POLITECNICO DI TORINO

Master of Science in Electronics Engineering

Master's Degree Thesis

Quantization Analysis for Face Image Detection Through Dense Neural Networks

Supervisors:
Prof. Guido Masera
Prof. Giovanni Ramponi

Candidate:
Alessandra Calzoni

Academic Year 2019 - 2020

A mamma e papà,
A mia sorella,
A Christian,
A Flaminia e Camilla,
A tutti quelli che mi sono
stati accanto fino alla fine

Abstract

The range of applications of facial recognition has greatly expanded, especially in critical fields as crime prevention, bank account access and payments. Machine learning has greatly helped the development and the diffusion of this biometric measurement, making it easier to develop and to implement.
The increasing popularity in mobile systems introduces the problem of low-power networks with high accuracies. This requirement is in contrast to the current trend of more complex and deeper networks. Many methods have been proposed to implement less computationally and memory intensive systems with low accuracy deterioration: knowledge distillation and quantization are some examples. This thesis focuses on post-training quantization on deep structures: two versions of a DenseNet architecture have been considered, both descended from a more complex network through knowledge distillation.

The investigation concentrates on two main families of quantization techniques: scalar quantization and vector quantization.
Three scalar quantization techniques have been taken into account. Firstly a reduced bit-width version of Floating-Point has been considered to cut the required memory with an almost null degradation on the accuracy. To further reduce the computational complexity, Dynamic Fixed-Point and Power of 2 Weights representations have been adopted.
Regarding vector quantization, K-Means Clustering has been considered. This method is able to reduce the storage requirement.
All the networks have been implemented in *Caffe*. Scalar quantization has been applied thanks to *Ristretto* framework, an expansion of *Caffe*, while K-Means Clustering have been carried out in Python thanks to the *sklearn* library.

The quantization has been performed taking into account a layer-by-layer approach. Firstly Convolutional and Fully Connected layers have been approximated. Then the attention moves on Batch Normalization layers, which are more sensitive to quantization and involves highly computational consuming

operations. To solve this latter issue, Folding has been applied to Batch Normalization layers to avoid complex operation, e.g. division and square root, with null accuracy degradation.

Each version of the two reference DenseNet networks have been classified in terms of accuracy through the LFW test, a standard procedure in Neural Network. To identify the solutions with an acceptable accuracy degradation, quantized DenseNets have been compared to the reference 32-bit FP networks. Good results in terms of both accuracy and model compression have been achieved with reduced Floating-Point, K-Means Clustering and an hybrid approach between the two:

– 8-bit Floating-Point is characterized by an almost null accuracy degradation (< 0.2) and a compression factor of the network size equal to 4 for both the networks;

- K-Means Clustering reaches good results if each layer is associated to a different set of 16 clusters. Both the networks achieve a parameter compression factor higher then 7.9, limiting the accuracy degradation to 2.48 in DenseNet 1.0 and to 4.35 for 2.0 version.

- An hybrid approach between the ones described above restricts accuracy degradation to 2.45 and 4.45, while the parameter size are decreased by a factor 8 and the total network size by a factor 5.3.

Power of 2 Weights representation unfortunately introduces too high accuracy deterioration in all the scenarios. Adopting Dynamic Fixed-Point good results can be achieved if only Convolutional and Fully Connected layers are quantized. Nevertheless, when Batch Normalization is taken into account, the representation format fails to cover the dynamic range of the new parameters.

Contents

List of Figures

List of Tables

List of Symbols

ANN Artificial Neural Network. 5, 6

BN Batch Normalization. I, II, IV, VII, 7, 12, 13, 16, 31–33, 37–42, 45, 49, 52, 54, 56, 57, 61

CNN Convolutional Neural Network. VII, 6, 7, 11–14, 25, 35, 63

Conv Convolutional. I, IV, IX, 7, 9–18, 31–33, 37–40, 45, 48–52, 54–57, 63, 64, 66–68

DenseNet Dense Convolutional Network. I, II, VII, IX, X, 4, 16, 29–34, 37–45, 47–58, 60–62

FC Fully Connected. I, IV, VII, IX, 7, 9–11, 13, 17, 18, 33, 37–40, 43, 45, 48–52, 54–56, 63, 64, 66–68

FP Floating-Point. I, II, IV, VII–X, 22–27, 35, 38, 50, 51, 53, 54, 56, 59–66

HW Hardware. 24, 35, 36, 39, 40, 61, 62

ILSVRC ImageNet Large Scale Visual Recognition Challenge. 3, 13, 14

KD Knowledge Distillation. 17, 29

LFW Labeled Faces in the Wild. II, IV, VII, 4, 43, 45–50, 53, 58

LMDB Lightning Memory-Mapped Database. 41

MLP Multi-Layer Perceptron. 43

NN Neural Network. II, 3–6, 9, 11, 12, 17, 18, 21–27, 29, 36, 38, 43, 45, 46

ReLU Rectified Linear Unit. 11, 13, 16, 31, 32, 43

ResNet Residual Neural Network. VII, 14–16, 29, 30

RGB Red, Blu and Green. 6

1 Introduction

1.1 Face Recognition

Facial recognition is a technology capable of determining the identity of a person through the analysis of his, or her, face.
In recent years this biometric measurement has become increasingly popular: it is the most "natural" way to identify someone, it is simple to be implemented and it does not require any physical interaction with the person to be identified.

To accomplish face recognition, the processing is typically divided in four steps. Firstly, *face detection* is performed in order to detect a face in an image, separating it from the background. Then an *alignment procedure* is carried out in order to increase the accuracy of the system: e.g., the output image is rotated to have the line joining the eye of the person in parallel with the horizontal line; moreover, all processed images are resized so that they have all the same scale. Thanks to this process, in each image the different features of a face (eyes, nose, mouth and chin) are placed in their typical position. The third step is *feature extraction*, in which the facial features of the image, such as elements position and distance, are measured and a feature matrix is created. The last step is *image recognition*, for which the obtained features are typically compared to a database, in order to determine the person in the processed image.

1.1.1 Historical Hints

The pioneers of face recognition were Woody Bledsoe, Helen Chan Wolf and Charles Bisson, which in the 1960s started programming computers to identify

human faces [1][2]. Their work involved a manual measurement of face features: an example is shown in figure 1.1. These values were then used by the computer to compute the distances between the landmarks of the face with the ones in their database to determine the identity of the person. At the time, the research was not successful and most of their work was not published. Nevertheless it was the first fundamental step for the face recognition history.

Figure 1.1: An Image of Woody Bledsoe from a 1965 Study [3]

A first improvement was achieved in the late 1980s with the introduction of linear algebra approaches, in particular the Eigenface method [4]: the human face is encoded as a weighted combination of eigenvectors, which are then processed to identify the person. This was the first attempt for automatic face recognition and it laid the groundwork for the future researches. During the 1990s and the first 2000s, the efforts focused on the identification of new methods to increase the accuracy of the system: examples are Local Binary Patterns [5] and Elastic Bunch Graph Matching [6]. However, the proposed methods introduced very small improvements on the accuracy and most of them failed to deal with facial changes, such as expression, lighting, disguise or aging.

The turning point was achieved in the 2010s thanks to deep learning. In these years, the increasing computing power of computers allowed to support the training of more complex Neural Networks. Since then, these systems have been used for a wider range of applications, including face recognition. In 2012 the AlexNet network [7] won the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC), with an accuracy of 86.7% and by a margin of more than 10% from the other proposed systems. From this moment on, Neural Networks were used extensively in facial recognition, achieving within few years accuracies above 95%.

1.2 Problem Statement

Nowadays face image recognition can be used in a wide range of applications. In security services face recognition is adopted for crime prevention and detection. Together with fingerprints, it is used to verify the identity of a person during board checks, comparing the digital image on the passport with the passenger's face. Identification through face recognition has become a popular feature in smartphones, game consoles and computers and can be used to access critical services, as bank account or mobile payment. Also social media has started to use face recognition algorithm, for example to automatically identify and tag people on different photos. In the future, this technology is going to be used in a larger number of applications: hospitality, health-care, transports, retail are some of the possible fields where face recognition will rapidly spread.

The growing usage in mobile systems has introduce low-power requirements, which have called attention to the need of systems less computationally and memory intensive. On the other hand, the wider number of application where ID verification is critical, such as bank accounts or trading, imposes a limit on the degradation of the system accuracy, to avoid errors in the correctness of the identification.
The research of a trade-off between power consumption and accuracy is still an unsolved problem. Over time, the accuracy of Neural Networks has increased together with their complexity and their size: the harder task, the deeper and

the more complex will be the system. But the requirements for an high accuracy seems to be incompatible with a low-power system. It is particularly true for face recognition, in which extremely high-accuracy networks are required. In the last years, many methods have been proposed to reduce the parameters and the amount of computations of a system. However, most of the approaches have been validate on simple networks. This thesis focuses on the analysis of deeper structures to verify their reaction to quantization, i.e. a particular compression method.

1.3 Outline

The thesis is organized in 6 total chapters:

- This is *Chapter 1*, which is the introductory chapter. A brief introduction on face image recognition is provided to the reader;
- *Chapter 2* gives an overview on the basic concepts of Neural Networks. Among the described architectures there is the Dense Convolutional Network, which is the driving example for the quantization discussion in this thesis. Lastly, the most important techniques for model compression are described;
- *Chapter 3* illustrates quantization and the different approaches that can be found in literature;
- *Chapter 4* describes the structure of the two driving examples of this thesis: two Dense Convolutional Networks. Moreover the considered quantization strategy is depicted;
- *Chapter 5* collects the obtained results from the different approaches and explains the Labeled Faces in the Wild (LFW) test, widely used to evaluate and compare accuracies;
- *Chapter 6* discusses the results and proposes future prospects.

2 Neural Networks Background

An Artificial Neural Network (ANN) [8] is a computing system based on the biological neural network of the living beings. Its basic block is the *artificial neuron* which, as the biological one, is able to make decisions and it is able to learn when it is wrong.

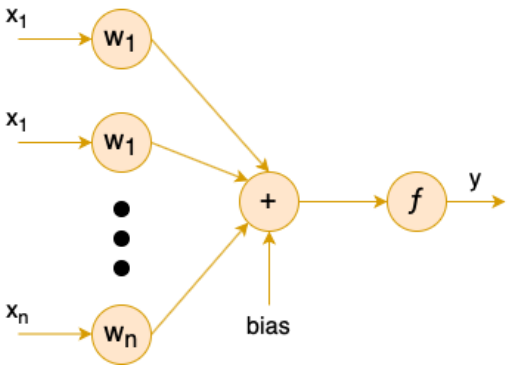


Figure 2.1: Artificial Neuron Structure

An artificial neuron (figure 2.1) can be modeled as a computational unit that takes decisions from a weighted sum of inputs, which can consider also a bias. An activation function is then applied to this sum, generating the output signal of the neuron. Several types of activation functions exist, as described in section 2.1.1. In a Neural Network (NN) neurons are grouped together to constitute several independent layers.

The main aspects of a Neural Network are *inference* and *training*. The former is characterized by the forward algorithm describe above: through its weights and bias, the NN classifies its inputs. The latter allows the network to learn

how to make decisions. This mechanism is based on an iterative back propagation algorithm [9]: the inputs are forward propagated in the network and a loss function is computed with respect to the expected values in the training database; in order to minimize this loss, a gradient method is applied to modify the weights and the bias in the network, following a backward algorithm. Commonly used methods are Gradient Descent [10] and Stochastic Gradient Descent [11].

Taking into account enough training data and a sufficiently large structure, a Neural Network can approximate any function to arbitrary precision.

2.1 Convolutional Neural Network

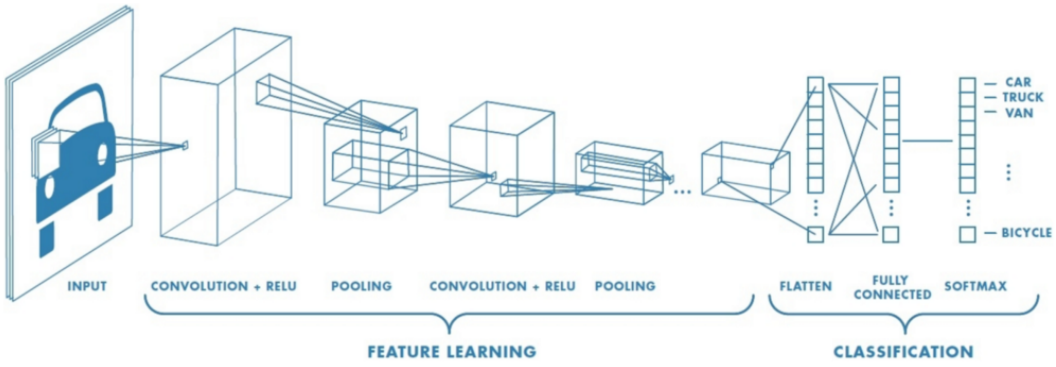


Figure 2.2: Basic Architecture of a CNN [12]

Convolutional Neural Network (CNN) is a particular type of ANN, commonly used in image analysis. For this reason, the input feature of a CNN is typically a tensor or a matrix of pixels: an Red, Blu and Green (RGB) image is represent by a tensor with 3 channels, one for each color, of pixels organized in matrices of dimension *height* x *width*; a grey-scale image is represented by a matrix, so it has just one channel.

A Convolutional Neural Network is based on the convolution operation, which is nothing but a filter applied on the input image. Thanks to this operation,

in CNN *parameter sharing* and *local connectivity* are possible: the filter parameter could be shared among the neurons and each output is related only to a small portion of inputs.

