POLITECNICO DI TORINO

Master's Degree in ICT for smart societies

Master's Degree Thesis

Efficient Deep Multi-Image Super-Resolution for low-power devices: performance optimization through fusion and registration techniques.

Supervisors

Prof. Diego VALSESIA

Dr. Andrea BORDONE MOLINI

Candidate

Luca DE MATTEIS

October 2023

Abstract

Image super-resolution is a well-known problem in computer vision that aims to enhance a low-resolution image into a high-resolution one. In recent years, deep neural networks have achieved remarkable results in tackling image super-resolution tasks, often employing large models and subsequently huge amounts of computational resources. However, the wide diffusion of mobile devices and portable photography necessitates the design of computationally efficient solutions.

This thesis aims to address these challenges by adapting an existing lightweight neural network designed for single-image super-resolution to perform burst super-resolution while preserving its mobile-friendly characteristics. The main goal is to improve the quality of super-resolved images, assessed using the Peak Signal-to-Noise Ratio (PSNR) metric, by inputting more than one image into the network, while trying to keep the model's inference time as low as possible.

To achieve this goal, several elements like fusion techniques and registration strategies will be studied and compared. Moreover, a way to synthetically generate a burst image dataset is also exploited, in order to evaluate PSNR performance and compare the results with those of the original neural network.

This work includes an introduction to the problem statement, a theoretical background on the used tools, a literature review of methods for both single and multi-image super-resolution, as well as a description of the developed methodology and the results achieved.

Keywords Image super-resolution, Burst super-resolution, Convolutional Neural Networks, Feature fusion, Images registration, PSNR, Inference time.

Table of Contents

List of Tables

List of Figures

Acronyms

AI

Artificial Intelligence

SISR

Single Image Super Resolution

MISR

Multi Image Super Resolution

BSR

Burst Super Resolution

PSNR

Peak Signal to Noise Ratio

LR

Low Resolution

HR

High Resolution

NN

Neural Network

DNN

Deep Neural Network

CNN

Convolutional Neural Network

Chapter 1

Introduction

This thesis project aims to solve the burst super-resolution task using a lightweight neural network model, specifically a Convolutional Neural Network one. This project has been carried out with the aid of *Zebra Technologies*, which proposed the topic and provided the computational resources needed to carry out all the studies that will be explained in the following.

1.1 Problem statement

The **image restoration** problem is an inverse problem that aims to recover clean latent images starting from their degraded observations. Since infinite possible mappings exist between multidimensional degraded observations and the restored images, most of the time the inverse mappings are unknown, making it an **ill-posed** problem.
This results in an infinite solution space that requires regularisation techniques in order to derive feasible and optimal solutions. There are different sub-problems in which the image restoration task can be divided:

- Image **deblurring**: it aims at repairing images degraded by factors such as the camera motion caused by natural hand movements, or the movement of the objects depicted in the scene;

- Image **denoising**: aims at removing noise corrupting the image. Extremely important for computer vision task that needs denoising as a pre-processing step;

- Image **dehazing**: aims to remove the poor visibility of photos due to the presence of elements like mist, dust, and rain;

- Image **super-resolution**: aims to reconstruct high-resolution images starting from their low-resolution versions.

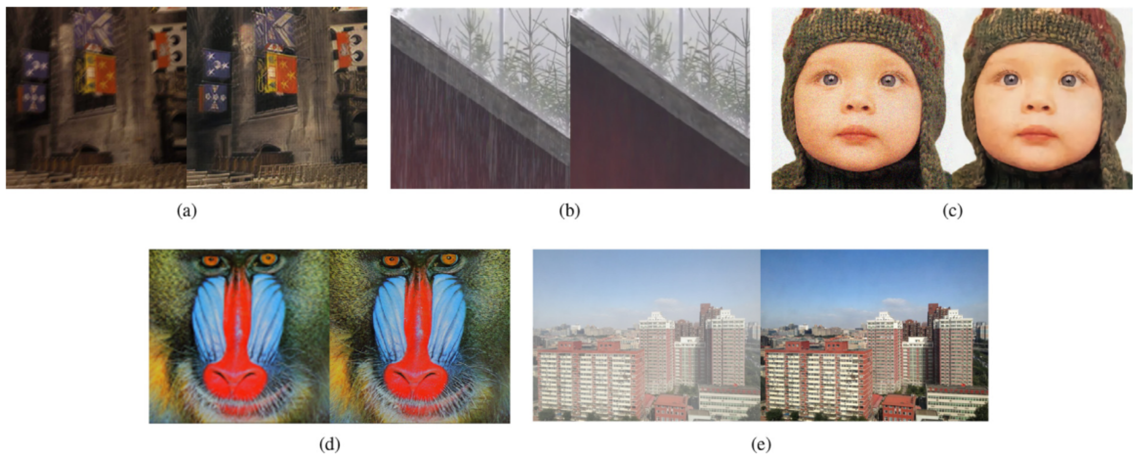


Figure 1.1: Examples of image restoration tasks: (a) deblurring, (b) draining, (c) denoising, (d) super-resolution, and (e) dehazing [1].

Image super-resolution

As said before, image **super-resolution** is the process of reconstruction of a high-resolution (HR) image starting from at least one low-resolution (LR) image. It is a fundamental topic in which deep learning models can be employed, bringing many benefits such as boosting performance and efficiency as well as making applications more realistic. More formally, given one or more LR images:

$$x \in \mathbb{R}^{\overline{w} \times \overline{h} \times c} \tag{1.1}$$

the goal is to reconstruct the HR image:

$$y \in \mathbb{R}^{w \times h \times c} \quad \text{with} \quad \overline{h} \leq h, \quad \overline{w} \leq w \tag{1.2}$$

Let $\mathbf{s} \in \mathbb{N}$ be a **scaling factor**, it holds that $h = \overline{h} \cdot s$ and $w = \overline{w} \cdot s$.
Furthermore, let:

$$D : \mathbb{R}^{w \times h \times c} \to \mathbb{R}^{\overline{w} \times \overline{h} \times c} \tag{1.3}$$

be the unknown degradation mapping that describes the relationship that holds between the LR and HR images, taking into account elements like blur type, usually modeled with bicubic downsampling with different downscaling factors. The objective is to perform an inverse mapping, even though this problem is ill-posed and a single LR image can lead to multiple non-identical HR images.
In general, there are two main branches of the image super-resolution problem based on the amount of information used to reconstruct the image, and each problem is classified as one of them.

Single-image super-resolution

In the case of Single-image super-resolution, the HR image is reconstructed given a single degraded observation that corresponds to it. However, due to the ill-posed nature of the SISR problem, these methods are limited to adding high-frequency details obtained through the learning process.

Multi-image super-resolution

Differently from SISR, in Multi-image super-resolution (MISR) more than a single LR image is utilized for producing one, or in certain cases, even more HR images. When compared to SISR, MISR has received little attention in recent years, even though it offers the possibility of reconstructing images with a finer level of detail, by combining the information coming from different images. Usually, these techniques have been exploited to address the video super-resolution problem where subsequent frames provide the additional information needed to solve the task.

Burst super-resolution

Burst super-resolution (BSR) is a particular case of the MISR problem, made popular with the increasing popularity of burst photography, in which the multiple images used to reconstruct the HR image depict the same scene and are characterized by sub-pixel shifts with respect to each other, due for example to camera motion. They provide in this way different LR samplings of the underlying scene containing additional signal information when compared to the usage of a single image, which can be exploited to generate an image that usually has a higher grade of detail when compared to the SISR case. This key advantage has made BSR an important problem for real-world applications, and it will be the main focus of this work.

Before the deep learning era, the problem of image SR was tackled by reconstruction-based and exampled-based methods. However, due to the fast development of hardware technologies and the thriving of deep learning, SR is now frequently tackled by solutions based on deep neural networks, which in recent years grew in size and depth, mainly focused on the SISR task. In this case, the problem translates into finding a model that minimizes the difference between the estimation of the HR image and the ground truth under a given loss function. A parallel direction is indeed trying to design an efficient deep neural network, characterized by a low memory footprint and low power consumption, able to tackle the BSR problem. Indeed, it naturally arises as a real-world application due to the increasing popularity of mobile burst photography, where the captured images have different sub-pixel shifts due to the natural hand tremors that happen while taking a picture.

1.2 Goal

The main goal of this thesis work is to design a deep learning architecture able to perform burst super-resolution, achieving a resolution of the reconstructed High-Resolution image comparable with State-of-the-art models. Moreover, the architecture has to be lightweight enough in order to be deployed on devices that are not equipped with huge computational resources. Another metric that will be taken into account is the inference time, with the aim of maintaining it as low as possible.

1.3 Thesis outline

Aside from this initial introduction, this thesis is organized as follows:

- Chapter 2 contains theoretical background on the fundamental concepts regarding machine learning and deep learning, including a description of the most common network architectures;
- Chapter 3 is focused on a review of deep learning models for image super-resolution, while also describing the most utilized metrics in this field;
- Chapter 4 explains the developed methodology, including the obtained results and a theoretical description of the used tools;
- Chapter 5 concludes this work, highlighting the possible future works and improvements.

Chapter 2

Theoretical background

This chapter is focused on giving an initial theoretical background on the key concepts of machine learning and deep learning, as well as describing the most common network architectures in the latter field.

2.1 Machine learning

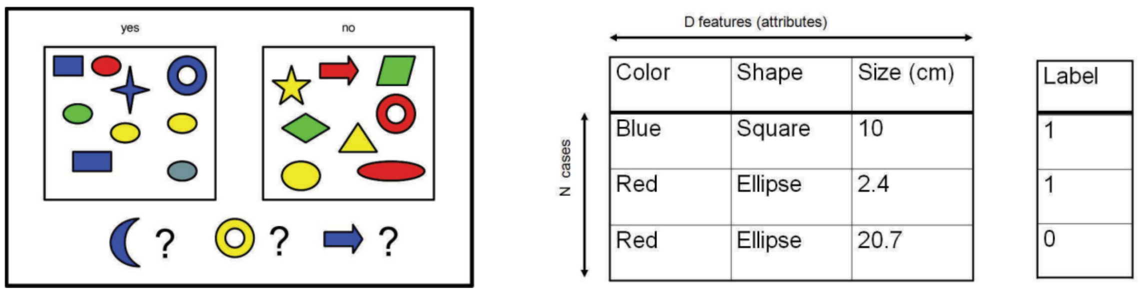
The huge amount of data that is produced nowadays highlights the need for automated methods of data analysis, which is what machine learning provides. With the term **machine learning** (ML), we usually refer to a model representing the mathematical relationships between the provided data and the desired output, able to detect patterns and relevant features from the input data, and then use the discovered patterns to process them and make predictions about future data or to perform other kinds of decision making, as depicted in figure 2.2. This is achieved by using algorithms that learn through experience, and according to the type of data used the machine-learning process can be classified into the following categories:

- **Supervised learning**: with this particular technique the work is performed using data labeled with the correct answer that should be produced by the model. Example of tasks carried out in this way is regression and classification, also known as pattern recognition;

- **Semi-supervised learning**: this technique's learning process is based on semi-labeled datasets, with one of the advantages being that the amount of labeled data needed is minimized;

- **Unsupervised learning**, also known as **knowledge discovery**, where the goal is to find interesting patterns in the data, since the learning process is performed in the absence of available labeled data, meaning that no correct answers are provided. An example application of this kind of learning process is clustering;

- **Reinforcement learning**: meaning that the models make predictions by getting rewards or penalties based on actions performed in an environment.



Color	Shape	Size (cm)	Label
Blue	Square	10	1
Red	Ellipse	2.4	1
Red	Ellipse	20.7	0

Figure 2.1: Examples of labeled data [2].

Recently, ML has become very widespread in research and has been exploited in a variety of real-world applications such as text mining, spam detection, video recommendation, image classification, and multimedia concept retrieval.

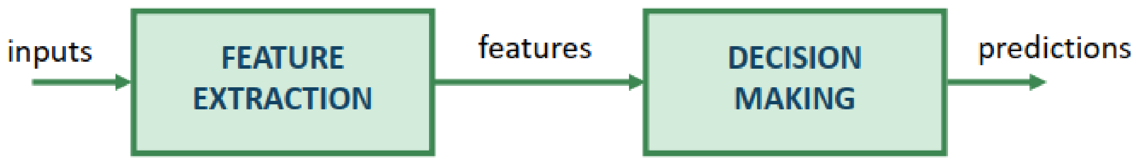


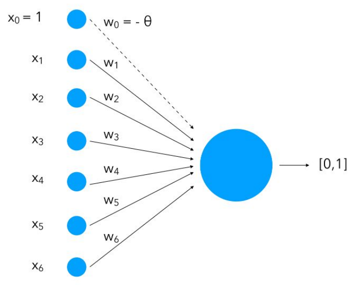
Figure 2.2: Machine Learning pipeline.

2.2 Deep Learning

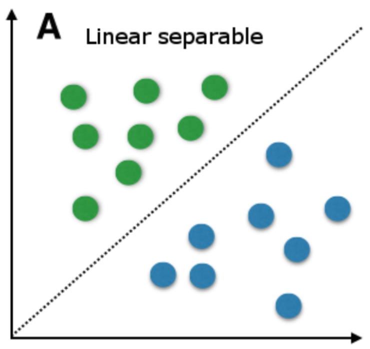
In recent years, due to the exponential growth of the amount of available data, as well as the increase of the computational power provided by the hardware components, the usage of neural networks has become very common for all the tasks that previously took into account machine learning algorithms. In the context of ML, the term Deep Learning (DL) refers to the use of deep neural networks (DNN) with the aim of solving problems of various natures and entities, which are characterized by a great number of layers composing the network and consequently a great number of neurons.

2.2.1 The perceptron

Neural networks were conceived to mimic the structure of the human brain, indeed the basic component of a neural network is an **artificial neuron**, which performs a **weighted sum** of several inputs that is compared to a threshold in order to produce an output. The first example of this kind of architecture comes from **Rosenblatt's Perceptron** (1958), which is a linear binary classifier trained using a supervised learning approach:



(a)



(b)

Figure 2.3: (a) Rosenblatt's Perceptron, (b) Linear classifier

To produce its output, it performs a weighted sum of the received inputs, which quantifies the importance of that input to the output, and confronts the result with a threshold named **bias**:

$$y = \begin{cases} 0, & \text{if } \mathbf{w}^T\mathbf{x} < b \\ 1, & \text{if } \mathbf{w}^T\mathbf{x} \geq b \end{cases} \tag{2.1}$$

Since the output is a binary one, it splits the space with a hyperplane. During their training phase, perceptrons will gradually learn a linear separation in the data: if the data is not linearly separable, the training task will fail and the perceptron will not converge to a solution where all data is correctly classified.

2.2.2 Neural Networks

Differently from the perceptron case, a **neural network** is an architecture composed of a collection of neurons in which the core operations resemble the ones introduced by the perceptron, and where the amount of neurons is far greater than one. These neurons composing the network can be arranged in several **layers**, as depicted in figure 2.4, each one with a different function:

- the **input layer** is the first layer in the architecture, composed of the neurons taking as input the data to be analyzed;
- a variable number of **hidden layers**, which are placed between the input and the output layer, specifically the number of input layers determines how **deep** a neural network is;
- the **output layer**, which is the last layer in the architecture and is made by the neurons that produce the result.

We refer to a model as a **Deep Neural Network** when the number of hidden layers is far greater than one.

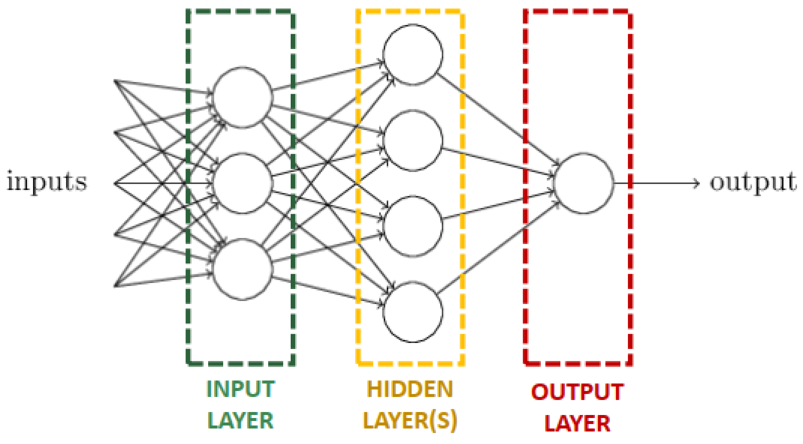


Figure 2.4: Neural network layers

Based on the types of connections made among the neurons composing the different layers, neural networks can be classified into 2 different categories:

1. **Feedforward** neural networks: in this type of network loops are not allowed, meaning that the connections follow a sequential flow. In such a way, for all the layers in the network, except for the input layer, the inputs to a certain layer correspond to the outputs of the previous layer;

2. **Recurrent** neural networks, in which connection loops are allowed, meaning that connections between nodes of the same or previous layers are allowed to feed information back into the network.

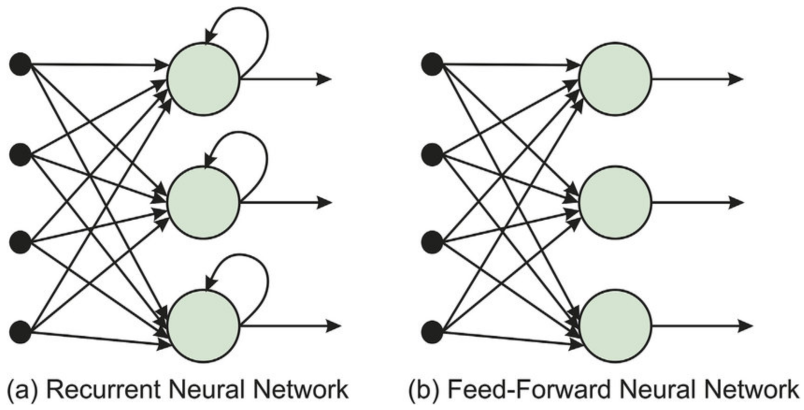


Figure 2.5: Examples of feedforward and recurrent neural networks

Learning process

The parameters that characterize a neural network are its **weights**, used by the neurons to perform the weighted sum previously mentioned, and its **biases**. These parameters have to be optimized in order to produce a reliable result, and this is done during the **training phase**, which is an iterative process that has the goal of finding the optimal value of the parameters by minimizing errors until the network is not able to improve its accuracy anymore. This process is defined by a specific algorithm that involves the use of a specific subset of data, as well as the definition of a cost function used to measure the errors.

Datasets

The data available for the learning process has to be divided into three main subsets:

1. The **training set** is the subset used during the training stage;

2. The **validation set** is also used during the training stage, but differently from the training set it is used to check the performance of the model and make decisions on the hyperparameters, mainly to avoid the problem of **overfitting**;

3. The **training set**, used for the final testing of the model

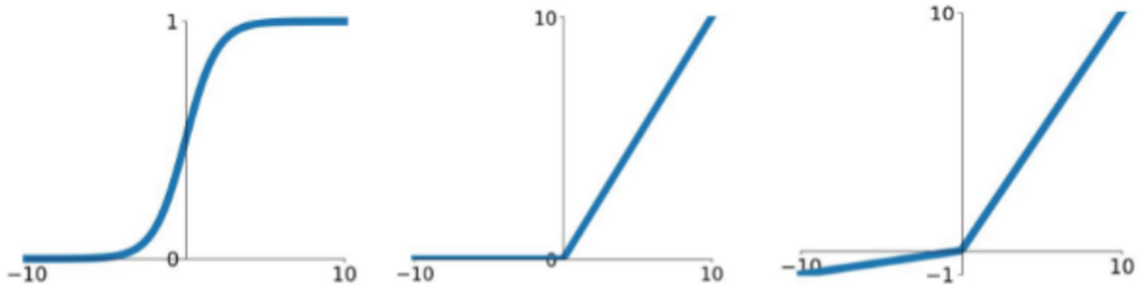
The amount and quality of the data available to carry on a learning process is the key to achieve good results. Indeed, unlike machine learning algorithms, with DNN, feature extraction, and decision-making are **data-driven**, meaning that all parts of the model are learned from training the model.

Activation function

In today's architecture thresholds are not used anymore, indeed the idea is that small changes in the inputs have to produce small changes also in the output. This is the reason why the weighted sum x of the neuron is passed to a **non-linear activation function**, some examples of which are reported both in table 2.1 and figure 2.6:

ReLU	$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$
LeakyReLU	$f(x) = \begin{cases} 0.1x & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$
Sigmoid	$\sigma_x = \frac{1}{1+e^{-x}}$

Table 2.1: Examples of activation functions, operating either element-wise or vector-wise, depending on the function.



Sigmoid function | RELU function | Leaky RELU function

Figure 2.6: Activation functions examples.

Cost function

A **cost** or **loss function** is used to measure if the output obtained from the NN is good or bad, some examples of which are reported in table 2.2. It returns a scalar that quantifies how much the output of the NN is different from the desired output, and it depends on the type of task that is carried out, even though the goal is always to try to minimize it as much as possible.

MSE / L2 Loss / Quadratic Loss	$\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}$
(Binary) Cross Entropy (average reduction on higher dimensions)	$\frac{\sum_{i=1}^{N}\sum_{j=1}^{C}\hat{y}_i \log(y_{i,j})}{N}$
Categorical Cross Entropy (sum reduction on higher dimensions)	$-\sum_{i=1}^{N}\hat{y}_i + \log\left(\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,j}\right)$

Table 2.2: Cost functions examples.

Where y is the output of the network, N is the batch size multiplied by the number of outputs (e.g. pixels), C is the number of classes, and $\hat{y}$ is the correct output.

Gradient descent

The training stage is performed by means of a **gradient descent iterative algorithm**, which allows finding the values of weights and biases corresponding to a local minimum of the cost function, starting from any point in the loss function space. Suppose the objective is to minimize a function $C(v)$ with $v = v_1, v_2, ...$ When the values of the variables are changed, the cost function changes as:

$$\Delta C \simeq \frac{\delta C}{\delta v_1}\Delta v_1 + \frac{\delta C}{\delta v_2}\Delta v_2 \tag{2.2}$$

The goal is to find $\Delta v = (\Delta v_1, \Delta v_2)^T$ such that ΔC is negative. If the gradient of C is defined as:

$$\nabla C \cong (\frac{\delta C}{\delta v_1}, \frac{\delta C}{\delta v_2})^T \quad \text{and so} \quad \Delta C \simeq \nabla C \cdot \Delta v \tag{2.3}$$

Suppose $\Delta v = -\eta \nabla C$, where η is called **learning rate**, then:

$$\Delta C \simeq \nabla C \cdot \Delta v = -\eta ||\Delta C||^2 \tag{2.4}$$

is always negative, even though the learning rate has to be accurately chosen. Indeed, a larger learning rate allows for shortening the training time with the risk of lowering the overall accuracy, on the contrary, a lower learning rate increases the time needed for training the network, but with the achieved accuracy is better. Then, in order to minimize $C(v)$, the following update rule of the variables defining the cost function is followed:

$$v \leftarrow v - \eta \nabla C \tag{2.5}$$

For fixed $||\Delta v||$, the gradient descent algorithm provides the update rule of the variables that maximize the decrease of C, thus converging to a local minimum of the cost function. Indeed, the gradient descent algorithm is not particularly efficient in avoiding local minima, which leads to the learning process stopping, thus avoiding the improvement of the model.
The basic idea of the gradient descent algorithm has been expanded by more complex algorithms, such as Stochastic Gradient Descent and the Adam optimizer (listed in algorithm 1). A fundamental feature of these algorithms involves the estimation of the gradient of the cost function from a small subset of the training data, called **minibatch**.

