الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research
جامعة غرداية
University of Ghardaia
Registration n°:
...../...../...../...../.....
كلية العلوم والتكنولوجيا
Faculty of Science and Technology
قسم الرياضيات والإعلام الآلي
Department of Mathematics and Computer Science

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master

Domain: Mathematics and Computer Science
Field: Computer Science
Specialty: Intelligent Systems for Knowledge Extraction

Topic

a DL model for image resolution enhancement and optimization for edge devices

Presented by:

Yahia DOUDOU & Bakir Saber CHEKHAR

Publicly defended on June 23, 2024

Jury members:

Mr. Slimane Oulad-Naoui	MCB	Univ. Ghardaia	President
Ms. Asma Bouchekkouf	MAA	Univ. Ghardaia	Examiner
Mr. Abdelkader Bouhani	MAA	Univ. Ghardaia	Supervisor

Academic Year: 2023/2024

In deepest gratitude, I first thank ALLAH for the guidance, strength, a sound mind, protection, the healthy leaving and for everything. This thesis is wholeheartedly dedicated to:

- **My beloved family:** My mother my father the source of unwavering inspiration, who offered strength during moments of weakness. Their continuous moral, spiritual, emotional, and financial support has been invaluable.
- **My brothers, sisters, relatives, mentor, friends, and classmates:** Your words of advice and encouragement have fueled my perseverance throughout this endeavor.

Acknowledgement

First of all, we thank Almighty God for giving us the strength and courage to carry out this work.

We are deeply thankful to our supervisor, Mr. Bouhani Abdelkader, for his guidance, understanding, and advice throughout this project.

We thank each member of the jury for this honor they do us by agreeing to judge our work.

We are grateful to our colleagues for their support and encouragement. Finally, We extend our appreciation to everyone who helped and encouraged us from near or far in the realization of this work to find here our gratitude and sincere thanks.

ملخص

في السنوات الأخيرة، اجتذب تحسين الصور القائم على التعلم العميق اهتماماً كبيراً وحققت نجاحاً كبيراً في وحدات معالجة الرسومات المتقدمة (GPUs). لكن معظم الأساليب الحديثة تتطلب عدداً كبيراً من المعلومات والذاكرة والموارد الحسابية، مما يؤدي غالبا إلى أوقات تنفيذ أقل على الأجهزة المحمولة الحالية.
في هذه الأطروحة، نقترح شبكة عميقة وبسيطة معززة بخوارزمية التفاف الجار الأقرب وتكميم 8 بت لتحقيق دقة فائقة للصورة الواحدة في الوقت الفعلي على وحدات المعالجة العصبية (NPUs). علاوة على ذلك، نقوم بتقييم كفاءة بنية شبكتنا من خلال مقارنة التجارب على الأجهزة المحمولة لاختيار العمليات المطلوب اختيارها. يتكون النموذج من 52 **ألف معامل** فقط، وينفّذ تكبيرا بمقدار 4× **في** 0.065 **ثانية** على هاتف يعمل بمعالج Snapdragon 865. وبالمقارنة مع خوارزميات تحسين دقة الصور الأخرى، وجدنا أن نموذجنا يمكنه تحقيق نتائج دقة فائقة عالية الدقة مع استخدام أزمنة استدلال أقل.

الكلمات المفتاحية: .تحسين الصور، تكميم، التفاف الجار الأقرب، وحدة المعالجة العصبية (NPU)

Abstract

In recent years, deep learning-based single image super-resolution (SISR) has attracted considerable attention and achieved significant success on advanced GPUs. Most state-of-the-art methods require a large number of parameters, memory, and computational resources, often resulting in inferior inference times on mobile devices.
In this thesis, we introduce a plain convolution network augmented with a nearest-neighbor convolution module and 8-bit quantization to achieve real-time SISR on NPUs. Furthermore, we evaluate the efficiency of our network architecture by comparing experiments on mobile devices to select the tensor operations to implement. The model comprises only **52 K parameters**, achieves **4× upscaling in 0.065 s** on a Snapdragon 865 CPU smartphone, and by comparing to other SR methods, we found that our model can achieve high fidelity super resolution results while using fewer inference times.

Keywords: Single image super-resolution (SISR), Quantization, Nearest-Neighbor Convolution, Neural Processing Unit (NPU).

Résumé

Ces dernières années, la super-résolution d'image unique (SISR) basée sur l'apprentissage profond a suscité un intérêt considérable et a rencontré un succès considérable sur les GPU avancés. La plupart des méthodes de pointe nécessitent un grand nombre de paramètres, de mémoire et de ressources de calcul, ce qui entraîne souvent des temps d'inférence inférieurs sur les appareils mobiles.
Dans cette thèse, nous présentons un réseau de convolution simple, complété par un module de convolution par le plus proche voisin et une quantification 8 bits, pour obtenir une SISR en temps réel sur les NPU. De plus, nous évaluons l'efficacité de notre architecture réseau en comparant des expériences sur des appareils mobiles afin de sélectionner les opérations tensorielles à implémenter. Le modèle ne comprend que **52 K paramètres**, effectue un **agrandissement 4× en 0,065 s** sur un smartphone équipé d'un processeur Snapdragon 865. En le comparant à d'autres méthodes de SR, nous avons constaté que notre modèle permet d'obtenir des résultats de super-résolution haute fidélité tout en réduisant les temps d'inférence.

Mots Clée : super-résolution d'image unique, quantification, convolution du plus proche voisin, Unité de traitement neuronal (NPU).

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence.

API Application Programming Interface.

BN Batch Normalization.

CNN Convolutional Neural Network.

CPU Central Processing Unit.

DL Deep Learning.

FLOP Floating Point Operations Per Second.

GAN Generative Adversarial Networks.

GPU Graphics Processing Unit.

HR High Resolution.

LR Low Resolution.

ML Machine Learning.

MSE Mean Squared Error.

PReLU Parametric Rectified Linear Unit.

PSNR Peak Signal-to-Noise Ratio.

QAT Quantization Aware Training.

RAM Random Access Memory.

ReLU Rectified Linear Unit.

RGB Red, Green, Blue Colors.

SISR Single Image Super Resolution.

SR Super Resolution.

SRCNN Super-Resolution Convolutional Neural Network.

SRGAN Super-Resolution Generative Adversarial Network.

TFLite TensorFlow Lite.

Introduction

Image enhancement or Image Super Resolution (ISR) as they are called in computer vision field, is a task aimed at enhancing low-resolution images by recovering high-frequency details. Recently, with the advancement of deep learning approaches [31], impressive results in single image super-resolution have been achieved. It has been applied to many real-world applications, such as digital photography, video compression, file transmission, and even for the latest image generation model like Midjourny and Stable Diffusion as a part of its process. However, the big problem with these high-performance models is requiring a vast computational resource, which leads to limiting their deployment on edge and mobile devices with strict constraints of low resources. As a consequence, there is a growing need to develop light and efficient networks that are friendly to low-resource devices to overcome this problem.

To address this challenge, there is increasing interest in developing lightweight and efficient network architectures capable of running in real time on mobile hardware without sacrificing output quality. Bringing such models to devices like smartphones and embedded AI accelerators would significantly expand the practical use of super-resolution, making it accessible for daily applications where performance and efficiency must be balanced.

In our thesis, we present different studies in image Super Resolution, from the traditional image enhancement approaches to the first deep learning state of the art algorithms SRCNN [12], SRGAN [31] passing to the lightweight efficient one and their strategy to design a friendly network system, such as re-parameterization, FLOP reduction, and network quantization.

We propose a new lightweight efficient architecture inspired by [15], with just 7 convolutional layers and the nearest-neighbor method to reconstruct the desired high-quality image.

The thesis consists of four chapters. In the first chapter, we present a background and some basic concepts about convolutions neural network (CNN), Machine Learning (ML), and then some details of the image super resolution. In the second chapter, we present a detailed study of different approaches in image super-resolution and pass to the strategy

used to overcome the complex models. In the third chapter, we provide a comprehensive understanding of the steps we took to develop and train our image enhancement model, as well as how we evaluated its performance. In the last chapter, we overview our experiment results, the quantization process, and the performance of our model on mobile devices. We discuss and analyze the results we obtained, insisting on the strengths and advantages of the network.

Chapter 1

Background

In today's life, image is an important source for users to exchange and obtain abundant information in the form of pictures, videos, graphics, two-dimensional (2D) data, three-dimensional (3D) data, etc. Due to that the quest for higher image resolution is a persistent challenge with widespread applications. Whether in medical imaging, surveillance, or various other fields, the demand for clearer images is constant. However, achieving this clarity presents significant obstacles. In this chapter, we delve into the world of image resolution enhancement, exploring advanced technologies such as convolutional neural networks (CNNs) and generative adversarial networks (GANs). These technologies hold the potential to reshape how we perceive and utilize high-resolution images. We'll uncover the intricacies of this field, bridging the gap between art and science, where pixels become the canvas for digital transformation, and innovation expands the boundaries of visual perception.

1.1 Machine learning

It is the science of teaching computers to learn from data through a range of algorithms that iteratively acquire knowledge from data. As the algorithms absorb more training data, it becomes easier to generate increasingly accurate models. As your machine learning algorithm is trained, it develops a model [23].

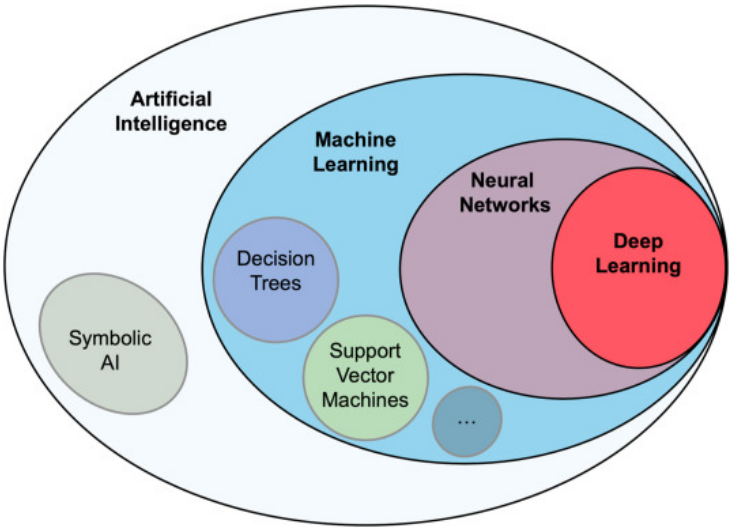

Figure 1.1: Artificial intelligence encompasses machine learning deep learning and neural networks.

1.1.1 Supervised Learning

In Machine Learning and Deep Learning, Supervised Learning is the labeled learning paradigm. It refers to supervising machine learning by showing it examples (data) of the task to be performed [48]. The supervised learning method is used for a wide variety of problems: computer vision, regression, classification... Most Machine Learning and Deep Learning problems involve supervised learning.

1.1.2 Unsupervised Learning

Unsupervised learning is best suited when the problem requires a massive amount of data that is unlabeled. For example, social media applications, such as twitter, Snap chat, and so on all have large amounts of unlabeled data.

1.2 Deep Learning

Deep learning (DL) is a subset of machine learning that focuses on training artificial neural networks with multiple layers (hence "deep") to learn patterns and representations from data. It is inspired by the structure and function of the human brain.

1.2.1 Neural network

Typically, a neural network has three layers: an input layer, one or more hidden layers, and an output layer. Using the input layer, data is ingested. Based on the weights applied to the nodes, the data is modified in the hidden layer and in the output layer. There are thousands or millions of simple processing nodes in a typical neural network [6].

