UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

JOÃO VÍTOR RAFAEL CHRISÓSTOMO

ACELERANDO REDES NEURAIS CONVOLUCIONAIS PARA SUPER-RESOLUÇÃO DE VÍDEO EM GPUS INTEGRADAS

ACCELERATING CONVOLUTIONAL NEURAL NETWORKS FOR VIDEO SUPER-RESOLUTION ON INTEGRATED GPUS

Campinas
2024

JOÃO VÍTOR RAFAEL CHRISÓSTOMO

ACELERANDO REDES NEURAIS CONVOLUCIONAIS PARA SUPER-RESOLUÇÃO DE VÍDEO EM GPUS INTEGRADAS

ACCELERATING CONVOLUTIONAL NEURAL NETWORKS FOR VIDEO SUPER-RESOLUTION ON INTEGRATED GPUS

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica, na área de Engenharia de Computação.

Thesis presented to the Faculdade de Engenharia Elétrica e de Computação of the Universidade Estadual de Campinas in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering, in the area of Computer Engineering.

Supervisor/Orientador: Prof. Dr. GILMAR BARRETO
Co-supervisor/Coorientador: Prof. Dr. PAULO VICTOR DE OLIVEIRA MIGUEL

Este trabalho corresponde à versão final da Dissertação defendida por JOÃO VÍTOR RAFAEL CHRISÓSTOMO e orientada pelo Prof. Dr. GILMAR BARRETO.

Campinas
2024

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

C461a

Chrisóstomo, João Vítor Rafael, 1996-
Acelerando redes neurais convolucionais para super-resolução de vídeo em GPUs integradas / João Vítor Rafael Chrisóstomo. – Campinas, SP : [s.n.], 2024.

Orientador: Gilmar Barreto.
Coorientador: Paulo Victor de Oliveira Miguel.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Aprendizado de máquina. 2. Processamento de imagens. 3. Processamento de sinal de vídeo. 4. Redes neurais convolucionais. I. Barreto, Gilmar, 1958-. II. Miguel, Paulo Victor de Oliveira, 1960-. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Informações Complementares

Título em outro idioma: Accelerating convolutional neural networks for video super-resolution on integrated GPUs
Palavras-chave em inglês:
Machine learning
Image processing
Video signal processing
Convolutional neural networks
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora:
Gilmar Barreto [Orientador]
Clenio Figueiredo Salviano
Leandro Tiago Manêra
Data de defesa: 29-05-2024
Programa de Pós-Graduação: Engenharia Elétrica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0009-0004-1347-2637
- Currículo Lattes do autor: http://lattes.cnpq.br/9610922536989007

COMISSÃO JULGADORA - DISSERTAÇÃO DE MESTRADO

Candidato: João Vítor Rafael Chrisóstomo RA: 264450

Data da Defesa: 29 de Maio de 2024

Título da Dissertação: "Acelerando redes neurais convolucionais para super-resolução de vídeo em GPUs integradas".

Prof. Dr. Gilmar Barreto (Presidente)
Prof. Dr. Clenio Figueiredo Salviano
Prof. Dr. Leandro Tiago Manêra

A ata de defesa, com as respectivas assinaturas dos membros da Comissão Julgadora, encontra-se no SIGA (Sistema de Fluxo de Dissertação/Tese) e na Secretaria de Pós-Graduação da Faculdade de Engenharia Elétrica e de Computação.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Resumo

Numa era dominada pelo consumo multimédia, a procura por conteúdos visuais de alta qualidade está aumentando. No entanto, há uma biblioteca cada vez maior de conteúdos produzidos no passado visando telas de resoluções mais baixas e fatores como a banda disponível limitam a resolução em que o conteúdo moderno pode ser transmitido.

As técnicas tradicionais de reamostragem geralmente não conseguem trazer o vídeo de baixa resolução aos padrões modernos, e a imagem acaba ficando borrada ou desprovida de detalhes.

Este trabalho aborda esse desafio aproveitando os avanços recentes em aprendizado de máquina, especificamente Redes Neurais Convolucionais (CNNs), para super-resolução de vídeo. A arquitetura foi meticulosamente ajustada para encontrar um equilíbrio entre qualidade de reconstrução e eficiência computacional, visando inferência em tempo real em GPUs integradas de baixo consumo de energia.

As técnicas exploradas neste trabalho vêm de modelos estabelecidos como SRCNN, FSRCNN, ESPCN, VDSR, ESRGAN, RDN e RCAN. Estes modelos citados, em sua maioria, focaram principalmente no aumento da qualidade da reconstrução, resultando em modelos muito lentos para inferência em tempo real. Neste trabalho, as mesmas técnicas foram adaptadas para arquiteturas menores na tentativa de encontrar o modelo ideal para inferência em tempo real.

O modelo de partida, entitulado de EDSR baseline, alcançou um desempenho de 0.6 quadros por segundo para um fator de escala de 2x com entradas em 720p na GPU integrada Intel Iris Xe LP. O modelo final, após as várias simplificações, aumentou esta métrica para 28 quadros por segundo.

O treinamento perceptual e adversário também se mostraram viáveis na rede final, tornando possível que o modelo alucine detalhes realistas. Finalmente, o mesmo modelo também foi treinado com imagens com compressão JPEG, forçando-o a aprender como limpar os artefatos de compressão.

Ao explorar os limites das técnicas de super-resolução em hardware simples, este trabalho abre caminho para experiências multimídia de alta qualidade e em tempo real para uma ampla variedade de dispositivos.

Abstract

In an age dominated by multimedia consumption, demand for high-quality visual content is rising. However, there's an ever-growing library of content produced in the past targeting lower resolution displays, and factors like the available bandwidth limit the resolution in which modern content can be served.

Traditional resampling techniques are usually unable to bring low resolution video up to modern standards, and the image ends up looking blurry or devoid of fine-detail.

This work addresses this challenge by leveraging recent advancements in machine learning, specifically Convolutional Neural Networks (CNNs), for video super resolution. The architecture was meticulously fine-tuned to strike a balance between reconstruction quality and computational efficiency, aiming for real time inference on low-power consumer-grade integrated GPUs.

The techniques explored in this work come from established models such as SRCNN, FSRCNN, ESPCN, VDSR, ESRGAN, RDN and RCAN. These models focused mainly on increasing reconstruction quality, resulting in ever-growing models that are too slow to be used in real time. The same techniques were adapted to smaller architectures in an attempt to find the optimal model for real time inference.

The starting model, the EDSR baseline, had the inference throughput of 0.6 frames per second on the Intel Iris Xe LP integrated GPU with a 2x scaling factor on 720p inputs. The final model, after all the simplifications, improved this metric to 28 frames per second.

Perceptual and adversarial training were also viable with the final model, allowing it to hallucinate realistic details. Finally, the same model was also trained with images with JPEG compression, which forced the model to learn how to clean compression artifacts.

By pushing the boundaries of super-resolution techniques on consumer-grade hardware, this work paves the way for real-time, high-quality multimedia experiences on a wide array of devices.

List of Figures

List of Tables

List of Abbreviations

BRISQUE	Blind/Referenceless Image Spatial Quality Evaluator
CNN	Convolutional Neural Network
CSFM	Channel-wise and Spatial Feature Modulation
EDSR	Enhanced Deep Super-Resolution
ESPCN	Efficient Sub-Pixel Convolutional Network
ESRGAN	Enhanced Super-Resolution Generative Adversarial Network
FSRCNN	Fast Super-Resolution Convolutional Neural Network
HDR	High Dynamic Range
HR	High-Resolution
LCD	Liquid-Crystal Display
LPIPS	Learned Perceptual Image Patch Similarity
LR	Low-Resolution
MAE	Mean Absolute error
MLP	Multi-Layer Perceptron
MSE	Mean Squared error
MSSSIM	Multi-Scale Structural Similarity Index Measure
OLED	Organic Light-Emitting Diode
PSNR	Peak Signal to Noise Ratio
RCAB	Residual Channel-Attention Block
RCAN	Residual Channel Attention Network
RB	Residual Block
RDN	Residual Dense Network
RSAB	Residual Spatial-Attention Block
SISR	Single-Image Super-Resolution
SDR	Standard Dynamic Range
SRCNN	Super-Resolution Convolutional Neural Network
SR	Super-Resolution
SR3	Super-Resolution via Iterative Refinement
SRGAN	Super-Resolution Generative Adversarial Network
SSIM	Structural Similarity Index Measure
VDSR	Very Deep Super-Resolution

Contents

Chapter 1

Introduction

In contemporary society, multimedia content has become a dominant force in online consumption, exemplified by the widespread popularity of platforms like Netflix and YouTube. Furthermore, the undeniable growth of the video game industry in the last few decades showcases the profound presence of multimedia entertainment on modern culture.

Naturally, the capabilities of the displays used to consume multimedia content have also improved over time. Modern displays exhibit wider colour gamut, higher refresh rates, higher resolutions and higher dynamic range. While modern multimedia productions tend to leverage these capabilities, there's a multitude of old content produced in the past targeting lower resolution SDR displays.

Image resampling is generally used to map low-resolution content into high-resolution displays, but classic approaches such as bilinear and bicubic interpolation can not synthesise high-frequency information, resulting in images that are perceived as soft or blurry.

Machine learning has been eclipsing classical algorithms in computer vision and image processing, showing state-of-the-art performance in several classification and segmentation problems. Multiple models have also been proposed for image resampling (ANWAR; KHAN; BARNES, 2020). SRCNN (DONG; LOY; HE, et al., 2015), FSRCNN (DONG; LOY; TANG, 2016), ESPCN (SHI et al., 2016), VSDR (KIM; LEE, J. K.; LEE, K. M., 2016), ESRGAN (LEDIG et al., 2017), RDN (ZHANG, Y.; TIAN, et al., 2018) and RCAN (ZHANG, Y.; LI, et al., 2018) are all examples of single image super-resolution convolutional neural networks. Super-resolution can be used to either improve perceived quality or to reduce the amount of data being processed, transferred or rendered without sacrificing quality (WATSON, 2020).

Chapter 2

Motivation and Objectives

2.1 Motivation

Machine learning has been successfully applied in various fields of study, usually achieving state-of-the-art performance, however, these achievements are generally attained with computationally-expensive approaches that are usually unsuitable for real-time "online" inference.

Super-resolution models have surged as an alternative to signal resampling to bridge the gap between low-resolution multimedia content and high-resolution displays. However, the GPUs used to train and test these models are often orders of magnitude faster than the widely-available integrated GPUs found on consumer-level devices such as tablets and laptops.

These same devices often come equipped with modern high-resolution displays, which are almost never used to the limit due the lack of widely-available high-resolution content or other technical constraints that limit the resolution in which the content is served, like the available bandwidth. There's also a multitude of old low-resolution content produced in the past, and as display technology only tends to improve over time, this gap will only widen.

2.2 Objectives

The question is whether it's possible to use machine learning techniques in real time to bridge the gap and effectively increase the quality of lower resolution content even when running on low-power integrated GPUs.

This work aims to evaluate several super-resolution architectures and their building blocks, adapt their ideas to small-sized networks and optimise inference performance in an attempt to accelerate them for a wide array of consumer devices.

Starting from EDSR, the hyperparameters will be explored in search of a good balance between quality and performance. Newer architectural tricks such as the inclusion of dense connections or attention mechanisms will also be evaluated to check whether they're still beneficial after scaling the model down.

The primary objective will be to achieve the target performance of at least 24 frames per second upsampling 720p content to 1440p, which is a common resolution found on phones, tablets and computer displays. Then, architectural improvements will be explored in an attempt to increase reconstruction quality as much as possible within the available computational budget.

Adversarial training with perceptual loss functions will also be attempted to drive the model into being able to synthesise photo-realistic details. And, lastly, compression artefacts will be added to the training dataset in an attempt to push the model into learning how to denoise the picture and clean these artefacts, providing a superior image quality experience with lossy content.

Chapter 3

Digital Image Processing Concepts

3.1 The Electronic Visual System

The electronic systems we use to capture photos and videos are heavily inspired by the human visual system. Lenses are responsible for capturing and focusing the beam of light into the sensor, which is an array of photodiodes. The canvas is usually covered by a Bayer filter, which is used to capture the chromatic information, done via wavelength filtering in a patterned grid. The standard Bayer filter is half green, a quarter blue and a quarter red. This can be seen in figure 3.1. The green colour is favoured over the other two primaries to replicate human physiology.

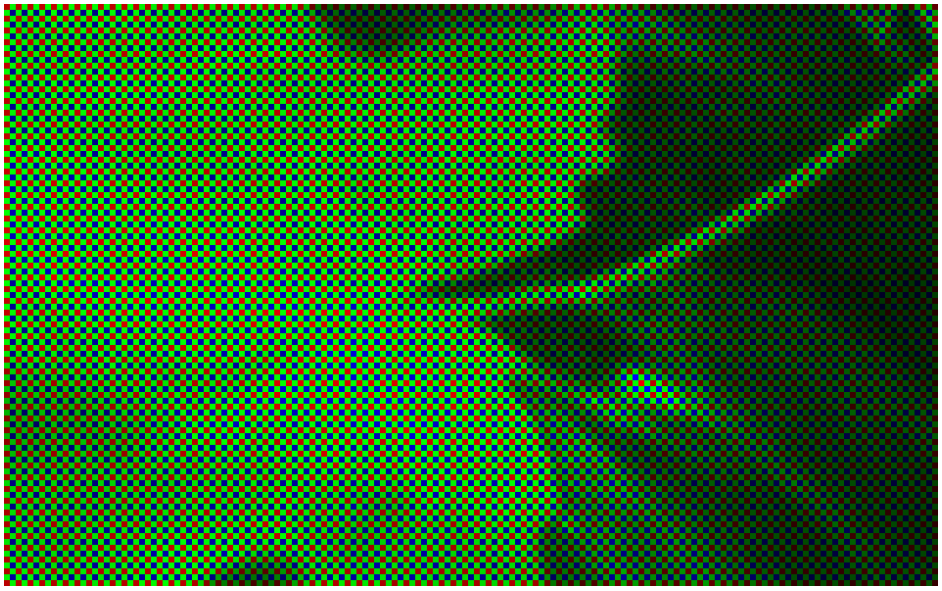


Figure 3.1: Example of Bayer pattern. Taken from a RAW picture before post-processing.

The raw Bayer pattern needs to be heavily processed before it resembles the captured scene. These processing steps often include lens shading correction, bad pixel correction, black level correction, demosaicing, colour correction, denoising, gamma correction, image sharpening, tone mapping, etc. These steps are generally done by a specialised fixed-function IP block called the image signal processor for performance reasons.

3.2 Colour Spaces

A colour space is a system of coordinates that can be used to define a colour subset composed by the linear combination of its primaries. The CIE 1931 XYZ colour space was one of the first studies to quantitatively define the relationship between the visible wavelengths in the electromagnetic spectrum and the human perception of colour (SHARMA; TRUSSELL, 1997). The CIE XYZ colour space covers the entire visible spectrum, however, electronic devices such as LCD or OLED displays can generally only output a subset of it. These subsets are derived colour spaces and they're usually shaped like triangles within the XYZ chromaticity diagram. The triangular shape comes from the RGB choice of primaries.

The CIE XYZ colour space is widely used as a connection space for colour conversions between smaller colour spaces such as sRGB (STOKES et al., 1996). Colour space conversions are usually aided by ICC profiles. Colour conversions are not needed when the data is encoded in the format the electronic device in question already expects, which is usually the case for sRGB images and sRGB computer monitors, but with premium consumer devices quickly expanding their colour gamuts towards Rec. 2020 (ZHU, R. et al., 2015), proper colour space conversions are required to avoid images that either look too saturated or washed out.

While most colour spaces used for multimedia content, like Rec. 709, Rec. 2020, DCI-P3 and sRGB, are usually defined with red, green and blue primaries, they're generally used as YCbCr when it comes to video and photography. Y stands for luminance, and for each luminance point there's a CbCr plane with all possible chromaticities. The Cb coordinate goes from blue to yellow while the Cr coordinate goes from red to green.

YCbCr is preferred to RGB because it allows luminance information to have its own channel, which is useful for image processing pipelines, image compression and image coding algorithms. The human visual system is generally more sensitive to luminance variations than it is to chromatic variations , and this is taken into account when processing the channels.

3.3 Chroma Subsampling

The human visual system has lower acuity for differences in colour than in brightness, this fact has motivated engineers to manipulate the data accordingly in order to save bandwidth and achieve reasonable quality at a fraction of the bitrate.

Chroma subsampling is a technique currently present in any modern video codec like H.265/HEVC, H.264/AVC, VP9 or AV1 (ZHANG, Y.; ZHU, L., et al., 2022). Lossy still image codecs like JPEG (WALLACE, 1991) also implement chroma subsampling. The technique consists of downsampling the chromatic information in the CbCr channels. Video codecs almost always use the "4:2:0" scheme which means the chromatic channels have half as many samples in each axis. In other words, there are 4 different luma samples for each pair of chromatic samples.

This can be also taken into account when optimising the performance of super-resolution systems. The amount of information the neural network needs to process and produce can be reduced if only the luminance channel flows through it. The hypothesis is that upscaling the Cb and Cr channels with classic FIR filter techniques won't affect the perceived quality of the final image in a meaningful way, but it might provide a speed-up in inference performance. This will be expanded upon in a later section.

3.4 Image Compression

The Discrete Cosine Transform is the key algorithm within JPEG's compression and coding pipeline, and the manipulation of its output is also the source of JPEG's grid-like compression artefacts.

Assuming an RGB input image, the first step to create a JPEG is converting the channels to YCbCr. Afterwards, 4:2:0 chroma-subsampling can be used to halve the amount of information. DCTs are then applied to the channels segmented into 8x8 blocks and its frequency components are quantised and truncated. This is a non-reversible step and information is lost, but the DCT captures most of the signal's information in the lower frequency components closer to the DC level in the top left. The resulting values are then written in a zig-zag order so zeroes are placed at the end of the bitstream, which increases the efficiency of the Huffman coding step that comes afterwards. This procedure is illustrated in figure 3.2.

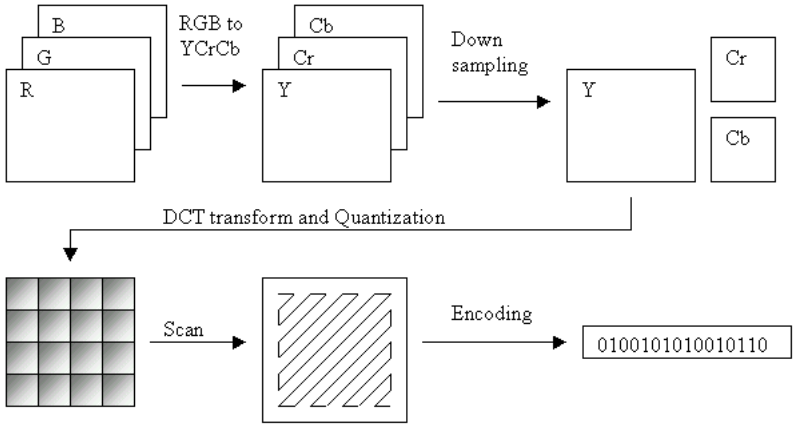


Figure 3.2: JPEG compression steps. Source: (SINGH, S., 2014).

Modern video codecs are designed with the same concepts in mind, but they expand the spatial model of intra-frame image compression to also take the temporal axis into account. Frames that contain the entire spatial information required to compose their own information are called I-frames. Frames that depend on a previous frame, and only contain the differences required to form itself, are called P-frames. Frames that depend on both past and future frames are called B-frames.

Image and video compression greatly reduce the amount of data needed to store multimedia content, but the highest compression ratios are only achieved with lossy techniques that create compression artefacts. SR CNNs can be trained with lossy LR images and lossless HR images to aid the network in learning how to undo the artefacts created by common compression schemes. This will also be expanded upon in a later section.

3.5 Image Resampling with FIR Filters

Resampling is the process of changing the number of samples of a discrete signal to obtain a new discrete representation of it that's either "bigger" or "smaller". The easiest and most classic way of resampling to a higher sample rate is via linear interpolation. Linear interpolation can be done in a cartesian plane, through both axes, creating what is called bilinear interpolation. Bilinear interpolation is the simplest interpolation algorithm, the easiest to compute and probably the most widespread one.

Bicubic interpolation is a higher quality alternative to bilinear interpolation. Instead of simply drawing a line between the nearest adjacent known values, the interpolation curve is made smoother by "looking further" into the next 2 pixels and fitting a polynomial curve

between all 4. The 2 most common bicubic filters are the Catmull-Rom (CATMULL; ROM, 1974) and the Mitchell-Netravali BC-Splines (MITCHELL; NETRAVALI, 1988).

This problem can be generalised as the convolution between a given resampling filter and the nearby input pixels, and filters with radii larger than 2 like the Lanczos filter are also used (DUCHON, 1979). The problem of using classic resampling filters to increase the resolution of an image is that these filters are incapable of creating sharp edges and fine detail without also introducing artefacts such as ringing or blocking. Figure 3.3 shows the difference between a real HR image and its resampled LR counterpart.

Figure 3.3: Difference between the HR reference and its upsampled LR counterpart.

Upsampling can increase the number of pixels but the missing information is not restored. Interpolation merely blends existing information to find points in between, it's not able to regenerate the missing high frequency information.

In short, the most common way of resampling images is treating each row/column as an independent 1-D signal and simply going over all of them until you have resampled the entire image. This means you have to choose a dimension to resample first but this is inconsequential to the end result. The resampling algorithm itself is pretty simple, for each output sample you simply centralise the filter on top of it and see which input samples end up inside of its window after you compute their equivalent positions, then you multiply these inputs by

their corresponding weights in the filter based on distance and sum these values. This is often implemented as a series of short convolutions. Figure 3.4 illustrates the process.

