Corso di Dottorato di ricerca

in Informatica

ciclo 33

Tesi di Ricerca

Convolutional Neural Networks Compression for Embedded Industrial Applications

SSD: INF/01

Coordinatore del Dottorato
ch. prof. Agostino Cortesi

Supervisore
prof. Andrea Albarelli

Supervisore cotutela
PhD Filippo Bergamasco

Dottorando
Dalila Ressi
Matricola 839745

Università Ca' Foscari Venezia

Dottorato di Ricerca in Informatica, 33° Ciclo

(A.A. 2017/2018 – 2020/2021)

Convolutional Neural Networks Compression for Embedded Industrial Applications

Settore scientifico disciplinare di afferenza: INF/01

Tesi di dottorato di Dalila Ressi
Matr. 839745

Tutori del dottorando
Prof. Andrea Albarelli

Coordinatore del dottorato
Prof. Agostino Cortesi

December, 2021

Author's e-mail: dalila.ressi@unive.it

Author's address:

Dipartimento di Scienze Ambientali, Informatica e Statistica
Università Ca' Foscari Venezia
Via Torino, 155
30172 Venezia Mestre – Italia
tel. +39 041 2348465
fax. +39 041 2348419
web: `http://www.dais.unive.it`

To my family.

- Dalila

Abstract

Convolutional Neural Networks have proved to be a powerful tool to solve a wide range of Computer Vision tasks, especially where is difficult to implement a solution in a purely algorithmic way. In the industry, the availability of powerful deep models to address classification, detection, and image segmentation now offers new possibilities for automating not only the production, but also the quality assessment of the final products. Unfortunately, industrial applications have to face some limitations, especially when dealing with the so called ”Embedded Vision” solutions where such models have to be transferred and used directly on-camera. Indeed, limited memory and computational capability pose important challenges on the architecture of the model to use. During the last years a large amount of research aimed to face such problems, proposing various compression algorithms to reduce the number of parameters of neural networks. The purpose of this thesis is to explore existing literature and to provide some general guidelines that can be beneficial during the deployment of industrial applications. Among all possible problems and challenges industrial settings present, two very common issues concern the complexity of the models and the scarcity of data. This work addresses some of the most exploited techniques to tackle such problems, as well as contributes by proposing two novel methods: a novel data augmentation technique to compensate heavily unbalanced classes, and a filter pruning algorithm that greatly improves the inference time and reduces the memory footprint of a model. The algorithms have been implemented in real-case scenarios, and they have been compared to other methods in the existing literature.

Contents

List of Figures

List of Tables

1 Introduction

Over the past few years industrial applications have been extensively exploiting the advantages coming from Deep Learning, in particular when using Convolutional Neural Networks (CNNs). The intrinsic flexibility of these networks makes them widely adopted in a variety of practical applications, from medical to industrial. In particular, image and signal processing tasks are well-suited for the convolutional architecture, therefore a huge number of Computer Vision solutions have been proposed in the literature. Deploying a large and accurate model to perform a certain task takes considerable energy and space. Even if this might not be a problem during the training phase, it becomes a big issue at inference time, especially if the model has to run on devices with reduced computational resources or small storage space. With the rising of CNNs popularity and the number of their possible applications, the scientific community focused on how to adapt these models to make them accessible in an easier way. As a consequence an enormous number of compression techniques has been proposed, aiming to reduce the memory consumption and computational complexity without compromising the original performance. Given the complexity of Convolutional Neural Networks, or of Neural Networks in general, there is an unlimited amount of possible solutions to reduce the model footprint. This thesis aims to provide a gentle introduction to existing compressing algorithms as well as to contribute with some novel techniques to improve industrial development of applications exploiting deep learning.

1.1 Neural Networks Background

When talking about *Artificial Intelligence (AI)* it is impossible to not cross with the word *Deep Learning (DL)*. Deep Learning takes advantage of *Artificial Neural Networks (ANNs)*, often simply referred as *Neural Networks (NN)*, to tackle *Machine Learning (ML)* problems in a more efficient way. It allows for *learning* a meaningful representation of the raw data without the high-level expertise or engineering skills needed to hand-craft feature extractors. The term *deep* instead is linked to the structure of the models trained on ML tasks, as deep learning usually exploits networks with a large number of layers. Artificial intelligence applications like image and speech recognition, natural language processing, object detection, autonomous driving, medical diagnosis and game playing

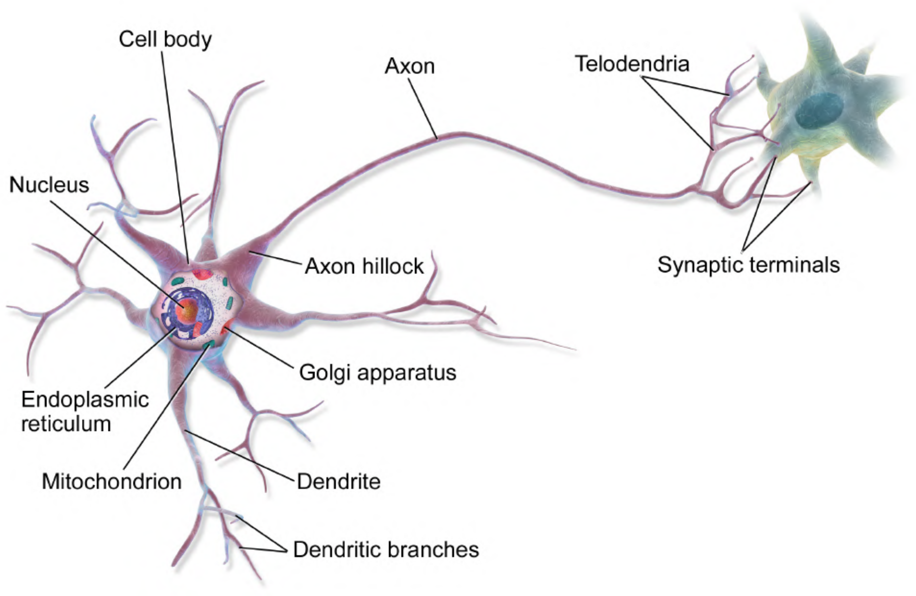


Figure 1.1: Model of biological neuron.

revealed impressive breakthroughs when using deep learning [33]. This technology covers a wide range of modern life aspects. Tasks we perform daily, like buying a product online, making a search on the web, using social medias, sending a message or taking a picture, are often involved with some kind of artificial intelligence algorithm. For example, instead of inserting a code to protect your privacy, you can unlock your phone by using your fingertips or showing your face, and while typing in a textbox most applications use auto-completion or smart reply.

1.2 Supervised Learning

One of the most common machine learning problems consists of *Supervised Learning (SL)* [105]. The goal is to learn a classifier (or a regressor) given a dataset of labelled data. The classifier can be seen as a function mapping some data given as input to one of the classes in a finite set. Deep learning consists on training a Neural Network to fit such function. Even if the observations need to be labelled to know which class they belong to, no prior information on how the classes are divided need to be fed to the network in order to train a good classifier. The dataset has to meet only one requirement: to provide enough samples to cover all possible cases.

Neural networks are structured in *layers* made by non linear *units* which extract information from the input data. They are inspired by the shape of human brain's neurons and they aim to mimic the brain functioning. A neuron, showed in figure 1.1, can be simplified as a structure with many incoming connections (*dendrites*), a main body (*soma*), and an output channel (*axon*). Even if the biological neuron can be more complex, neural networks are made of neurons (often referred also as *computational units* or *nodes*), which adopt this simplified structure.

The output of a single unit i of a neural network is usually defined as :

$$y_i = \phi\,(b_i + \sum_{j=0}^{m} w_{ij}x_j) \tag{1.1}$$

where the unit takes m input values x and a bias b, and after some computations returns an output y.

Each element of the input vector has a certain importance, defined by its corresponding *weight* w. The *bias* b can be seen just as a constant value added to a linear function. It increases the network's flexibility by allowing the layers to add a shift. Biological neurons send an output signal only if they receive a big stimulus, usually higher than a certain threshold. In this case the neuron "fires" and it sends a signal to the other neurons it is connected to. In a similar way a unit performs a weighted sum of the inputs followed by an non-linear activation function $\phi(\cdot)$, which returns an output dependent on the magnitude of the input received.

Some of the most common activation functions are:

$$\begin{aligned}
Sigmoid(z) &= \frac{1}{1+e^{-z}} \\
tanh(z) &= \frac{2}{1+e^{-2z}} - 1 \\
ReLU(z) &= max(0,z) \\
LeakyReLU(z) &= max(0.1z,z) \\
ELU(z) &= \begin{cases} z, & z \geq 0 \\ \alpha(e^z-1), & z < 0 \end{cases}
\end{aligned} \tag{1.2}$$

The most popular and widely used function is the *Rectified Linear Unit (ReLU)*. It is not only simple, but also most of the networks using this activation converge faster [147].

Neurons or units are arranged in groups, or layers, and they are connected only to neurons in a different layer. In neural networks, layers are sequentially arranged and their units can be connected only to the neurons in the previous or the following layer. If all the neurons in a certain layer have a connection (often just referred as weight) to all the units in the next layer, then such layer is called *fully connected (fc)*.

The number of layers used defines the *depth* of the network and deeper networks learn higher level representations over the data. The depth of a network, indeed, is highly correlated to its complexity, as the non linearity of the units allows to fit a more intricate function around the data. For difficult tasks, deeper networks usually achieve the best performance and it is for this reason that deep learning has this name.

1.3 Training

In the context of supervised learning, the final layer or *output layer* usually returns a vector of scores as long as the number of classes, which represents the probability of the input to belong to each category. The learning process consists of tuning the importance the units give to their input, by changing their weights (and the bias of their corresponding layer). It is possible to configure the network such that the output scores for every possible input data belonging to the dataset converge to the desired output (the labels). This is referred as the *training process* and it is usually achieved by using *error backpropagation*.

Training usually consists of two phases: the *forward* pass and the *backward* pass. In the forward pass one observation or a small set of data are fed to the network. The input flows through the layers, where each neuron processes it and then passes it to the units in the next layer, until the output vector is computed. Since the optimal values of weights and biases are initially unknown, they are usually randomly initialized which results in the network to return an output very far from the desired one. According to the activation function used by the network there are different distributions that can be chosen to initialize the weights [48, 65], which is a very important step to make sure that the network eventually converges toward a good solution. For example, starting with weights too small or too large can result in the network in not learning or to diverging [89].

After the forward pass we need to adjust the weights and biases of the network such that the computed output gets closer to the actual labels. This update is performed from the output layer towards the input layer, and for this reason it is called the backward pass. To adjust the network parameters toward an optimal configuration we need the *objective function*, or *error loss*:

$$L = \frac{1}{2}||Y_{output} - Y_{target}||_2^2 \tag{1.3}$$

representing how far the obtained scores (Y_{output}) are from the desired ones (Y_{target}). The loss can be seen as a high-dimensional space made of hills, saddle points and local minima. To minimize the error, we can exploit the information coming from the gradient. Indeed, the opposite direction of the gradient vector indicates the path toward which a small change in the weights corresponds to a descent in the landscape, getting closer to a local minima.

To better understand how the weights are updated, we can imagine the objective function similar to a mountain-like landscape. Since the weights are randomly initialized and the shape of the objective function is unknown, we can think about being blind-folded in a random position of this multi-dimensional place made of hills and valleys. The goal is to reach one of the valleys, which correspond to a weight configuration minimizing the error. The only tool we have available is taking a small step at a time, and trying to understand the slope of the ground under our feet. Even if we cannot see around us, the gradient can tell us in which direction the floor starts to get higher. By taking a step in the opposite direction we hopefully get further from the peak and walk toward a valley, or local minima.

There are different ways to take a step, or different *optimizers*. The most used optimizers are:

- *Gradient Descent(GD)*
- *Stocastic Gradient Descent(SGD)*
- *Mini-Batch GD*
- *RMSProp [69]*
- *Momentum [172]*
- *Nesterov Accelerated Gradient (NAG) [149]*
- *Adaptive Gradient (ADAGRAD) [37]*
- *AdaDelta [216]*
- *Adaptive Moment Estimation (ADAM) [93]*

The goal of using optimizers is to reach the best configuration in the minimum number of steps. They define after how many observations to take a step (only one for GD, all the samples possible for SGD, or something in the middle using a batch-based approach), but they also specify how much to move, which correspond to the *learning rate* (*lr*). Optimizers also have to take into consideration problematic situations like being in the middle of a saddle point (in this case the gradient would make the network configuration oscillate indefinitely between the two highest directions) [162].

Once we decided which direction to go toward, we have to adjust the network parameters accordingly. Weights are then updated in the backward pass using the backpropagation algorithm. Backpropagation computes the gradients of the objective function with respect to all the weights in all the layers. It can be seen as the practical application of the chain rule as the gradients are computed (and updated) backwards starting from the output layer toward the input layer in a single pass.

An *epoch* is what takes the network to see all the data in the training set during the training process. Several epochs are usually required for the network to converge to an optimal configuration, and the loss is usually one of the good indicators to understand if the training process is performed correctly. The training set can be divided in *batches*, where a batch is an integer number larger than zero and equal or less than the number of samples in the training set. The batch size defines when to update the network weights, so if the batch size is equal to the number of observation in the training set we talk about *Batch Gradient Descent*, if the batch size is equal to one than we have the classic SGD, while all the other values, usually a power of two, are referred as mini-batch Gradient Descent. The samples are usually randomly sorted into batches in a different way for every epoch. The size of the batch is critical for the training time, as using larger batches allows the network to converge faster, but if the batches are too big the algorithm will struggle to converge close to the optimal minima.

Layers in the middle of a Neural Network are called *hidden layers*, and the number of such layers, as well as how many units per layer to use, correspond to the *depth* and the *width* of the network. They are two of the *hyperparameters* to set before training a network. Hyperparameters are very important because if the network is too deep the function modelled can be too complex for the given dataset and the network might lose the generalization capability. As opposed to the mentioned problem, called *overfitting*, also a too shallow network might not be able to extract a meaningful enough representation of the data, leading to *underfitting*. Neural networks with more than one hidden layer are known as *Deep Neural Networks (DNNs)*.

Other than the hyperparameters defining the architecture of the network, before starting to train a network it is crucial to decide also how the training will be performed. Weight initialization, learning rate, number of training epochs, batch size are only some of the hyperparameters that have to be initially set, and they can greatly affect the training time and avoid the network to be stuck on a not good enough local minima. One of the best approaches to train a model able to achieve high accuracy is to make sure the network is complex enough. This usually is assured by using a small balanced portion of the dataset as *validation set*. After every epochs the loss or the accuracy on the validation set is computed and confronted with the one of the *training set*. If the architecture is complex, or deep enough, then the loss on both the training and the validation sets will decrease very fast, but after few epochs the validation loss will not improve anymore, or it will even increase. This is a clear effect of overfitting. At this point, decreasing the depth of the network or even trying a different architecture with less parameters is a laborious and slow process.

Using a very deep model to solve a simple task often results in overfitting. To mitigate such problems we can take advantage of some mechanisms classified as *regularizers*. The most intuitive solution for the case just described is to just stop the training as soon as the validation loss differs from the one computed on the training set. This regularization method is called *early stopping*. Sometimes the data available are too scarce to afford the luxury of dedicating part the dataset for the validation set, and we need other techniques to fight overfitting. When talking about regularization the most common used method consists in adding a penalty, or *regularization term* to the objective function:

$$Objective function = Loss + \lambda\ Regularization\ term \tag{1.4}$$

The most used regularizations are *L1* and *L2*, also called as *Lasso Regression* and *Ridge Regression*:

$$\begin{aligned} L1 &= \sum ||w|| = \sum_j |w_j| \\ L2 &= \sum ||w||^2 = \sum_j w_j^2 \end{aligned} \tag{1.5}$$

The regularization term poses a constraint on the magnitude of the weights, forcing the smallest values toward zero. L2 is also called *weight decay*, as it forces the values toward zero (close to, but not zero). On the other side L1 can reduce the weights exactly to zero. In this way the least important features are ignored, and as a lot of matrices are close to zero the relative neurons are technically not used. This is equivalent to use a smaller, simpler network than the original one.

A different technique consists in inserting one or more *dropout* layers, usually toward the end of the network. These layers work only at training time, and they randomly filter out some of the connections with a fixed probability p. This forces the network to exclude some of the nodes, which can be seen as training a network smaller than the original one. This subnetwork is simpler and it has less parameters therefore is able to generalize better. Neurons which weights should not be updated are changed at every epoch, and this forces the network to not heavily rely always on the same nodes. The approach has some drawbacks: the probabilities of the dropout layers, as well as their number and position are all hyperparameters that need to be tuned and it might require training the same network multiple times to find a good configuration.

One of the best, if not mandatory, practices when training a network is to exploit *data augmentation*. This powerful tool allows to greatly increment the size of the dataset, not only reducing overfitting, but also increasing the accuracy of the network. The idea is simple: it is possible to add new observations to the dataset by simply performing some kind of transformations on the available data. In the case of convolutional neural networks, the most common transformations consist in shifting, scaling, rotating and flipping the data (usually an image), but there are also a lot of other possible operations, e.g. shearing and adding noise. Even if the pooling operation, as well as the convolution layers, provides a certain degree of invariance to the position of the features, they are usually not able to model such transformations. In this way every observation added to the original dataset can be considered by the network as a completely new instance that it has never observed, greatly increasing the performance at inference time.