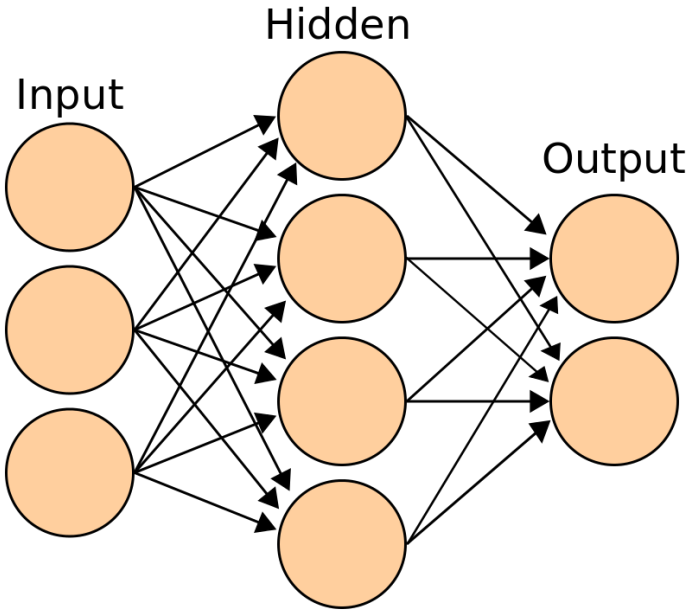

Figure 1.2: The architecture of a neural networks.

1.2.2 Components of a Neural Network

1.2.2.1 Convolutional Layer

A convolutional layer is a fundamental component of a convolutional neural network (CNN). It receives a three-dimensional input image, typically represented as a matrix of pixel values with height, width, and depth (corresponding to the RGB color channels). The layer applies a set of learnable filters, also called kernels or feature detectors, which slide over the input image to detect the presence of specific features such as edges, textures, or patterns. This process is known as the convolution operation, and it is essential for extracting spatial hierarchies of features from the input image[30].

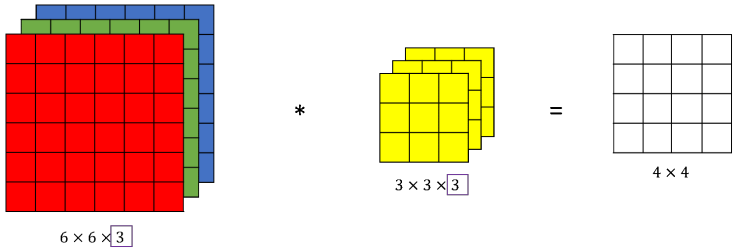

Figure 1.3: Convolutional layer.

1.2.2.2 Filters

The filter or kernels consists of a two-dimensional (2D) array of weights, which represents part of the image. While they can vary in size, a 3x3 matrix is usually used, this also determines the size of the receptive field. the convolution can be:

- **Valid padding:** It is also known as no padding. It simply drops the last convolution if its dimensions don't match.
- **Same padding:** This padding ensures that the output layer has the same size as the input layer.
- **Full padding:** In this type of padding, zeros are added to the input border to increase the output size.

1.2.2.3 Activation Function

In CNN, the activation function enables nonlinear mappings between layers, which allows the network to learn complex representations of the data. It would not make sense to have several hidden layers if they did not behave in a non-linear manner, as one hidden layer could achieve the linear transformation[17].

1.2.2.4 Batch Normalization

A technique used to improve the stability and speed of neural networks. It normalizes the outputs of each layer to ensure that activations remain well-scaled. By doing so, it helps the networks learn more efficiently and reduces the chances of unstable training behavior.

1.2.2.5 Pooling Layer

Is a technique used to reduce the spatial demension of feature maps while retaining important information. The pooling process sweeps a filter across the entire input, similar to the convolutional layer, but the filter does not have weights. By aggregating values

from the receptive field, the kernel populates the output array. The pooling layer consists of two categories:

- **Max pooling:** When the filter moves across the input, it selects the pixel with the highest value to send to the output array. This approach is more common than average pooling.
- **Average pooling:** By moving across the input, the filter computes the average value of the receptive field to be sent to the output array.

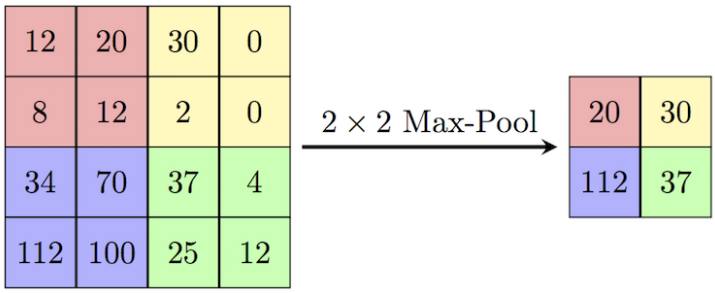

Figure 1.4: effect of using max pooling.

1.2.2.6 Fully-Connected Layer

Fully connected layers are self-explanatory. In partially linked layers, the input image pixel values are not directly connected to the output image. Alternatively, each output node connects directly to a node in the previous layer in the fully connected layer. In this layer, classification tasks are performed based on the features retrieved by the previous layers and their various filters. Fully connected layers use soft max activation functions to produce probabilities from 0 to 1, while convolutional and pooling layers often use ReLU functions to categorize inputs.

1.2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) proposed by Yann Lucan[30], are the main architecture behind computer vision applications. They consist of three different types of layers:

- Convolution layer.
- Pooling Layer.
- Fully Connected Layer.

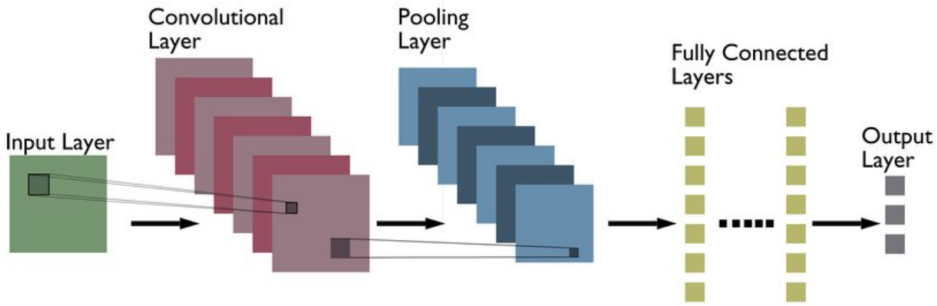

Figure 1.5: The main component of CNN architecture.

1.3 Image Super Resolution Principle

1.3.1 What is image super resolution?

Super-Resolution (SR) is a branch of Artificial Intelligence (AI) that focus on enhancing the resolution and recovering fine details of low-resolution images by using models called image super-resolution. See figure 1.6. Those models are used to replicate a high-resolution image (HR) from a lower-degraded version (LR), filling in missing high-frequency information in the resulting super-resolution image (SR).

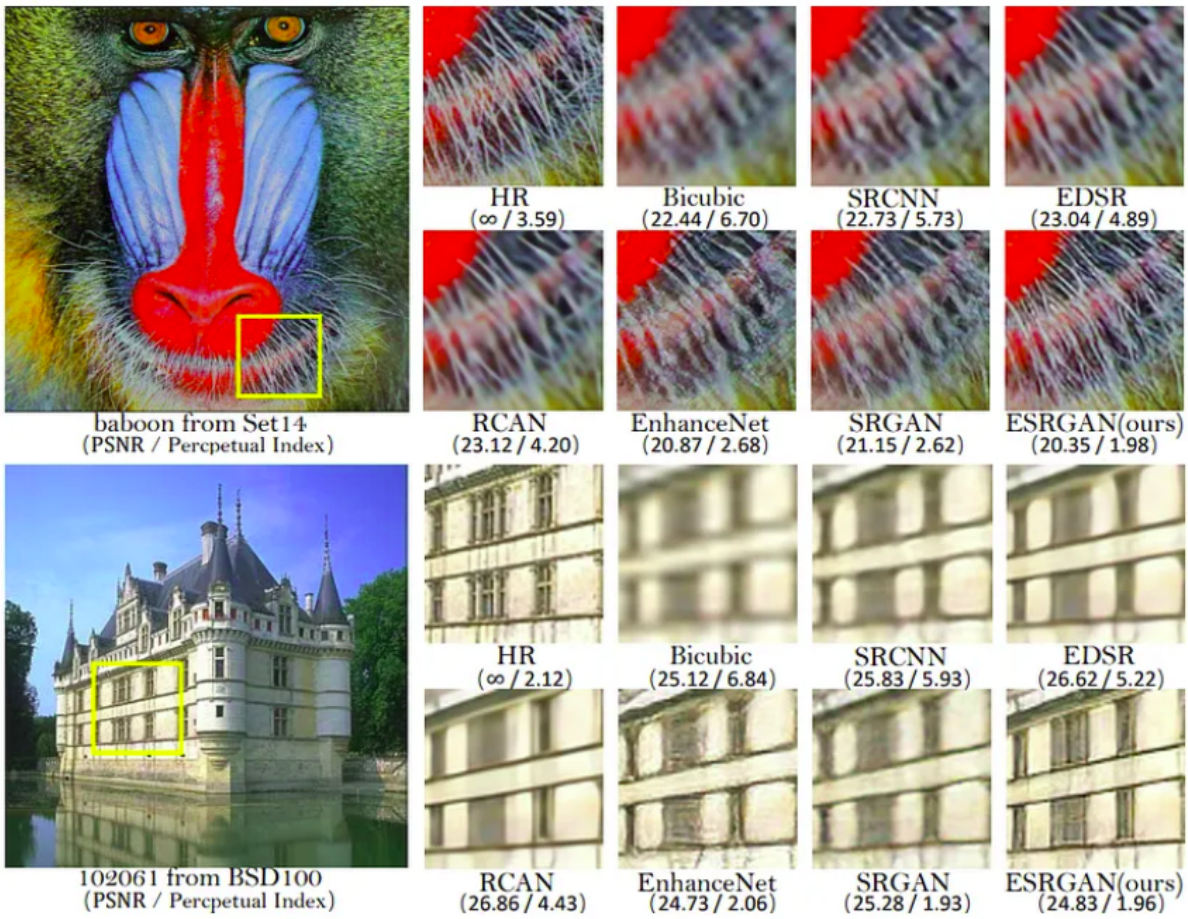

Figure 1.6: example of the results produced by various SR methods, image from [59].

1.3.2 Image Degradation Model

Image super-resolution, as the primary goal of our research, involves the recovery of a high-resolution image (HR) from a given low-resolution (LR) or degraded input. The LR image denoted by I_{LR} can be represented as the output of the degradation function, as shown in eq 1.1:

$$I_{xLR} = d\left(I_{yHR}, \delta\right) \tag{1.1}$$

Where d is the SR degradation function, that is responsible for the degradation of HR image to LR image, I_{HR} is the HR image, (x, y) are the spatial coordinates (pixel position) in the image, and δ are the parameters for the function d. Degradation is a complex process that's unknown. It's influenced by a lot of things, like noise (sensor, speckle), compression, blur (defocus and motion) [60]. Figure 1.7 shows a sketch map of the degradation procedure:

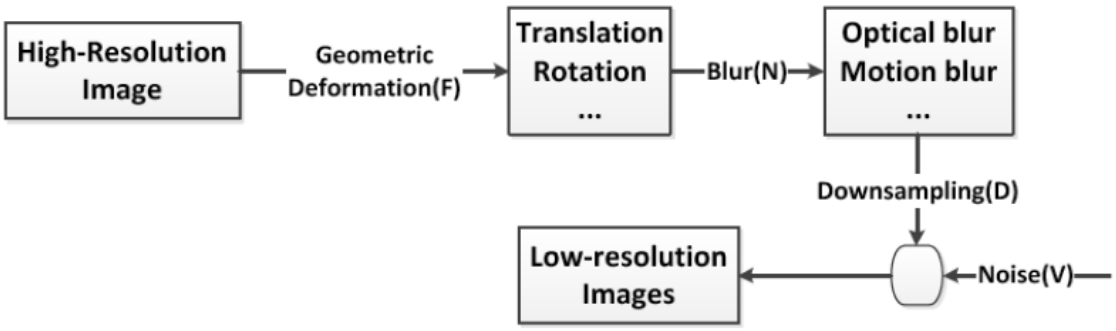

Figure 1.7: image degradation model

However, contrary to eq 1.1, most research studies prefer the following degradation model:

$$I_{xLR} = (I_{yHR} \otimes k) \downarrow_s + n \tag{1.2}$$

With k and n as the blurry kernel and noise, respectively. $I_{yHR} \otimes k$ as the convolution between the HR image and the blur kernel, and $\downarrow_s$ as a downsampling operation with a scaling factor s. This study focuses on the degradation caused by downsampling operation, with commonly used scaling factors such as 2, 3, 4 or 8. Bicubic interpolation is frequently employed as a standard downsampling method. The case of noisy images is beyond the scope of this study.