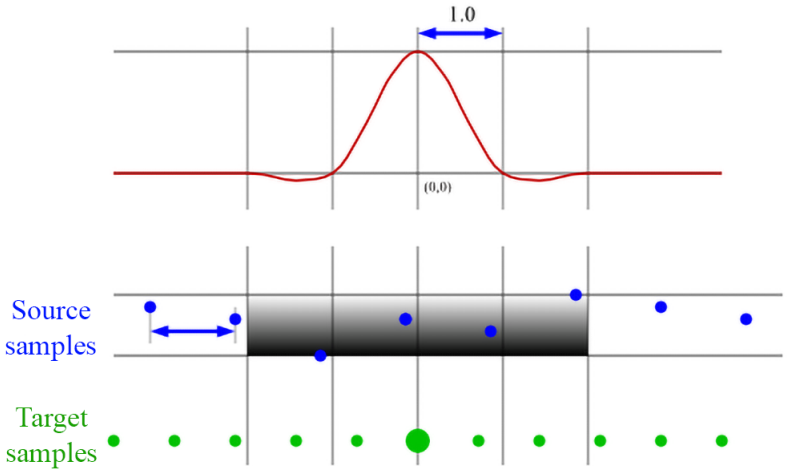


Figure 3.4: Orthogonal Resampling with FIR filters, as depicted by Jason Summer on ImageWorsener's documentation.

There's a different method, usually called polar, cylindrical or elliptical resampling, that does the operation in the 2-D domain directly. The only difference here is that all samples that fit inside the 2-D filter are now weighted simultaneously, which provides more accurate results but is also slower due to an increase in the number of mathematical operations for each output pixel. Figure 3.5 shows the impulse response for some filters when used in orthogonal (first row) and polar resampling (second row).

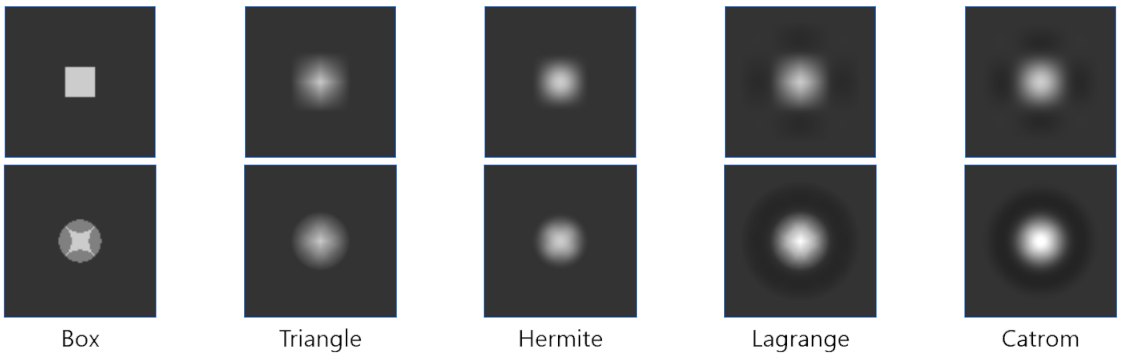


Figure 3.5: Orthogonal Resampling compared against polar resampling. Adapted from ImageMagick's documentation.

Chapter 4

Image Quality Metrics

While it is relatively easy for humans to judge image quality in a seemingly subjective way, computers need well-defined algorithms that can be mathematically and objectively computed. Efforts have been made to design metrics that correlate well to the human perception of quality, but this is still an open field of study.

When an image free of distortions is available, it can be used as a reference to measure the quality of other images derived from it. For example, different lossy codecs can be compared against the uncompressed original image. The techniques that require a lossless version of the image under test to also be available as a reference are collectively called full-reference image quality metrics.

Yochai Blau and Tomer Michaeli have demonstrated the counter-intuitive phenomenon that distortion and perceptual quality are actually at odds (BLAU; MICHAELI, 2018). The work shows that the human visual system does not necessarily prefer distortions that are numerically closer to the reference, and that algorithms that achieve lower distortion can deviate from natural scene statistics.

Perception metrics attempt to correlate better with how humans perceive image quality. Modern works usually employ machine learning techniques to learn how humans rate various distortions. The downside is that datasets with human opinion scores are usually required for training. While most perception metrics do not require reference images, it was also found that the distance between deep activations from classification networks such as VGG and AlexNet can be used as excellent full-reference perceptual metrics (ZHANG, R. et al., 2018).

4.1 Peak Signal to Noise Ratio

The mean squared error of a given distorted image against its reference is essentially the average of the squared differences between the pixels or sub-pixels. The MSE does not correspond to the human perception of quality, it treats all distortions numerically equal regardless of spatial significance in the image.

Since MSE is computed from pixel intensities, different bit depths can change the order of magnitude of the result and the numbers are not consistent between different images. PSNR solves this issue by using the MSE result as the noise metric and putting it against the signal's maximum level.

PSNR, however, does not solve the disconnection between MSE and the human perception of quality. It is bit-depth invariant and can be more easily used in comparison with multiple images of varying bit-depths, but it ultimately does not always correlate to how humans perceive quality.

4.2 Structural Similarity Index Measure

The SSIM (WANG, Z. et al., 2004) was one of the first attempts at solving the issues with MSE and PSNR. SSIM uses the averages, variances and covariance between the images to compute its result rather than simply using the absolute difference between pixel intensities. The result is a metric that's much more sensitive to high-frequency information such as acutance and sharpness, which correlates better to how humans perceive quality.

4.3 Blind Referenceless Image Spatial Quality Evaluator

BRISQUE (MITTAL; MOORTHY; BOVIK, 2012) is a natural scene statistic-based distortion-generic no-reference image quality assessment model that operates in the spatial domain. The model uses scene statistics of locally normalised luminance coefficients to quantify possible losses of naturality instead of computing distortion-specific features such as ringing, blur or blocking. The original implementation of the algorithm employs a support vector machine to perform regression from the extracted scene statistics and human-provided quality scores. In the original paper, the LIVE IQA database was used for training.

4.4 Learned Perceptual Image Patch Similarity

LPIPS (ZHANG, R. et al., 2018) stands for Learned Perceptual Image Patch Similarity, the metric is given by calculating the distance between features extracted from an image classification CNN, Alexnet by default. The reference and the input image are both given as inputs to the CNN individually, the feature maps created by the network's layers are then extracted, weighted and averaged. The rationale behind it is very straightforward, image classification CNNs are trained to understand contextual information so they can detect and classify objects. Naturally, similar images produce similar feature maps, but unlike pixel-based metrics such as MSE, perceptually similar images also produce similar feature maps, even when the pixels themselves greatly diverge.

Chapter 5

Machine Learning Concepts

5.1 Multi-Layer Perceptron

5.1.1 Neurons, Synapses and Activations

In artificial neural networks, a neuron is classically modelled as an element that receives, transforms and passes data. When the neuron has multiple inputs, the values are usually multiplied by weights and added together before an activation function is applied. Activation functions are generally non-linear to allow the networks to solve challenging complex problems (HORNIK; STINCHCOMBE; WHITE, 1989). This is illustrated in Figure 5.1.

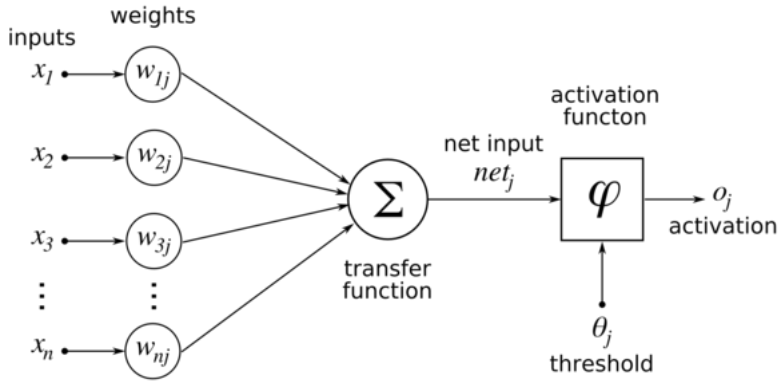


Figure 5.1: Neurons, synapses and activation functions.

Networks that take several inputs and propagate the data forwards through multiple layers with densely connected neurons are called multilayer perceptron networks (MLPs). The network is densely connected when every neuron in a given layer has a connection to all the neurons in the previous layer. This is illustrated in Figure 5.2.

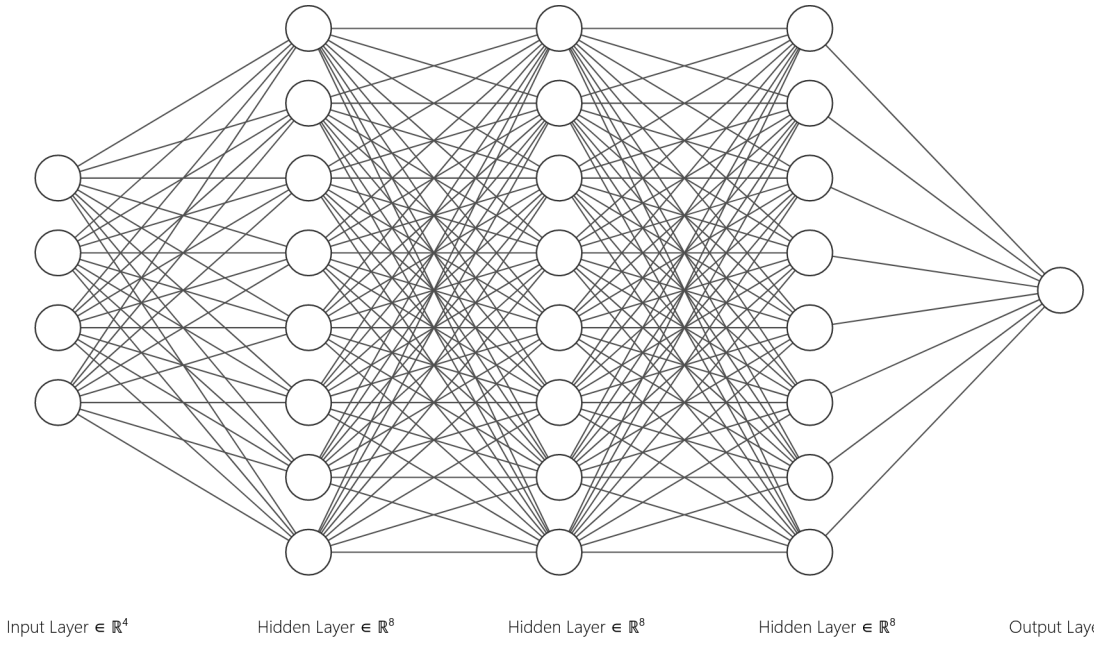


Figure 5.2: A densely connected MLP with 3 hidden layers.

Neural networks excel when there's an abundance of data describing the input to output relationship, but finding a mathematical solution that solves the problem is difficult. As an example, we can easily transform coloured pictures into their grayscale variants, but proposing an algorithm to do the opposite would be an incredibly arduous task. There is, however, a nearly infinite number of coloured images we can turn to grayscale and train a neural network to accomplish the inverse task.

5.1.2 Loss Functions

Neural networks that receive input-output pairs during training are called supervised. Supervised training consists of attempting to minimise the error between the output generated by the network and its corresponding reference. A loss function is used to compute the error, which is ideally reduced after each update in the weights. The most common loss functions in use are the mean squared error and the mean absolute error, which are usually favoured due to being easy to compute and differentiate.

5.1.3 Stochastic Gradient Descent and Backpropagation

Backpropagation (LINNAINMAA, 1976) (ROSENBLATT, 1962) is usually employed to efficiently compute the gradient vector with respect to the network weights and consequently

update their values throughout the network after each iteration, due to the ease of computing the gradients of a given layer using gradients from the following layer.

The major limitation with the various stochastic gradient descent algorithms is that as the weights can only be updated in small steps after each iteration, it's not guaranteed the global minimum will be found. The network converges when moving into any direction in vector space will cause the error to increase, which also happens when it is stuck inside a local minimum. Momentum can be added to the training algorithm, sometimes also called the optimiser, to stabilise the training path and potentially escape local minima. Adam is a good example of a widely used modern optimiser that employs momentum.

5.2 Convolutional Neural Networks

While MLPs can be used to solve problems related to images, they do not exploit the intrinsic spatial relationship between nearby pixels. Ignoring locality by fully connecting the output of every pixel to every neuron is computationally inefficient and semantically disastrous, images are formed by a grid of pixels and their individual intensities are mostly irrelevant when taken out of context.

Convolutional neural networks (FUKUSHIMA, 1980) are designed to explicitly exploit the spatial information in matrix-like data structures such as images (LECUN et al., 1998). Contrary to the MLPs, which have densely connected neurons, CNNs employ convolutions to extract information from nearby data points, sliding through them to form feature maps. Naturally, the same nonlinear activation functions are still used to allow the networks to solve complex problems.

Convolutional layers may have many filters, and their weights are generally trainable parameters. Each filter is responsible for outputting a single feature map from all channels in the input. For that to be possible, the filters actually have 3 dimensions, width, height and depth. Since each convolutional layer can have multiple filters and each filter outputs a single feature map, the number of feature maps (or channels) in the output matches the respective number of filters. This is illustrated in Figure 5.3.

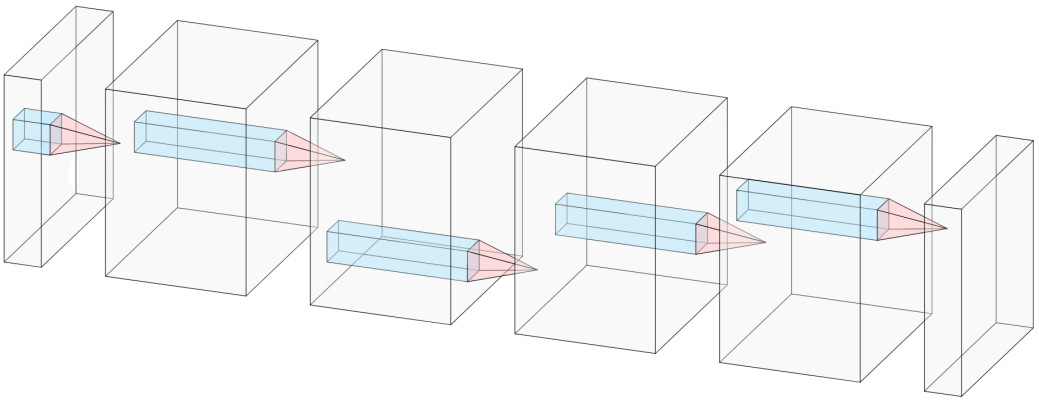


Figure 5.3: An example of CNN with 5 convolutional layers.

According to the PyTorch implementation of Conv2D, where the star is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels, the output value of the layer with input size (N, Cin, H, W) and output size (N, Cout, Hout, Wout) can be described as:

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) \star input(N_{in}, k)$$

Chapter 6

Related Work

SISR CNNs can be classified as either distortion-driven or perceptually-driven. The former attempts to minimise the pixel-wise difference between the supersampled output and its HR reference. The latter attempts to generate images that are allowed to objectively deviate from the HR reference as long as they look real.

Figure 6.1 depicts the difference between distortion-driven and perceptually-driven networks. On the left, the high-resolution version of the reference image is given, with dark rectangles with different orientations. The low-resolution counterpart, acquired artificially via downsampling, completely lacks the orientation information and has 9 squares that look identical. The distortion-based network attempts to minimise the error towards all possible outputs, which ends up creating a blurry image. The perception-based network, however, prefers to pick one of the possible answers, even if it is the wrong one.

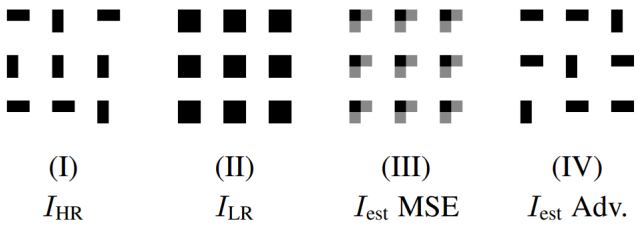


Figure 6.1: Difference in output between distortion-based loss functions and adversarial training. Source: (SAJJADI; SCHÖLKOPF; HIRSCH, 2017).

Despite the difference in intent, the building blocks that compose these 2 distinct types of SISR CNNs are generally the same. Perceptual approaches usually only diverge in how the network is trained.

6.1 SRCNN

The paper (DONG; LOY; HE, et al., 2015) defining the SRCNN (Super-Resolution Convolutional Neural Network) model was published in 2014 and it describes an architecture (Figure 6.2) that receives a bicubic interpolated image as its input and improves its quality through consecutive convolutional layers. There are no skip connections to bypass the low-frequency information throughout the network, which means the final layer needs to reconstruct the entire HR image. The authors describe that increasing the number of layers or the number of filters causes the model to become mathematically unstable and difficult to train, a problem that was subsequently solved with the introduction of residual training.

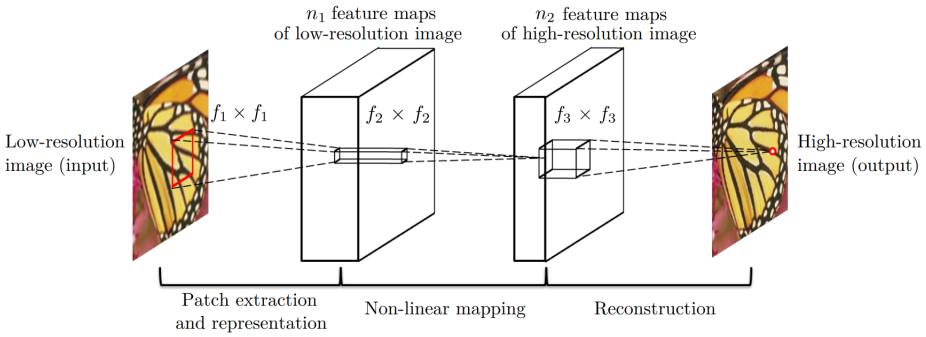


Figure 6.2: The model proposed by SRCNN. Source (DONG; LOY; HE, et al., 2015).

6.2 FSRCNN

FSRCNN (Fast Super-Resolution Convolutional Neural Network) (DONG; LOY; TANG, 2016) came as an attempt to make the SRCNN model faster. The major difference between the two is that FSRCNN receives the original low-resolution image as its input, introducing a transposed convolution at the end that increases the scale to the target resolution. The new network introduces channel shrinking and expanding layers with 1x1 filters, and it replaces the internal layers by a series of convolutional layers with smaller 3x3 filters. This allows the network to have a similar receptive field, but with fewer weights which was found to be more stable. Figure 6.3 compares SRCNN and FSRCNN.

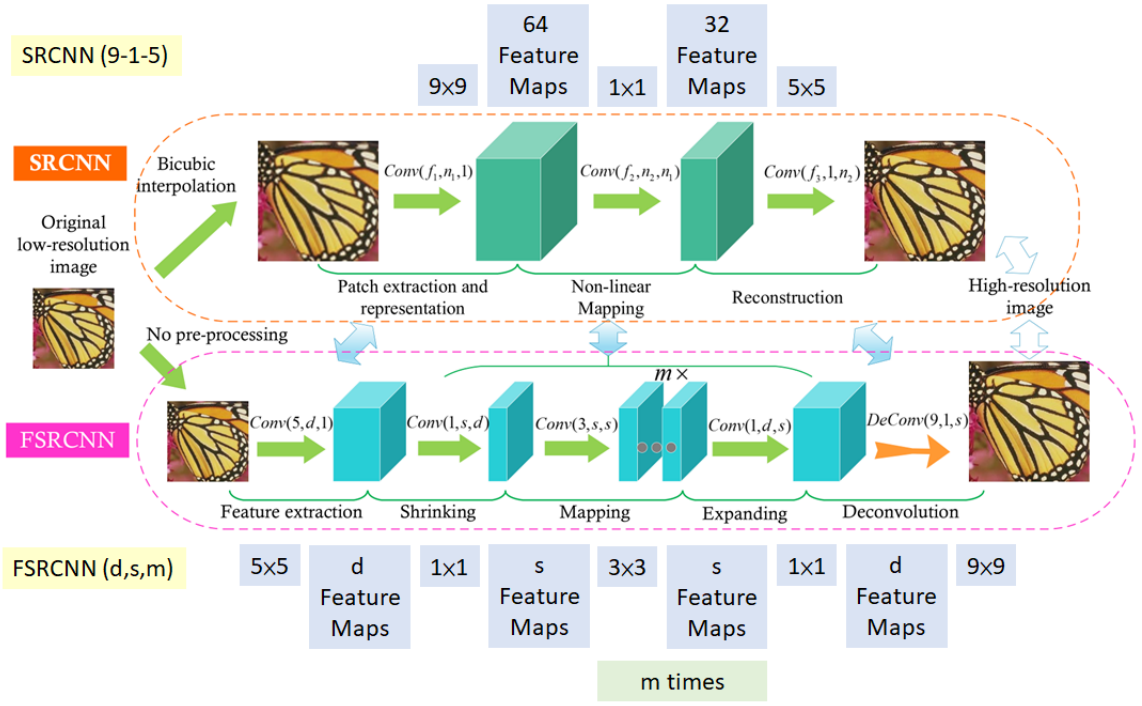


Figure 6.3: The model proposed by FSRCNN. Source (DONG; LOY; TANG, 2016).

6.3 ESPCN

ESPCN (Efficient Sub-Pixel Convolutional Neural Network) (SHI et al., 2016) introduced the pixel shuffling operation into the super-resolution problem. Pixel shuffling, sometimes also called subpixel convolution or depth to space operation, is the act of simply rearranging the pixels to increase the size of the height and width dimensions while reducing depth. Pixel shuffling quickly replaced the transposed convolution as the most popular upsampling method employed to increase spatial resolution as it was found to be more stable and easier to train. Figure 6.4 illustrates the model.

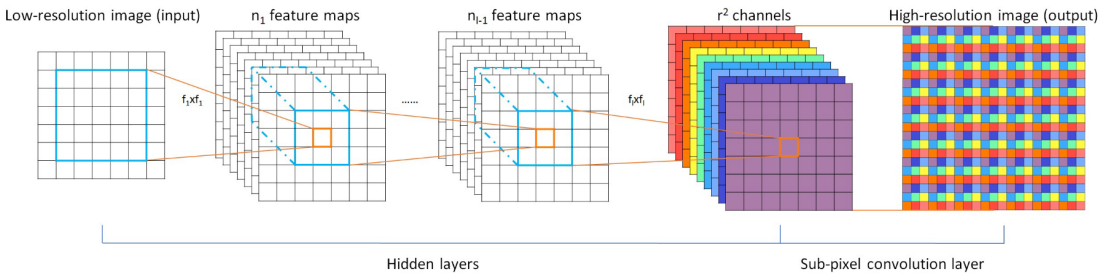


Figure 6.4: The model proposed by ESPCN. Source (SHI et al., 2016).