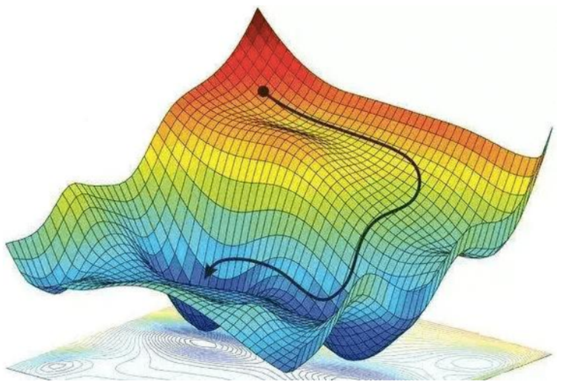


Figure 2.7: Gradient descent.

Algorithm 1: Adam optimizer algorithm. All operations are element-wise, even powers. Good values for the constants are $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$. ϵ is needed to guarantee numerical stability.

```
Data: $\alpha$ is the stepsize
Data: $\beta_1, \beta_2 \in [0, 1)$ are the exponential decay rates for the moment
      estimates
Data: $f(\theta)$ is the objective function to optimize
Data: $\theta_0$ is the initial vector of parameters which will be optimized
/*Initialization                                                  */
$m_0 \leftarrow 0$ /*First moment estimate vector set to 0        */
$v_0 \leftarrow 0$ /*Second moment estimate vector set to 0       */
$t \leftarrow 0$ /*Timestep set to 0                              */
/*Execution                                                       */
while $\theta_t$ not converged do
    $t \leftarrow t + 1$ /*Update timestep                        */
    /*Gradients are computed w.r.t the parameters to optimize     */
    /*using the value of the objective function                   */
    /*at the previous timestep                                    */
    $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$ /*Update of first-moment and second-moment
      estimates using                                             */
    /*previous value and new gradients, biased                    */
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ /*Bias-correction of estimates */
    $\hat{m}_t \leftarrow \frac{m_t}{1 - \beta_1^t}$
    $\hat{v}_t \leftarrow \frac{v_t}{1 - \beta_2^t}$
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$ /*Update parameters */
end
return $\theta_t$
/*Optimized parameters are returned                               */
```

Backpropagation

Backpropagation is a famous algorithm invented in 1970, which became popular in 1986 with a paper by Rumelhart, Hinton, and Williams since it helps in computing the gradient of the cost function, thus helping in minimizing the cost function value by adjusting the values of neurons weights and biases. In order to explain how this algorithm works, the following elements have to defined:

- w^l_{jk} is the **weight** associated to the j^{th} neuron in the l^{th} layer, coming from the k^{th} neuron in the $(l-1)^{th}$;
- b^l_j is the **bias** of the j^{th} neuron in the l^{th} layer;
- $a^l_j = \sigma(\sum_k w^l_{jk} a^{l-1}_k + b^l_j)$ is the **activation** of the j^{th} neuron in the l^{th} layer, given a generic non-linearity σ;

- The **Hadamard** product, which is the element-wise product of two vectors, described by the $\odot$ operator;
- The **error** δ_j^l, which is a measure of the error in the neuron:

$$\delta_j^l = \frac{\partial c}{\partial z_z^l} \tag{2.6}$$

It has to be remarked that the cost function can be written both as an average over its individual values given the value of a training sample and as a function of the outputs of the neural network.
For the case in which the training process is performed using minibatches obtained from the training data, the algorithm follows these steps:

1. **Input** set of training examples, initial weight/biases. For each s set $a^{x,1}$
2. **Feedforward pass**: for each layer l = 2, 3, ..., L compute $a^{x,l} = \sigma(w^l a^{x,l-1} + b^l)$
3. **Output error**: compute $\delta^{x,L} = \nabla_a C_x \odot \sigma'(w^l a^{x,l-1} + b^l)$
4. **Backpropagation**: for each layer l = L-1, L-2, ..., 2 compute $\delta^{x,l} = ((w^{l+})\delta^{x,l+1}) \odot \sigma'(z^{x,l})$
5. **Gradient descent**: For each layer l = L - 1, L -2, ..., 2 update weight and biases:

$$w^l \leftarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T \tag{2.7}$$

$$b^l \leftarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l} \tag{2.8}$$

The algorithm shows why the process is denoted as backpropagation, since errors are computed backward, starting from the final layer.

2.2.3 Convolutional Neural Networks

In the field of DL, Convolutional Neural Networks (CNN) are the most famous and commonly employed types of architecture for all tasks involving image processing, including computer vision, face recognition, etc. The usual architecture of a CNN has several typical features such as local receptive field, shared weights, and pooling strategy, all of them reported in figure 2.8, where an example architecture is reported.

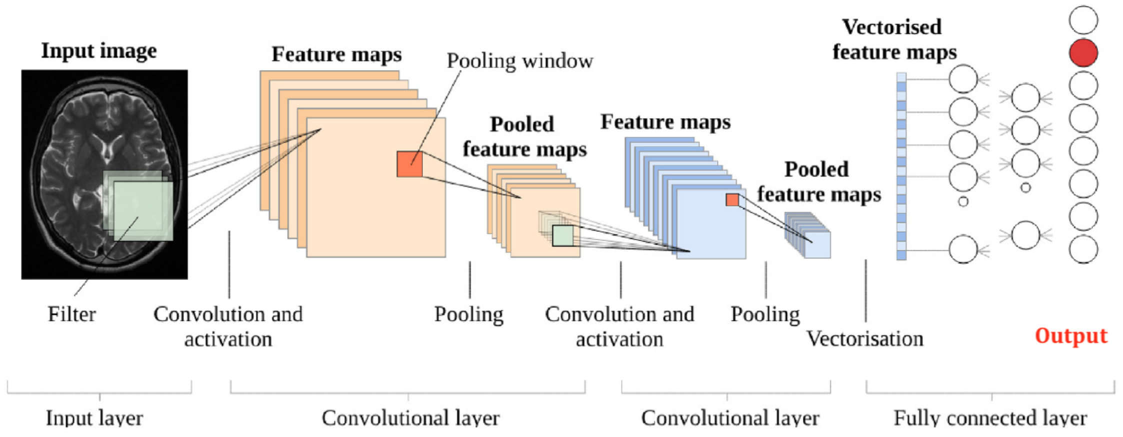


Figure 2.8: CNN architecture example.

Local receptive field

The main key feature of the CNN lies in the **convolutional layer**, which is in charge of performing a convolutional operation between the provided input x and one or more filters with weight w that produce one or more **feature maps**:

$$y_n = w * x + b = \sum_m w_m x_{n-m} + b \tag{2.9}$$

Due to this operation, an output neuron is not connected to all the input neurons of the previous layer, but just to a small subset of them, called the **local receptive field**, differently from a **fully-connected layer**. The local receptive field slides by a number, defined by the **stride** parameter, of neurons at a time across the entire input image, and at each step it associates a new region to a hidden neuron in the first hidden layer. This operation is particularly helpful when the input to the network is an image since most of the time an image pixel is highly **correlated** with its neighbors. It is particularly helpful since it allows for a reduction of the number of parameters analyzed by the network, making them much easier to train, that is why CNNs have gained so much popularity for image-related tasks. An example of a 2D convolution with a 3x3 filter is reported in figure 2.9:

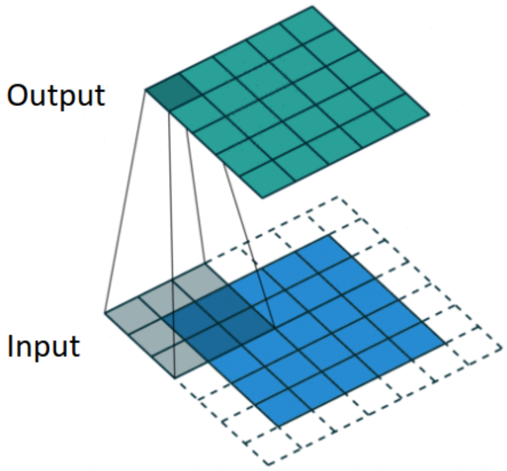


Figure 2.9: Example of 2D convolution with a 3x3 filter.

Figure 2.9 also shows the possibility of going outside the borders of the pixels of an eventual image and pad the necessary values with **zeros** or with **reflection/symmetric** padding. It has to be noted though that *valid* convolutions usually do not go outside of an image boundary.

Shared weights

Another important property of CNNs lies in the **shared weights** property, meaning that the only trainable parameters in a convolutional layer are the filter weights, and those weights are reused every time the filter shifts its position to cover all the analyzed image. This reflects the **stationarity property** of the data since the filters act as a feature detector and they can be reused regardless of the position inside the image.

Pooling strategy

In order to further reduce the size of the feature maps produced by the convolution operation, other strategies such as the **pooling** strategy can be applied, which consists of returning only a specific value within the feature map. Consider an example case of a 2x2 window, for each 2x2 non-overlapping window in the feature map, only the maximum value is selected in the **max pooling** case, or the average of the values in the window for the **average** pooling case. Moreover, feature maps can also be **upsampled** by making use of strategies such as **pixel shuffling** or **grid resampling + filtering**.

2.2.4 Generative Adversarial Networks

The architecture of Generative Adversarial Networks (GAN) has the peculiarity of being composed of two different sub-networks competing with each other: a generator G and a discriminator D. The generator sub-network learns how to produce samples that are as similar as possible to the ones of a given dataset that are used to try to fool the discriminator, which on the other hand is in charge of trying to distinguish between samples that have been generated from the ones coming from the actual dataset. This architecture is trained until the discriminator is no longer able to distinguish between samples that have been generated from those that are not. Specifically, the two sub-networks are trained independently in such a way that the generator learns how to produce better samples, while the discriminator learns how to recognize the synthetic samples.

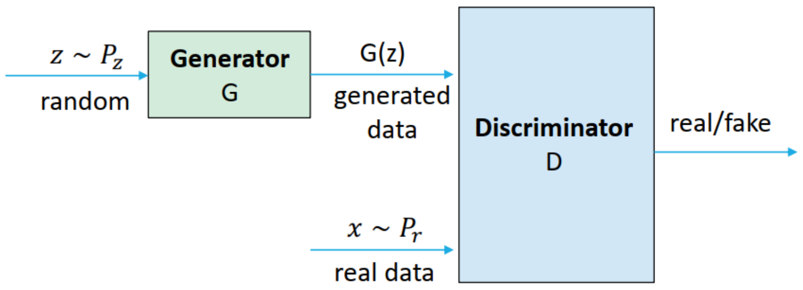


Figure 2.10: GAN architecture example.

The primary disadvantage of this kind of network is that sufficient training stability is hard to reach, reason why regularization terms can be introduced.

2.2.5 Residual Networks

Residual Neural Networks (ResNets) are an improved neural network architecture, the main feature of which is the introduction of the **skip connections**, which allows one to leap over a certain number of layers in the architecture, as depicted in figure 2.11. They are used if an input to a certain layer in the architecture is a good predictor of the output so that the layer is able to learn a correction, instead of a full transformation. Estimating the correction is a simpler process, so easier to fit with a few parameters, and also allows a better flow of information in backpropagation.

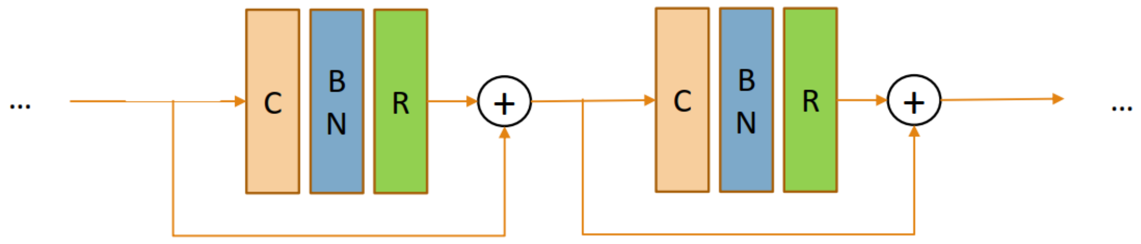


Figure 2.11: Residual network example.

The use of skip connections allows for mitigating the problem of vanishing gradients as well as the degradation problem. Moreover, it allowed the building of much deeper networks than before containing hundreds of layers instead of a few tens. An improvement to the ResNets is the introduction of the **attention** mechanism, which is used to let the network focus on specific parts on the input sequence, by assigning different weights to the different parts of the input sequence, leaving room for the highest weights for the most important parts. There are basically two categories of attention mechanisms, that can also be applied simultaneously:

- **Channel attention**: focuses on which channels carry crucial details among all the existing and lets the network focus more on them. This is achieved by spatially averaging each feature map and then applying two fully connected layers, one followed by RELU and one followed by sigmoid, then scaling the original feature map with the newly computed weights.
- **Spatial attention**, differently from channel attention, focuses on which positions of the input feature maps carry the most important details. This results in a feature map that contains a specific weight associated with each position.

2.2.6 Transformer

The transformer is an encoder-decoder architecture first proposed by Vaswani *et al.* [3] solely based on the attention mechanism, using also point-wise, fully connected layers for both the encoder and the decoder parts, both depicted in figure 2.12

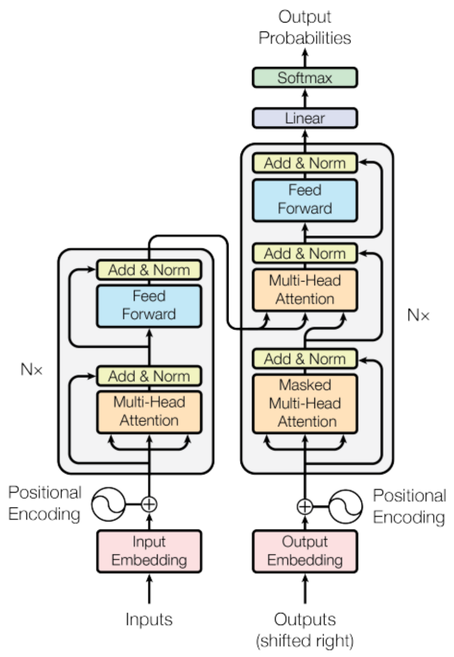


Figure 2.12: Transformer architecture [3].

The left part, depicting the encoder part, is made of 6 layers, each one of them composed of two sub-layers: the first one is a multi-head self-attention mechanism, while the second sub-layer is a fully connected feed-forward layer. Specifically, the output of each sub-layer is:

$$\text{LayerNorm}(x + \text{Sublayer}(x)) \tag{2.10}$$

where Sublayer() is the function implemented by the sub-layer itself, while the + operator describes the residual connection, and LayerNorm() is the use of a layer

normalization.
Similarly to the encoder part, the decoder is composed of 6 identical layers and equation 2.10 can be applied also to it. However, differently from the encoder part, a third sub-layer exists performing multi-head attention over the output of the encoder part.

Attention sublayer

The attention sublayer of the transformer implements an attention function that maps a set of key-value pairs to an output, by means of a query vector. Specifically, the query is used to measure the compatibility of the latter with the keys corresponding to the input values. The result is the weight associated with each input value and used for the weighted sum of the input values that returns the output of the function.
The particular type of attention is a **multi-head attention**, which is a module that allows running an attention function several times in parallel. An attention function, specifically a **scaled dot-product** attention, computes the weights assigned to the values as the softmax function of the ratio of $\sqrt{d_k}$ the dot product of the query and key vectors, where d_k is the dimensionality of both keys and query vectors:

$$\text{Attention}(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.11}$$

where Q, K, and V are matrices.
Each obtained output is transformed into the expected dimension by multi-head attention, which also concatenates all of them, allowing the model to join information coming from different representation subspaces:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, ..., head_h)W^O \tag{2.12}$$

where

$$head_i = \text{Attention}(QW_i^Q, QW_i^K, VW_i^V) \tag{2.13}$$

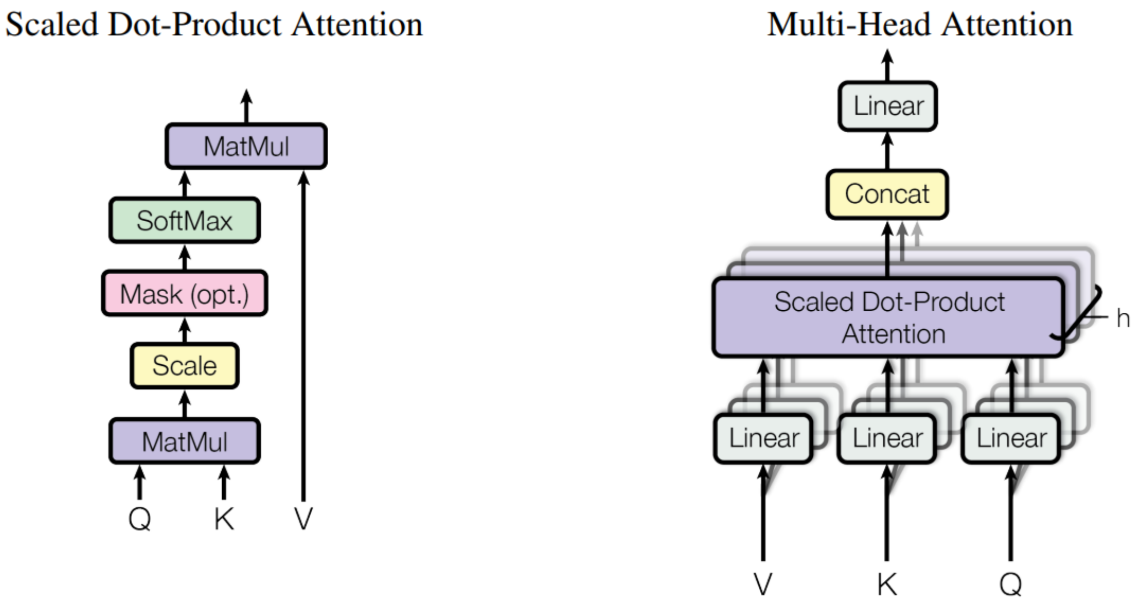


Figure 2.13: Attention mechanism [3].

Feed-forward sublayer

The fully-connected feed-forward sub-layer consists of two linear transformations with a RELU activation function in between:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{2.14}$$

The linear transformation is the same one, even if applied to different positions, thus using different parameters from layer to layer. This sub-layer can also be described as two convolution operations with a stride equal to 1.

Chapter 3

Deep learning models for image super-resolution

This chapter aims to give an overview of the most commonly used metrics and strategies for tackling the image super-resolution task in the field of deep learning. A focus will be on the useful metrics to quantify the performance of a deep learning model, as well as the most used loss functions used for training the latter. Moreover, a literature review of different state-of-the-art deep learning models designed for image super-resolution is reported, underlining what are the common characteristics of models designed for solving this problem.

3.1 Deep learning for image super-resolution

In order to enlarge and enhance the quality of an image, a traditional method would be to enlarge the image and fill the holes between pixels by copying the values from the closest pixels or by taking the weighted average of the neighboring pixels, which is the kind of operation performed by the usual interpolation methods. However, the results are not quite promising, leading to an image that is far from looking high-resolution, simply looking like a low-resolution image with larger pixels. Even though the problem might seem virtually impossible, the usage of Neural Networks has proven to be far superior to traditional methods, capable of hallucinating details learned from the larger set of images used to train them, learning in this way the mapping that exists between both the low-resolution and the high-resolution images, as proposed by the pioneering work by Dong *et al.* with SRCNN [4]. In this section, the main metrics used to evaluate a DL model trained to solve the super-resolution problem will be explained.

3.1.1 Evaluation: Image quality assessment

First of all the metrics used for the performance evaluation of the trained DL models will be introduced. Many of them fall under the term **Image Quality Assesment** (IQA), which refers to any metric that quantifies how realistic the image appears after applying SR methods. Image assessment can either be subjective or objective, and it is a challenging task since it has to take into account many properties associated with excellent image quality, such as sharpness, contrast, and absence of noise [5]. In the following are reported some examples of IQA evaluation methods:

- **Peak Signal-to-Noise-Ratio** (PSNR): It is the ratio between the maximum possible pixel value L (255 for 8-bit representations) and the Mean Square Error (MSE) of reference images. Given the approximation $\hat{\mathbf{y}}$ and the ground-truth $\mathbf{y}$, PSNR is a logarithmic quantity using decibel scale (dB):

$$PSNR(y, \hat{y}) = 10 \cdot \log_{10} \frac{L^2}{\frac{1}{N_y} \sum_{p \in \Omega_y} [y_p - \hat{y}_p]^2} \tag{3.1}$$

  where p denotes a generic pixel in the analyzed images, while Ω_y describes the set of all valid positions of p, given $\mathbf{y}$, $\hat{\mathbf{y}} \in \mathbb{R}^{w \times h \times c}$:

$$\Omega_y = \{(i, j, k) \in \mathbb{N}_1^3 | i \leq h, j \leq w, k \leq c\}$$

  It is widely used as an evaluation metric for SR models, even though it might lead to mediocre results in real scenarios, since it focuses on pixel-level differences instead of the human visual perception, thus poorly correlating with subjectively perceived quality.