1.4 Conclusion

In this chapter, we give a brief overview of Machine Learning (ML), Convolutional Neural Networks (CNNs), and an introduction to image super resolution is presented. In the next chapter we will discuss the different approaches of image super resolution.

Chapter 2

Related work

This chapter discusses the developing deep learning models for image super resolution on mobile and edge devices. Early convolutional neural networks improved over traditional methods, but are often too large for constrained hardware. More advanced architectures, such as adversarial generative networks and transformers, further boost results, but remain challenging to deploy efficiently. Key techniques aim to make models lightweight through techniques such as pruning parameters, quantization, and restructuring network computations. Optimizing for both speed and accuracy simultaneously on real devices poses challenges. Emerging approaches explore multi-objective optimization and hardware co-design.

2.1 Image Resolution Enhancement Techniques

In these sections, we discuss three kinds of traditional image enhancement techniques for super-resolution: interpolation-based methods [61], reconstruction-based methods [45], and learning-based methods [16].

2.1.1 Interpolation-based Methods

Interpolation-based single-image super-resolution methods,(e.g: bilinear, bicubic, and nearest neighbor) can be used to resize images, zoom them in, enhance images, reduce images, register subpixels, decompose images, and correct spatial distortion [47].

primarily rely on mathematical techniques. and the core concept is to increase the size of an image and estimate the value of an unknown pixel by considering its neighboring pixels. Compared to other methods, these methods are computationally efficient and straightforward and have the least computational complexities; however, they lack accuracy and often produce smoothed results with a loss of fine details [62].

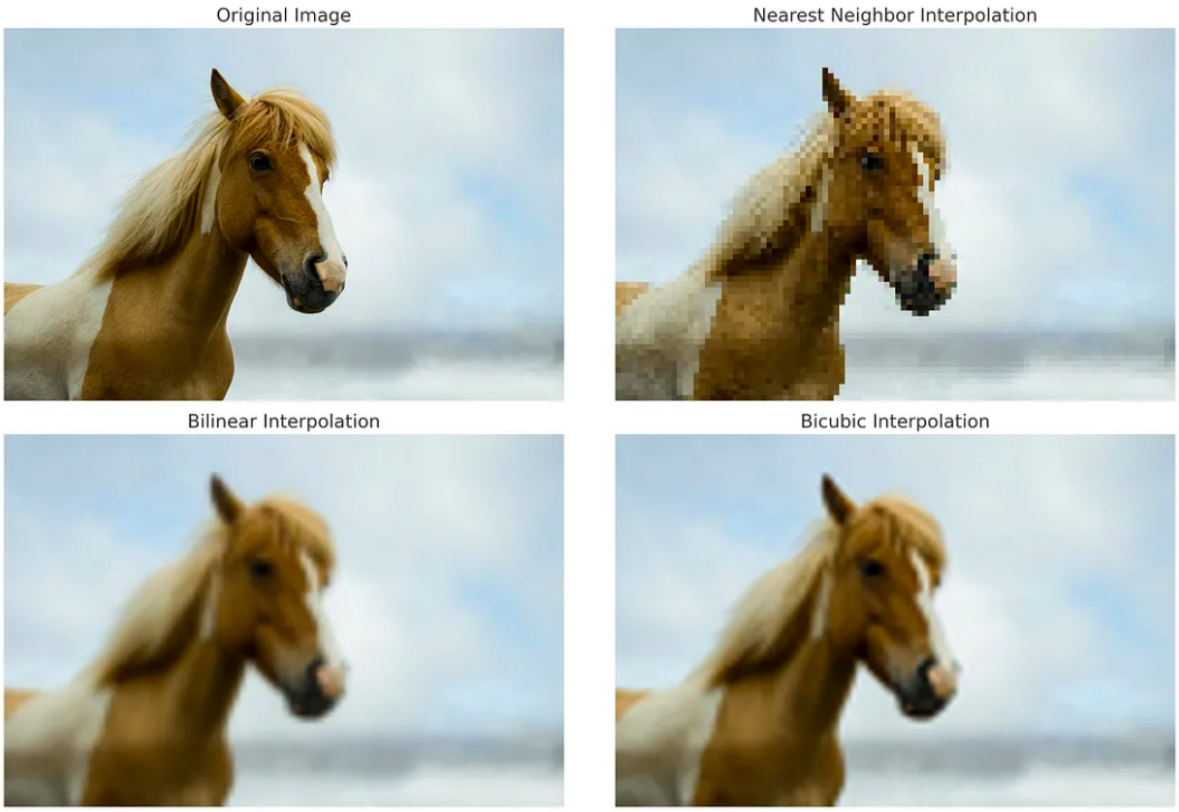

Figure 2.1: Comparison between interpolation-based methods. Among these, Nearest Neighbor (top right) produces the most jagged artifacts, Bilinear (bottom left) produces the softest results and Bicubic (bottom right) produces the best balance overall. The ground truth image (top left) can be observed for reference.

2.1.2 Reconstruction-based Methods

Super-resolution techniques that use reconstruction-based methods employ advanced knowledge to narrow down the possible solution space, leading to more distinct details. This approach involves applying linear constraints to the reconstructed high-resolution (HR) image that is derived from a low-resolution (LR) image. The procedure models image degradation by incorporating motion estimation and utilizing prior knowledge. The objective of these methods is to establish a forward observation model and solve the issue using an image degradation model (shown in figure 1.7). However, these algorithms may not be appropriate for large magnification factors or limited input images, as they might produce images that are excessively smooth and lacking in high-frequency details [5].

2.1.3 Learning-based Methods

A learning-based SISR method is also known as an example-based method because of its high computational speed and outstanding performance. These methods typically use machine learning algorithms to analyze statistical relationships between the (LR) and its corresponding (HR) counterparts from substantial training examples. Inspired by the sparse signal recovery theory researchers applied sparse coding methods [56] to SISR problems. In parallel, many researches have combined reconstruction-based and learning-based methods to reduce the artifacts brought by training example [66]. However, learning-based methods may face challenges when dealing with large upscaling factors, as the network's ability to generate fine details diminishes with significant scaling [69].

2.2 Deep Learning Methods for Super Resolutions

Image super-resolution (ISR), is meant to produce a high-resolution (HR) image from a counterpart low-resolution (LR) image that has been degraded, extensive methods of deep learning have been applied to solve this task, ranging from the early method based on convolutional neural networks (CNNs) [12, 13, 35, 51, 67] to recent promising (SR) approaches based on generative adversarial networks (GANs) [31, 59], and transformers [14, 38, 57], which have become popular for their ability to recover intricate image details. They find practical applications in tasks like improving image and video quality, transitions, and display [63].

1. **CNN-based Methods**

   Our first discussion will be about Image Super-Resolution utilizing Deep Convolutional Neural Networks (SRCNN) [12], The authors successfully acquired to learn the mapping function between (LR) images and their corresponding (HR) images through a training process. As a result, the learned model could reconstruct the HR image from its corresponding LR input image. See figure 2.2. SRCNN model included a three-layer structure. also included a pre-processing step, where the input image was upscaled using bicubic interpolation to the desired (HR) size. The goal was to produce an image $F(Y)$ as similar to the ground truth image X as possible, assuming that Y represents the LR image (with the same size as the HR).

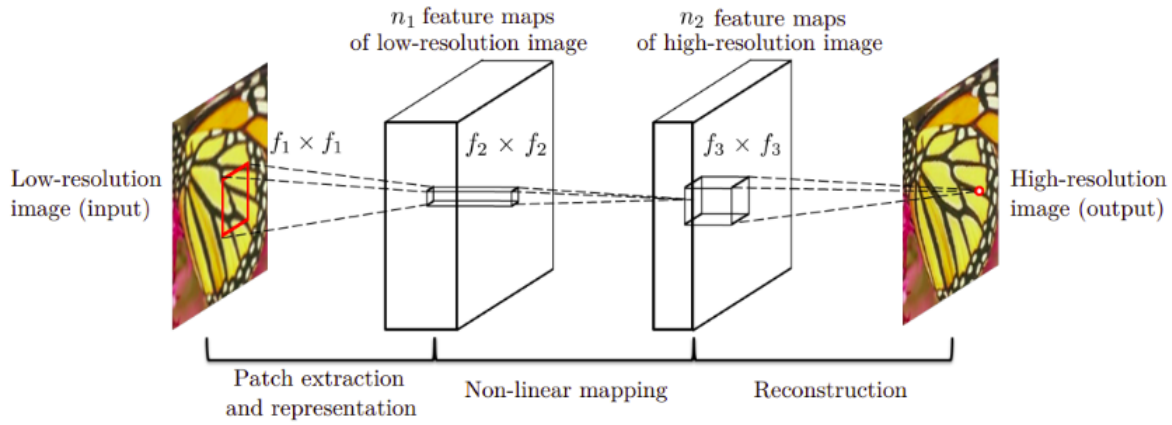

Figure 2.2: General architecture of the SRCNN model [12].

The above figure 2.2 shows the general architecture of the SRCNN model. It also shows the kernel sizes $f_x \times f_x$ and filters (n_1 and n_2) for the different convolutional layers. We can see clearly that the model takes a low resolution image patch, Y, and outputs a high resolution image, $F(Y)$. According to the authors, this process requires three operations are explained next.

(1) **Patch Extraction and Representation:** First, extracting overlapping patches from the low-resolution images. These patches will be fed to the SRCNN model, while the high-resolution patches will serve as ground truth.

(2) **Non-linear mapping:** During this phase, the model will learn the mapping, That is. it will learn to map the low-resolution patches to the high-resolution ones.

(3) **Reconstruction:** The final layer, where the model generates the final high-resolution image by aggregating the high resolution patches from the second layer.

As we see the SRCNN With such a lightweight structure composed with 3 stages, outperformed most of traditional SR algorithms. Moreover, it provided a reasonable theoretical interpretation on its inherent mechanism.

Inspired by SRCNN, Several other models are presented to improve the performance of deep learning methods based on CNNs. Most of them extract local features with spatially invariant kernels, which are inflexible to model pixels' relationships. Moreover, to enlarge the receptive field, usually, they employ complex and very deep network topologies [9, 42, 44, 67] to recover more details, resulting in much computational resource consumption. Among them, A residual-in-residual structure

and channel attention are proposed by Zhang et al [67] to train an extremely deep network over 400 layers, whereas dense blocks [20] are employed in memory network for image restoration method in [55] and residual dense network in [68] to utilize intermediate features across all layers. Additionally, some works, such as second-order attention network by [9], residual feature distillation network by [36], and Pocs based super-resolution image reconstruction [44] improve super resolution performance by analyzing feature correlations across spatial or channel dimensions.