6.4 VDSR

VDSR (Very Deep Super-Resolution) (KIM; LEE, J. K.; LEE, K. M., 2016) was the first super-resolution model to employ residual training. Long skip connections are generally used to forward abundant low frequency information to the end of the network. This technique allows the network to focus on learning the missing high frequency components, instead of learning how to reconstruct the entire image. This architectural choice allowed VDSR to be deeper and to converge faster than previous models. Figure 6.5 illustrates the model.

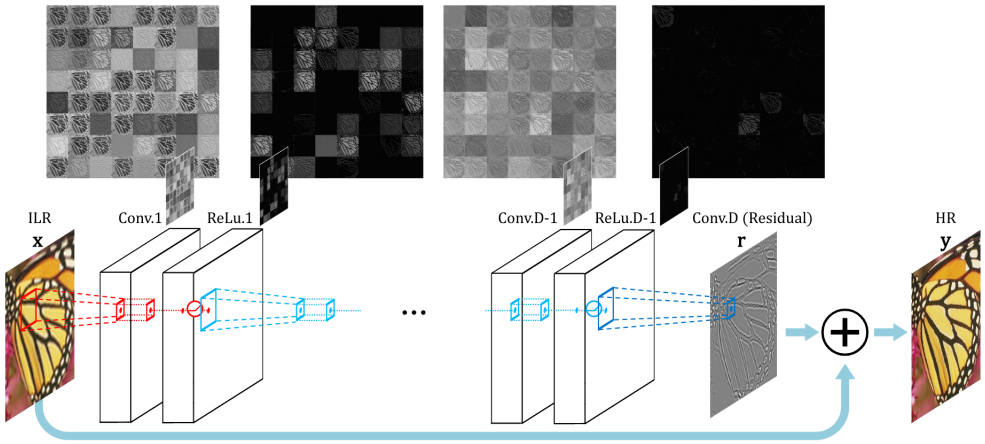


Figure 6.5: The model proposed by VDSR. Source (KIM; LEE, J. K.; LEE, K. M., 2016).

6.5 EDSR

EDSR (Enhanced Deep Super-Resolution) (LIM et al., 2017) was published in 2017 and it achieved state of the art performance at the time by leveraging residual training at multiple levels rather than employing a single long-skip connection. The residual blocks were adapted from ResNet and SRResNet. Upsampling is handled by the pixel-shuffle operation introduced by ESPCN. The authors also found that adding residual scaling before the short skip connections in large configurations helps training stability significantly. Figure 6.6 illustrates the model.

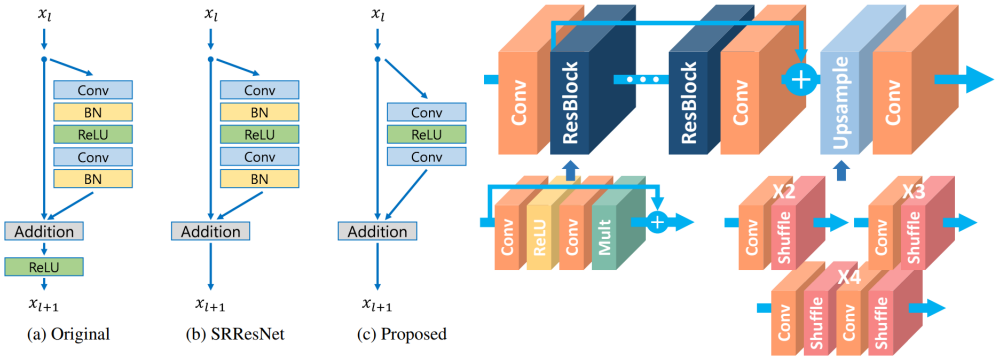


Figure 6.6: The model proposed by EDSR. Source (LIM et al., 2017).

6.6 SRDenseNet

SRDenseNet (Super Resolution Dense Network) (HUANG, G. et al., 2017) introduced dense skip connections in a very deep network. In the proposed model, the feature maps of each layer are propagated into all subsequent layers, providing an effective way to combine the low-level features and high-level features to boost the reconstruction performance. In addition, the dense skip connections in the network enable short paths to be built directly from the output to each layer, alleviating the vanishing-gradient problem of very deep networks. Figure 6.7 illustrates the model.

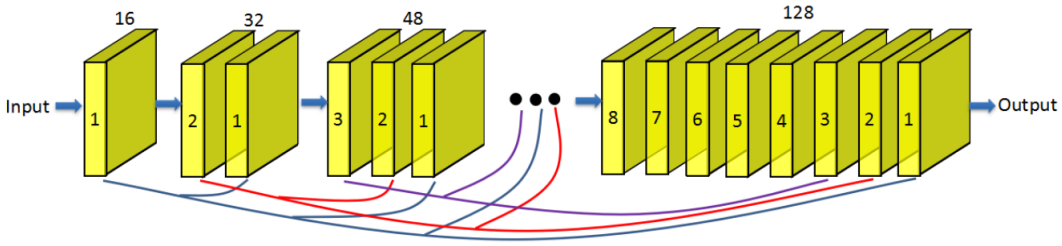


Figure 6.7: The model proposed by SRDenseNet. Source (HUANG, G. et al., 2017).

6.7 RDN

Feature map concatenation and element-wise addition do not need to be mutually exclusive. The idea of combining both techniques first showed up in RDN (Residual Dense Network) (ZHANG, Y.; TIAN, et al., 2018), in which a new building block was proposed, the residual dense block (Figure 6.8). The block employs channel concatenation through convolutional layers in an attempt to maximise learning potential, but the result is compressed back to the

original dimension and added back to the original feature maps the exact same way it did in the residual blocks that preceded it.

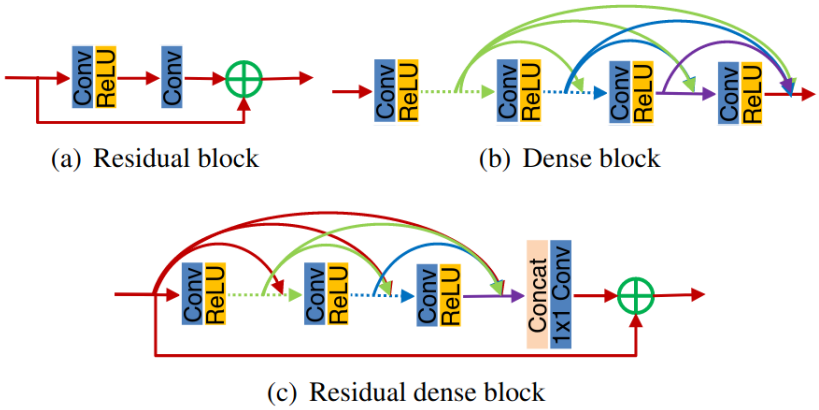


Figure 6.8: The residual dense block. Source (ZHANG, Y.; TIAN, et al., 2018).

According to the RDN authors, previous networks neglect to fully use the information from each convolutional layer and therefore are incapable of capturing hierarchical features of varying scales, viewing angles and aspect ratios. The residual dense block introduced by RDN consists of dense connected layers and local feature fusion with local residual learning. The output of one RDB has direct access to each layer of the next RDB, resulting in a contiguous state pass. For that reason, each convolutional layer in a RDB has access to all the subsequent layers and passes on the information that needs to be preserved. Figure 6.9 illustrates the model.

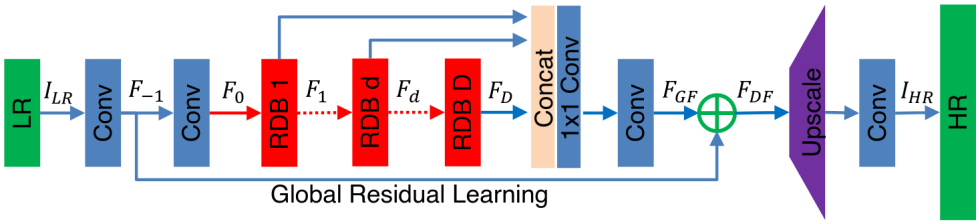


Figure 6.9: The model proposed by RDN. Source (ZHANG, Y.; TIAN, et al., 2018).

RDN receives a LR image as its input and performs the pixel shuffle operation from ESPCN to upscale the LR feature maps. However, concatenations are also used to extract information from all residual dense blocks before the feature maps are added to the LR input.

6.8 RCAN

Attention is being widely used in several other areas within machine learning, especially in natural language processing and it was the basis for the development of the transformer. Attention mechanisms can be used in the SR problem to adaptively rescale features and allow the network to give more emphasis to informative pixels.

Attention mechanisms come in different ways, the feature maps can be rescaled either by channel, by spatial position or individually. The rescaling factor can be computing in different ways as well.

RCAN (Residual Channel Attention Network) (ZHANG, Y.; LI, et al., 2018) was one of the first models to introduce attention mechanisms into the SR problem. The channel attention block used in RCAN (Figure 6.10) was first introduced by [Squeeze-and-Excitation Networks]. The channel reduction and expansion steps, alongside the sigmoid activation function, allow the mechanism to learn non-linear relationships between channels.

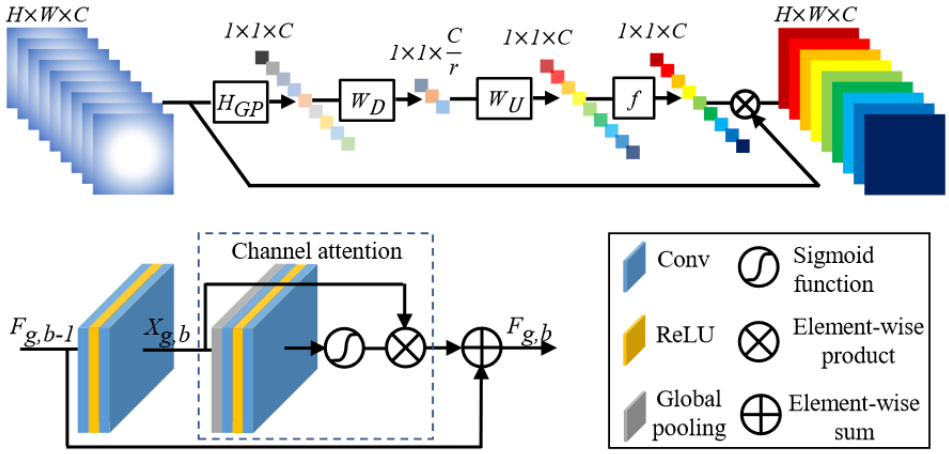
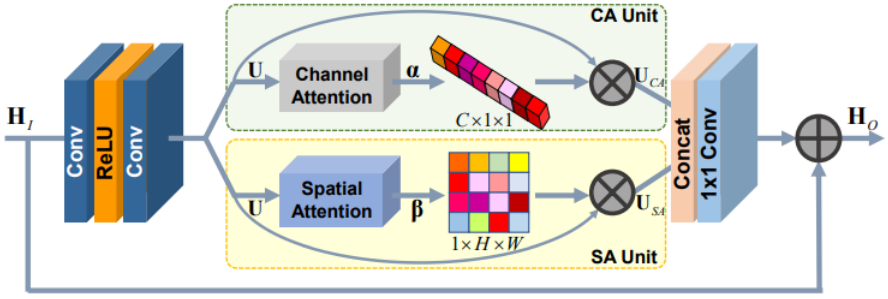


Figure 6.10: The channel attention mechanism. Source (ZHANG, Y.; LI, et al., 2018).

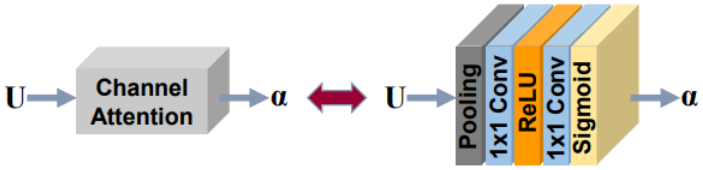
RCAN arranges the residual-channel attention blocks in a way that allows residual training to be done in multiple levels. Unlike RDN, RCAN does not employ channel concatenation. Instead, element-wise sums are used whenever information needs to be passed to the following layers. This reduces the amount of redundant information being passed on and also decreases the number of parameters since concatenation increases the size of the filters linearly.

6.9 CSFM

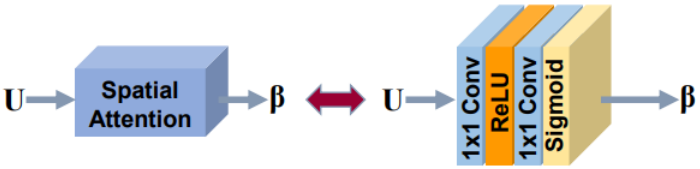
CSFM (Channel-wise and Spatial Feature Modulation) (HU et al., 2018) expands the idea of using attention mechanisms to rescale information by employing both spatial and channel attention mechanisms simultaneously to adaptively rescale features with pixel specific granularity (Figure 6.11). The modules in the network are also densely connected so the information can be adaptively combined to maximise reconstruction quality.



(a) Channel-wise and spatial attention residual (CSAR) block



(b) The operations of channel-wise attention



(c) The operations of spatial attention

Figure 6.11: The attention modules in CSFM. Source (HU et al., 2018).

6.10 SRGAN and ESRGAN

Adversarial training is employed to encourage the network to favour natural looking solutions that are perceptually similar to real images. The drawback is that these networks usually score worse in pixel space metrics like MSE and PSNR.

Perceptual quality is achieved by introducing a discriminator capable of classifying the images as real or fake. The discriminator's loss function is based on its ability to distinguish the 2 classes while the generator's loss function is based on its ability to trick the discriminator

into classifying fake images as real. Figure 6.13 illustrates the models employed by SRGAN (Super-Resolution Generative Adversarial Network) (LEDIG et al., 2017).

SRGAN was the first to generate photo-realistic pictures full of detailed texture. However, in order to stabilise training and improve image quality, a VGG based content loss is also employed to prevent the generator from creating images that diverge too much from the ground truth. Figure 6.12 shows the difference caused by changing the loss function.

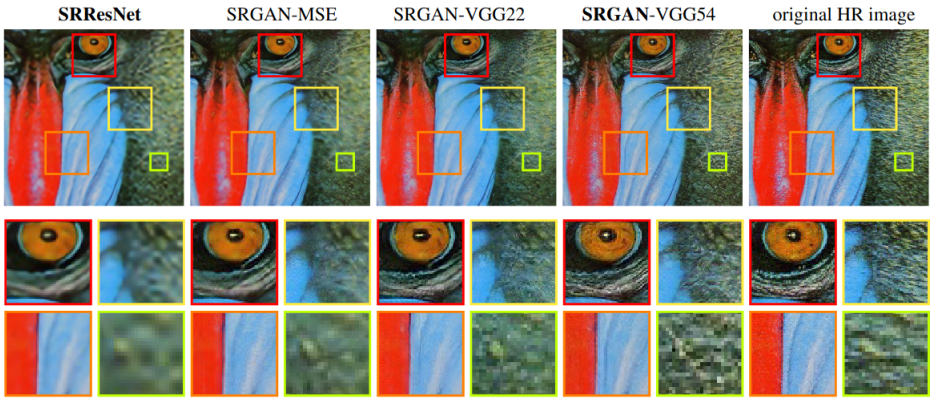


Figure 6.12: The difference caused by using different loss functions on the SRGAN model, starting from SRResNet which shares the same generator. Source (LEDIG et al., 2017).

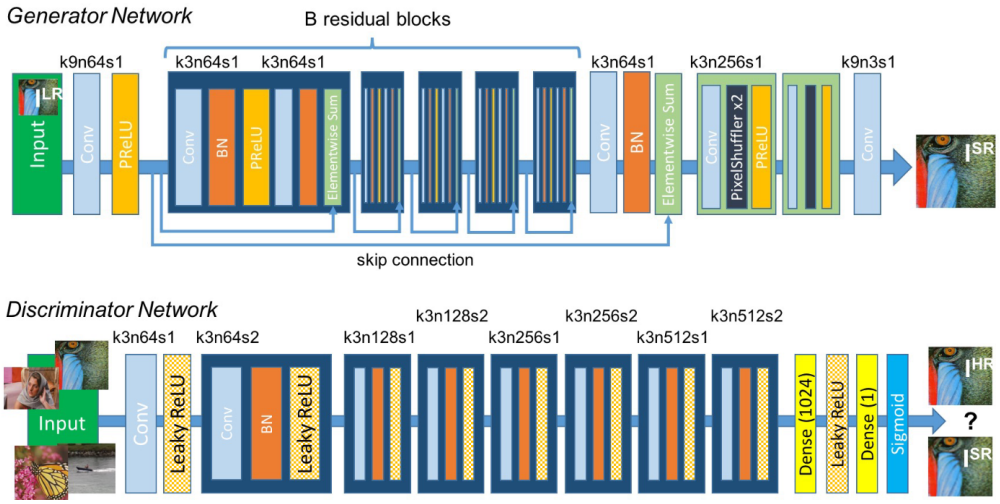


Figure 6.13: The models proposed by SRGAN. Source (LEDIG et al., 2017).

ESRGAN (Enhanced Super-Resolution Generative Adversarial Network) (WANG, X. et al., 2018) improved the SRGAN model by modifying the generator using residual training techniques found in EDSR and RDN. ESRGAN also borrows the relativistic average discriminator introduced by [The relativistic discriminator: a key element missing from standard GAN]. Instead of simply classifying images as real or fake, the relativistic discriminator estimates the

probability that the given real data is more realistic than fake data. This modification helps the discriminator learn sharper edges and more detailed textures.

6.11 SR3

SR3 (Super-Resolution via Repeated Refinement) (SAHARIA et al., 2021) is an approach to image Super-Resolution via Repeated Refinement. The model adapts denoising diffusion probabilistic models to conditional image generation and performs super-resolution through a stochastic denoising process. Inference starts with pure Gaussian noise and iteratively refines the noisy output using a U-Net model trained on denoising at various noise levels. Figure 6.14 illustrates the model.

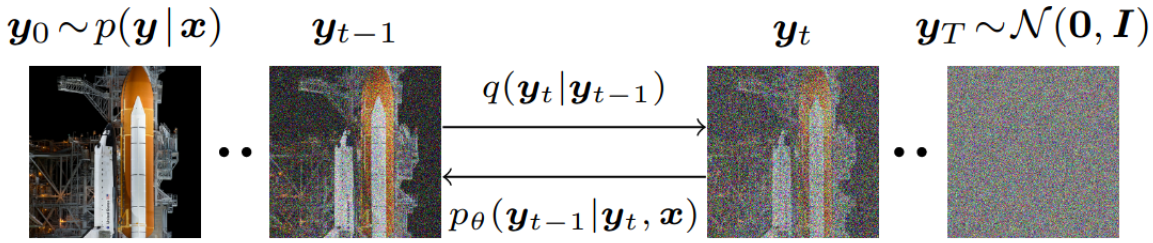


Figure 6.14: The stable diffusion training in SR3. Source (SAHARIA et al., 2021).

Chapter 7

Methodology

Films and TV series are usually distributed with 24 frames per second, but it's not rare for other types of content such as online videos to be streamed at 30 or 60 frames per second. Most displays have a baseline refresh rate of 60 Hz as well, and the premium panels usually have native refresh rates of 120 Hz. With that in mind, a SR system that targets real-time video content should be able to generate at least 24 high resolution frames per second as the bare minimum.

Inference performance of the exact same model varies greatly between different ML libraries and hardware devices. This issue could be tackled from 2 different angles, it's possible to either choose a specific model and then attempt to optimise it well enough to run at the available hardware, or create a hardware constraint from the get go and design the ML model around it.

Inference performance is a very active field of study and the recent smartphones are already shipping with dedicated NPU IPs to perform fast and power-efficient inference of ML models. Nvidia pioneered the use of GPUs in the ML field and their current products have dedicated tensor cores that are optimised to do well in ML specific tasks. However, most consumer grade personal computers do not have powerful dedicated GPUs, relying solely on much less capable integrated hardware. Table 7.1 illustrates the theoretical FP16 and INT8 performance of some common hardware accelerators in TFLOPS.

To maximise the number of devices capable of utilising the model for real-time inference, the model needs to be constrained to a number of operations per second that can be supplied by integrated GPUs.

Table 7.1: Theoretical throughput for various computing devices.

Devices	Type	FP16 TFLOPS	INT8 TOPS
Nvidia Tesla A100	Server GPU	312	624
Nvidia RTX 3080	Gaming GPU	119	238
Apple A15 Neural Engine	Mobile NPU	15.8	15.8
Intel Iris Xe-LP	Mobile iGPU	4.2	8.3

The following sections will focus on finding the best model that fits within this predetermined computational capacity to ensure real-time inference of video content is possible even when running on low-power devices such as smartphones and laptops.

7.1 Environment Setup

All models described in the following sections were implemented using TensorFlow and Keras. The choice of software stack was done purely by familiarity, as similar results are expected regardless of what is used to train the models.

The Adam optimiser was used for backpropagation with a learning rate of 1E-4. The mean absolute error was used as the distortion-based loss function since it has been experimentally proven to converge well, and because it became the standard in recent works. The following equation depicts the loss function.

$$L_{MAE} = \frac{1}{n} \sum \left| Y_{Pred} - Y_{Ref} \right|$$

All preprocessing steps, such as downsampling the images, applying the degradation models and splitting them into smaller patches were done with Wand, ImageMagick's binding for Python. The Tensorflow implementations of PSNR and SSIM, the PyTorch implementation of LPIPs and the Matlab implementation of BRISQUE were used to compare the results.

The models were trained on cloud GPUs provided by Google Colab Pro. In order to make the service more affordable, Google does not make any guarantees about which GPU will be available at any given time, which means different sessions might get different hardware making training time comparisons difficult to make. The GPUs available in Colab often include Nvidia K80s, T4s, P4s and P100s.