The standard architecture of a CNN consists on:

- Convolutional layer;
- Fully Connected layer;
- Pooling layer;
- Batch Normalization;
- Activation Function.

The building blocks of a CNN system are the *feature extractor* and the *classifier*. The former is intended to reduce the redundancy of the input image, which is transformed into a reduced version containing more relevant information. This block typically consists on a combination of Convolutional and Pooling layers. The latter processes the feature extracted by the previous block and gives a prediction on the identity of the input image. It is composed of one or more Fully Connected layers.

2.1.1 Internal Architecture

Convolutional Layer

The Convolutional (Conv) layer is based on the convolution operation, which is a filtering operation. This procedure involves a square matrix, called *kernel* or *filter*, that select a group of inputs. The convolution (figure 2.3) is computed as the sum of the dotted products between the kernel and the selected inputs. The filter slides along the input matrix and the convolution results are collected in the output feature matrix. The step size used for this procedure is called *stride*. A bias can be added to compute the output value. Typically, a Convolutional layer is made up by a several number of filters.

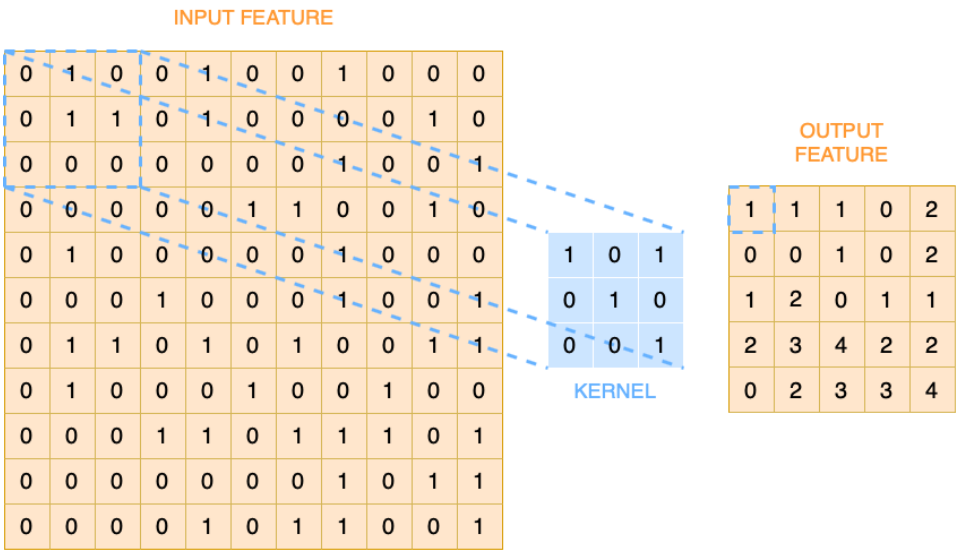


Figure 2.3: Convolution Example: 3x3 *kernel* and *stride* = 2

Zero Padding For a kernel with size higher than one, the height and the width of the output feature is reduced with respect to the dimensions of the input feature. However, sometimes it may be required to preserve the input dimensions and the zero padding mechanism is able to give higher degrees of freedom in the output size identification: zeros are added on the border of the input matrix. The amount of rows and columns only containing zeros depends on stride and on the kernel and input sizes.

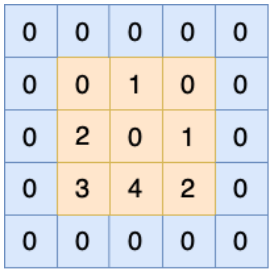


Figure 2.4: Padding Example

Spacial Arrangement The dimensions of the output feature could be controlled by the following hyper-parameters:

- *Number of Filters*, N_f;
- *Kernel Field Size*, K, which identifies the width and the height;
- *Stride*, S;
- *Padding*, P, the number of added rows and columns of zeros.

The input feature has a size equal to $D_i \cdot H_i \cdot W_i$, respectively depth, height and width. From these quantities, the size of the output feature, $D_o \cdot H_o \cdot W_o$, are:

$$D_o = N_f \tag{2.1}$$

$$H_o = \frac{H_i - K + 2P}{S} + 1 \tag{2.2}$$

$$W_o = \frac{W_i - K + 2P}{S} + 1 \tag{2.3}$$

Fully Connected Layer

In Fully Connected (FC) layers, each output value is a weighted sum of all the inputs plus a bias term:

$$y[i] = \sum_{i=1}^{N_o} \sum_{j=1}^{N_i} weight[i][j] \cdot x[j] + bias[i]$$

where N_i is the size of input features and N_o is the size of output features.

These layers are close to the concept of neuron of Neural Networks: each output can be seen as an artificial neuron and it is connected to all the neurons in the previous layer (figure 2.5).
When a FC layer follows a Convolutional or a Pooling layer, it flattens them into a vector. In fact, the output is always a feature vector of dimension N_o.

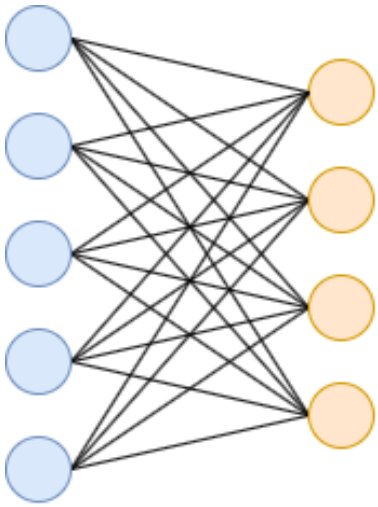


Figure 2.5: Fully Connected Structure

Pooling Layer

The Pooling layer is generally used to lower the size of the intermediate features. Its insertion among Convolutional layers is helpful to reduce the number of parameters and computations in the network.
The Pooling kernel can be seen as a window that selects a set of input values that are combined together to form a new output (figure 2.6). The two main types of combinations are:

- *Average Pooling*, in which the output is equal to the average of the selected values;
- *Maximum Pooling*, in which the output is equal to the maximum input values.

The hyper-parameters are similar to the ones reported in section 2.1.1: stride, padding and kernel field size. Number of filters is instead a fixed quantity, because in a Pooling layer the depths of input and output features are the same: $D_o = D_i$.

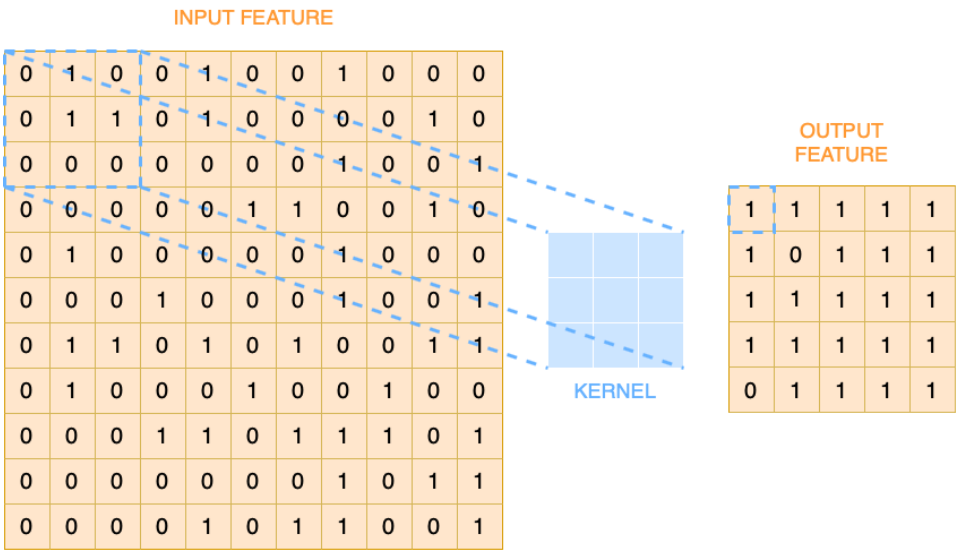


Figure 2.6: Maximum Pooling Example

Activation Functions

As explained at the beginning of this chapter, activation functions are part Neural Networks. In CNN, *non-linear* functions are typically adopted. This kind of functions introduces non-linearity in the network. This is an important characteristic to model complex relations between inputs and output: this property allow the network to learn better and more precisely how to classify complex data, e.g. images. The most used non-linear functions (fig. 2.7) are:

- *Sigmoid*

$$y(x) = \frac{1}{1 + e^{-x}}$$

- *Hyperbolic Tangent*

$$y(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- *Rectified Linear Unit (ReLU)*

$$y(x) = \max(0, x)$$

ReLU is the most used function because it is able to reduce the training time of the network without a significant loss in the accuracy. Activation functions are adopted after Convolutional and FC layers.

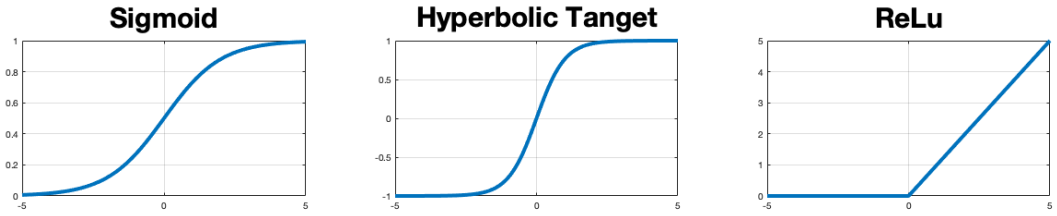


Figure 2.7: Non-Linear Activation Functions

Batch Normalization

Batch Normalization (BN) [13] is an important method used to obtain a faster training and a simpler initialization of the parameters in the network. In fact, BN stabilizes the features inside the network, reducing their changes, thus making easier the identification of the hyper-parameters of the overall network. It is typically performed after an activation function or between a Convolutional layer and an activation function.
A batch is a subset of input samples processed before updating the internal network parameters. Typically in Neural Network, when the batch size is more than one sample and less than the size of the input dataset, this subset is called mini-batch. During training each input feature in a mini-batch of the entire training set is normalized through its mean and variance. The normalized feature has zero mean and unit variance.

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

where ϵ is an arbitrary small constant.

The normalization procedure may reduces the non-linearity property of the CNN, lowering the representation power of the network. For that reason a scaling and a shifting procedure are considered to adapt the normalization. The scale (γ) and shift (β) parameters are updated during the training, as the

other hyper-parameters of the network.
The final output of the Batch Normalization is:

$$y_i = \gamma \cdot \hat{x}_i + \beta$$

During inference all the batch parameters are fixed, including the mean and the variance. Typically, during the training, the moving averages of these two quantities are updated after each mini-batch and the resulting average values are considered in the inference, together with the trained γ and β.

2.1.2 AlexNet

AlexNet [7] is a CNN proposed in 2012 by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton. In the same year it won the ILSVRC with an error rate of the 15.3%, far ahead of its opponents. Its structure (figure 2.8) was designed considering five Conv layers, two Local Response Normalization layers, three Maximum Pooling Layers and three FC layers. Compared to the architectures of the time, it was very large with its 60 millions parameters.

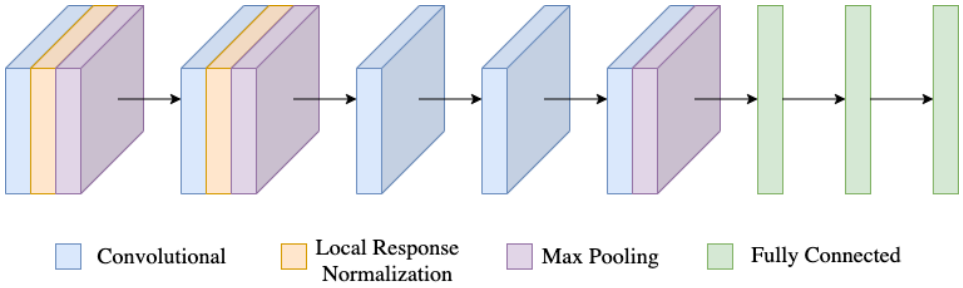


Figure 2.8: AlexNet Structure

AlexNet introduced two main innovations:

- ReLU was used as activation function, replacing the sigmoid and tanh functions which were used as a standard at that time. The ReLU allowed for faster training of the network;
- Training on multiple parallel GPUs was supported, further lowering the training time.

2.2 Residual Neural Network

Residual Neural Network (ResNet) [14] was proposed in 2015 for the ImageNet Large Scale Visual Recognition Challenge, winning the first place with an error rate of 3.57%.

Since AlexNet, CNN architectures were becoming deeper and deeper. The depth increasing had introduced a problem related to accuracy saturation: during training the backward propagation along big networks, may produce infinitely small gradients, preventing the weights from changing. This phenomenon is called *gradient vanishing* and its effect is a saturation, or even a degradation, of the accuracy with respect to the network depth.
ResNet architecture was proposed to overcome this problem. Shortcuts connections are added to the classic CNN architecture, to skip typically two, or three, layers. During training gradients are back-propagated through the shortcuts, reducing the gradient vanishing issue. Thanks to this mechanism, ResNets can be made by more than 100 layers.

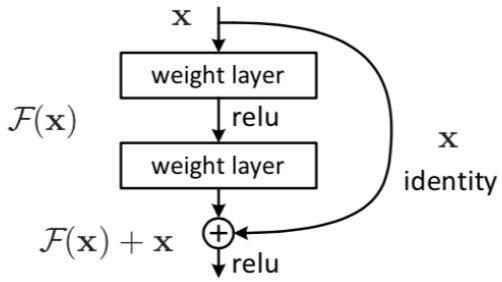


Figure 2.9: ResNet Building Block [14]

Furthermore, a *bottleneck design* was proposed for those architectures with a very large number of layers. ResNet architectures are typically made by 3x3 Convolutional layers, as shown in figure 2.10, and shortcuts skip two layers. To reduce the overall amount of parameters in the network, the original building block, made by two 3x3 Convolutional layers, could be replaced with a stack of three layers: 1x1, 3x3 and 1x1 Conv layers. The 1x1 Conv layers are responsible to the reducing and then the increasing of the dimensions of their outputs,

allowing a reduction the size of the 3x3 Conv.