2. **GAN-based Methods**

Different to previous works, generative adversarial networks (GANs) are most commonly used for SR applications [18]. As a result of GAN-based methods, HR images can be generated with much sharper details [31] allowing for a much more realistic reconstruction of texture details with a large upscaling factor.

Methods based on deep CNNs. It was described above, the majority of them trying to minimize the loss between (SR) and (HR) images by using Mean Squared Error (MSE) as the objective function. This optimization strategy is straightforward, easy to implement, and can be applied to a wide range of tasks, including pattern recognition and image processing. The (MSE), however, has some serious problems, such as unfaithful results and a poor visual appeal. As MSE loss assumes that noise is not a factor in image characteristics, it is incompatible with many applications. This delicate quality and structure, on the other hand, is more sensitive to human perception. Consequently, the perceptual and adversarial loss are introduced in order to solve this problem. Based on feature maps extracted from deep CNNs that have been pre-trained, the perceptual loss is calculated. With GANs, the term adversarial loss is introduced, meaning the adversarial loss guides the network to produce more realistic super resolution images.

In [31], Ledig developed the first generative adversarial network for SISR tasks, which included both generators and discriminators (fig 2.3). It uses a novelty learning strategy that combines a perceptual loss and an adversarial loss. With the help of these two losses, the SRGAN can improve the visual impact of SR images produced.

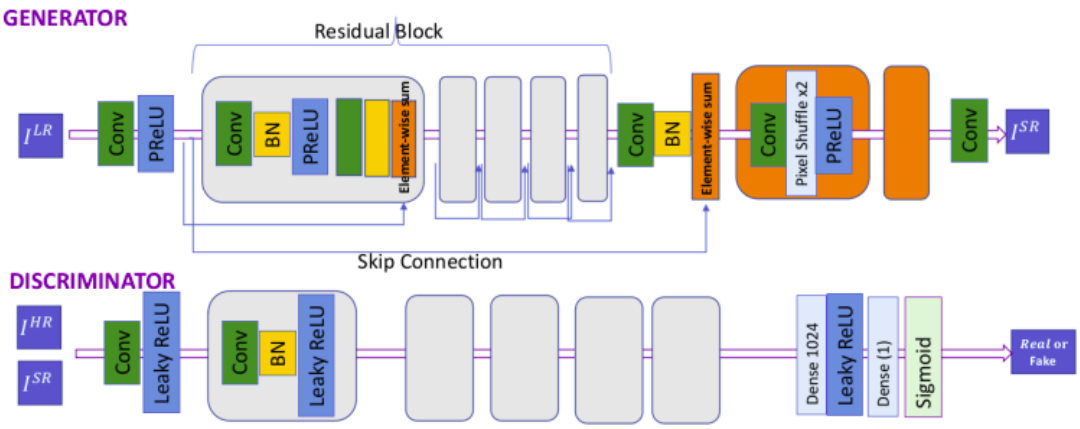

Figure 2.3: General architecture of the SRGAN model

The figure 2.3 shows the general architecture of the SRGAN model. It consists of two modules, the generator (G) and the discriminator (D). Each block has a generic deep network structure, such as convolution layers (Conv), batch normalization (BN), and parameterized ReLU (PReLU) in (G) with additional pixel-shuffle. A skip connection is also implemented in G much like it is in ResNet [19]. In contrast, the discriminator (D) uses Leaky ReLU non-linearity instead of PReLU. Furthermore, SRGAN is based on the structure of both the VGG [52] and DCGAN [49] networks.

Moreover, Wang et al [59]. Proposed an improvement to SRGAN called enhanced SRGAN (ESRGAN). Their approach enhances the SRGAN in three aspects: network architecture, discriminator, and objective loss. Compared to SRGAN, ESRGAN produces more desirable images. anouther GAN framework propodes by Park (SRFeat) [46]. Concerning perceptual loss and pixel-wise loss, SRFeat optimizes the training process and produces fine, and detailed results.

3. **Transformer-based Methods**

Recently, Transform-based methods have been demonstrated to be highly effective at modeling self-attention (SA) in input data in natural language processing [58]. A great deal of success has been achieved by transformers when it comes to natural language processing, which has led researchers to explore how they can be applied to computer vision problems. Multiple high-level vision tasks have produced interesting results [14, 38, 50]. Generally, the input image is divided into nonoverlapping patches of different scales as tokens, and the local feature extraction and SA modules are applied to the collection of patches. Despite its effectiveness, SA is limited to low-level tasks, such as SR, where feature sizes are typically large, because its complexity

increases quadratically with feature size.

Efforts have been made to make SA more efficient for super-resolution, but the large bandwidth consumption and computational complexity still make it time consuming. Mei et al [42] divided an image into nonoverlapped patches to model local features and SA independently, However, this approach can introduce artifacts at the patch borders and potentially reduce the visual quality of the final images.

Another approach, SwinIR [34], took inspiration from the Swin Transformer [39]. It starts by extracting local features using two successive 1x1 convolutions and then applies SA within small windows (like 8x8) with a shifting mechanism to establish connections with neighboring windows. However, using 1x1 convolutions with a limited receptive field might not capture enough information for long-range SA. Moreover, using small windows restricts the model's ability to capture long-range dependencies among pixels.

There's also Restormer [65], which tries to streamline SA by focusing on channel space, making it more efficient. However, this approach might sacrifice some important spatial information necessary for generating high-quality images.

While deep learning based models have been successful in reproducing fine image details, their deep and complex architectures pose challenges for practical deployment. The high computational demands and memory consumption associated with these models hinder their use in real-world applications, particularly on resource-constrained edge and mobile devices.

2.3 Model Optimization for Mobile and Edge Deployment

While many challenges and works employ deeper and more complicated network architectures for better image quality proposed recently, Their developed solutions are generally evaluated on desktop CPU and GPU, making the obtained solutions unusable due to the limited amount of RAM available on edge devices, the limited capacity of deep learning operators, and the power consumption limitation. as a result of that it is demanded to design lightweight and efficient super resolution models. We will now discuss the existing designs and optimizing strategies for lightweight and efficient SR for resource-constrained devices.

2.3.1 Efficient Super Resolution Network

The main focus of existing efficient SR network design methods is to reduce the number of parameters and FLOPs. Using a weight-sharing strategy, the parameter-reduction strategy decreases the model parameters without changing the computational complexity of the network. Using residual learning [27], the authors proposed a very deep SR network (VDSR) along with a deep recursive convolutional network (DRCN) [28] to reduce the number of parameters. And by combining residual and recursive learning, Tai et al [54] improved the recursive convolutional network algorithm in order to achieve better performance with fewer parameters.

As part of the latter strategy, low-FLOPs operations (such as group convolutions, depthwise convolutions, and element-wise operations) and FLOP-free operations (such as feature splitting, concatenation, and shuffling / reshaping) are introduced in order to reduce FLOP consumption while maintaining competitiveness with large SR models. In [2], Ahn et al. Presented an efficient cascading residual network (CARN) with group convolutions. Hui et al. [22] presented the idea of compressing the number of filters per layer by using an IDN (information distillation network). Their next step was to extend IDN to an information multi-distillation network (IMDN) [21]. As a result they won the constrained image SR challenge at AIM 2019.

With Liu et al. [37], the residual feature distillation block (RFDB) of the IMDN was further improved and the AIM 2020 SR challenge was won. For the purpose of finding a lightweight and effective SR network, authors in [6, 7] used FLOPs as a constraint of latency objective. It is important to note, however, that fewer parameters and FLOPs don't necessarily translate to greater efficiency on mobile devices. Recent studies have shown that not necessarily to lead into faster running speeds. In addition SR designs are most often studied on GPU servers, which cannot accurately represent the speed at which they run on limited devices.

2.3.2 Re-parameterization

Recentely, several studies have demonstrated the effectiveness of re-parameterization for a variety of high-level vision tasks, including image classification, object detection, and semantic segmentation. According to Arora et. al [3], reparameterizing the fully connected layer enables quicker network training as the network depth increases. The DiracNet [64] offers comparable performance to the ResNet series when training a plain CNN. in ACNet [10] As part of its structural re-parameterization, uses asymmetric convolution to enhance the normal convolution. RepVGG [11] boost the performance of traditional VGG to the level of series [19] on several high level vision tasks by decouples a normal (3x3)

convolution into multibranch block consisting of identity mapping, (1x1) convolution and (3x3) convolution.

Although previous re-parameterization blocks have been validated on high-level tasks, they have not been applied successfully to low-level vision tasks, sush super resolution problem

2.3.3 Low-Cost Computation (quantization and pruning)

A number of attempts have also been made to reduce computational costs while preserving SR performance. By quantizing features and weights of SR models, low-bit quantization [8, 32, 41] significantly reduces the model size and computational cost, as a resulte of reducing the number of bits required to represent each weight or feature component. in the MAI 2021 Challenge [25] Ayazoglu et al. [4] presented an extremely lightweight, quantization-robust real-time super resolution network. Du et al. [15], (Hence were we inspired our architecture) aim to design and deploy an efficient architecture for 8-bit quantization on a mobile device. To boost performance further, they proposed anchor-based plain nets (ABPNs) along with Quantization Aware Training strategies (QATs). In addition, many mobile devices employ the INT8 quantization since can accelerate the inference and save memory. see table 2.1.

Quantization	`#Parameters`	CPU	GPU Delegate	PSNR-int8
INT 8	42K	216ms	77ms	30.21
float 32	42K	215ms	62ms	30.06

Table 2.1: Runtime and PSNR on ABPN quantization model [15]

PSNR: The Peak Signa-to-Noise Ratio (PSNR) is a widely used metric to evaluate image quality. It measures the ratio between the maximum possible signal power and the power of corrupting noise that affects the image fidelity. A higher PSNR value indicates better image quality.

Model pruning can reduce computational costs as well. DHP [33] employs a differentiable pruning method via hyper networks for automatic network pruning, demonstrating its effectiveness on SR tasks. Some researchers wish to replace multiplication with more economical operations. AdderSR [53] replaces multiplication with addition, whereas GhostSR [43] generates redundant features using the shift operator. Nevertheless, for commodity mobile devices that only support 8/16/32 bit arithmetic calculations and limited operations, these special operations, including binary/ternary convolutions, fully adder convolutions, and shift operations, require customized hardware optimization to achive infernce speed. As a result of computational reduction techniques, we have primarily focused our thesis on efficient SR design based on quantization, which can be further accelerated.

Chapter 3

Methodology

In this chapter, we analyze the key components that constitute our research methodology. First we will start by invastigating with the help of table 3.1 and Meta-node Latency to find a perfect network architecture specifically for efficient deployment on mobile and edge devices. Then we delve into the crucial elements of dataset, network architecture, training strategies, model optimization, and the metrics used to evaluate the model's performance.

3.1 Proposed Image Enhancement Method

3.1.1 Overview

The goal of our thesis is to develop a real-time ISR model for resource-constrained devices. However, the limited amount of RAM, the inability to support many CNN operators, and the power consumption of mobile and edge devices prevent CNN deployment on mobile devices. In order to achieve this, it requires a careful design of the network architecture. In addition, we also use a full 8-bit Quantization-Aware Training (QAT) strategy and design neural architectures using hardware-friendly operations. The overall methodology, detailing these stages from raw image input to final model evaluation, is illustrated in Figure 3.1.