1.4 Convolutional Neural Networks

The most simple deep learning networks are *feedforward Neural Networks*, where a fixed-sized input flows through the network and produces a fixed-sized output without any loops. A particular case of Neural Networks are *Convolutional Neural Networks (CNNs)*, specialized networks that take advantage of *convolutions*. Classical neural networks are usually made only of fully connected layers and process one-dimensional data. They are often simply referred as *Multilayer Perceptrons (MLPs)*, even if the name is technically incorrect as the *perceptron* introduced by Rosenblatt in 1958 [170] was purely linear [90]. Typical inputs for CNNs, instead, are 2D signals (e.g. images), but they can also be used to process 1D or 3D signals (e.g. audios and videos). Some examples of 1D CNNs are presented in section3.2. Convolutional layers use *filters*, or *kernels*, to extract meaningful features from the input and produce a feature map for each filter. In the 2D case, the filter is shifted over the image and every shift corresponds to a discrete convolution, which computes a single value of the *feature map* computed as output. The input image, as well as the feature maps produced by the convolutional layer can be seen as multidimensional arrays, and they are often called *tensors*. The weights to learn are the values of the filters. *Weight sharing* is one of the key concept of CNNs: the idea is that if a meaningful feature is detected within an image, it should not matter in which position the feature is, so the weights of a filter should not change as it computes the convolution over the whole input.

Convolutional layers are often followed by a *pooling layer*, which performs a subsampling operation, reducing the dimensionality and allowing the network to be invariant to small shifts and distortions. Last layers are usually fully connected layers, like in traditional NNs, where all the units in a layer are fully connected to the units in the next layer by single weights. The number of output nodes is usually the same as the number of classes if the network is trained to solve a classification task. In this case the activation function of the last layer returns a vector of probabilities, and the specific input image is assigned to the class or label with the highest probability. There are also other possible outputs, for example in object detection problems the network returns two coordinates, which represent the opposite corners of the bounding box containing detected object.

Convolutional Neural Networks are of undeniable importance when working with images, to the point that they have become *de facto* standards for various computer vision and machine learning operations. Since neural networks consist mostly of fully connected layers, the number of weights involved while using images would be unfeasible high. For example, connecting all values (pixels) of a small RGB image belonging to CIFAR-10, which is $32 \times 32 \times 3$, to a single fully connected neuron, would mean to learn $32 \times 32 \times 3 = 3072$ weights. Considering that we need multiple neurons and multiple layers to build a network, the number of parameters would grow exponentially. On the other side, when we use CNNs we have more hyperparameters involved.

While the only parameter to set for a certain layer in a NN is the number of units per layer, when working with CNNs it is of utmost importance not only the number of filters, but also their size and how to shift them on the image, which is defined by the *stride* and *padding* parameters.

The stride parameter defines how many pixels or values separate one convolution from the next one. So, having a stride $S = 1$ means the output of the convolution will have the same dimension of the input, while a stride $S > 1$ will shrink the output. A significant problem is how to perform the convolution operation on the borders, as the filter has to be placed inside the input. For this reason a padding is usually used to either add a border of zero values (*zero padding*) either adds the same values on the borders, but flipped (*same padding*). Even if it is possible to reduce the output dimension of a convolutional layer simply by setting a stride higher than 1, usually it is preferable to to use a *pooling layer*. The pooling layer simply performs a downsampling operation. It has its own window, as well as padding and stride, and there are different types of pooling (average or max are the most exploited).

The typical structure of a Convolutional Neural Network consists of the *input layer* followed by a sequence of convolutional layers, usually interleaved by pooling layers (sometimes a convolutional layer and the pooling layer after it are referred together as a *convolutional block*). The last layers of a CNNs usually are fully connected layers, which are connected to the *output layer*. In supervised learning the output layer is a probability vector that assigns to the input a certain probability to belong to each of the possible classes. The class with the highest probability is usually declared as the class the input belongs to. Sometimes to better assess the accuracy of the network, not only the class with the highest probability (*TOP-1 accuracy*) but also if the true class is among the five classes with the highest probability (*TOP-5 accuracy*) is taken into account. A scheme of

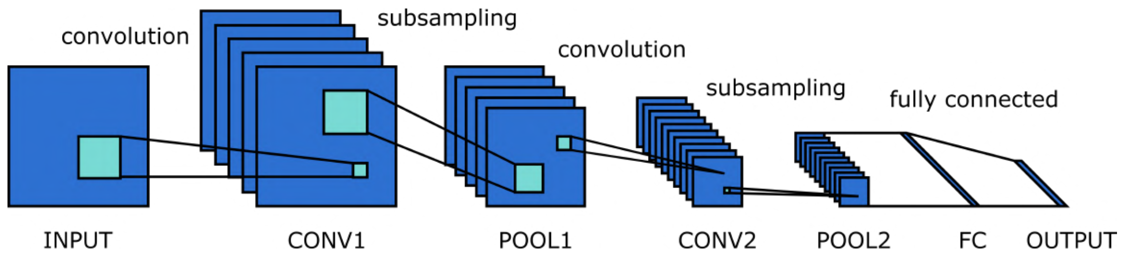


Figure 1.2: Example of a classic CNN architecture. There are two convolutional blocks (convolution + ReLU + pooling). Every operation has its own window size, stride and padding. The last two layers are fully connected layers (the last one is also the output layer).

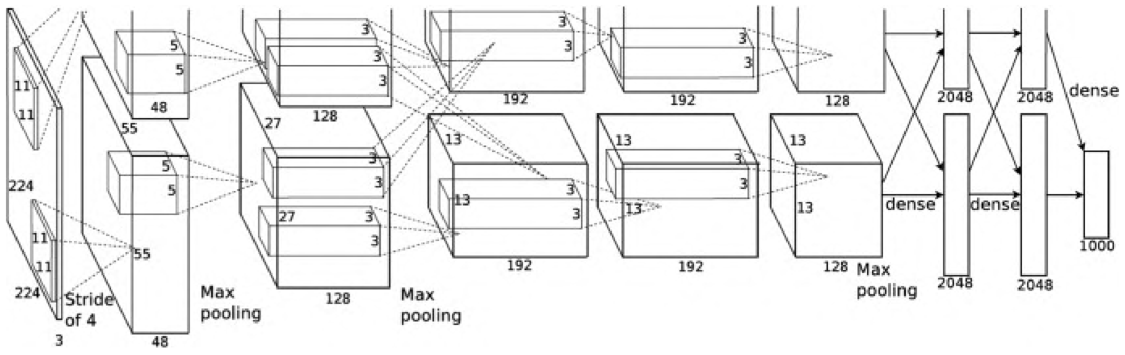


Figure 1.3: AlexNet architecture as shown by the authors.

a classical CNN architecture can be seen in figure 1.2.

1.5 Evolution of CNNs

In 2012 Alex Krizhevsky et al. won the famous ImageNet Large Scale Visual Recognition Challenge (ILSVRC), proposing his famous architecture, later known as *AlexNet* [102]. In figure 1.3 we can see the AlexNet architecture. It is made of five convolutional blocks (some of them containing max pool layers), followed by three fully connected layers. It reshapes the ImageNet 2012 data into 224x224 RGB images, returning the probability distribution over the available classes.

Given the extraordinary jump in the performance of this network with respect to other previously existing methods, CNNs started to become extremely popular and hundreds of other architectures and algorithms have been proposed since then.

Another popular network is *GoogLeNet* [183], which was presented only two years later. Its main advantages are the simplicity (9 identical and relative simple blocks), the parallelism of the network (each block is structured in 4 parallel pathway) and its efficiency in terms of both computation time and memory usage. The high efficiency comes at a small cost in term of model accuracy on ILSVRC 2012. In figure 1.5 we can see

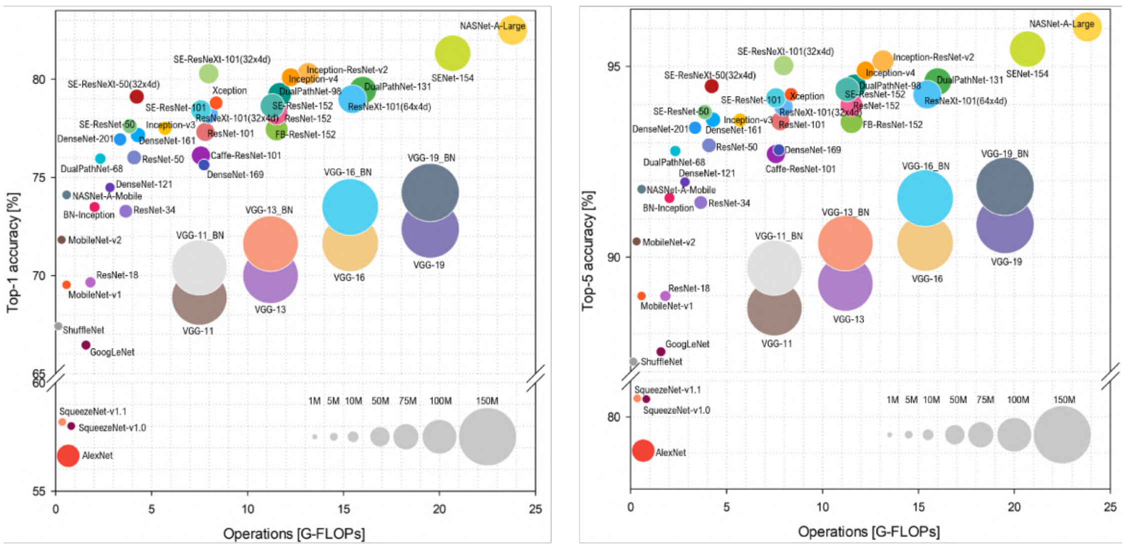


Figure 1.4: Image from [8]. The two ball charts represent the top-1 (left) and top-5 (right) accuracy on the ImageNet validation set with respect to the computational complexity of the considered architectures for a single forward pass. The size of the ball represents the model complexity. There is a direct correlation between the computational complexity (number of Giga-Flops) and the accuracy of the architectures.

some of the problematic images belonging to the ImageNet validation set being classified by GoogLeNet.

The scientific community started to realize that the depth of the architecture is of great importance when it comes to solve complex problems which require high accuracy (see figure 1.4). In particular, deep neural networks are good at learning multiple levels of feature representation. Basic features, such as edges and corners, are detected in the first layers and then combined to allow the network to extract more complex features, like curves or shapes. As the depth increases, so does the level of abstraction. This concept is known as representation learning [7].

As a consequence, the networks which became popular in the following years have more and more layers, such as *ResNet* [66] and *VGG* [178]. ResNet stands for *Residual Network*. It is a very deep network proposed by He et al. and it comes into more variants. The most famous ones are ResNet-50 (which counts 50 layers) and ResNet-101 (101 layers deep). The depth and the complexity of the networks represents its major drawback.

VGG network also comes into two different variants, VGG-16 and VGG-19. The former has 16 layers with learnable weights: 13 convolutional layers and 3 fully connected layers. VGG-19 has 19 layers with learnable weights: 16 convolutional layers and 3 fully connected layers. In both networks, all convolutional layers have filters of size 3-by-3. VGG networks are larger and typically slower than other pretrained networks (in particular with respect to GoogLeNet and ResNet).

Finally, another well known architecture is *Inception-v3* [185], which is an evolution of the GoogLeNet architecture. Compared to GoogLeNet, Inception-v3 is larger, deeper, typically slower, but more accurate on the original ILSVRC data set. Inception-v3 is 48 layers deep.

During the last years researchers keep proposing deeper and deeper architectures, with up to even billions of parameters [158, 217], as using networks with more layers achieves higher accuracy most of the time. One of the main drawbacks with these deep networks is the *vanishing gradient* problem and the generalization capability. In particular, the vanishing gradient problem might occur in large networks with certain activation functions, as going toward deeper layers the gradient of the loss function approaches zero, making hard for the network to learn. For example, the sigmoid function takes a large input space, but returns only values between 0 and 1. Even when the input values change significantly, the corresponding variation in the output space will be minimal, leading to a small derivative. The solutions range from using a different activation function (e.g. ReLU), to exploit batch normalization to normalize the input. Using residual connections can also help mitigate the vanishing gradient problem. Indeed, the family of Residual Neural Networks (ResNets) allow to train very deep neural networks by adding residual connections between the input of a residual block and its output (after the activation function is applied). In this way the initial information is also carried through the network, together with the output of each block.

To overcome these issues and to address various types of tasks, during the years the architectures proposed have grown in complexity. Contrary to feedforward neural networks, *Recurrent Neural Networks (RNNs)* allow some of the neurons to be connected to backward layers. They are specially designed to work on sequences, or in general on data where time takes high importance. They are extensively exploited for speech recognition and language processing with context information. In particular, the Long Short Term Memory (LSTM) and the Gated Recurrent Unit (GRU) variations are the most widely used, as they have improved memory capability with respect to vanilla RNNs.

Since AlexNet was introduced in 2012, the interest of the scientific community moved from classic machine learning algorithms to deep learning. The intrinsic flexibility of Neural Networks allowed researchers to propose an overwhelming number of different architectures, as well as new training techniques. As a result, even if Neural Networks can achieve outstanding performance, using deep learning to solve a specific problem requires to adopt an existing solution and adapt it in a heuristic way.

1.6 Dealing with scarcity of data

Convolutional Neural Networks (CNNs) are a very effective and versatile tool to address a wide range of Computer Vision classification problems.

There are numerous remarkably large datasets available for such purpose, like MNIST [106], CIFAR-10/CIFAR-100 [100, 101] ImageNet [173], MS-COCO [121], but also for Natural Language Processing [29, 128, 209] and Audio/Speech analysis [3, 108]. These datasets are very helpful in providing the huge amount of samples required to train a

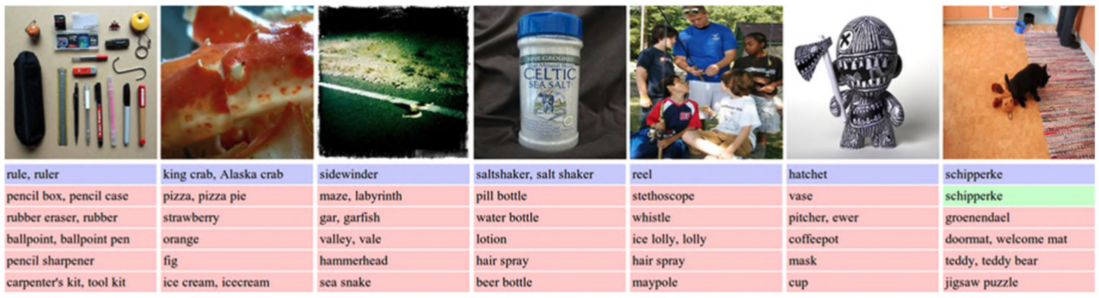


Figure 1.5: Interesting image taken from [173]. It shows representative validation images that highlight common sources of error. For each image, the ground truth is displayed in blue, followed by top 5 predictions from GoogLeNet (red wrong, green right).

CNN, but most of the times we want to train a CNN on our own data. Collecting enough observations is time-consuming and sometimes impossible. For this reason, specialized techniques to deal with sample scarcity have been proposed in literature and widely studied during the last years.

Two of the most important categories of approaches to tackle this problem are certainly *Data Augmentation* (already mentioned in section 1.3) and *Transfer Learning*.

Data Augmentation is about variance: while the exact number of samples is strictly tied to the particular case-of-study, it is central for the data to have a good diversity to allow the network to efficiently generalize the object of interest. In other words, the more diversity the network can *see*, the better the results. Data Augmentation tackles the problem by creating new samples out of the available ones. In particular, new data is generated through the application of several transformations to the original images. Scaling, translation, rotation, flipping, noise addition, perspective transform, color balance are some of the most widely used transformation when it comes to augmentation [207]. To this end, these techniques can be regarded more as a way of simulating new capturing conditions rather than new samples for the categories of interest.

One of the most popular data augmentation techniques is *PCA Color Augmentation*, which was used during the training of AlexNet [102]. They performed PCA on the set of RGB pixel values of the training set and modified the single images by adding multiples of the found principal components, where the magnitude is proportional to the relative eigenvalues. The magnitude is also multiplied by a random variable from a Gaussian distribution with mean 0 and standard deviation 0.1. This means that to each RGB pixel $I_{xy} = [I^R_{xy}, I^G_{xy}, I^B_{xy}]^T$ they added a quantity $[p_1, p_2, p_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$, where p_i and λ_i are ith eigenvector and eigenvalue of the 3×3 covariance matrix of RGB pixel values, respectively, and α_i is the random variable, which is drawn again for the next image. This technique significantly reduced the top-1 error training of AlexNet, as it allows the network to be invariant to light changes, especially to color and intensity of the scene.