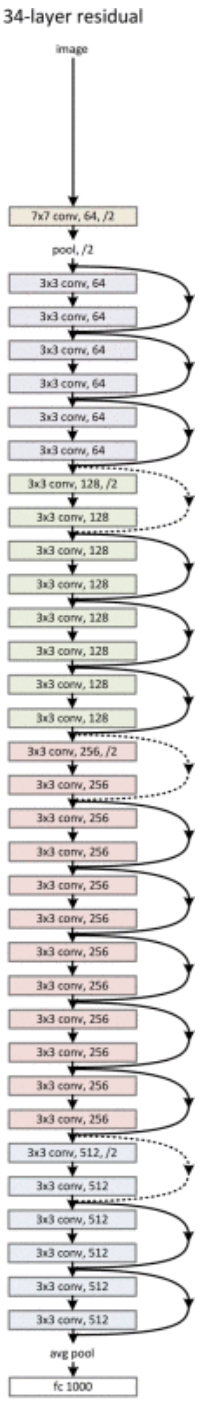


Figure 2.10: ResNet Structure [14]

2.3 Dense Convolutional Network

Dense Convolutional Network (DenseNet) [15] was proposed in 2017 as an evolution of ResNets.

The internal architecture of a DenseNet is divided in blocks, called *dense blocks*. On these blocks, the concept of shortcuts is amplified: each dense block is connected to all the following blocks (figure 2.11).
In addition to solving the vanishing gradient problem, this network is able to speed up the training thanks to its denser connections: it is like each dense block gets access to the "collective knowledge" of the system. Furthermore, the usage of concatenated input features learned by different layers increases the training power, obtaining less and more efficient hyper-parameters.
The dense blocks consists on a Batch Normalization, a 3x3 Convolutional layer and a ReLU activation function. The bottleneck design explained in section 2.2 could be applied also on DenseNets.

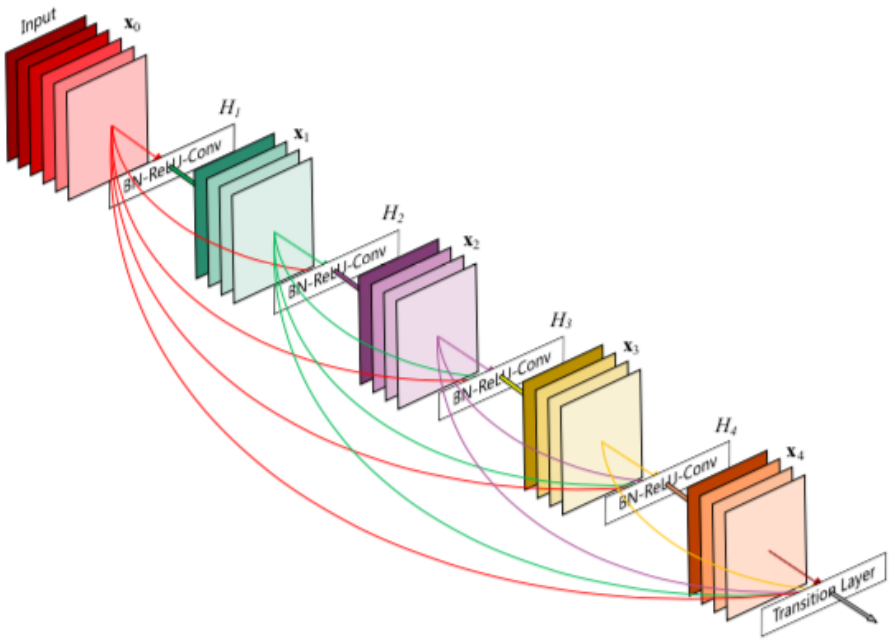


Figure 2.11: DenseNet Structure [15]

2.4 Model Compression

Recently the trend of Neural Network has been characterized by an increase in complexity to obtain higher accuracies. This leads to structures containing hundreds or thousands of base-level classifiers, hindering their employment in portable devices. Therefore, in the last years great efforts have been made to develop techniques that can compress models and limit redundancy, without affecting accuracy. The most important techniques [16] that have been used for the model compression of networks with Convolutional and Fully Connected layers are:

- Knowledge Distillation;
- Low-Rank Factorization;
- Parameter Quantization;
- Parameter Pruning.

Knowledge Distillation

Knowledge Distillation (KD) [17] is based on the fact that training and inference can be seen as separated tasks of a Neural Network. Therefore, they can be implemented by different models. In particular KD *transfers the knowledge of the training network into a smaller one* that can be used for inference. The compressed network, called "student", can be implemented with a completely different architecture and it requires a training to mimic the behavior of the larger architecture, called "teacher".
The Knowledge Distillation technique has high degrees of freedom in identifying a compressed solution: the architecture selection, the internal structure, etc. Nevertheless, this complexity is rewarded with an higher flexibility in finding a solution.

The solution in [18] adopted this compression technique on a dense neural network, reducing the number of parameters by more than half and obtaining compatible accuracies with the original one (around 3-2%).
Moreover, [19] made further improvements in the achievable mimic capability

if the "student" is able to learn information also from the intermediate layers of the "teacher". The obtained networks were deeper but thinner. The lower width allowed the network to be more compact, thus to have a lower number of total parameters (the compression rate was between 3 and 10). Moreover, the "student" networks reached higher accuracies with respect to the "teacher" ones.

Low-Rank Factorization

A rank is the number of linear-independent columns in a matrix. If the rank is lower than the number of total columns, some redundant information is present in the matrix. In NN this concept can be extended to filters and channels, which are typically redundant. The low-rank factorization approximate dense networks with a linear combination of fewer number of filters through *matrix/tensor factorization or decomposition*. Then a training process is required to fix the model and to reduce the loss of accuracy.

In [20] low-rank factorization is applied on different NN architectures. For AlexNet, they achieved an average reduction factor of 3.4 for the energy consumption, with a dropout on the accuracy lower than the 2%.

Parameter Pruning

The goal of parameter pruning is the reduction of the size and the complexity of a NN removing the redundancy in the non-critical parts. This is achieved by *eliminating connections, neurons, channels or entire filters*. Depending on what is the target of the pruning method, a ranking of the possible elements to be eliminated is drawn up according to their contribution on the correctness of the network prediction. The less meaningful ones could than be removed: the dense network is transformed in a sparse one. Thanks to a re-training process the remaining elements can compensate the erased ones.

An example of pruning is given in [21], where Conv and FC weights below a certain threshold were removed from the network to reduce the connections. The techniques were applied to three architectures: Lenet5 [22], AlexNet and

VGGNet [23]. The results showed that a reduction of one order of magnitude in the number of parameters with an almost null accuracy degradation is possible, taking into account a re-training procedure.
In [24], pruning was considered together with weights quantization and Huffman coding to further reduce the required memory and the related energy consumption. Testing their method on the Lenet5, AlexNet and VGGNet, they reached a parameter compression rate between 35 and 40, without no loss of accuracy.

3 Quantization

Quantization focuses on *lowering the size of data*. In a Neural Network the aim is the reduction of the number of bits used to represent information. Parameters and features are mapped from a large set of values to a smaller one introducing an error, the *quantization error*.
Nevertheless, NNs are inherently resilient to small errors and fluctuations thus they should be able to dealt with quantized information.

In Neural Network quantization gives advantages both on computational and storage point of views: a lower number of bits means that a smaller memory is required to store the data; moreover, operations become simpler allowing a speed-up and a lower power consumption.

3.1 Quantization Types

Quantization techniques can be divided in two main groups, depending on the approach considered for the approximation of the weights: post-training quantization and quantization aware-training.

Post-Training Quantization

Post-training quantization [25] focuses on inference. In fact weights are trained in full-precision and then a dynamic range analysis is carried out to identify the correct formats to represent the parameters.
This approach is the simplest one, even if quantization could lead to an high loss of accuracy, especially for integer or fixed-point quantization.

Quantization Aware-Training

Quantization aware-training [25] considers the quantization of the parameters also during training. According to this approach, networks are firstly trained in full-precision to identify the starting parameters. Then a dynamic range analysis is performed and quantization is applied. At this point the network is re-trained, taking into account also the quantization error which becomes part of the network loss that the training process minimizes.
This approach is more complex than the simpler post-training quantization. Nevertheless, it makes the network more robust to quantization errors, allowing for almost null accuracy losses even after extreme low bit-width integer or fixed-point quantization.

3.2 Quantization Formats

Traditionally Neural Networks adopt 32-bit Floating-Point (FP) (IEEE-754 standard [26]) representation for training and inference. However, other types of format can be taken into account to reduce the computational cost and the required memory:

- Mini Floating-Point representation;
- Fixed-Point representation;
- Power of 2 representation.

3.2.1 Mini Floating Point

Mini Floating-Point representation is a particular version of FP where numbers are expressed with a lower number of bits with respect to the IEEE standard. As in the 32-bit and 64-bit versions, this format is described by a sign, an exponent and a mantissa.
According to figure 3.1, the total number of bit is $S + E + M$, where S is typically one, while the rest of the total bits can be divided between M and E as needed.
Following the IEEE standard, a value is expressed as:

$$value = (-1)^{sign} \cdot mantissa \cdot 2^{exponent-bias}$$

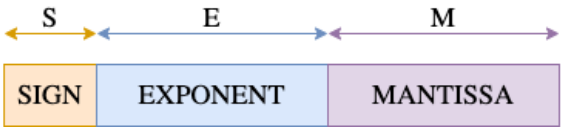


Figure 3.1: Floating-Point Bit Division

In mini FP the bias term may be considered as another degree of freedom. However, the IEEE standard expression is typically considered:

$$bias = 2^{E-1} - 1$$

The adoption of this format in Neural Network leads to a reduction on parameters and features sizes, with an almost null dropout on the accuracy. Further improvements on the complexity of the system can be achieved by neglecting the special cases of the IEEE standard: denormal values, infinite and NaN. In particular, denormalized values are quite computationally expensive, thus their absence can bring benefits on the performances, even if the range of represented values will be smaller.

3.2.2 Fixed-Point

Fixed-point numbers are integer values scaled by a fixed factor, which is a power of 2. It is based on the concept of the *binary point*, that, as the decimal one, divides the integer and the fraction part.

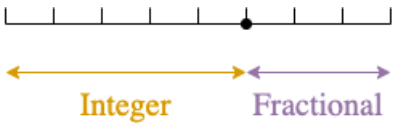


Figure 3.2: Fixed-Point Bit Division

The binary point position, which is equivalent to the number of bits for the fractional part, identifies the scaling factor to be applied to get the correct Fixed-Point number. Considering B as the number of total bits, F binary point number and x_i the bit at position i, the represented Fixed-Point values

is:

$$value = int \cdot 2^F = (-2^{B-1} \cdot x_{B-1} + \sum_{i=0}^{B-2} 2^i \cdot x_i) \cdot 2^F$$

With respect to FP representation, Fixed-Point requires a simpler Hardware, with further improvements on the power consumption and speed points of view. However, the cost of these progresses is a reduced dynamic range.

Dynamic Fixed Point

In Dense Neural Network, the dynamic ranges of the values in the layers can be very different from each other. The total dynamic range of the network is typically very large and Fixed-Point representation has not the capabilities to cover it with an acceptable quantization error.
For this reason, dynamic fixed point could be a good solution to achieve higher accuracies. Each layer is represented with a different Fixed-Point format: values of a layer share the same fractional length, but values of different layers may have different binary point positions.
A further improvement can be achieved if the values of each layer are divided in other three groups. In fact, the dynamic ranges of the inputs, the outputs and the parameters of a layer are typically quite different: parameters are small, while the inputs and the outputs of the layers are the results of a large number of accumulations. A different Fixed-Point mapping for each of the three types of information increases the covering capability of the representation.

3.2.3 Power Of 2

This representation approximates values as the closer powers of 2. Numbers could be just represented by their sign and their exponent, reducing the overall number of bits used to represent the value:

$$value = (-1)^{sign} \cdot 2^{exponent}$$

This approach has high benefits also on the computation side. In fact, multiplications between a value and a power of two can be implemented just by *shift operations*. Typically in NN, multipliers require most of the area and the power of the system thus Power of 2 values can greatly benefit the system.

3.3 Examples of Quantized Models

In the last five years, research has pushed towards the identification of increasingly optimized and quantized networks. In [27] a layer-by-layer approach was introduced to quantize networks from 32-bit Floating-Point to Fixed-Point representation. For each layer, firstly a statistical analysis of the weights was performed. From the obtained range, the integer and the fractional bit-widths were selected to cover the possible values and the accuracy of the new network was computed. This procedure was then used for the analysis of the weights of all the following layers. After the identification of the formats for the weights, the process was iterated for the data quantization. This approach allowed to have different total bit-widths for each layer. The method was applied to a simple CNN, the Lenet5, obtaining good results in terms of accuracy degradation: smaller than 0.2%.
Another example is [28], where a quantization investigation was carried out on three simple CNNs, considering different representations: starting from single precision Floating-Point, different (32-bit, 16-bit, 8-bit) Fixed-Point representations and power of two weights representation were tested. In order to faster identify the correct formats Ristretto had been considered. Moreover, the aware-quantize approach had been adopted, to increase the accuracy: the maximum dropout was around the 6%. The bit reduction proposed in [29] and [30] gone even further, introducing binary representation format obtained through aware-training quantization. The approach was applied to a simple CNN, ConvNet, achieving very good results with an almost null accuracy degradation.
Even if papers reported good reactions to quantization, it should be pointed out that most of the results found in the literature take into account only very simple networks, while investigations on Deep Neural Network are left behind.