- **Structural Similarity Index** (SSIM) [6]: it is a better measure of a perceptual image than the PSNR, since differently from the latter, it does not focus on pixel-wise differences, but depends instead on an image luminance, contrast, and structures. For a given image y and its approximation $\hat{y}$, the SSIM is computed as:

$$SSIM(y, \hat{y}) = [C_l(y, \hat{y})]^\alpha \cdot [C_c(y, \hat{y})]^\beta \cdot [C_s(y, \hat{y})]^\gamma \tag{3.2}$$

  where α, β, γ are all positive constants for weighting the relative importance of all comparison components related to illuminance, contrast, and structure, which are computed as:

$$C_l = S(\mu_y, \mu_{\hat{y}}, c_1) \quad \text{and} \quad C_c = S(\sigma_y, \sigma_{\hat{y}}, c_1) \tag{3.3}$$

$$C_s(y, \hat{y}) = \frac{\sigma_{y,\hat{y}} + c_3}{\sigma_y \cdot \sigma_{\hat{y}} + c_3} \tag{3.4}$$

  where, the S present in equation 3.3 is a similarity comparison function proposed by the authors, given certain scalar variables x and y to compare, and $c = (k \cdot L)^2$ with $0 < k << 1$, defined as:

$$S(x, y, c) = \frac{2 \cdot x \cdot y + c}{x^2 + y^2 + c} \tag{3.5}$$

  while $\sigma_{(y,\hat{y})}$ present in 3.4 is the empirical covariance computed as:

$$\sigma_{(y,\hat{y})} = \frac{1}{N_y - 1} \sum_{p \in \Omega_y} (y_p - \mu_y) \cdot (\hat{y}_p - \mu_{\hat{y}}) \tag{3.6}$$

Lastly, μ_y is the luminance, computed as the mean of the intensity, while σ_y is the contrast estimated as its standard deviation:

$$\mu_y = \frac{1}{N_y} \sum_{p \in \Omega_y} y_p \tag{3.7}$$

$$\sigma_y = \frac{1}{N_y - 1} \sum_{p \in \Omega_y} [y_p - \mu_y]^2 \tag{3.8}$$

Different from the PSNR, it better meets the requirements of human perception assessment.

3.1.2 Learning objectives

The ultimate scope of training a DL model for SR is to reconstruct an HR image from an LR one, thus defining the right cost function to minimize is a key point when training such an architecture. Among all the possible learning objectives, the focus of this section is on the *regression-based* ones. They attempt to explicitly model the relationship between the input to the model and the output, and possible categories of such loss function are *pixel-loss* ones, as well as the *content-loss* ones.

Pixel-loss

Pixel-loss functions focus on the measurement of the pixel-wise difference between the ground-truth image $\mathbf{y}$ and its estimation $\hat{\mathbf{y}}$ produced by the model. Examples of this kind of function are the ones that follow, note that p denotes a generic pixel in the analyzed images, while Ω_y describes the set of all the possible pixel locations, as in equation 3.1. :

- **Mean Absolute Error** (MAE) or **L1-loss**, which takes the absolute differences between every pixel in the pair of analyzed images and returns the average value:

$$L_1(y, \hat{y}) = \frac{1}{N_y} \sum_{p \in \Omega_y} |y - \hat{y}_p| \tag{3.9}$$

- **Mean Squared Error** (MSE) or **L2-loss**, which is similar to the L1-loss, but weights high-value differences higher than low-value ones, due to an additional square operation, being extremely sensible to extraordinary values in doing so:

$$L_2(y, \hat{y}) = \frac{1}{N_y} \sum_{p \in \Omega_y} |y - \hat{y}_p|^2 \tag{3.10}$$

- **Charbonnier-loss**, defined by:

$$L_{Charbonnier}(y, \hat{y}) = \frac{1}{N_y} \sum_{p \in \Omega_y} \sqrt{|y - \hat{y}_p|^2 + \epsilon^2} \tag{3.11}$$

where $0 < \epsilon << 1$ is a small constant to avoid zero terms.

Note that multiple variants of these functions exist, depending on the specific task that has to be carried out. As for the PSNR, *pixel-loss* functions focus on pixel-wise differences and do not take into account structural information about the content of an image, they subsequently allow to reach higher values of PSNR during the training stages.

Content-loss

Differently from pixel-loss functions, the aim of **content-loss** functions is to incorporate image features rather than reaching pixel-level detail. This type of loss function utilizes the feature maps produced by an external feature extractor, which remains fixed during the training process, that has been pre-trained on another task, different from the model that is training. The content loss has indeed the goal of minimizing the differences between the feature maps produced by the pre-trained network and the SR model as much as possible. For a given layer l it is defined as:

$$L_{content}(y, \hat{y}, l) = ||\phi^l(\hat{y}) - \phi^l(y)||_2 \tag{3.12}$$

Thus, the goal of using this loss function is not to generate pixel-perfect estimations, but instead to produce images whose features are close to the features of the target.

3.1.3 Upsampling methods

A crucial aspect when designing an SR model is the **upsampling** method used to increase the spatial size of a certain feature map. which can either be *interpolation-based* or *learning-based.*

Interpolation-based methods

Most SR models utilize interpolation-based methods mainly due to their simplicity, and the most known methods are the nearest-neighbor, bicubic, and bilinear interpolation. The **nearest-neighbor** one is the most straightforward since the interpolated value is based on the nearest pixel values, even though it may result in block artifacts. **Bilinear** interpolation on the other hand does not produce block artifacts, since it produces smoother transitions even though it needs a receptive field of 2x2. Finally, **bicubic** interpolation is the one producing the smoothest results, even though a receptive field of 4x4 and a higher computation time is needed.

Learning-based methods

With learning-based upsampling methods are used to increase the spatial size of the learned feature maps. An example of this kind of method is the **sub-pixel** layer introduced with [7]. It is made of a convolutional layer that takes a feature map as input and rearranges the output to increase its spatial size. The upsampling is defined by a scaling factor s, and it is carried out in the channel dimension. Specifically, given an input feature channel with a number of channels c, it produces an output with $s^2 \cdot c$ channels. Moreover, a convolution operation with zero padding is applied,

in such a way that the dimensions of the input and the output feature map remain the same. Finally, it reshapes the feature map to produce a spatially upsampled output.
Another alternative is **attention-based upsampling** [8]. It follows the definition of attention-based convolution (or scaled dot product attention) and replaces the 1x1 convolutions with upsampling methods. In more detail, it replaces the convolution for the query matrix with bilinear interpolation and the convolution for the key and value matrix with zero-padding upsampling similar to transposed convolution. It needs fewer parameters than transposed convolution but a slightly higher number of operations, which translates into a slower training time.
Besides the choice of the upsampling method to employ in the super-resolution architecture, another fundamental aspect concerns **where** placing it in the architecture. The available possibilities are:

1. **Pre-upsampling**: upsampling is applied at the beginning of the pipeline. Afterward, convolutional layers are applied to refine the upsampled input and extract features;

2. **Post-upsampling**: upsampling is performed at the end of the pipeline, thus reducing memory and computational costs. It helps in reducing the model complexity, and feature extraction is performed in lower dimensional space;

3. **Progressive upsampling**: unlike the previous alternatives, progressive upsampling gradually increases the feature map size within the architecture. It divides the upscaling problem into different small tasks, thus fitting the multi-scale SR problem;

4. **Iterative Up-and-Down upsampling**: this technique also downsamples the feature maps produced within the architecture to better understand the mapping that exists between LR and HR image.

3.2 State of the art review

This second section lists a literature review of some existing deep-learning models for both the multi and single-image cases of the image super-resolution problem. Lightweight models deployable on mobile devices will also be described.
The objective is to try highlighting the common strategies and techniques that are used to efficiently tackle this task. It will become clear how the preferred operation to transfer the images into feature space is the convolution, and how lightweight models strongly rely on sequences of convolution operations to also learn the mapping that exists between the LR and HR image, while more complex networks strongly use transformer (section 2.2.6) layers. Also, skip connections are widely used to learn residuals and the most used operation to upsample the feature maps and produce the super-resolved image is a pixel shuffle layer, usually placed at the end of the architecture.

3.2.1 Single-image super-resolution

SRCNN

The pioneering work in the field of deep learning for image super-resolution is the *Super-Resolution Convolutional Neural Network* (SRCNN) architecture presented by Dong *et al.* [4]. The authors were the first to propose the use of a CNN to learn the mapping that exists between the input LR image and the output HR image to reconstruct an image that is as similar as possible to the ground-truth HR image. The proposed network reached a reconstruction quality comparable to all the previous existing solutions based on sparse-coding-based methods, while also being faster and incorporating all the operations previously performed individually into a single unified workflow.

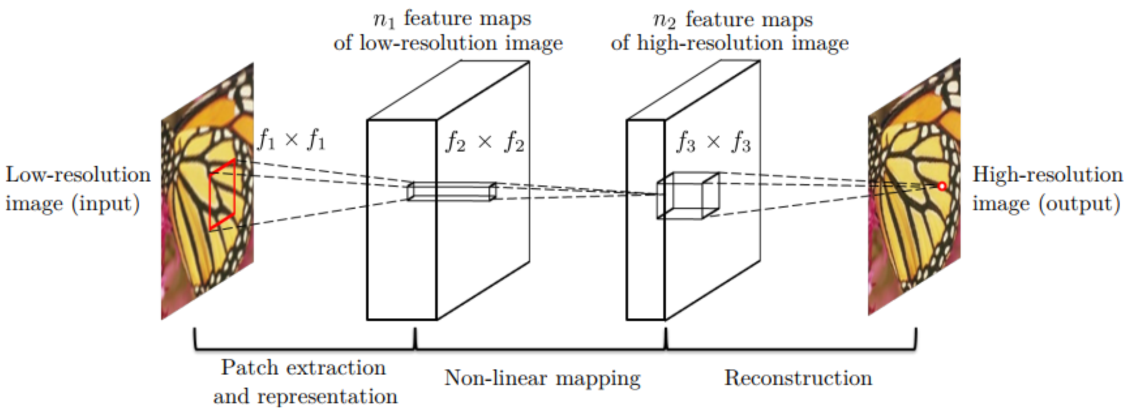

n_1 feature maps of low-resolution image
n_2 feature maps of high-resolution image
Low-resolution image (input)
$f_1 \times f_1$
$f_2 \times f_2$
$f_3 \times f_3$
High-resolution image (output)
Patch extraction and representation
Non-linear mapping
Reconstruction

Figure 3.1: SRCNN network architecture [4].

As depicted in figure 3.1, the proposed pipeline consists of 3 steps used to learn the mapping between LR and HR images, which are:

1. **Patch extraction and representation**;
2. **Non-linear mapping**;
3. **Reconstruction**;

Even if not depicted, the workflow begins with a **pre-processing** step in which the LR image is spatially upsampled via **bicubic interpolation**.
The first step consists of **extracting** from the LR interpolated image overlapping **patches** that, by means of convolutions followed by RELU activation functions, are represented as a high-dimensional vector. These vectors comprise a variable number of feature maps that depend on the dimensionality of the filters used inside the convolution operation.
In the following step, each one of the high-dimensional vectors previously obtained is **non-linearly** mapped onto another high-dimensional vector. This operation results in a set of vectors that comprises other feature maps that conceptually are the representation of a high-resolution patch.
Finally, the *reconstruction* step generates the final HR image by aggregating the high-resolution patch-wise representations.

Even with such a lightweight architecture, the SRCNN achieved results greater than the state-of-the-art sparse-coding-based methods, demonstrating also how all the operations performed by sparse-coding-based methods could be unified under a single feedforward architecture.

SWINIR

The SwinIR model is a complex architecture developed by Lian *et al.* [9] able to perform different image restoration tasks aside from image super-resolution, such as image denoising as well as JPEG compression artifacts. Differently from other state-of-the-art methods that rely on convolution operations, SwinIR is based on the usage of the Swin Transformer, as depicted in figure 3.2. The representation of the network architecture highlights how it mainly consists of three modules:

1. **Shallow feature extraction**;
2. **Deep feature extraction**;
3. **HQ image reconstruction**;

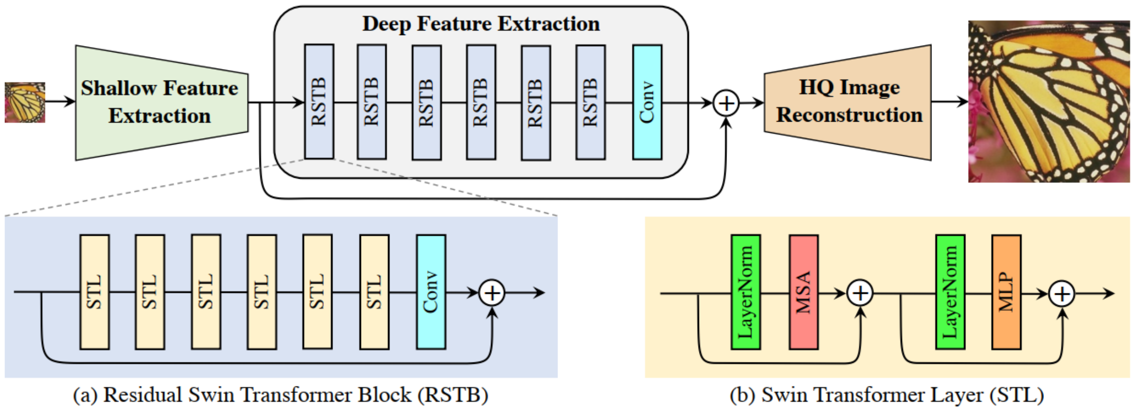


Figure 3.2: SwinIR architecture for image restoration proposed by [9].

The shallow feature extraction is performed on given an LR image input of a certain dimension and number of channels by means of a 3x3 convolutional layer that allows mapping the image into a feature space of higher dimension. The convolution layer has also the advantage of allowing better results and leading to a more stable optimization.
Subsequently **the deep feature extraction** is performed by a module composed of a number K **Residual Swin Transformer blocks (RSTB)** and a final 3×3 convolutional layer, that facilitates the aggregation of the extracted shallow and deep features.
For what concerns the structure of the RSTB, figure 3.2 (a) shows how it is mainly composed of a sequence of **Swin Transformer Layers** (STL), used to extract intermediate features, and a final convolutional layer, along with a residual connection used to add the input of the block to the output of the final convolution. To extract the intermediate features, the STLs first reshape the input into non-overlapping

local windows, and for each of them separately compute the standard self-attention. As depicted in figure 3.2 (b) a single swing transformer is composed of 3 modules. Specifically, in the multihead self-attention (MSA) block the attention function is performed in parallel for a certain number of times, and the results are then concatenated. The multi-layer perceptron (MLP) is then used for further feature transformations and it has two fully-connected layers with GELU non-linearity between them. Before both the MSA and MLP blocks, a LayerNorm (LN) layer is added. Moreover, the input and the output of the sequence of the LN and MSA/MLP layers are added by means of a residual connection.
Finally, the **HQ image reconstruction** is achieved by a reconstruction module made of a sub-pixel convolution layer used before upsampling the features if needed by the task that has been performed. If that is not the case a single convolution layer is used for reconstruction. However, before this operation, a long skip connection allows aggregating the shallow features, which mainly contain low frequencies, and the deep ones, which on the other hand focus on recovering lost high-frequencies, both extracted in the previous steps. This has the purpose of helping the deep feature extraction module to mainly focus on recovering high-frequency information.

3.2.2 Lightweight single-image super-resolution

Anchor-Based Plain Net

Anchor-Based Plain Net (ABPN) is a mobile-friendly model able to perform SISR on mobile devices. Du *et al.* [10] designed an efficient architecture, depicted in figure 3.3, that after an 8-bit quantization can effectively be deployed on a mobile device. As shown, it mainly consists of four parts:

1. **Shallow Feature Extraction**;
2. **Deep Feature Extraction**;
3. **Reconstruciton part**;
4. **Post-processing part**.

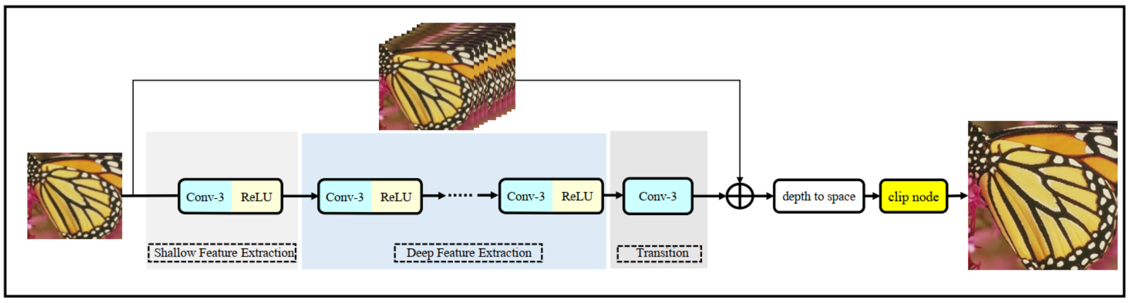
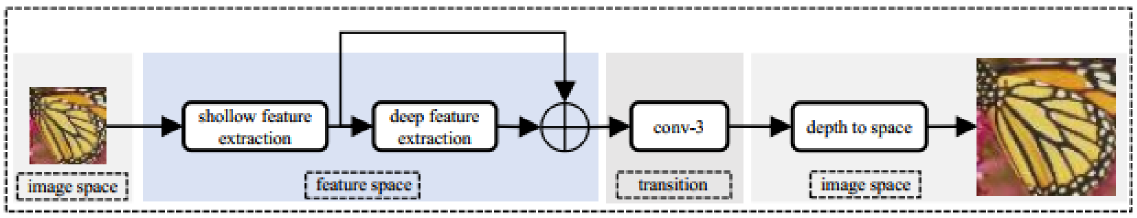


Figure 3.3: Anchor-based plain net architecture [10].

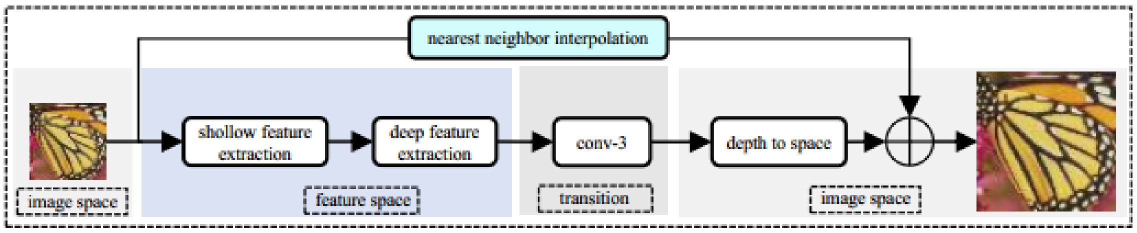
The **shallow feature extraction** part transfers the image to the feature space by means of a 3x3 convolutional layer followed by a RELU activation function.
This block is followed by the **deep feature extraction** part which is composed of 5 pairs of 3x3 convolutional layers and RELU activations, with a single final convolutional layer not followed by RELU used to transfer the produced features to the HR image space. It has to be noted that the convolution-RELU pair resulted in being the fastest pair to gradually refine details, after several experiments carried out by the authors.
The output of the last convolutional layer is added to the result of the **anchor-based residual learning** to get the super-resolved image. The **anchor-based residual learning** (ABRL) is achieved by directly repeating every pixel in the LR space **9 times**, generating in this way anchors for every pixel in HR space. It is a special case of nearest neighbor interpolation, achieved when the scale factor is an integer. It can be easily realized with one channel concatenation and addition node.
Finally, as a part of the **post-processing** part, a pixel shuffle layer is used to upsample the features, while the final clip node is used to restrict those values lower than 0 and higher than 255. This is crucial in order for the 8-bit quantization to be performed without errors.
As with other types of residual learning, examples of which are reported in figure 3.4, ABRL is used to produce weights and activations with lower standard deviation. Residual learning can be mainly split into two categories: **image-space residual learning** (ISRL) and **feature-space residual learning** (FSRL). FSRL is widely adopted by state-of-the-art models since it performs better in floating-point space, but ISRL is better for models that have to be quantized to INT8 because, as demonstrated by the authors, it forces them to learn small residuals, thus leading to a small drop in performance. ABRL has the advantage of improving the performance of the model quantized to INT8 compared with FSRL, while also exploiting parallelization thanks to the multi-branch architecture.

QuickSRNet

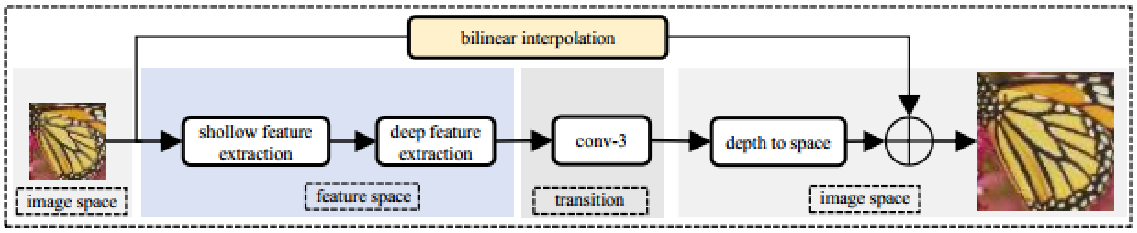
Berger *et al.* [11] developed QuickSRNet, a lightweight architecture for real-time image super-resolution applications on mobile devices. As depicted in figure 3.5, the model does not utilize an input-output residual connection in its architecture as the previous models and is composed of a certain number m of intermediate convolutional blocks characterized by a number of f feature channels. Since the network has to be deployed on mobile devices, which is a challenging task due to several constraints such as compute, thermal, and power constraints, it has to be quantized. In order to increase the robustness to the quantization process, a RELU activation function clipped between 0 and 1, as well as a residual learning motivated initialization scheme is used. The proposed architecture achieved an accuracy-to-latency trade-off better than existing neural architectures for single-image super-resolution.



(a) Residual learning in feature space



(b) Nearest neighbor interpolation in image space



(c) Bilinear interpolation in image space

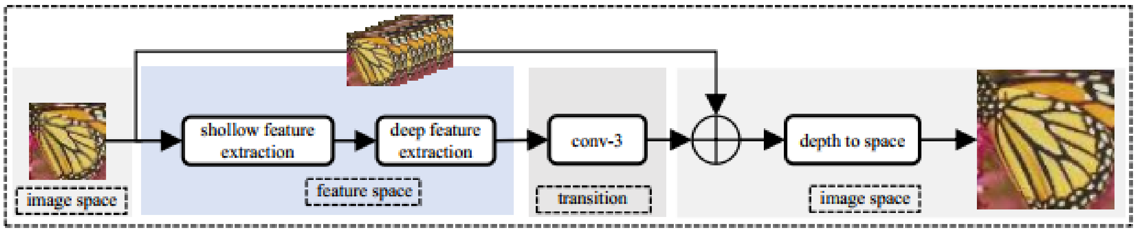


Figure 3.4: Examples of residual learning [10].

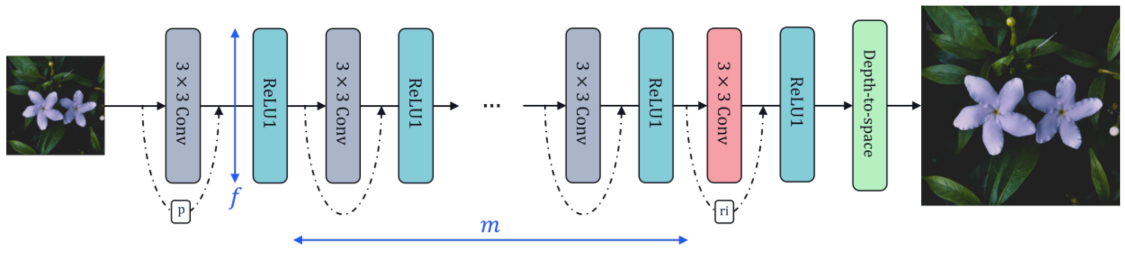


Figure 3.5: QuickSRNet architecture [11].