Anyhow, generic data augmentation [190] is not the only successful data augmentation technique. Indeed, more advanced techniques have been proposed recently. In particular, Goodfellow et al. [49] show how to generate new samples after being trained

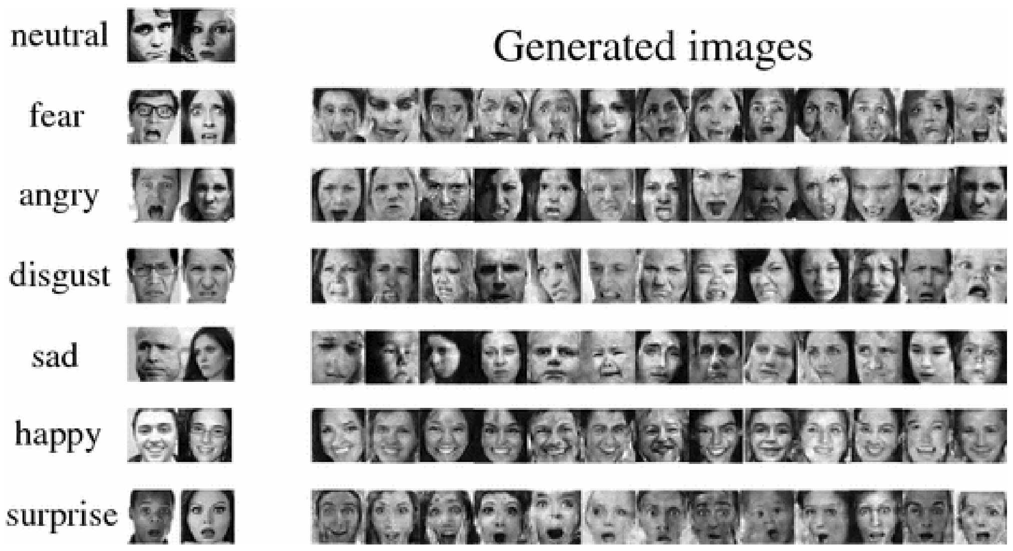


Figure 1.6: Some examples of synthetic data created with CycleGANs for emotion classification [175]. The new data are then integrated with existing images to augment the original dataset.

on samples drawn from some distribution in their seminal work *Generative Adversarial Networks (GANs)*. An example of images generated with CycleGAN [227] for emotion classification can be seen in figure 1.6. Rogez and Schmid [168] propose a scheme that artificially inflate the data set by using domain specific synthesization to produce more training data. A similar approach has been proposed by Peng et al. [157].

A very different approach to the same task is the one taken by Transfer Learning [211]. In this case the problem is solved by means of a machine learning method where a model developed for a task is reused as the starting point for a model on a second task, or, as defined in [152], *transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned.* From a more practical point of view, transfer learning involves the usage of pre-trained models (which are usually general enough to cope with different tasks) as the starting point, allowing to re-train just a part of the network to improve the performance on a specific setting.

The advantages are twofold. First, training a network from scratch is a time and resource consuming task, in particular if such training happens on a very large dataset. Secondly, training a network for a more general purpose task allows to use huge database freely available, like ImageNet [32] (which counts millions of labelled images belonging to a thousand different categories).

Transfer learning comes in two different flavours. The first takes the name of *Develop Model Approach*. This technique uses a related predictive modelling problem (for which a lot of data is available) to train the network. Next, the model is reused as the starting point for a model of our task of interest, usually after a fine tuning step. The second approach, which is the most common one, takes the name of *Pre-trained Model Approach*, and makes use of model trained on large and challenging datasets. The use of datasets with a great variety of objects allows to train networks which are capable to generalize different task. Again, these network are used as a starting point and re-trained to cope with the specific case-of-study that must be dealt with.

1.7 Convolutional Neural Networks for the Industry

During the last years the machine learning algorithms have been widely adopted by the manufacturing industry to improve and optimize the production processes. In particular, the Industry 4.0 paradigm focuses on exploiting smart sensors and machines to collect more data in the production phase to further improve the performance of the current protocols. Indeed, machine learning techniques are exploited to find patterns in complex production setups and optimize the overall process, by using continuous inspection, predictive maintenance, quality improvement, process optimisation, supply chain management, and task scheduling [163].

One of the main problems when adopting such intelligent machine learning algorithms in the industry is that these methods have to be transferred to specific applications. The extreme flexibility of deep learning algorithms and the large amount of already trained architectures available in the literature allows us to adopt machine learning to solve a vast range of problems in the industrial field. However, most of the existing solutions are either aimed to solve a specific task, either they are theoretical methods that have to be adapted to actual industrial application systems. For these reasons, when dealing with real-world scenarios it is never trivial to implement and insert deep learning techniques in the pipeline of an already existing application.

Among all possible tasks that can be addressed using machine learning, and in particular deep learning techniques, during my PhD I focused mainly on supervised learning. When dealing with classification tasks, the best choice is to exploit Convolutional Neural Networks (CNNs). CNNs are powerful tools, as they greatly reduce the time otherwise spent on creating hand-crafted filters and easily outperform every other machine learning algorithms for classification tasks.

Having a solid dataset, free of ambiguous images or mislabelled observations is probably the most important ground rule. Data labelling is a very long an tedious process, since it can require to manually label thousands of images, and it is not rare to find some images that are very hard to assign to a certain class with respect to another. These observations are what could damage the most the creation of a dataset, and have to be handled carefully. It is crucial, yet often not trivial, to define specifically which features an image has to present in order to belong to a specific class. The more these features are different among the classes, the easier the classification task will be. The instances presenting at the same time features belonging to different classes can be treated in different manners. If the image presents features belonging to multiple classes in equal measure, then it has to be discarded regardless. If instead the image shows mostly characteristics typical of a certain class, it can be either added to that class either discarded.

Adding this ambiguous image can teach the network to have a certain level of tolerance during the classification, but if the number of uncertain images is large, it is more difficult for the classifier to draw a line between the classes. For these reason during the labelling phase I would recommend to discard such samples, as long as a large amount of images are available for all classes. In the post implementation phase, or in the actual real-world scenario, the classifier will probably find plenty of such observations, but the output of the network will naturally reflect how much such images belong to each class.

As most of the times the classes have different importance, and the misclassification errors have different weight, it is better to manage these instances in the post classification phase. An example could be the quality assessment of a certain product, if the object presents a defect or malformation it has to assigned to a specific class. Misclassifying a defective product as one good for sale has a different impact than doing the opposite. Having multiple customers to complain about a product because of its quality or its behaviour could lower the reliability of the company, and this is often worse than simply discarding a defectless product.

Another advantage in simply discarding ambiguous images involves that solving a simpler classification task is highly correlated with the complexity of the network required to address the task. Using a more shallow network implies to shorten the inference time, a feature often required when classification has to occur in real-time.

To fasten the labelling process, one of the most common techniques is to train a simple network on a subsample of the data, that have been manually assigned to each class carefully, and then to let such network classify the remaining unlabelled data (similarly to what happens when using Reinforcement Learning). This approach can be useful if the number of images to classify is very large, but when we adopted it we found that it is has a side effect to not underestimate. Even if the classifier will assign a lot images correctly, they still need to be manually checked and eventually be reassigned or discarded. The real problem is not to detect such images, rather than how the new labelled images can introduce a bias in the user manually reassigning them. The only explanation we could find of this phenomenon is that as a person has to manually inspect hundreds or thousands of images (often during the same day), if the number of ambiguous images is high, the person charged with the task often tends to ”accept” the suggestion of the network, instead of discarding such images.

Such inconvenience introduced a delay in the creation of the database, as the images had to inspected multiple times and even by different people. For this reason I would suggest to use this technique very carefully, at least for datasets where a lot of images need to be discarded.

As already mentioned, if we want to address a classification task, CNNs are usually the best choice. Unfortunately, having large and accurate networks that have been trained on millions of images can be rarely useful in an industrial application.

The main reasons limiting the usage of pretrained networks in industrial applications concern the type of classification task, the data format and the inference time. In particular, using a network trained to recognize handwritten digits, as in the MNIST dataset, will most likely perform poorly if it is used to distinguish images of animals. Another significant difference can be the size of the images used to train the network with respect

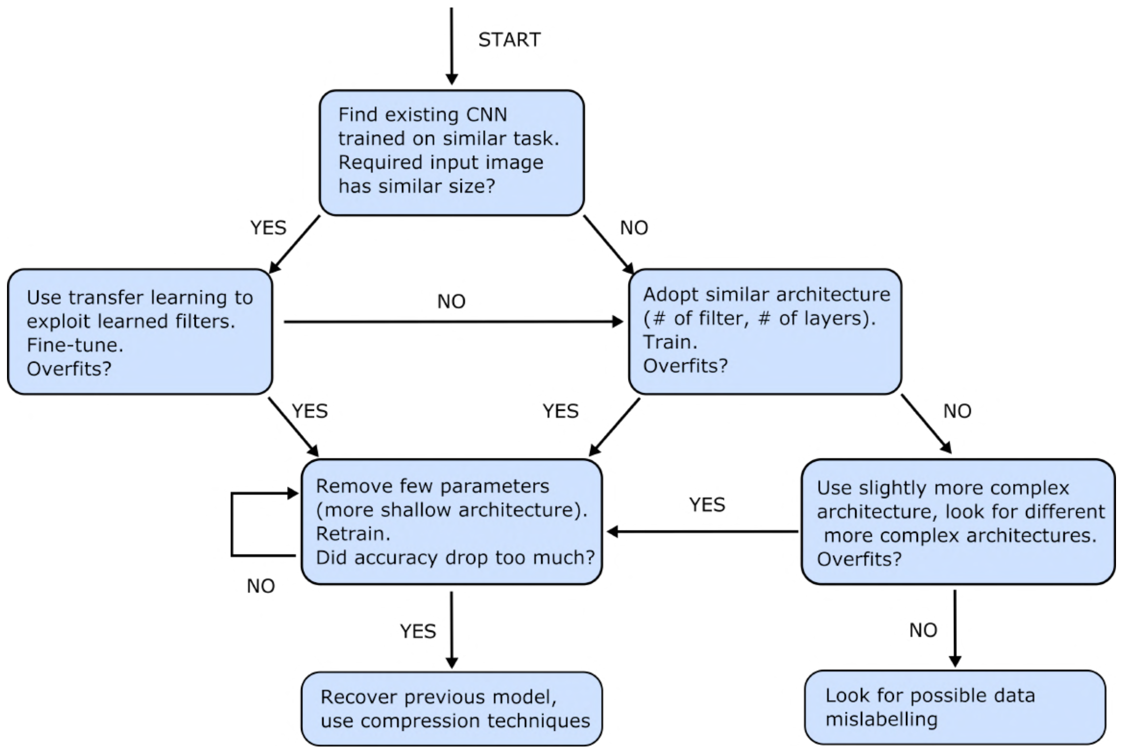

START
Find existing CNN trained on similar task. Required input image has similar size?
YES
NO
Use transfer learning to exploit learned filters. Fine-tune. Overfits?
NO
Adopt similar architecture (# of filter, # of layers). Train. Overfits?
YES
YES
NO
Remove few parameters (more shallow architecture). Retrain. Did accuracy drop too much?
YES
Use slightly more complex architecture, look for different more complex architectures. Overfits?
NO
YES
NO
Recover previous model, use compression techniques
Look for possible data mislabelling

Figure 1.7: Convolutional neural networks are largely exploited in the supervised learning field. Among multiple possible strategies, this chart shows one of the best ways to tackle this problem in an industrial application.

to the resolution of the images available in our specific industrial setup. Finally, what is probably the most important feature for an industrial application is the time required by the network to classify new images.

Anyway, pretrained network can be often adapted to solve a different classification task. This usually implies to use transfer learning, cropping the network and keeping the first layers, where numerous filters have been trained to detect a large range of features. This solution still presents numerous problems: as already mentioned, the size of the images of our dataset can be significantly smaller or bigger, and adapting such images to the size required as input by the pretrained neural network might create artifacts or harm the quality of the data. Another issue can be the number of channels (or depth) of the image: it is not rare that industrial cameras exploit extra channels, for example to store the near-infrared (NIR) capture of the object. This means to have images with two or four channels (two if the image is grayscale, four if if it is an RGB or HSV, plus the NIR frame). Most pretrained networks only allow the inputs to have one (grayscale) channel or three (RGB) channels, so the extra information coming from infrared has to be ignored.

One of the most problematic issues is the inference time: even if with transfer learning we can discard most of the pretrained network parameters and train a new network predicting the class of a single image with impressively shorter time, such network is often not the fastest one. Having so many nodes (or filters) in the first layers most likely implies that the network does not exploit all of them. This means that the network trained by using transfer learning, even if it usually achieves very high accuracy, is not the optimal architecture that can be used, as most like there exists a network with less parameters and shorter inference time that can reach the same performance. A short inference time is of crucial importance not only for real-time applications but also for embedded systems. The complexity of the network, the prediction speed, the battery and the memory used to store the model are all highly related, and optimizing the size of the networks is vital to exploit these technologies in small portable devices.

For all these reasons the adoption of state-of-the-art CNNs networks to solve supervised learning problems has to be tackled very carefully, and when the classification has be in real time or requires very short time for processing, it is more advantageous to train a new network from the start. Using an *ad hoc* solution allows to take into consideration all the variables and aspects of the setup, as the dataset format, the complexity of the task and the requirements on the accuracy and on the inference time.

Training your own network requires to draw a configuration of hyperparameters from a very large pool of possible values and methods. The choice of the architectures is probably the most important task and it usually requires multiple runs. Indeed, it is not only sufficient for the architecture to achieve high accuracy, as it also has to prevent overfitting. One of the most common strategies consists in looking for architectures used on a similar task and adopt one of them. If the network performs poorly it might be not complex enough to solve the specific task, and it probably requires to have more parameters. This can be easily achieved by adding more filters to the existing layers and/or changing their size, or even adding a whole new convolutional layer to the network to improve its capability to extract high level features. If rising complexity of the network in different ways did not result in improving the performance, this might hint to a poorly labelled dataset, where numerous images could have been assigned to the wrong class. A flowchart showing how to exploit CNNs to solve an industrial classification problem can be seen in figure 1.7.

Even if in the academic field a lot of time is spent in introducing a certain level of novelty, for example by proposing new modules or training algorithms, the industrial world is mostly application oriented, and problems have to be faced and solved in an heuristic fashion. Most of the time is occupied in adapting existing solutions, rather than creating new ones, especially when there are deliverables and deadlines. The need to present a functioning classifier in a limited amount of time is the main reason why looking for an existing similar model is preferable rather than blindly trying random architectures.

After finding a model that overfits on the dataset (which can be simply tested by running the CNN on the test set) it is possible to prune the architecture to make it smaller and faster. Actually it is not required for model to overfit, as long as it achieves high accuracy on the training set after the training phase. A good technique is to decrease the number of nodes and/or layers and to train such completely new architectures, in order to find the smallest possible model able to match the complexity of the classification problem that has to be solved.

The next step, after finding an architecture that optimizes the trade-off between the complexity of the network and the test accuracy, is to even further reduce this model by exploiting compression techniques. Chapter 2 presents a general overview of the most popular compression methods.

The training phase is also very important: as changing training parameters can affect the performance of an architecture, using a wrong initialization could mislead to think that a particular architecture is not suited to solve the specific classification task. A good approach is to start from the same parameters (e.g learning rate, optimization function, decay, etc.) used to train the network we initially adopted (if it was possible to find a network solving a similar task), and as we use a smaller architecture we need to adapt certain values, for example the learning rate, to match with the complexity of the network. The amount of possible hyperparameters and different techniques available sometimes makes it extremely hard to find the best configuration. Even if some comparative studies are available (for example it is known that ReLU activation and Adam optimizer are the best choices as they make the network to converge faster and easier to train), most of the time the experience of the developer and the complexity of the classification task are what matters the most.

1.8 Thesis Contents

The content of this thesis is divided in the following way: chapter 1 provides a gentle introduction to Neural Networks (section 1.1), in particular showing how Convolutional Neural Networks (section 1.4) are the best choice to address Supervised Learning (section 1.2) tasks in Industrial Applications (section 1.7).

Chapter 2 presents most popular compression methods and techniques. Compression covers a crucial role in the development of industrial applications, as it allows to reduce existing models such that they can be stored and run on embedded devices. Specifically, section 2.1 considers methods concerning the network's architecture, some examples are tensor decomposition, knowledge distillation and compact networks. In section 2.2 there are most popular pruning techniques, with a special consideration for filter pruning. Finally section 2.3 focuses on quantization algorithms to further compress the models.

In the second half of this thesis we present some practical problems coming from an industrial environment. We show how we applied compression and other techniques to such cases of study. Specifically, in chapter 3 we present different problems concerning two industrial applications, respectively in sections 3.1 and 3.2, while in chapters 4 and 5 we show how we addressed such tasks.

As our contribution, we propose a novel technique to mitigate the lack of data in one of the classes in a binary classification problem (4.2.1). The algorithm exploits a particular type of data augmentation and it has been published in [45]. We also examine a more complex problem: how to compress compact CNNs to perform camera tasks to deploy them on a FPGA support. The advantage coming from this approach is the capability to process images in real time on-board of the camera, extracting valuable information like interesting areas on which further computing can then be run. We show a possible implementation of these models, and we also propose a effective yet simple technique to compress them and transfer them on embedded systems (5.2).

For both projects we present the experiments that have been performed using previously mentioned techniques and examine the results. We provide an accurate evaluation of advantages and possible drawbacks of proposed methods, especially compared to other similar methods published so far.

The last chapter contains final considerations and personal learning I acquired during my PhD, as well as future works and challenges I would like to face.

2
Model Compression Overview

The amount of research studies on the Deep Learning field during the last years has exponentially increased. Neural networks are capable of unquestionable flexibility since they can be exploited to solve an infinite number of tasks. The most stunning results from studying these architectures is the capability to solve complex tasks, sometimes even better than humans [138].

As a consequence, over the past few years, industrial applications have been exploiting extensively the advantages coming from Deep Learning, in particular when using Convolutional Neural Networks (CNNs). The intrinsic flexibility of these networks makes them widely adopted in a variety of practical applications, from medical to industrial. In particular, image and signal processing tasks are well-suited for the convolutional architecture, therefore a huge number of Computer Vision solutions have been proposed in the literature. Deploying a large and accurate model to perform a certain task takes considerable energy and space. Even if this might not be a problem during the training phase, it becomes a big issue at inference time, especially if the model has to run on devices with reduced computational resources or small storage space [126, 208].

This is the case with many modern applications, including IoT devices, smart sensors and cameras, drones, robots, smartphones or any other kind of device characterized by limited resources and low energy consumption requirements. For this reason, the adaptation of inference networks to embedded systems has been covered by many researchers [182], devising solutions ranging from architectures specially crafted for Field Programmable Gate Arrays (FPGAs) [82, 161] to techniques focused on low consumption for wireless and mobile devices [218].