3.4 Quantization Tools

Neural Networks are typically implemented in software through tools like TensorFlow [31], an open-source library developed by Google, Caffe [32] [33], an open-source framework developed by Berkeley University, or Keras [34], an

open-source library in Python. The growing interest in mobile applications has led to the development of several tools to help the description of a quantized NN. In particular, Tensorflow and Caffe have been expanded to support quantization.

3.4.1 Tensorflow Quantization: Tensorflow Lite

At the end of 2017, Google announced a new software, Tensorflow Lite [35], for the development of NNs for mobile and embedded devices. Starting from a full-precision pre-trained network described in Tensorflow, this extension is able to quantize a network into 16-bit Floating-Point (FP) or 8-bit integer representation. During the last year the software has been updated and now it supports four different types of quantization: integer weights, float16, integer, aware training.

Integer Weights Quantization Tool

Integer weights quantization tool converts weights from 32-bit Floating-Point to 8-bit integer. Therefore, the model size is reduced by a factor 4. Features are instead stored in full-precision: during inference they are converted to integers and re-converted to FP after processing.

Float16 Quantization Tool

Float16 quantization tool converts weights from 32-bit FP to 16-bit FP. The model size is halved and the floating point flexibility in covering a wider dynamic range is maintained.

Integer Quantization Tool

Integer quantization tool was introduced in 2019 as an update of integer weights quantization tool. This approach is able to convert from 32-bit FP to integer representation both the parameter and the features of a network. In particular different conversions are possible:

- 8-bit weights and feature;

- 16-bit features and 8-bit weights.

Aware Training Quantization Tool

Aware Training quantization tool was released only in April 2020. It is able to consider weights quantization also during training, converting them from 32-bit FP to 8-bit integer representation. As already explained in section 3.1, this type of quantization leads to higher accuracies especially for integer representations.

3.4.2 Caffe Quantization: Ristretto

Ristretto [36] is an extended version of the Caffe framework. It is able to perform post-training quantization on a NN. Furthermore, it can carry out aware-training quantization on the obtained quantized network, thanks a fine-tuning procedure that takes into account quantized parameters.
The tool supports three different quantization formats: Mini Floating-Point, Dynamic Fixed-Point and Power of 2 Weights. Starting from a pre-trained network described in the Caffe format, the tool is able to perform a dynamic range analysis of the parameters, the inputs and the outputs of each layer in order to identify the correct bit division of the chosen format. Ristretto then tests the network considering different bit-width representation of the same format, identifying the one with smaller number of bits but acceptable accuracy. The information of each layer can be quantized with a different bit use (exponent-mantissa or integer-fractional). Moreover, in Dynamic Fixed-Point representation, inputs, outputs and parameters of the same layer may be quantized with a different number bits. The quantized network can be re-trained considering quantization, to achieve better performance.

The higher flexibility of Ristretto with respect to the different formats and the different bit-widths makes it the ideal tool for the quantization analysis of this thesis. For this reason, in the section 3.2 the representation formats supported by Ristretto are explained in detail. Furthermore, the Ristretto tool is further described in appendix A.

4 Case of Study: DenseNet

The aim of this work is the analysis of the reaction of Deep Neural Networks to quantization, limiting as much as possible the accuracy reduction.
The investigation focuses on the quantization of two different versions of a feature extractor, both implemented through a DenseNet structure. In this chapter, the reference architectures are described and the adopted quantization strategy is depicted.

4.1 Reference Networks

The driving examples of this quantization analysis are two feature extractor, presented in [37] and [38]. In particular, the reference networks are two reduced versions of a DenseNet-121: referring to the formalism in [37] and [38], DenseNet 2.0 and DenseNet 1.0 are the chosen feature extractors to which quantization is applied.

The models have been obtained through the Knowledge Distillation of a larger network based on the ResNet-34 structure. This "teacher" network is characterized by almost 6 millions parameters. The two "students" have been trained to mimic the behavior of this network. In particular, the train set was made by tuple (I,T): the inputs I are images from the Casia Web Face [39] database while the targets T are the corresponding outputs of the "teacher". The adopted loss function, minimized by the training process, is the Euclidean distance between the targets and the output features of the distillated networks.

"Students" require a much lower number of parameters with respect to the original ResNet: DenseNet 2.0 is able to reduce the size by a factor 3.7, while DenseNet 1.0 by a factor of 14.6.
Furthermore, all the distillated networks have been trained to recognize smaller images. In fact, the "teacher" works with 150x150 input images, while the DenseNets are able to deal with 80x80 images.
Thanks to the reduction of the number of parameters and the size of the input images, the distillated networks require a smaller amount of memory and fewer computations.

Both the networks have been implemented in Python through the Tensorflow and Keras libraries.

4.1.1 Internal Structures

The two reference architectures differ in depth and number of parameters:

- DenseNet 1.0 is made by a 50 Keras layers, for a total number of parameters equal to 1.48M;
- DenseNet 2.0 is made by a 139 Keras layers, for a total number of parameters equal to 381K;

Nevertheless, the networks have similar internal structures and common groups of layers can be identified within the two.
DenseNet 1.0 can be divided in three main sections: input, dense blocks sequence, output. The same partition is present in DenseNet 2.0, which however has a further type of layers, grouped together in the transient section. The overall structures as shown in figure 4.1.

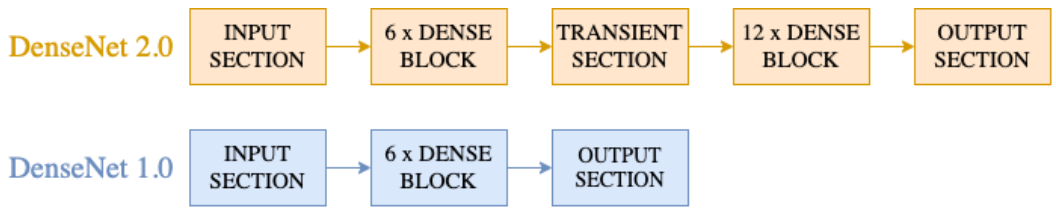


Figure 4.1: DenseNets 2.0 and 1.0 Internal Partition

Input Section

The first group of layers (figure 4.2) is the same for both the networks and it is made by a Convolutional layer and a Maximum Pooling layer, both involving a padding mechanism. The activation function of the Conv layer is a ReLU function. Furthermore, Batch Normalization is considered.

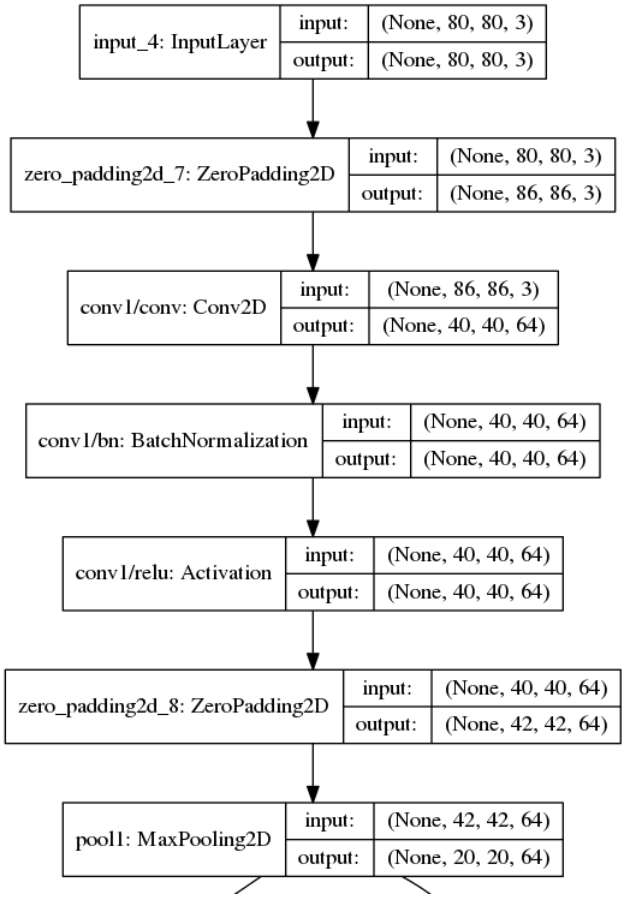


Figure 4.2: Input Block in Keras

Dense Blocks Sequence Section

As already explained in section 2.3, the main characteristic of a dense block is the connection with all the following blocks of the same type. These connections can be easily implemented in Keras through the employment of *concatenation* layers. In the two DenseNets, a sequence of these type of blocks is

considered. In particular:

- DenseNet 1.0 has 6 sequential dense blocks, placed after the input layers;
- DenseNet 2.0 has a first sequence of 6 dense blocks and a second sequence of 12 dense blocks. The first sequence is placed after the input layers and the two series of dense blocks are divide by the transient section described below;

Each dense block (figure 4.3) is made by a concatenation of Convolutional layer and Batch Normalization. The adopted activation function is a ReLU.

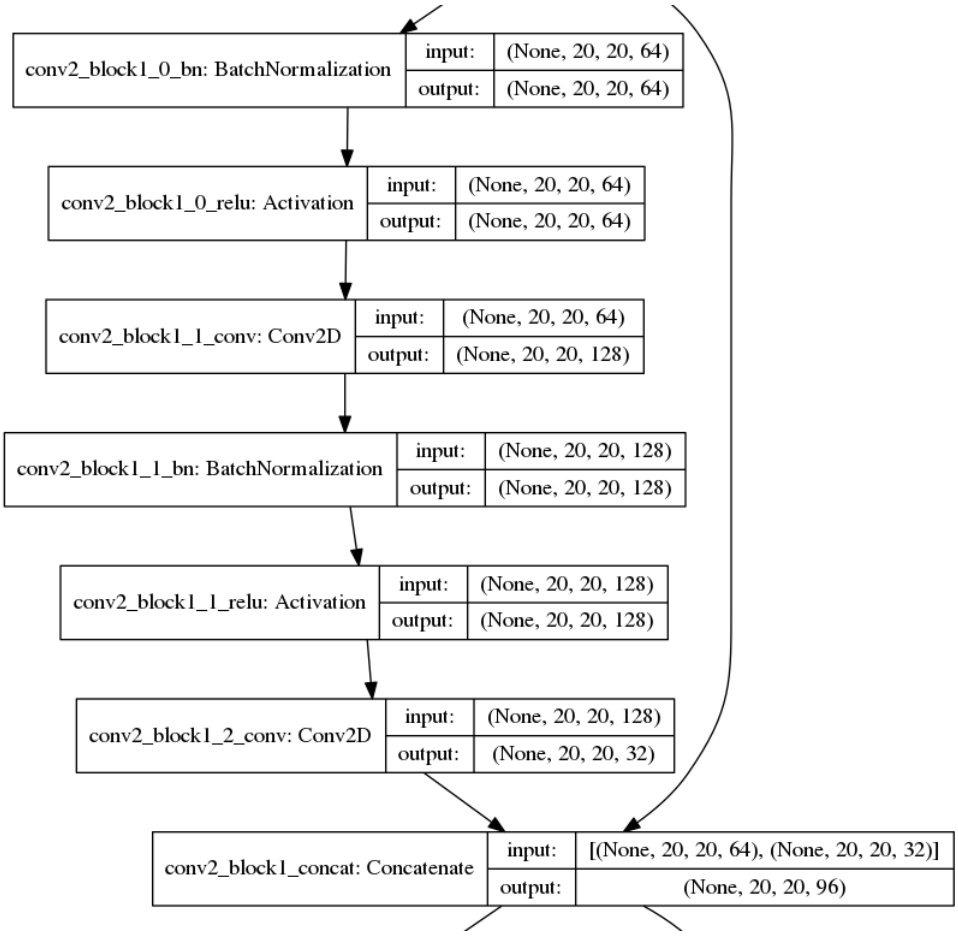


Figure 4.3: Dense Block in Keras

Transient Section

Only in DenseNet 2.0 a transient series of layers is present between the two different sequences of dense blocks. As shown in figure 4.4, it is made by a combination of Batch Normalization, Convolutional and Average Pooling layers.