The residual learning motivated initialization scheme proposed by the authors makes the input image propagate throughout the entire network. The **identity initialization** technique allows each intermediate convolutional layer in the architecture to simulate a localized skip connection, as depicted in the previous figure:

$$y = W \circledast x + x \tag{3.13}$$

The skip connection is collapsed into the convolution module, and it only works if x and y have the same dimensions. Therefore it is not directly applicable to the first and last layer of the architecture since these layers respectively change the number of channels from 3 to f and f to $3 \times s^2$, where s is the upscaling factor.
For these layers, the initialization scheme is modified into either **partial identity initialization** or **repeat-interleaving identity initialization**. In the first case, the 3 channels in input to the first convolutional module in the architecture are added to the first 3 channels of the output produced by the same module, in such a way that the remaining f-3 channels are left unchanged.
In the second case, the first 3-channels of the input to the last convolutional module are repeat interleaved s^2 times and added to the output of the layer, to mimic the nearest-neighbor upscaling typically performed in the input-to-output connection of state-of-the-art residual architectures.
All the previous initialization techniques are obtained by adding ones to the kernel weights of the corresponding convolutional module at the appropriate location.

3.2.3 Burst super-resolution

PIUnet

PIUnet [12] is a neural network designed for solving *multi temporal image super-resolution* with the particularity of addressing invariance to permutations in the input burst and the estimation of the uncertainty in the produced HR image. Indeed, one of the key features of PIUnet is its **invariance to permutations** of the LR input burst of images, since the order of the images within the burst does not carry any relevant information for producing images with a finer level of detail. The network takes as input a burst of an arbitrary length of raw LR images, which are assumed to be roughly registered with each other and it estimates **both** the HR image and a corresponding pixel-by-pixel uncertainty value. This is useful for understanding how good is the generated image, as well as improving the overall model performance. As illustrated in figure 3.6, this is achieved by means of two parallel heads in the last part of the architecture that transfer to the two sets of information the features produced by the backbone of the network. It can also be seen how the uncertainty head does not use a global skip connection that carries the residuals that are estimated from bilinear upsampling and by averaging the LR inputs. Moreover, Pixel shuffling is used as the upsampling technique.

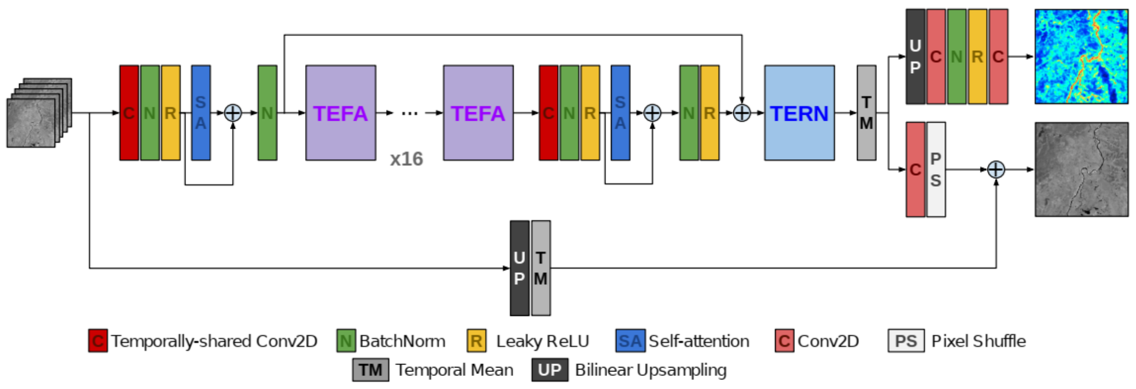


Figure 3.6: PIUnet architecture [12].

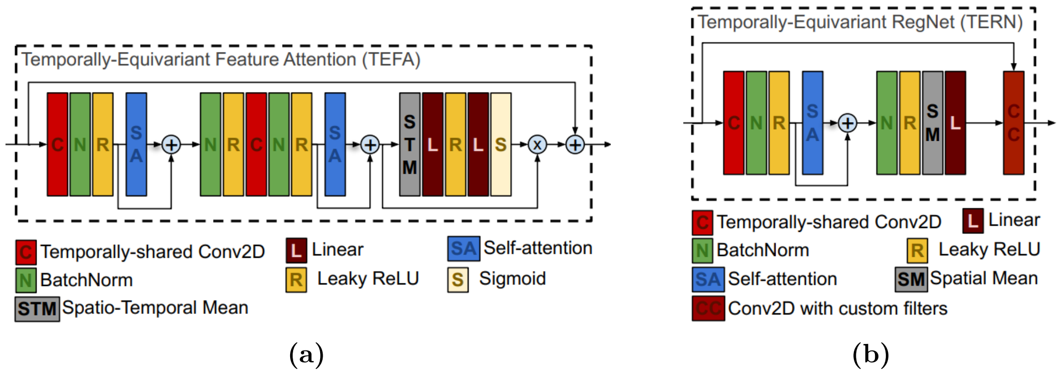

Temporally-Equivariant Feature Attention (TEFA)
C Temporally-shared Conv2D
N BatchNorm
STM Spatio-Temporal Mean
L Linear
R Leaky ReLU
SA Self-attention
S Sigmoid
(a)
Temporally-Equivariant RegNet (TERN)
C Temporally-shared Conv2D
L Linear
N BatchNorm
R Leaky ReLU
SA Self-attention
SM Spatial Mean
CC Conv2D with custom filters
(b)

Figure 3.7: (a) Temporally-equivariant feature extraction (TEFA) block, (b) Temporally-Equivariant RegNet (TERN) block.

As depicted in red in the network architecture in figure 3.8, 2D convolutions are shared across the temporal dimension to extract spatial features that are combined by using self-attention. Indeed, the use of **self-attention** in the temporal dimension is proposed by the authors since it constitutes a permutation-equivariant operation that allows for the combination of the information coming from multiple time instants exploiting their cross-correlation. As explained in section 2.2.6, self-attention uses three learnable matrices to project its input feature vector into three different subspaces, namely the key, query, and value vectors. Moreover, the cross-correlation matrix between the key and query is used to compute the weights used inside the weighted sum of the value vectors that generate the output.
In order to maintain **equivariance** to the permutation of the input burst, is essential for the 2D convolutions to be shared along the dimension describing the images composing the burst, namely the temporal dimension. Based on this idea, the authors designed a module shown in 3.7 (a) called Temporally Equivariant Feature Attention (TEFA) to compute classic residual feature attention, as well as a temporally-equivariant extension of the RegNet module called TERN, reported in figure 3.7 (b). Similarly to RegNet, TERN is used to compute a $K \times K$ spatial filter that is different for each image, but it is the same for multiple feature maps. To compute the values of this filter, TERN exploits self-attention to compute the cross-correlation of the

computed features over the temporal dimension, thus obtaining the values. Indeed, the overall architecture is equivariant to temporal permutations from its input to the output of the TERN module, and to make it also invariant, the output of TERN is simply averaged along the temporal axis.
The fact that the model is fully invariant to temporal permutation significantly improves its performance and data efficiency. Moreover, the uncertainty of the super-resolved image correlates with temporal variation in the series, and how quantifying it further improves model performance.

MLB-FuseNet

Multi-Level Burst Fusion Network (MLB-FuseNet) [13] is a neural network architecture for burst SR, that in the same way as PIUnet is capable of extracting features being **invariant** to permutations of the input burst. The network takes as input a burst of T raw registered LR images to generate the corresponding HR image. As depicted in figure 3.8, the architecture is divided into two branches: the upper one processes only one of the input T images, which is taken as the reference one, while the lower branch processes the remaining $T-1$ images all at once in a permutation-invariant way. The features extracted in the bottom branch are used to augment the features extracted in the top branch.

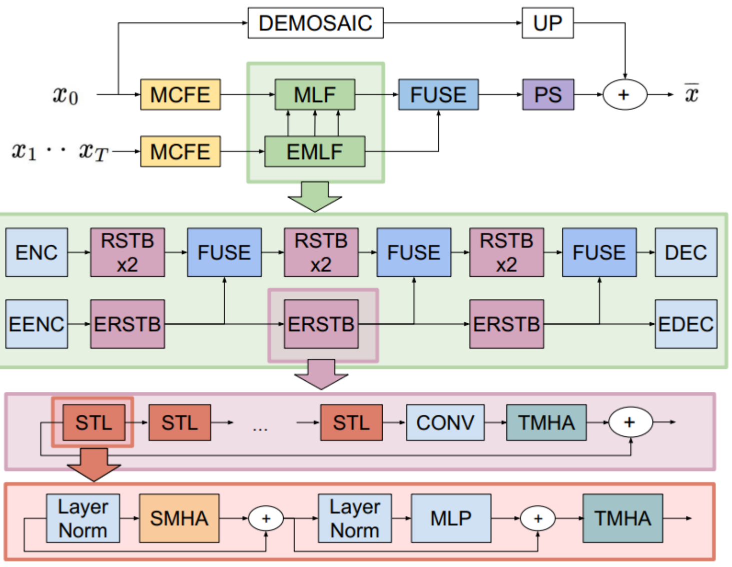


Figure 3.8: FuseNet architecture [13].

As depicted in the previous figure, both branches begin with a **Mosaiced Convolution Feature Extractor** (MCFE), used to extract high-level features without disrupting the Bayer color arrangement of the raw input images that have been previously mosaiced. This is achieved by using a polyphase decomposition where different kernels work with a stride of 2 and with 4 different initial positions, as shown in figure 3.9, ensuring in this way that each kernel only sees a consistent pattern.

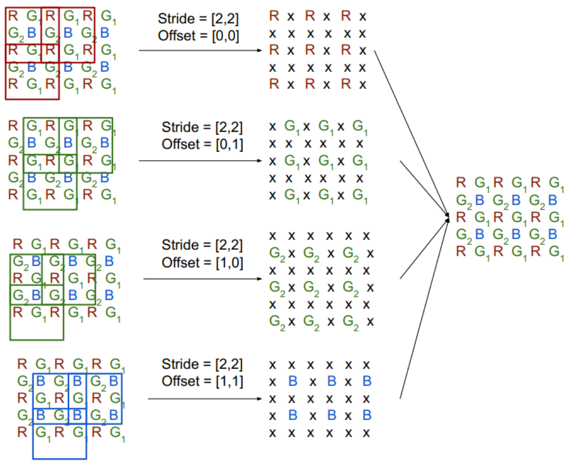


Figure 3.9: MFCE principle

The features estimated in both branches by MCFE are the inputs to two modules that allow to combine them considering the correct spatial position in the original Bayer grid. These two modules are the **Multi-Level-Fusion** (MLF) block in the top branch, and the **Equivariant Multi-Level Fusion** (EMLF) block in the bottom branch, both inspired by the SwinIR architecture [9].
In the lower branch, the equivariant MLF module adapts the SwinIR Deep Feature Extraction block already explained before to a multi-image case. EMLF is used to extract features equivariant to temporal permutation, in order to derive a permutation-invariant representation by averaging in the temporal dimension. To do so, a temporally equivariant encoder (EENC), followed by a sequence of K Equivariant Residual Swin Transformer blocks (ERSTBs) and a temporally equivariant decoder (EDEC) are used. In each ERSTB, the sequence of Swin Transformer Layers (STLs) previously mentioned is used to extract the spatial features of each image by exploiting the Spatial Multi-Head Attention (SMHA). The spatial features of the different images are combined using Multi-Head Attention along the temporal dimension (TMHA). This operation allows mixing the features of the different images along the temporal dimension, in a mathematically equivariant way.
The EENC is defined in a similar way: a sequence of 2D convolutional layers is applied to each frame separately in order to mix up the spatial features obtained by the MFCE and combine the pixels of the Bayer mosaic, then a TMHA combines the features of the different images. Similarly, the EDEC is formed by a sequence of 2D convolutions and a TMHA.
The MLF module in the upper branch of the network is composed of an Encoder (ENC), 2K Residual Swin Transformer blocks (RSTBs), made of a sequence of STLs, alternated with fusion modules (FUSE) and a final Decoder (DEC), made of traditional convolutional layers.
The fusion modules combine the features of the reference frame obtained in the upper branch with the ones obtained in the lower branch from the other images of the burst. The fusion is performed by averaging the features of the lower branch along the temporal dimension and merging, through a 1D convolution after channel concatenation. These fusion modules allow the upper branch to produce a super-resolved image guided by the information extracted from the multiple images in the

lower branch. Multiple stages of the fusion module are used to slowly incorporate the features of the burst at various levels of abstraction. After the MLF module, there is a last fusion block that merges the final outputs of the MLF and EMLF blocks. Finally, the Pixel Shuffle block generates the super-resolved image. There is also a skip connection composed of demosaicing operation and bilinear upsampling so that the network only learns a residual correction.

Chapter 4

Methodology

The aim of this chapter is to explain the work performed to adapt a SISR model into a BSR one. The goal is to enhance the performance in terms of PSNR, to ensure a superior image quality and level of detail while maintaining mobile-friendly characteristics, as well as trying to keep the inference time as low as possible. Several operations will be listed, such as the synthetic generation of a dataset of bursts of RGB and RAW images, as well as a description of the studied fusion and registration techniques, followed by the results obtained, underlining which are the ones that satisfied the objectives. Moreover, a comparison between different networks will be carried out.

4.1 The AI mobile challenge

The **Mobile AI challenge** is a part of the *Mobile AI and AIM Workshops and Challenges* where the participants are asked to develop a lightweight and quantized deep learning model for the single-image super-resolution problem and evaluate it directly on mobile devices by means of an AI Benchmark application [14]. The efficiency of the proposed solutions is evaluated on the Synaptics Dolphin platform featuring a dedicated NPU that can efficiently accelerate INT8 quantized neural networks. The score achieved by each model is computed based on its PSNR and runtime results, thus balancing between the image reconstruction quality and the efficiency of the model.

4.2 NCNet

Among the participants of the Mobile AI 2022 challenge [15], we selected one to start our experiments with, which is the efficient fast Nearest Convolution Network (NCNet) with only 54k parameters designed for real-time single-image super-resolution [16]. It is lightweight enough to be easily deployed on mobile devices after an 8-bit quantization, moreover, it is fully compatible with all major mobile accelerators. Its CNN-based plain network structure is reported in figure 4.1, which also highlights its most important component, the nearest convolution as residual learning architecture. NCNet achieved a remarkable trade-off between performance and inference time.

Moreover, thanks to the exploitation of parallelization, the model's runtime on mobile devices is even faster than the traditional bilinear upsampling.

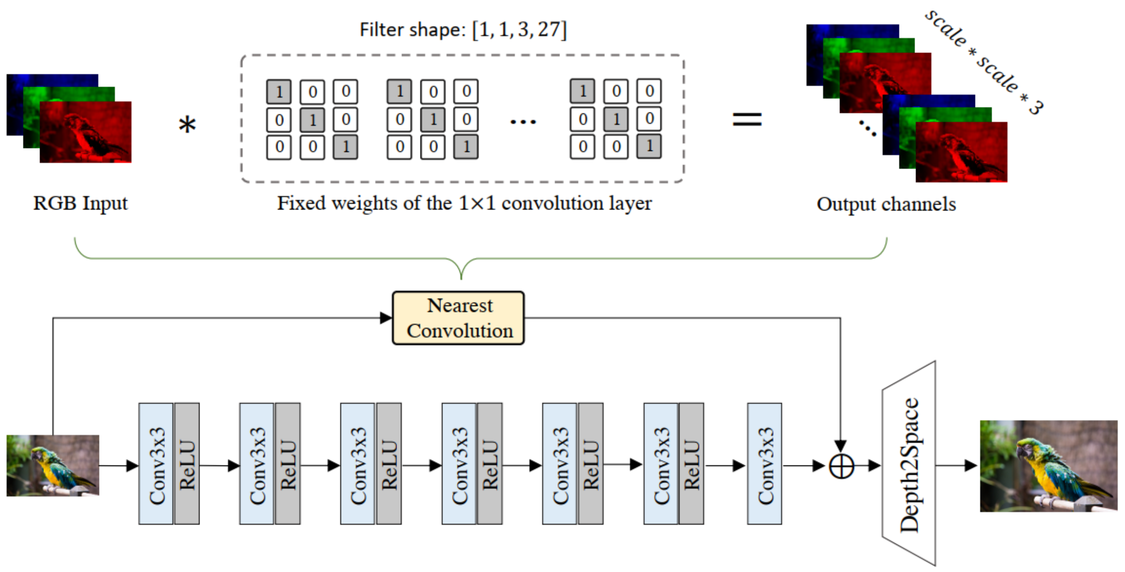


Figure 4.1: NCNet model architecture [16].

As reported before, the plain network is composed of 6 pairs of 3x3 convolutional layers with RELU activation function, followed by a final 3x3 convolutional layer, all of them with a number of feature channels fixed to 32. The output of the sequence of convolutions is then added to the result of the nearest convolution module, right before the features upsampling performed by a depth-to-space operation.
As said before, the core component of the NCNet architecture lies in the **Nearest Convolution module**, similar to the anchor-based residual learning proposed by [10]. It is implemented by a special 1×1 convolution layer with stride 1, characterized by the fact that it is not trainable, meaning that the weights set equal to s^2 groups of 3×3 identity matrix (where s is the upscale factor) and kept fixed during the training process. Each one of the groups previously mentioned is used to create an exact copy of the input RGB image, finally resulting in s^2 copies. These copies are crucial in order to reconstruct the HR image through a depth-to-space operation. This leads to a reconstructed image equal to the one that would be obtained by means of the nearest interpolation, but since the 1×1 nearest convolution exploits parallel computations, the HR image is obtained in a much faster way.

4.2.1 Training replication

As a preliminary step, the NCNet training was replicated using the code available at [17].
The training settings are the same as those reported in [16]. DIV2K (section 4.3.1) is used both as the training set, composed of 800 image pairs, and the test set, which contains 100 image pairs. The scale factor is fixed to 3 and the batch size is set to 64, while the patch size of the LR images is also set to 64x64, thus producing an HR

patch of size 192x192. The Adam optimizer is the selected learning algorithm utilized, with an initial learning rate of 10^{-3} decreased by half every 200,000 iterations, for a total number of iterations set to 500,000. Also, all the weights are initialized using Xavier initializer. The obtained results are listed in the following table:

	Paper	Reproduction
PSNR FP32 (db)	30.21	30.15

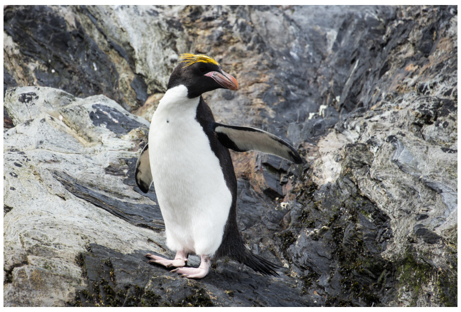
Table 4.1: NCNet SISR training results.

The obtained result refers to the case in which the network parameters have not been quantized to INT8 yet, thus referring to the Floating Point 32 (FP32) case. Even if with a slight difference, the replication allowed to confirm the results reported in the paper, as well as the validity of the NCNet network.
In the table that follows is reported a comparison carried between NCNet and the much more complex SwinIR (section 3.2.1) architecture:

Network	PSNR FP32 [db]	Number of parameters	Inference time [ms]
NCNet	30.15	54k	0.629
SwinIR	34.89	886k	169.04

Table 4.2: NCNet/SwinIR comparison.

As for NCNet, SwinIR has been trained on the DIV2K dataset, and the gap in performance between the two models is evident. However, it can also be seen how NCNet has much fewer parameters than SwinIR, which translates into a much faster inference time. Note that the inference times have been measured have been measured on GPU NVIDIA GeForce RTX 2080 Ti, for a random 3-channel tensor of size 150×150.


SwinIr

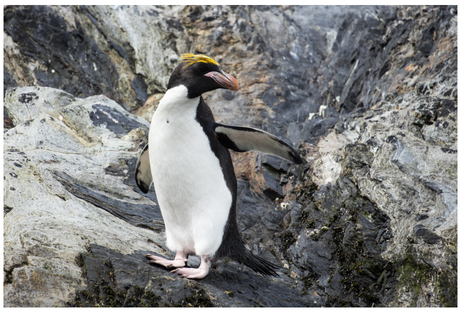
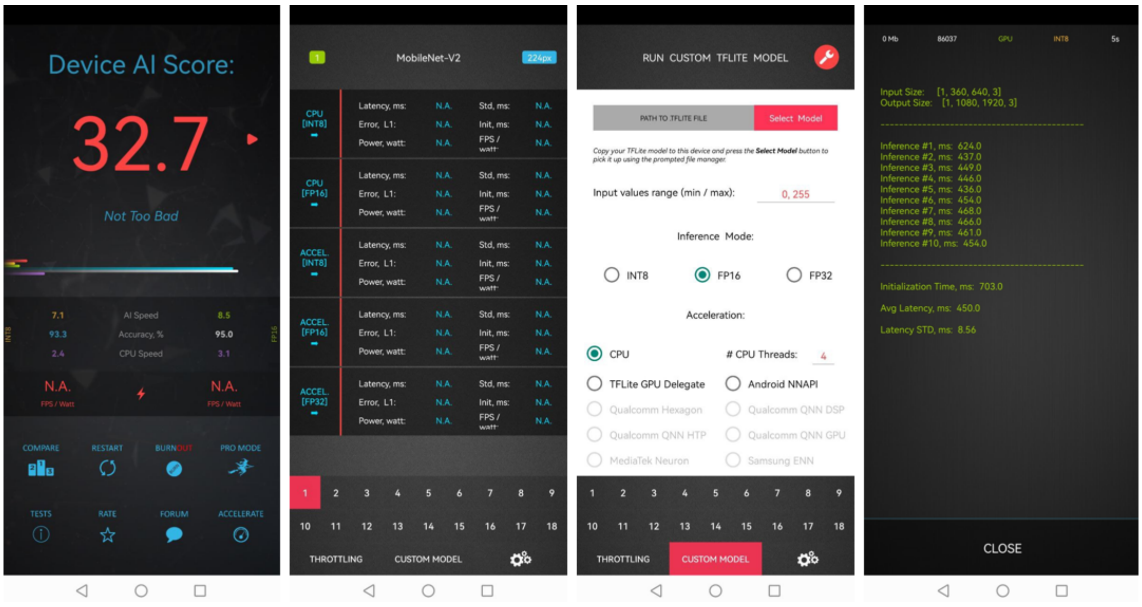

NCNet

Figure 4.2: HR images produced

After the INT8 quantization, the model can effectively be deployed on mobile. Moreover, by means of the mobile *AI benchmark* application its inference time. In figure 4.3 are reported 4 screenshots of the mobile application installed on a Huawei P smart+ 2019 smartphone.