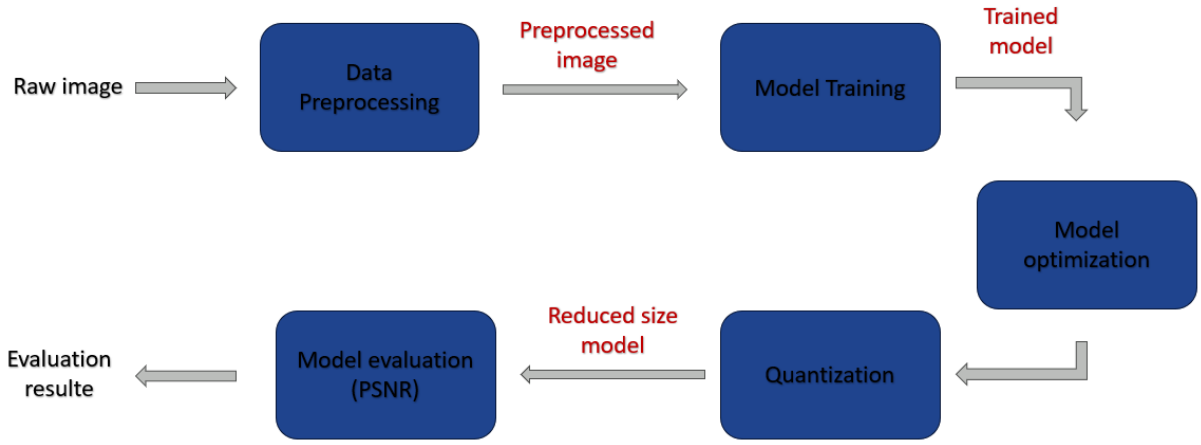

Figure 3.1: The overall workflow of our proposed image enhancement model.

Our first step is to determine which tensor operators are compatible and efficient for deployment on portable devices. Our baseline operation node has a 3x3 convolution layer and 16 output channels. and one multichannel (elementwise) tensor operation, which was crucial to achieving the desired image enhancement result.

Table 3.1: Meta-nodes inference time (ms) on Synaptics Dolphin Platform(NPU) [15].

Main type	Meta-node	Time (ms)
Tensor operator nodes	Channel split	9.8
	Channel concat	10.4
	Add two tensors	5.2
	Multiply two tensors	9.6
	Global max pooling	20.0
	Global average plooling	13.1
Convolution nodes	3 x 3 Convolution	4.3
	1 x 1 Convolution	2.9
Activation nodes	ReLU	1.3
	Leaky ReLU	3.6
Resize nodes	Nearest neighbor	57.6
	Bilinear	75.4

To guide our architectural choices, we analyze the inference time for various tensor operators on the Synaptics Dolphin Platform Neural Processing Unit (NPU), proposed by the authors of abpn [15]. As we can see in the table 3.1, the runtime of channel splitting and concatenation on NPUs are much higher compared to summation (element

wise), we prioritize the element wise operations to ensure the real-time performance on portable devices. iIn the selection of activation functions, we chose ReLU over Leaky ReLU due to its significantly faster execution speed (approximately 1.2 times faster) the marginal performance gain by Leaky ReLU is quite small (within 0.03dB). For the choice of convolution layers, we prefer to use 3x3 convolution ahead of 1x1 convolution even though it's faster but the convincing to achieve a good performance isn't good, so we favored 3x3 convolutions for their superior performance and efficiency in image enhancement tasks. we adopt to all convolution layers a kernel size of 3x3 multi-layer structures, we keep all layers with the same number of channels without changing them to ensures the consistency of the network.

In summary, our network is designed with only 3x3 convolution layers and ReLU activation as its main component. A residual learning algorithm for RGB images is integrated into our proposed network, inspired by ABPN [15]. Figure 3.2 illustrates the proposed network.

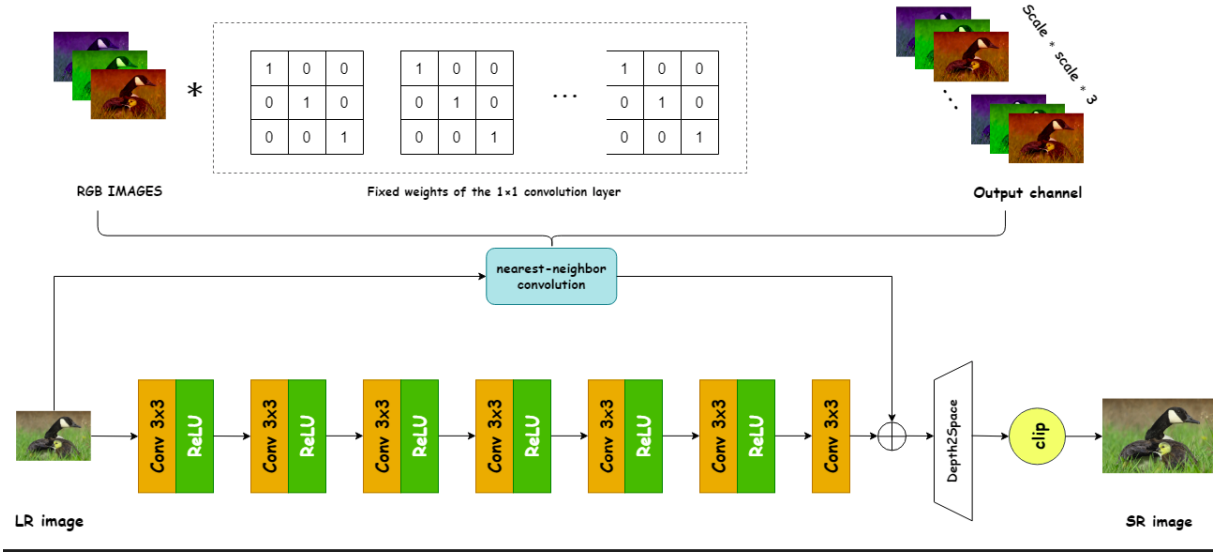

Figure 3.2: The network architecture of our proposed model. formed with only 7 efficient layers (to accelerate the inference time) of 3x3 conv and 1x1 conv in the nearest-neighbour convolution to fed the final result with the low-frequency information [40].

1. **Nearest-neighbour Convolution**

   The nearest-neighbour convolution module is the most critical component of this network, which plays a crucial role in achieving efficient image resolution enhancement.

   We make a slight difference from traditional Nearest-neighbour convolution. To achieve nearest interpolation, we adopt a distinctive approach, freezing the convo-

lution layer weights and initializing them manually using s^2 groups of 3x3 identity matrices, where 's' represents the upscale factor.

$$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}...\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} ===> s^2\ of\ 3x3\ matrix$$

As each group generates an RGB image, it replecate the input image s^2 times, just as a copy operation. Subsequently, through the depth-2-space operation, these s^2 RGB images can then be reconstructed into a HR image.

Thus, the reconstructed image will be identical to the nearest interpolated HR image. However, it is much faster since it is based on fixed weights, especially on limited GPUs/NPUs. As can be seen in the table below 3.2, the proposed nearest convolution can save approximately 40 ms in comparison with the original nearest upsampling.

Table 3.2: the difference runtime of commonly used upsample methods between the new nearest convolution and traditional nearest and bilinear on CPU and GPU mobile accelerator (those performance achieved on a OnePlus 9 Pro smartphone)

Upsamling method	CPU	GPU Delegate	PSNR
bilinear	33.1ms	9ms	**27.67dB**
nearest	9.7ms	**8ms**	26.67dB
new nearest+ depth2space	**6.7ms**	8.5ms	26.63db

The implementation of the nearest convolution:

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, ReLU,
    Concatenate,Lambda, Layer, Add
    out_c = 3 # outuput channle
    scale_factor = 4

    # Define the input tensor for the Nearest Convolution
    x_in = Input(shape=(None, None, 3))

    # Create a weight matrix 'w' for new nearest convolution
    # This matrix consists of a 3x3 identity matrix repeated
    #'scale_factor^2' times
    w = np.transpose(np.array([[
```

```
    [[1, 0, 0],
     [0, 1, 0],
     [0, 0, 1]]*scale_factor**2
    ]]), (0, 1, 3, 2))

    res_conv = Conv2D(out_c*(scale_factor**2), kernel_size=1,
    padding='same', use_bias=False)

    # the Conv layer are frozen here we set trainable to false
    res_conv.trainable = False
    res_conv.build(x_in.shape)
    res_conv.set_weights([w])

    # by applying the Nearest Convolution to
    # the input tensor 'x_in'
    x_res = res_conv(x_in)
```

List of Source codes 3.1: The code snippet demonstrates the implementation of the Nearest Convolution

2. **Residual Learning**
   ABPN [15] inspired us to incorporates residual learning from RGB images to enhance the final result.

   In our case, we added the output channels obtained from the new nearest convolution ($s^2 \times 3$ channels in total, if the scale factor is 4, you'll get 48 feature maps (channels)) to the layer output of the plain network immediately before the depth-2-space layer. By doing that, we allow the plain network to focus on learning residual information. And since we are trying to shorten the inference time we incorporate our network with only 7 layers of 3 x 3 convolution with the ReLU activation function, and we fix the channels to 32, see the code in 3.2.

3.1.2 The Network Function

Assuming that $\mathbf{x}$ is the HR image and $\mathbf{y}$ is the degraded LR image, we could obtain the super-resolved image $\hat{\mathbf{x}}$ as follows:

$$\hat{\mathbf{x}} = \text{clip}\left(D2S\left(f_{\text{res}}(\mathbf{y};\theta) + f_{nc}(\mathbf{y})\right), 0, 255\right) \tag{3.1}$$

Where:

- $f(.)$ denote to the super resolution SR network with θ is the paramater.

- $f_{\mathbf{res}}$ represents the residual learning.
- $f_{\mathbf{nc}}$ represents the nearest neighbour.
- $D2S$ represnt the depth to space orerator.
- **clipped** function to ensure the range of image pixel outputed from the $D2S$ function falls within the range from 0 to 255.

Therefore, the final output High resolution image $\hat{\mathbf{x}}$ is the clipped result of $D2S\left(f(\cdot) + f_{nc}(\mathbf{y})\right)$

The implementation of the network architecture:

```
    n_feat=32, out_c=3, scale_factor=4

    # Define a series of 6 convolutional layers for feature extraction
    # followed with activation ReLU
    x = Conv2D(n_feat, kernel_size=3, padding='same', activation='relu', kernel_initializer=glorot_normal(), bias_initializer='zeros')(x_in)
    x = Conv2D(n_feat, kernel_size=3, padding='same', activation='relu', kernel_initializer=glorot_normal(), bias_initializer='zeros')(x)
    x = Conv2D(n_feat, kernel_size=3, padding='same', activation='relu', kernel_initializer=glorot_normal(), bias_initializer='zeros')(x)
    x = Conv2D(n_feat, kernel_size=3, padding='same', activation='relu', kernel_initializer=glorot_normal(), bias_initializer='zeros')(x)
    x = Conv2D(n_feat, kernel_size=3, padding='same', activation='relu', kernel_initializer=glorot_normal(), bias_initializer='zeros')(x)
    x = Conv2D(out_c*(scale_factor**2), kernel_size=3, padding='same', activation='relu', kernel_initializer=glorot_normal(), bias_initializer='zeros')(x)

    # Define the last convolutional layer for transition to D2S without ReLU
    x = Conv2D(out_c*(scale_factor**2), kernel_size=3, padding='same', kernel_initializer=glorot_normal(), bias_initializer='zeros')(x)

    # Add the result of Nearest Convolution ('x_res')
    # to the feature extracted tensor ('x')
    x = Add()([x_res, x])
```

