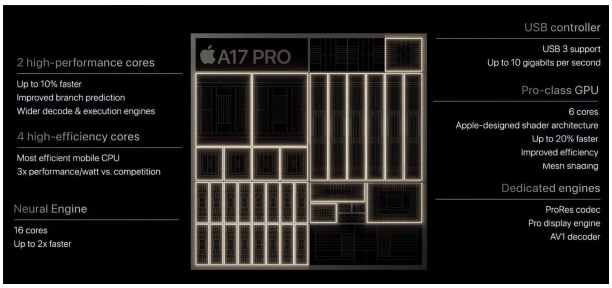


Figure 1: The A17 chip on our target hardware device with a 6 core Pro-class GPU, a maximum interconnect bandwidth of 10 Gbps and a total of 8GB RAM

3 Methods

3.1 Model Architecture and Design

Our baseline model is SwinIR, a transformer-based architecture designed for image restoration tasks. Unlike traditional convolutional neural networks (CNNs), SwinIR uses Swin Transformer Layers (STL), which process images hierarchically to capture both local and global features. These layers are grouped into Residual Swin Transformer Blocks (RSTB), enabling effective learning of fine details and complex patterns in high-resolution images.

We selected SwinIR because of its demonstrated success in tasks such as super-resolution, denoising, and artifact removal. It outperforms CNN-based models like EDSR and RCAN in balancing image quality and computational efficiency. The model's hierarchical structure and attention mechanisms make it well-suited for high-detail image enhancement, which is critical for mobile applications. For this project, we experiment and evaluate three different variants of the SwinIR model having different sizes, and different training sets. These variants are summarized in Table 2

Model	Training Data	Parameters
Classical	DIV2K	11.75 M
Lightweight	DIV2K+Flickr2K	0.91 M
Real	DIV2K+Flickr2K+OST	11.68 M

Table 2: Different Variants of the SwinIR model

3.2 Optimization Techniques

3.2.1 Quantization

Initially, we tried PyTorch's built-in static and dynamic quantization (using FP16) as well as mixed-precision training. However, these methods significantly degraded performance for our SISR task, which is sensitive to slight precision changes.

To address this, we switched to using CoreML tools, which are better aligned with Apple's hardware. We tested:

- **FP16 Compute with FP32 Storage:** Balanced precision and size, but still relatively large.
- **FP16 Compute with FP16 Storage:** Reduced size further with minimal quality loss.
- **Mixed-Precision (FP16/FP32):** Offered the best trade-off, delivering high quality at a smaller size.

We chose the methodology described above because SISR (and image reconstruction) tasks are inherently sensitive to minor distortions. Unlike classification tasks, where the model must only identify high-level features, SISR must reconstruct fine textures and details. Reducing precision too aggressively (e.g., int8) often leads to visually noticeable issues. Using CoreML's quantization tools, we customized the quantization process to hardware-supported formats, ensuring that the reduced precision still closely matches the iPhone's computation patterns. This approach is ideal for our problem setup—on-device SISR—because it lets us:

- Exploit iPhone hardware optimizations (native FP16 support).
- Achieve smaller model sizes and faster inference without severely compromising image quality.
- Ensure the quantized model is able to be integrated smoothly with the device's runtime, reducing the risk of datatype mismatch issues seen when directly converting PyTorch-quantized models.

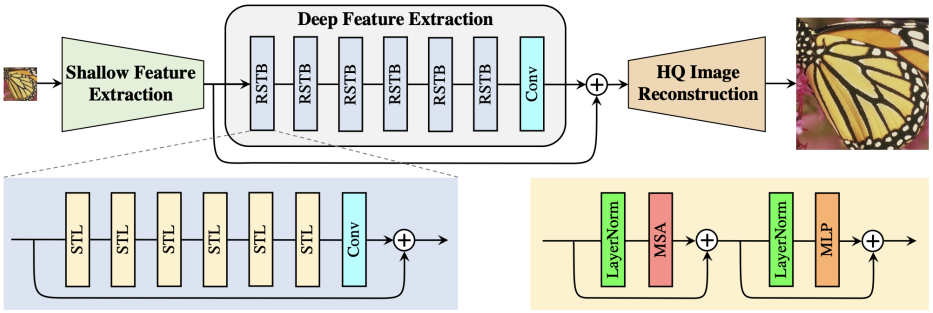
3.2.2 Pruning

Structured Pruning (Attention Heads) focuses on reducing the model complexity by selectively removing specific components, such as attention heads in transformer architectures. We implemented attention head pruning using a custom method based on the L1-norm of the weights. This approach identifies and removes less significant heads that contribute minimally to the model's overall performance.