Inference performance was evaluated on the Intel Edge Cloud using the DL Workbench, which optimises inference performance using the OpenVINO toolkit and allows the user to experiment with different targets, including the Iris Xe LP GPU.

7.2 Datasets

All models described in the following sections were trained with the training portion of the DIV2K dataset (AGUSTSSON; TIMOFTE, 2017) (TIMOFTE et al., 2017) (illustrated in Figure 7.1), which is composed of 800 high resolution images with a large content diversity. The reasoning behind this choice of datasets comes from the fact that DIV2K gives a very good representation of real life scenes captured by cameras and was widely used by previous works, including EDSR, RDN and RCAN. The standard testing datasets, Set5 (BEVILACQUA et al., 2012), Set14 (ZEYDE; ELAD; PROTTER, 2010), B100 (MARTIN et al., 2001), Urban100 (HUANG, J.-B.; SINGH, A.; AHUJA, 2015), and Manga109 (MATSUI et al., 2017) (AIZAWA et al., 2020) were used for testing. Some samples from the test datasets are shown in Figures 7.2, 7.3, 7.4, 7.5 and 7.6.

Figure 7.1: Samples from the DIV2K dataset.

Figure 7.2: Samples from the Set5 dataset.

In order to increase computational efficiency and better utilise the GPU's VRAM during training, the HR images were split into smaller 64x64 patches. Since super-resolution requires

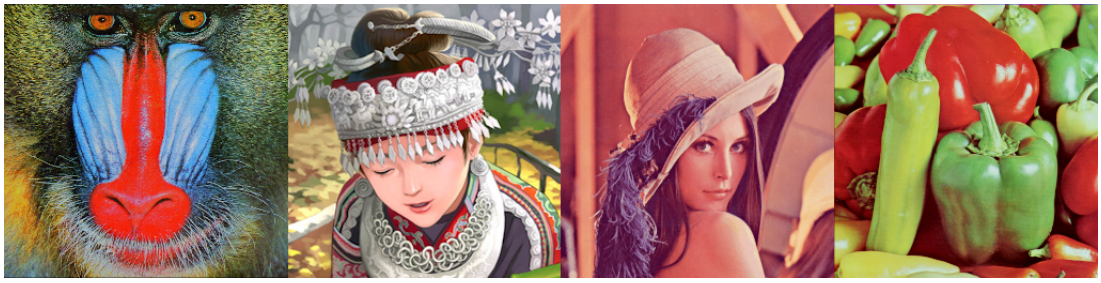


Figure 7.3: Samples from the Set14 dataset.

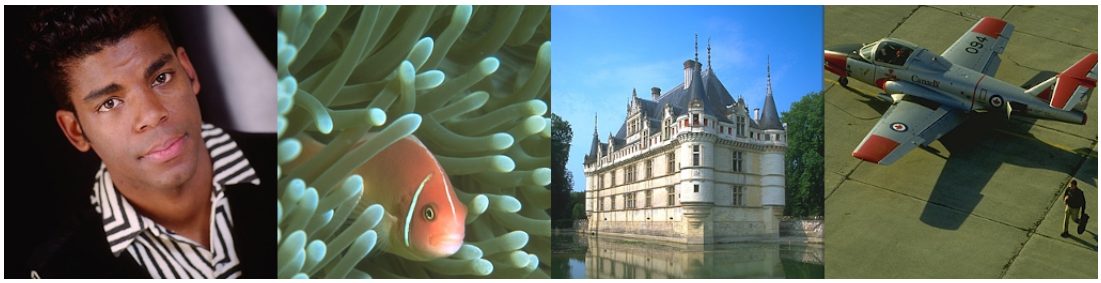


Figure 7.4: Samples from the B100 dataset.

Figure 7.5: Samples from the Urban100 dataset.

Figure 7.6: Samples from the Manga109 dataset.

a LR/HR pair for training, the LR patches were artificially obtained from the HR counterparts using a box filter to average the pixels. While bicubic resampling is usually preferred and more visually pleasing to the human eye, it also creates artefacts such as ringing that can not be found in the original image. This is a problem because it forces the network to learn how

to fix these artefacts, since they'll always be present in the data fed into it. However, ringing artefacts are not guaranteed to be present in real low resolution images that have not been bicubic downsampled, which means the training data would not be representative.

Lastly, a different degradation model applied JPEG compression to the downsampled images in an attempt to force the network into learning how to restore real world compression artefacts caused by DCT quantisation and chroma subsampling.

Chapter 8

Architectural Experiments

8.1 Choosing Starting Point

As discussed before, several super-resolution networks have been proposed and they all implement ideas like channel concatenation, residual connections and attention mechanisms slightly differently. Super-resolution is still an open field of study and there is no consensus on what is the best possible architecture to solve the problem.

Comparing the building blocks introduced by different networks is a problematic endeavour. It's common for newer networks to change several aspects about their architectures from paper to paper, metaparameters such as the number of layers, how many filters are used in each convolutional layer and where the long and short skip connections happen are good examples of things that can wildly change between different works. For this reason, it's usually difficult to make direct comparisons or be certain about where the improvements are coming from.

Nevertheless, it's still possible to infer a few things when these papers introduce game-changing ideas. SRCNN showed us that CNNs can be used for SR. FSRCNN showed us that slower inference speed does not necessarily mean worse reconstruction quality. VDSR and EDSR showed us that it's possible to train very deep networks if skip-connections are employed. RDN showed us that dense connections can improve learning capacity. RCAN introduced channel attention for better fine detail. SRGAN and ESRGAN showed that CNNs can benefit from adversarial training for photo-realistic results. Finally, SR3 introduced the idea of iteratively denoising a previously-upsampled version of the LR image to find its HR counterpart. Figure 8.1 gives an overview of the various architectural choices.

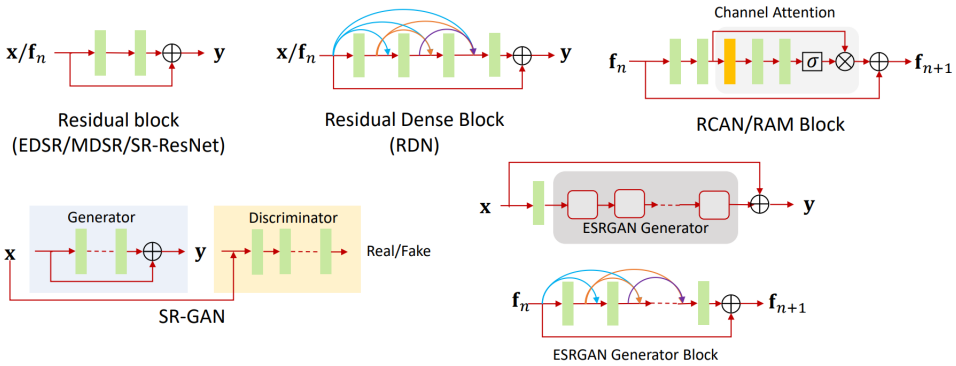


Figure 8.1: An overview of the various SISR CNN architectures. Adapted from: (ANWAR; KHAN; BARNES, 2020).

As seen before, RDN beats EDSR when it comes to reconstruction quality despite the fact that it has fewer parameters. This fact indicates that dense connections help the network learn how to solve the super-resolution problem more efficiently. However, concatenating the feature maps between consecutive convolutional layers increases the size of the filters in the depth dimension, which might not be ideal when the objective is to perform real-time inference.

Likewise, RCAN beats RDN despite having fewer parameters. The authors showed that the attention mechanisms only provide a marginal PSNR improvement in set5, which is a good indicator that deeper networks like RCAN might perform better than networks with more filters in each convolutional layer such as EDSR and RDN. Since 3x3 is by far the most common kernel size for convolutional layers in SISR CNNs, deeper networks have increased receptive fields for each output pixel and might be able to learn more contextual information from the surrounding pixels. However, whether or not these assumptions hold true in a shallow network is something that needs to be investigated.

It's important to note that this study will intentionally skip SR3 as a candidate for real-time super-resolution due to the fact that it needs multiple forward passes to completely denoise the LR image, making it intrinsically slower than the alternatives.

Since EDSR employs all the common architectural choices introduced by the earlier models, and both RDN and RCAN seem to be different attempts at improving it further, it's reasonable for this work to start from EDSR as well.

The final model described by the EDSR paper contains 32 residual blocks and 256 filters in each convolutional layer. However, the authors have also defined a smaller baseline model

with 16 residual blocks and 64 filters, which is better aligned with the real-time inference objective. The following image illustrates the architecture, it's important to remember that the baseline model does not require residual rescaling to stabilise training. Figure 8.2 illustrates the baseline model.

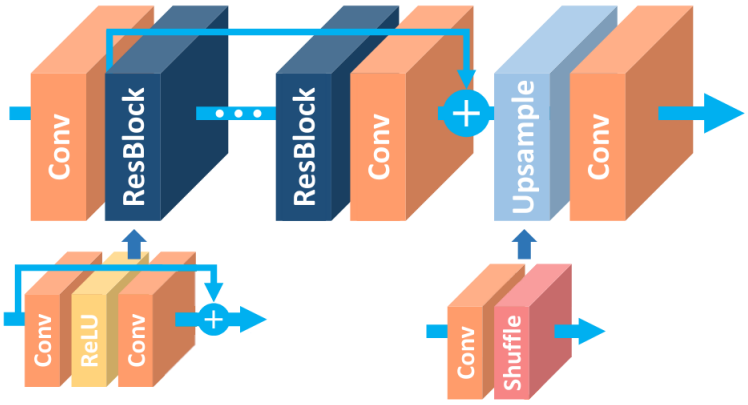


Figure 8.2: The EDSR baseline architecture. Source: (LIM et al., 2017).

Table 8.1 compares EDSR, RDN and RCAN and the smaller baseline model defined by the EDSR authors. The number of operations were calculated for a single RGB 1280x720 image and a 2x scaling factor.

Table 8.1: Throughput requirements for various SISR CNN models.

CNN	Configuration	Weights	TFLOP/Frame	TFLOPS 24 FPS
RCAN	RG10RCAB20	15 M	28	672
RDN	D16C8G64	22 M	41	984
EDSR	B32F256	40 M	74	1776
EDSR Baseline	B16F64	1.34 M	2.5	59

Despite the baseline EDSR model being much smaller than the final EDSR model, at only around 3% the number of weights or operations, it would still require a fast and power-hungry discrete GPU for real-time inference at almost 60 TFLOPS for 24 frames per second. The following sections will describe how this model can be optimised to achieve the best compromise between image reconstruction quality and inference performance.

8.2 The Baseline Model

The baseline model was trained for 100 epochs for initial evaluation. The Adam optimiser was quickly able to reduce the error between the supersampled images and their respective HR references, but the improvements in reconstruction quality became smaller with each consecutive epoch. Table 8.2 gives an overview of the baseline model's reconstruction quality across different test datasets.

Table 8.2: The baseline model's reconstruction quality.

Dataset	MAE	PSNR	SSIM	MSSSIM
Set5	0.0117	34.69	0.9386	0.9969
Set14	0.0208	29.86	0.8763	0.9914
B100	0.0214	29.60	0.8726	0.9936
Urban100	0.0224	29.05	0.9054	0.9953
Manga109	0.0113	34.18	0.9608	0.9980

When it comes to training speed, each epoch took approximately 157 seconds running on an Nvidia Tesla P100, which amounts to 4 hours and 20 minutes of total training time. The baseline model was tested on an Intel Iris Xe GPU on the Intel Edge Cloud and it achieved a throughput of 0.6 frames per second. The GPU is rated at 2.07 TFLOPs for FP32, which translates to a maximum theoretical throughput of approximately 0.8 frames per second running this model in single precision. The difference between the numbers can be explained by the software overhead, setup time, memory copying operations, etc.

8.3 Separable Convolutions

Separable convolutions are drop-in replacement for normal convolution layers that break the common operation in 2 distinct steps. Instead of taking information from all channels simultaneously in each kernel, a smaller convolution is performed for each channel individually, and this information is then mixed with a 1x1 convolution that receives information from all channels.

This modification reduces the computational cost by a factor of almost k^2 due to the reduction in the number of operations. With 3x3 filters, that results in layers that are theoretically

almost 9 times easier to compute. However, most convolutional layers in a CNN depend on the feature maps computed by a previous layer, which means it might not always be possible to saturate the available resources with parallel operations and therefore we should not expect a reduction of k^2 in the inference latency of a single frame. Figure 8.3 describes a modified Residual Block built with separable convolutions. Table 8.3 shows the difference between the starting point and its first modification with separable convolutions.

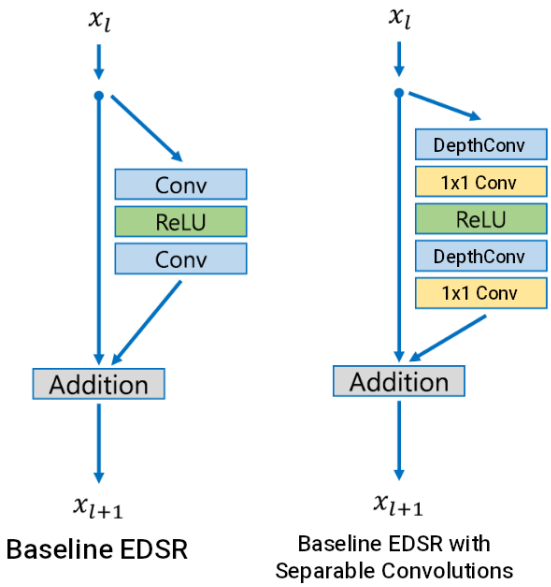


Figure 8.3: EDSR's residual block with separable convolutions. Adapted from: (LIM et al., 2017).

Table 8.3: The EDSR baseline with and without separable convolutions.

Configuration	Weights	TFLOP/Frame	TFLOPS 24 FPS	TFLOPS 60 FPS
B16F64	1.34 M	2.47	59.3	148.2
B16F64 (Sep)	174 k	0.32	8.0	19.4

A B16F64 model with separable convolutions was trained in order for it to be evaluated. When it comes to performance, the training time for each epoch actually increased from 157 to 218 seconds, a number that is 38% higher. Separable Convolutions are relatively niche and only used on models aimed at mobile devices with limited resources, which means the operation might not be as well optimised as the conventional convolution in cuDNN, Nvidia's CUDA Deep Neural Network library.

Server GPUs like the Tesla P100 are extremely good at performing parallel operations, it might be possible that the overhead created by having to perform 2 distinct convolutional steps might be enough to make the entire operation slower due poor arithmetic intensity, the ratio of compute to memory operations.

Unlike GPUs, which expect the workload to be massively parallel in an explicit way, modern CPUs are designed to execute a few threads as quickly as possible. This is done with out of order execution and branch prediction, a CPU can find the implicit parallelism within a thread and split the workload in independent portions so they can be executed out of order and in parallel spread across multiple execution units such as ALUs and FPUs. A good portion of a modern CPU is dedicated to predicting which branches will be taken so the correct instructions can be fetched from memory and put into the high speed caches ahead of time.

For these reasons, CPUs are usually not the ideal hardware devices for machine learning tasks, which are mostly compute intensive and easily parallelizable. With that in mind, the lower arithmetic intensity provided by the separable convolutions might provide a performance increase when running on a CPU even if the operations are not well optimised to exploit SIMD or vector extensions such as SSE and AVX. CPUs simply don't have enough resources to execute as many parallel operations as GPUs.

To confirm this hypothesis, the models were also benchmarked on an Intel Xeon Gold 6338N CPU. The baseline model achieved a throughput of 1.31 FPS and its version with separable convolutions increased it to 2.33 FPS. Despite being a clear performance improvement, it's also clear that reducing the number of arithmetic operations by a given factor does not necessarily correspond to a performance increase of the same factor.

When it comes to reconstruction quality, the model with separable convolutions had a loss function that was roughly 17% higher at the end of training. If the objective was to perform inference on CPU targets, the tradeoff would certainly be worth it. On GPUs however, it seems like the increased number of temporal dependencies ends up outweighing the reduction in the number of operations. All tests done in the following sections were done using normal convolutions.

8.4 Number of Filters per Layer

When it comes to model size, the two main factors that contribute to the computational complexity and number of weights are the number of layers and the number of convolutional filters per layer. As explained before, the baseline EDSR model contains 16 residual blocks and 64 filters per convolutional layer, but this is not necessarily optimal and a different configuration might better balance reconstruction quality and inference performance.

Keeping every other characteristic about the baseline model intact, models with 32, 16 and 8 filters per layer were trained for 100 epochs in order to evaluate how changing this parameter affects quality and performance. As expected, the relationship between the loss function and the number of filters is nonlinear. Figure 8.4 shows the trajectory of the loss function during training across the different models.

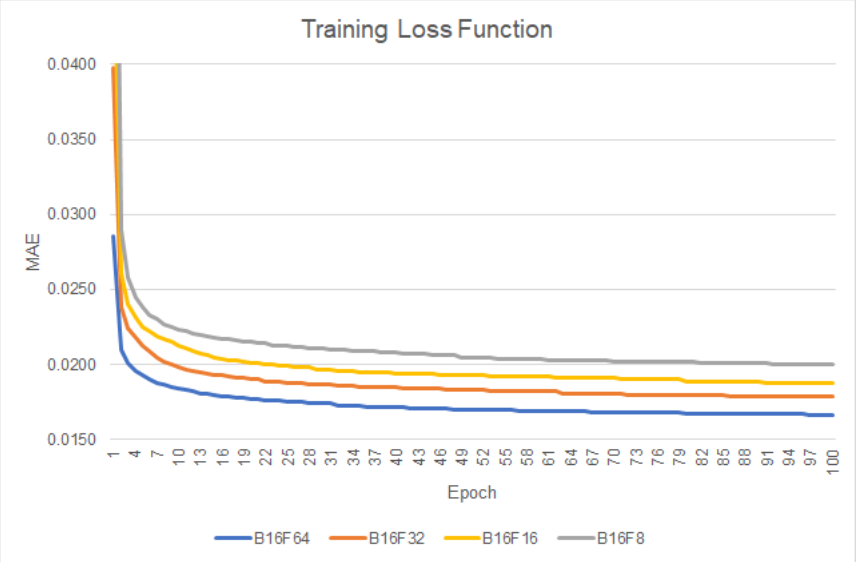


Figure 8.4: Loss comparison for number of filters.

While not ideal, it's possible to approximately determine how well each model managed to solve the problem by how much they could reduce the MAE loss function. Naturally, models with more parameters have higher learning capacity and therefore should be able to achieve better results unless there's a bottleneck in the architecture itself. Tables 8.4 and 8.5 summarise the differences between the models.

It's possible to notice that there's a quadratic relationship between the number of filters and the number of operations, this happens because increasing the number of filters also increases the number of feature maps between the convolutional layers, which means each filter gets bigger in the depth dimension.

Table 8.4: Experimenting with the number of filters.

Configuration	Weights	TFLOP	PSNR/SSIM	Inference Throughput
B16F64	1,337,091	2.47	31.48/0.9107	0.6 fps
B16F32	335,747	0.62	31.38/0.9099	1.7 fps
B16F16	84,675	0.16	31.17/0.9075	4.1 fps
B16F8	21,539	0.04	31.09/0.9068	5.6 fps

Table 8.5: Experimenting with the number of filters, relative comparison.

Change	Weights	TFLOP	Inference Throughput
F64->F32	25.11%	25.10%	283%
F32->F16	25.21%	25.49%	240%
F16->F8	25.43%	25.82%	138%

When it comes to training speed, it's possible to see that there's a nonlinear relationship between the number of operations in a forward pass and the actual time required to finish a single epoch. This can be explained by the fact that there's a temporal dependency between layers and the GPU also spends some time retrieving data from system memory. The first experiment, reducing the number of filters from 64 to 32, improved inference throughput by 183% while reducing the number of weights and operations by 75%. Reducing the number of filters down to 16 shows a similar performance increase, albeit slightly lower now at 140%, while maintaining the same size reduction of 75%.

However, unlike in the other 2 experiments, reducing the number of filters from 16 to 8 does not provide a meaningful increase in performance. This experiment shows that there are diminishing returns when reducing the number of filters and performance may not correlate well with it as it may be limited by other factors.

In any case, going from the B16F64 model to the B16F32 model results in the smallest quality degradation while still providing the biggest relative performance increase. This can also be seen in Figures 8.5 and 8.6.

Figure 8.5: Manga109 comparison for number of filters.

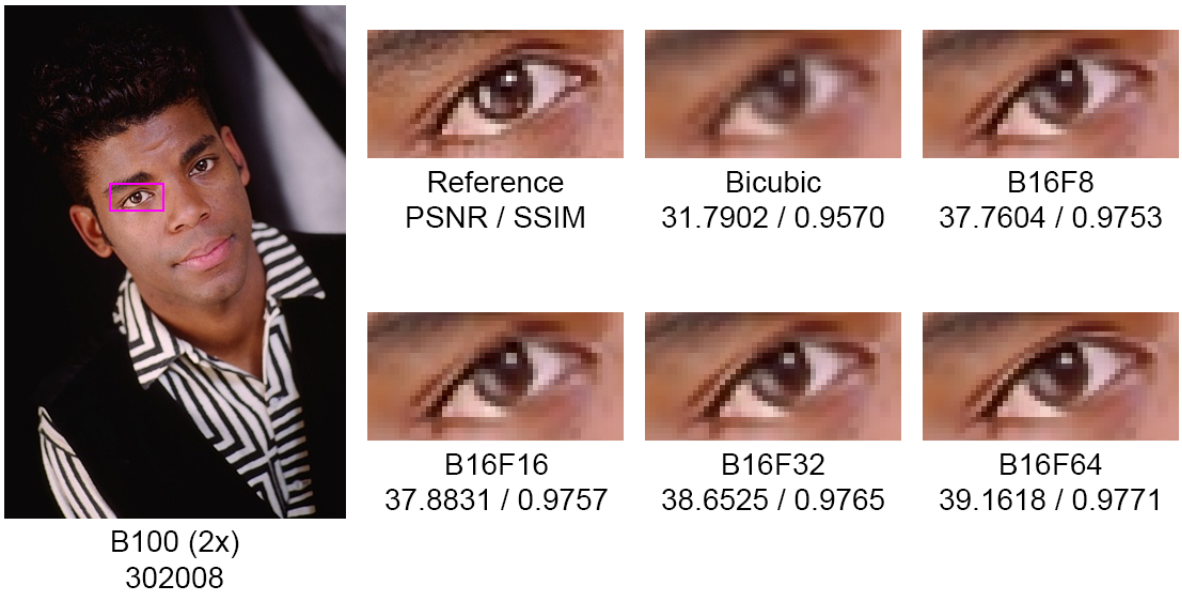


Figure 8.6: B100 comparison for number of filters.