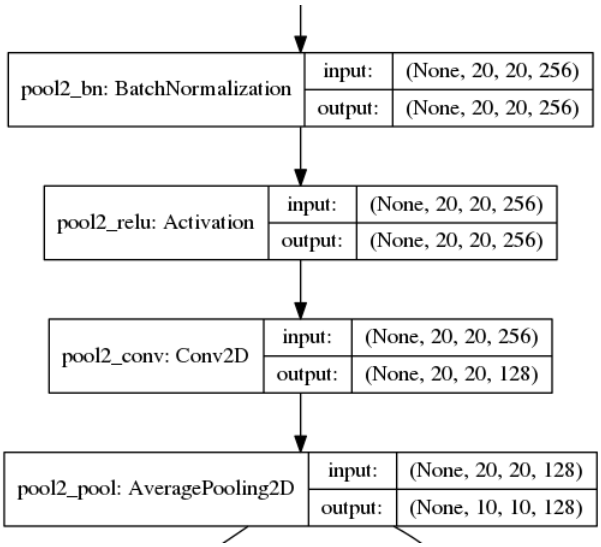
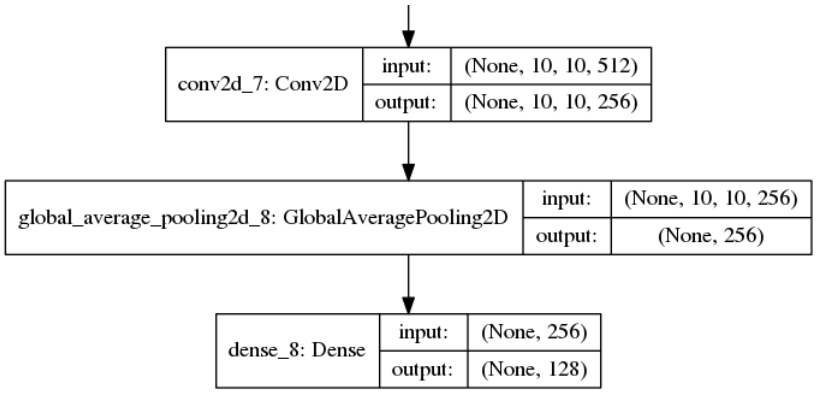


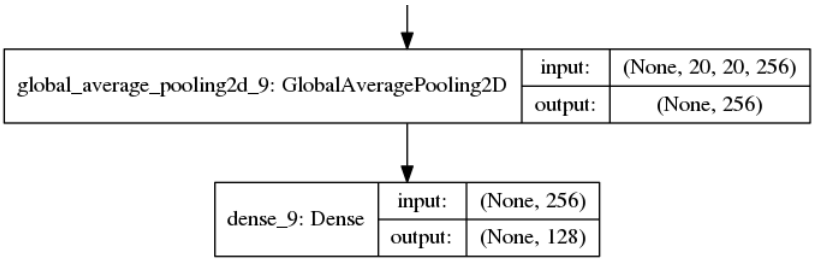
Figure 4.4: Transient Section in Keras

Output Section

The last layers for both the networks are a Fully Connected (FC) and Global Average Pooling layers. The latter is a particular type of Pooling layer: the kernel size is equal to the input matrix, therefore each channel is compressed by the average of its values.
DenseNet 2.0 has a further layer, a Convolutional one.
The two complete output sections are shown in figure 4.5.



(a) DenseNet 2.0



(b) DenseNet 1.0

Figure 4.5: Output Sections in Keras

4.2 Quantization Strategy

The Dense Convolutional Networks described in section 4.1 are the starting point for the carried out post-training quantization analysis. Specifically, this work focuses on inference, while training is laid aside.

A good trade-off between prediction accuracy and power consumption is mandatory for face recognition purposes.
A layer-by-layer approach is a useful strategy to minimize the introduced quantization error: each layer is processed independently and, within it, parameters and features are examined separately.
The solution set for the different quantization schemes is peculiarly broad. For that reason an automatic range analysis can be adopted to speed up the quantization process. From this point of view, the Ristretto framework, mentioned in

section 3.4.2, is a suitable solution: it is able to analyse the dynamic ranges of the parameters and the features of each layer in a CNN and to identify the best bit width, providing an approximation of the accuracy related to the selected quantization strategy. Further details on Ristretto are reported in appendix A.

Several quantization types have been considered to verify their different effects on accuracy and model compression. In particular, two different families of quantization techniques have been taken into account: scalar quantization and vector quantization.
Quantized results have been then compared to the 32-bit Floating-Point reference representation to identify the solutions with an acceptable accuracy degradation. Moreover, the different versions of the feature extractor allow observations to be made also regarding reactions to quantization with respect to the different depths of the networks.

4.2.1 Scalar Quantization

In scalar quantization each input value is processed separately: the quantized output is selected as the nearest value from a fixed set.
The traditional approaches adopted in HW design are the ones described in section 3.2. In particular, the main characteristic of these formats are:

- *Mini Floating-Point* representation, described in section 3.2.1, is typically able to strongly reduce the storage requirements with an almost null degradation on the accuracy. The computational cost is reduced too, but the Floating-Point operations still leave an high computational complexity;
- *Dynamic Fixed-Point* representation, described in section 3.2.2, is able to strongly reduce both the computational and memory requirements. The cost of this lowering of complexity is a deterioration in the prediction accuracy;
- *Power of 2 Weights* representation, described in section 3.2.3, further reduces the computational complexity thanks to its ability to map multiplication through shifters. However, the accuracy degradation can be even higher than the one in Dynamic Fixed-Point representation.

4.2.2 Vector Quantization

In vector quantization inputs are grouped together in a vector. The vector is then divided into groups of similar values, each represented by its centroid.
The processing of inputs as a single entity allows for optimal quantization, especially for large set of values. Nevertheless, this procedure increases the computational complexity with respect to scalar quantization.
This approach can be adopted for parameter quantization in NN, as the procedure can be performed prior to inference.
One famous vector quantization algorithm is *k-means clustering*.

K-Means Clustering

K-Means Clustering [40] is a particular vector quantization algorithm. It divides the N input values into K groups, where $N < K$. For each subset of values, a centroid is identified in order to minimize the cluster variances identified by the square Euclidean distance:

$$\arg\min_{C} \sum_{i=1}^{K} \sum_{x \in C_i} \|x - \mu_i\|^2$$

where K is the number of clusters, C_i is one of the K clusters in C, x is an input to be quantized and μ_i is the centroid of the cluster C_i.

Thinking about a future HW implementation of a Neural Network, the parameters could be approximated by a K number of values. The K centroids would be stored with a proper number of bits Q, while each parameter would be identified by an index that refers to the cluster to which it belongs. These indexes would require a number of bit $I = \log_2 K$ much lower than the number of bits Q required by the centroids. By properly selecting the numbers of bits Q and I, a Neural Network could be strongly optimized from the point of view of the required memory: storage power could be reduced while accuracy may remain nearly unchanged.

The main problem of the K-Means algorithm is its complexity, which grows rapidly as the number of clusters K and the number of inputs N increase.

K-Means Clustering can be implemented in Python, thanks to the *sklearn* library and its internal class *KMeans* [41]. An important feature of this class is the capability to verify if the selected number of clusters is above the optimal value for the current input set: above this optimal K, the efficiency of the algorithm saturates.

4.3 Layer-by-Layer Investigation

As claimed in section 4.2, a layer-by-layer analysis is a good approach to quantize large networks, as it limits the accuracy degradation by analyzing each single layer separately. This is especially true for Fixed-Point and Power Of 2 approximations, which can introduce larger quantization errors within a network.

DenseNets are typically made by four different types of layers: Convolutional, Fully Connected, Pooling and Batch Normalization layers.
Quantization may be critical in Convolutional, Fully Connected and Batch Normalization layers, widely used within the networks, while is less crucial in Pooling Layers.

4.3.1 Pooling Layers

Pooling layers are less affected by quantization. Furthermore in Maximum Pooling computations do not introduce any kind of error in the results: the output is just equal to one of the inputs.
In these particular driving examples, only two or three Pooling layers are present, respectively within DenseNets 1.0 and 2.0: the first one is a Maximum Pooling and it is placed in the Input Section, the second one is a Average Pooling with kernel size equal to 2 and it is present in the DenseNet 2.0 Transition Section, and the last one is a Global Average Pooling in the Output Section of the networks. The first layer do not introduce any kind of error also considering quantization while the last two may introduce small errors due to the average process. The Average Pooling error is very small because just a

division by 4 is required, the Global Pooling performes a division by an higher number, so the error may be larger. However the distortion propagates in just one another layer and so its effects can be negligible.

4.3.2 Convolutional and Fully Connected Layers

Convolutional and Fully Connected layers are most responsible for the complexity of deep Neural Network. Indeed, the effort of a network can be divided in two main contributions: arithmetic operations and memory capability.
Typically Convolutional layers are responsible for more than the 90% of the required computational resources, while Fully Connected layers are responsible for over the 90% of the required storage, due to their high number of parameters. Moreover, Deep Neural Networks currently consist on hundreds or thousands of these layers. Therefore the high number of accumulations can lead to large quantization errors, making quantization a problem for the correct behavior of a deep network.

Both DenseNet 1.0 and DenseNet 2.0 consist on just one FC layer in the Output Section. Regarding Convolutional, both the networks are made by a huge number of these layers: DenseNet 1.0 is made by 13 Conv layers, while DenseNet 2.0 by 39 ones.
The layers have been quantized as Mini Floating-Point, Dynamic Fixed-Point and Power of 2. To identify the correct bit-width and the correct bit division, a dynamic range analysis has been carried out thanks to the Ristretto framework. In particular, inputs, outputs and parameters of each layer have been examined on their own.

4.3.3 Batch Normalization Layers

As depicted in section 2.1.1, Batch Normalization is an essential operation for the speed-up of the training process. Nevertheless, it leads to an higher complexity during inference, especially because of the required arithmetic operations, such as multiplication, division and square root.
In the two reference architectures, a huge number of Batch Normalization Layers is present, so the related complexity can be significant: DenseNet 1.0

consists on 13 BN layers, while DenseNet 2.0 on 38 layers.
To solve this problem a particular technique can be adopted during inference: *Batch Normalization Folding*.

4.3.4 Folding

Folding is a widely used approach for the HW implementation of Batch Normalization layers during inference and it is based on the fact that training and inference can be seen as two different tasks. [42], [43], [44] and [45] are just few examples where it is adopted.
Following this technique Batch Normalization is merged with a preceding Convolutional or Fully Connected layer during inference: this mechanism is possible because during this task the parameters of a BN are fixed. The resulting layer is a Convolutional or Fully Connected one, with modified parameters:

$$\begin{cases} z = W \cdot x + B & \text{Conv } or \text{ FC} \\ y = \gamma \frac{z-\mu}{\sqrt{\sigma^2+\epsilon}} + \beta & \text{BN} \end{cases}$$

$$y = \gamma \frac{(W \cdot x + B) - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

$$\implies y = W_f \cdot x + B_f$$

$$where \quad W_f = \frac{\gamma \cdot W}{\sqrt{\sigma^2 + \epsilon}}, \quad B_f = \gamma \frac{B - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

The manipulation of the characteristic expressions within a network leads to:

- Simplification of the involved operations: taking into account the new parameters, division and square root are avoided;
- Reduction of the total number of layers: less operations are required, therefore computational complexity is reduced while speed is improved;
- Reduction of the total amount of parameters, therefore the required memory capability is reduced.

Folding may introduces small variations on the output features but it does not involved any accuracy degradation.

The approach can be extended to those Batch Normalization layers which are not preceded by Conv and FC layers. It is quite important for the quantization of the reference DenseNet, since half of their Batch Normalization layers follow concatenation or pooling layers.
In practise, even by itself a BN layer can be seen as Convolutional one by manipulating its equation. The resulting layer has a 1x1 kernel:

$$y = \gamma \frac{z - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

$$\implies y = W \cdot x + B$$

$$where \quad W = \begin{pmatrix} \frac{\gamma_1}{\sqrt{\sigma_1^2+\epsilon}} & 0 & \cdots & 0 \\ 0 & \frac{\gamma_2}{\sqrt{\sigma_2^2+\epsilon}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\gamma_N}{\sqrt{\sigma_N^2+\epsilon}} \end{pmatrix},$$

$$B = \begin{pmatrix} \beta_1 - \frac{\gamma_1 \mu_1}{\sqrt{\sigma_1^2+\epsilon}} \\ \beta_2 - \frac{\gamma_2 \mu_2}{\sqrt{\sigma_2^2+\epsilon}} \\ \vdots \\ \beta_N - \frac{\gamma_N \mu_N}{\sqrt{\sigma_N^2+\epsilon}} \end{pmatrix}$$

These extended approach is able to simplify the involved operations. Moreover, the total number of parameters to be stored can be reduced. Indeed weights are organized as matrices in which only the diagonals contain non-null values. Taking into account a proper HW implementation, only these non-null values actually need to be stored. Therefore, the total number of parameters to be stored can be halved: considering N the number of filters, original BN layers require a total of 4N parameters (μ, $sigma$, γ and β) while the new BN-Conv layers requires just 2N parameters (weight and bias).

In addition to all the benefits deriving from the reduction of parameters and layers, the adoption of these two techniques allows to quantize Batch Normalization through the Ristretto framework, speeding-up the process.

4.4 Network Preparation for Ristretto

As declared in section 4.1, DenseNets 2.0 and 1.0 have been described and trained thanks to Keras and Tensorflow libraries. However, quantization will be performed with the help of the Ristretto framework. To work, this tool requires:

- A pre-trained network described in Caffe format. A conversion from Tensorflow to Caffe is needed;
- An input tuple of images, to analyze the network. Inputs can be provided considering a Lightning Memory-Mapped Database (LMDB), an high efficient database in the format image-key. The database stores each input image and the corresponding label. This label will be used by Ristretto in the accuracy evaluation;
- A network considering a classifier, due to the way accuracy is evaluated during the quantization process. The DenseNets are just feature extractors, so a classifier has to be added after them.

The LMDB database can be easily assembled through a script available in the Caffe material. The other two requirements are covered below, as they need a more in-depth explanation.

4.4.1 From Tensorflow to Caffe

A network is described in the Caffe format by two files: a prototxt file holds the internal structure while a caffemodel file stores the parameters of each layer described in the prototxt.

Regarding the prototxt file, most of the Tensorflow and the Caffe layers are quite similar, what differs is just the formalism.