```
18
19      # Perform depth-to-space operation for upscaling
20      depth_to_space = Lambda(lambda x: tf.nn.depth_to_space(x,
    scale_factor))
21      x = depth_to_space(x)
22
23      # Clip the pixel values to be within the range [0, 255]
24      clip_func = Lambda(lambda x: tf.clip_by_value(x, 0., 255.))
25      x = clip_func(x)
26
27      return Model(x_in, x)
```

List of Source codes 3.2: The code snippet demonstrates the implementation of the plain network

This code show the implemented plain network (or the architect network), which serves as the foundation for our proposed model.
The network can be segmented into three main stages:

(1) **Feature Extraction layers**

- We define a series of 6 convolutional layers for feature extraction.
- Each layer uses a 3x3 kernel, and uses Glorot normal initialization for weights.
- The padding is set to 'same,' which means that the output feature maps have the same spatial dimensions as the input image
- These layers followed by ReLU to perform typical feature extraction operations.

(2) **Transition Layer to Depth-to-Space**

- A final convolutional layer follows, with the same kernel size and padding.
- However, this layer does not apply ReLU activation, its purpose is to transition to the depth-to-space operation.
- The output of this layer is added to the result of the nearest neighbor upsampling **x_res** using an **Add** layer.

(3) **Depth-to-Space Operation**

- A depth-to-space upsampling operation is performed using the **depth_to_space** Lambda layer, effectively increasing the spatial dimensions by a factor of **scale_factor**.

- The pixel values are clipped to the range [0, 255] using the **clip_func** Lambda layer, ensuring valid image values for the final output.

The final output of the entire architecture is a Keras model that takes **x_in** as input and produces the processed tensor **x**.

3.1.3 The Loss Function

as we know the $\mathcal{L}_2$ Mean Squared Error loss function (MSE) is sensitive to outliers and suffers from noisy data, we choose $\mathcal{L}_1$ as it is less sensitive and robust when dealing with noisy data. And The following formulation is used:

$$\mathcal{L}_1(\theta) = \frac{1}{N}\sum_{i=1}^{N}(f(\mathbf{y};\theta) - \mathbf{x}) \tag{3.2}$$

where:

- θ represents the model's parameters that are optimized during training.
- $f(\mathbf{y};\theta)$ represents the predictions made by our model when given input data $\mathbf{y}$ and parameterized by θ. This is essentially the output of our neural network.
- $\mathbf{x}$ represents the actual target values that our model is trying to predict.
- N is the total number of the dataset examples.
- $\sum_{i=1}^{N}$ denotes the summation over all the image in the dataset, where we calculate the absolute difference between the model's prediction and the ground truth for each image.
- $\frac{1}{N}$ is used to compute the average absolute difference by dividing the sum of absolute differences by the total number of images.

Table 3.3: The table shows the runtime on CPU and GPU delegate and the PSNR performance of the proposed network on OnePlus 9 pro smartphone using AI benchmark application [24]

OnePlus 9 pro	`#Parameters`	CPU	GPU Delegate	PSNR-int8	PSNR float32
our model	52K	0.185 s	0.065 s	29.57	29.68

3.2 Training Methodology

Since Python is efficient, robust, and powerful in computer vision and deep learning, we chose it to implement our super-resolution model. As a programming language, Python is known for its extensive library and tool ecosystems that make development and experimentation easy. We can conduct our experiments with the help of its powerful libraries tensorflow keras and open cv.

We utilize TensorFlow, an open-source machine learning framework, along with the Keras api, to build and train our super-resolution model. Keras, integrated within TensorFlow, offers a high-level interface for designing and configuring neural networks, streamlining the model-building process.

We also use the specialized framework from TensorFlow, TensorFlow Lite. It plays a crucial role in optimizing and converting our model for efficient execution on resource-constrained edge devices. This enables us to achieve real-time super-resolution capabilities on those limited devices. We also use OpenCV and Numpy for the implementation.

As the computer vision demand a lot of GPU due to the extensive calculation, we trained the model in google colab pro under the following resourses:

- L4 GPU 24 GB GDDR6 VRAM
- 48 GHz CPU

Data Preprocessing

For the dataset, we use the **DIV2K** in order to train the model, it is one of the most popular datasets used for image super-resolution tasks, proposed by Agustsson et al. [1] for NTIRE2017 and NTIRE2018 Super-Resolution Challenges. It contains 1000 HR images in total: 800 HR images for training, 100 HR images for validation, and 100 HR images for testing. Examples are shown below in Fig 3.4.

We download it from (https://data.vision.ee.ethz.ch/cvl/DIV2K/)

Preprocessing is an important step in preparing our dataset for training. It involves the following operations to ensure the input data is training-ready and aligns with the model's specifications:

Figure 3.3: Samples from **DIV2K**. (HR ground truth image, LR images downscaled by 2, 3, 4, and 8 factors, from top to bottom.)

1. **Downgrade:** the downgrade operation is a crucial step in super resolution applications, the image in real world may encounter varying quality due to factors such as noise, compression, or imperfect acquisition devices. Due to that, we offer the option to apply downgrades to the high-resolution images. We support various scaling factors (2x, 3x, 4x, 8x) with bicubic methods to create low-resolution counterparts of the high-resolution images.

```
class DIV2K:
    def __init__(self,
                 scale=2,
                 subset='train',
                 downgrade='bicubic',
                 images_dir='/data/dataset',
                 caches_dir='/data/dataset/'):

        _scales = [2, 3, 4, 8]

        if scale in _scales:
            self.scale = scale
        else:
            raise ValueError(f'scale must be in ${_scales}')

        if subset == 'train':
            self.image_ids = range(1, 801)
        elif subset == 'valid':
            self.image_ids = range(801, 901)
        else:
            raise ValueError("subset must be 'train' or 'valid'")

        _downgrades_a = ['bicubic', 'unknown']
        _downgrades_b = ['mild', 'difficult']

        if scale == 8 and downgrade != 'bicubic':
            raise ValueError(f'scale 8 only allowed for bicubic
    downgrade')

        if downgrade in _downgrades_b and scale != 4:
            raise ValueError(f'{downgrade} downgrade requires scale
     4')

        if downgrade == 'bicubic' and scale == 8:
            self.downgrade = 'x8'
        elif downgrade in _downgrades_b:
            self.downgrade = downgrade
        else:
            self.downgrade = downgrade

        self.subset = subset
        self.images_dir = images_dir
        self.caches_dir = caches_dir

```

```
43          os.makedirs(images_dir, exist_ok=True)
44          os.makedirs(caches_dir, exist_ok=True)
```

List of Source codes 3.3: The code snippet demonstrates the downgrad fuction

2. **Data Augmentation:** The use of data augmentation techniques enhances the diversity of our training data and improves the generalization of our models. These include random cropping, flipping, and rotation. With random cropping, we can extract random patches from the high-resolution images, while flipping and rotating provide additional information.

```
1  def random_crop(lr_img, hr_img, hr_crop_size=192, scale=2):
2      lr_crop_size = hr_crop_size // scale
3      lr_img_shape = tf.shape(lr_img)[:2]
4
5      lr_w = tf.random.uniform(shape=(), maxval=lr_img_shape[1] -
       lr_crop_size + 1, dtype=tf.int32)
6      lr_h = tf.random.uniform(shape=(), maxval=lr_img_shape[0] -
       lr_crop_size + 1, dtype=tf.int32)
7
8      hr_w = lr_w * scale
9      hr_h = lr_h * scale
10
11     lr_img_cropped = lr_img[lr_h:lr_h + lr_crop_size, lr_w:lr_w +
       lr_crop_size]
12     hr_img_cropped = hr_img[hr_h:hr_h + hr_crop_size, hr_w:hr_w +
       hr_crop_size]
13
14     return lr_img_cropped, hr_img_cropped
15
16
17 def random_flip(lr_img, hr_img):
18     rn = tf.random.uniform(shape=(), maxval=1)
19     return tf.cond(rn < 0.5,
20                    lambda: (lr_img, hr_img),
21                    lambda: (tf.image.flip_left_right(lr_img),
22                             tf.image.flip_left_right(hr_img)))
23
24
25 def random_rotate(lr_img, hr_img):
26     rn = tf.random.uniform(shape=(), maxval=4, dtype=tf.int32)
27     return tf.image.rot90(lr_img, rn), tf.image.rot90(hr_img, rn)
```

List of Source codes 3.4: The code snippet demonstrates the random and flip and rotate function

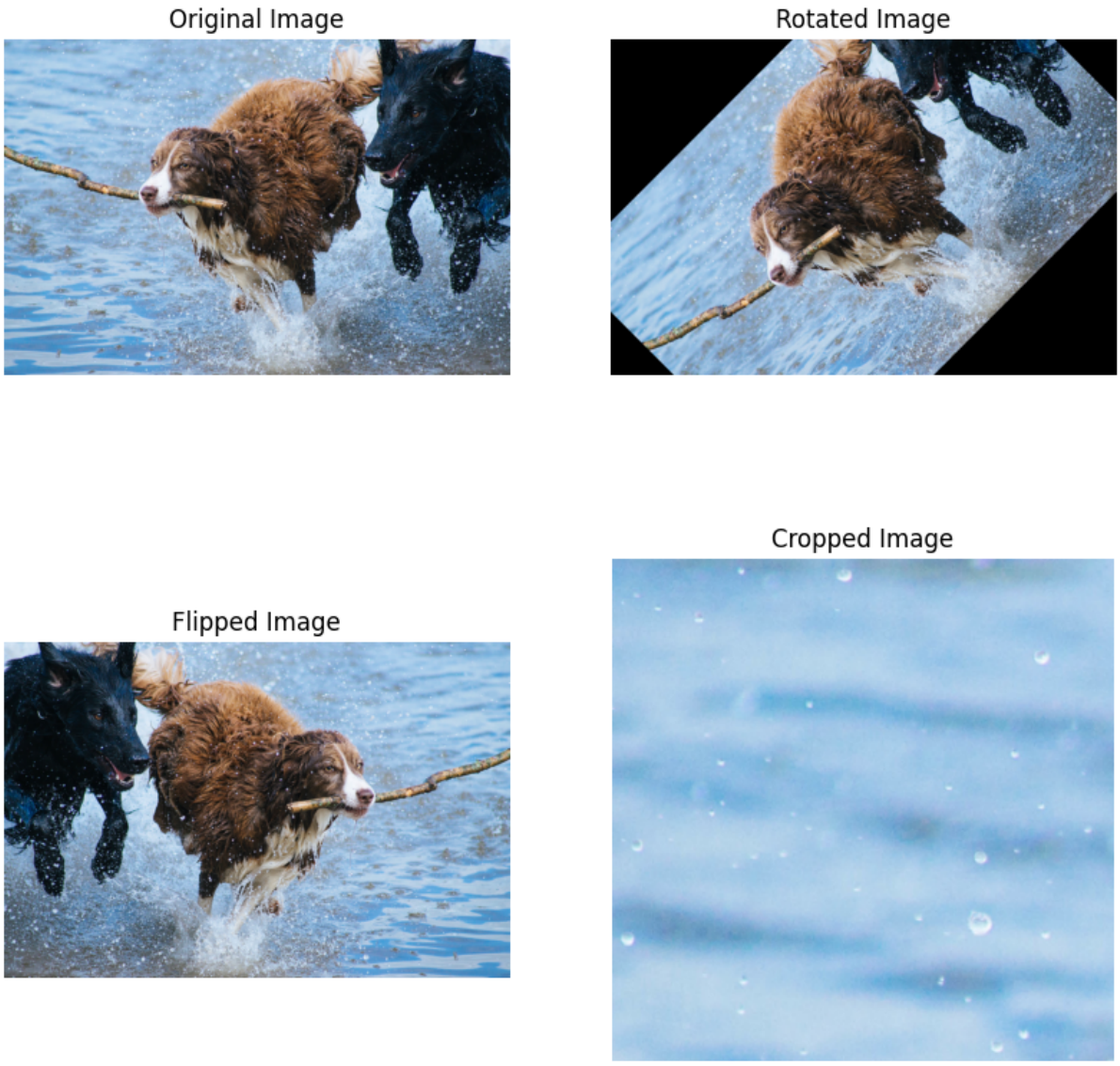

Figure 3.4: the result of code 3.4 on an image from div2k.

As shown in the figure 3.4, the data augmentation step include image cropping, flipping, and rotation, which help augment the training dataset.

3. **Caching**: another technique we use is the caching. Caching involves storing the preprocessed images in a cache directory. This step significantly reduces data loading time during subsequent training iterations.

3.3 Evaluation Metric

To present our experimental results, we implemented the function (as shown in the code snippet in 3.5) to calculate the PSNR.

PSNR: The Peak Signa-to-Noise Ratio (PSNR) is the most popular metric in the image super resolution tasks, calculated as follow 3.3:

$$\text{PSNR}(img_1, img_2) = 20 \cdot \log_{10}\left(\frac{255.0}{\sqrt{\text{MSE}}}\right) \tag{3.3}$$

where, **255.0**: This is the maximum possible pixel value.
MSE refer to the Mean Square Error, which measures the average difference between two images img1 and img2, High PSNR values correspond to high similarity between two images, while low PSNR values mean low similarity.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(I_1(i) - I_2(i))^2 \tag{3.4}$$