8.5 Number of Residual Blocks

Fixing the number of filters at 8 per convolutional layer, in this section the number of residual blocks was reduced in order to evaluate how this architectural choice affects the model. Figure 8.7 shows the trajectory of the loss function during training across the different models. Tables 8.6 and 8.7 summarise the differences between the models. Figures 8.8 and 8.9 show a qualitative comparison between the models.

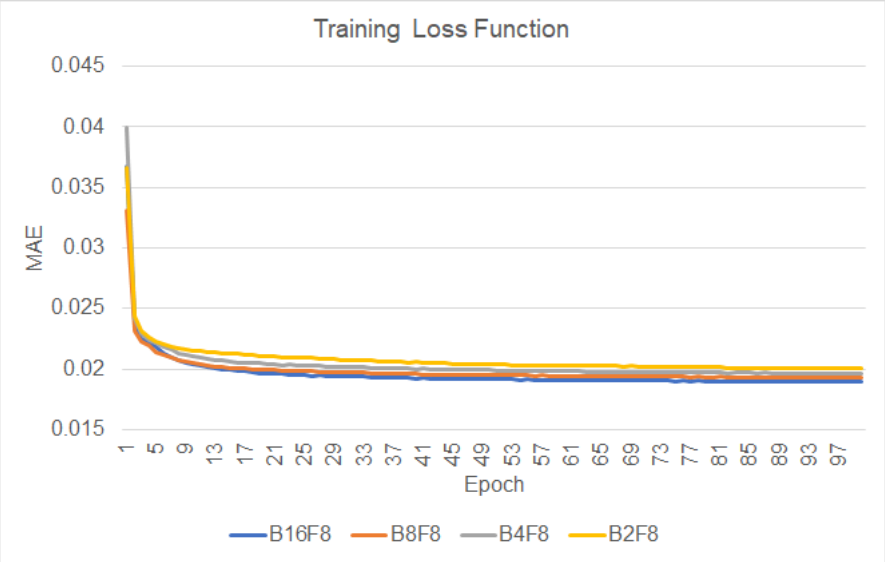


Figure 8.7: Loss comparison for number of residual blocks.

Table 8.6: Experimenting with the number of residual blocks.

Configuration	Weights	TFLOP	PSNR/SSIM	Inference Throughput
B16F8	21,539	0.041	31.09/0.9068	5.6 fps
B8F8	12,195	0.024	30.95/0.9047	8.8 fps
B4F8	7,523	0.015	30.73/0.9028	12.2 fps
B2F8	5,699	0.011	30.38/0.8992	17.4 fps

It's possible to see that reducing the number of residual blocks results in a reduction in the number of weights that is only almost linear. The only reason for it not to be perfectly linear is because the model has a few convolutional layers placed before and after the residual blocks, and these remain untouched.

It's important to remember that the number of filters per layer has a quadratic relationship with the number of weights in the model, but reducing the number of residual blocks also

Table 8.7: Experimenting with the number of residual blocks, relative comparison.

Change	Weights	TFLOP	Inference Throughput
B16->B8	57%	59%	157%
B8->B4	62%	63%	139%
B4->B2	76%	73%	143%

increases inference throughput considerably while not hurting model capacity as significantly. This experiment shows that, under constrained hardware resources, reducing the number of layers should take precedence over reducing the number of filters per layer.

When it comes to reconstruction quality, models with fewer weights and consequently lower learning capacity showed lower PSNR and SSIM scores on the test datasets. In any case, none of the configurations tested managed to achieve real-time performance on the Intel Iris Xe LP iGPU. This shows that further optimisations are needed.

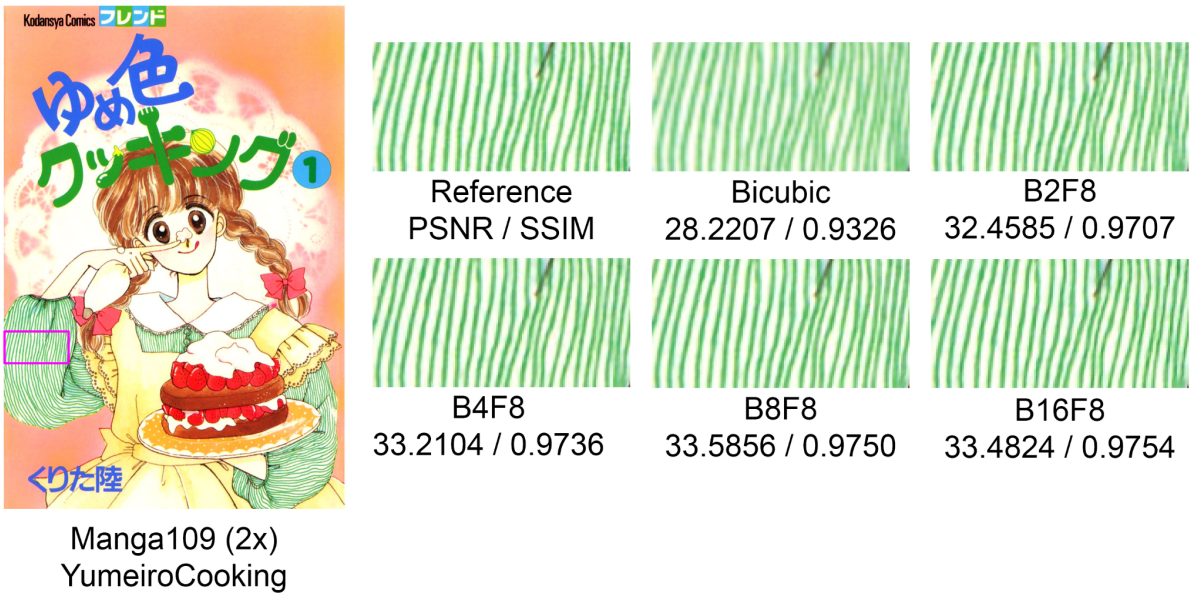


Figure 8.8: Manga109 comparison for number of residual blocks.

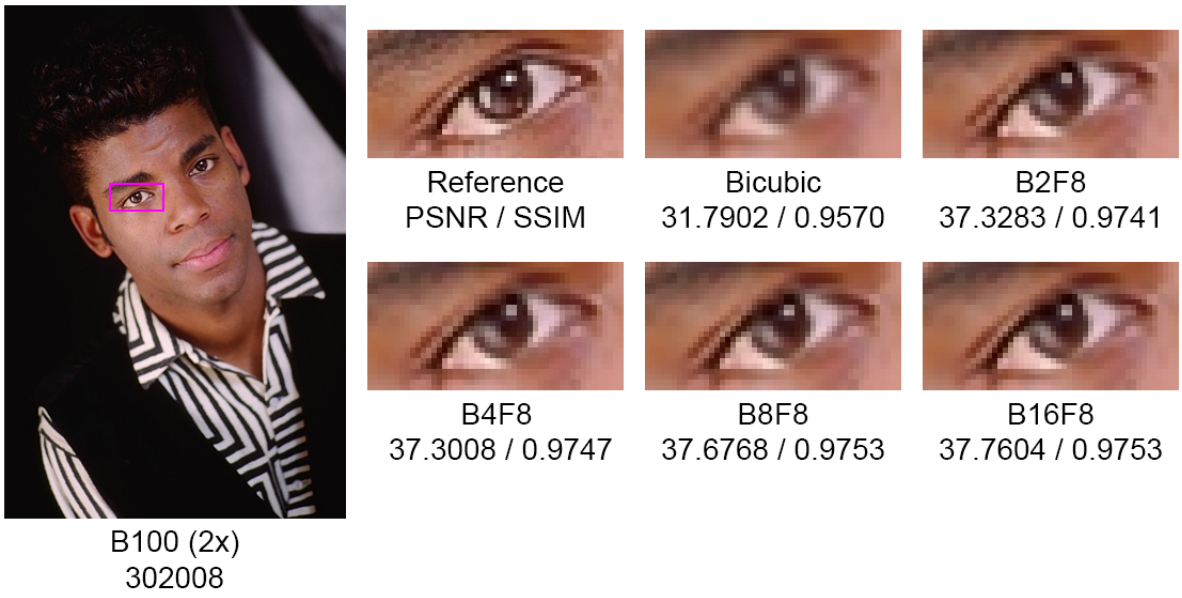


Figure 8.9: B100 comparison for number of residual blocks.

8.6 Finding the Bottlenecks

The previous sections explored the potential of optimising the network by either reducing the number of filters in each convolutional layer or the number of residual blocks in the model. In order to optimise the network further, it's important to understand how each layer affects execution time. Figure 8.10 shows how much time is taken by each layer from the B2F8 model.

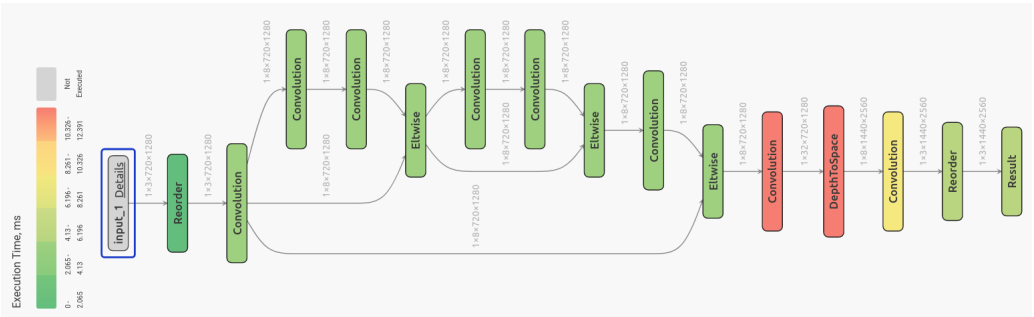


Figure 8.10: Execution time per layer with the B2F8 model.

It's easy to see that most of the execution time is spent on the upsampling module. This module first increases the number of LR feature maps from 8 to 32 using a convolutional layer, which are then rearranged as 8 HR feature maps by the depth to space layer. After that, these 8 HR feature maps go through a final convolutional layer with 3 filters to produce the RGB output. The upsampling module used in EDSR increases the number of feature maps by a

factor that's equal to the square of the scaling factor, which means 4 for 2x, 9 for 3x or 16 for 4x. This is done to preserve the original number of feature maps after the pixel shuffle operation.

Naturally, it's to be expected that the convolutional layer with 4 times as many filters, the one placed right before the depth to space layer, and the convolutional layer with feature maps that are 4x as big, the one right after the depth to space layer, would both take longer to compute. This experiment shows that simplifying the upsampling module might be a good way of improving performance.

8.7 YCbCr Model

As seen in the previous section, the layers that compose the upsampling module in the baseline model are the ones that take the most amount of time during inference. It's easy to see how this module can be simplified if it's simply changed in a way that allows the pixel shuffle operation to output the image directly. This would allow the second convolution to be removed while also reducing the number of filters in the first one.

Taking into consideration 4:2:0 chroma subsampling is virtually imperceptible to the HVS, which is why it is standard in most video compression schemes, using a CNN to supersample chromatic information seems extremely wasteful. It's easy to see how limiting the model to the luma channel would increase inference throughput, as we can reduce the size of each filter in the first convolutional layer and reduce the number of filters in the upsampling module. Figure 8.11 depicts the original upsampling module alongside its simplified versions. Tables 8.8 and 8.9 show the performance difference between them on the B2F8 model. Figure 8.12 depicts the difference between the RGB and the Luma models.

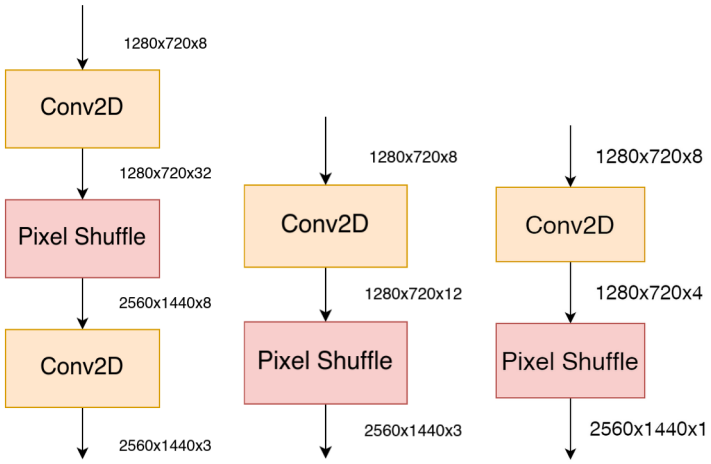


Figure 8.11: The changes done to the upsampling model. The version on the left depicts the original module. The version in the middle simplifies it by removing one of the convolutional layers and changing the number of filters in the remaining one for the resulting picture after the pixel shuffling operation to have 3 channels exactly. The version on the right simplifies this further by reducing the number of channels from 3 to 1.

Table 8.8: Comparison between the different upsampling modules.

Module	Weights	TFLOP	PSNR/SSIM	Inference Throughput
Original	5,187	11	30.38/0.8992	17.4 fps
Simplified	4,020	7.4	30.35/0.8986	27.3 fps
Luma	3,292	6.1	30.89/0.9076	35.7 fps

Table 8.9: Relative comparison between the different upsampling modules.

Change	Weights	TFLOP	Inference Throughput
Original->Simplified	78%	67%	157%
Simplified->Luma	82%	82%	131%

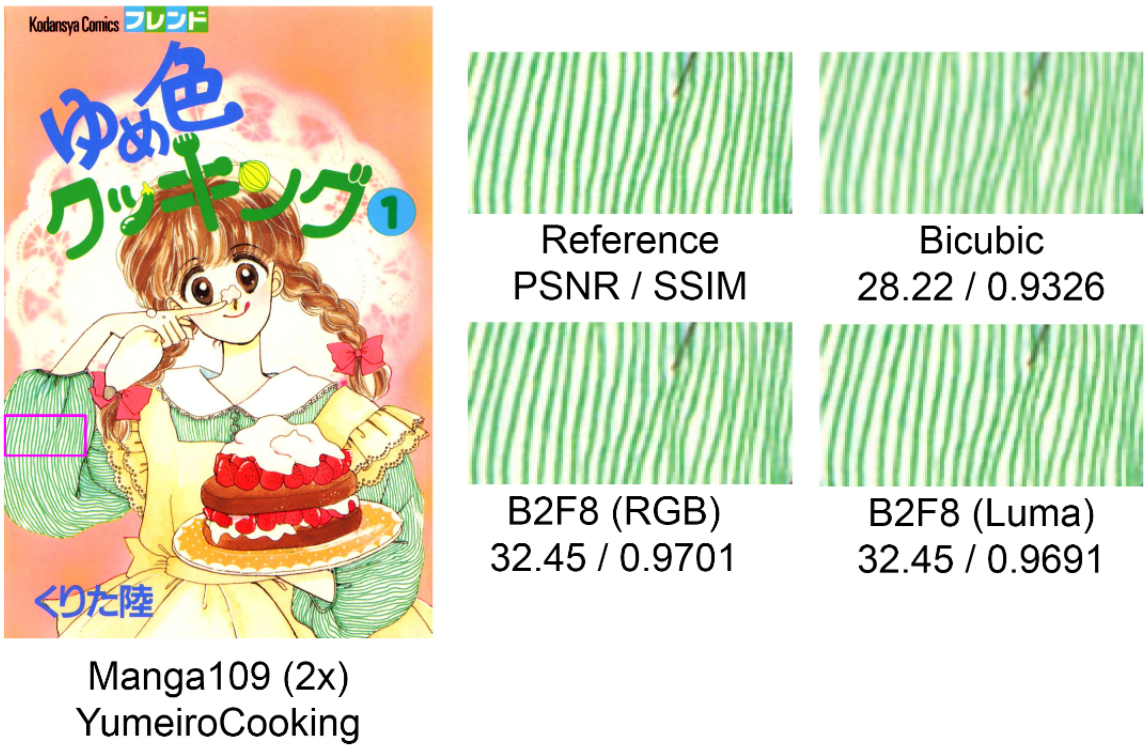


Figure 8.12: Manga109 comparison for different upsampling modules.

Figure 8.13 depicts the image processing pipeline with a luma-only super-resolution CNN, which leverages FIR filters (OpenCV's bicubic interpolation filter in this case) to upsample chroma.

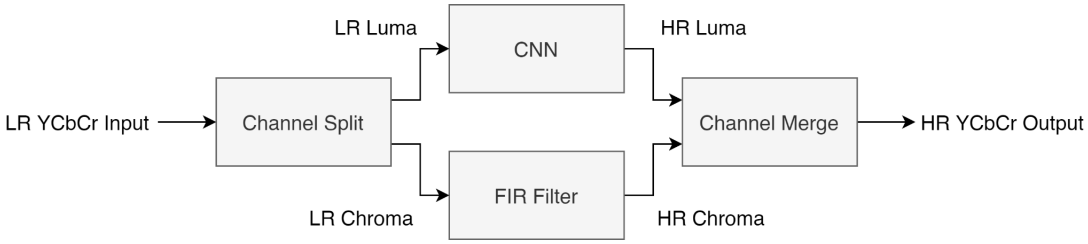


Figure 8.13: Luma-only CNN leveraging FIR filters for chroma.

Figure 8.14 shows how much time is taken by each layer in the luma-only model. It's easy to see how most of the execution time is now spent on the convolutional layers inside the residual blocks. This experiment shows that leaving chroma upsampling to the classic approaches such as bilinear or bicubic interpolation has a minimal drop in quality despite providing a significant performance increase on small networks. Naturally, as the size of the network increases, the contribution of the upsampling module to execution time decreases. However, for small configurations such as B2F8, the quality downgrade can be easily justified

since it provides a performance uplift of approximately 31%. This is very attractive for the real-time inference objective and from this point on all experiments are conducted on luma-only models.

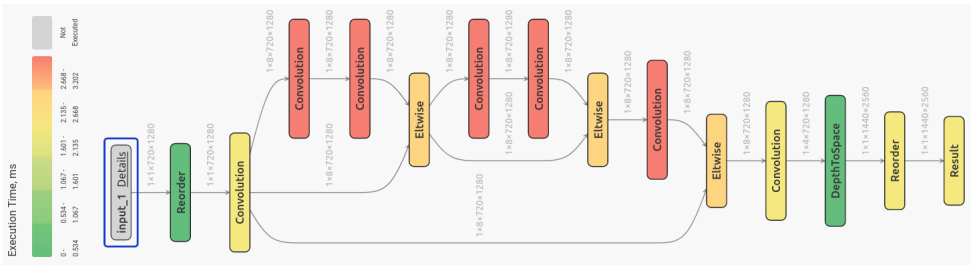


Figure 8.14: Execution time per layer with simplifying upsampling module and the luma-only model.

8.8 Attention Mechanisms

In order to evaluate how well attention mechanisms work in a small model aimed at real time inference, the residual blocks in the B2F8 model were replaced by residual channel attention and residual spatial attention blocks.

As seen before, the channel attention mechanism employed by RCAN is capable of adaptively rescaling channel-wise features by considering interdependencies among channels. The following image depicts the RCAB.

While channel attention is experimentally proven to improve reconstruction quality in deep super-resolution networks, the performance overhead might make it unfeasible for real-time inference.

As shown by CSFM, it's possible to turn the channel attention mechanism found in RCAN into a spatial attention mechanism if the global average pooling layer is removed. Table 8.10 summarises the differences between the models.

Table 8.10: Comparison between the different attention mechanisms.

Configuration	Weights	TFLOP	PSNR/SSIM	Inference Throughput
B2F8	3,292	6.1	30.89/0.9076	35.7 fps
RCAB2F8	3,444	6.4	30.77/0.9067	7.4 fps
RSAB2F8	3,444	6.4	31.13/0.9092	24.2 fps

As it's possible to see in the table above, reconstruction quality took a small penalty with channel attention, this might indicate that the mechanism works better with more channels or when the network is deeper. Inference throughput with channel attention is significantly worse despite the similar number of weights. As shown in Figure 8.15, most of this performance overhead comes from the global average pooling layers.

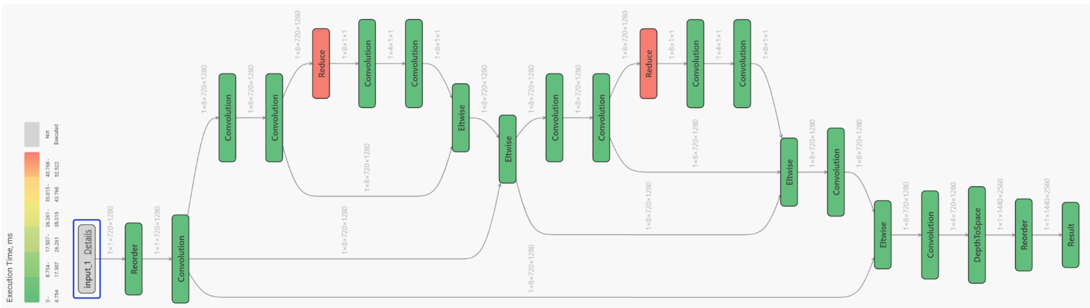


Figure 8.15: Execution time per layer with channel attention.

In any case, the performance penalty created by the global average pooling layer makes it difficult to justify employing channel attention mechanisms when the objective is to perform fast inference. The original RCAN paper reported a PSNR improvement of 0.3 dB in Set5 with channel attention, which was not reproduced with 2 RCABs and 8 filters per convolutional layer only.

Spatial attention also results in an inference performance penalty, but the difference is nowhere as severe as it was with channel attention. Reconstruction quality actually improved when compared to the B2F8 model, which might indicate the mechanism adapts well to smaller models with fewer blocks and filters. Figure 8.16 depicts the time spent on each layer and as expected, removing the global average pooling layers greatly improved inference throughput.