Results:

- Minimal performance degradation was observed, with better preservation of image quality compared to unstructured pruning.
- Structured attention pruning maintained high-resolution outputs and was better suited for high-detail image tasks like enhancement and reconstruction.

Unstructured Pruning is similar to structured pruning in that we remove model weights based on



(a) Residual Swin Transformer Block (RSTB) (b) Swin Transformer Layer (STL)

Figure 2: Swin Transformer architecture which is composed of Residual Swin Transformer Blocks (RSTB which are in turn composed of Swin Transformer Layers (STL). The main components of each layer are a LayerNorm, Multihead Self Attention (MSA) and an MLP.

their L1-norms. However, in unstructured pruning, we prune across all linear layers of the model. We discovered that this caused a steep decline in model performance with respect to our PSNR accuracy metric. As a result, we determined structured pruning was a better choice for our application.

3.3 Implementation Details

3.3.1 Base Model

Our experiments were primarily conducted using PyTorch. We utilized pre-trained models corresponding to three distinct architectures tailored to Super-Resolution (SR) tasks: (1) Classical SR models, (2) Lightweight SR models, and (3) Real-world SR models. We started with available pretrained models[2] for these architectures. We leveraged PyTorch's built-in functionalities for pruning and quantization, and supplemented them with custom code for more quantization and pruning methods. For conversion and subsequent device-level optimizations, CoreML tools were used to target Apple's iOS hardware, ensuring more accurate modeling of the platform's specific requirements.

For the base SwinIR model applied to the SISR task, the following hyperparameters are used:

Hyperparameter	Value
Upscale factor	2
Image size	48
Patch size	1
Embed dimension	180
Depths	[6, 6, 6, 6, 6, 6]
Number of heads	[6, 6, 6, 6, 6, 6]
Window size	8
MLP ratio	2
Dropout rate	0.0
Attention dropout rate	0.0
Stochastic depth rate	0.1
Upsampler	pixelshuffle
Residual connection	1conv

Table 3: Hyperparameters for the base SwinIR model

3.3.2 Quantization

We initially used the built-in PyTorch quantization framework, exploring both static and dynamic quantization using FP16 as the target datatype. These attempts, as well as mixed-precision methods in PyTorch, significantly degraded model performance. When converting these quantized models to CoreML format, mismatches between the expected and actual datatypes further complicated deployment on the iOS simulator.

In remediate to these issues, we switched to using CoreML tools for quantization. We first experimented with two static quantization schemes:

- FP16 compute with FP32 storage
- FP16 compute with FP16 storage

Both approaches yielded similar accuracy, but the FP16 storage type model was notably smaller in size. Additionally, we tested a mixed-precision quantization strategy using FP16 storage alongside both FP16 and FP32 compute types. This configuration outperformed all other quantized versions, including the PyTorch-quantized models. This is likely because CoreML-based quantization techniques more precisely aligned with iPhone hardware constraints and performance capabilities.

3.3.3 Pruning

For unstructured pruning, we utilized PyTorch's built-in pruning library and its unstructured L1-norm-based pruning method. However, regarding structured pruning of the model's attention heads, we designed a custom function to iteratively prune transformer blocks by pruning their attention heads. In pruning the attention heads, we not only reduce the number of FLOPs for inference, but also eliminate the respective parameters of those attention heads. To prune, we again used L1-norm-based pruning.