RUN CUSTOM TFLITE MODEL

PATH TO .TFLITE FILE | Select Model

Copy your TFLite model to this device and press the ***Select Model*** *button to pick it up using the prompted file manager.*

Input values range (min / max): 0, 255

Inference Mode: INT8 [✓ FP16] FP32

Acceleration: [✓ CPU] # CPU Threads: 4; TFLite GPU Delegate; Android NNAPI; Qualcomm Hexagon; Qualcomm QNN DSP; Qualcomm QNN HTP; Qualcomm QNN GPU; MediaTek Neuron; Samsung ENN

```
0 Mb        86037       GPU       INT8       5s

Input Size:    [1, 360, 640, 3]
Output Size:   [1, 1080, 1920, 3]
-----------------------------------------
Inference #1, ms:  624.0
Inference #2, ms:  437.0
Inference #3, ms:  449.0
Inference #4, ms:  446.0
Inference #5, ms:  436.0
Inference #6, ms:  454.0
Inference #7, ms:  468.0
Inference #8, ms:  466.0
Inference #9, ms:  461.0
Inference #10, ms:  454.0
-----------------------------------------
Initialization Time, ms:  703.0

Avg Latency, ms:  450.0

Latency STD, ms:  8.56
```

Figure 4.3: Mobile AI benchmark application.

The mobile inference time has been measured on a 3-channel input of size 360×640 and resulted in an average value of 450 ms, a value considerably larger than the one achieved on a desktop GPU.

4.3 Dataset synthetization

Another preliminary fundamental step for the training of a BSR model is to retrieve a dataset in which a burst of LR images univocally corresponds to a ground truth image. Since there are few existing real-world datasets where an HR image corresponds to a burst of LR images, it will be generated synthetically.
For what concerns the training set, the burst of images will be synthesized on the fly, just before entering the network. This is done to let the model train on images that always have different displacements between each other, thus resulting in a greater number of information that it can use to learn the mapping between LR and HR. On the other hand, the validation set that will be used to measure the PSNR performance of the model will be pre-generated, thus keeping the displacements between the images fixed. The DIV2K dataset and The Zurich RAW-to-RGB are the datasets chosen to perform this process.

4.3.1 DIV2K dataset

DIVerse 2K resolution (DIV2K) image dataset was proposed for the first time during the New Trends in Image Restoration and Enhancement (NTIRE) 2017 challenge [18]. The dataset was collected by the authors who manually downloaded 1000 RGB images from several websites that freely shared high-quality 2K resolution photos. The images present a **small amounts of noise** and other corruption, depicting a large diversity of contents, ranging from people, handmade objects, and environments, to flora and fauna, and natural sceneries including underwater and dim light conditions. Since the most common upsampling factors in the image super-resolution literature are of $\times 2$, $\times 3$, and $\times 4$, all the images were cropped to a multiple of 12 pixels on both axes. After the 1000 images were gathered, they were randomly partitioned into 800 for the training set, 100 for the validation set, and finally 100 for the test set, trying to maintain a good balance firstly in visual contents and then on the average entropy.

4.3.2 Zurich RAW-to-RGB dataset

Zurich RAW to RGB is a large-scale dataset consisting of 20 thousand photos that were collected using both a professional Canon camera and a Huawei P20 smartphone for capturing RAW photos, as well as the resulting RGB images, captured in a variety of places with different illumination and weather conditions [19]. It is composed of 48043 RAW-RGB image pairs, respectively of size $448 \times 448 \times 1$ and $448 \times 448 \times 3$, which are extracted from images that are preliminarily matched, using a non-overlapping sliding window. The images have then been split into 46.8K for training and 1.2K for the validation set. Moreover, RAW image patches are additionally reshaped into $224 \times 224 \times 4$, where the four channels correspond to the four colors of the RGBG Bayer filter.

4.3.3 Synthetic RGB burst generation

The pipeline followed to synthetically generate the burst of images is based on the code available at [20]. Originally, the code was conceived for producing a burst of raw images starting from a single RGB image. However, in this first part of the work, it has been modified in order to produce, instead, a burst of RGB images given the input one, with no explicit addition of elements like noise.
A burst of a given **length** is generated by applying random transformations to a given HR input image, and the result is downsampled by a given **downsample factor**. After having initialized the parameters the procedure is the one that follows: firstly the image pixel values are scaled to the range [0, 255], in the case they were ranging from 0 to 1, then a sampling grid is created for the affine transformation, and it is initialized with the input image dimensions. For the first image of the burst that has to be created, no random transformations are applied except for a translation to center the sampling grid. For what concerns the subsequent images in the burst, random transformations that include translation, rotation, shear, scale, and aspect ratio changes are applied. This is achieved by relying on the *OpenCV* **warpAffine** function, which applies the affine transformation to the image, defined by an affine transformation matrix that is generated based on the sampled transformation parameters. Finally, a border crop is applied to the transformed image, which is then downsampled based on the downsampling factor. Also, the inverse transformation matrix and the inverse transformation of the sampling grid are calculated to track pixel positions.
The resulting image is added to the list of burst images, which after being completed is stacked into a single tensor.

4.4 Training details

Once the dataset has been prepared the training process can be carried out. Note that the baseline for the training code has been provided by [21], which is based on the usage of the *Pytorch* python library. Thus, this will be the preferred library that will be used for the experiments that will follow. The same goes for the parameters of the training phase that are reported in the following:

- The training set of DIV2K is used to synthetically generate on the fly a burst of arbitrary length given the ground truth HR image;
- The validation set is pre-generated, for a total of 100 burst-hr images pairs, used to compute the PSNR;
- **Scale factor**: 3;
- **Batch size**: 64;
- **Patch size**: 64;
- **Total iterations**: 35k (2500 epochs);
- **Loss funtion**: L1-loss;
- **Training algorithm**: Adam optimizer, **inital learning rate**: 10^{-3}, halved every 15k iterations.

4.4.1 Resize problem

The process explained 4.3.3 involves a cropping operation and the use of a resizer for downsampling the image different from the one originally used by the authors who created the dataset in the first place. This translates into a change in the performance of the model:

	Original DIV2K	Resized DIV2K
PSNR (db)	30.21	28.01

Table 4.3: NCNet SISR training results.

Also, the lesser number of pixels available to the model for learning the mapping from LR to HR results in a worsening of the performance. Note that this case of SISR is the one that will be taken into account in the following.

4.5 Fusion techniques

The first step for the reconstruction of a single HR image from the input burst is to utilize a fusion technique, to amalgamate information coming from multiple images effectively, without relying on any explicit registration between the images. Thus, the network has been modified as depicted in figure 4.4, where the residual

is computed using only the first 3 input channels, which corresponds to the image taken as reference, that in this specific case also corresponds to the only image of the input burst that is not modified by any transformation. Moreover, it can be seen how the fusion step counts as a **preprocessing** step for network input data since this operation is performed *off-line*.
The goal of this section is to carry out the comparative analysis of the two fundamental fusion techniques that were taken under study: **early fusion** and **slow fusion**.

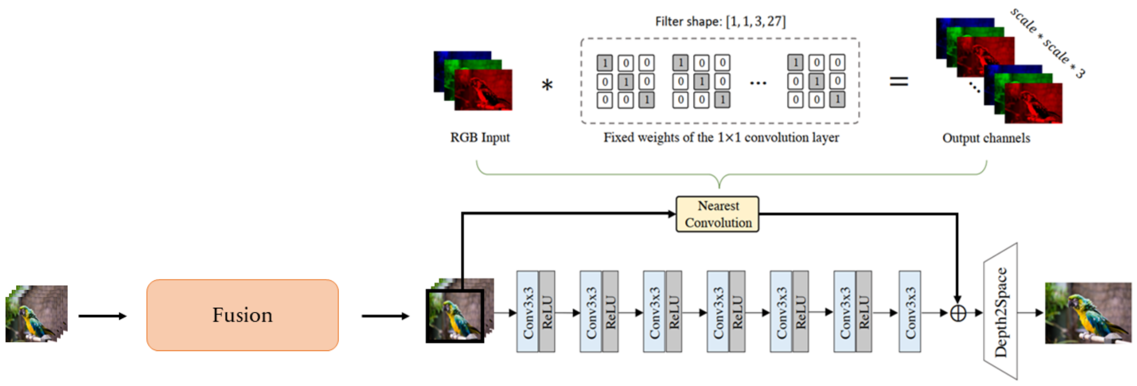


Figure 4.4: Modified NCNet architecture.

4.5.1 Early Fusion

One of the most straightforward techniques for the processing of multiple images inside a CNN is to adapt the temporal depth of the network input layer to the number of input images, where the temporal depth D_l represents the actual number of images that constitute the burst [22]. This will enable the consolidation of all temporal information within the initial layer, allowing all the remaining information within the network to remain identical to the SISR case. In our specific case, the images, having size $3 \times H \times W$, are concatenated along the channel dimension, resulting in an input tensor to the network of size $3 \cdot D \times H \times W$. A visual representation of this operation is depicted in figure 4.5.

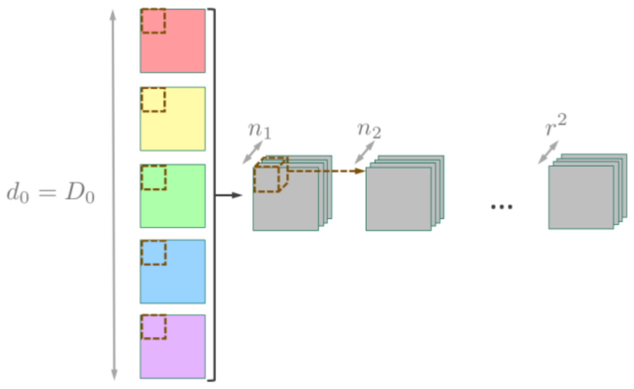


Figure 4.5: Early fusion visual representation [22].

4.5.2 Slow Fusion

An alternative to the early fusion operation is to partially merge temporal information in a hierarchical structure. Figure 4.6 shows how the burst of images is slowly fused into a single tensor that is the input to the network. As for the early fusion case, also with slow fusion the images are concatenated along the channel dimension, but in this case, the operation is performed considering just a pair of images, and this is iterated until a single tensor is created. This results in a much greater quantity of information that is placed as input to the network.

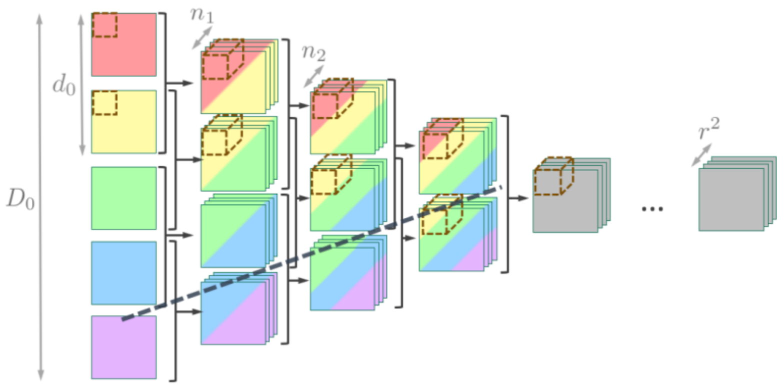


Figure 4.6: Slow fusion visual representation [22].

Note that early fusion is a special case of slow fusion, where instead of concatenating the images in pairs, all of them are processed at once.

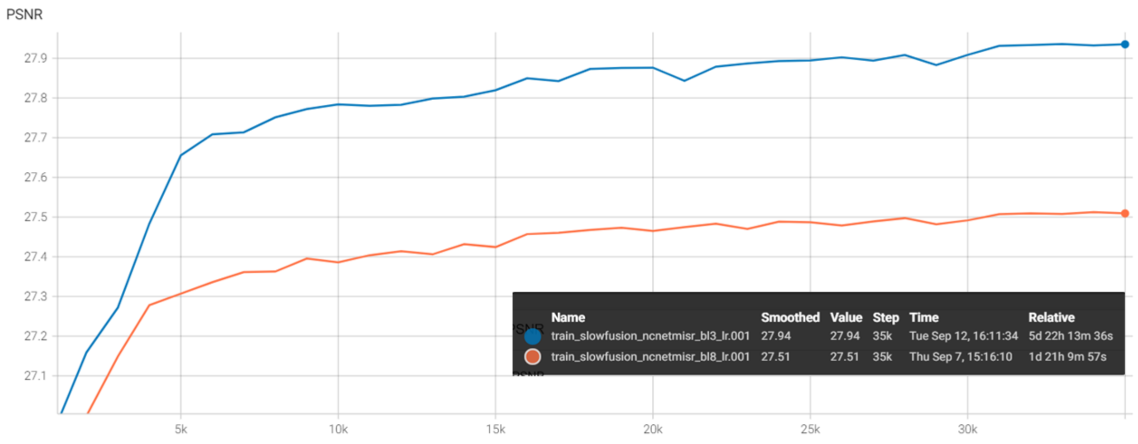
4.5.3 Results

In this subsection are reported the results of the NCNet training done with the specifics reported in section 4.4 for both the early and slow fusion cases, considering bursts composed of 3, 8, and 16 images. In the following will be also reported the obtained inference times, measured on a GPU NVIDIA GeForce RTX 2080 Ti, for an input of size (3· burst length ×150 × 150).

	PSNR FP32 [dB]	Inference time GPU [ms]
Burst size = 3	27.94	0.075
Burst size = 8	27.51	0.438

Table 4.4: Slow Fusion technique results.

The results for a burst size of 16 are missing since the process is killed due to the excessive dimensionality reached by the fused images.

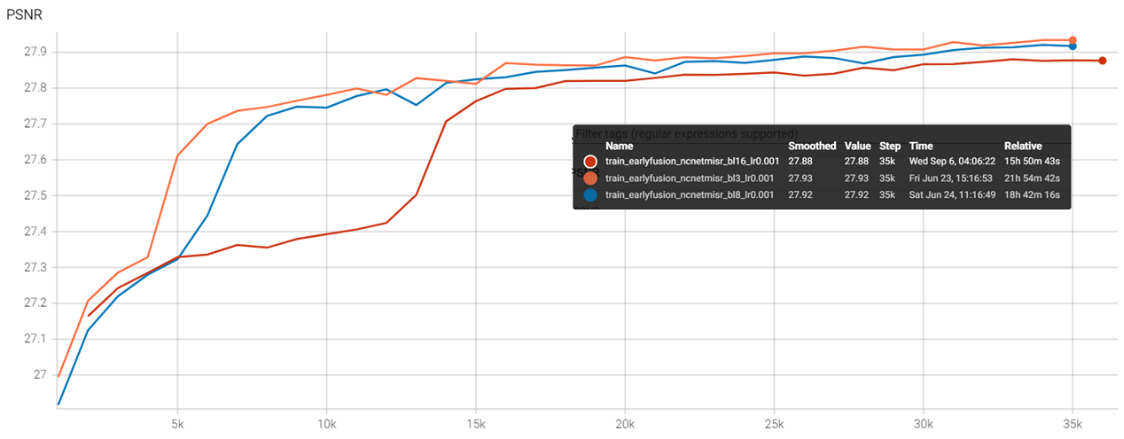


	Name	Smoothed	Value	Step	Time	Relative
🔵	train_slowfusion_ncnetmisr_bl3_lr.001	27.94	27.94	35k	Tue Sep 12, 16:11:34	5d 22h 13m 36s
🟠	train_slowfusion_ncnetmisr_bl8_lr.001	27.51	27.51	35k	Thu Sep 7, 15:16:10	1d 21h 9m 57s

Figure 4.7: Slow fusion Tensorboard log.

	PSNR FP 32 [dB]	Inference time GPU [ms]
Burst size = 3	27.93	0.102
Burst size = 8	27.92	0.238
Burst size = 16	27.88	0.283

Table 4.5: Early Fusion technique results.



	Name	Smoothed	Value	Step	Time	Relative
🔴	train_earlyfusion_ncnetmisr_bl16_lr0.001	27.88	27.88	35k	Wed Sep 6, 04:06:22	15h 50m 43s
🟠	train_earlyfusion_ncnetmisr_bl3_lr0.001	27.93	27.93	35k	Fri Jun 23, 15:16:53	21h 54m 42s
🔵	train_earlyfusion_ncnetmisr_bl8_lr0.001	27.92	27.92	35k	Sat Jun 24, 11:16:49	18h 42m 16s

Figure 4.8: Early fusion Tensorboard log.

The first important result that emerges is that **none** of the different combinations of fusion technique and burst size gave a PSNR score higher than the single-image case of table 4.3. This can be traced back to the absence of the registration between the images, therefore, not having filled the displacements between the images does not allow the network to reconstruct a better image than the single image case. The increase in the amount of information available for the network has not led to an improvement, but rather to a worsening due to the fact that the images are poorly correlated with each other. These results highlight the **need** of registration when working with multiple images.
Comparing the two fusion techniques, early fusion resulted in being the fastest fusion

technique, being also lighter than the slow fusion case. Moreover, the obtained PSNRs are quite similar, except for the case of burst size = 8. The additional complexity of slow fusion did not lead to results that were significantly superior to those of early fusion, being even worse and slower in one case, and even not allowing to work with a burst of size 16. Since faster and lighter techniques are preferred for his work, in the following **early fusion** will be kept as the preferred fusion technique.

4.6 Registration techniques

In real-world scenarios, objects may undergo complex motions, and optical flow estimation techniques are designed to handle such complex motions. As the last section highlighted, also in the architecture under study, a registration of the burst images is needed. This process improves the overall quality of the super-resolved image and makes it free from motion artifacts, and can be divided into two steps:

1. **Optical flow estimation**: estimation of a 2-channels vector of the same size as the input images, which quantifies the misalignments that the depicted objects may undergo in the two different images;
2. **Warping phase**: the motion in the images is compensated by means of a warping operation that uses the optical flow previously estimated, thus aligning the images and ensuring temporal consistency.

Optical flow estimation can provide sub-pixel accuracy in estimating motion vectors, which is essential for aligning frames accurately.
The architecture that results can be summarized by figure 4.9, where the NCNet architecture remains unchanged, but the preprocessing also includes the registration performed before the early fusion strategy. As for the fusion, the alignment is performed *off-line*.

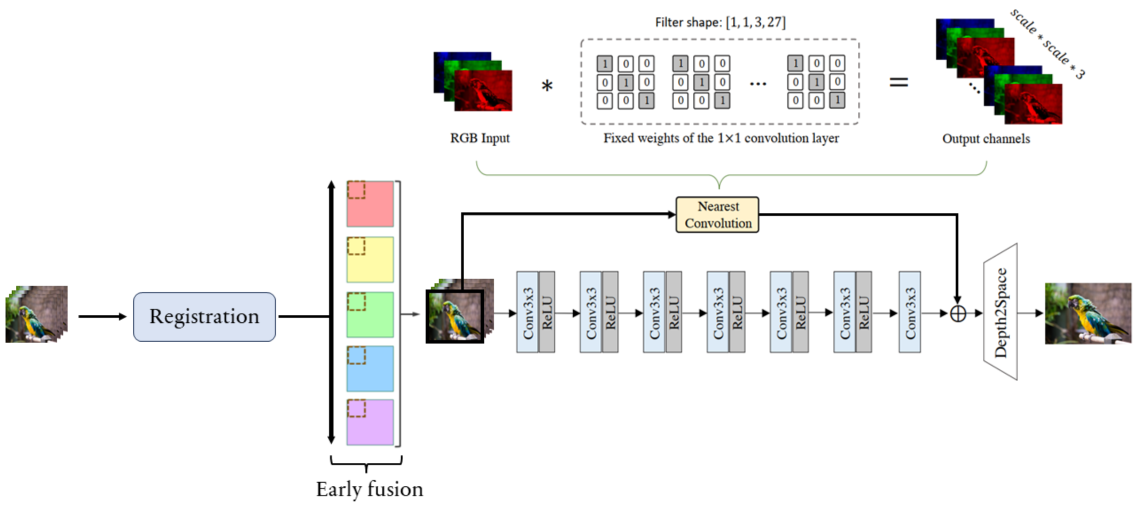


Figure 4.9: Final architecture.

The aim of this section is to introduce the different registration strategies analyzed, employing both neural networks and traditional algorithms, as well as comparing the results achieved.

Warping phase

Before describing the different ways in which the optical flow is estimated, here is listed the warp function used to align the images using the estimated optical flows. It has been extracted from the code available at [20], while in figures 4.10 and 4.11 are reported two examples of registered and not registered images. They highlight how, once the images are aligned, the border are padded with zeros, thus resulting in black outlines surrounding the image.

```python
def warp(feat, flow, mode='bilinear', padding_mode='zeros'):
    """
    warp an image/tensor (im2) back to im1, according to the optical flow im1 --> im2

    input flow must be in format (x, y) at every pixel
    feat: [B, C, H, W] (im2)
    flow: [B, 2, H, W] flow (x, y)

    """
    B, C, H, W = feat.size()

    # mesh grid
    rowv, colv = torch.meshgrid([torch.arange(0.5, H + 0.5), torch.arange(0.5, W + 0.5)])
    grid = torch.stack((colv, rowv), dim=0).unsqueeze(0).float().to(feat.device)
    grid = grid + flow

    # scale grid to [-1,1]
    grid_norm_c = 2.0 * grid[:, 0] / W - 1.0
    grid_norm_r = 2.0 * grid[:, 1] / H - 1.0

    grid_norm = torch.stack((grid_norm_c, grid_norm_r), dim=1)

    grid_norm = grid_norm.permute(0, 2, 3, 1)

    output = F.grid_sample(feat, grid_norm, mode=mode, padding_mode=padding_mode)

    return output
```

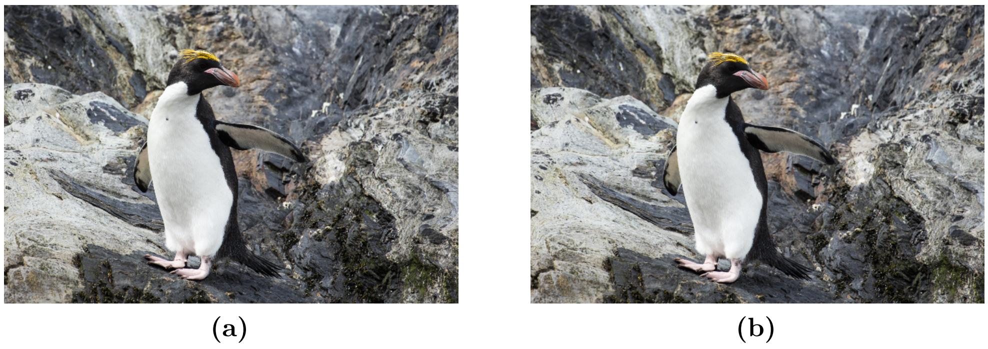


Figure 4.10: Not registered images.