Searching for the best architecture to train over a certain task is not a trivial problem. As large architectures lead to overparametrization and possible overfitting, using a too shallow configuration certainly lowers the overall performance. Finding the best trade-off between smallest network capacity and highest accuracy often requires many attempts and good expertise.

Indeed, according to the type of task to perform, two main approaches are to be found in literature. The choice is between training a small specialized network from scratch, or compressing a large pre-trained network and adapting it for the task.

Given the wide range of existing algorithms used to compress Neural Networks, it can be very challenging to categorize them into completely separated groups, and multiple hierarchies have been proposed.

Keeping in mind that usually a network is either compressed, either used as a starting point to build another more performing model, we propose to partition the studies present in literature into three main groups, namely:

- *Architecture search*
- *Pruning*
- *Quantization.*

We consider belonging to the architecture search group all the algorithms where an initial architecture is modified. So, for example, looking for the smallest possible architecture, introducing a new design for a certain layer, or using the initial network to train another one, are all techniques which will be presented inside this section.

On the other hand, both pruning and quantization do not seek a new architecture, but focus on modifying an existing model to reduce its footprint without affecting the accuracy in a relevant way.

Pruning methods focus on removing unnecessary parameters from a model. These parameters can be weights, nodes or even whole layers. For sake of simplicity, we consider removing a layer as searching for a completely new architecture, therefore we explore these types of approaches within the related section.

The last group relates to quantization. While pruning reduces the number of parameters of a model, quantization instead reduces the amount of memory used to store such parameters.

Even if compression algorithms can be semantically divided into groups, at implementation time there are no constraints on choosing only one method belonging to a single category. Indeed, what most often happens is that multiple strategies are applied together in a pipeline, in order to produce a model compressed as much as possible. As the number of publications on new compression algorithms grows noticeably every year, the way it is possible to combine different techniques together also grows exponentially. At the same time, this makes it extremely hard to compare the performance of such joint-way or hybrid methods between each other.

Other barriers to comparison are the ambiguity of the architecture or the metrics used, like choosing outdated architectures for comparison, or picking a dataset too small or not complex enough [10]. Moreover, there might be different implementations of the same architecture as the authors did now share their code, or different evaluation metrics, often dependant on the implementation environment.

To overcome these problems it is of uttermost importance to clearly define which evaluation metrics have to be used. The *compressing ratio* is defined as *original size / compressed size* of the model, while another commonly used parameter is the number of floating-point operations (FLOPs) performed by the network. Some papers misuse the compression ratio term by not defining if it refers to the size of the model, to the number of parameters or how many FLOPs have been been avoided. In other cases the compressing ratio is computed as *1 - (original size/compressed size)*. It is important to notice that the number of FLOPs can variate across different implementation platforms and libraries, so sometimes for the same architecture and dataset, different papers report different numbers for the FLOPs value.

2.1 Architecture-Related Compression Methods

Given a determined dataset and a specific task, there is an infinite number of possible architectures, and their related training processes, that can greatly perform over the proposed problem. The scientific community mostly focuses on reaching the best results, often regardless of time or resources constraints. In an industrial environment, unfortunately, sometimes such limits require to find new and smart solutions just to solve a simple task. Even if deep learning models are extensively used in many industrial applications, time has a key role in this field. Looking for a performing network has to be done as fast and as efficiently as possible. For this reason, most of the time it is not feasible to just train a large range of network configurations, while also looking for the best hyperparameters to match. Instead, it is of vital importance to reduce as much as possible the time spent for architecture search. To accommodate such requirements it is very important not only to start from an already almost optimal architecture, but also to use simple models that can be retrained in a short time, if needed. Even if in some cases a lot of time can be spent on searching for the best architecture (or the hardware available allows to train multiple networks in parallel), the program still has to produce the output as fast as possible, since it is often introduced in the online production pipeline.

On the other hand, one of the most important features when using deep learning to solve industrial tasks is the level of accuracy. Reducing the inference time is strictly related to limit the depth or complexity of the model used, leading to an unavoidable drop in accuracy. To create a balance between performance and inference time of a network the choice of the architecture has the absolute priority.

Fortunately, a lot of literature deals with the search of the best architecture, sometimes just to increase the performance, other times to reduce the model complexity. A good strategy consists in looking for an architecture that has been used to solve a similar task. If such architecture exists but after training it on our dataset did not fit our data too well, the best strategy is to either add some layers and/or nodes and try this new, more complex architecture, either to look for a completely different, deeper structure. In the eventuality that after training the model instead performed well on our data, hopefully even overfitting, we use such architecture as the starting point to look for a smaller one. Even if the smaller network we are looking for is not intended to be transferred on embedded systems, it is always a good practice to not exploit a complex network, but rather to look for one that fits the data just right.

This section focuses on some of the most popular methods that target the architecture of a network to make it more computationally efficient. In particular, tensor decomposition focus on simplifying the tensor representation by factorizing the weight matrices, lightweight networks target the convolutional layers by modifying their structure, and finally knowledge distillation use the information produced by a complex trained model to train a simpler one.

2.1.1 Tensor Decomposition

The computational cost during training and testing time of Convolutional Neural Networks heavily depends on the convolutional operations. As a consequence, a lot of effort has been put on improving the efficiency of these operations [19, 136]. One of the techniques most widely applied to CNNs is *Tensor Decomposition* [35, 71, 81, 104, 176].

The convolutional operation in 2D CNNs takes as input a 3D tensor where the first two dimensions are the spatial dimensions (height and width), and the third dimension is the number of feature maps, or channels (depth). The output is still a 3D tensor, where the number of output feature maps, or depth, depends on the number of filters, or kernels, used in the convolutional layer. The output spatial dimensions depend on the choice of stride and padding, but usually they are similar to the input tensor dimensions. The convolution kernel itself is a 4D tensor: the first two dimensions are the spatial dimensions, followed by the input image maps and the output image maps. Tensor decomposition is usually applied to the kernel tensor, consisting of the network's weights, to reduce the overall memory footprint.

Tensor decomposition is the application of *Matrix Decomposition* or *Matrix Factorization* principles to tensors. Tensors are just the generalization of matrices, where a single dimensional tensor is an array, a two dimensional tensor is a matrix, and when the number of dimensions is higher than two it is usually just called a d-dimensional tensor or a d-way array (with $d \geq 3$).

Decomposing or factorizing a matrix into smaller parts is an old and well known problem in mathematics. Transforming complex data in an equivalent, but smaller, representation not only helps with analysing the data, but also provides a compressed version of them. Probably the most known algorithms that are applied to decompose matrices are the Principal Components Analysis (PCA) and the Singular Value Decomposition (SVD).

When dealing with tensors, such algorithms have to be generalized, with the advantage that the mathematics hold for an arbitrary number of dimensions [99]. Both the CANDECOMP/PARAFAC (CP) [14, 62] and Tucker [192] tensor decompositions can be considered to be higher order generalizations of SVD and PCA. Other well-known algorithm used for tensor decompositions are INDSCAL [14], PARAFAC2, CANDELINC [15], DEDICOM [61], and PARATUCK2 [63].

Tensor decomposition methods are based on two important properties:

1. if a d-way tensor has *Rank* $R = 1$ it can be decomposed into the *outer product* of d vectors;

2. the rank R of a tensor is defined as *the minimum number of rank-1 tensors which are needed to produce such tensor as their sum.*

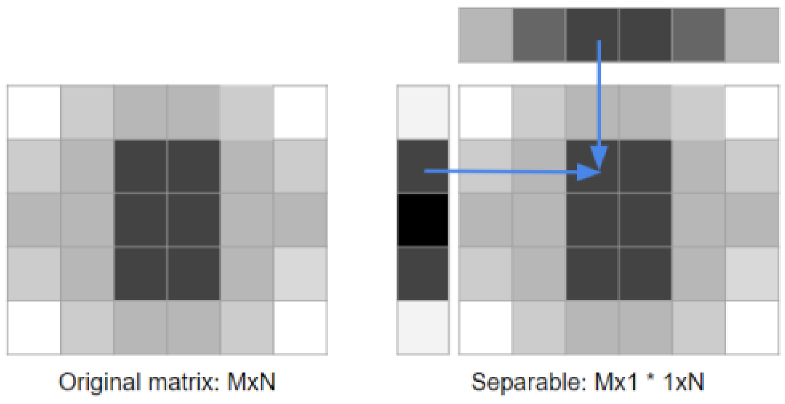


Figure 2.1: Decomposition of a rank-1 matrix.

In the simplest case, given matrix $A \in \mathbb{R}^{m \times n}$ with rank $r \leq min(m, n)$, the rank decomposition of A can be represented as $A = WH$, where $W \in \mathbb{R}^{m \times r}$ and $H \in \mathbb{R}^{r \times n}$. This means that the spatial complexity can be reduced from $\mathcal{O}(mn)$ to $\mathcal{O}(r(m+n))$ if the rank r is much smaller than m or n [33]. A graphic exemplification can be seen in figure 2.1.

The problem is that filters are usually far from having small rank, so they are usually approximated by a low-rank representation before being decomposed. This approximation introduces an error that we want to minimize. Unfortunately, such approach leads to a great loss in accuracy and can be usually applied only to fully connected layers [34]. A good low rank approximation of a matrix, instead, can be obtained by using SVD (see figure 2.2), as most information of the filters is described by the singular values. Such method has been proposed by Denton et al. in [35] where they propose to find an appropriate low-rank approximation for each convolutional layer through decomposition and clustering based on SVD in order to take advantage of similarities between learned features.

The principle for tensors is the same, if the rank is small, we can use a small number of operations between vectors to express the tensor, for this reason tensor decomposition is also defined as splitting a tensor into the sum of a finite number of rank-1 tensors. If the rank of a d-way tensor is smaller than d, not only it takes less space, but it is also computationally convenient to express it as its decomposition.

Ideally, we want tensors to have low ranks, such that they can be decomposed exactly, or approximated, as the sum of few rank-1 tensors. One solution might be to force the network to learn separable filters, but this poses a heavy constraint on the filter space, so a more effective option is to force the network to learn filters with a low rank, such that they can be decomposed or approximated.

One of the first tensor decomposition approaches has been proposed by Rigamonti et al. [167] to speedup codebook learning. They decompose a bank of filters X into linear combinations of a shared bank of separable (decomposable) filters Y. The decompositions of filters within Y are independent (components are not shared). In the same year Jaderberg et al. [81] applied the method to CNNs and improved it by approximating the

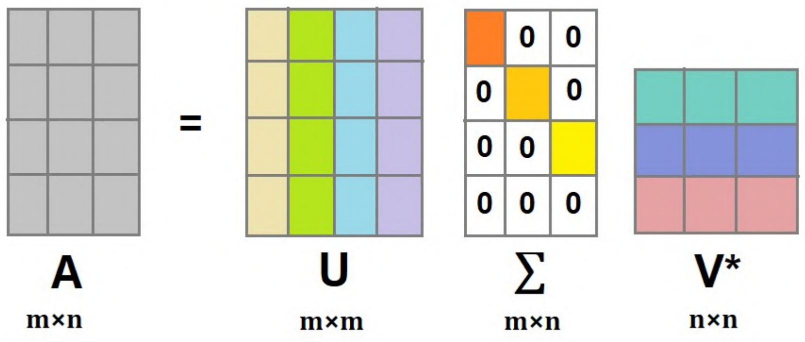


Figure 2.2: Singular Value Decomposition of a $m \times n$ complex matrix A. U is a $m \times m$ unitary matrix, Σ is a $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal (called singular values of A), and V is a $n \times n$ complex unitary matrix. If A is real then U and V are orthogonal matrices, and SVD is also defined as $U\Sigma V^T$. The number of non-zeros singular values is the same as the rank (r) of A. Taking the k largest singular values (with $k < r$) allows to produce the best approximation of A having rank k.

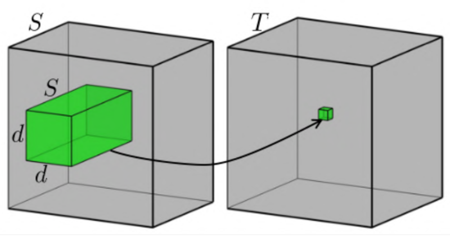
4D kernel tensor as a composition of two 3D tensors. Figure 2.3 shows the difference between the conventional convolution, the decomposition proposed by [81] and the CP-Decomposition of [167]. In [104] Lebedev et al. use the canonical polyadic decomposition (CP-decomposition, also called as CANDECOMP/PARAFAC model) [99] combined with the fine-tuning of the entire network, achieving considerable speedups with minimal loss in accuracy.

Tensor decomposition has gained increasing interest in the scientific community, and a large number of works have been published on the subject. Unlikely, current decomposition methods are more suitable for fc layers than convolutional layers, as fc layers have more redundancy and can be better compressed [33]. For these reason we do not focus further on such techniques.

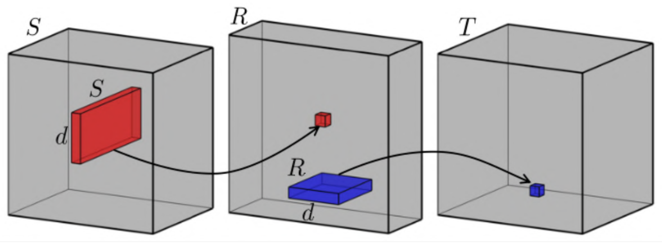
2.1.2 Lightweight Networks

Another somehow related approach consists on re-designing the convolutional layers, creating architectures that take advantage of small filters to reduce the memory footprint. Such networks are usually referred to as *Lightweight CNNs* or *Compact CNNs*.

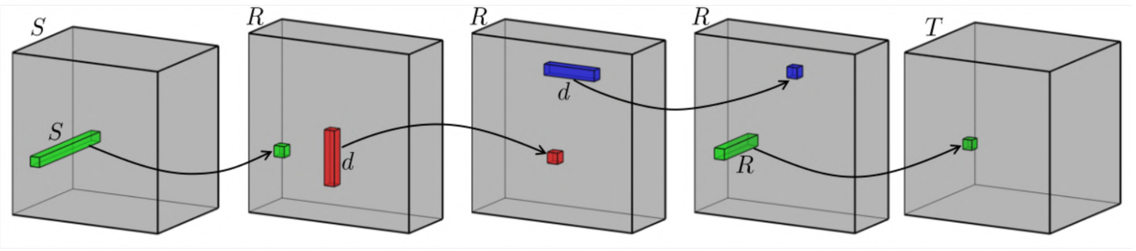
In this category belong the ResNet series [43, 67, 204] and NASNet series [123, 187, 229, 230]. Additionally, the MobileNet series [73, 74, 174], ShuffleNet series [132, 222], and EfficientNet series [188,189] achieve impressing performance while being lightweight



(a) Full convolution



(b) Two-component decomposition (Jaderberg et al., 2014a)



(c) CP-decomposition

Figure 2.3: Figure from [167]. Gray boxes correspond to 3D tensors of a CNN. Full convolution applies one filter to the input tensor S to compute a single value of the output tensor T. Jaderberg et al. [81] (b) approximate the initial convolution as a composition of two linear mappings with the intermediate map stack having R maps (where R is the rank of the decomposition). Each of the two mappings computes each target value based on a spatial window of size $1 \times d$ or $d \times 1$ in all input maps. Finally, CP-decomposition [167] (c) approximates the convolution as a composition of four convolutions with small kernels, so that a target value is computed based on a 1D-array that spans either one pixel in all input maps, or a 1D spatial window in one input map.

[119].

The idea of using small filters, in particular 1x1, was already exploited by Network in Network [117] and by GoogLeNet [184, 186] and their inception modules. In the literature, several efficient network architectures have become popular due to their compact size and low computational requirement [27]. SqueezeNet, developed by Iandola et al. [79], is a small CNN based architecture. The authors exploit three strategies: they replace 3×3 filters with 1×1 filters, they decrease the number of input channels to 3×3 filters, and they use downsampling late in network, so convolutional channels have larger inputs. They also define what is called a *Fire module* (figure 2.6) which substitute the convolutional layers by using a *Squeeze* and an *Expand* module. SqueezeNet has 50 times less parameter than the AlexNet and achieves the same accuracy on the ImageNet dataset. Moreover, using the compression technique introduced by Han et al. [58], authors reduce the size of SqueezeNet to less than 0.5 MB, which is 510× smaller in size compared to the original AlexNet model.