3.3.4 Combined Techniques

After exploring pruning and quantization separately, we combined these techniques by first pruning the models and then performing quantization. We followed this sequence because pruning effectively reduces the model complexity and size beforehand, making the subsequent quantization step operate on a leaner, more structured network. This ensures that quantization is applied to an already optimized and smaller model, potentially also improving quantization efficiency and model accuracy post-compression.

Overall, we were able to construct our best model by choosing the order and styles of model compression techniques for pruning and quantization, and by migrating to CoreML-based quantization workflows to achieve more accurate and hardware-aligned models on iOS devices. The order of pruning before quantization emerged as a logical and beneficial strategy for achieving both reduced model size and high performance on target hardware.

4 Related Work

4.1 Task-Specific/Application Side

Existing image restoration methods vary widely in complexity and focus. Approaches like EDSR [3] and RCAN [5], which build on traditional CNN frameworks, are good at enhancing local details but sometimes struggle to incorporate broader contextual cues. SwinIR [2], on the other hand, leverages hierarchical Swin Transformer architectures to capture both local and global features through a three-stage pipeline: shallow feature extraction, deep feature extraction, and image reconstruction. This helps enables it to handle complex textures and global image structures more effectively than models relying purely on convolutional layers.

Beyond these CNN-based solutions, GAN-oriented methods (e.g., ESRGAN or SRGAN) can produce visually striking results, but they typically require significant computational resources. Give our resources, we found them to be somewhat impractical for deployment on devices with limited memory and compute power.

Our work differs from these existing methods by addressing the challenge of deploying high-quality image restoration models directly on mobile hardware. Instead of assuming data-center level resources or sacrificing detail for speed, we focus on refining a transformer-based model for on-device inference. Through pruning and quantization, we aim to maintain the rich contextual understanding and detail preservation that advanced architectures like SwinIR offer, while simultaneously ensuring that the resulting model is compact and efficient enough for real-time operation within the strict computational and memory constraints of mobile devices.

4.2 Efficiency and Optimization Strategies

TPrune is an area of related research focused on efficiency improvements. The TPrune pipeline utilizes Block-wise Structured Sparsity Learning (BSSL) to analyze and prune transformer models. It also uses Structured Hoyer Square (SHS) for line-wise pruning. It extends the SHS regularizer from CNNs to Transformer models. TPrune achieves 1.16×–1.92× speedup on mobile devices with minimal BLEU score degradation (0%–8%), outperforming prior methods in balancing model compression and accuracy. The pruned Transformer models are evaluated directly on mobile devices to demonstrate real-world performance improvements. Additionally, the method achieves higher compression rates while maintaining minimal performance degradation [4].

This paper's techniques for Transformer pruning and optimization are relevant to SISR, where efficient models are crucial for deployment on resource-constrained devices. Pruning methods like TPrune can reduce latency and resource usage in real-time SISR applications without significantly sacrificing performance. These methods could help optimize the Swin Transformer by increasing parameter sparsity to achieve faster inference and lower memory consumption while maintaining image quality.

Our work differs from TPrune in that, while

TPrune's pruning strategies are effective for language models, they may not directly address the intricate high-frequency detail requirements of SISR tasks. Furthermore, our focus on iPhone-specific hardware constraints sets our approach apart, as TPrune's general-purpose optimizations are not inherently tailored to this mobile environment.

5 Experimental Results

5.1 Performance Metrics

As seen in Table 4, 5, and 6 are the experimental results for our Real, Classical, and Lightweight SR models, respectively. The tables show a comparison between metrics over differenet technqiues. The first row is the baseline model without any compression or optimization. This is followed by structured attention head pruning then finally unstructured pruning. Across all models, the baseline model is the largest in terms of parameters, disk and CoreML size. Accuracy measures like PSNR and SSIM for classical and real SR models are highest with the baseline models compared to models after optimization. This differs from lightweight SR models where accuracy measures like PSNR and SSIM actually increase with some compressed models.