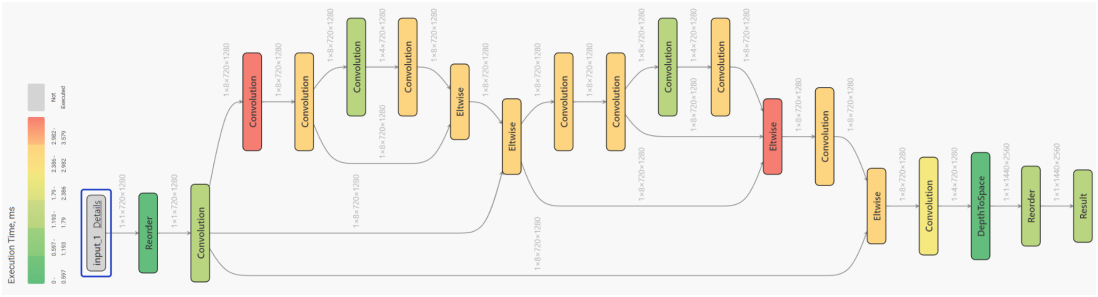


Figure 8.16: Execution time per layer with spatial attention.

8.9 Skip Connections

8.9.1 Element-Wise Addition

The skip connections found in many state of the art convolutional neural networks were introduced to help train very deep networks. However, the experiments explored in this document so far have shown that running deep networks in real time on integrated GPUs is still unfeasible. For this reason, it's important to ask whether shallow networks also benefit from skip connections. The hypothesis is that removing these connections would most likely improve inference throughput since the number of operations would be reduced, but training these networks might become harder and reconstruction quality may degrade.

In order to test this hypothesis, the skip connections in the B2F8 model were removed and these networks were also trained over 100 epochs. Figure 8.17 depict the loss function during training for networks with different combinations of skip connections. It's easy to see how the long and short skip connections in the B2F8 model helped the network reduce the loss function further and in a more stable way. Table 8.11 summarises the differences between the models.

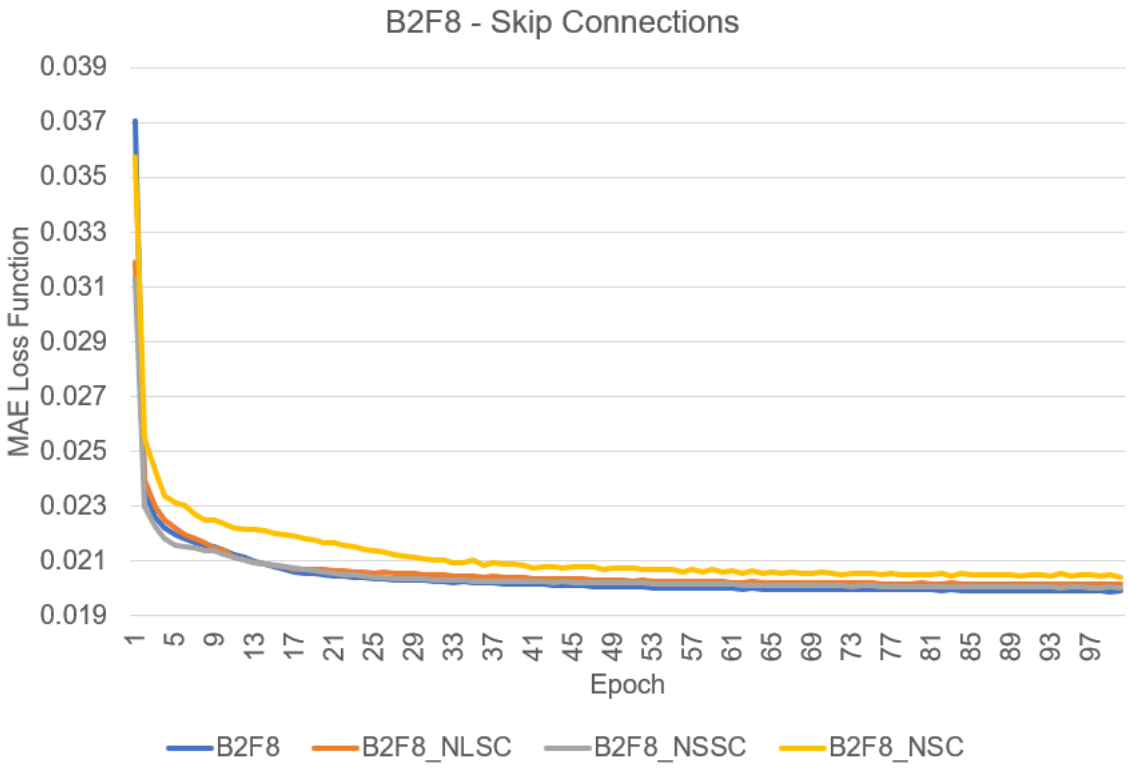


Figure 8.17: Loss comparison for different skip connections. NLSC stands for "no long-skip connection", NSSC stands for "no short-skip connections" and NSC stands for "no skip connections".

Table 8.11: Comparison between the different skip connections. LSC stands for long-skip connection and SSC stands for short-skip connection.

LSC	SSC	PSNR/SSIM	Inference Throughput
No	No	30.75/0.9058	47.3 fps
No	Yes	30.82/0.9067	38.2 fps
Yes	No	30.79/0.9062	43.5 fps
Yes	Yes	30.89/0.9076	35.7 fps

It's important to remember that the original EDSR model does not include activation functions in the convolutional layers placed before the skip connections, which is done to give the model more flexibility when adding the feature maps, as leaving the ReLU activations would turn any negative residuals into zero.

With only 2 residual blocks though, since EDSR's residual blocks are still being used, the B2F8 model under test in this section only has 2 non-linear layers, and since the number of filters in each layer is also low, this might limit how much the model can learn, restricting it to mostly linear solutions. In order to evaluate this hypothesis, these models were retrained with activation functions added to all layers that aren't used in skip connections. Table 8.12 summarises the findings.

Table 8.12: Comparison between the different skip connections with added activations. LSC stands for long-skip connection and SSC stands for short-skip connection.

LSC	SSC	PSNR/SSIM	Inference Throughput
No	No	30.75/0.9058	47.3 fps
No	Yes	30.97/0.9079	38.2 fps
Yes	No	31.02/0.9080	43.5 fps
Yes	Yes	30.89/0.9076	35.7 fps

As expected, adding activation functions to all the convolutional layers in the residual blocks improves reconstruction quality. The model with a long skip connection achieved the highest PSNR score between the configurations tested. This model keeps the external convolutional layers as they were beforehand, without activation functions, but adds ReLU activations to the internal convolutional layers that were previously placed at the end of the residual blocks. Since the model only has a single element-wise addition layer, it is also faster to run than the alternatives with short skip connections.

8.9.2 Channel Concatenation

Channel concatenation is a different way of forwarding abundant low frequency information into deeper layers inside a CNN, potentially increasing reconstruction quality. The obvious downside is that concatenating channels increases the size of the feature maps in the depth dimension, which increases the size of the convolutional filters, ultimately reducing inference throughput.

The technique has been thoroughly explored in recent works such as SRResNet. The advantage over element wise addition as a skip connection is that channel concatenation allows the network to learn how to combine the information.

To limit the growth of depth as the feature maps get concatenated together, the technique is usually used alongside element wise addition in the most recent works such as RDN, DLRN and ESRGAN. Figure 8.18 depicts channel concatenation with and without residual connections.

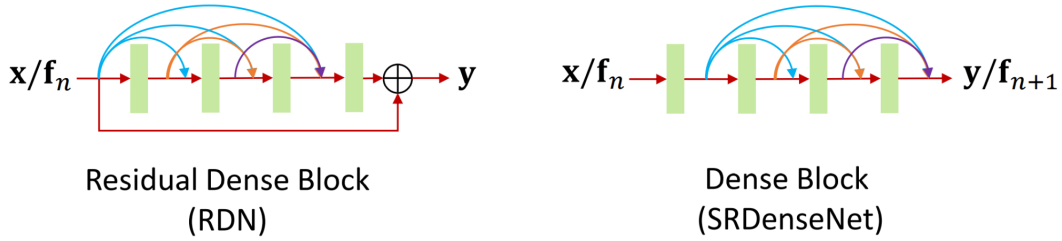


Figure 8.18: The channel concatenation technique. Adapted from: (ANWAR; KHAN; BARNES, 2020).

In order to evaluate channel concatenation as a skip connection alternative, keeping all other aspects about the network unchanged, the residual blocks were replaced by densely connected convolutional layers with ReLU activations. The external layers are generally used to aid element-wise addition by standardising the number of channels before these connections, but they were also kept the same across the comparisons. Table 8.13 and Figure 8.19 summarise the differences between the models.

Table 8.13: Comparison between residual vs dense connections.

Filters	Skip Connections	Weights	PSNR/SSIM	Inference Throughput
8	None	3,292	30.75/0.9058	47.3 fps
8	Residual Short	3,292	30.97/0.9079	38.2 fps
8	Residual Long	3,292	31.02/0.9080	43.5 fps
8	Residual	3,292	30.89/0.9076	35.7 fps
8	Dense	9,052	31.29/0.9110	30.1 fps
8	Residual Dense	9,052	31.26/0.9103	26.1 fps

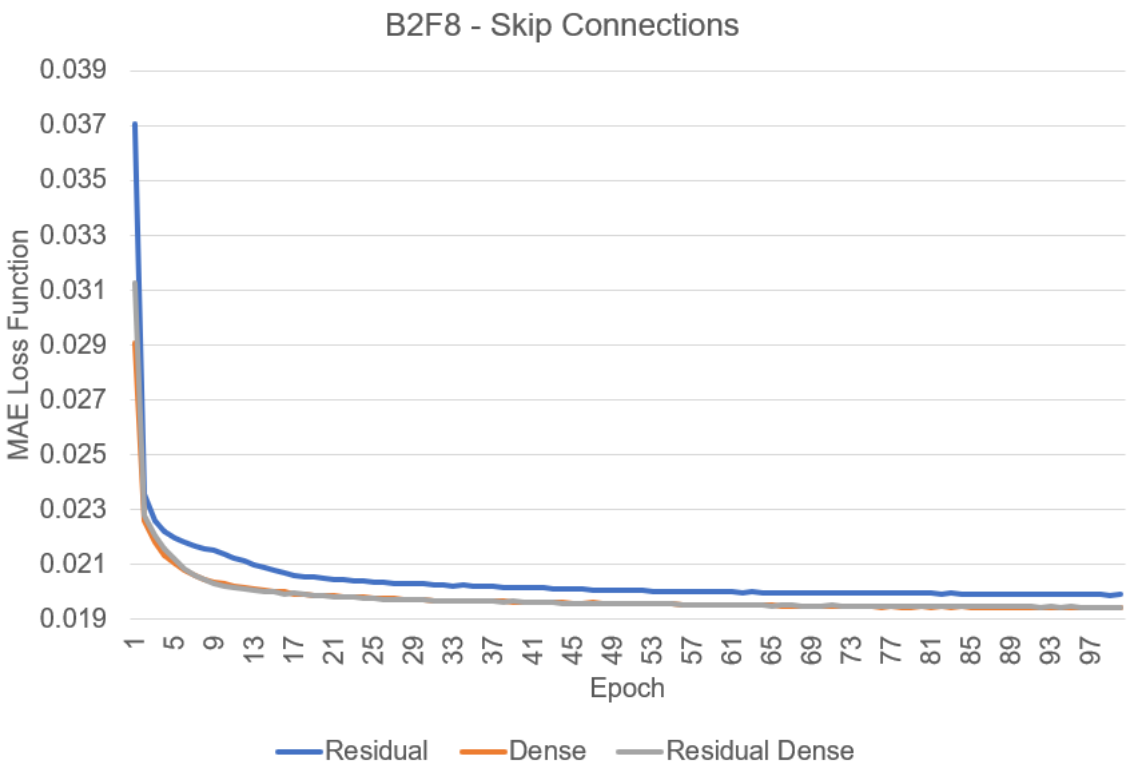


Figure 8.19: Loss comparison for residual vs dense connections.

While densely connecting the convolutional layers increases reconstruction quality significantly, it also decreases inference throughput by roughly 30%. The performance decrease comes from the fact that concatenation increases the size of the feature maps in the depth dimension, consequently increasing the size of the convolutional filters and ultimately increasing the number of operations in the model. Figure 8.20 shows the time taken in each layer from the residual dense model.

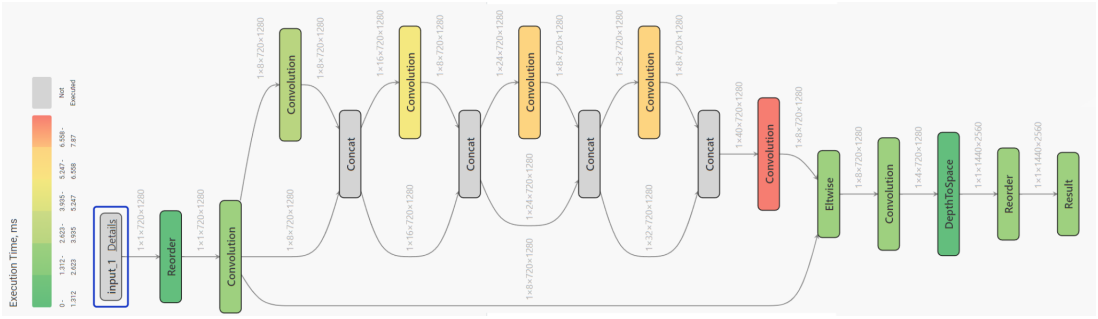


Figure 8.20: Execution time per layer with dense connections.

It's worth mentioning that the reconstruction quality improvement seen with channel concatenation comes partially from the increased number of weights in the model. However, it's

not clear from these tests whether a network with a comparable number of weights but a different architecture wouldn't be able to achieve higher performance. To verify this possibility, a residual network with 13 filters per layer, to approximately match the number of weights, and an element-wise addition layer as a long skip connection was also trained. Table 8.14 compares it against the residual dense model with 8 filters.

Table 8.14: Comparison between residual vs dense connections at a similar learning capacity.

Filters	Skip Connections	Weights	PSNR/SSIM	Inference Throughput
8	Residual Dense	9,052	31.26/0.9103	26.1 fps
13	Residual Long	8,272	31.35/0.9118	33.7 fps

Despite having slightly fewer weights, the model with 13 filters per layer and a simple long skip connection achieved not only higher reconstruction quality in both metrics, but also higher inference throughput.

8.10 Activation Functions

In neural networks, activation functions are used to allow the network to manipulate the data in non-linear ways. This makes it possible for these networks to learn how to solve complex problems that would've been impossible with linear combinations of the data.

The Rectified Linear Unit (FUKUSHIMA, 1969) has eclipsed older activation functions such as the hyperbolic tangent and the sigmoid in usage due to its superior runtime performance and because it allows networks to converge more quickly (LECUN et al., 2012).

However, ReLUs can reach states in which they essentially become inactive regardless of the input being given to it, effectively decreasing the network's learning capacity. This happens because variations in negative inputs do not result in any gradient, which may cause the neuron to get perpetually stuck in an inactive state in which the output is always zero. This, however, gives the model some sparsity and the network might converge faster.

In an attempt to fix the dying ReLU issue, several variations of the activation function have been proposed. Usually, the goal is preserving linearity in the positive half while suppressing negative activations. LeakyReLU prevents the gradient from being zero with negative

inputs, which can prevent neurons from becoming perpetually inactive. However, LeakyReLU activation does not bound negative inputs and these can have an undesirable impact on the following layers. Methods like GELU (HENDRYCKS; GIMPEL, 2023) and Swish (RAMACHANDRAN; ZOPH; LE, 2017) attempt to provide well defined gradients for negative inputs to stop neurons dying while also limiting how much negative outputs are allowed to have an effect in the following layers.

A residual model with 4 internal convolutional layers with 16 filters each and a long skip connection (C4F16) was used as the baseline to test different activation functions. The size of the model is being increased in this section from C4F13 to C4F16 because this increase still keeps it within our performance budget, and as seen before increasing the number of filters per layer offers a better quality/performance compromise than increasing the number of layers. This is the final architecture and all other experiments will use it. Figure 8.21 shows the loss trajectory on the C4F16 model with various activation functions. Table 8.15 summarises the differences between the models.

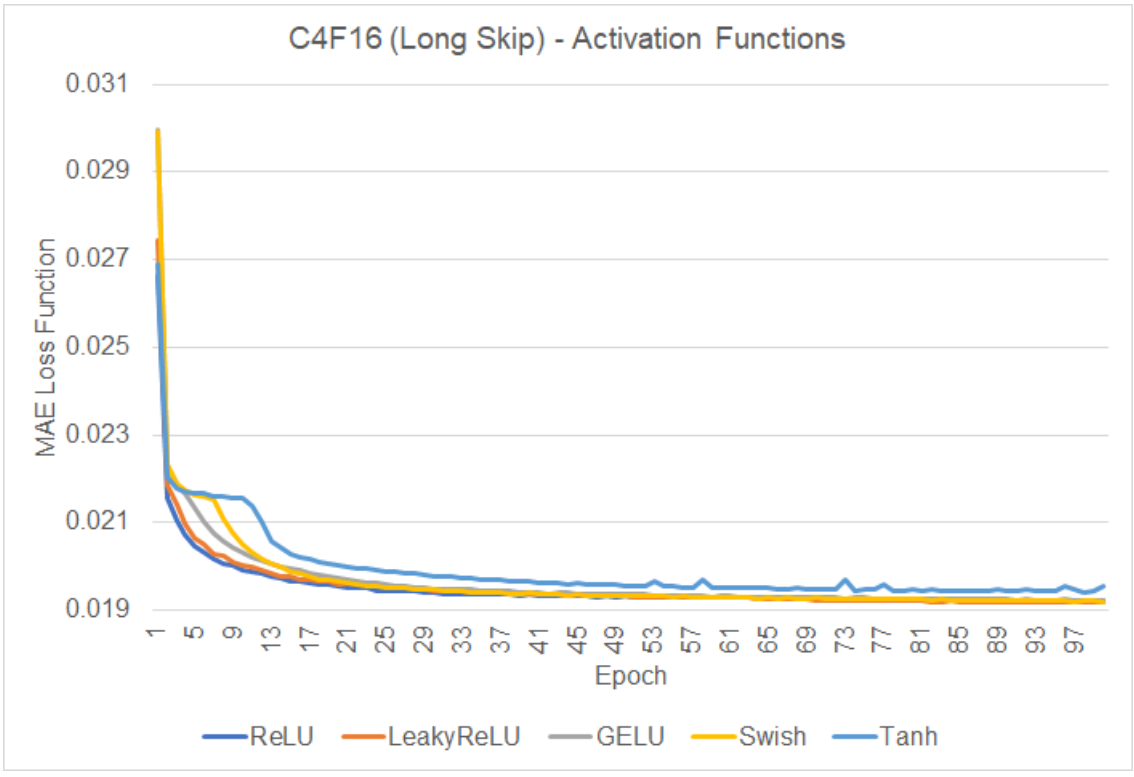


Figure 8.21: Loss comparison for different activation functions.

Table 8.15: Comparison between different activation functions.

Activation	PSNR/SSIM	Inference Throughput
Tanh	31.37/0.9114	28.8 fps
ReLU	31.38/0.9119	28.8 fps
LeakyReLU	31.38/0.9119	28.8 fps
GELU	31.44/0.9121	13.4 fps
Swish	31.43/0.9124	28.8 fps

When it comes to the loss function, it's possible to see that all ReLU variations are extremely close to each other, with the hyperbolic tangent lagging behind. GELU achieved the best PSNR scores in the test datasets, while Swish beat it slightly in SSIM. The GELU is the only activation function that's significantly slower than the other options however, which makes it less desirable in this research since the objective is real-time inference.

The differences seen here are usually small and definitely not substantial enough to reliably discard run to run variations. It seems reasonable to stick with the options that are more likely to be well optimised by the various machine learning frameworks and libraries. The ReLU activation function will be used in the following sections.

8.11 INT8 Quantisation

Post-training quantization is a technique that can be used to reduce model size while also improving inference latency or throughput, but with a small penalty in model accuracy. For INT8 quantization, the minimum and maximum values of all floating point tensors in the model are needed. Unlike constant tensors such as weights and biases, variable tensors such as inputs, outputs and internal feature maps can not be calibrated without a few inference cycles.

All ML inference engines, toolkits and libraries focused on fast inference offer model optimisation routines including model quantisation. On GPUs, FP16 is usually preferred over INT8 because the hardware is generally designed to perform tensor operations in half-precision

floating point and, typically, there's no significant performance improvement going down to INT8. However, INT8 quantisation is usually preferred on CPUs, FPGAs and TPUs. In order to evaluate INT8 quantisation, a few models already discussed in the previous sections were quantised and benchmarked. Table 8.16 summarises the findings.

Table 8.16: INT8 quantisation summary on the Intel Iris Xe LP GPU.

Model	FP16	INT8
B2F8 (RGB)	17.4 fps	19.7 fps
B2F8 (Luma)	35.7 fps	21.1 fps
B2F8 (NSC)	47.3 fps	22.1 fps

The table shows that bigger models benefit from INT8 quantisation better than smaller ones, and this can be easily explained by the extra quantisation layers required before and after convolutions. Since the operations in lower precision can usually be performed more quickly, when you have many operations to be performed this will outweigh the overhead created by the quantisation steps and therefore result in a performance uplift. With small models, however, the time saved by performing the operations in INT8 does not outweigh the time taken to quantise the tensors and the result is a performance reduction on the Intel Iris Xe LP GPU.

8.12 The Final Model

Starting from the EDSR baseline with 16 residual blocks and 64 filters per convolutional layer, the first experiment consisted of evaluating the usage of depth-wise separable convolutions but the operation isn't well optimised and performance ended up worse despite a significant reduction in the number of operations in the model.

Continuing from the same EDSR baseline, the following experiment consisted of reducing the number of filters and observing the outcome in terms of reconstruction quality and inference throughput. The ideal number of filters that results in a good quality and performance compromise is 32 per convolutional layer, but the Iris Xe LP integrated GPU target does not have enough resources to run a network this big in real time.