Another popular work has been published by Howard et al., who introduced MobileNets [74], an efficient architecture specifically thought for mobile devices. To create

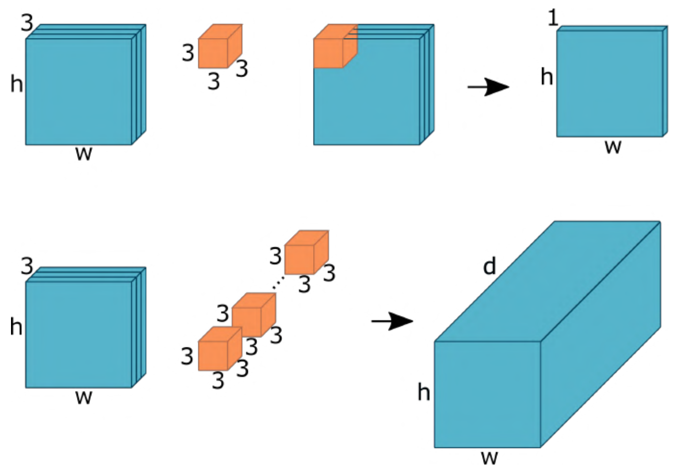


Figure 2.4: Traditional convolution. Given in input tensor hxwx3, for example an RGB image (top row left, in blue) a 3x3x3 filter (orange) is passed over the input. The depth of the tensor and of the filter have to be the same, while the window (height and width of the filter) can be arbitrary. As the filter passes over the input, it produces a feature map with depth one (top right). If multiple filters are used (bottom) the resulting tensor will be made of d feature maps, where d is the number of filters used.

the lightweight NN, MobileNets merge the idea of separable convolutions and depth-wise convolutions, proposing their depthwise separable convolutions: a combination of depthwise and 1×1 (point-wise) convolution. To create the depth-wise separable filter, standard filters are replaced by two layers: depth-wise convolution and point-wise convolution (see figure 2.5). The paper also introduces two hyper-parameters: the width multiplier α and the resolution parameter ρ. The parameter α can improve the computational cost and the number of network parameters, while ρ is applied to the input image to reduce the computational cost by an order of ρ^2. MobileNets are tested on various tasks like classification, object detection showing improved results over other existing methods. The MobileNets architecture has 22 times less parameters than SqueezeNet, while achieving 60% top-1 accuracy (better than both SqueezeNet and AlexNet). In 2018 Sandler et al. introduced MobileNetsV2 which revised the previous version. A similar solution has been adopted by another popular network called ThinNet [13].

Another computationally efficient CNN based architecture has been proposed by Zhang et al. [222], where they introduce new operations: pointwise group convolution and channel shuffle. ShuffleNet achieves approximately $13\times$ speed-up over AlexNet with similar accuracy. Tested on the ImageNet and the MS COCO [121] dataset, ShuffleNet reported better performance than the MobileNets. The MS COCO (Microsoft Common Objects in Context) dataset is a large-scale object detection, segmentation, key-point detection, and captioning dataset (the dataset consists of 328K images).

In another research, Huang et al. [76] introduced DenseNet, in which each layer of the network is connected to every alternate layer. DenseNet has several observable benefits like reduction in the number of parameters, feature-reuse and in lowering the vanishing

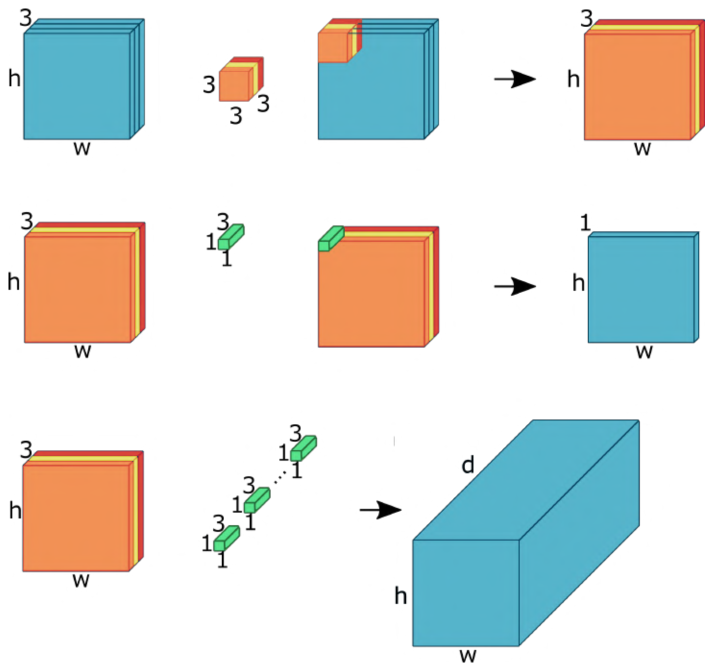


Figure 2.5: Depthwise separable convolution introduced in [74]. There are two steps. At the top row we can see a depthwise convolution: the depth of the feature map this time has the same depth of the filter used. The output on the right (hxwx3) is produced by the filter (3x3x3, in this case) passing over the input (hxwx3, in blue). After this operation a pointwise convolution (middle row) is used to reduce the depth and to produce the same output of a conventional convolution. If d 1x1 filters are used (bottom row) we have a tensor of depth d. The depth, or number of channels is 3 in this case, but it can be arbitrary.

gradient issue. DenseNet contains multiple dense blocks each dense block contains multiple layers. The experiments results with SVHN, CIFAR10, CIFAR100 and ImageNet datasets shows that DenseNet has performance improvements over other existing architectures [27]. Table 2.1 reports the number of parameters, FLOPs, and accuracy on the ImageNet dataset of the popular efficient networks mentioned above.

Such solutions are only some of the networks proposed in the last years. They are usually targeted for mobile devices, like [85, 199], but can also be implemented in large networks to reduce their size.

2.1.3 Knowledge Distillation

The idea behind *Knowledge Distillation (KD)* is to exploit a large trained network, called *teacher*, to train a smaller *student* network on the same task.

The concept has been introduced by Baciluă et al. in 2006 [12], when they used a large ensemble of models to label a large unlabelled dataset. Then, they trained a small neural network on the newly labelled data. This new network can be considered as a

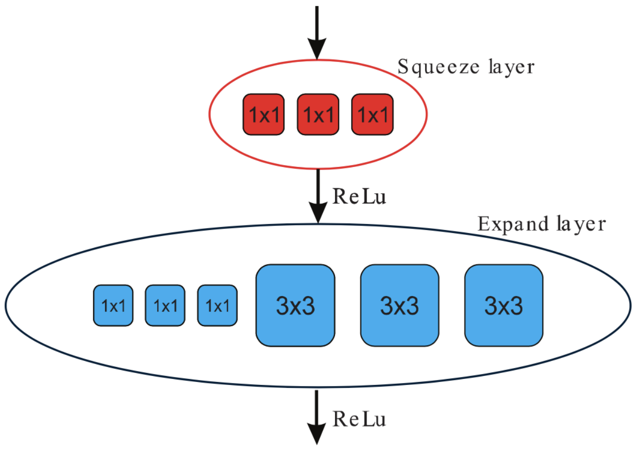


Figure 2.6: SqueezeNet Fire module [159].

Model	# Param (M)	FLOPs (M)	Accuracy (Top1)
SqueezeNet	1.25	1700	57.5
MobileNets	4.2	569	70.6
MobileNetsV2	3.4	300	72.0
ShuffleNet	3.4	527	71.5
ShuffleNetV2	5.3	292	73.7
DenseNet-201 (k = 32)	22	295	77.42

Table 2.1: Comparison of some Compact CNNs on the ImageNet dataset as reported in [174].

compression of the original setup, since it is several times smaller than the ensemble and it has almost no loss in performance.

In 2015 Hinton et al. in [70] further explored this concept by identifying the *knowledge* to learn as the mapping between input and output vectors, rather than the architecture and the learned parameter values of a single model. In particular, they discovered a lot of information is contained in the output vector and in the *logits*, the input values fed to the output layer. When the network produces the vector of probabilities related to a single observation, we expect the probabilities of belonging to the wrong classes to be much smaller than the correct class. Still, the distribution of these output probabilities holds plenty of information about the network generalization capability. They exploit the class probabilities produced by the complex, or ”cumbersome” model (as they called it), named *soft targets*, to train the student model.

The output vector of a neural network trained on a classification task is usually the output of a softmax layer, which takes the logit z_i for each class and gives a probability q_i, by comparing z_i with the other logits. They introduce a term T called *temperature* to

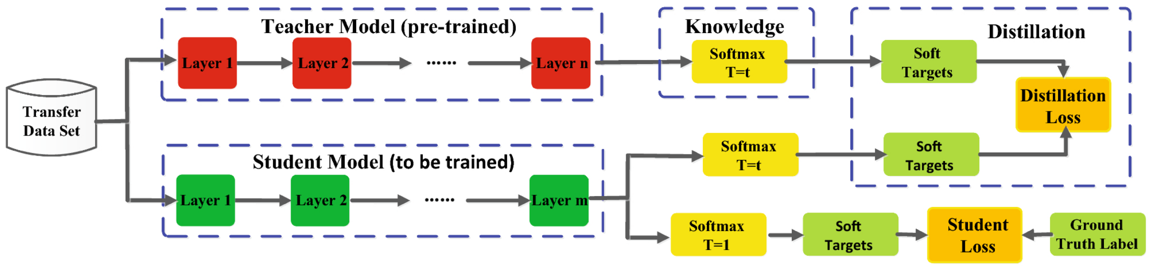


Figure 2.7: Model Hinton's knowledge distillation.

produce a softer probability distribution over the classes:

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)} \tag{2.1}$$

The *distillation* process tunes the final softmax using a certain temperature, such that the teacher model produces a suitably soft set of targets. The same temperature is set when training the student model.

The temperature T is usually set to 1, but higher values of T produce a softer probability distribution over classes. They found that the method can be significantly improved by forcing the student not only to match the soft targets, but also to produce the correct labels. For this reason the objective function is split into two. The first part consists of the cross entropy with the soft targets and the same high temperature for both models. The second part is the cross entropy with the correct labels. The best results are obtained by using a weighted average of the objective functions with a considerably lower weight on the second one.
Figure 2.7 shows the basic model of knowledge distillation proposed by Hinton and his team. An important result they obtained was that their method works remarkably well even if the dataset provided to the student network during the training lacks any observations of one or even more of the classes.

The knowledge distillation paradigm has received increasing attention from the research community in the recent years. It is a powerful method to compress neural network, improving the accuracy of low-precision networks and being widely applicable to a lot of models [142].

Distilling the knowledge from large powerful deep neural networks involves decisions on three different components: the type of knowledge to learn, the distillation algorithm to use, and which teacher-student architecture to deploy [51].

The information transferred by Hinton is considered a feature-based knowledge, where the output layer and the logits are used to teach the student network. This concept has been further extended in Fitnets [169] by using also the output of previous hidden layers (feature maps). Since then, a lot of other feature-based works have been published, for example, focusing on what to put the attention to [77,215], using also the parameter sharing of intermediate layers [224] or proposing a cross-layer knowledge that assigns adaptively proper teacher layers for each student layer [21]. Feature-based knowledge algorithms are the most common, but knowledge can be also response-based [22, 141], where only information from the output layer is used, and relation-based, that further explores the relationships between different feature maps [156, 210].

Another important topic when applying knowledge distillation consists in which architectures to use. Most works focus either on which knowledge transfer scheme to use between the two models [42, 150, 198], either on designing their respective architectures [52, 127, 203].

How to transfer the teacher knowledge to a student is one of the key steps. There are different possible distillation schemes, according to when the the knowledge is transferred. Offline distillation is the vanilla version, where the teacher model is first trained on a set of samples before the distillation takes place. Methods using the offline scheme [70, 169] focus on the transfer process. Online distillation [20, 53] instead aims to improve the student performance by training both models at the same time. One last scheme is called self-distillation, where same architecture is used both for the teacher and the student [144, 220].

Given the complexity of the KD paradigm, a large number of distillation algorithm has been proposed. From using the adversarial networks [50] to generate synthetic data to use by the teacher [23, 125], to transferring the knowledge of more than one teacher to a single student [41, 212], as well as using graphs [111], or even exploiting quantization [160].

Knowledge distillation has gained rising popularity in the last years. Even if traditionally it is used on classification problems and visual recognition, it has also been widely exploited for natural language processing (NLP) [124] and speech recognition [18]. It has also plenty of other applications: for example it has been used to preserve data security [198], to fight adversarial attacks [155] and also to avoid catastrophic forgetting [109].

2.2 Pruning

Fitting large inference networks to embedded systems is a topic that attracted the attention of many researchers in the recent past. One possible solution involves deploying the DNN over the cloud [113], where there are no limits on computational power and memory. Keeping the model on the cloud means to send the data to process and wait for the results. This process raises many problems: there is not only the latency introduced by the exchange of information (which highly depends on the available internet connection), but also introduces privacy issues, as we might be sending sensitive data. One last drawback is the total cost of ownership (TCO) of the data center, since the model is not stored and run locally.

For these reasons, when using small devices or embedded systems, it is more convenient to process the data locally on the device rather than sending them to a server for processing, fetching the results, and then making the decision [27]. On the other side, running a DL model on a mobile system implies to take into consideration the available storage space, the energy used to run the model, which highly affects the battery, and the computational capability, which is related to the time to produce a response. In particular, the time required for a DRAM operation is several times higher than having the network stored on the cache. This implies that it is utterly important to have very light models with as few as possible parameters.

Fortunately, already in 1990 LeCun [107] has verified that not all the parameters of a neural networks are important, meaning that most of the trained models are over-parameterized [34]. When such parameters are removed, or *pruned*, the network achieves great compression, often with minimal performance drop.

During the last years, great effort has been put on techniques that compress existing networks in such way. In particular, the compression is achieved by reducing the number and/or precision of each weight. Methods that focus on edges/nodes removal are usually referred as *Pruning* methods [10], while algorithms that target the precision fall into the *Quantization* category [30, 58, 164].

Pruning comes in a lot of different flavours, and sometimes it is difficult to actually understand which method performs the best [226], especially considering the high inconstancy of performance for different application scenarios [130]. Most pruning algorithms, though, belong to two main categories: either *weight pruning* either *filter pruning*.

Both methods have their advantages and drawbacks, and will be analyzed in the next sections.

Even if there are numerous works published tackling both approaches, there is always a common pipeline that most methods follow. The first step of course concerns deciding *what to remove*: pruning is often referred also as *network sparsification*, and the object targeted for pruning reflects the granularity of sparsification. For example, pruning can be element-wise, vector-wise, block-wise or even layer-wise. Weight pruning and filter pruning are the most used approaches, and they correspond to element-wise and block-wise sparsity. The second decision concerns how to decide *what is superfluous*. For example a connection that has weight zero, or a filter which weights have very low magnitude. This step is very important, as pruning random connections or nodes can damage the trained network beyond repair [213]. Most of the algorithm use one of the following criteria [148]:

- using magnitude-based pruning, looking at the weights or filters with the smallest absolute value [114]
- forcing the network to learn small weights, exploiting the regularization mechanism (*e.g. l1, l2, lasso*) [112, 197, 200]
- analyzing how removing a certain parameter affects the network (*sensitivity analysis*) [39, 88, 110, 146]

The next step involves *where* the pruning occurs: can the algorithm be run on all of the network or should only some layers be taken in consideration? The pruning process halts according to a threshold, for example we can decide to prune a certain percentage of weights or filters from specific layers, for *layer-wise* pruning methods, or set a *global* limit and let the algorithm pick which layers to target.

Finally after every pruning operation usually we need to assess if there has been any loss in accuracy, and if possible to *recover* the performance loss.

2.2.1 Weight Pruning

An early approach to pruning was the *Biased Weight Decay* [60], but the pioneering works on weight pruning are the *Optimal Brain Damage (OBD)* [107] and the *Optimal Brain Surgeon (OBS)* [64]. These methods reduced the number of connections based on the Hessian of the loss function, which achieved higher accuracy than magnitude based pruning methods (*e.g.*weight decay). In particular, OBD estimates weight importance by making a local approximation of the loss with a Taylor series and uses the 2^nd derivative of the loss with respect to the weight as a criterion to perform a type of weight sharing constraint. OBS algorithm improves the previous work by preserving the off diagonal values of the Hessian, as they claim these terms to be important for pruning. As 2^nd order derivatives are expensive to compute and the approximation proposed may not be sufficient to represent the full Hessian, other approaches use instead the $1st$ order derivative [134, 191]. Another famous work on weight pruning is the deep compression proposed by Han *et al.* [58, 59], demonstrating how a large portion of weights can be removed without affecting the network performance. Their simple method performs an iterative ablation on weights with small absolute magnitude, followed by a fine-tuning step. They also further compress the pruned network by grouping the connection that survived using Huffman coding and exploiting quantization (see figure 2.8).

A following work by Guo *et al.*introduced dynamic network surgery [55] to reduce the complexity of the network by connection pruning. The proposed method has two-part, pruning and splicing. In pruning, the unimportant weight connections are removed while in splicing, if any time the pruned or mistakenly pruned parameters are found to be important, splicing enables the recovery of such connections. Pruning and splicing are performed in an iterative way. Different from Han's algorithm, which needs more than 4800K iterations to get a 9x compression rate, dynamic network surgery only requires 700K iterations to get 17.7 times compression rate with similar accuracy. In 2017 Zhu *et al.* [226] proposed a simple magnitude-based gradual pruning method that can be incorporated in training and requires minimal tuning to achieve the preset level of sparsity. The results show that the proposed algorithm decreases the number of non-zero elements in the network by pruning small magnitude weights.