```
def calculate_psnr(img1, img2):
    # img1 and img2 have range [0, 255]
    img1 = img1.astype(np.float64)
    img2 = img2.astype(np.float64)
    mse = np.mean((img1 - img2)**2)
    if mse == 0:
        return float('inf')
    return 20 * math.log10(255.0 / math.sqrt(mse))
```

List of Source codes 3.5: The code snippet show the PSNR function

3.4 Optimization Strategies

In the optimization strategy, we mainly focus on preparing the super-resolution model for efficient deployment on edge devices using TensorFlow Lite (TFLite) quantization tool. which is basically a Post-Quantization strategy. But Before we talk about anything else we need to discuss what is Quantization (QAT)?.

- **What is Quantization?**
  Basically the quantization [26] is the process of reducing the precision of the weights, biases, and activation's function.

In simple word, is the process of converting a neural network, which typically uses 32-bit floating numbers for parameter representation, into a smaller representation, like 8-bit integers (our example). However, floating numbers have high precision, but they can be very memory intensive and computationally expensive which is not ideal for the real time image super resolution.

Going from 32-bit to 8-bit for example, and performing some or all of the operations on 8-bit integers, would reduce the model size and memory requirements by a factor of 4 (fig 3.5), which is really impressive for constrained devices resources.

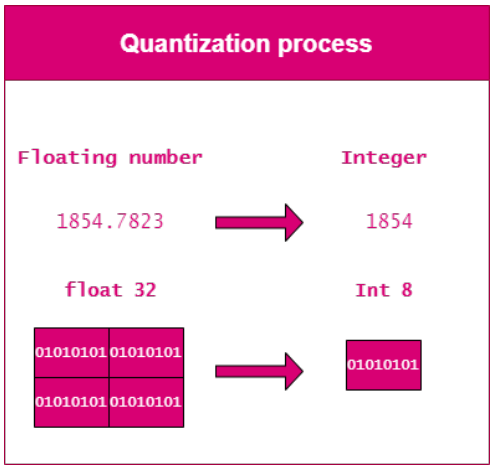

Figure 3.5: Quantization decreasing the precision of float-32 to INT-8 can reduce the model size by a factor of 4.)

However, Performing quantization to go from 32-bit to 8-bit is tricky. Only 256 values can be represented in int8, while float32 can represent a wide range of values. The idea used from TFLite quantization that we use in our model is to find the best way to project range [a, b] of float32 values to the int8 space, so values outside of the [a, b] range are clipped to the closest representable value.

The importance of quantization in training the image super-resolution model cannot be overstated. Due to the limited resources in our hand when we talking about portable devices, even though, as we see before in the table 3.3loss in the quality of the super image isn't big just (0.1dB), and with this negligible sacrificing we are gaining a lot of memory consummation led us to approximately 4 time smaller model size.

The code below 3.6 represent the function we use to generate the TF lite version of the model by using 'tf.lite.TFLiteConverter' and it's built in function. note that for evaluation perpose of the qauntizaied model (float 32 and INT 8) in PSNR metric, we set the input shape of the image to [1, None, None, 3] to be able to Handel arbitrary shape

images. and for evaluation of inference time in mobile, we set the input shape to [1, 360, 640, 3]. and we test the model in the AI Benchmark application proposed by [24] on a OnePlus 9 pro smartphone.

```
def generate_tflite(name, model_path, folder):
    # Define input shape and representative dataset based on 'name'
    if name == 'model':
        # Input shape for the TFLite model
        input_shape = [1, 360, 640, 3]
        # Representative dataset for quantization
        rep_data = representative_dataset_gen
    else:
        # Input shape with dynamic dimensions
        input_shape = [1, None, None, 3]
        # Representative dataset for quantization
        rep_data = representative_dataset_gen_none

    # Load the TensorFlow saved model
    model = tf.saved_model.load(model_path)
    # Create a concrete function from the saved model
    concrete_func = model.signatures[tf.saved_model.
   DEFAULT_SERVING_SIGNATURE_DEF_KEY]
    # Set the input shape of the concrete function
    concrete_func.inputs[0].set_shape(input_shape)
    # Create a TFLite converter from the concrete function
    converter = tf.lite.TFLiteConverter.from_concrete_functions([
   concrete_func])
    # Enable experimental features for the converter
    converter.experimental_new_converter = True
    converter.experimental_new_quantizer = True
    # Apply default optimizations to the converter
    converter.optimizations = [tf.lite.Optimize.DEFAULT]
    # Set the representative dataset for quantization
    converter.representative_dataset = rep_data

    # Configure quantization options if 'name' is 'model' or 'model_none
   '
    if name == 'model' or name == 'model_none':
        print("====== uint8 quantification ======")
        converter.target_spec.supported_ops = [tf.lite.OpsSet.
   TFLITE_BUILTINS_INT8]
        converter.inference_input_type = tf.uint8
        converter.inference_output_type = tf.uint8
```

```
36
37      # Convert the model to TFLite format
38      tflite_model = converter.convert()
39      # Define the path for the TFLite model and write it to disk
40      tflite_model_path = f"{output_folder}/{name}.tflite"
41      open(tflite_model_path, "wb").write(tflite_model)
```

List of Source codes 3.6: The code snippet demonstrat how we use tensorFlow lite to generate the quantized model in float 32 and INT 8

Chapter 4

Experimental Results and Discussion

In this chapter, we present the experimental results. We will discuss the settings and results of the experiments conducted on four benchmarks and and their efficiency on a typical hardware, to provide evidence of the efficacy of the proposed approach.

4.1 Configuration Details

We used the DIV2K dataset for our image super-resolution experiment, consisting of 800 training images and 100 testing images. Using a batch size of 64, we trained our model iteratively, increasing image resolution by a factor of 4. To train with LR images, we fixed the size of patches to 64x64. We set the number to 500,000 iterations. Using the Adam optimizer, and we started with an initial learning rate of 10^{-3} as we see in code 4.1, which was halved every 200,000 iterations. AT the end, we try to fine-tune the model by adding 200,000 iterations performed by LR patches of (128x128). For fine-tuning, all the experiments were conducted on a single L4 GPU from Google Colab Pro, to ensure the consistent performance of our original model without quantize model throughout the training and evaluation phases.