Next, keeping the number of filters per convolutional layer locked at 8, the next experiment was aimed at evaluating what happens when the number of residual blocks is reduced. Empirically, reducing the number of blocks should be preferred over reducing the number of filters when focusing on reconstruction quality, because the latter has a quadratic relationship with the number of weights. However, it was found that reducing the number of blocks reduces the number of temporal dependencies in the model giving a somewhat consistent performance improvement. The ideal number of blocks before the quality penalties become significant would be 4, but the Iris Xe LP integrated GPU target isn't capable of running a B4F8 model in real time.

The fourth experiment consisted of finding the bottlenecks and optimising the model to make it faster. The upsampling module was simplified with the removal of a convolutional layer, and the model was changed to receive and output the luma channel only, leaving colour upsampling to the conventional methods with FIR filters.

The fifth experiment tested 2 different attention mechanisms and whether or not they would be able to result in an improvement when implemented within a small model. The variant with channel attention was surprisingly worse than the baseline, but the model with spatial attention managed to improve reconstruction quality slightly. In any case, the throughput penalty created by attention mechanisms makes them problematic for real time inference.

Following that, the sixth experiment evaluated several skip connection mechanisms. It was empirically found that adding skip connections not only accelerates model convergence but it also allows the weights in the model to be used more effectively, increasing PSNR and SSIM scores on the test datasets. Afterwards, the sixth experiment consisted of testing different activation functions, but no function tested displayed an expressive improvement over the ReLU baseline.

The seventh experiment evaluated various activation functions. While they were all relatively close in quality, the GELU showed significantly lower throughput.

Finally, INT8 quantisation was also tested. The experimental results on the Intel Iris Xe LP integrated GPU show that INT8 quantisation provides a throughput improvement when applied to bigger models, but the extra overhead has a negative impact on small models. GPUs are usually optimised for FP32 and FP16 operations, so these results are not unexpected.

With all that said, a residual luma-only C4F16 model (4 internal convolutional layers with 16 filters each and only a single long-skip connection) was chosen as the final model based on

the findings from all architectural experiments. This architecture is illustrated in Figure 8.22. Table 8.17 summarises model performance across the different test datasets.

Figure 8.22: The final C4F16 architecture.

Table 8.17: The final C4F16 model evaluated on all test datasets individually.

Dataset	MAE	PSNR	SSIM	MSSSIM
Set5	0.0111	35.1235	0.9488	0.9975
Set14	0.0200	30.2316	0.8893	0.9922
B100	0.0219	29.4348	0.8709	0.9935
Urban100	0.0244	28.2849	0.8944	0.9948
Manga109	0.0115	34.1319	0.9652	0.9982
All	0.0178	31.4413	0.9137	0.9953

Chapter 9

Training Experiments

9.1 Loss Functions

9.1.1 SSIM Loss

As explained before, pixel specific metrics such as MSE, MAE and PSNR consider all numerical distortions in the image as equal regardless of how the HVS perceives them. This has led to the development of several metrics, such as SSIM, MSSSIM, VIF and perceptual metrics such as BRISQUE, NIQE and LPIPS. Many papers such as (ZHAO et al., 2018) studied the differences created by using different loss functions in image restoration and super resolution neural networks. The paper describes that MAE usually beats MSE due to better convergence characteristics, but mixing them can help avoid getting stuck in suboptimal local minima. The authors have also explored the use of SSIM and MSSSIM as loss functions, which showed positive results when used alongside other metrics.

In an attempt to see whether tweaking the loss function can improve the results, the final network was retrained with a SSIM-based loss function. As recommended by the paper, SSIM was used alongside MAE. The following expression depicts the loss function.

$$L_{MAE+DSSIM} = MAE(Y_{Pred}, Y_{Ref}) + (1 - SSIM(Y_{Pred}, Y_{Ref}))$$

Table 9.1 depicts the performance of the model trained for 100 epochs with the new loss function on all 5 test datasets.

Table 9.1: The final model evaluated with the SSIM-based loss function.

Dataset	MAE	PSNR	SSIM	MSSSIM
Set5	0.0112	34.9087	0.9492	0.9970
Set14	0.0206	30.0181	0.8923	0.9902
B100	0.0227	29.1587	0.8770	0.9906
Urban100	0.0249	28.1627	0.8992	0.9933
Manga109	0.0113	34.0654	0.9657	0.9975
All	0.0181	31.2627	0.9167	0.9937

It's possible to see that the SSIM scores improved across the board, which was expected. All other metrics, including MSSSIM, got worse however. This might indicate that training the model with a SSIM based loss function swayed it into focusing into creating sharp edges while overlooking side effects such as ringing, aliasing and blocking. Figures 9.1 and 9.2 show a qualitative comparison between the loss functions.

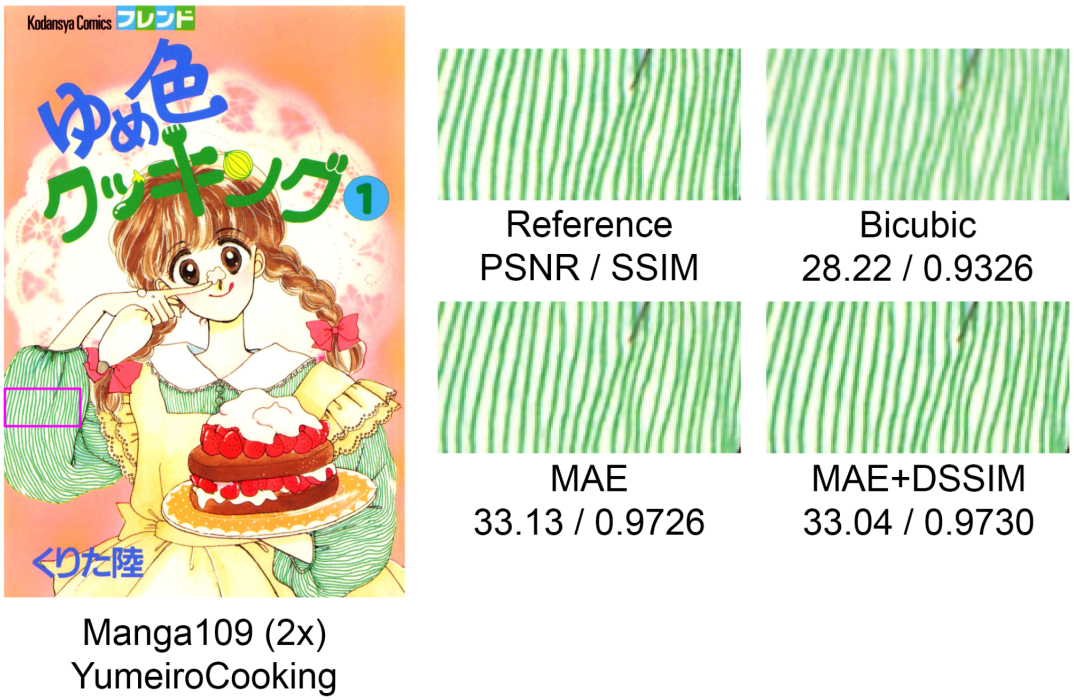


Figure 9.1: Manga109 comparison for the SSIM-based loss function.

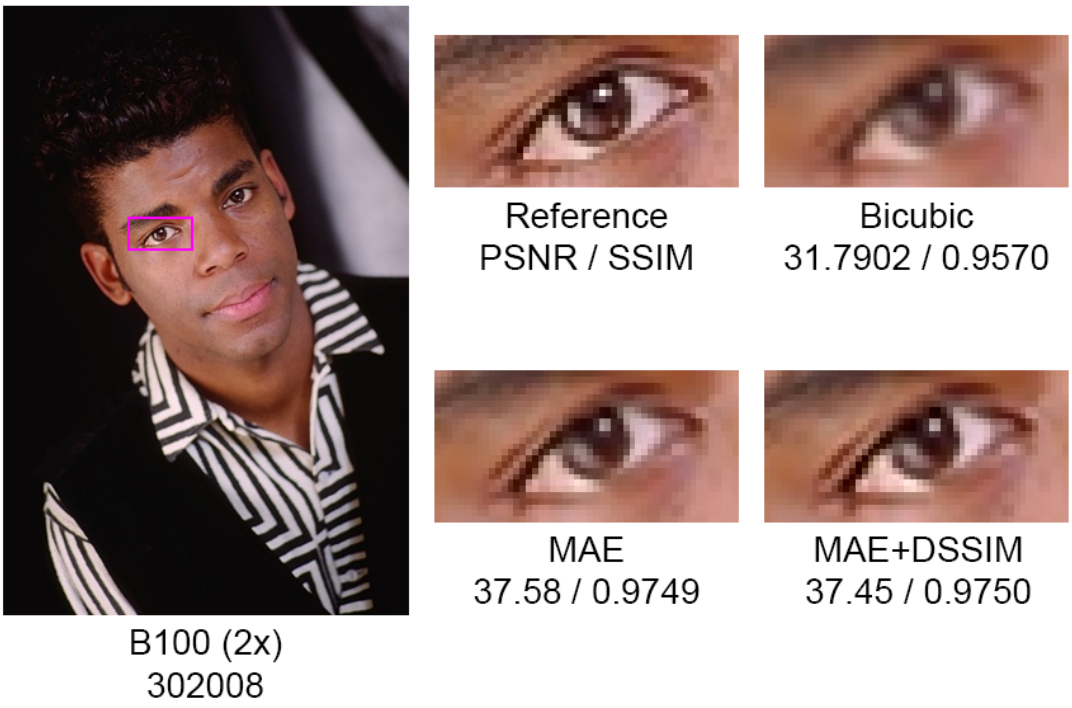


Figure 9.2: B100 comparison for the SSIM-based loss function.

9.1.2 VGG Loss

Deep feature extraction has emerged as a viable perceptual image quality metric (ZHANG, R. et al., 2018) (JOHNSON; ALAHI; FEI-FEI, 2016). Perceptual metrics aim to better correlate with the HVS, and perceptual metrics aim to better restore fine details and texture, which is usually missing in distortion-based networks.

SRGAN introduced adversarial training into the super-resolution problem, achieving photo-realistic pictures capable of deceiving a discriminator network. Initially, MSE was paired alongside the adversarial loss to prevent the network from generating images that diverge too much from the ground truth, however, their paper also showed that their result greatly improved when the distortion based metric was replaced by VGG deep feature extraction.

Inspired by the SRGAN procedure, and adopting a perceptual metric defined by the mean absolute error between the VGG features extracted from the 4th convolutional layer in the 5th block, the final model was retrained with a perceptual component in its loss function. The following expression depicts the loss function.

$$L_{MAE+VGG} = MAE(Y_{Pred}, Y_{Ref}) + \alpha \cdot MAE(VGG_{54}(Y_{Pred}), VGG_{54}(Y_{Ref}))$$

The alpha coefficient is used to control the influence of the perceptual metric in the loss function and it was set to 0.05 to put both metrics in the same order of magnitude.

Perceptual image quality metrics are required to evaluate whether training the model with a VGG-based loss function improves perceptual image quality. Figures 9.3 and 9.4 show the difference between the models trained with different loss functions. The PSNR and SSIM metrics were replaced by BRISQUE and LPIPS to better evaluate whether the VGG loss is increasing perceptual scores as intended, as the perception metrics are expected to get worse.

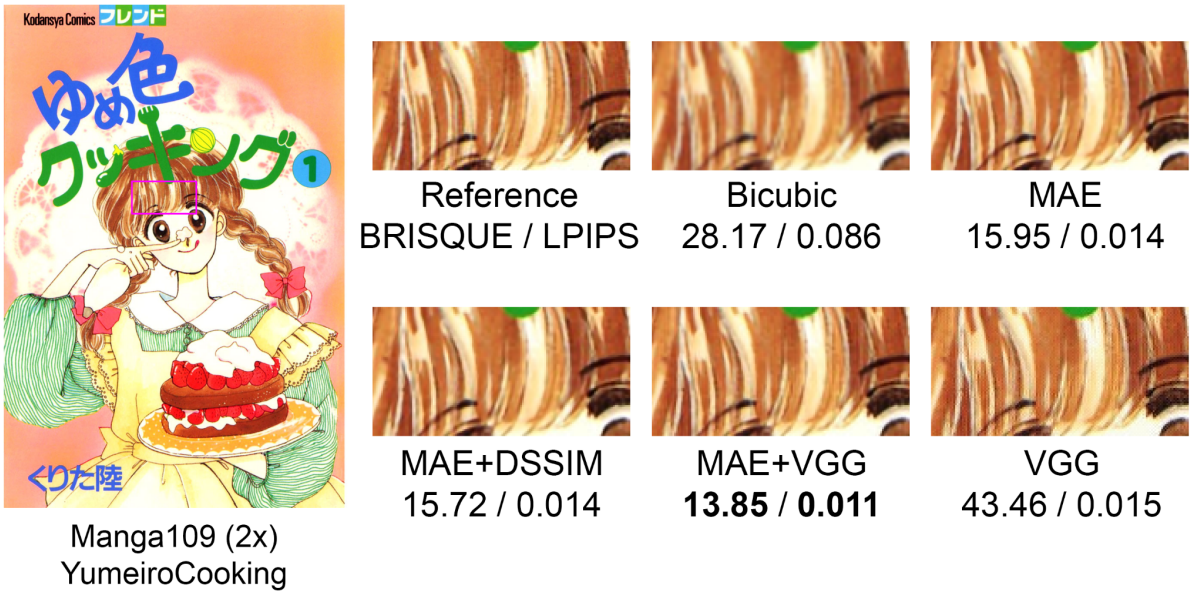


Figure 9.3: Manga109 comparison for the VGG-based loss function.

As expected, both metrics seem to agree that pairing VGG feature extraction with MAE results in images that look more natural. Adding a perceptual component to the loss functions helps the network learn how to create texture in high-frequency regions that generally look too soft when using distortion-based loss functions. However, using the VGG-based loss function by itself makes training extremely unstable and often unreliable, creating images full of checkerboard patterns and other artefacts such as ringing and brightness deviations. This phenomenon can be seen in figure 9.5.

9.2 Adversarial Training

Generative Adversarial Networks were originally meant to fabricate real looking images from pure noise in a latent space. Structurally, GANs generally have 2 internal networks aimed at solving different problems, the generator and the discriminator. The generator is responsible

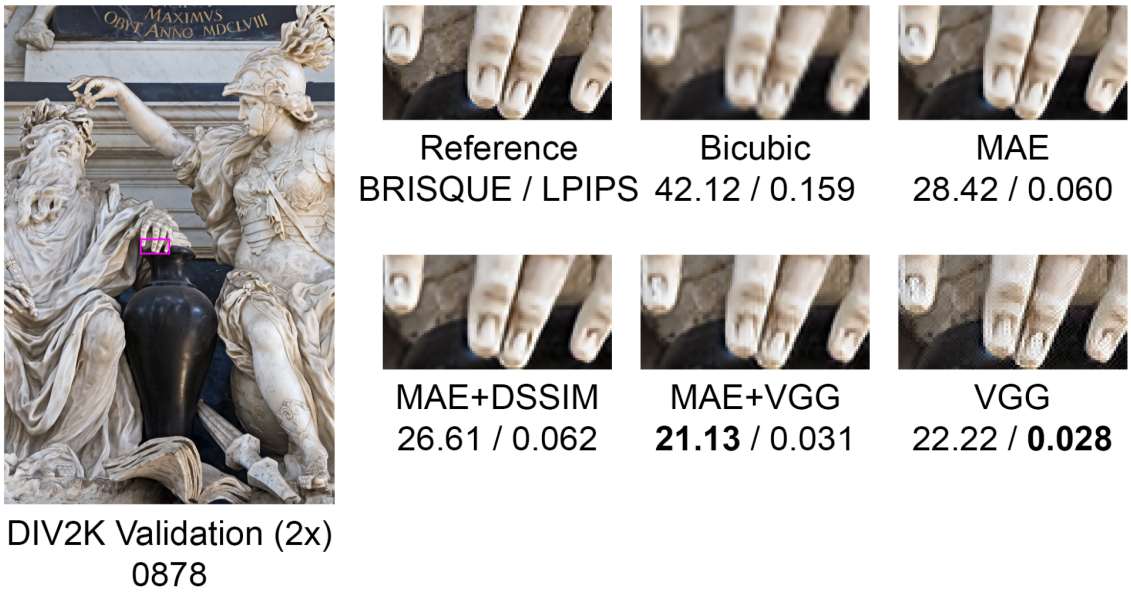


Figure 9.4: DIV2K comparison for the VGG-based loss function.

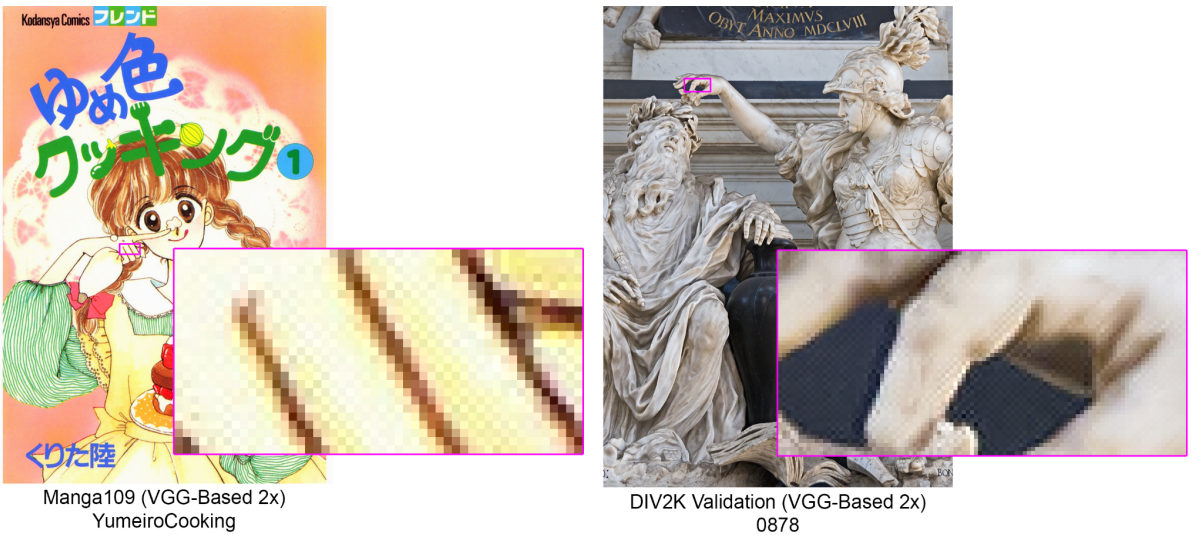


Figure 9.5: The checkerboard patterns created by the VGG loss.

for generating fake images from a set of inputs, while the discriminator is responsible for discriminating the images between fake and real. The discriminator is trained to minimise the errors in its labelling while the generator is trained to fool the discriminator into labelling fake images as real. Several GAN variations have emerged in the last few years, usually with the objective of making the training procedure more stable. A popular modification is replacing the discriminator with a critic that simply gives quality scores to the images. The key difference between the discriminator and the critic is that the latter usually can't saturate, it converges to a linear function providing reliable gradients, which makes it easier to train the generator.

Adversarial training can not be used by itself in the super-resolution problem because there's nothing preventing the generator from creating images that look wildly different than their references. However, the technique can be used alongside other metrics to incentivise the generator into learning how to create texture and fine-detail, which is exactly what is usually missing when images get upsampled.

Super-resolution GANs are usually trained in 2 distinct steps. In the first step, the generator is individually trained using a distortion-based loss function, the objective is increasing the PSNR scores of the generated images up to where it would be with normal distortion-based training. In the second phase, the discriminator and the image classification networks are added to the system to aid training the generator with distortion-based, perception-based and adversarial-based losses.

While the discriminators and hyperparameters described by SRGAN and ESRGAN can be used for reference, their generators are orders of magnitude bigger and certainly can't run in real time. In any case, using a discriminator designed to be paired with a much more powerful generator causes adversarial training to be drastically unbalanced. Attempts were made to follow their respective strategies while only modifying the generator, but the models failed to converge, creating results visibly worse than when training with distortion losses only.

With that said, reducing the size of the discriminator to roughly match the number of weights in the generator while also adopting state of the art GAN training techniques worked reasonably well.

Batch normalisation was replaced by spectral normalisation and the minimax loss was replaced by the hinged Wasserstein loss, both techniques were taken from (MIYATO et al., 2018), the state of the art at the time of writing. The following mathematical expression depicts the respective loss functions.

$$L_{GAN_{Gen}} = MAE(Y_{Pred}, Y_{Ref}) + \alpha \cdot MAE(VGG_{54}(Y_{Pred}), VGG_{54}(Y_{Ref})) - \beta \cdot Disc(Y_{Pred})$$

$$L_{GAN_{Disc}} = max(1 - Disc(Y_{Ref}), 0) + max(1 + Disc(Y_{Pred}), 0)$$

Figures 9.6, 9.7, 9.8, 9.9 show the results obtained from adversarial training with and without the VGG-based component. The alpha and beta parameters were empirically tuned to 0.05 and 0.005, respectively.

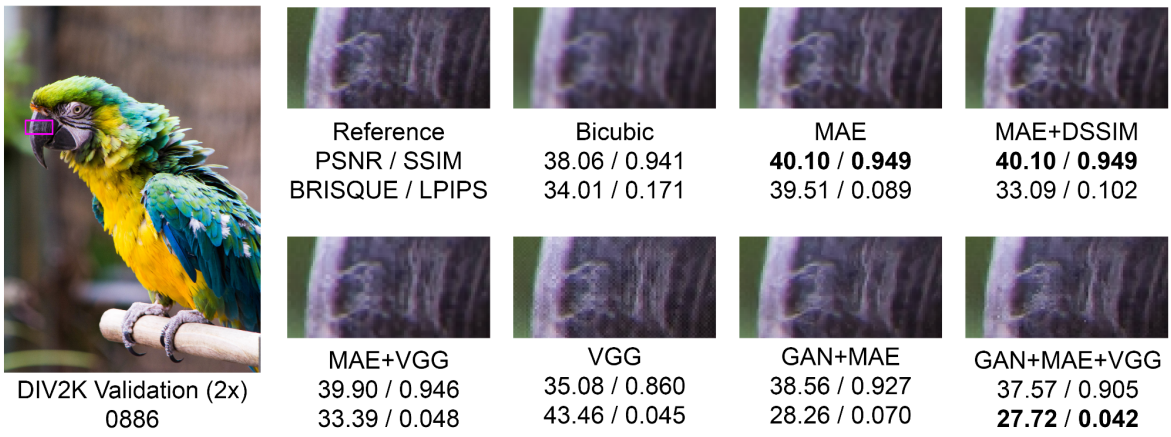


Figure 9.6: DIV2K 0886 comparison for the GAN-based loss function.