Recently, the paper published by Frankle *et al.* [40] has received increasing attention. They claim that in dense, randomly-initialized, feed-forward networks there *winning tickets*: subnetworks that achieve test accuracy comparable to the original unpruned network. They identify a ticket by training a network and then pruning its smallest-magnitude weights. The difference with any previous work is that after the pruning process they recover the original random initialization used before the same network was trained. Finding such configurations highly depends on the random initialization, usually requiring many attempts, but generally deeper network have a higher chance to contain such subnetworks. Winning tickets are less than 10-20% of the the size of many fully-

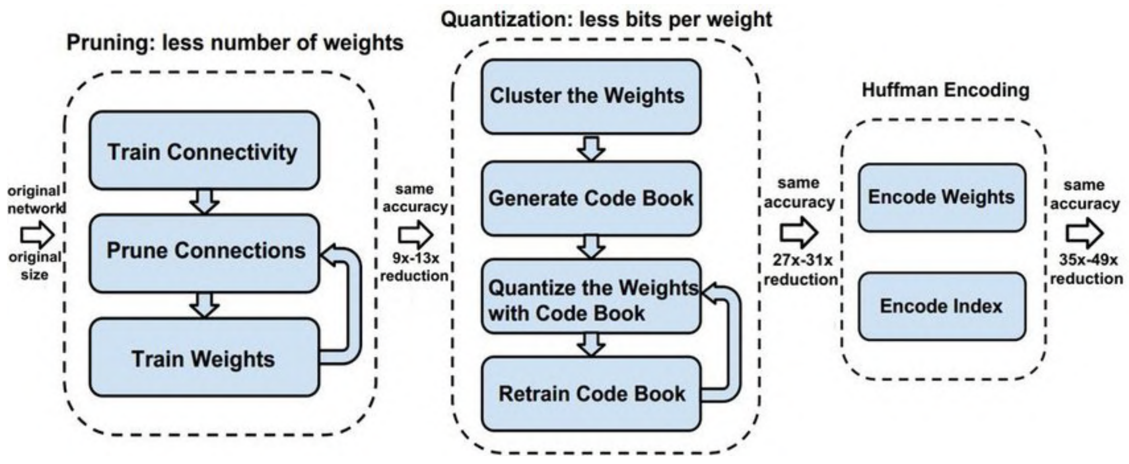


Figure 2.8: Pruning and quantization scheme of deep compression proposed by Han [58]. After pruning the network, the weights are quantized. This promotes weight sharing, and by using a Code Book the network can be efficiently compressed.

connected and CNNs architectures for MNIST and CIFAR10, without any performance loss. They are recently gaining a lot of interest in the scientific community, since there is an interesting implication: winning tickets can be transferred to work on a different tasks [133, 145, 153, 194]. They can been found also in architectures trained for object recognition [47] and pre-trained bert networks [24]. Another important consequence of this work is that it is always better to start with a larger network and prune it, rather than using a smaller configuration from the start, as large trained networks are publicly available, and they can be fine-tuned and then compressed for the specific task.

Sparse pruned networks achieve higher accuracy than their unpruned dense versions, and also work better than having the sparse architecture re-trained from scratch [33], but the connections to remove have to carefully chosen.

2.2.2 Filter Pruning

Weight pruning techniques remove single connections by setting some weights to zero, but the resulting sparse matrices cannot exploit BLAS libraries and are hard to implement on FPGA [58]. A simpler and more structured manner is to prune whole *kernels* from a Convolutional Neural Network. This procedure is often referred as *filter* or *kernel pruning* or sometimes *trimming* [75].

There is a large amount of literature about filter pruning, but one of the most natural grouping has been proposed by Lin et al. in [118]: in this paper they separate existing methods considering what the pruning algorithm takes into consideration to assess which filter to remove. In particular, they distinguish methods focusing on *property importance* and others concentrating on *adaptive importance*.

In the first group we find methods which prune filters according to intrinsic properties of the networks, and do not modify the training loss.

In 2016, Hu et al. [75] proposed a layer-wise method which analyses the neuron outputs to compute the *Average Percentage of Zero (APoZ)* activations after the ReLU mapping. The idea is to remove neurons with an APoZ larger than one standard deviation from the average APoZ of the target trimming layer. Instead of looking at the outputs, the method proposed by Li et al. [114] measures the relative importance of a filter in each layer by computing the sum of its absolute weights (i.e. its *L1-Norm*). A more recent approach has been suggested by He et al. [68]. It expands the norm-based filter criterion by computing the Geometric Median (*GM*) of the filters within the same layer. The idea is that filters close to the GM can be represented by the other filters, and therefore are good candidates to be pruned. The authors illustrate how the smallest norm filters can be very important, as they could actually be larger than zero or they can have a small norm deviation. In 2020 Lin et al. [118] proposed a method called Filter Pruning using High-Rank Feature Maps (*HRank*). They claim that average rank of multiple feature maps generated by a single filter is independent from the distribution of the images. Filters which generate lower-rank feature maps are less important and can be removed first in a one-shot manner, requiring only a few fine-tuning epochs after the pruning phase.

Adaptive importance methods like [120, 129] usually achieve better compression and speed-up than property importance based ones. On the other hand, these techniques change the loss function up to the point that retraining becomes a separated problem, usually requiring to search again a new best set of hyper-parameters.

There is a last class of filter pruning algorithms that deserves to be mentioned. Sometimes filter pruning is exploited to find the best sub-network from an original one. It is the case of [44] and [180] where they use PCA to compressing both length and width of the network. Their goal however is more related to the architecture search topic, rather than to compress an existing network without the need to retrain it completely.

Methods aiming to remove parameters from the network, regardless where the connections or the filters are, can be considered *global*. Usually there is only one threshold to be set, such as the number of filters or the compression rate to achieve. Some methods focus on specific layers, usually relying on certain statistics to pick the most promising ones. Rather than global methods, these *layer-wise* pruning algorithms require more than one threshold or other parameters to be set, making them less robust.

Finally, pruning is usually performed in three stages: (i) preparation of an appropriately large network either by training it from scratch or by adapting an existing trained network using transfer learning; (ii) removal of superfluous parameters and (iii) fine-tuning to recover the loss of accuracy. The second and third stages are often repeated until the network reaches the desired level of compression. Some methods, however, perform pruning in a *one-shot* fashion by removing a chosen set of weights in a single pass [118].

Filter pruning presents several advantages with respect to weight pruning. It is a more structured way to remove parameters from the network, and for this reason it is often called *structured pruning* while weight pruning is referred as *unstructured pruning*. As pruning the weights gives more control on what to discard from the architecture, pruned connections are never actually removed, but rather just zeroed-out. Even if this can reduce the memory required to store the model, the number of operations (both Multiply Accumulate (MACs) and Floating Point operations (FLOPs)) does not change. Multiplications are still performed, the only difference is that the matrices are sparse. Accelerating such operations requires specific libraries and even specific hardware. On the other hand, each operation performed by a filter pruning algorithm corresponds to creating a new architecture with one (or multiple) filter missing, skipping all the multiplications previously related to such filter. The number of parameters involved by removing a kernel is surely more significant than removing a single connection. This results in filter pruning algorithms to remove more parameters in a single fashion than any weight pruning techniques. Removing a batch of weights, rather than a single one, might cause the accuracy to drop much faster. For this reason filter pruning techniques have to pay much more attention to the performance loss and quickly recover it if necessary. Anyway, the advantages of filter pruning over weight pruning, such as the higher model compression ratio, the reduced number of memory accesses and the efficiency from a hardware perspective, makes structured pruning the best approach for mobile implementation [135].

2.3 Quantization

The weights of CNNs are typically stored as 32-bits single precision floating point (FL) numbers. Using a smaller number of bits to represent the weights not only results in reducing the memory required to storage the model, but also in lowering the power consumption [131]. For this reason, half-precision and integers (16 bits) are commonly used. Such low-precision can be even further decreased, by using only 8, 4, 2 or even 1-bit representation to store the weights. The adoption of fixed point representation for the network weights and data can significantly reduce memory footprint and computation resources, but this often comes with a price: less precision most likely leads to lower accuracy. To recover the loss in performance quantized networks are usually fine-tuned to settle on the new weights.

There are multiple data objects that can be quantized in a neural network, not only weights and activations, but also error [225], gradient, and weight update [202]. There is a plentiful number of quantization techniques proposed in the literature, proposing different levels or quantization [30, 54], also designed for specific devices [201]. For instance, ternary neural networks Mellempudi et al. [139] ternarize pretrained full precision models and constrain activations to 8 and 4-bits. In [143] the authors propose to use non-uniform, base-2 logarithmic representation to encode weights, taking advantage of the non-uniform distribution of weights and activations in a trained network.

The maximal level of quantization is given by network binarization: the work in [164] proposes two efficient approximations of standard CNNs, namely Binary-Weight Networks and XNOR-Networks. In Binary-Weight Networks the filters are approximated with binary values, while in XNOR-Networks both the filters and the input to convolutional layers are binary, so that convolutions use primarily binary operations. Other methods involving binarized networks can be found in [31, 78, 137, 177]. Other approaches such as [219] propose to jointly train a quantizer together with the CNN, instead of using some fixed quantization schemes. Such technique can be applied to both weights and activations, with an arbitrary-bit precision.

I

Industrial Applications

During my PhD I have been working to find and optimize solutions for industrial applications for a specific company. In particular, I studied extensively the best setups for different Convolutional Neural Networks to solve image classification problems. CNNs are a very effective and versatile tool to address a wide range of Computer Vision tasks, and they are largely exploited in industrial applications. Contrary to the academic environments, where data are usually manually collected from few experiments, this process is often automatized by industrial machines, where thousand of images can be easily collected and stored to be used during the training. Even if a certain degree of human inspection is always required, for example to correctly label each image or to discard inaccurate acquisitions, there is an undeniable advantage in dealing with industrial machines when creating a proper dataset. Indeed, one of the most important requirements to properly train a CNN is to have a large dataset available, with a good variance of the observations. Since the data acquisition step is usually not a problem, most of the focus is usually centered at maximizing the accuracy of the model, while minimizing the inference time. Finding the best trade-off between performance and number of parameters of a model is the main topic of this thesis, and in this part I present some of the problems that I had been working on during these years.

3 Related Work

3.1 Fruit processing

One of the industrial applications of computer vision algorithms is fresh produce classification. Classifying fruit and vegetables results particularly challenging, especially due to interclass similarities and irregular intraclass characteristics [57]. The sensors used to acquire the images, as well as the type of preprocessing and the selection of relevant features play a very important role for a successful analysis of the acquisitions. Current literature on the topic covers problems like quality assessment and robotic harvesting, but the algorithms are usually very specific and use extremely limited datasets.

Machine learning techniques like Support Vector Machine (SVM), K-Nearest Neighbour (KNN), Decision Trees, and finally Artificial Neural Net-works (ANN) and Convolutional Neural Networks (CNN) are the most exploited methods to extract features meaningful for the classification. The work proposed by Sun *et al.* [181] suggest that exploiting hyperspectral information can improve the performance of the classification algorithm. Similar works use such methods in the food industry [17,26].

In table 3.1 we can see some interesting papers on fruit and vegetable classification. Most existing algorithms usually focus on classification [11,223], quality assessment [16, 193], fruit harvesting [87], object detection [28], bruise detection [84], disease detection [56] and grading [4].

year	Fruit/veg	Dataset size	Classifier	Accuracy	Ref
2015	Olive	77	Fisher Discriminant Analysis (FDA)	100.00%	[46]
2015	Fruit	(5 classes)	Transfer Learning	50.00%	[103]
2016	18 fruit	1653	NN and Deep Learning(DL)	99.88%	[223]
2017	Tomato	-	ANN	98.50%	[214]
2018	Lettuce	320	CNN.	86.00%	[16]
2018	Dates	8000	Caffee Net	99.24%	[72]
2018	Tomato	150	BPNN	100.00%	[196]
2018	Orange	355	Naive Bayes, ANN, Decision Tree	93.45%	[195]

Table 3.1: Comparison of machine vision techniques for fruit and vegetable classification inspired by [57].

3.2 Signal processing

Given the incredible results CNNs have achieved when working on 2D signals, 1D CNNs have recently been proposed and already reached state-of-the-art performance levels in several applications, such as personalized biomedical data classification and early diagnosis, structural health monitoring, anomaly detection and identification in power electronics and motor-fault detection [94].

For example, many researchers have tried to use deep CNNs for fault diagnosis of bearings, where the data are collected by multiple accelerometers. One of the most common approach consist in simply reshaping the 1D signal (*vibration*) into a matrix (*vibration image*) [221]. Another technique applies the Discrete Fourier Transform (DFT) to the vibration signals collected from two accelerometers [83]. For electrocardiogram (ECG) beat classification and arrhythmia detection, the common approach is to first compute power- or log-spectrogram to convert each ECG beat to a 2D image [171, 228]. After the data are transformed in such a way, it is possible to simply feed the matrices to a conventional CNN. The drawback of these approach is that 2D CNNs require a conspicuous size of data for the training process, while for many practical 1D signal applications labelled data are often scarce. Another limitation consists on the hardware side, as 2D CNNs usually require a GPU for the training process, while most of 1D applications are simple enough to use only the CPU. Finally, given their high computational complexity they are not suitable for real time applications on embedded system with memory and power constraints.

The first compact and adaptive 1D CNNs to operate directly on patient-specific ECG proposed [98] by Kiranyaz et al. in 2015 overcame these problems. In a very short time it became popular and the same approach has been used not only for early arrhythmia detection in electrocardiogram (ECG) beats [96, 97], but also for other purposes, such as structural health monitoring and structural health detection [1, 2, 5, 6], high power engine fault monitoring [80], real-time monitoring of high-power circuits and damage detection in bearings [95].

Peak Detection A typical application of 1D CNNs is Peak detection. In particular the approach in [140] proposes peak detection and the peak integration in raw liquid chromatography–mass spectrometry (LC–MS) data. Authors developed a pipeline named *peakonly*, which excludes low-intensity noisy peaks and shows good results in the detection of true positive peaks. First some interesting ROIs are identified within the signal, then a CNN classifies ROIs into three classes: (1) ROI does not contain peaks (2) ROI contains one or more peaks (3) ROI contains something like a peak, but a specialist is required. A second CNN similar to U-Net is used for a segmentation problem to tell whether the point in a ROI corresponds to the peak or not.

The method presented by Kanazawa *et al.* [86] introduces a technique to generate fake chromatograms using a generative adversarial network (GAN). Results show that such synthetic data are effective for training and evaluating peak-picking neural networks. In [92] the authors present an automated peak detector based on Faster R-CNN [165]

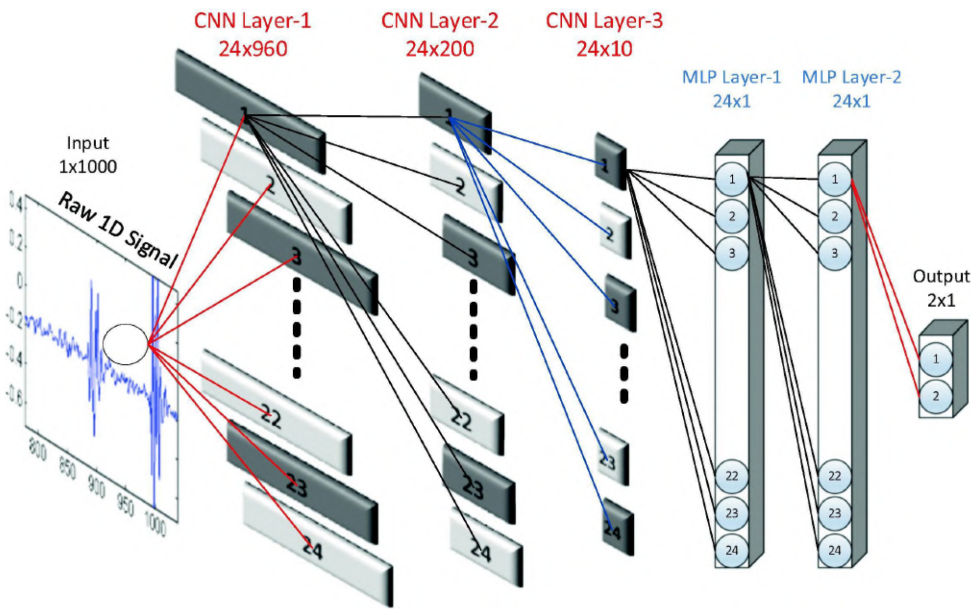


Figure 3.1: Example of a 1D CNN to classify ECG data.

optimized for the task. The peak detector architecture is considerably simpler than the architecture used for natural image classification since peak detection involves only two classes (i.e., peaks and non-peaks) and the geometric patterns of peaks (like relative size and sharpness) are the only information needed by the network. Finally, the work proposed in [151] involves the usage of a CNN to identify human-labelled peaks in ChIP-seq (chromatin immunoprecipitation followed by sequencing) data. This task is particularly challenging since sequencing errors, local bias and biological variability make complicate solving the peak-calling problem.

There are several differences between one and two dimensional convolutional neural networks. In figure 3.1 we can see an example of a classic one dimensional CNN used to classify if the electrocardiogram signal presents signs of arrhythmia. While conventional CNNs work on tensors, all the operations in a single dimension CNN are array operations, simplifying tremendously both the forward and backward pass. As a result most of the time they do not need dedicated hardware for the training phase, since they are easier to implement and faster to train. Another advantage is that there is usually no need for complex architectures, as using only a few hidden layers with a few neurons is enough to reach and even surpass the performances achieved by other methods that do not exploit neural networks.

One dimensional CNNs are often not considered, as the community focuses most of the attention on its 2D counterpart, but it is one of the most powerful tools when dealing with signals and 1D data.

Ellipse fitting and detection In the literature, the amount of applications proposed for CNNs in image processing tasks is huge [38]. Some of the most popular tasks include image segmentation [179], feature extraction [25, 206], object recognition [36, 165].

Among the wide range of algorithms, a number of works employ CNNs for ellipse detection and fitting. This can been as addressing the peak detection task in the two dimensional space. Some of them rely on a well-known CNN to preprocess data in order to identify the regions of interest where an ellipse is present. For instance, in [205] Faster R-CNN is first used to get the regions where ellipses may lie. In [116] Mask R-CNN is first used for pupil region segmentation, then estimation of pupil localization parameters is performed introducing anchor ellipses and ellipse regressions. Finally, Li *et al.*developed a Gaussian Proposal Networks [115]. Such model learns to propose bounding ellipses as 2D Gaussian distributions on the image plane. An interesting application for detection of knots in sawn lumber is presented in [154]: the authors adapted Faster R-CNN with to model elliptical objects with a Gaussian function, and extended the Gaussian Proposal Network architecture. The Wasserstein distance is employed in the loss function to predict the precise locations of elliptical objects.