The main difference between the two is on the implementation of Batch Normalization. In fact, the Tensorflow BatchNormalization() layer is equivalent to a sequence of two Caffe layers: the BatchNorm layer and the Scale layer (figure 4.6). The former subtracts the mean from the inputs and divides them by the variance. The latter is used to perform the scaling and shifting of the normalized distribution.

```
layer{
    name: "batch1"
    type: "BatchNorm"
    bottom: "conv1"
    top: "batch1"
    batch_norm_param{
        use_global_stats: true
    }
}
layer{
    name: "scale1"
    type: "Scale"
    bottom: "batch1"
    top: "scale1"
    scale_param{
        bias_term: true
    }
}
```

Figure 4.6: Batch Normalization in Caffe

To create the caffemodel file, firstly the parameters of the trained DenseNets have to be extrapolated from their Tensorflow descriptions. These values can be then assigned to the Caffe network and saved in the caffemodel file.
A special attention should be paid to the weight assignment. In fact, Tensorflow and Caffe organize the weights with different orders:

$$\boldsymbol{Tensorflow} \longrightarrow Height_{kernel} \cdot Width_{kernel} \cdot Depth_{input} \cdot N_{filter}$$

$$\boldsymbol{Caffe} \longrightarrow N_{filter} \cdot Depth_{input} \cdot Height_{kernel} \cdot Width_{kernel}$$

The extrapolated weights have to be reordered from the Tensorflow to the Caffe format, before the generation of the caffemodel file.

4.4.2 Classifier

The distillated networks from [37] and [38] are dense feature extractors. These systems can be adopted for different purposes:

- for face identification or for face verification. The former is defined as the capability to identify a person (one-to-many approach), while the latter is the capability to validate an identity (one-to-one approach);
- for close set or open set faces. For the former, the network is able to deal with a finite number of identity while for the latter the network can work with an infinite number of faces. Typically, face recognition with an open set of faces requires an "unknown ID" class, to classify all the images not belonging to the reference database.

For all the possible applications, a feature extractor has to be followed by a classifier to complete the identification process.
In particular, the Ristretto tool is based on face recognition with close sets of faces: a finite number of known identities has to be selected and a classifier has to be designed. For this last purpose, a Multi-Layer Perceptron (MLP) is a good solution to implement the required structure. This type of Neural Network is made by three or more Fully Connected layers and non-linear activation functions.
Following the work in [38], a Multi-Layer Perceptron made by three FC layers was adopted (figure 4.7): the first two layers consists on 100 nodes, while the last one on 30 nodes. This last quantity is equal to the chosen number of known identities. The selected non-linear function was a ReLU.
Once the structure has been described, the parameters of the classifier have to be identified. Actually, two different training processes were performed in order to implement two classifiers, one for each DenseNet.
Firstly the two DenseNets were tested with the training images of the 30 known identities to extrapolate their output features. These inputs were selected from the train set of possible faces of the Labeled Faces in the Wild (LFW) [46][47][48] database.

After this testing process, two databases were assembled. Each database con-

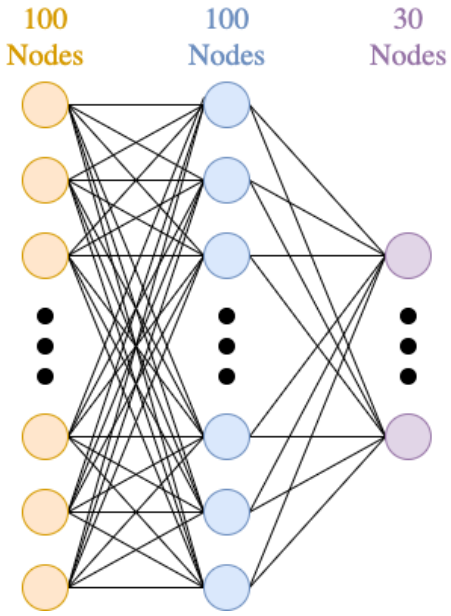


Figure 4.7: Classifier Architecture

sists on tuple (F, L) of feature F and corresponding label L. The former is an output feature from the distillated network, while the latter determines the corresponding "friend". The total number of tuple is 450: 15 images for each known identity.

The databases have been employed in training the classifiers, for 100 epochs on batches of 30 images. The resulting accuracy of the classifier related to DenseNet 2.0 was 99.77%, while the accuracy of the classifier for DenseNet 1.0 was 99.36%.
These quantities are the starting point for the Ristretto quantization, which requires a classifier. Nevertheless, the final results reported in section 5 refer to a testing procedure that involves only feature extractor contributions.

5 Quantized Networks

In order to verify the effectiveness of the quantization techniques, the accuracies of the reference Tensorflow networks have been computed and the Caffe versions have been validated. At this point a first quantization has been performed taking into account only Convolutional and Fully Connected layers: scalar quantization has been considered. After that, folding has been applied to the original architectures: the resulting networks have been validated and scalar quantization has been carried out, in order to take also into account Batch Normalization. A comparison between the quantization results with and without BN has been performed, underlining the problems related to Batch Normalization quantization. Lastly, K-Means Clustering has been taken into account to try to enlarge the quantization solution set with Batch Normalization layers.
Results are reported in terms of accuracy and, when present, compression rate. In particular, each version of the networks has been verified following the LFW test, a standard procedure for NN verification. The LFW test has been applied only to DenseNet 2.0 and DenseNet 1.0, while the classifiers required by Ristretto have not been considered.

5.1 Labeled Faces in the Wild

Labeled Faces in the Wild [46] [47] is a popular database used for NN verification. It has been proposed in 2007 to address the *pair matching* problem for which face recognition deals with deciding whether two images represent the same person or not.
The main characteristics of this database are:

- 13233 target images;
- 5749 different individuals among which 1680 people have more than one image;
- An unique name of each individual is provided and each person should appear only under one name;
- The input images are provided as 250x250 pixel JPEG and most of them are colored;
- Images represent individuals with a large range of variation in pose, lighting, expression, background, race, ethnicity, age, gender, clothing, hairstyles, camera quality, color saturation, focus.

In the database site [48], images are present together with different txt files, which report the labeling for different sub-databases. Two main types of these subsets exist: txt files to identify matched pairs and txt files to identify mismatched pairs. An example of the possible pairs is reported in figure 5.1.

5.1.1 LFW Test

LFW test is divided in two main phases: *view 1* and *view 2*. The former has been created to assist researchers during model selection and algorithm development, while the latter should be used only for the final performance report of a network.

(a) Matched Pairs (b) Mismatched Pairs

Figure 5.1: Examples of Image Pairs from LFW Database [46]

The main steps of the performed test are reported below.

1. DenseNets have been tested by the LFW database and the output features have been extracted;
2. A binary classifier has been designed. The structure evaluates the Euclidean distance between a couple of features. If this distance is below a fixed threshold, the classifier marked the current couple as a matched pair otherwise as a mismatched pair. The accuracy of the classifier is computed as the ratio between the correct predicted pairs over the total number of input couples. The threshold of the classifier is the only trainable parameter of the network;

3. The classifier has been trained considering the couples of images established in the "pairsDevTest.txt" file, available in [48]. The training procedure evaluates the Euclidian distances between the features of the 2200 pairs and verifies the accuracies by varying the threshold between 0 and 1, with a step size of 0.1. The optimal threshold is selected as the one with higher accuracy;

4. The selected threshold is then used for the accuracy evaluation considering the couples in the "pair.txt" file [48]. In particular, 10 groups of 300 pairs with the same individual and 300 pairs with different individuals are provided. The test accuracy is evaluated as the mean of the results from the 10 different tests on the 10 groups. Moreover a standard error is evaluated to report the network performances.

$$\overline{acc} = \frac{\sum_{i=1}^{10} acc_i}{10}$$

$$s_{err} = \frac{\sigma}{\sqrt{10}} = \sqrt{\frac{\sum_{i=1}^{10}(acc_i - \overline{acc})^2}{9}}$$

5.2 Original Networks

Taking into account the LFW test, the original networks have been characterized and the Caffe networks have been validated. After that a first quantization has been performed on Convolutional and Fully Connected layers.

5.2.1 Full Precision

The Tensorflow DenseNets have been tested and the obtained LFW accuracies are reported in table 5.1. These quantities are the reference values for all the network versions covered in the next sections.

Network	Accuracy	Threshold
DenseNet2.0	$(98.05 \pm 0.20)\%$	0.54
DenseNet1.0	$(95.50 \pm 0.35)\%$	0.5

Table 5.1: LFW test accuracy of original DenseNet 2.0 and DenseNet 1.0

5.2.2 Caffe Network Validation

To quantize the networks through the Ristretto framework, the Caffe DenseNets have been implemented and validated through a comparison with the original versions. Therefore, the networks have been tested and their LFW accuracies have been computed. To compare the Caffe networks and the original Tensorflow ones, a difference between the accuracies has been evaluate.

Network	Accuracy	Accuracy Difference	Threshold
DenseNet2.0	$(98.05 \pm 0.20)\%$	0.00	0.54
DenseNet1.0	$(95.50 \pm 0.35)\%$	0.00	0.5

Table 5.2: LFW test accuracy for Caffe versions of DenseNet 2.0 and DenseNet 1.0: Accuracy Difference is computed between the Caffe accuracy and the Tensorflow accuracy reported in 5.1

As it can be noticed in table 5.2, the Caffe and the original DenseNets are characterized by the same exact results in terms of accuracy, accuracy error and threshold, proving the correctness of the Caffe networks.

5.2.3 Scalar Quantization

After the Caffe implementation of DenseNet 1.0 and DenseNet 2.0, Convolutional and Fully Connected layers have been quantized. This partial quantization has been considered to verify the contributions of different layers in the quantization error. In fact, in the literature it is often highlighted that Convolutional and Fully Connected layers are more easily to be quantized with respect to Batch Normalization, which instead typically requires an aware-training procedure.

Quantization has been conducted taking into account *Mini Floating-Point*, *Dynamic Fixed-Point* and *Power of 2 Weights*. Different optimal bit widths and different bit divisions have been achieved.

Mini Floating-Point

Adopting a Mini FP quantization, both the networks can be reduced from 32 bits to 8 bits, with a total reduction by a factor 4 of the network sizes. The number of bits required for the exponent and the mantissa of the represented values for both the networks are reported in table 5.3.

Network	Total Bits	Exponent Bits	Mantissa Bits
DenseNet2.0	8	5	2
DenseNet1.0	8	5	2

Table 5.3: Bit Width and Bit Division After Mini FP Quantization on Conv and FC Layers

The quantized FP versions have been tested through the LFW test and the accuracies are reported in table 5.4. Both the networks react well to this type of quantization. DenseNet 1.0 even achieves an higher accuracy with respect to the reference model.

Network	Accuracy	Accuracy Degradation
DenseNet2.0	$(97.78 \pm 0.18)\%$	0.27
DenseNet1.0	$(96.05 \pm 0.33)\%$	not present

Table 5.4: LFW test accuracy of DenseNet 2.0 and DenseNet 1.0 After Mini FP Quantization on Conv and FC Layers: Accuracy Degradation is computed between the Caffe accuracy and the Tensorflow accuracy reported in 5.1

Dynamic Fixed-Point

The quantization of the DenseNets considering Dynamic Fixed-Point representation achieves good results. In particular, 8 bits can be adopted to represent

the features of Convolutional layers and all the weights, moreover 4 bits are enough to represent the Fully Connected features. Therefore, this quantization is able to reduce the sizes of the networks by a factor slightly higher than the Mini FP version (factor $\simeq 4.02$). The internal bit division is then different for weights, input and output features of each layer.

Network	Conv Weight Bits	Conv Feature Bits	FC Weight Bits	FC Feature Bits
DenseNet2.0	8	8	4	8
DenseNet1.0	8	8	4	8

Table 5.5: Bit Widths After Dynamic Fixed-Point Quantization on Conv and FC Layers

The quantization of Conv and FC layers in DenseNet 2.0 and DenseNet 1.0 reaches good results even on accuracy side, with very small or even null degradation with respect to the 32 bit reference architecture.

Network	Accuracy	Accuracy Degradation
DenseNet2.0	$(96.87 \pm 0.27)\%$	1.18
DenseNet1.0	$(96.03 \pm 0.38)\%$	not present

Table 5.6: LFW test accuracy of DenseNet 2.0 and DenseNet 1.0 After Dynamic Fixed-Point Quantization on Conv and FC Layers: Accuracy Degradation is computed between the Caffe accuracy and the Tensorflow accuracy reported in 5.1

Power of 2 Weights

Power of 2 Weights quantization expresses features in Dynamic Fixed-Point, while weights are approximated by the closer power of two: only the exponent of this power is stored. Adopting this type of quantization on Convolutional and Fully Connected layers, 8 bit can be used to express the features, while 5 bits for the exponents of the weights: parameters are represented by 32 different powers of 2.

Network	Conv and FC Weight Exponent Bits	Conv Feature Bits	FC Feature Bits
DenseNet2.0	5	8	8
DenseNet1.0	5	8	8

Table 5.7: Bit Widths After Power of 2 Weights Quantization on Conv and FC Layers

Unlike the other types of quantization, Power of 2 Weights introduces a significant degradation of performance. This can be explained by the greater approximation error that this representation inserts in the networks.

Network	Accuracy	Accuracy Degradation
DenseNet2.0	$(85.52 \pm 0.54)\%$	12.53
DenseNet1.0	$(72.73 \pm 0.61)\%$	22.77

Table 5.8: LFW test accuracy of DenseNet 2.0 and DenseNet 1.0 After Power of 2 Weights Quantization on Conv and FC Layers: Accuracy Degradation is computed between the Caffe accuracy and the Tensorflow accuracy reported in 5.1

5.3 Folding

After the partial quantization depicted in section 5.2.3, folding has been applied to simplify the Batch Normalization implementation during inference and to introduce the BN quantization. As explained in section 4.3.4, this technique is also able to compress the network size by acting on the numbers of required layers and parameters. The effective reductions introduced in DenseNet 2.0 and DenseNet1.0 are reported in table 5.9.