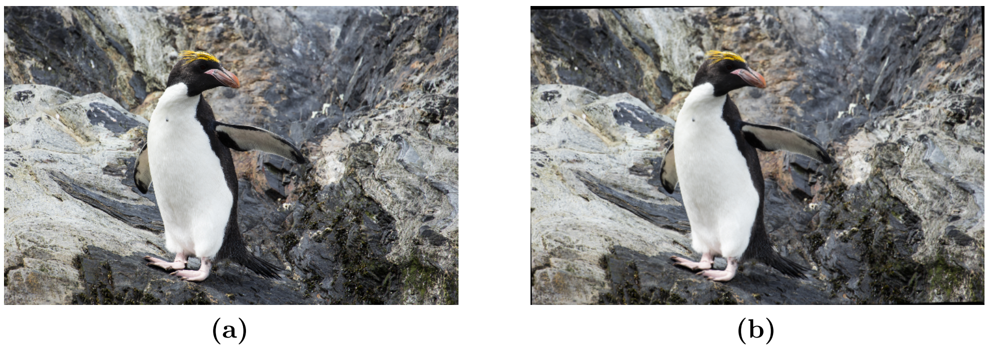


Figure 4.11: Registered images.

4.6.1 PWCNet

The first optical flow estimator used is PWCNet, a compact CNN model using principles like pyramidal processing, warping, and the use of a cost volume, developed by Sun *et al* [23], the architecture of which is depicted in figure 4.12.

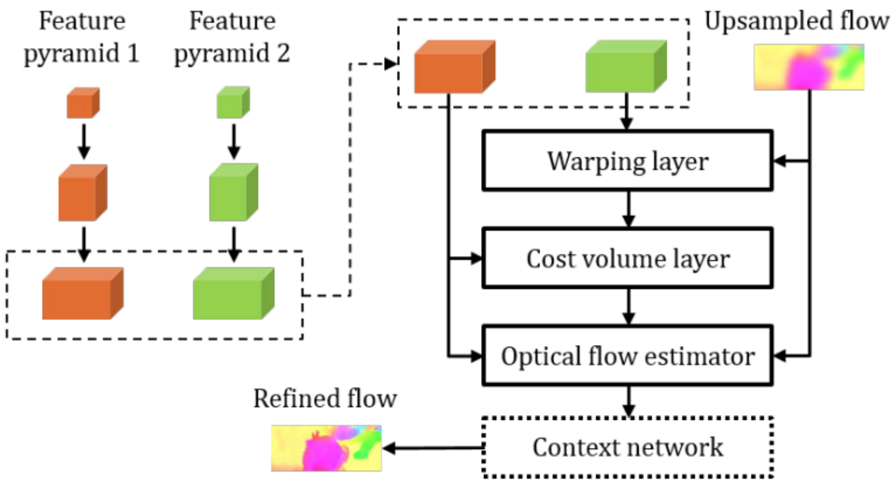
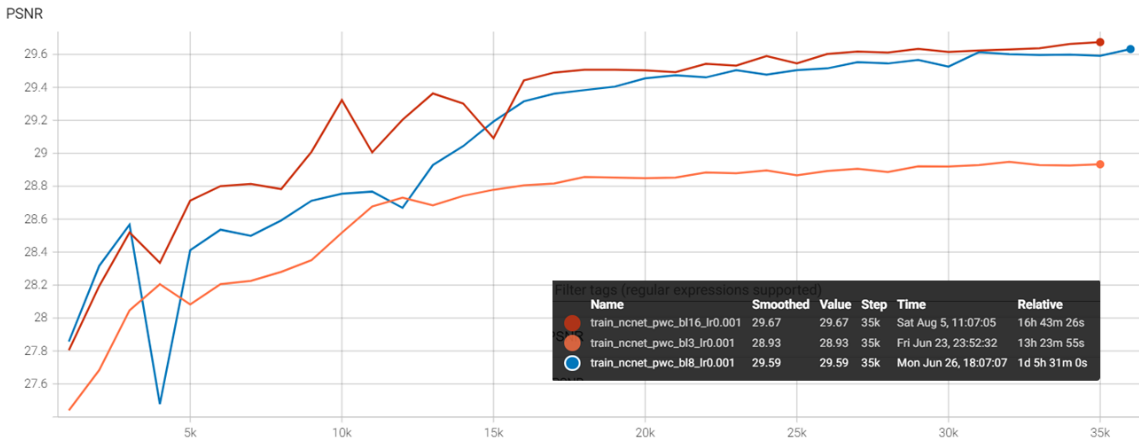


Figure 4.12: PWCNet architecture [23].

The first step for the optical flow estimation when provided with two input images is the creation of **pyramids** by the neural network, namely a hierarchy of L levels of feature representations. The lowest level of the hierarchy corresponds to the input images, and by sequentially applying Convolutional operations the features are downsampled till reaching the l-*th* layer.
After the pyramids' creation, inside the **warping layer**, at the l-*th* level, the $\times 2$ upsampled flow from the l+1-*th* level is used to warp the features of the second image toward the first image, using bilinear interpolation.
The **cost volume** layer uses the previously constructed features to generate a cost volume, which takes track of the matching costs between pixels in the current image and their counterparts in the other image. Specifically, the matching cost is determined by measuring the correlation between features extracted from the first image and the corresponding ones obtained from the second image after the warping operation has been performed.
The previously computed cost volume, the pyramidal features of the first image, and the upsampled optical flow from the l+1-*th* level are the inputs to the **optical flow estimator** layer. This layer is a multi-layer CNN responsible for producing the optical flow at the l-*th* pyramid level. The number of feature channels remains consistent across all pyramid levels, with each level's estimator having its unique set of parameters rather than goes on until it reaches the desired level.
Finally, the output and the features produced by the optical flow estimator layer are the inputs to a sub-network known as the **Context network**, used to produce the refined flow. The Context network itself is a feed-forward CNN consisting of seven 3×3 convolutional layers, each utilizing a different dilation constant. Using given a dilation constant k means that the inputs to a filter in the layer are k units apart in both the vertical and horizontal directions.
After training NCNet with the same settings already explained in section 4.4 and the addition of the PWCNet architecture, the results that follow are obtained:



	Name	Smoothed	Value	Step	Time	Relative
●	train_ncnet_pwc_bl16_lr0.001	29.67	29.67	35k	Sat Aug 5, 11:07:05	16h 43m 26s
●	train_ncnet_pwc_bl3_lr0.001	28.93	28.93	35k	Fri Jun 23, 23:52:32	13h 23m 55s
●	train_ncnet_pwc_bl8_lr0.001	29.59	29.59	35k	Mon Jun 26, 18:07:07	1d 5h 31m 0s

Figure 4.13: PWCNet Tensorboard log.

	PSNR FP32 [dB]
burst length = 3	28.93
burst length = 8	29.67
burst length = 16	29.67

Table 4.6: PWCNet training results.

4.6.2 FastFlowNet

The second alternative for fast optical flow prediction is FastFlowNet, a lightweight model that follows the widely-used coarse-to-fine paradigm like PWCNet, to estimate a gradually refined optical flow. However, differently from PWCNet, FastFlowNet introduced some innovations to reduce model size, thus reducing parameters and computation costs.
Indeed, as depicted in figure 4.14, the core components of the architecture are the **"Head Enhanced Pooling Pyramid"** (HEPP), which substitutes PWCNet's dual convolution feature pyramid for enhancing pyramid features of high-resolution, the **"Center Dense Dilated Correlation"** (CDDC), proposed for computing cost volume in a lightweight way, even for the cases in which large search radius are employed, and finally the **"Shuffle Block Decoder"** (SBD), employed to compute the optical flow at each level with significantly cheaper computation.

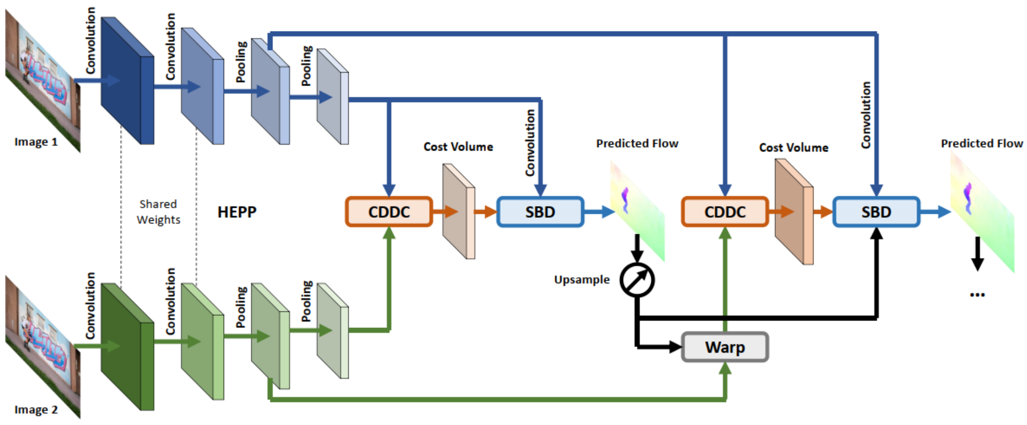
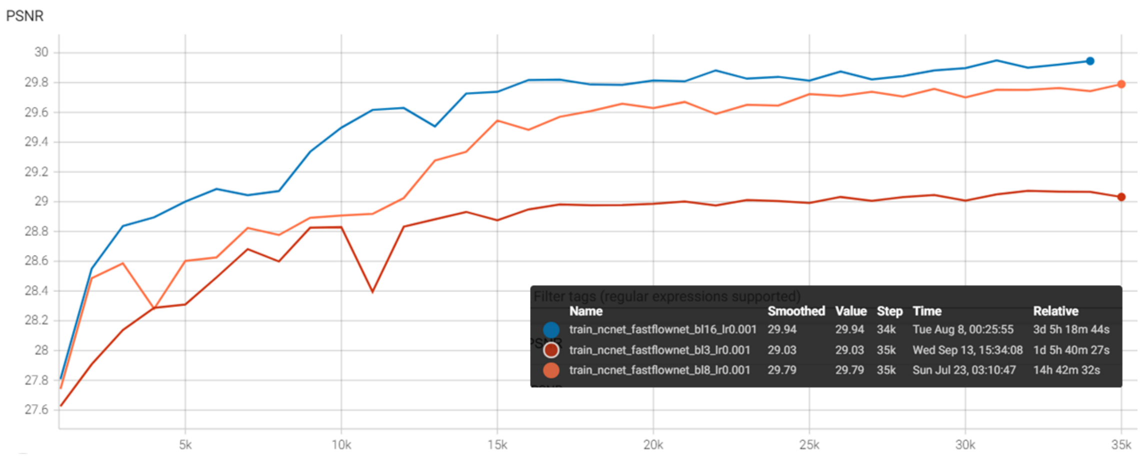


Figure 4.14: FastFlowNet architecture [24].

HEPP is a module able to extract features initially using convolutions and later pooling layers. Convolutions are also employed to further refine pyramid features at a minimal computational cost. This approach results in the creation of six pyramid levels, each time downsampled by a scale factor of 2, effectively distributing computation across the different levels.
As PWCNet, also in this case bilinear interpolation-based warping is used to warp the features of the second input image based on a previously computed flow, upsampled by 2x. This warping significantly reduces displacement caused by large motions, thus reducing the search region and simplifying the estimation of small residual flows.
The CDDC layer allows reducing computation when computing large cost volumes, by downsampling grid points in areas with significant motion, and densely sampling

search grids around the center. It outputs 53 feature channels, motivated by the fact that the residual flow distribution primarily focuses on small motions.
Thanks to the compact cost volume generated by CDDC, the maximum feature channels in the decoder network can be reduced with respect to PWCNet without encountering issues. To further decrease computational costs, a group convolution followed by SBD is used. Each decoder network belonging to SBD includes three shuffle blocks with a group size of 3, efficiently reducing computation with only marginal drops in accuracy.
After completing the training with FastFlowNet (code available at [25]) the obtained results are the ones that follow.



	Name	Smoothed	Value	Step	Time	Relative
●	train_ncnet_fastflownet_bl16_lr0.001	29.94	29.94	34k	Tue Aug 8, 00:25:55	3d 5h 18m 44s
●	train_ncnet_fastflownet_bl3_lr0.001	29.03	29.03	35k	Wed Sep 13, 15:34:08	1d 5h 40m 27s
●	train_ncnet_fastflownet_bl8_lr0.001	29.79	29.79	35k	Sun Jul 23, 03:10:47	14h 42m 32s

Figure 4.15: Fastflownet tensorboard log.

	PSNR FP32 [dB]
burst length = 3	29.03
burst length = 8	29.79
burst length = 16	29.94

Table 4.7: Fasflownet training result.

4.6.3 Handheld Multi-Frame Super-Resolution algorithm

The method "Handheld Multi-Frame Super-Resolution algorithm" by Wronski et al. [26] is the one used in the **Google Pixel 3** smartphone to produce a single high-quality picture starting from a raw burst of photographs captured by the smartphone camera.
In this work the implementation done by Lafentre *et al.* [27] will be tested for estimating the optical flow without relying on neural networks. Their implementation is mainly composed of 3 phases:

1. **Pre-processing phase**: converts raw images into grayscale;
2. **Registration stage**: in this first step optical flows between the reference image and each of the remaining images within the input burst are predicted;
3. **Accumulation stage**: the second step produces the HR image, by merging together burst images exploiting the optical flows previously computed.

For the scope of this thesis work, only the registration process listed in algorithm 2, as well as the pre-processing step, will be used. For the sake of completeness, the accumulation stage will be detailed in the Appendix section. The pre-processing step exploits a low-pass filter to filter out the frequencies of the images above $\frac{\pi}{2}$, to convert the raw images into grayscale images with the same size.

Algorithm 2: Registration algorithm [27]

```
Data: Reference $J_1$, Moving $J_n$, tile size $T$, reference radius $R$
Result: Patchwise flow $V_n = \{V_n^{(1)}, V_n^{(2)}, ..., V_n^{(P)}\}$
/*Compute a high-resolution version of the images                 */
$G_1 \leftarrow ComputeGrayscaleImage(J_1)$;
$G_n \leftarrow ComputeGrayscaleImage(J_n)$;
/*Decompose the reference image into a collection of P tiles.
  $P_1$ refers to the non-overlapping patches of size $T \times T$ that
  are a partition of $J_1$.                                         */
$P_1 \leftarrow ToTiles(G_1, T)$;
$p \leftarrow 1$;
/*For each tile, predict the local parametric flow                 */
for $p_1 \in P_1$ do
    /*Predicts a patchwise subpixel translation with BM             */
    $V_n^{(p)} \leftarrow MultiScaleBlockMatching(p_1, G_n)$;
    /*Predicts a refined patchwise subpixel translation with 3
      ICA iterations.                                               */
    $i \leftarrow 0$;
    while $i<3$ do
        $i \leftarrow 0$;
        $V_n^{(p)} \leftarrow ICA(p_1, G_n, V_n^{(p)})$;
    end
    $p \leftarrow p + 1$;
end
```

Registration stage

The registration step has the goal of estimating a sub-pixel optical flow to align all the images composing the burst over a reference one chosen among them. To do so, each one of the images is partitioned into patches and the optical flows are estimated between the corresponding patches. This is achieved by relying, again, on a **two-stage** strategy: first, a coarse flow with a **multi-scale block-matching** (BM) algorithm is predicted, which is subsequently used as an initialization for the **inverse compositional algorithm** (ICA), which is a variant of the Lucas-Kanade (LK) algorithm [28], that is iterated three times in order to reach a satisfying refined flow with sub-pixel accuracy.

Multi-scale BM algorithm

The **multi-scale BM algorithm** (algorithm 3) roughly estimates the optical flow between corresponding patches of the two different images, by using a coarse-to-fine approach on a 4-level Gaussian pyramid.
At each pyramid level, patch-wise misalignment are estimated and then used to initialize the next level. Formally, at a given level of the pyramid and for a given pixel location in the reference image, corresponding patches are extracted from both the reference image and the second image, in such a way that the tile of the second image has a larger dimension, that depends on the value of the search radius R. At the highest level, for a given patch of the reference image, the optical flow is initialized by considering the 3 closest tiles in the second image, among which, the first patch is fully contained by construction.
Once the flow has been initialized, the flows of each of the 3 tiles are separately computed, and to each of them is associated the value of l_1 error between p_1 and the matching position in the patch for which the error is being computed. The flow that minimizes the error is the one chosen as the candidate for the next level. In this way, at each level are contained all the motions estimated up to that level plus the new information computed by the latter.

Inverse Compositional Algorithm

The result of the BM algorithm previously explained is an optical flow precise up to one pixel, whereas the usage of the Inverse Compositional Algorithm (ICA) (algorithm 4) aims to reach sub-pixel accuracy.
The ICA method solves the problem of finding a new flow estimate as the sum of the BM initial guess of the optical flow and the subpixel refinement estimated by ICA for each tile returned by the BM algorithm. The subpixel refinement is obtained after 3 ICA iterations are performed on each one of the input patches. It is necessary to stop the algorithm if a maximum number of iterations are reached, since the method may converge very slowly to the solution.
The input parameters of the algorithm are the two images, an initial approximation of the flow p, and a small constant ϵ, that is used to stop the iterative process, according to the convergence criterion $||\Delta p|| < \epsilon$. It follows a coarse-to-fine strategy by creating

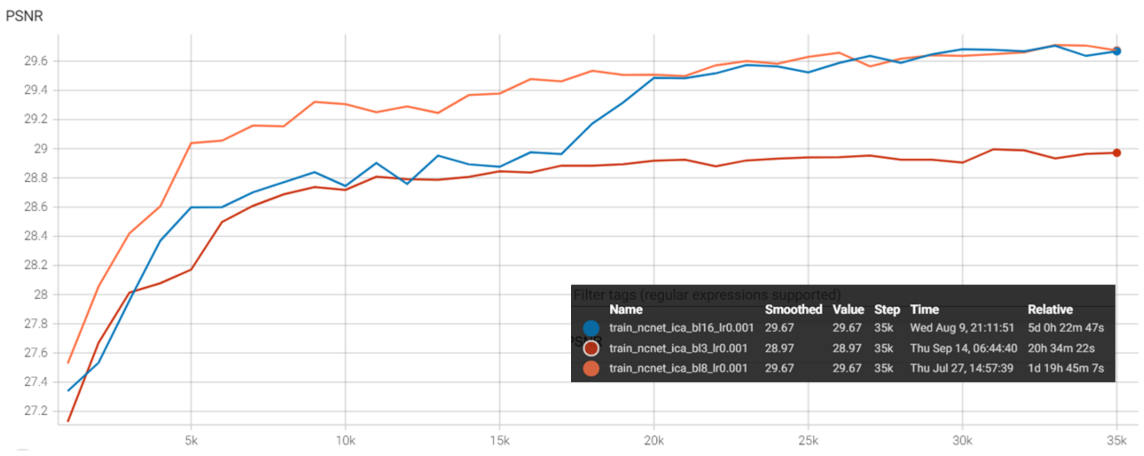
Algorithm 3: Multi Scale BM algorithm [29]

```
Data: $\{I_0, ..., I_{N-1}\}$ burst of $N$ downsampled grayscale images, where $I_0$ is
      the reference image
Data: $\{p_0, ..., p_3\} \in \{1, 2\}^4$ norm power at each pyramid level
Result: Set of aligned tiles at reference image location $(i, j)$
for $k \in \{0, ..., N-1\}$ do
    Compute $G_k^l, l \in \{0, ...,3\}$
end
for $k \in \{0, ..., N-1\}$ do
    $(u_{-1}, v_{-1})(i, j) \leftarrow (0,0)\forall i, j$ ;
    for pyramid level $\in \{0, ...,3\}$ do
        Divide $G_l^0$ in equally spaced tiles;
        if $l > 0$ then
            Upsample the previous level alignments $(u_{l1}; v_{l1})(i, j), \forall i, j$
        end
        for reference tile $T_0^l$ at location (i, j) do
            if $l > 0$ then
                Update the upsampled $(u_{l-1}, v_{l-1})(i, j)$ by keeping the
                  alignment that minimizes $D_1(u_{l-1}, v_{l-1})$among those of the 3
                  nearest coarse-scale tiles
            end
            Get all possible $(u, v)$ locations in $G_k^l$ of tiles of size $n_l \times n_l$ in a
              search area of size $(n_l + 2r_l) \times (n_l + 2r_l)$ centered around
              $(i + u_{l-1}; j + v_{l-1})$ ;
            Compute all possible distances $D_{pl}(u, v), pl \in \{1,2\}$;
            $||\mu|| < 1$
        end
        $(u_l, v_l)(i, j) \leftarrow (u_{l-1}, v_{l-1})(i, j) + D_{pl}(u, v)$; if $l<3$ then
            Compute subpixel displacement vector $\mu(u_l, v_l)$ if $||\mu|| < 1$ then
                $(u_l, v_l)(i, j) \leftarrow (u_l, v_l)(i, j) + \mu$
            end
        end
    end
    Associate to the reference tile at finest scale $T_0^3$ at location $(i, j)$ the tile
      $T_k^3$ of $I_k$ at location $(i + u_3; j + v_3)$
end
```

a pyramid of images, and at each scale, a variant Lucas Kanade algorithm is used to update the estimation of the optical flow for the next finer scale. Indeed, differently from the latter the Hessian matrix remains constant during the iterations and remains constant during the cycle, thus reducing the computational costs. Moreover, bilinear/bicubic interpolations are used to estimate the neighborhood of the pixel for which the optical flow is being refined.

```
Algorithm 4: Inverse Compositional Algorithm [30]
  Data: $I_1$, $I_2$, p, $\epsilon$
  Result: p
  Compute $\nabla I_1(x)$;
  Compute the Jacobian $J(x)$;
  Compute $\nabla I_1(x) J(x)$;
  Compute the hessian;
  while $||\Delta p < \epsilon||$ do
      Compute $I_2(x'(x;p))$ using bilinear or bicubic interpolation
        Compute $I_2(x'(x;p)) - I_1(x)$
        Compute $\sum_x (\nabla I_1(x) J(x))^T I_2(x'(x;p)) - I_1(x)$
        Solve for $\Delta p$ using equation $x'(x;p) \leftarrow x'(x;p) \circ (x'(x;p))^{-1}$
  end
```

After terminating the training stage with the inclusion of the ICA algorithm, using the code available at [31], as the optical flow estimator the obtained results are the ones that follow:



	Name	Smoothed	Value	Step	Time	Relative
●	train_ncnet_ica_bl16_lr0.001	29.67	29.67	35k	Wed Aug 9, 21:11:51	5d 0h 22m 47s
●	train_ncnet_ica_bl3_lr0.001	28.97	28.97	35k	Thu Sep 14, 06:44:40	20h 34m 22s
●	train_ncnet_ica_bl8_lr0.001	29.67	29.67	35k	Thu Jul 27, 14:57:39	1d 19h 45m 7s

Figure 4.16: ICA Tensorbaord training log.