4
Fruit Image Classification

One of the first tasks I had been working on during my PhD was the classification of images of fruit. In particular, I worked on training models to classify the goodness of different fruit, especially olives and dates. The new classifier had to be integrated with the already working software of a multisensorial optical sorter, an industrial machine that provides optimized sorting of fruits and vegetables. The machine has ultra-high-speed image processing with a resolution of up to 0.1 mm/pixel, and can detect even the smallest and most difficult external quality defects.

The image acquisition process occurs in a controlled environment by this industrial machine (see fig. 4.1). The fruit is dropped at one end of the sorter on a carousel, which is made of parallel rolls, rotating the fruits in place while also moving them forward. The fruit enters in an obscuration box containing cameras and synchronized lights. The RGB camera and the infra-red sensor (NIR) are synchronized with the illumination system (the NIR image has to be captured in the dark). By adjusting the speed settings of the machine, it is possible to take between 7 and 20 images for each fruit. This results in a collection of images from different angles, as the machine forces the fruit to spin around their major axis (see figure 4.2). Images of the positions where the fruit was missing are automatically discarded by using a simple threshold (this is not a problem as the background is black). The acquired images are cropped according to the olives borders, where the size of the largest image is $86 \times 96 \times 3$ pixels, while the smallest one is $58 \times 56 \times 3$ pixels.

After collecting the images, they have to be classified in real time, as the next and final part of the machine has to separate the fruit according to their quality, usually by driving them to different directions by means of compressed air.

The main task consisted in creating a model for the classification of a particular variety of green Turkish olives called *Edremit*. A small portion of the olives presented some kind of malformations which made them unqualified for selling. In particular, two kind of defects occurred, the first one could be identified as the presence of gibberish parts, a deformity which often happens in many other kind of fruit, modifying the usually smooth surface of the olives in an unpleasant way. The gibbosity often was more visible around the pedicel, where the fruit is attached to the tree. A certain degree of deformation was still considered normal and acceptable. Such defect was defined by the suppliers as *Takoz*, and we maintained the Turkish name for the olives presenting this kind of feature (see fig. 4.3).

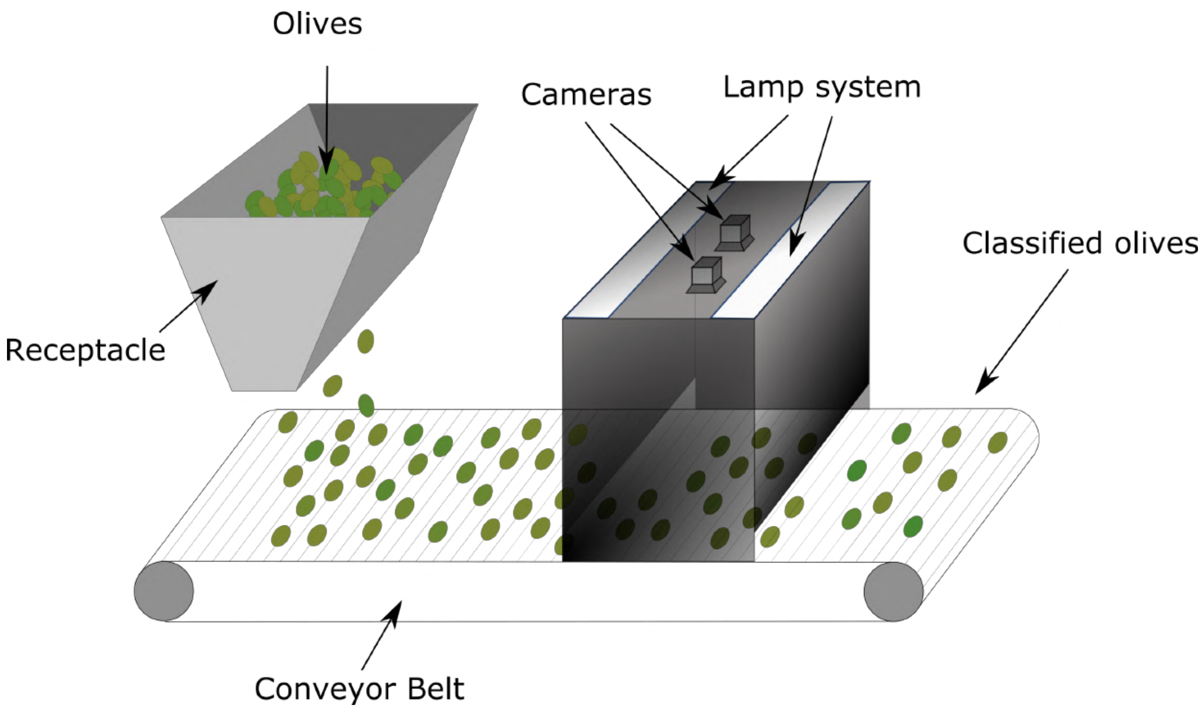


Figure 4.1: Image acquisition system. The conveyor belt is specifically designed to place and align the olives on the same axes, while rotating them in place. After the images are captured there is not much time before the olives have to be sorted: the classification algorithm has to predict the correct class in real time.

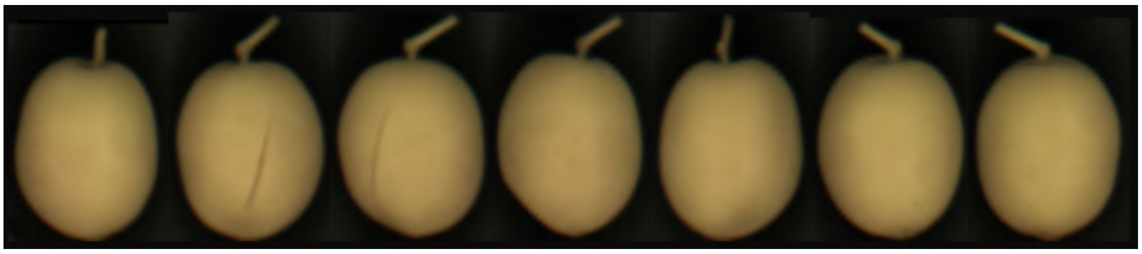


Figure 4.2: Multiple images from different angles of the same olive. The number of sequences depends on the settings of the machine. This particular fruit presents some interesting features: firstly it has a long petiole, while it is rare to even find fruits with a small one still attached. Secondly, the fruit shows some cuts (second and third acquisition). Such features have both to be learned by the network as not relevant to the classification task. To achieve this, it is vital for the network to see plenty of such instances.

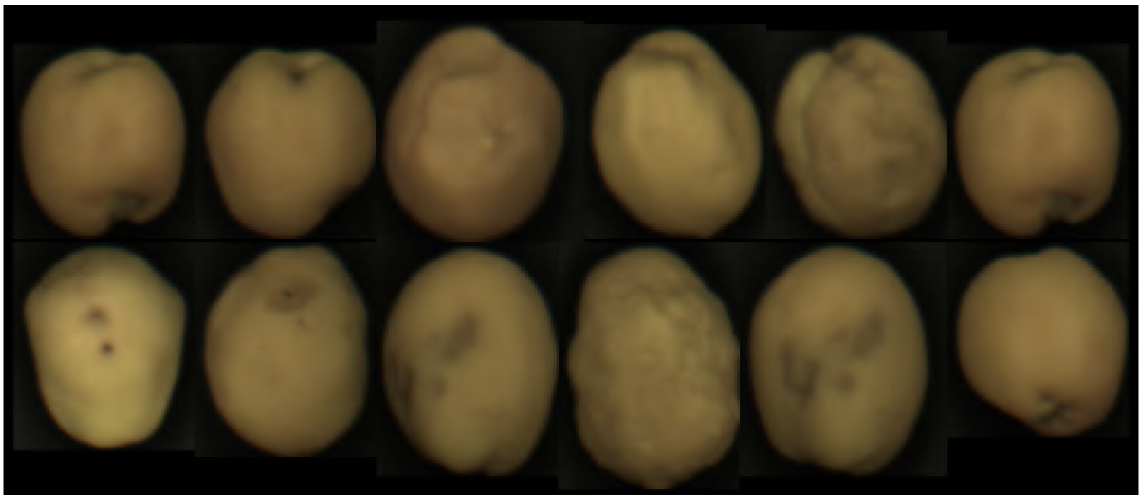


Figure 4.3: Some examples of olives with the Takoz malformation. This defect does not affect the goodness of the olive, but only the external appearance. This fruit can be still used for other purposes, e.g. the production of oil.

Even if only a small portion of the olives belonged to this category, a even smaller part presented another type of defect. Some of the olives displayed a furrowed surface, often coupled by patches of darker skin, similar to bruises. Such kind of rare defect was named as *Wrinkled* and fruit presenting such feature had to be discarded.

As already mentioned in section 1.7, the creation of the dataset is one of the most important tasks to produce accurate models. Some of the acquisitions had to be discarded. In particular, olives presenting both types of malformation could not be included in the dataset. Another problem was the degree of defect for the Takoz class. The humps showed by the image had to bee prominent enough to be selected as not saleable without possible doubt. Some of the olives have been damaged before the image acquisition, and the only way to distinguish if the olive belonged to the wrinkled quality or the darker spots were hollow parts, was using the near infrared (NIR) channel see figure 4.4. For all these reasons, only the images presenting the defects (or the lack of them) in unmistakable manner were included in the dataset.

Since the number of defective olives was significantly smaller than the normal, round, smooth fruits, collecting the images for the database creation lead to have unbalanced classes. This problem was even more prominent for another breed of olives called *Pink* that we had to classify. These kind of olives, reddish in color, only presented the wrinkled malformation, or at least only the fruit we were provided with did. In particular, the number of Pink olives we had available for the data collection process was extremely small, to the point where it was impossible to train a network able to find the defective reddish olives. The client required to train multiple models for different tasks, for example to distinguish good-for-sale olives only of the Edremit variety, or to classify the quality when both the green and reddish olives are processed at the same time. Even if the Pink olives are similar to the green type, they still are smaller and present higher values for the

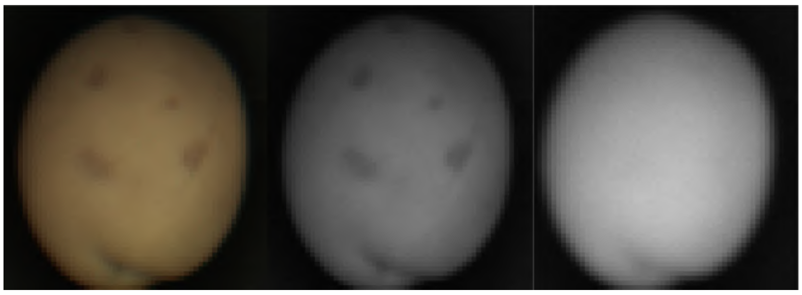


Figure 4.4: An example of an olives presenting darker spots. From the RGB (left) or the grayscale (middle) images it is not easy to understand if such dark patches are bruises or just a discoloration, but using the NIR (right) we can see that the surface of the fruit is actually smooth.

red channel, so the network needs to see a good amount of such olives (see figure 4.5). To face such problem, we exploited data augmentation and the large available dataset of green olives to forge new images of the Pink quality. This method has been published in [45] and is further discussed in section 4.2.

4.1 Classifying Edremit olives

At the beginning of my PhD I worked extensively in finding the best architectures to solve different classification tasks on fruit images. In particular, one of the requests of the supplier was to classify olives of the Edremit variety using CNNs, achieving both small inference time and high accuracy. I was required to train multiple models to classify between the three possible classes (*good*, *takoz* and *wrinkled*), but also to focus only on the distinction between olives good for sale and olives presenting only the Takoz deformity. Such models needed to have a very small number of parameters and FLOPs, while still reaching high accuracy. Such limitations depended on the implementation hardware, as exploiting a GPU in industrial machine is expensive, especially when maintenance is required and because we need to use licensed external libraries.

The faster approach to obtain high performance classification methods is to find a network trained on a similar task (for example small networks with high accuracy on CIFAR10) and try the same architecture on the available data. Such process is particularly time consuming, as a lot of possible configuration exists, especially when considering what optimizer to use, the optimal batch size, the degree of data augmentation (how many transformations) to implement, which activation function works the best, the type of pooling to apply and so on.

Even if such parameters can be decided in a heuristic way, the number of layers and the relative number of filters play an important role in the model complexity and the resulting inference time. Before trying any possible compression approach, it is vital to try many possible architecture configurations and compiling options, to find the best

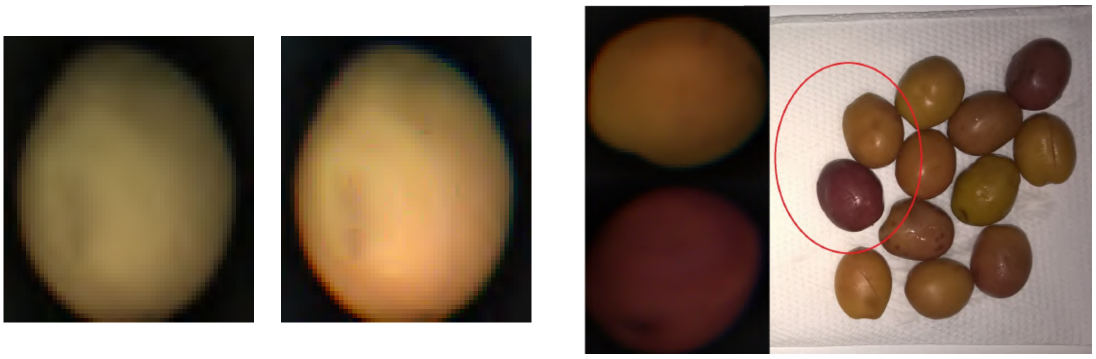


Figure 4.5: On the left there are the images of an olive before and after the compensation. After the compensation process the olive looks a lot more orange than what someone would like it to be. Anyway, comparing the images with the physical olives we can see that the compensation is correct with respect to the actual color of the olives. On the right picture we can see how the color of the olives after the compensation is closer than the real color of the fruits. The olives belongs to two different breeds, the *Endremit* veriety, which is the one we have in our dataset, and the *Pink* variety, that is reddish.

trade-off between model size and accuracy.

Even when a small trained network performs well on the data, there might be some issues depending on the image acquisition process. As the images of the fruit are acquired by using a specific machine, sometimes the settings of the camera or different version of the software can result in having very different captures of the same fruit in a different machine of the same series. Such inconvenient often requires to resume the training process and add new images presenting the new variance.

Unfortunately, the final version of the architectures implemented in the sorter cannot be disclosed due to secrecy, but advanced draft of possible configurations is presented in section 5.3.2. On the other way, it is not necessary to have highly optimized architectures to be able to appreciate the advantages coming from using handmade models. Even during my master's thesis [166] I have achieved significant results by working only on the Edremit dataset.

In particular, I studied how the complexity of the network affects the prediction time and the accuracy, and I compared standard architectures widely used as AlexNet, GoogleNet, ResNet and various versions of VGG networks with small custom networks that I designed and trained personally. Table 4.1 shows the accuracy and inference time of some of the most popular neural network, that have been used to classify the images of Edremit olives in two classes: *good* or *not-for-sale* (in case of presence of the takoz deformity).

Finally, I implemented and trained from scratch a new custom architecture, by using only 5 convolutional blocks and a softmax layer for the prediction, similar to this:

conv	**max pool**	**relu**	**conv**	**relu**	**avg pool**	**conv**	**relu**	**avg pool**	**conv**	**relu**	**conv**	**softmaxloss**
5x5x3x32	3x3	-	5x5x32x32	-	3x3	5x5x32x64	-	3x3	4x4x64x64	-	1x1x64x2	-
stride 1	stride	-	stride 1	-	stride 2	stride 1	-	stride 2	stride 1	-	stride 1	-
pad 2	pad [0 1 0 1]	-	pad 2	-	pad [0 1 0 1]	pad 2	-	pad [0 1 0 1]	pad 0	-	pad 0	-

	time per image	# layers	best on olives	top-5 error
AlexNet	0.0391	8	0.9881	19.2
VGG-F	0.0457	8	0.9877	18.8
VGG-M-2048	0.0683	8	0.9915	15.8
VGG-M-128	0.0781	8	0.9793	18.4
VGG-S	**0.0890**	8	**0.9923**	15.3
VGG-VD-16	**0.2299**	16	**0.9965**	9.9
GoogleNet	0.1209	22	0.9905	12.9
ResNet	0.1734	50	0.9912	7.7

Table 4.1: In this table we can see the results of using state-of-the-art architectures to solve the classification problem on the Edremit dataset. The networks have been adapted by using transfer learning. In particular, only the output layer has been replaced, and they have been fastly retrained freezing the filters in the first layers. Notice that the best accuracy is achieved by VGG-16, but it takes about 230 ms to classify one single image. Even considering the smaller version VGG-S, about 90 ms are required to process a single image.

From my work emerged that the size of the input plays a crucial role in the complexity of the network, as a lower number of layers can be used to extract and combine significant features across all the input image. The number of channels, instead, is what impacted the most on the accuracy, as using grayscale images lead to lower performances than using the RGB version, and a small further improvement could be obtained by using the NIR channel too. Since the small inference time was a strong requirement in the deployment of the models, using only one channel (grayscale images) and resizing the images from 64×64 pixels (a size similar to the original) to 32×32 pixels was the best combination to use to reduce the model complexity. While this solution implied to lose a lot of information and lower the performance, using basic data augmentation and doubling the training set by simply mirroring the images, greatly improved the overall accuracy. The resulting networks achieve 98.80% accuracy while needing only about $1.6ms$ for predicting a single image. This work is important to underline how most of the times it is unfeasible to exploit pretrained networks in an industrial context, especially when there is the need of real time algorithms. The small inference time is even more important in this context due to the requirements of the application: as the fruit rolls and expose different sides, it is vital to classify multiple views of the same olive in a short time, as there might be only one or two acquisitions where the defect is noticeable.