Network	Layer Reduction	Percentage Layer Reduction	Parameter Reduction	Parameter Layer Reduction
DenseNet2.0	19	$\approx 14\%$	19008	$\approx 1.3\%$
DenseNet1.0	7	$\approx 13.7\%$	5056	$\approx 1.3\%$

Table 5.9: Layers and Parameter Reduction After Folding

5.3.1 Full Precision

Before quantization, the folded versions of the DenseNets have been validated. Folding involved only a manipulation of the characteristic equations inside the networks, so it should not modify the behavior of the systems. This statement is confirmed by the LFW test results, reported in table 5.10: the obtained accuracies are equal to the reference ones reported in section 5.2.1.

Network	Accuracy	Accuracy Difference	Threshold
DenseNet2.0	$(98.05 \pm 0.20)\%$	0.00	0.54
DenseNet1.0	$(95.50 \pm 0.35)\%$	0.00	0.5

Table 5.10: LFW test accuracy of DenseNet 2.0 and DenseNet 1.0 After Folding: Accuracy Difference is computed with respect to the Tensorflow accuracy reported in 5.1

5.3.2 Scalar Quantization

After the validation of the folded networks, the quantization has been applied to the architectures. Again, the procedure has been conducted taking into account *Mini Floating-Point*, *Dynamic Fixed-Point* and *Power of 2 Weights* and different optimal bit widths and different bit divisions have been obtained.

Mini Floating-Point

As for the original versions, the folded networks can be reduced from 32 to 8 bits by Mini FP quantization, compressing the sizes of the networks by a

factor 4. The number of bits required for the exponent and the mantissa of the represented values for both the networks are reported in table 5.11.

Network	Total Bits	Exponent Bits	Mantissa Bits
DenseNet2.0	8	4	3
DenseNet1.0	8	4	3

Table 5.11: Bit Width and Bit Division of Folded DenseNet 2.0 and DenseNet 1.0 After Mini FP Quantization

The quantized networks have been tested and the results are reported in table 5.12. Again, both the networks react well to this type of quantization.

Network	Accuracy	Accuracy Degradation
DenseNet2.0	$(97.9 \pm 0.19)\%$	0.15
DenseNet1.0	$(95.87 \pm 0.33)\%$	not present

Table 5.12: LFW test accuracy of Folded DenseNet 2.0 and DenseNet 1.0 After Mini FP Quantization: Accuracy Degradation is computed between the Caffe accuracy and the Tensorflow accuracy reported in 5.1

Dynamic Fixed-Point

Taking into account Batch Normalization, Dynamic Fixed-Point quantization is not sufficient to approximate the DenseNets. Indeed results are not good both from the point of view of bit reduction and accuracy.
Features can be represented by 8 bits and the weights of FC layers by 4 bits. Nevertheless, Convolutional weight representation is a problem: Ristretto is not able to identify a format with a number of bits lower than 32. The compression of the network is so strongly limited (reduction factor $\simeq 3$).

Network	Conv Weight Bits	Conv Feature Bits	FC Weight Bits	FC Feature Bits
DenseNet2.0	32	8	4	8
DenseNet1.0	32	8	4	8

Table 5.13: Bit Widths of Folded DenseNet 2.0 and DenseNet 1.0 After Dynamic Fixed-Point Quantization

Moreover, from the accuracy point of view both the networks are characterized by an high degradation.

Network	Accuracy	Accuracy Degradation
DenseNet2.0	$(75.18 \pm 0.66)\%$	22.87
DenseNet1.0	$(70.80 \pm 0.87)\%$	24.70

Table 5.14: LFW test accuracy of Folded DenseNet 2.0 and DenseNet 1.0 After Dynamic Fixed-Point Quantization: Accuracy Degradation is computed between the Caffe accuracy and the Tensorflow accuracy reported in 5.1

Power of 2 Weights

Taking into account Power of 2 Weights quantization, 8 bit can be used to represent the features inside the network, while 5 bits for the exponents of the weights: parameters are represented by 32 different powers of 2.

Network	Conv and FC Weight Exponent Bits	Conv Feature Bits	FC Feature Bits
DenseNet2.0	5	8	8
DenseNet1.0	5	8	8

Table 5.15: Bit Widths of Folded DenseNet 2.0 and DenseNet 1.0 After Power of 2 Weights Quantization

Also in this case, accuracies after quantization show a unacceptable degradation.

Network	Accuracy	Accuracy Degradation
DenseNet2.0	$(63.22 \pm 0.66)\%$	34.83
DenseNet1.0	$(57.82 \pm 0.74)\%$	37.68

Table 5.16: LFW test accuracy of Folded DenseNet 2.0 and DenseNet 1.0 After Power of 2 Weights Quantization: Accuracy Degradation is computed between the Caffe accuracy and the Tensorflow accuracy reported in 5.1

Conv-FC-BN Quantization vs Conv-FC Quantization

Analysing the accuracy results, Mini Floating-Point representation is the best scalar solution to quantize the deep networks involved in this work. In fact, this representation format is the only one able to get good results both in terms of model compression and accuracy.

Power of 2 Weights representation may have good performance in terms of computational cost, but the accuracy degradation is always too high. A fine-tuning training with quantized parameters may produce an increment in accuracy, making the network usable.

Quantization through Dynamic Fixed-Point representation achieves good results when only Conv and FC layers are considered. Nevertheless, when it is applied to the complete network, degradation becomes unacceptable. This difference is related to the higher sensitivity of Batch Normalization layers to quantization. In fact, this type of layers are the most critical from this point of view. Moreover, folding increases the number of outliers on the new parameters, making more difficult the coverage of the entire range of values with an acceptable precision. A good Batch Normalization quantization requires aware-training procedures as *fine-tuning with quantized parameters* or *freezed training* [25]. In particular for the latter approach, after the classical training technique the network is re-trained fixing the mean and the variance of the BN layers to the averages values used during inference.

5.3.3 K-Means Clustering

Due to the limited capability of Dynamic Fixed Point to deal with Batch Normalization quantization, K-Means Clustering (section 4.2.2) has been considered in order to enlarge the set of quantization solutions for large networks with Batch Normalization. It is a vector quantization techniques which mainly focuses on the reduction of the required memory.
For each network, three different scenarios have been taken into account:

1. All the parameters have been approximated by the same K-Means algorithm and so with the same set of clusters;

2. The parameters have been divided in three groups: classical Convolutional layers, folded Convolutional and Batch Normalization layers, BN layers represented by Convolutional ones with 1x1 kernel. Each group has been approximated with a different K-Means algorithm, therefore three different sets of clusters have been considered;

3. The parameters of each layers have been represented through a different K-Means algorithm. In the case of DenseNet 2.0 a total of 64 set of clusters has been considered, while in DenseNet 1.0 a total of 19.

For each scenario, different powers of 2 have been considered as the number of clusters K: from 8 to 4048. Among them, the optimal identified values in terms of accuracy and model compression are reported in table 5.17.

Network	Scenario 1	Scenario 2	Scenario 3
DenseNet2.0	512	512	16
DenseNet1.0	512	512	16

Table 5.17: Optimal number of clusters K

As described in section 4.2.2, each weight can be identified by an index pointing to a particular centroid value. It means that a number of bits $I = \log_2 K$ are sufficient to store this information, while the centroid can be stored by 32-bit maximum precision. Starting from this consideration, the compression rate of the K-Means Clustering approach have been identified. Results in

terms of reduction size of the parameters are reported in table 5.18: both the contribution of indexes and centroids are considered in the parameter reduction rate.

Network	Set of Clusters	K	I	Parameters Reduction Rate
DenseNet2.0	1	512	9	3.55
	3	512	9	3.54
	64	16	4	7.95
DenseNet1.0	1	512	9	3.54
	3	512	9	3.50
	19	16	4	7.94

Table 5.18: Parameters Reduction Memory Due to K-Means Clustering

Once the centroids of each cluster have been identified thanks to the *sklearn* library of Python, the weights of the original networks have been substituted with the centroid of the belonging cluster. The new approximated networks have been then tested and the LFW accuracies have been evaluated. The results are reported in table 5.19.

Network	Set of Clusters	K	Accuracy	Accuracy Degradation
DenseNet2.0	1	512	$(58.17 \pm 0.66)\%$	39.88
	3	512	$(58.00 \pm 0.45)\%$	40.05
	64	16	$(93.70 \pm 0.22)\%$	4.35
DenseNet1.0	1	512	$(56.22 \pm 0.66)\%$	39.28
	3	512	$(58.36 \pm 0.42)\%$	37.14
	19	16	$(93.02 \pm 0.44)\%$	2.48

Table 5.19: LFW test accuracy of DenseNet 2.0 and DenseNet 1.0 After K-Means Clustering: Accuracy Difference is computed with respect to the Tensorflow accuracy reported in 5.1

Optimal K-Means Approach

Among the considered scenarios, a different K-Means algorithm for each layer is the most suitable way to apply this vector quantization to the deep reference networks. In particular, this approach can be seen as an non-uniform quantization of the parameters in each layer: the representation power is similar much higher than the classical Dynamic Fixed-Point representation, allowing a better coverage of the folded parameters.

5.3.4 Hybrid Quantization: K-Means Clustering and Mini Floating-Point

Since K-Means Clustering has led to good results, a hybrid approach between scalar and vector quantization has been implemented to decrease even more the complexity and the size of the networks. This quantization focuses on the two best approximation techniques identified: Mini Floating-Point and K-Means Clustering applied to each layer with a different set of clusters.
The main characterists of this method are:

- Parameters are identified by an index pointing to a particular centroid;
- Centroids are stored considering Mini FP quantization;
- Operations are performed in Mini FP.

This new approach is able to reduce both the computational and storage requirements thanks to the benefits of the two involved techniques. An approximation of the reduction rates is reported in table 5.20.

Network	Set of Clusters	K	Parameters Reduction Rate	Network Reduction Rate
DenseNet2.0	64	16	7.99	5.33
DenseNet1.0	19	16	7.99	5.33

Table 5.20: Reduction Rate Due to K-Means Clustering and Mini Floating-Point

This hybrid quantization obtains good results also in terms of accuracy degradation.

Network	Set of Clusters	K	Accuracy	Accuracy Degradation
DenseNet2.0	64	16	$(93.63 \pm 0.23)\%$	4.42
DenseNet1.0	19	16	$(93.08 \pm 0.41)\%$	2.42

Table 5.21: LFW test accuracy of DenseNet 2.0 and DenseNet 1.0 After K-Means Clustering and Mini Floating-Point Quantization: Accuracy Difference is computed with respect to the Tensorflow accuracy reported in 5.1

6 Conclusions

Taking into account all the considerations reported in chapter 5, Mini Floating-Point and K-Means Clustering are the best identified results for the quantization of the reference DenseNets:

- Mini FP is able to reduce the size and the complexity of each architecture by a factor 4 with an almost null accuracy degradation;

- K-Means Clustering is able to compress the network sizes by a factor almost equal to 8, leading to a smaller required memory, with an acceptable degradation on the accuracies.

The hybrid quantization described in 5.3.4 is able to further increase the quantization performance, leading to a possible total network size reduction by a factor higher than 5 for both the DenseNets.
Nevertheless, the obtained results highlight a problem in the application of quantization on deep networks where the folding technique, which is a standard for the HW implementation of Batch Normalization layers during inference, is adopted. The issue is particularly relevant in the case of quantization through Dynamic Fixed-Point and Power of 2 Weights representations, due to the limited coverage capability of the two formats and to the significant introduced quantization errors. This problem is only partially visible in the literature because of the limited depth of the networks typically involved in quantization: for example in [27] and [28] acceptable accuracy degradation have been introduced by Fixed-Point or Dynamic Fixed-Point representations, but the involved networks (Lenet5 and AlexNet) are much simpler than the DenseNets involved in this analysis.

Future work can proceed in two different directions.
Firstly, DenseNet 2.0 and DenseNet 1.0 can be implemented in HW taking into account 8-bit Floating-Point and hybrid quantizations in order to verify the effective power consumption of the quantized networks. The hardware design can be carried out both thinking of silicon or FPGA implementation: indeed 8-bit is a bit-width supported by current devices.
Focusing on FPGA, this design can be performed taking into account Xiling Vivaldo Design Suite, a C-based software able to simplify and to automate the implementation of HW design. Among the different features of the tool, power, delay and area analysis of the implemented network can be automatically carried out, making simpler the investigation on effective power consumption reduction due to quantization.

Moreover, another possible step is the improvement of Dynamic Fixed-Point and Power of 2 Weights quantization, which have enormous potential from the point of view of power reduction. Aware-training quantization techniques, as *freezing* [25], can be considered to increase the accuracy of deep quantized networks.

A Ristretto Framework

The Ristretto [36] framework is an open-source software that automatically performs quantization on a Convolutional Neural Network: different bit-widths are considered to identify the best solution in terms of accuracy. It is an extension of the Caffe framework: its starting point is a trained network described in the Caffe format, stored in a prototxt file.

The framework concentrates on the quantization and the resource reduction during inference. In particular, the tool focuses on the optimization of Convolutional and Fully Connected layers. In fact, these two types of layer are the most consuming in terms of computational power and required memory: typically, Conv layers are responsible for the major amount of computations in a CNN while FC layers are responsible for the major amount of parameters to be stored.