```
1 def __init__(self, model, checkpoint_dir,
2                learning_rate=PiecewiseConstantDecay(boundaries=[50000,
    150000, 300000, 400000], values=[1e-3, 5e-4, 2e-4, 1e-4, 5e-5])):
3     super().__init__(model, loss=MeanAbsoluteError(),
    learning_rate=learning_rate, checkpoint_dir=checkpoint_dir)
4
5 def train(self, train_dataset, valid_dataset, steps=300000,
    evaluate_every=1000, save_best_only=True):
6     super().train(train_dataset, valid_dataset, steps, evaluate_every,
    save_best_only)
```

List of Source codes 4.1: The code snippet show how we configuer the learning rate

4.2 Model Quantization

As we mentioned before, we use the standard TensorFlow Quantization tool TFLite to quantize the trained model as the Post-Quantization strategy. To evaluate the PSNR of the quantized models (float32 and int8), we set the input shape to [1, None, None, 3].

To evaluate the inference time, we fix the input shape to [1, 360, 640, 3] and test the model in the AI Benchmark [24] application on OnePLUS 9 pro smartphone.

Here is some qualitative examples of our model compared with some lightweight SR model:

Figure 4.1: Visual comparison of two img from the DIV2K validation dataset. the result by INT8 quantization model, and the scale factor 4.

4.3 Results Benchmark

Table 4.1: A performance comparison of various SR lightweight models is conducted across four benchmarks on X4 scale. PSNR the Y channel are reported for each dataset. The metrics # Params, # FLOPs, # Acts, and # Conv denote the total number of network parameters, floating-point operations, activation, and convolution layers, respectively. FLOPs and Acts are measured when generating an SR image of 256x256 resolution. Results for our model and the best-performing candidate are highlighted in red and blue, respectively.

Model	#param	#FLOPs(G)	#Acts(M)	#conv	Set5 (dB)	Set14 (dB)	Urban100 (dB)	DIV2K (dB)
Bicubic	-	-	-	-	28.43	26	23.14	28.10
SRCNN [12]	57K	52.7	89.39	3	30.48	27.49	24.52	29.25
FSRCNN [13]	12K	5	10.81	8	30.70	27.59	24.60	29.36
LapSRN [29]	813K	149.40	264.04	27	31.54	28.19	25.21	29.88
EDSR-C32 [35]	241.80K	14.15	50.69	13	31.46	28.07	25.21	29.87
Our model	52K	25.1	11.05	7	30.72	27.88	25.11	29.68

We evaluate our proposed model against well-established single image super-resolution (SR) lightweight models on tasks involving x4 upscaling. These established models include SRCNN [12], FSRCNN [13], LapSRN [29], and EDSR [35].

Since the standard version of EDSR is optimized for powerful GPU servers and has a very complex architecture, we choose a more streamlined version, EDSR-C32. EDSR-C32 consists of five residual blocks, each containing convolutional layers with 32 channels.

Table 4.1 summarizes the performance comparison of various SR models on four benchmark datasets. We've included not only the PSNR index but also the number of parameters, FLOPs (Floating-point Operations), activation layers, and convolution layers for a more in-depth comparison. These values are calculated for the task of upscaling an image to 256x256 resolution at x4 task. Recent research suggests that the number of activations is a more reliable indicator of model efficiency compared to parameters and FLOPs [138, 99].

Key takeaways from Table 4.1: Our proposed model significantly outperforms both SRCNN and FSRCNN on all four benchmarks. This is achieved while using fewer resources then SRCNN approximately 2x fewer FLOPs, and 9x fewer activations. And also demonstrates that our model can achieve superior performance to bicubic upsampling while requiring a bit computational power and memory.

Secondly, our model show some advantages over the other two SR model LapSRN and EDSRC32 especially when we compare on term of complexity as we see, we use roughly 18x/ 9x fewer parameters, and 26x/ 5x fewer in activation layers and 3.5x / 2x fewer convolutional layers, respectively, and yet can achieve some comparable results to them,

and from our point of view, a small loss in quality of images while gaining a significant margin of time and complexity, especially since it is designed to use on weak devices such as mobile and edge devices, can be considered somewhat worth it.

4.4 Mobile Benchmarking

While factors like model size, parameters, FLOPs, and activations provide insights into model complexity, they don't directly translate to real-world performance on mobile devices. To address this, we conducted real-time speed evaluations on mobile platforms. We used a OnePlus 9 Pro with a Snapdragon 865 processor (representing a high-end phone) and ensured a consistent software environment with the same SDK and settings for all models. The AI Benchmark App by ignatov [24] served as the execution platform, see figure **??**, while TFLITE GPU and Hexagon NN, common inference engine delegates, were employed. To further optimize for mobile performance, all models were quantized for 8-bit arithmetic. The task involved upscaling images to 1080p resolution with a 4x factor. The results, presented in Table 4.2, reveal that most evaluated models struggle to achieve real-time processing speed. However, our proposed model stands out close to FSRCNN for its efficiency, to reaching real-time performances.

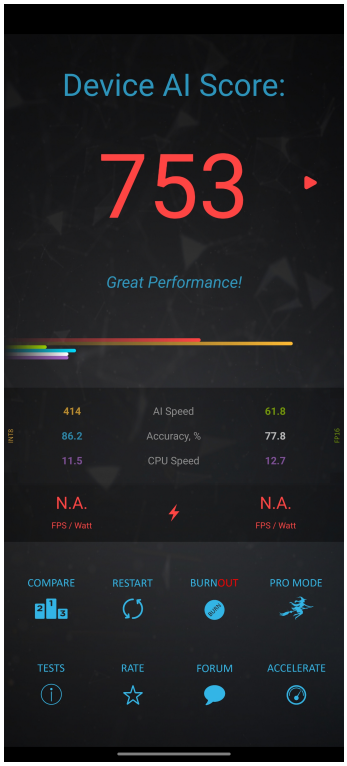

Figure 4.2: Samples from **DIV2K**. (the performance of the OnePlus 9 pro smartphone with (865 snapdragon CPU) based on the AI app tested on a large collection of the state of the art computer vision models.)

Table 4.2: Compares the processing speed of SR models on upscaling image to 1920x1080 using Oneplus 9 pro mobile devices with SNAPDRAGON 865. and achieving the performance of (>15 fps)

Model	OnePlus 9 pro (SD 865)
SRCNN [12]	1.583 s
FSRCNN [13]	0.032 s
LapSRN [29]	5.378 s
EDSR-C32 [35]	0.101 s
OURS	0.065 s

Among the compared models, our model is the second after FSRCNN the only one that achieves real-time performance. This is likely due to its very neat architecture. While SRCNN also utilizes a simple design, its pre-upsampling approach significantly increases computational and memory demands, leading to slower execution. Even the lightweight versions, EDSR-C32 and LapSRN, fall short of real-time processing on both devices. This

can be attributed to their complex structures with dense connections and multi-branch pathways, ultimately limiting their speed.

Conclusion

Super-resolution (SR) design based on deep neural networks (DNNs) has many real-world applications, such as image and video restoration, transition, and display. Recent advancements in DNN have also enabled remarkable progress in SR tasks. Still, it remains to be seen whether to design SR networks that are both lightweight and efficient and can be used on commodity devices, such as GPU servers and mobile devices.

In this thesis, we have attempted to design an effective and efficient SR network optimized for mobile and edge devices. By comparing the state-of-the-art SR lightweight methods with our designs, we achieve a good PSNR comparable to those of conventional designs, while using significantly less hardware resources in order to satisfy real-world needs.

In the future, we will try to explore further how deep learning and resources limited hardware can be combined. Lightweight and efficient-based SR will be the next generation of designing efficient SR for mobile devices.

Bibliography

[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] Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. Fast, accurate, and lightweight super-resolution with cascading residual network, 2018.

[3] Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization, 2018.

[4] Mustafa Ayazoglu. Extremely lightweight quantization robust real-time single-image super resolution for mobile devices, 2021.

[5] S. Baker and T. Kanade. Limits on super-resolution and how to break them. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9):1167–1183, 2002.

[6] Xiangxiang Chu, Bo Zhang, Hailong Ma, Ruijun Xu, and Qingyuan Li. Fast, accurate and lightweight super-resolution with neural architecture search, 2020.

[7] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Hailong Ma. Multi-objective reinforced evolution in mobile neural architecture search, 2019.

[8] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations, 2016.

[9] Tao Dai, Jianrui Cai, Yongbing Zhang, Shu-Tao Xia, and Lei Zhang. Second-order attention network for single image super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[10] Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks, 2019.

[11] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again, 2021.

[12] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks, 2015.

[13] Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the super-resolution convolutional neural network, 2016.

[14] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[15] Zongcai Du, Jie Liu, Jie Tang, and Gangshan Wu. Anchor-based plain net for mobile image super-resolution, 2021.

[16] Daniel Glasner, Shai Bagon, and Michal Irani. Super-resolution from a single image. In *2009 IEEE 12th International Conference on Computer Vision*, pages 349–356, 2009.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[18] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[20] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[21] Zheng Hui, Xinbo Gao, Yunchu Yang, and Xiumei Wang. Lightweight image super-resolution with information multi-distillation network. In *Proceedings of the 27th ACM International Conference on Multimedia*. ACM, oct 2019.

[22] Zheng Hui, Xiumei Wang, and Xinbo Gao. Fast and accurate single image super-resolution via information distillation network, 2018.

[23] Judith Hurwitz. Machine Learning For Dummies®, IBM Limited Edition. 2018.

[24] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. Ai benchmark: Running deep neural networks on android smartphones, 2018.

[25] Andrey Ignatov, Radu Timofte, Maurizio Denna, Abdel Younes, Andrew Lek, Mustafa Ayazoglu, Jie Liu, Zongcai Du, Jiaming Guo, Xueyi Zhou, Hao Jia, Youliang Yan, Zexin Zhang, Yixin Chen, Yunbo Peng, Yue Lin, Xindong Zhang, Hui Zeng, Kun Zeng, Peirong Li, Zhihuang Liu, Shiqi Xue, and Shengpeng Wang. Real-time quantized image super-resolution on mobile npus, mobile ai 2021 challenge: Report, 2021.

[26] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017.

[27] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks, 2016.

[28] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Deeply-recursive convolutional network for image super-resolution, 2016.

[29] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Deep laplacian pyramid networks for fast and accurate super-resolution, 2017.

[30] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[31] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2017.

[32] Huixia Li, Chenqian Yan, Shaohui Lin, Xiawu Zheng, Yuchao Li, Baochang Zhang, Fan Yang, and Rongrong Ji. Pams: Quantized super-resolution via parameterized max scale, 2020.

[33] Yawei Li, Shuhang Gu, Kai Zhang, Luc Van Gool, and Radu Timofte. Dhp: Differentiable meta pruning via hypernetworks, 2020.

[34] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer, 2021.

[35] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution, 2017.

[36] Ding Liu, Bihan Wen, Yuchen Fan, Chen Change Loy, and Thomas S. Huang. Non-local recurrent network for image restoration, 2018.

[37] Jie Liu, Jie Tang, and Gangshan Wu. Residual feature distillation network for lightweight image super-resolution, 2020.

[38] Yun Liu, Yu-Huan Wu, Guolei Sun, Le Zhang, Ajad Chhatkuli, and Luc Van Gool. Vision transformers with hierarchical attention, 2022.

[39] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021.

[40] Ziwei Luo, Youwei Li, Lei Yu, Qi Wu, Zhihong Wen, Haoqiang Fan, and Shuaicheng Liu. Fast nearest convolution for real-time efficient image super-resolution, 2022.

[41] Yinglan Ma, Hongyu Xiong, Zhe Hu, and Lizhuang Ma. Efficient super resolution using binarized neural network, 2018.

[42] Yiqun Mei, Yuchen Fan, and Yuqian Zhou. Image super-resolution with non-local sparse attention. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3516–3525, 2021.

[43] Ying Nie, Kai Han, Zhenhua Liu, Chuanjian Liu, and Yunhe Wang. Ghostsr: Learning ghost features for efficient image super-resolution, 2022.

[44] Ben Niu, Weilei Wen, Wenqi Ren, Xiangde Zhang, Lianping Yang, Shuzhen Wang, Kaihao Zhang, Xiaochun Cao, and Haifeng Shen. Single image super-resolution via a holistic attention network, 2020.

[45] S. S. Panda, M. S. R. S Prasad, and G. Jena. Pocs based super-resolution image reconstruction using an adaptive regularization parameter, 2011.

[46] Seong-Jin Park, Hyeongseok Son, Sunghyun Cho, Ki-Sang Hong, and Seungyong Lee. Srfeat: Single image super-resolution with feature discrimination. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[47] Ankit Prajapati, Sapan Naik, and Sheetal Mehta. Evaluation of different image interpolation algorithms. *International Journal of Computer Applications*, 58:6–12, 2012.

[48] Bin Qian, Jie Su, Zhenyu Wen, Devki Nandan Jha, Yinhao Li, Yu Guan, Deepak Puthal, Philip James, Renyu Yang, Albert Y. Zomaya, Omer Rana, Lizhe Wang, Maciej Koutny, and Rajiv Ranjan. Orchestrating the development lifecycle of machine learning-based iot applications: A taxonomy and survey, 2020.

[49] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[50] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jonathon Shlens. Stand-alone self-attention in vision models, 2019.

[51] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network, 2016.

[52] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[53] Dehua Song, Yunhe Wang, Hanting Chen, Chang Xu, Chunjing Xu, and Dacheng Tao. Addersr: Towards energy efficient image super-resolution, 2021.

[54] Ying Tai, Jian Yang, and Xiaoming Liu. Image super-resolution via deep recursive residual network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2790–2798, 2017.

[55] Ying Tai, Jian Yang, Xiaoming Liu, and Chunyan Xu. Memnet: A persistent memory network for image restoration, 2017.

[56] Radu Timofte, Vincent De Smet, and Luc Van Gool. A+: Adjusted anchored neighborhood regression for fast super-resolution. volume 9006, pages 111–126, 04 2015.

[57] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers and distillation through attention, 2021.

[58] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[59] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, and Xiaoou Tang. Esrgan: Enhanced super-resolution generative adversarial networks, 2018.

[60] Zhihao Wang, Jian Chen, and Steven C. H. Hoi. Deep learning for image super-resolution: A survey, 2020.

[61] Zhe Wei and Kai-Kuang Ma. Contrast-guided image interpolation. *IEEE Transactions on Image Processing*, 22(11):4271–4285, 2013.

[62] Wenming Yang, Xuechen Zhang, Yapeng Tian, Wei Wang, Jing-Hao Xue, and Qingmin Liao. Deep learning for single image super-resolution: A brief review. *IEEE Transactions on Multimedia*, 21(12):3106–3121, dec 2019.

[63] Linwei Yue, Huanfeng Shen, Jie Li, Qiangqiang Yuan, Hongyan Zhang, and Liangpei Zhang. Image super-resolution: The techniques, applications, and future. *Signal Process.*, 128:389–408, 2016.

[64] Sergey Zagoruyko and Nikos Komodakis. Diracnets: Training very deep neural networks without skip-connections, 2018.

[65] Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. Restormer: Efficient transformer for high-resolution image restoration, 2022.

[66] Kai Zhang, Wangmeng Zuo, and Lei Zhang. Learning a single convolutional super-resolution network for multiple degradations, 2018.

[67] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. Image super-resolution using very deep residual channel attention networks, 2018.

[68] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution, 2018.

[69] Fangyuan Zhu. A review of deep learning based image super-resolution techniques, 2022.

Appendix A

Deposit Permission

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur Et de La Recherche Scientifique

Faculté des Sciences et de la Technologie

Département de Mathématiques et d'Informatique

جامعة غرداية

Université de Ghardaïa

كلية العلوم و التكنولوجيا

قسم الرياضيات و الاعلام الالي

Ghardaïa le 14/06/2025

Rapport de correction de mémoire de Master SIEC

Je soussigné M~~/Mme/Mlle~~ : **Slimane Oulad-Naoui**

Président du jury du mémoire de Master intitulé :

A DL model for image resolution enhancement and optimization for edge devices

Après les corrections apportées au rapport, je déclare que les étudiants :

Yahia DOUDOU & Bakir Saber CHEKHAR

Sont autorisés à déposer leur manuscrit au niveau du département.

Fait et délivré pour servir et valoir ce que de droit

Signature

Slimane Oulad-Naoui