	PSNR FP32 [dB]
burst length = 3	28.97
burst length = 8	29.71
burst length = 16	29.71

Table 4.8: ICA training results.

4.6.4 OpenCV

The other alternative that does not rely on a neural network for the estimation of the optical flow involves the usage of *OpenCV* python library, specifically the method *cv2.calcOpticalFlowFarneback*. It involves an implementation of the algorithm proposed by Gunnar Fanerback [32], who developed a two-frame motion estimation algorithm based on polynomial expansion.
Given two input images, the first step of the algorithm involves approximating the neighborhood of each pixel x of both images with quadratic polynomials, which can be done using the polynomial expansion transform:

$$x^T A x + b^T x + c \tag{4.1}$$

where A is a symmetric matrix, b is a vector, and c is a scalar. These coefficients are estimated by assigning the highest weights to the neighboring pixels of the analyzed pixel x, and by letting them decrease according to the distance from x.
The result of polynomial expansion is that each neighborhood is approximated by a polynomial. Then the idea is to start analyzing what happens if a polynomial undergoes an ideal translation. Given a certain polynomial $f_1(x)$:

$$f_1(x) = x^T A_1 x + b_1^T x + c_1 \tag{4.2}$$

A new signal $f_2(x)$ can be constructed considering a global displacement d:

$$\begin{aligned} f_2(x) = f_1(x-d) &= (x-d)^T A_1 (x-d) + b_1^T (x-d) + c_1 = \\ &= x^T A_1 x + (b_1 - 2A_1 d)^T x + d^T A_1 d - b_1^T d + c_1 = \\ &= x^T A_2 x + b_2^T x + c_2 \end{aligned} \tag{4.3}$$

The unknown displacement d can be derived by equaling the coefficients, which yields to the following equations:

$$A_2 = A_1 \tag{4.4}$$

$$b_2 = b_1 - 2A_1 d \tag{4.5}$$

$$c_2 = d^T A_1 d - b_1^T d + c_1 \tag{4.6}$$

Thus, leading to:

$$2A_1 d = -(b_2 - b_1) \tag{4.7}$$

$$d = -\frac{1}{2} A_1^{-1} (b_2 - b_1) \tag{4.8}$$

The assumptions that allow reaching the previous expressions are that an entire signal is a single polynomial and a global translation involves the two signals. Even though equation 4.8 can be used for real signals, the assumptions that allow obtaining it are not realistic, these assumptions are then relaxed, thus resulting in the introduction of some errors. The goal is to try and keep these errors as small as possible, by changing the global polynomials with local polynomial approximations, achieving the following expressions:

$$A(x) = \frac{A_1(x) + A_2(x)}{2} \tag{4.9}$$

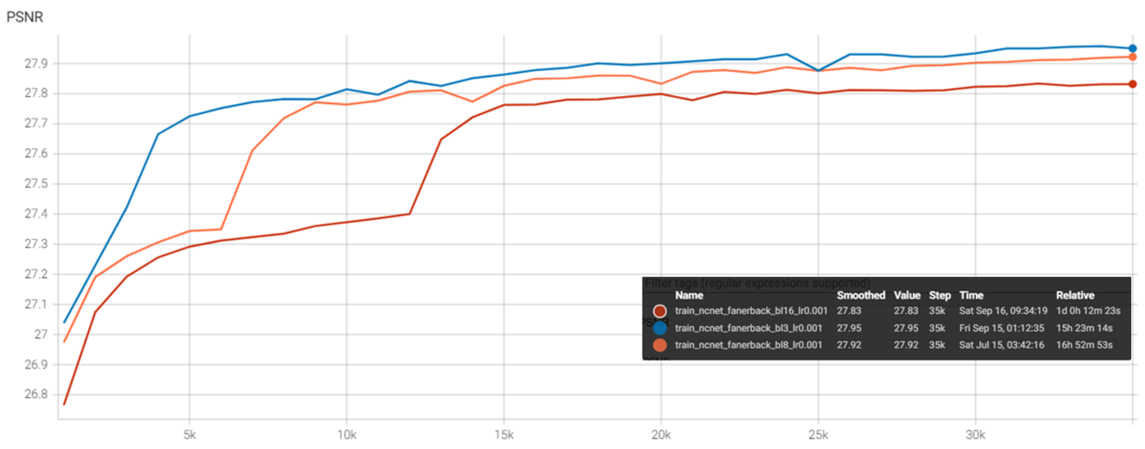
$$\Delta b(x) = -\frac{1}{2}(b_2(x) - b_1(x)) \tag{4.10}$$

$$A(x)d(x) = \Delta b(x) \tag{4.11}$$

where *d(x)* indicates a spatially varying displacement field, not a global displacement anymore. Even if equation 4.11 could be solved for each pixel in the images, the authors proceeded to make the assumption that the unknown displacement field varied slowly over a neighborhood of a pixel, resulting in the possibility of integrating such information. Thus the objective is to find d(x) satisfying as well as possible over a neighborhood of x, or more formally minimizing:

$$\sum_{\Delta x \in I} w(\Delta x)||A(x + \Delta x)d(x) - \Delta b(x + \Delta x)||^2 \tag{4.12}$$

where $w\Delta x$ is a weight function for the points in the neighbors of the considered pixel x.
The algorithm robustness can be improved if the displacement field can be parameterized according to some motion model, which is straightforward for motion models that are linear in their parameters.
Another problem with the method is that the local polynomials are assumed to be at the same coordinates in the two signals that are identical except for the unknown displacement. However, due to the fact that polynomial expansions are local models, these will vary spatially, introducing errors that increase with larger displacements. If prior knowledge about the displacement is known, the polynomial at x of the first image and the polynomial at x+*d(x)* in the second image can be compared, where *d(x)* is the known a priori displacement field. Then the difference between the real displacement and its rounded a priori estimate has to be computed. A more precise a priori estimate means that a smaller difference has to be estimated, thus improving the chances of a good displacement estimate. There are two different approaches to do this: iterative displacement estimation and multi-scale displacement estimation.
After incorporating this technique in the pre-processing step of the NCNet training, the following results are obtained:



	Name	Smoothed	Value	Step	Time	Relative
●	train_ncnet_fanerback_bl16_lr0.001	27.83	27.83	35k	Sat Sep 16, 09:34:19	1d 0h 12m 23s
●	train_ncnet_fanerback_bl3_lr0.001	27.95	27.95	35k	Fri Sep 15, 01:12:35	15h 23m 14s
●	train_ncnet_fanerback_bl8_lr0.001	27.92	27.92	35k	Sat Jul 15, 03:42:16	16h 52m 53s

Figure 4.17: OpenCV Fanerback algorithm tensorboard log.

	PSNR FP32 [dB]
burst length = 3	27.95
burst length = 8	27.92
burst length = 16	27.93

Table 4.9: OpenCV Fanerback algorithm results.

4.6.5 Results

In the following is reported a summary of the obtained results:

	PWCNet	FastFlowNet	ICA	OpenCV
Burst length = 3	28.93	29.03	28.97	27.95
Burst length = 8	29.67	29.79	29.67	27.92
Burst length = 16	29.67	**29.94**	29.67	27.93

Table 4.10: Registration strategies PSNR FP 32 [dB] results summary.

The most noticeable result is that except for the *OpenCV* technique, all of the proposed registration strategies allowed surpassing the PSNR score related to the SISR case. This underlines how the registration process is essential when working with a burst of images, to let the neural network take advantage of the great amount of information in inputs, thus reducing the search space for the reconstruction of the HR image, which finally results in more detailed than then the Single-Image case.
From table 4.10 it can be understood how working with a burst of 8 images is the most optimal choice. Indeed, using 16 image bursts does not result in far superior performances, in some cases it also reaches the same PSNR values as the 8 image case, while the time needed for both the HR image reconstruction, and the images registration is almost doubled. While using 8 images instead of 3 results in a noticeable boost in performance, while only slowing down a bit the inference time.
To further compare the studied registration strategies, table 4.11 reports the values of the different inference times measured on a desktop GPU NVIDIA GeForce RTX 2080 Ti, needed for the estimation of the optical flow of two images of size (3, 150, 150):

Registration method	Inference time [ms]	Number of parameters[M]
OpenCV	1.584	/
ICA	5.082	/
FastFlowNet	6.675	5.38
PWCNet	8.047	9.37

Table 4.11: Inference times.

The strategies that do not involve the usage of a neural network resulted in being the fastest. However, they are not optimized for working with a batch of images. Indeed the training time needed by them is far superior when compared to the time taken by PWCNet and FastFlowNet.

The number of parameters of the two neural networks however highlights how these models are much bigger than NCNet, thus quantizing them and trying to deploy them on low-power and mobile devices is a very challenging task, almost impossible. Since ICA is an algorithm already used in a mobile environment, it results in being the most suitable candidate for a mobile deployment.
To make a visual comparison here are reported some examples of reconstructed images:

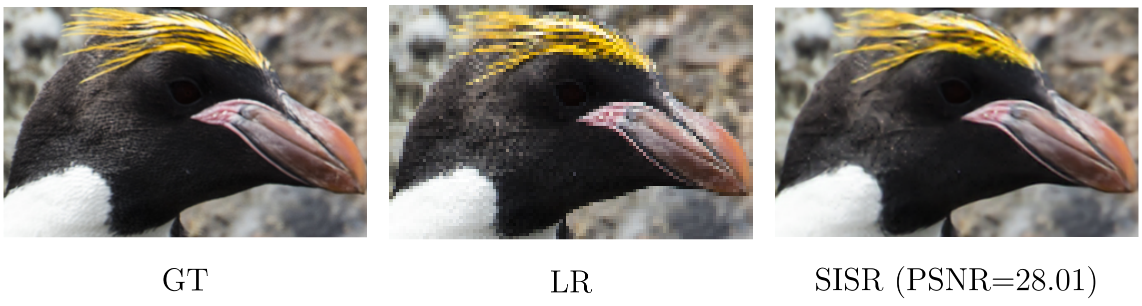


Figure 4.18: Ground truth, Low resolution, and SISR produced image details.

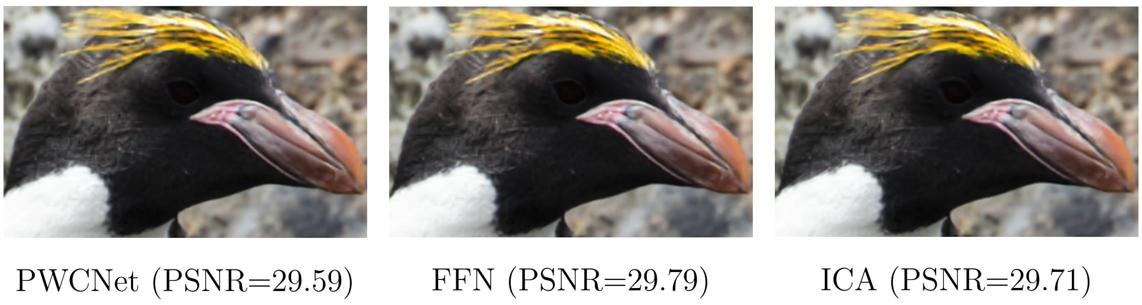
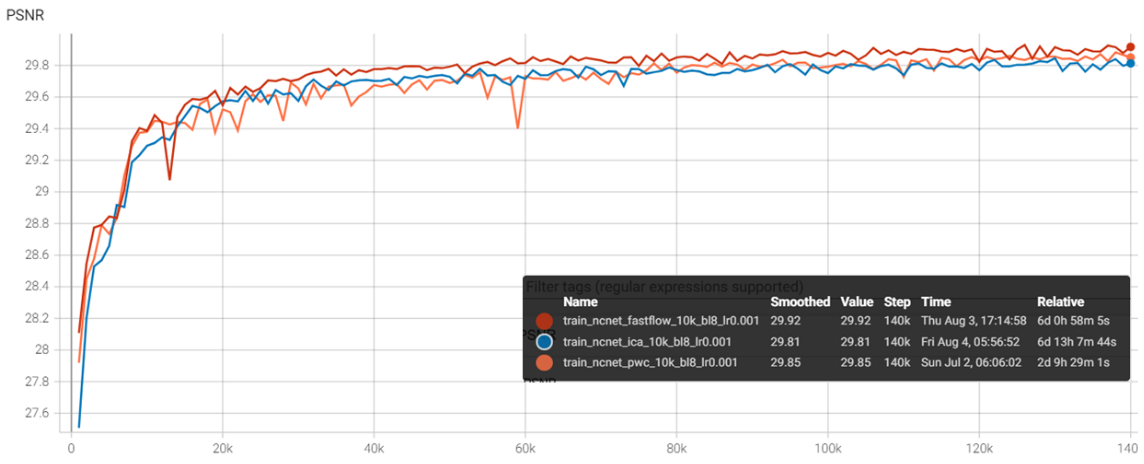


Figure 4.19: Reconstructed image details using PWCNet, FastFlowNet (FFN), and ICA.

Another experiment, involving a training stage extended for 10k epochs has also been performed for all the techniques except for the OpenCV, using bursts of 8 images to try to understand if the models could achieve higher PSNR scores. Indeed, using more epochs resulted in just slightly better results, even though if the training had continued probably the overfitting problem might have arisen.

	PSNR FP32 [dB]
PWCnet	29.88
FastFlowNet	29.93
ICA	29.84

Table 4.12: 10k epochs training results.



	Name	Smoothed	Value	Step	Time	Relative
●	train_ncnet_fastflow_10k_bl8_lr0.001	29.92	29.92	140k	Thu Aug 3, 17:14:58	6d 0h 58m 5s
●	train_ncnet_ica_10k_bl8_lr0.001	29.81	29.81	140k	Fri Aug 4, 05:56:52	6d 13h 7m 44s
●	train_ncnet_pwc_10k_bl8_lr0.001	29.85	29.85	140k	Sun Jul 2, 06:06:02	2d 9h 29m 1s

Figure 4.20: 10k epochs training tensorboard log.

4.7 Model alternatives

Other lightweight networks for super-resolution have also been studied, respectively a lighter and a bigger model compared to NCNet. As the latter, these two other models have been designed to perform SISR, so before training they have been modified accordingly to perform BSR. In this subsection, their structure will first be described and later a training will be done to compare the performances to NCNet.

4.7.1 Extreme Low-Power Super Resolution

The first model is the Extreme Low-Power Super Resolution (ELSR) network [33] which was developed for tackling the video super-resolution task, with the aim of being a mobile-friendly network that consumes as little energy as possible, thus complex operations such as optical flow estimation and multi-frame feature alignment were initially discarded (code available at [34]). The architecture is reported in the following:

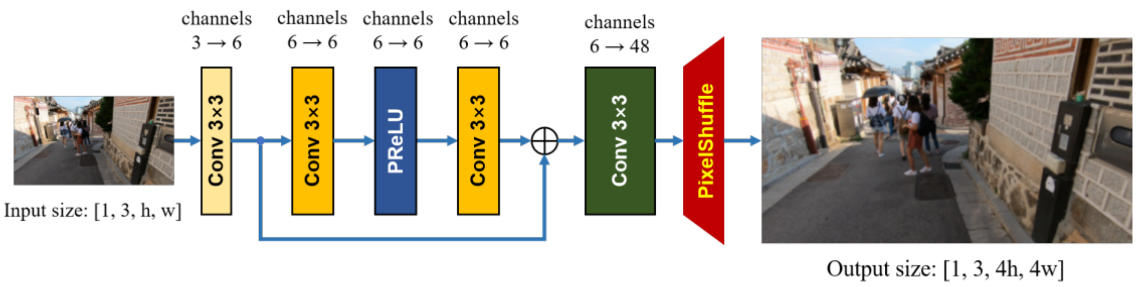


Figure 4.21: ELSR architecture [33].

Differently from NCNet, the architecture is composed of 6 layers, of which only 5 have learnable parameters, including 4 convolution layers and a Parametric ReLU activation layer, which has been studied to boost the performance while having a lower power consumption when compared to other activation functions. In this architecture, the first convolution has been modified in order to let it accept as

input a tensor of a number of channels equal to $3 \times burstlength$. The output of the convolution, as well as, the intermediate feature channels are all set to 6.
As for NCNet, the Pixel-Shuffle operation (also known as depth2space) is used as the last element in the pipeline to upscale the size of the produced feature maps. Moreover, it also uses a skip connection to transfer the input features directly to the last convolution layer for compensating the loss of information.

4.7.2 MobileSR

The heaviest alternative is the mobileSR architecture depicted in figure 4.22 chosen among the participants of the NTIRE 2022 challenge [35]. The three main modules of this model are: the convolutional layers, the upsampling layer and the proposed hybrid module composed of a transformer layer, that combines the qualities of convolution and multi-head self-attention, and an inverted residual block.

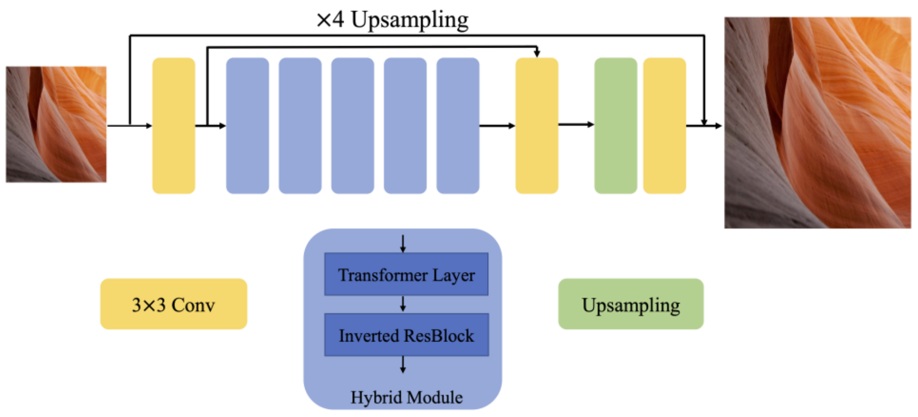
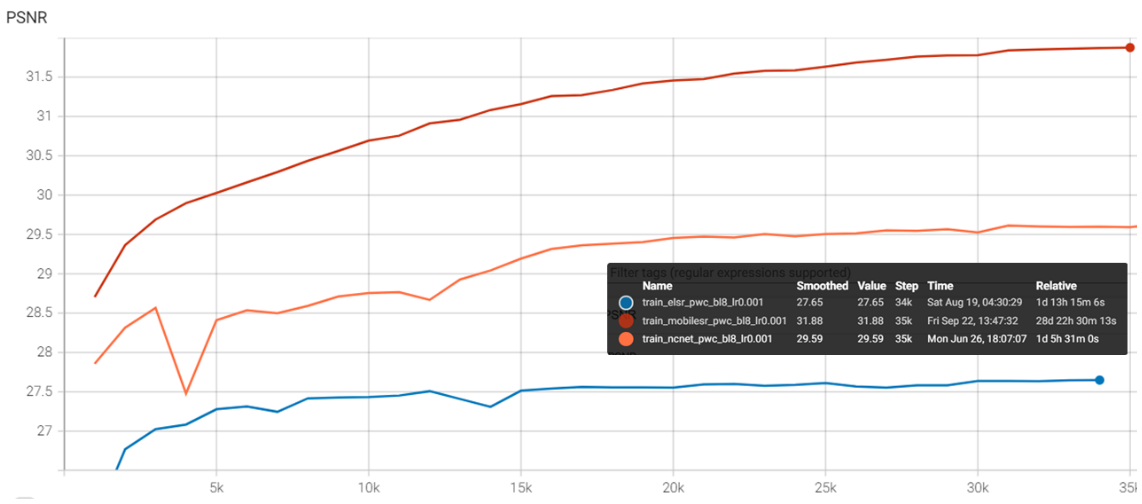


Figure 4.22: MobileSR architecture [35]

The convolution layer is used to map the input to feature space and also in this case it has been modified to accept a number of channels compatible with the input bursts of images. The hybrid module is employed to simultaneously extract local and global information, specifically, it is repeated five times to learn discriminative feature representation. To upsample the feature maps and produce the HR image multiple convolution layers are employed, as well as a final upsampling layer, specifically a pixel shuffle one. The code used to replicate and train this network is available at [36].

4.7.3 Results

The results of the training stage are reported along with the ones obtained by NCNet, in order to carry out a comparison. The models have been trained considering bursts of 8 images, PWCNet as the optical flow estimator, and early fusion as the fusion strategy.



	Name	Smoothed	Value	Step	Time	Relative
●	train_elsr_pwc_bl8_lr0.001	27.65	27.65	34k	Sat Aug 19, 04:30:29	1d 13h 15m 6s
●	train_mobilesr_pwc_bl8_lr0.001	31.88	31.88	35k	Fri Sep 22, 13:47:32	28d 22h 30m 13s
●	train_ncnet_pwc_bl8_lr0.001	29.59	29.59	35k	Mon Jun 26, 18:07:07	1d 5h 31m 0s

Figure 4.23: Model alternatives tensorboard log.

Also for the results that will follow, the inference times have been measured on desktop GPU NVIDIA GeForce RTX 2080 Ti, needed for an input of size (3*burst length, 150, 150):

Network	Number of parameters	Inference time [ms]	PSNR FP32 [dB]
NCNet	58k	0.62	29.59
ELSR	3k	0.39	27.65
MobileSR	185k	9.62	31.88

Table 4.13: Models numerical comparison.

Having more than 100k parameters than NCNet and using strategies like self-attention and elements like transformers, MobileSR achieved a PSNR 1 dB higher than the one obtained by NCNet. Moreover, the red curve of figure 4.23 suggests that if the training had been prolonged it would have achieved even better results, given that, unlike the other two models, the curve has not yet saturated. However, this model has also resulted in being the slowest, with an inference time far superior to the other two cases.
For what concerns ELSR, the model resulted in a PSNR even smaller than the SISR setting of NCNet. Its really small structure made it the fastest model among the three, even if the difference with NCNet inference time is negligible, resulting in NCNet being a superior architecture.
It can be concluded that depending on the constraints of the application for which the model has to be employed one may prefer NCNet or MobileSR, depending on whether the application requires an image reconstruction process that is faster or more detailed.

4.8 Working with RAW images

The previous results demonstrated how the models were able to produce detailed HR images starting from images that presented no other source of corruption. This does not hold for a real-world application, where images could present high levels of noise. In order to test the models' effectiveness with images captured by real devices the following changes have been made.