5.2 Efficiency vs. Accuracy Trade-off

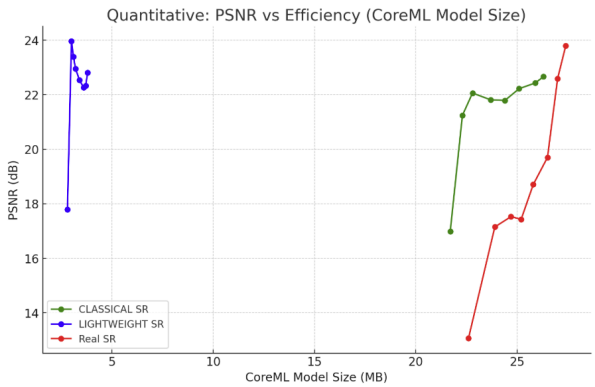


Figure 3: Quantitative efficiency and accuracy trade-off curve. Efficiency is shown on the x-axis as model size (MB) and PSNR (dB) accuracy on the y-axis.

When comparing PSNR vs. efficiency, as we see in Figure 3, there are a few interesting observations. For each model type, classical, lightweight and real SR, the right most point is our uncompressed baseline model. Then moving from right to left, each point is structured attention head pruning with model sparsity increasing by roughly 5%. This remained true until the left most point for all models, which correlates to the unstructured pruning of the respective model. Recalling that PSNR is the measure of the output quality of the restructured image, we can see that for Real SR and Classical SR Models, PSNR drops as model size decreases and more attention heads are pruned. This drop in accuracy as the model is increasingly compressed is more sudden and steep for Real SR Models compared to classical SR model, which gradually decrease and hover around a PSNR value of 20 dB. However, for the lightweight model, PSNR initially decreases but then increases past its' baseline as more attention heads are pruned. For all models, unstructured pruning results in a steep decline in accuracy.

For the qualitative accuracy vs. efficiency comparison, we assigned scores for how good the output looked using our visual judgment. The results can be seen in figure 4. For classical, real and lightweight SR we see a drop in score for the models that underwent unstructured pruning. Real SR had remained relatively the same for increasingly more attention heads pruned while classical and lightweight SR models' qualitative accuracy score increased as more attention heads were pruned.

What these trade-off curves show us is that unstructured pruning leads to worse preform ace across models. For attention pruning, more heads pruned seems to increase performance, both qualitatively and quantitatively for lightweight models. For classical models, more heads pruned might make the image look better qualitatively but worse when measured using PSNR. As attention heads are pruned for classical models, the seem to improve using visual judgment but slightly drecrease using quantitative accuracy measurements.

5.3 Performance-Under-Budget Curve

The model performance (PSNR) vs inferences under a 10Wh budget are illustrated in Figure 5. For this analysis we evaluated the average inference time per sample for each baseline model, and from 1 - 6 pruned heads. We exclude unstructured pruning, and quantization from this study because the qualitative results were extremely poor.

From our ablations we find that sparsity has minimal impact on energy usage, likely due to the fact that sparse representations don't directly reduce the number of FLOPS required for these

Technique	Remaining Params	Iters	% Sparsity	Disk Size (MB)	SSIM	PSNR (dB)	CoreML Model Size (MB)	Latency (ms)
Baseline	11,678,631	0	0	288.5	0.7441	23.79	27.4	1160
SP - AH	11,095,867	1	4.99	276.86	0.7097	22.59	27	1154
SP - AH	10,511,936	2	9.99	265.2	0.5844	19.7	26.5	1129
SP - AH	9,929,031	3	14.98	253.53	0.5524	18.71	25.8	1104
SP - AH	9,345,831	4	19.97	241.86	0.4707	17.43	25.2	1068
SP - AH	8,762,631	5	24.97	230.2	0.4524	17.53	24.7	1040
SP - AH	8,179,431	6	29.96	218.54	0.4165	17.15	23.9	965
UP	7,824,683	1	33	194.27	0.462	13.07	22.6	920

Table 4: Real SR Model Results: SP - AH = Structured Pruning - Attention Heads, UP = Unstructured Pruning.