Ristretto can work with three different types of data representation: Dynamic-Fixed Point, Mini Floating-Point or Multiplier-Free Arithmetic. Regarding the last format, weights are approximated with the closer power of 2 and so multipliers can be replaced by shifters.

A.1 Quantization Tool

Quantization is applied through the Ristretto command *quantize* (figure A.1). The user has to specify the following parameters:

- *Model*, the prototxt file where the full-precision network is described;

- *Weights*, the caffemodel file where the full-precision parameters of the network are stored;
- *Trimming Mode*, the quantization representation format (minifloat, dynamic_fixed_point or integer_power_of_2_weights) that the tool will analyze;
- *Quantized Model*, the name for the prototxt file where the quantized network will be described. This file is automatically created by the tool;
- *Error Margin*, the maximum acceptable percentage accuracy degradation with respect to the 32-bit FP network;
- *GPU*, the GPU ID, if Ristretto is used in GPU mode. If CPU mode is considered, the parameter must not be specified;
- *Iterations*, number of batch iterations.

```
./../../caffe/build/tools/ristretto quantize \
        --model=nets/DenseNet10_class30_train_test.prototxt \
        --weights=nets/DenseNet10_class30_tt.caffemodel \
        --model_quantized=nets/DenseNet10_clas30_quantized_fp.prototxt \
        --trimming_mode=minifloat --iterations=90 \
        --error_margin=3
```

Figure A.1: Ristretto *quantize* Command Example

To analyze the dynamic ranges of the input and output features, Ristretto requires a database to test the network. In particular, in the prototxt file the test and training LMDB databases have to be declared. This type of database stores each input image together with its label. The latter is used by the tool to verify the correctness of the network prediction during inference or training. It is important to point out that the tool is able to work only with networks including a classifier: the accuracy (used in the quantization procedure) is evaluated as the correctness of the predictions with respect to the labels of the database and not according to the euclidean distances between the expected features and the predicted ones.

The quantization procedure takes into account different bit-width representations for Convolutional and Fully Connected layers: layers of the same type

are characterized by an equal total number of bits.

The quantization flow consists on four main steps:

1. Analysis of the weights in the pre-trained full-precision network, to identify their dynamic ranges;
2. Processing of the test images to obtained the dynamic ranges of the input and output values of each layer;
3. Analysis of the overall network with respect to different numbers of total bits. This step has to identify the possible bit-width strategies that satisfy the user-defined error margin;
4. Creation of the prototxt file for the quantized network, considering the lower number of bits.

From the dynamic ranges Ristretto evaluates the number of bits for the integer part (Fixed-Point) or for the exponent (Floating-Point), which are fixed during the analysis with different bit-widths. The values are selected to avoid saturation. In particular, the adopted strategies are:

$$\#bit_{integer} = \lceil log_2(max\ x + 1) \rceil$$

or

$$\#bit_{exponent} = \lceil log_2(log_2(max\ x) - 1) + 1 \rceil$$

A.1.1 Ristretto Layers

Ristretto introduces two types of layer in the traditional Caffe catalogue: *ConvolutionRistretto* and *FcRistretto*.
The Ristretto layers require an additional set of parameters, grouped together under the name of *quantization_param*. Depending on the selected trimming mode, three types of quantization parameters can be generated.
Layers of the same type share the same total number of bits to represent information, while the internal bit division may be different.

Mini Floating-Point

```
layer {
  name: "conv1"
  type: "ConvolutionRistretto"
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 64
    bias_term: true
    pad: 3
    kernel_size: 7
    stride: 2
  }
  quantization_param {
    precision: MINIFLOAT
    mant_bits: 3
    exp_bits: 4
  }
}
```

(a) Ristretto Conv Layer

```
layer {
  name: "fc128"
  type: "FcRistretto"
  bottom: "global_pool"
  top: "fc128"
  inner_product_param {
    num_output: 128
  }
  quantization_param {
    precision: MINIFLOAT
    mant_bits: 3
    exp_bits: 4
  }
}
```

(b) Ristretto FC Layer

Figure A.2: Mini Floating-Point Layers

The Mini Floating-Point quantization is applied in the same way to parameters, input and output features of all the layers. For this trimming mode, the additional parameters are:

- *Precision*, which is the current representation format;
- *Mantissa*, the number of bits for the mantissa;
- *Exponent*, the number of bits for the exponent.

Dynamic Fixed-Point

For Dynamic Fixed-Point format, quantization is performed separately for Conv and FC layers. Moreover, the values of each layer are divided in three groups: input, output and parameter.
The quantization parameters are:

- *Precision*, which is the current representation format;
- *Bit Width*, the number of total bits. There are three different entries of this type: one for the inputs, one for the outputs and one for the parameters;

- *Fractional Width*, the number of bits for the fractional part. As for the total bit width, there are three different entries of this type.

```
layer {
  name: "conv1"
  type: "ConvolutionRistretto"
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 64
    bias_term: true
    pad: 3
    kernel_size: 7
    stride: 2
  }
  quantization_param {
    bw_layer_in: 8
    bw_layer_out: 8
    bw_params: 8
    fl_layer_in: 0
    fl_layer_out: 9
    fl_params: 18
  }
}
```

(a) Ristretto Conv Layer

```
layer {
  name: "fc128"
  type: "FcRistretto"
  bottom: "global_pool"
  top: "fc128"
  inner_product_param {
    num_output: 128
  }
  quantization_param {
    bw_layer_in: 8
    bw_layer_out: 8
    bw_params: 4
    fl_layer_in: 9
    fl_layer_out: 10
    fl_params: 3
  }
}
```

(b) Ristretto FC Layer

Figure A.3: Dynamic Fixed-Point Layers

Power of 2 Weights

Power of 2 Weights quantization format adopts different quantized representations for parameters and features: weights are approximated by the closer power of 2 while input and output features are quantized considering Dynamic Fixed-Point. Therefore, the additional parameters are:

- *Precision*, which is the current representation format;
- *Bit Width*, the number of total bits. There are two different entries: one for the inputs and one for the outputs;
- *Fractional Width*, the number of bits for the fractional part. Again, there are two entries of this type;
- *Minimum Exponent*, minimum selected exponent for the power of 2 parameters;

- *Maximum Exponent*, maximum selected exponent for the power of 2 parameters;

```
layer {
  name: "conv1"
  type: "ConvolutionRistretto"
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 64
    bias_term: true
    pad: 3
    kernel_size: 7
    stride: 2
  }
  quantization_param {
    precision: INTEGER_POWER_OF_2_WEIGHTS
    bw_layer_in: 8
    bw_layer_out: 8
    fl_layer_in: 0
    fl_layer_out: 9
    exp_min: -20
    exp_max: -13
  }
}
```

(a) Ristretto Conv Layer

```
layer {
  name: "fc128"
  type: "FcRistretto"
  bottom: "global_pool"
  top: "fc128"
  inner_product_param {
    num_output: 128
  }
  quantization_param {
    precision: INTEGER_POWER_OF_2_WEIGHTS
    bw_layer_in: 8
    bw_layer_out: 8
    fl_layer_in: 9
    fl_layer_out: 10
    exp_min: -9
    exp_max: -2
  }
}
```

(b) Ristretto FC Layer

Figure A.4: Multiplier-Free Arithmetic Layers

A.2 Fine-Tuning

After the quantization, Ristretto suggests to fine-tune the network in order to have a better accuracy. Starting from the quantization formats identified by the previous quantization step, the training command of Caffe has been modified in order to accept also the Ristretto quantized layers. If this type of layers is part of the considered network, the training procedure is carried out taking into account quantized weights. However, this training does not take into account quantized activations. Due to this unconsidered error, the framework keeps under control its bit reduction during the training, to avoid too large drop off.

This feature of Ristretto corresponds to the aware-training quantization described in section 3.1.

Bibliography

[1] W. W. Bledsoe. *The Model Method in Facial Recognition.* Tech. rep. Panoramic Research, Inc., 1964.

[2] W. W. Bledsoe and H. Chan. *A Man-Machine Facial Recognition System-Some Preliminary Results.* Tech. rep. Panoramic Research, Inc., 1965.

[3] *An image of Woody Bledsoe from a 1965 study.* URL: `https://media.wired.com/photos/5e0fd5ef190eac0008951e39/master/w_1600%5C%2Cc_limit/WI020120_FF_WoodyBledsoe_02.jpg`.

[4] M. Turk and A. Pentland. "Eigenfaces for Recognition". In: *Journal of Cognitive Neuroscience* 3.1 (1991).

[5] T. Ahonen, A. Hadid, and M. Pietikainen. "Face Description with Local Binary Patterns: Application to Face Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.12 (2006).

[6] L. Wiskott et al. "Face recognition by elastic bunch graph matching". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19.7 (1997).

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems* 25 (2012).

[8] Oludare Abiodun et al. "State-of-the-art in artificial neural network applications: A survey". In: *Heliyon* 4 (2018).

[9] Al-Masri A. *How Does Back-Propagation in Artifical Neural Networks Work?* URL: `https://towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7`.

[10] *Undestanding The Mathematics Behind Gradient Descent.* URL: https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e.

[11] Wikipedia. *Stochastic Gradient Descent.* URL: https://en.wikipedia.org/wiki/Stochastic_gradient_descent.

[12] *The Basic Layers of CNN.* URL: https://miro.medium.com/max/1000/1*irQx0wao-u9jOfLIM9C-rA.png.

[13] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of The 32nd International Conference on Machine Learning*. 2015.

[14] K. He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[15] G. Huang et al. "Densely Connected Convolutional Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[16] Y. Cheng et al. "Model Compression and Acceleration for Deep Neural Networks: The Principles, Progress, and Challenges". In: *IEEE Signal Processing Magazine* 35.1 (2018).

[17] Cristian Bucilǔ, Rich Caruana, and Alexandru Niculescu-Mizil. "Model Compression". In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2006.

[18] L.J. Ba and R. Caruana. "Do deep nets really need to be deep?" In: *Advances in Neural Information Processing Systems* 3 (2014).

[19] Adriana Romero et al. "FitNets: Hints for Thin Deep Nets". In: *3rd International Conference on Learning Representations, ICLR 2015, Conference Track Proceedings*. 2015.

[20] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. "Speeding up Convolutional Neural Networks with Low Rank Expansions". In: *BMVC 2014 - Proceedings of the British Machine Vision Conference*. 2014.

[21] S. Han et al. “Learning both weights and connections for efficient neural networks”. In: *Proc. 28th Int. Conf. Neural Information Processing Systems*. 2015.

[22] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998).

[23] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, Conference Track Proceedings*. 2015.

[24] S. Han, H. Mao, and W. Dally. “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding”. In: *Proc. Int. Conf. Learning Representations*. 2016.

[25] R. Krishnamoorthi. “Quantizing deep convolutional networks for efficient inference: A whitepaper”. In: *ArXiv* (2018).

[26] “IEEE Standard for Floating-Point Arithmetic - Redline”. In: *IEEE Std 754-2019 (Revision of IEEE 754-2008) - Redline* (2019).

[27] Jiali Ma et al. “Layer-by-layer Quantization Method for Neural Network Parameters”. In: *ICNSER2019: Proceedings of the International Conference on Industrial Control Network and System Engineering Research*. 2019.

[28] Soheil Hashemi et al. “Understanding the impact of precision quantization on the accuracy and energy of neural networks”. In: 2017.

[29] Matthieu Courbariaux, Y. Bengio, and Jean-Pierre David. “BinaryConnect: Training Deep Neural Networks with binary weights during propagations”. In: *NIPS* 28 (2015).

[30] Matthieu Courbariaux et al. “Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1”. In: 2016.

[31] *Tensorflow*. URL: `https://www.tensorflow.org/overview?hl=en`.

[32] Yangqing Jia et al. “Caffe: Convolutional Architecture for Fast Feature Embedding”. In: (2014).

[33] Berkeley University of California. *Caffe*. URL: https://caffe.berkeleyvision.org.

[34] *Keras*. URL: https://keras.io.

[35] *Tensorflow Lite - Model Optimization*. URL: https://www.tensorflow.org/lite/performance/model_optimization?hl=en.

[36] Philipp Gysel et al. "Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* (2018).

[37] F. Guzzi et al. "Distillation of a CNN for a high accuracy mobile face recognition system". In: *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2019.

[38] F. Guzzi et al. "Distillation of an End-to-End Oracle for Face Verification and Recognition Sensors †". In: *Sensors* 20.5 (2020).

[39] Dong Yi et al. "Learning Face Representation from Scratch". In: *CoRR* (2014).

[40] Wikipedia. *K-Means Clustering*. URL: https://en.wikipedia.org/wiki/K-means_clustering.

[41] Scikit Learn. *K-Means*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

[42] B. Jacob et al. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018.

[43] J. Duan et al. "The Speed Improvement by Merging Batch Normalization into Previously Linear Layer in CNN". In: *2018 International Conference on Audio, Language and Image Processing (ICALIP)*. 2018.

[44] Q. Zhang et al. "FPGA Implementation of Quantized Convolutional Neural Networks". In: *2019 IEEE 19th International Conference on Communication Technology (ICCT)*. 2019.

[45] R. Li et al. “Fully Quantized Network for Object Detection”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[46] Gary B. Huang et al. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. Tech. rep. University of Massachusetts, Amherst, 2007.

[47] Gary B. Huang and Erik Learned-Miller. *Labeled Faces in the Wild: Updates and New Reporting Procedures*. Tech. rep. University of Massachusetts, Amherst, 2014.

[48] Amherst University of Massachusetts. *Labeled Faces in the Wild*. URL: `http://vis-www.cs.umass.edu/lfw/`.