Synthetic RAW burst generation

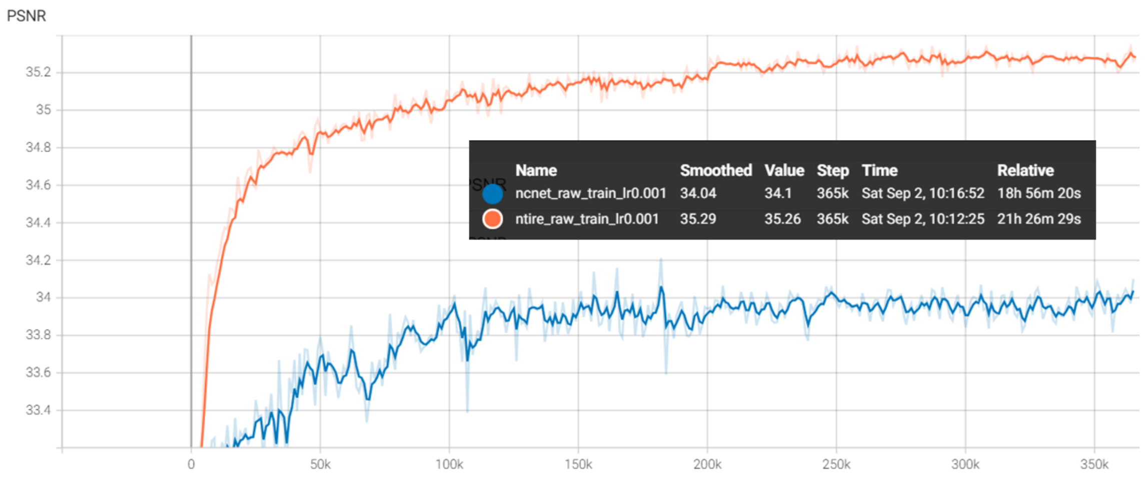
Differently from before, the models were also tested on images in RAW format. Indeed, the full pipeline available at [20] gives the ability to generate a burst of RAW images starting from a single RGB image. Specifically, the generated bursts obtained by relying on the operations already explained in section 4.3.3 are then mosaicked, and corrupted by random noise to obtain the RAW burst. The images composing the resulting bursts have 4 channels corresponding to the 'R', 'G', 'G', and 'B' values in the RGGB bayer mosaic.
In this case the Zurich RAW-to-RGB dataset (section 4.3.2) has been employed.

Architecture update

The mosaicking operation, which extracts RGGB Bayer planes from an RGB image, involves an additional downscaling of the input image by a factor of 2, thus resulting in a final downscaling of the input image size of $2 \cdot downscalingfactor$. To recover this additional downscaling, at the end of the network architectures has been added an additional Pixel Shuffle layer.

4.8.1 Results

Different from the settings of section 4.4, in this case, training has been performed by upscaling the images by a 4x factor, and for 500 epochs, that resulted in more than 350k iterations due to the great number of images composing the Zurich dataset. This process has been carried out for both the NCNet and the mobileSR architectures, using also in this case PWCNet as the optical flow estimator and early fusion as the fusion strategy. The obtained results are the ones that follow:



	Name	Smoothed	Value	Step	Time	Relative
🔵	ncnet_raw_train_lr0.001	34.04	34.1	365k	Sat Sep 2, 10:16:52	18h 56m 20s
🟠	ntire_raw_train_lr0.001	35.29	35.26	365k	Sat Sep 2, 10:12:25	21h 26m 29s

Figure 4.24: Raw training tensorboard log.

Network	PSNR FP32 [dB]	Number of parameters	Inference time [ms]
NCNet	34.04	54k	2.096
MobileSR	35.28	185k	10.605

Table 4.14: Raw images training results.

The most noticeable result is a big boost in PSNR that the models achieved, but this is mainly due to the large number of images that the Zurich dataset is able to provide. Moreover, working with 4 channel images and a scale factor of 4 translates to a slighter slower inference time for both the analyzed models.
To carry on a comparison of the produced super-resolved images, a burst of raw images captured with a *Zebra technologies* device will be used. Since no GT is available in this case, the HR image built by Fusenet [13] will be taken as a reference. Moreover, since the RGB images that correspond to the RAW images are also available, the models trained on RGB images without noise will also be tested.

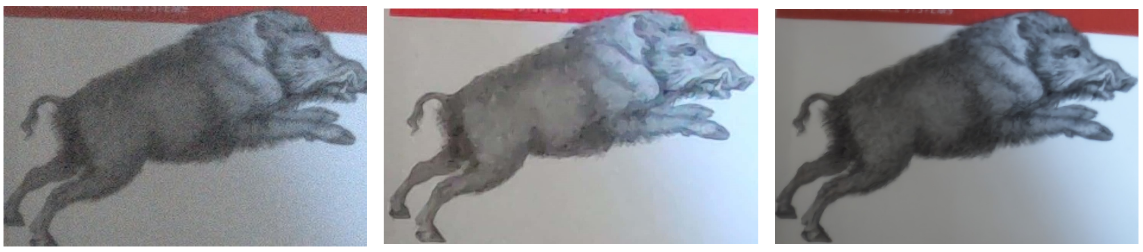
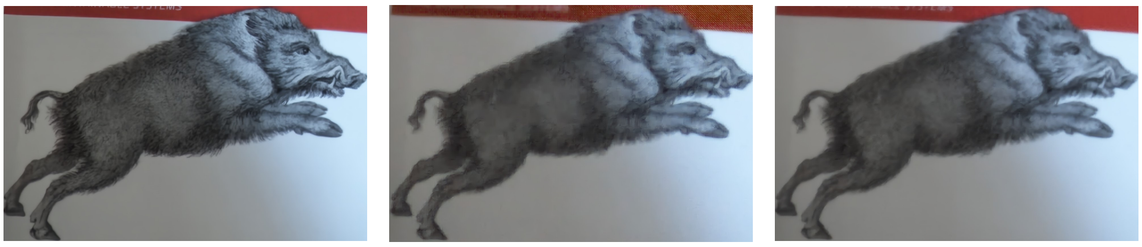


Figure 4.25: Comparison of details of the reconstruced image.

The table that follows also summarizes the total number of parameters achieved by the different combinations:

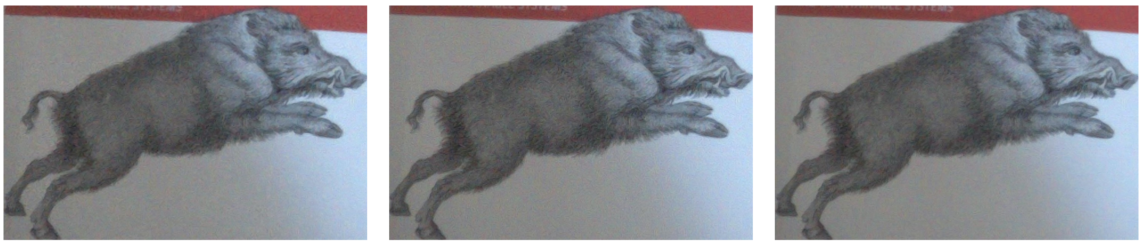
Model	Number of parameters
Fusenet + PWCNet	9.8 M + 9.37 M
NCNet + PWCNet (RAW)	54k M + 9.37 M
MobileSR + PWCNet (RAW)	185k + 9.37 M
NCNet + ICA (RGB)	54k
NCNet + PWCNet (RGB)	54k M + 9.37 M
MobileSR + PWCNet (RGB)	185k M + 9.37 M

Table 4.15: Models number of parameters comparison.



Fusenet [13]. NCNet RAW train. MobilseSR RAW train.

Figure 4.26: Details of images reconstructed by NCNet, MobileSR, and Fusenet using PWCNet as optical flow estimator.



NCNet RGB train (ICA). NCNet RGB train. MobileSR RGB train.

Figure 4.27: Details of images reconstructed by NCNet, and MobileSR trained on RBG images.

It is noticeable how all the models performing BSR are able to produce images far more detailed than the one reported in figure 4.25, confirming the superiority of BSR as the technique to produce super-resolved images. Moreover, looking at figure 4.26 it is evident how the models trained to perform denoising to the input images produce much cleaner and more detailed images with respect to the bottom samples of figure 4.27 which tend to be much more corrupted by noise and have a lower level of detail. Indeed, for applications that involve the use in real-world case scenarios should be advisable to train the network to perform denoising on the input images, along with super-resolution.
For both the NCNet and the MobileSR architectures the gap in performance with Fusenet is evident, even though it can be seen from table 4.15 how much bigger the Fusenet architecture is when compared to mobile-friendly architectures.
The difference in the number of parameters is also reflected in the images produced in all the cases by the mobileSR model when compared to NCNet. The case that seems to lack a greater level of detail is the case in which ICA is used as a registration strategy, even though it is the most lightweight solution for mobile deployment.

Chapter 5

Conclusions

This thesis work focused on the Burst Super-Resolution task, a particular case of the image super-resolution problem, a branch of the image restoration task. The interest in this task is motivated by the fact that allows reaching a finer level of details in the high-resolution images reconstructed, compared to the single image case, furthermore, given the enormous diffusion of mobile photography, it is a task that can very easily find an application in a real-world scenario. In contrast to this, however, the models developed in the literature employ huge computational resources, making their quantization and the subsequent deployment on low-power devices an almost impossible task. Being less popular than the single-image super-resolution case, the literature is not as developed as the latter case, thus this work began by picking a lightweight deep learning model for single-image super-resolution and adapting it to let it accept a burst of images as input. Changing an architecture from a single-image one to a burst one also involves the usage of strategies for both the fusion and the registration of the image features, thus a study and a comparison of different techniques has been carried out.
The conducted study proved the superiority of burst super-resolution for producing images more detailed than the ones produced using a single image. However, it has also been highlighted that to reach such a result burst super-resolution heavily depends on the registration stage of the images, without which there is no real advantage for preferring working with multiple images instead of only one image. Moreover, the image alignment has to be performed well enough for the images to reach a sub-pixel alignment. This is the most important obstacle for the deployment of low-power devices since networks used for optical flow estimation have an incredibly high amount of parameters, making their quantization an almost impossible challenge. However, strategies that do not rely on neural networks already developed for a mobile environment also exist, and could also make this task easier, even though they do not allow them to reach the same level of refinement that neural networks do. Moreover, it has also been verified that working with images corrupted with noise beside rotations and translations has the advantage of letting the model understand how to produce cleaner and more detailed high-resolution images, which is an essential feature for a real-case scenario. Different models for image super-resolution have also been compared, underlining how to reconstruct a more detailed image, greater complexity is required which consequently translates into a greater amount of time

to complete the task. Depending on the requirements of the application for which the model is developed, its various elements have to be chosen accordingly.

Future works

The steps that could be followed to continue this work include trying to understand how much more complex a model can get using mobile-friendly elements, to reach a finer quality of details in the produced HR image. Once it is done, it has to be understood if that model could be quantized for deployment in a mobile environment. If that is not the case an ablation study has to be conducted trying to understand the optimal trade-off between model complexity, thus quality in the reconstruction, and the mobile deployment. Also, the subsequent inference time has to be taken into account to satisfy the requirements of the application.

Appendix A

A.1 Handheld: Accumulation

Once the images composing the input burst are aligned with sub-pixel accuracy to a reference one, the next step followed by the handheld multi-frame super-resolution algorithm is to merge all the images together to produce an HR image with a size s times greater than the starting image, where s is the *upscaling factor*.
To do so, the images are divided into a certain number of N collections of tiles of size T $\times$ T, whose positions relative to the reference one are known. Then, a memory-efficient variant of the nearest convolution sequentially aggregates the different tiles into the final HR image I. More formally:

$$I(x,y) = \frac{\sum_{n=1}^{N} r_n(x,y) \sum_{(u,v)\in N_n^3} w_n(x,y,u,v) J_n(u,v)}{\sum_{n=1}^{N} r_n(x,y) \sum_{(u,v)\in N_n^3} w_n(x,y,u,v)} \tag{A.1}$$

For each LR image, for a given HR pixel (x, y) the values of the closest 3 $\times$ 3 neighborhood are accumulated. In this way, each frame is responsible for 9 weighted observations for generating the HR pixel at (x, y). This strategy is optimal for mobile devices since it allows optimizing the memory footprint of the algorithm no matter the burst size being a single image is processed at each time.

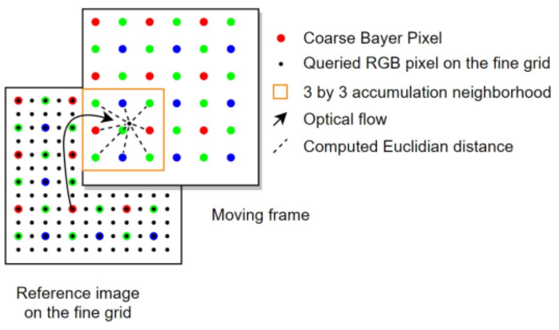

Coarse Bayer Pixel
Queried RGB pixel on the fine grid
3 by 3 accumulation neighborhood
Optical flow
Computed Euclidian distance
Moving frame
Reference image on the fine grid

Figure A.1: Accumulation representation [27]

Algorithm 4: `Accumulation`

```
Data: Frame $J_n$, kernels $\Omega_n$, robustness map $r_n$, patch-wise flow $V_n$, zoom $s$, numerator image
      num, denominator image den
Result: Updated numerator and denominator of the accumulator: num and den
// Main loop over the pixels of the fine grid
1  for $(x, y) \in num$ do
       // Corresponding location on $J_1$'s coarse grid
2      $(x_1, y_1) \leftarrow (x, y)/s$;
       // Projection from $J_1$'s to $J_n$'s coarse grid
3      $(x_n, y_n) \leftarrow V_n(x_1, y_1)$;
       // Fetch the 4 closest pre-computed covariance matrices
4      for $(\hat{x}_n^i, \hat{y}_n^i) \in \mathcal{N}_2(x_n, y_n)$ do
5          $\Omega^i \leftarrow$ Fetch$(\Omega_n, (\hat{x}_n^i, \hat{y}_n^i))$;
6      end
       // Interpolate to the desired position based on the known Omegas
7      $\Omega \leftarrow$ BilinearInterpolation$(\Omega^1, \Omega^2, \Omega^3, \Omega^4, x_n, y_n)$;
8      for $(u, v) \in \mathcal{N}_3(x_n, y_n)$ do
           // Build on-the-fly the merge kernel with $(u, v)$ back-projection on $I$'s fine grid
9          $d \leftarrow (u, v) - (x_n, y_n)$;
10         $w \leftarrow \exp\left(-d^\top \Omega^{-1} d/2\right)$;
           // Look which color component will be added
11         $c \leftarrow$ GetLocalChannel$((u, v))$;
           // Do the aggregation in the image and accumulator
12         num$(x, y, c) \leftarrow$ num$(x, y, c) + r w J_n(u, v)$;
13         den$(x, y, c) \leftarrow$ den$(x, y, c) + r w$;
14     end
15 end
```

Bibliography

[1] Jingwen Su, Boyan Xu, and Hujun Yin. «A Survey of Deep Learning Approaches to Image Restoration». In: *Neurocomputing* 487 (Feb. 2022). DOI: 10.1016/j.neucom.2022.02.046 (cit. on p. 2).

[2] Kevin P. Murphy. *Machine learning : a probabilistic perspective.* Cambridge, Mass. [u.a.]: MIT Press, 2013. ISBN: 9780262018029 0262018020. URL: https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8&qid=1336857747&sr=8-2 (cit. on p. 6).

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Lukasz Kaiser, and Illia Polosukhin. «Attention Is All You Need». In: (June 2017) (cit. on pp. 17, 18).

[4] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. «Image Super-Resolution Using Deep Convolutional Networks». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38 (Dec. 2014). DOI: 10.1109/TPAMI.2015.2439281 (cit. on pp. 20, 25).

[5] Brian B. Moser, Federico Raue, Stanislav Frolov, Sebastian Palacio, Jörn Hees, and Andreas Dengel. «Hitchhiker's Guide to Super-Resolution: Introduction and Recent Advances». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.8 (Aug. 2023), pp. 9862–9882. DOI: 10.1109/tpami.2023.3243794. URL: https://doi.org/10.1109%2Ftpami.2023.3243794 (cit. on p. 21).

[6] Zhou Wang, Alan Bovik, Hamid Sheikh, and Eero Simoncelli. «Image Quality Assessment: From Error Visibility to Structural Similarity». In: *Image Processing, IEEE Transactions on* 13 (May 2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861 (cit. on p. 21).

[7] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. *Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network.* 2016. arXiv: 1609.05158 [cs.CV] (cit. on p. 23).

[8] Zbigniew Wojna, Vittorio Ferrari, Sergio Guadarrama, Nathan Silberman, Liang-Chieh Chen, Alireza Fathi, and Jasper Uijlings. «The Devil is in the Decoder: Classification, Regression and GANs». In: *International Journal of Computer Vision* 127 (Dec. 2019). DOI: 10.1007/s11263-019-01170-8 (cit. on p. 24).

[9] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Gool, and Radu Timofte. «SwinIR: Image Restoration Using Swin Transformer». In: (Oct. 2021), pp. 1833–1844. DOI: `10.1109/ICCVW54120.2021.00210` (cit. on pp. 26, 33).

[10] Zongcai Du, Jie Liu, Jie Tang, and Gangshan Wu. «Anchor-based Plain Net for Mobile Image Super-Resolution». In: (May 2021) (cit. on pp. 27, 29, 36).

[11] Guillaume Berger, Manik Dhingra, Antoine Mercier, Yashesh Savani, Sunny Panchal, and Fatih Porikli. «QuickSRNet: Plain Single-Image Super-Resolution Architecture for Faster Inference on Mobile Platforms». In: (Mar. 2023) (cit. on pp. 28, 29).

[12] Diego Valsesia and Enrico Magli. «Permutation invariance and uncertainty in multitemporal image super-resolution». In: (May 2021) (cit. on pp. 30, 31).

[13] Martina Cilia, Diego Valsesia, Giulia Fracastoro, and Enrico Magli. «Multi-Level Fusion for Burst Super-Resolution with Deep Permutation-Invariant Conditioning». In: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2023, pp. 1–5. DOI: `10.1109/ICASSP49357.2023.10096252` (cit. on pp. 32, 62, 63).

[14] `https://ai-benchmark.com/` (cit. on p. 35).

[15] Andrey Ignatov et al. *Efficient and Accurate Quantized Image Super-Resolution on Mobile NPUs, Mobile AI AIM 2022 challenge: Report.* Nov. 2022. DOI: `10.48550/arXiv.2211.05910` (cit. on p. 35).

[16] Ziwei Luo, Youwei Li, Lei Yu, Qi Wu, Zhihong Wen, Haoqiang Fan, and Shuaicheng Liu. «Fast Nearest Convolution for Real-Time Efficient Image Super-Resolution». In: Feb. 2023, pp. 561–572. ISBN: 978-3-031-25062-0. DOI: `10.1007/978-3-031-25063-7_35` (cit. on pp. 35, 36).

[17] `https://github.com/Algolzw/NCNet` (cit. on p. 36).

[18] Eirikur Agustsson and Radu Timofte. «NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study». In: July 2017, pp. 1122–1131. DOI: `10.1109/CVPRW.2017.150` (cit. on p. 38).

[19] Andrey Ignatov, Luc Van Gool, and Radu Timofte. *Replacing Mobile Camera ISP with a Single Deep Learning Model.* Feb. 2020 (cit. on p. 39).

[20] `https://github.com/goutamgmb/NTIRE21_BURSTSR` (cit. on pp. 39, 45, 61).

[21] `https://github.com/cszn/KAIR` (cit. on p. 40).

[22] Jose Caballero, Christian Ledig, Andrew Aitken, Alejandro Acosta, Johannes Totz, Zehan Wang, and Wenzhe Shi. «Real-Time Video Super-Resolution with Spatio-Temporal Networks and Motion Compensation». In: July 2017, pp. 2848–2857. DOI: `10.1109/CVPR.2017.304` (cit. on pp. 41, 42).

[23] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. «PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume». In: June 2018, pp. 8934–8943. DOI: `10.1109/CVPR.2018.00931` (cit. on p. 46).

[24] Lingtong Kong, Chunhua Shen, and Jie Yang. «FastFlowNet: A Lightweight Network for Fast Optical Flow Estimation». In: May 2021, pp. 10310–10316. DOI: `10.1109/ICRA48506.2021.9560800` (cit. on p. 48).

[25] https://github.com/ltkong218/FastFlowNet (cit. on p. 49).

[26] Bartlomiej Wronski, Ignacio Garcia-Dorado, Manfred Ernst, Damien Kelly, Michael Krainin, Chia-Kai Liang, Marc Levoy, and Peyman Milanfar. «Handheld Multi-Frame Super-Resolution». In: *CoRR* abs/1905.03277 (2019). arXiv: 1905.03277. URL: http://arxiv.org/abs/1905.03277 (cit. on p. 50).

[27] Jamy Lafenetre, Gabriele Facciolo, and Thomas Eboli. «Implementing Handheld Burst Super-Resolution». In: *Image Processing On Line* 13 (2023). https://doi.org/10.5201/ipol.2023.460, pp. 227–257 (cit. on pp. 50, 66).

[28] Simon Baker, Ralph Gross, and Iain Matthews. «Lucas-Kanade 20 Years on: A Unifying Framework». In: *International Journal of Computer Vision* 56 (Mar. 2004). DOI: 10.1023/B:VISI.0000011205.11775.fd (cit. on p. 51).

[29] Antoine Monod, Julie Delon, and Thomas Veit. «An Analysis and Implementation of the HDR+ Burst Denoising Method». In: *Image Processing On Line* 11 (2021). https://doi.org/10.5201/ipol.2021.336, pp. 142–169 (cit. on p. 52).

[30] Javier Sánchez. «The Inverse Compositional Algorithm for Parametric Registration». In: *Image Processing On Line* 6 (2016). https://doi.org/10.5201/ipol.2016.153, pp. 212–232 (cit. on p. 53).

[31] https://github.com/Jamy-L/Handheld-Multi-Frame-Super-Resolution (cit. on p. 53).

[32] Gunnar Farnebäck. «Two-Frame Motion Estimation Based on Polynomial Expansion». In: vol. 2749. June 2003, pp. 363–370. ISBN: 978-3-540-40601-3. DOI: 10.1007/3-540-45103-X_50 (cit. on p. 54).

[33] Tianyu Xu, Zhuang Jia, Yijian Zhang, Long Bao, and Heng Sun. *ELSR: Extreme Low-Power Super Resolution Network For Mobile Devices.* Aug. 2022 (cit. on p. 58).

[34] https://github.com/andreacoppari/ELSR-torch (cit. on p. 58).

[35] Yawei Li et al. *NTIRE 2022 Challenge on Efficient Super-Resolution: Methods and Results.* 2022. arXiv: 2205.05675 [cs.CV] (cit. on p. 59).

[36] https://github.com/sunny2109/MobileSR-NTIRE2022 (cit. on p. 59).