Technique	Remaining Params	Iters	% Sparsity	Disk Size (MB)	SSIM	PSNR (dB)	CoreML Model Size (MB)	Latency (ms)
Baseline	11,752,487	0	0	287.05	0.7724	22.66	26.3	1281
SP - AH	11,169,287	1	4.96	275.39	0.7633	22.43	25.9	1174
SP - AH	10,586,087	2	9.92	263.73	0.7537	22.22	25.1	1120
SP - AH	10,002,887	3	14.89	252.06	0.7343	21.79	24.4	1078
SP - AH	9,419,687	4	19.85	240.39	0.7301	21.81	23.7	1055
SP - AH	8,836,487	5	24.81	228.73	0.7233	22.06	22.8	1080
SP - AH	8,253,287	6	29.77	217.07	0.651	21.24	22.3	1013
UP	7,874,166	1	33	193.23	0.6807	16.99	21.7	1035

Table 5: Classical SR Model Results: SP - AH = Structured Pruning - Attention Heads, UP = Unstructured Pruning.

Technique	Remaining Params	Iters	% Sparsity	Disk Size (MB)	SSIM	PSNR (dB)	CoreML Model Size (MB)	Latency (ms)
Baseline	910,152	0	0	34.11	0.7782	22.81	3.8	580
SP - AH	866,952	1	4.75	33.24	0.7638	22.33	3.7	592
SP - AH	823,752	2	9.49	32.38	0.7528	22.26	3.6	550
SP - AH	780,552	3	14.24	31.51	0.7591	22.53	3.4	539
SP - AH	737,352	4	18.99	30.65	0.765	22.95	3.2	460
SP - AH	694,152	5	23.73	29.79	0.7629	23.39	3.1	451
SP - AH	650,952	6	28.48	28.92	0.7804	23.96	3	423
UP	609,802	1	33	26.59	0.5708	17.79	2.8	435

Table 6: Lightweight SR Model Results: SP - AH = Structured Pruning - Attention Heads, UP = Unstructured Pruning.

operations to a great extent. The largest gains for number of inferences comes from switching to a smaller model, the *lightweight_sr* model shows comparable performance in terms of PSNR with almost an order of magnitude more inferences under the same budget.

The main implication for this curve is that in an energy constrained environment, the most impactful change would be switching the model size from *real_sr* or *classical_sr* to *lightweight_sr*

6 Discussion

6.1 Key Challenges

Converting the models to a digestible CoreML framework posed some challenges early in our experiments. There is not substantial documentation on how CoreML converts PyTorch models to CoreML packages, so we feared there would be some information loss in the conversion which would hinder our model's performance on-device.

We also encountered a number of different challenges when applying PyTorch-based quantization methods. We initially explored both static