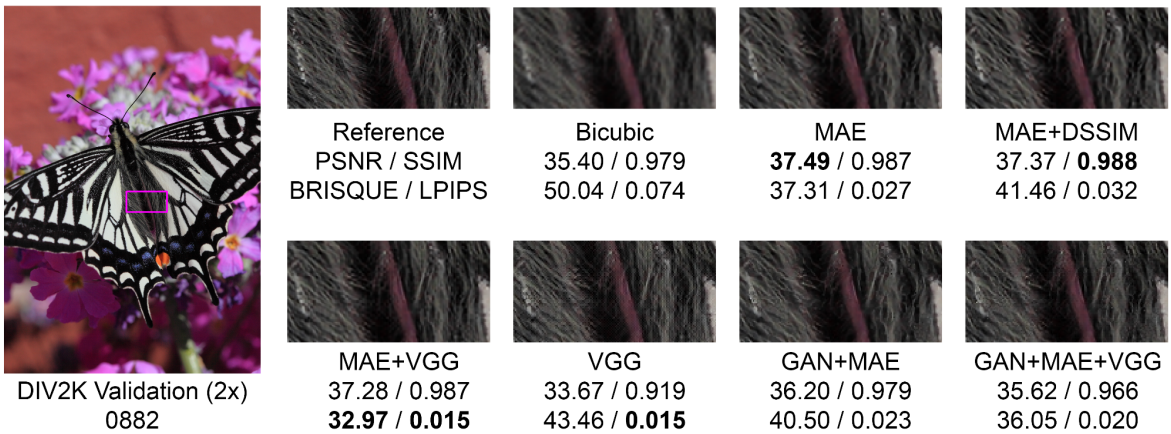


Figure 9.7: DIV2K 0882 comparison for the GAN-based loss function.

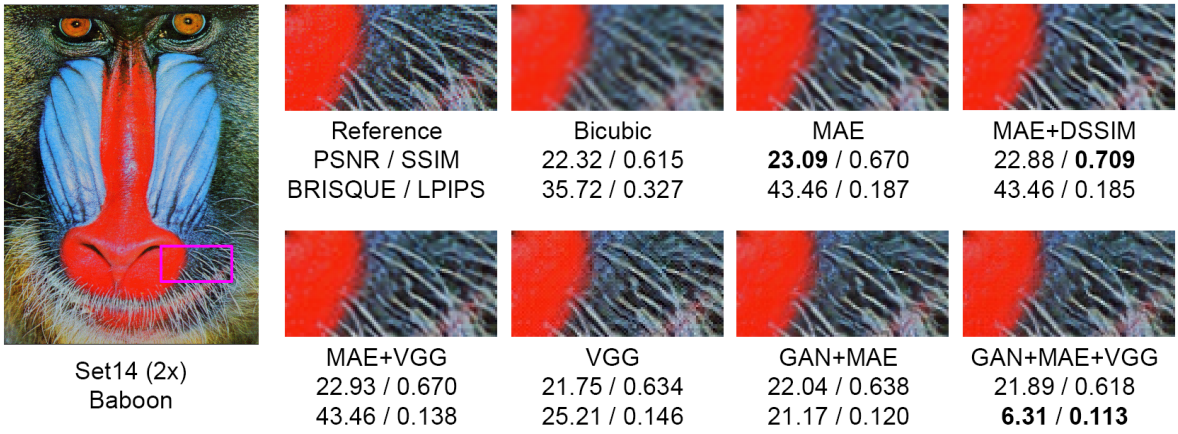


Figure 9.8: Set14 comparison for the GAN-based loss function.

Figure 9.9: Urban100 comparison for the GAN-based loss function.

It's easy to see that adding adversarial training to the system helps it synthesise realistic-looking details in an attempt to fool the discriminator. It also makes the artefacts created by the VGG-based perceptual loss function less apparent and more controlled, with most of the checkerboard patterns seemingly gone.

Unlike SRGAN and ESRGAN, networks that are much bigger both in number of weights and depth, the shallow C4F16 model employed here isn't capable of producing images that diverge too much from the ground truth, and the results are not consistently better than training with simpler distortion-only loss functions. However, it's clear that adding perceptual losses results in output images that look sharper and more texturised.

It's also worth noting that GANs in general have been mostly superseded by diffusion models, which are often much more stable. However, GANs only require a single forward step to generate an output, while diffusion models need multiple forward steps to fully denoise the

image. This is particularly important here because a GAN only differs from a normal CNN in how it's trained, the inference process is identical which means performance is also identical. Diffusion models would naturally be much slower since they require much more work to be done for each output frame.

9.3 JPEG Compression

Arising from the idea that convolutional neural networks are capable of learning how to reconstruct missing high-frequency components, achieving state of the art resampling quality in distortion and perception metrics, it's fair to conjecture that these networks can also learn how to reconstruct information lost due to lossy compression.

As explained before, modern image and video codecs usually employ chroma subsampling and DCT-domain quantisation as lossy irreversible steps in which information is thrown away and isn't normally recovered in the decoding steps. Naturally, simultaneously recovering the information lost from downsampling and compressing is tougher than only recovering the information lost to downsampling, which means that worse results are to be expected. Imagemagick was used to encode the datasets with the default quality settings. Figure 9.10 shows the difference between the models trained with different degradation models.

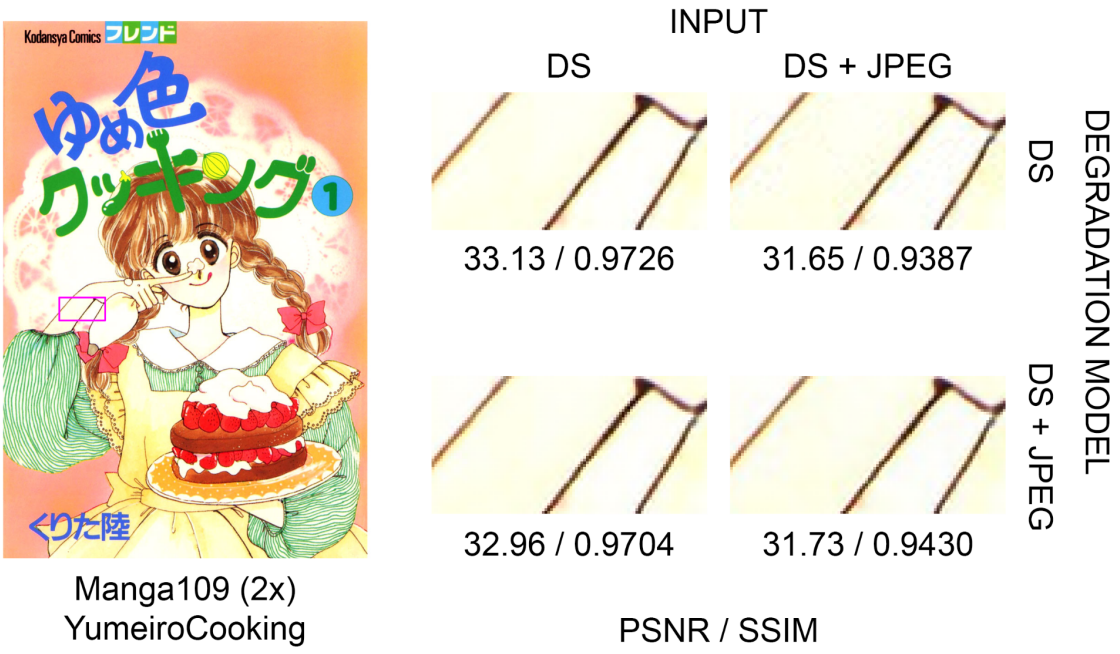


Figure 9.10: Manga109 comparison for the DS+JPEG degradation model.

It's easy to see how training the model with JPEG-compressed LR data results in much cleaner results when supersampling images with the same degradation model. The model training with downsampled-only data can't differentiate real details from compression artefacts, which is why compression artefacts are clearly visible in the top right example. What is relatively surprising is that the DS+JPEG model is still doing a good job when fed with uncompressed inputs, with scores that are only slightly worse than the model trained for this degradation model. Table 9.2 summarises the difference between the DS and DS+JPEG models with DS+JPEG test data.

Table 9.2: DS+JPEG model advantage over DS model with lossy test data.

Dataset	MAE	PSNR	SSIM	MSSSIM
Set5	-0.0009	+0.2895	+0.0067	+0.0015
Set14	-0.0007	+0.1099	+0.0029	+0.0010
B100	-0.0006	+0.0919	+0.0005	+0.0009
Urban100	-0.0007	+0.0990	+0.0042	+0.0011
Manga109	-0.0008	+0.1845	+0.0066	+0.0013
All	-0.0008	+0.1550	+0.0042	+0.0011

Chapter 10

Conclusion

At the beginning of this document, it was shown that multimedia systems are quickly becoming more capable and the gap between display technology and existing multimedia content is increasing. The original question was whether it's possible to use machine learning techniques in real time to bridge this gap and effectively increase the quality of lower resolution content even when running on integrated graphics.

The research presented in this document showed that it's possible to adapt the state-of-the-art super-resolution techniques into efficient convolutional neural networks in order to allow real-time inference on low-power devices such as phones, tablets and laptops.

The results show that using the state-of-the-art models as published is entirely unfeasible, with the Intel Iris Xe LP only achieving roughly 0.6 frames per second running the EDSR Baseline model. Henceforth, several modifications had to be made in order to achieve the objective. These modifications were done taking into account reconstruction quality and inference throughput, which revealed which architectural choices provide a good balance of performance and quality.

In short, reducing the number of layers provides a better performance-quality trade-off than reducing the number of filters per layer, due to the reduced number of temporal dependencies and the quadratic relationship between the number of filters and the number of weights. The baseline model went from running at 0.6 to 5.6 frames per second after the number of filters per layer was reduced from 64 to 8, which was then improved to 17.4 frames per second after reducing the number of residual blocks from 16 to 2.

Outputting RGB images directly also proved to be wasteful, as chroma upsampling can be done with classic FIR filters without a noticeable quality loss. The final model uses the pro-

posed CNN for the luma channel only, which increased performance from 17.4 to 35.7 frames per second. Removing the short-skip connections provided not only a performance increase but also a reconstruction quality improvement, increasing the PSNR score by 0.13 dB and the performance by 18.3 frames per second. Some architectural choices did not improve reconstruction quality enough to justify their respective performance hits, attention mechanisms, channel concatenation and short skip connections are amongst those.

The final model, named "C4F16" after the number of internal convolution layers and the number of filters per layer, achieved a final PSNR score of 31.44 dB and a final throughput performance of 28.8 frames per second. It was also shown that small CNNs capable of running in real-time are also capable of creating realistic-looking images with adversarial training and perceptual loss functions, as well as cleaning compression artefacts when trained with lossy inputs.

As for the future works, architectural choices and building blocks originally designed to solve natural language processing problems in large language models such as the Transformer have been gaining momentum in the computer vision scene as well, achieving similar and sometimes even better reconstruction quality than competing CNNs due to a scalability advantage. However, it remains to be seen whether these models can be scaled down for real-time inference on integrated graphics.

References

AGUSTSSON, E.; TIMOFTE, R. NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study. In: THE IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops. [S.l.: s.n.], July 2017.

AIZAWA, K. et al. Building a Manga Dataset "Manga109" with Annotations for Multimedia Applications. **IEEE MultiMedia**, v. 27, n. 2, p. 8–18, 2020. DOI: 10.1109/mmul.2020.2987895.

ANWAR, S.; KHAN, S.; BARNES, N. **A Deep Journey into Super-resolution: A survey**. [S.l.: s.n.], 2020. arXiv: 1904.07523 [cs.CV].

BEVILACQUA, M.; ROUMY, A.; GUILLEMOT, C.; ALBERI-MOREL, M. L. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. BMVA press, 2012.

BLAU, Y.; MICHAELI, T. The Perception-Distortion Tradeoff. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. [S.l.]: IEEE, June 2018. DOI: 10.1109/cvpr.2018.00652. Available from: <http://dx.doi.org/10.1109/CVPR.2018.00652>.

CATMULL, E. E.; ROM, R. A CLASS OF LOCAL INTERPOLATING SPLINES. **Computer Aided Geometric Design**, p. 317–326, 1974. Available from: <https://api.semanticscholar.org/CorpusID:118383557>.

DONG, C.; LOY, C. C.; HE, K.; TANG, X. **Image Super-Resolution Using Deep Convolutional Networks**. [S.l.: s.n.], 2015. arXiv: 1501.00092 [cs.CV].

DONG, C.; LOY, C. C.; TANG, X. **Accelerating the Super-Resolution Convolutional Neural Network**. [S.l.: s.n.], 2016. arXiv: 1608.00367 [cs.CV].

DUCHON, C. E. Lanczos Filtering in One and Two Dimensions. **Journal of Applied Meteorology and Climatology**, American Meteorological Society, Boston MA, USA, v. 18, n. 8, p. 1016–1022, 1979. DOI: 10.1175/1520-0450(1979)018<1016:LFIOAT>2.0.CO;2. Available from: <https://journals.ametsoc.org/view/journals/apme/18/8/1520-0450_1979_018_1016_lfioat_2_0_co_2.xml>.

FUKUSHIMA, K. Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. **Biological Cybernetics**, v. 36, p. 193–202, 1980.

FUKUSHIMA, K. Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements. **IEEE Transactions on Systems Science and Cybernetics**, v. 5, n. 4, p. 322–333, 1969. DOI: 10.1109/TSSC.1969.300225.

HENDRYCKS, D.; GIMPEL, K. **Gaussian Error Linear Units (GELUs)**. [S.l.: s.n.], 2023. arXiv: 1606.08415 [cs.LG].

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. **Neural Netw.**, Elsevier Science Ltd., GBR, v. 2, n. 5, p. 359–366, July 1989. ISSN 0893-6080.

HU, Y.; LI, J.; HUANG, Y.; GAO, X. **Channel-wise and Spatial Feature Modulation Network for Single Image Super-Resolution**. [S.l.: s.n.], 2018. arXiv: 1809.11130 [cs.CV].

HUANG, G.; LIU, Z.; MAATEN, L. van der; WEINBERGER, K. Q. Densely Connected Convolutional Networks. In: CVPR. [S.l.]: IEEE Computer Society, 2017. P. 2261–2269. ISBN 978-1-5386-0457-1. Available from: <http://dblp.uni-trier.de/db/conf/cvpr/cvpr2017.html#HuangLMW17>.

HUANG, J.-B.; SINGH, A.; AHUJA, N. Single Image Super-Resolution From Transformed Self-Exemplars. In: PROCEEDINGS of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], June 2015.

JOHNSON, J.; ALAHI, A.; FEI-FEI, L. **Perceptual Losses for Real-Time Style Transfer and Super-Resolution**. [S.l.: s.n.], 2016. arXiv: 1603.08155 [cs.CV].

KIM, J.; LEE, J. K.; LEE, K. M. **Accurate Image Super-Resolution Using Very Deep Convolutional Networks**. [S.l.: s.n.], 2016. arXiv: 1511.04587 [cs.CV].

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, 1998. DOI: 10.1109/5.726791.

LECUN, Y. A.; BOTTOU, L.; ORR, G. B.; MÜLLER, K.-R. Efficient BackProp. In: **Neural Networks: Tricks of the Trade: Second Edition**. Ed. by Grégoire Montavon, Geneviève B. Orr and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. P. 9–48. ISBN 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_3. Available from: <https://doi.org/10.1007/978-3-642-35289-8_3>.

LEDIG, C. et al. **Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network**. [S.l.: s.n.], 2017. arXiv: 1609.04802 [cs.CV].

LIM, B. et al. **Enhanced Deep Residual Networks for Single Image Super-Resolution**. [S.l.: s.n.], 2017. arXiv: 1707.02921 [cs.CV].

LINNAINMAA, S. Taylor expansion of the accumulated rounding error. **BIT**, BIT Computer Science and Numerical Mathematics, USA, v. 16, n. 2, p. 146–160, June 1976. ISSN 0006-3835. DOI: 10.1007/BF01931367. Available from: <https://doi.org/10.1007/BF01931367>.

MARTIN, D.; FOWLKES, C.; TAL, D.; MALIK, J. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In: IEEE. PROCEEDINGS Eighth IEEE International Conference on Computer Vision. ICCV 2001. [S.l.: s.n.], 2001. v. 2, p. 416–423.

MATSUI, Y. et al. Sketch-based Manga Retrieval using Manga109 Dataset. **Multimedia Tools and Applications**, v. 76, n. 20, p. 21811–21838, 2017. DOI: 10.1007/s11042-016-4020-z.

MITCHELL, D. P.; NETRAVALI, A. N. Reconstruction filters in computer-graphics. **SIGGRAPH Comput. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 22, n. 4, p. 221–228, June 1988. ISSN 0097-8930. DOI: 10.1145/378456.378514. Available from: <https://doi.org/10.1145/378456.378514>.

MITTAL, A.; MOORTHY, A. K.; BOVIK, A. C. No-Reference Image Quality Assessment in the Spatial Domain. **IEEE Transactions on Image Processing**, v. 21, n. 12, p. 4695–4708, 2012. DOI: 10.1109/TIP.2012.2214050.

MIYATO, T.; KATAOKA, T.; KOYAMA, M.; YOSHIDA, Y. **Spectral Normalization for Generative Adversarial Networks**. [S.l.: s.n.], 2018. arXiv: 1802.05957 [cs.LG].

RAMACHANDRAN, P.; ZOPH, B.; LE, Q. V. **Searching for Activation Functions**. [S.l.: s.n.], 2017. arXiv: 1710.05941 [cs.NE].

ROSENBLATT, F. **Principles of neurodynamics: perceptions and the theory of brain mechanisms**. Washington, DC: Spartan, 1962. Available from: <https://cds.cern.ch/record/239697>.

SAHARIA, C. et al. **Image Super-Resolution via Iterative Refinement**. [S.l.: s.n.], 2021. arXiv: 2104.07636 [eess.IV].

SAJJADI, M. S. M.; SCHÖLKOPF, B.; HIRSCH, M. **EnhanceNet: Single Image Super-Resolution Through Automated Texture Synthesis**. [S.l.: s.n.], 2017. arXiv: 1612.07919 [cs.CV].

SHARMA, G.; TRUSSELL, H. Digital color imaging. **IEEE Transactions on Image Processing**, Institute of Electrical and Electronics Engineers (IEEE), v. 6, n. 7, p. 901–932, July 1997. ISSN 1941-0042. DOI: 10.1109/83.597268. Available from: <http://dx.doi.org/10.1109/83.597268>.

SHI, W. et al. **Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network**. [S.l.: s.n.], 2016. arXiv: 1609.05158 [cs.CV].

SINGH, S. **Reduction of Blocking Artifacts In JPEG Compressed Image**. [S.l.: s.n.], 2014. arXiv: 1210.1192 [cs.GR].

STOKES, M.; ANDERSON, M.; CHANDRASEKAR, S.; MOTTA, R. **A Standard Default Color Space for the Internet — sRGB**. [S.l.: s.n.], Nov. 1996. http://www.color.org/contrib/sRGB.html.

TIMOFTE, R. et al. NTIRE 2017 Challenge on Single Image Super-Resolution: Methods and Results. In: THE IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops. [S.l.: s.n.], July 2017.

WALLACE, G. K. The JPEG still picture compression standard. **Communications of the ACM**, AcM, v. 34, n. 4, p. 30–44, 1991.

WANG, X. et al. **ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks**. [S.l.: s.n.], 2018. arXiv: 1809.00219 [cs.CV].

WANG, Z.; BOVIK, A.; SHEIKH, H.; SIMONCELLI, E. Image quality assessment: from error visibility to structural similarity. **IEEE Transactions on Image Processing**, v. 13, n. 4, p. 600–612, 2004. DOI: 10.1109/TIP.2003.819861.

WATSON, A. **Deep Learning Techniques for Super-Resolution in Video Games**. [S.l.: s.n.], 2020. arXiv: 2012.09810 [cs.NE].

ZEYDE, R.; ELAD, M.; PROTTER, M. On single image scale-up using sparse-representations. In: SPRINGER. INTERNATIONAL conference on curves and surfaces. [S.l.: s.n.], 2010. P. 711–730.

ZHANG, R. et al. **The Unreasonable Effectiveness of Deep Features as a Perceptual Metric**. [S.l.: s.n.], 2018. arXiv: 1801.03924 [cs.CV].

ZHANG, Y.; LI, K., et al. **Image Super-Resolution Using Very Deep Residual Channel Attention Networks**. [S.l.: s.n.], 2018. arXiv: 1807.02758 [cs.CV].

ZHANG, Y.; TIAN, Y., et al. **Residual Dense Network for Image Super-Resolution**. [S.l.: s.n.], 2018. arXiv: 1802.08797 [cs.CV].

ZHANG, Y.; ZHU, L., et al. **A Survey on Perceptually Optimized Video Coding**. [S.l.: s.n.], 2022. arXiv: 2112.12284 [cs.MM].

ZHAO, H.; GALLO, O.; FROSIO, I.; KAUTZ, J. **Loss Functions for Neural Networks for Image Processing**. [S.l.: s.n.], 2018. arXiv: 1511.08861 [cs.CV].

ZHU, R. et al. Realizing Rec. 2020 color gamut with quantum dot displays. **Opt. Express**, Optica Publishing Group, v. 23, n. 18, p. 23680–23693, Sept. 2015. DOI: 10.1364/OE.23.023680. Available from: <https://opg.optica.org/oe/abstract.cfm?URI=oe-23-18-23680>.