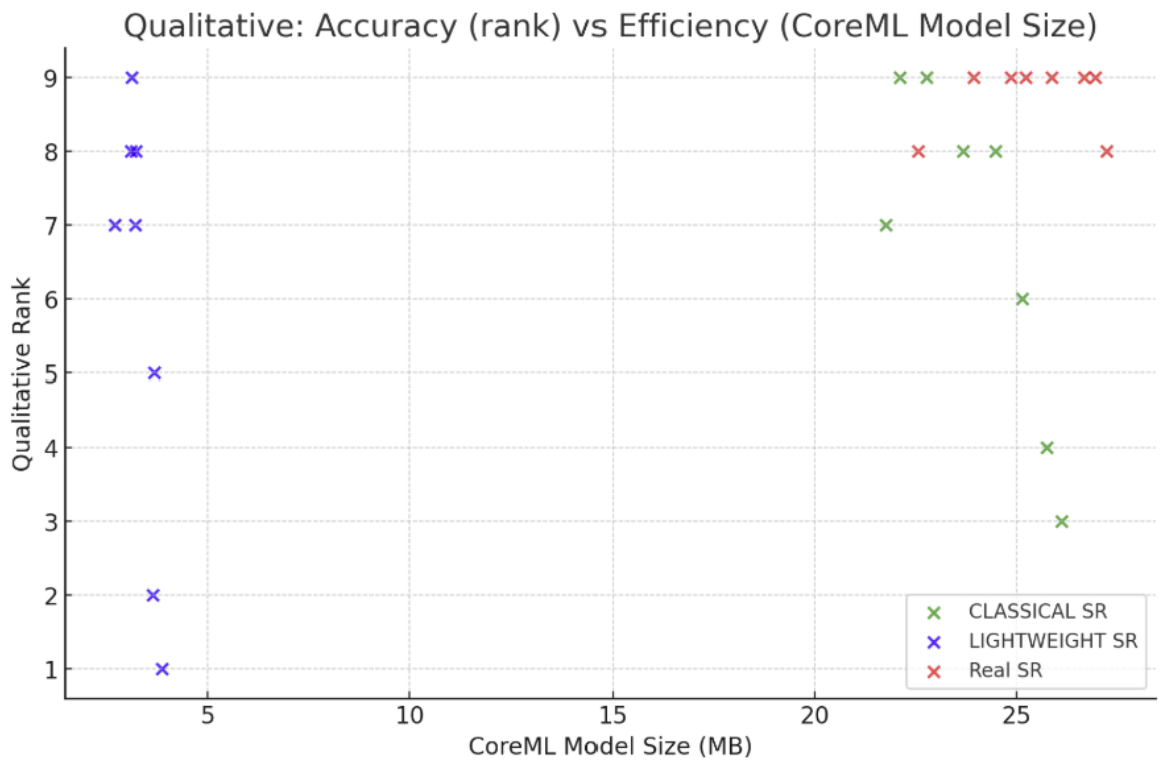


Figure 4: Qualitative efficiency and accuracy trade-off curve. Efficiency is shown on the x-axis as model size (MB) and qualitative rank as accuracy on the y-axis.

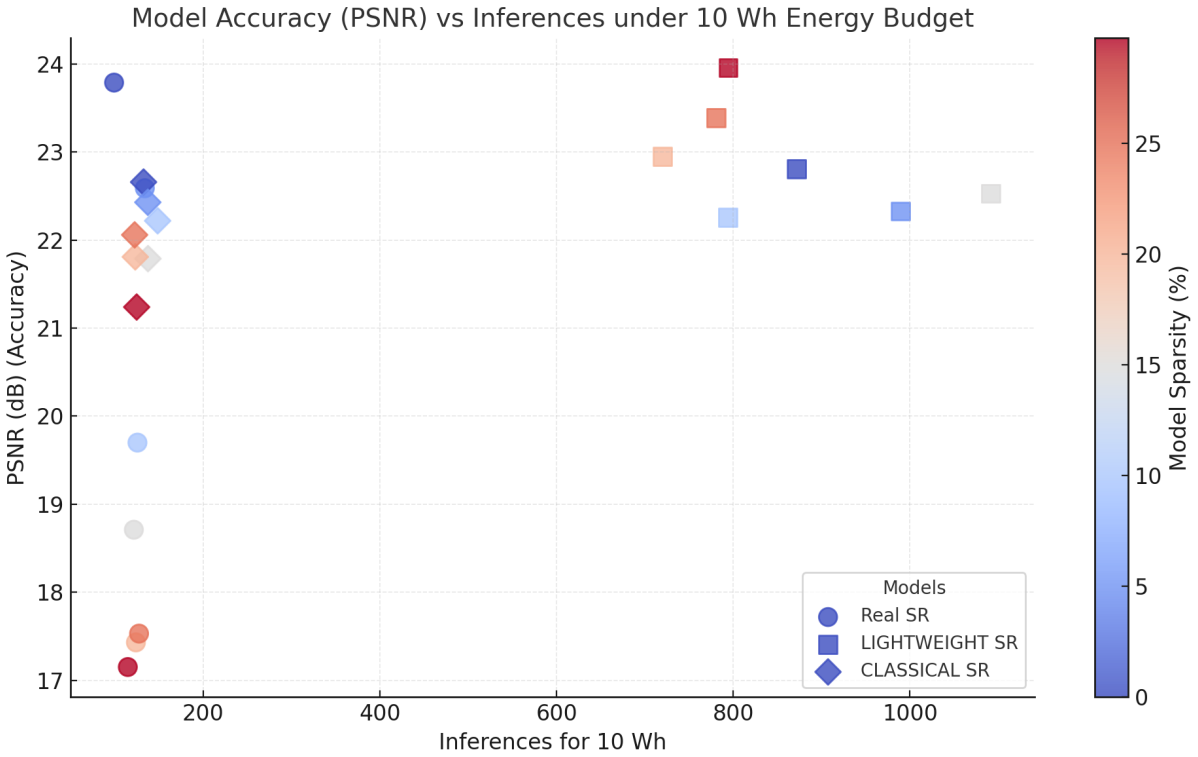


Figure 5: Number of inferences under a 10Wh budget. Different models are represented with different markers, and the hue of a data point encodes its sparsity.

and dynamic quantization in PyTorch, as well as mixed-precision training, all using FP16 as the target datatype. Unfortunately, these approaches often led to considerable performance degradation. When converted into the CoreML format, the PyTorch-quantized models also exhibited datatype inaccuracies in the iOS simulator environment. To address these issues, we pivoted to using CoreML's quantization workflows, which better aligned with the underlying iPhone hardware capabilities and led to more accurate and hardware-efficient models.

6.2 Insights and Takeaways

Some valuable insights and takeaways we gained through our experiments were:

- **Quantization** is particularly difficult for image enhancement tasks, which require precise numerical calculations to maintain fine visual details. Lower precision formats like int8 are more effective for classification tasks, where detail reproduction is less critical.
- **Unstructured Pruning** is highly effective for reducing storage requirements but detrimental to model performance, especially at higher sparsity levels.
- **Structured Attention Pruning** preserved image quality and was particularly advantageous for tasks requiring high-resolution outputs.
- **Model Complexity & Quantization Sensitivity**: Larger models seem more sensitive to quantization potentially because SISR and image reconstruction tasks rely on precise, high-fidelity computations and diverse weight distributions to accurately capture fine-grained details and textures.

6.3 Future Work

- **Selective Quantization:** Apply quantization to less sensitive layers, such as those involved in early feature extraction, while keeping critical layers in full precision.
- **Selective Structured Pruning:** Testing which attention heads in the transformer block to prune could be advantageous for future works. Oftentimes when pruning attention heads, some are more sensitive to others and thus will have varying impacts on model performance. In the future, it could be useful to identify which attanetion heads are best to prune, and how much to prune them.

7 Conclusion

In our project, we made significant strides in optimizing transformer-based image enhancement models for mobile devices, specifically focusing on adapting the SwinIR architecture for iPhone deployment. We developed comprehensive optimization experiments that balanced model compression with output quality, finding that structured attention head pruning was particularly effective while quantization proved challenging for image enhancement tasks. This enabled efficient on-device processing while maintaining high image quality.

Our work's impact extends beyond just image enhancement, demonstrating that complex transformer models can be effectively deployed on mobile devices with careful optimization. By achieving high-quality results without cloud processing, our approach enables privacy-preserving image enhancement and establishes a practical framework for future mobile-optimized computer vision applications. The insights we gained about different optimization techniques and their trade-offs provide valuable guidance for future work in on-device machine learning.

References

[1] 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.

[2] Jingyun Liang et al. *SwinIR: Image Restoration Using Swin Transformer*. 2021. arXiv: 2108.10257 [eess.IV]. URL: https://arxiv.org/abs/2108.10257.

[3] Bee Lim et al. *Enhanced Deep Residual Networks for Single Image Super-Resolution*. 2017. arXiv: 1707.02921 [cs.CV]. URL: https://arxiv.org/abs/1707.02921.

[4] Jiachen Mao et al. *TPrune: Efficient Transformer Pruning for Mobile Devices*. 2021. DOI: 10.1145/3446640. URL: https://doi.org/10.1145/3446640.

[5] Yulun Zhang et al. *Image Super-Resolution Using Very Deep Residual Channel Attention Networks*. 2018. arXiv: 1807.02758 [cs.CV]. URL: https://arxiv.org/abs/1807.02758.