4.2 Classifying both Edremit and Pink

Within modern Deep Learning setups, data augmentation is the weapon of choice when dealing with narrow datasets or with a poor range of different samples. However, the benefits of data augmentation are abysmal when applied to a dataset which is inherently

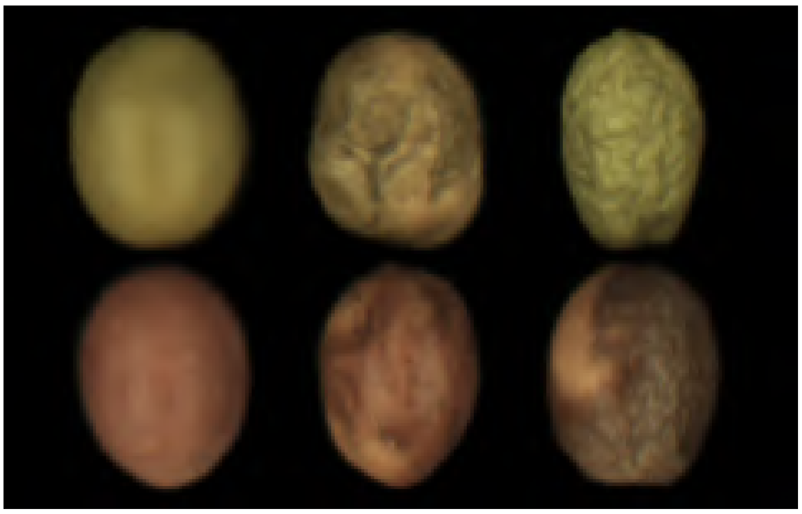


Figure 4.6: Sample of Edremit (top) and Pink (bottom) images. They are affected by similar defect patterns (wrinkled) which must be detected (middle and right). However Pink are more rare and it is difficult to gather a significant dataset. Would it be possible to design a training process exploiting the abundance of Edremit to better classify Pink ?

unable to cover all the categories to be classified with a significant number of samples. To deal with such desperate scenarios, we propose a possible last resort: Cross-Dataset Data Augmentation. That is, the creation of new samples by morphing observations from a different source into credible specimens for the training dataset. Of course specific and strict conditions must be satisfied for this trick to work. In the paper [45] we propose a general set of strategies and rules for Cross-Dataset Data Augmentation and we demonstrate its feasibility over a concrete case study (olive image classification). Even without defining any new formal approach, we think that the preliminary results of our paper are worth to produce a broader discussion on this topic.

The main idea underlying this paper is to exploit data augmentation and transfer learning to train a model able to classify both Edremit and Pink olives. Specifically, we are dealing with the case where very few samples are available for the family of objects to be classified, still large datasets exists for similar, but not identical, objects. Within this scenario we are seeking to exploit the information from the second set (and from networks trained on it) to obtain a network capable of classifying objects from the first set. For this process to be feasible a few conditions must hold. Broadly speaking, our ideal scenario provides objects that are mostly similar between the two datasets except for some features. Such set of inter-dataset distinctive features must be disjoint with the set of the inter-category features that are to be used for classification within each dataset. These inter-category features, in turn, must be shared between datasets.

While, at this stage, it is difficult to generalize some cross-dataset data augmentation approach, with this paper we try to define some basic principles and to define a general recipe to adopt to deal with similar scenarios. In particular, we are interested in comparing different combinations of augmentation strategies to see which one yields better results. In order to perform this task, we apply cross-dataset data augmentation to solve a real-world problem: the classification of olive fruits that can be affected by specific defects. By doing this, we supply additional collateral contributions by designing two novel network architectures to cope with fruit defect detection with special attention to performance and speed.

4.2.1 Cross-Dataset Data Augmentation

According to the generally adopted semantic within the Deep Learning field, data augmentation is any synthetic alteration of dataset images, performed with the goal of inflating the number of samples to be supplied to the training process.

We introduce the concept of *Cross-Dataset Data Augmentation* (CDDA), a term used to describe a novel pre-processing strategy involving the transfer of information between two datasets by means of image processing. The term recalls the idea of a data augmentation step which, instead of processing the samples internal to a dataset, works by transforming samples between a source and a destination dataset.

The main assumption is the existence of a strong relation between the source and destination datasets. That is, we assume that they share most of the features that are significant to perform a certain classification task, while they differ with respect to features that can be ignored by the classifier.

Specifically, we model each dataset as a (possibly infinite) set of features (see Fig. 4.7). Some feature are global, such as color histograms, orientation of the blob, PCA

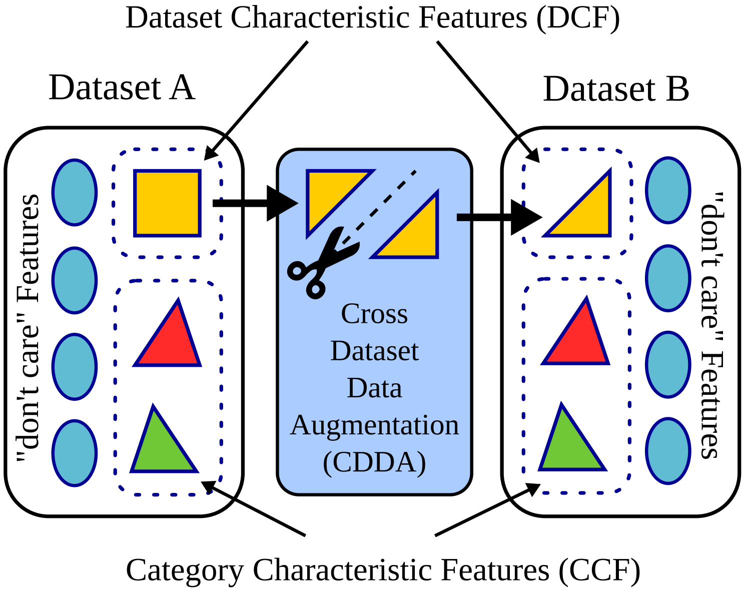

Dataset Characteristic Features (DCF)
Dataset A
Dataset B
"don't care" Features
Cross Dataset Data Augmentation (CDDA)
"don't care" Features
Category Characteristic Features (CCF)

Figure 4.7: A graphical representation of the main concept underlying Cross-Dataset Data Augmentation.

components, etc. Some others are local, such as intensity maxima, corners, and so on.

Given two datasets, A and B, each of their features can be deemed to belong to one of the following three sets:

- *Dataset Characteristic Features (DCF)*: these are features that distinguish one dataset from the other. The idea is that these features should not take a role in the classification we want to perform. For instance, if we want to classify vehicles with respect to the size of the wheels, then the number of the wheels is a DCF. In fact, it distinguishes bikes from cars, still it is not a constraint if the classification is about the size and not the number of wheels. Of course, this does not mean that DCFs are not distinctive in general. Of course a classifier trained to distinguish bikes from cars will exploit them. They are just not distinctive with respect to the specific classification task we are dealing with.
- *Category Characteristic Features (CCF)*: these are features that are important for the classification task we are considering. For CDDA to work, CCFs must be shared between datasets A and B. Going back to the example with vehicles and wheel sizes, the radius of the tire is a DCF and the number of spokes could be one, depending on the classifier. In principle a classifier that uses CCFs to distinguish A from B should not work as they are preserved between the two datasets.
- *"don't care" features*: these are simply the features that are in the datasets, but are not significant for a specific classification task.

Let A and B two *strongly related* datasets, with B that contains too few samples to be effectively used to train a CNN. We assume the two datasets to be strongly related if there exists a set of features in common (CCF) and a disjointed set of features that allows to distinct samples belonging to the two (DCF). Then, the CDDA is some image processing function $f_{CDDA} : A \rightarrow B$ that transforms samples belonging to the dataset A into samples that could belong to the dataset B. This is achieved by applying a transformation to the DCF features of the sample. This results in a new dataset with higher cardinality that could be used to train a more efficient CNN. Our bet is that the newly forged dataset allows to train more efficient networks which outperform networks trained using standard data augmentation strategies. Of course the nature of f_{CDDA} strongly depends on the characteristics of A and B and it is not possible to give a general rule at this point. However it is possible to experiment the effectiveness of the concept with a real-world case.

In our specific case-of-study, we address the problem of low cardinality dataset in an image classification task. While a more in-depth introduction of the datasets can be found in Section 4.2.2, we anticipate that the dataset contains olive images belonging to two different families. Figure 4.8 shows some sample images from the dataset. As will be shown in Section 4.2.4, the number of samples are not enough to achieve satisfying accuracy level, in particular when compared to the results obtained on a similar dataset (Green olives dataset). On the other hand, the Green dataset can be considered *strongly*

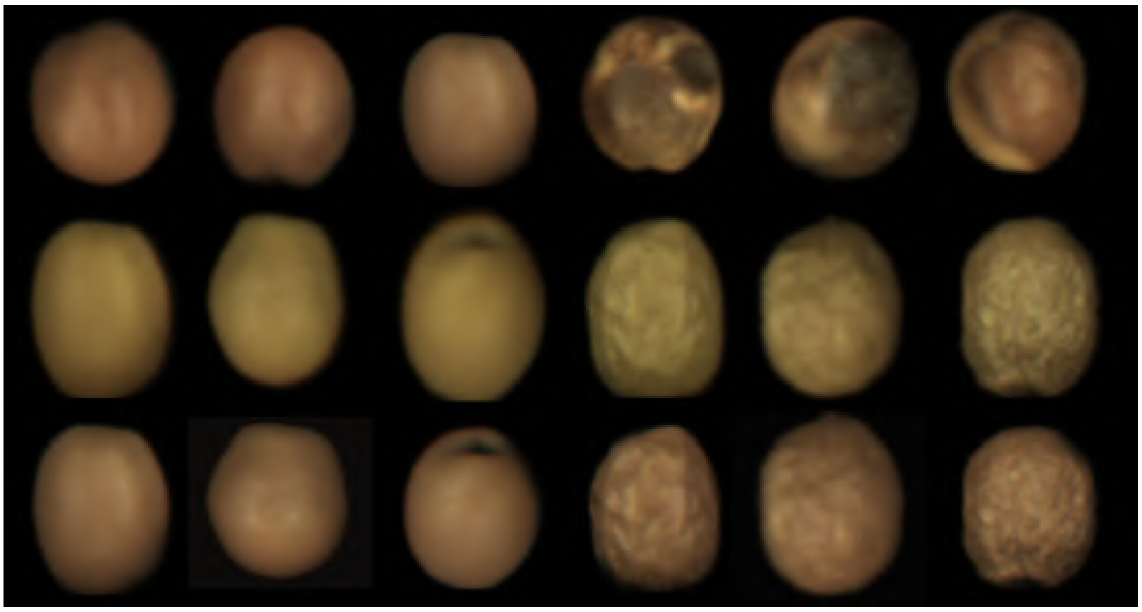


Figure 4.8: Some samples drawn from the dataset used in the experimental session. The rows show, respectively, sample from the Red dataset (good on the left, wrinkled on the right), from the Green dataset and the same images from the second row transformed according to the proposed Cross-Dataset Data Augmentation approach.

related to the Red dataset and thus a CDDA approach can be used to transform samples from the Green dataset to sample that belongs to the Red dataset.

In this first stage of the study, we decided to focus on the two most trivial differences between samples belonging to the two datasets, namely the shape and the colour of the olives. Thus, we define a two-step transformation function. Both transformations include a learning phase in which a standard representation of that particular feature is learned.

In the first step, we compute a set of parameters tied to the geometry (the shape) of the olives. In particular, after the extraction of the boundary from the images, we fit a bounding box and retrieve both the two perpendicular axes and their orientation. Then, we compute the *mean bounding box* representative of the red olives used to compute the affine transformation between each green olive bounding box to the mean red olive one. This results in the creation of several *reddish-shaped* green olives.

To deal with the colour difference, we simply learn the mean colour histogram for each channel and then apply a histogram equalizer in order to match the colour histogram of the green olives with the mean colour histogram of the red ones. The third row in Figure 4.8 shows the result of the application of these two steps to the olives in the second row.

4.2.2 Datasets Overview

The olives data evaluated in our paper consists of two fruit varieties: Endremit olives (characterized by a greenish texture) and Pink olives (which are more reddish).

We created a test bed to compare the performance of the proposed data augmentation strategy with respect to some of the more well-established data augmentation techniques. In particular, we tested a set of different datasets (which are the result of different pre-processing strategies) on both pre-trained networks (through transfer learning) and *ad hoc* networks, trained from scratch. The performance are assessed in terms of classification accuracy. In the proposed case-of-study, the datasets contain olive images belonging to two different categories, the ones labelled as *Good* and the the ones labelled as *Wrinkled*. The former contains defect-less olives, while the latter contains olives with wrinkles on the surface.

The first dataset (which will be referred to as *Green*) contains 14340 *green* olives, of which 7580 belongs to the *good* category while 6760 are *wrinkled*. The second dataset, which is much smaller, contains a different type of olives, which will be called the *red* ones. The *Red* dataset consists of 108 olives, divided into 60 belonging to the *good* class and 48 which show surface wrinkles.

As introduced in Section 4.2.1, our Cross-Dataset Data Augmentation counts two independent pre-processing steps that are pipelined together in order to transform a numerous but different dataset of images (the Green dataset) into the less-numerous dataset containing red olives (see figure 4.9). We will refer to this dataset as the *Green2Red* (*G2R* in shorts) dataset.

The performance of each net is assessed against eight different datasets. Each dataset is a combination of olives took from the *Green*, the *Red* and the *Green2Red* datasets (some examples are shown in figure 4.10). Images contained in a dataset are usually splitted into three subsets. The first, the most numerous one, contains the images which will be used to learn the weights of the net and is referred to as the *training set*. Basically, these are the sample data used to fit the model. The second is the *validation set*. It contains the sample data used both to provide an unbiased evaluation (at least at the beginning) of a model

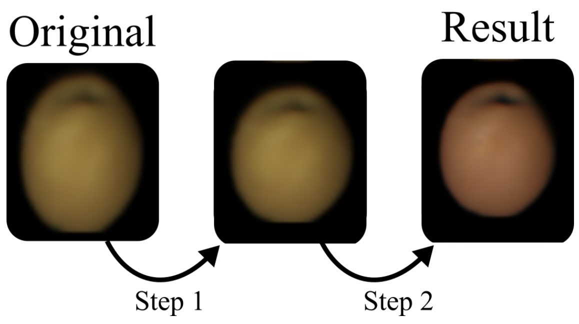


Figure 4.9: How a Green olive is transformed to be a sample of the Red type. In our case, the shape (step 1) and the color (step 2) are the only two features that separate the observations between the two datasets.

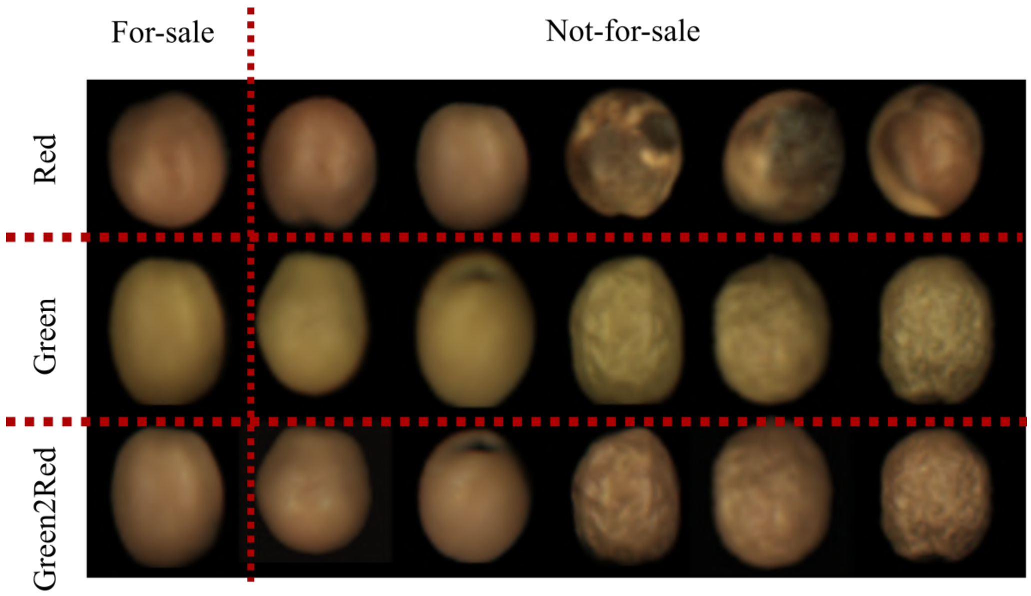


Figure 4.10: Images belonging to the Red class (top row) and to the Green class (middle row). The last row shows how the same Green images have been transformed into synthetic samples for the Red class. For all three categories, the first image from the left displays olives from the Good class, while the rest of the images correspond to olives with are not good for sale and need to be discarded.

fit on the training dataset and tuning model hyper-parameters. The last set is the test set, which contains the sample data used to provide an unbiased evaluation of the final model. Each network has been trained and tested on the following aggregated datasets.

- *Green*: the networks are trained and tested only on olives contained in the Green dataset. The green dataset contains a large number of samples, giving a basic idea of the performance that can be achieved in this specific case-of-study in the best possible scenario.

- *Red*: the dataset contains all (and only) the olives belonging to the Red dataset. The images are divided into the three sub-datasets introduced above, training (90%), validation (5%) and test (5%). Like for the previous one, the performance achieved on this dataset should be used only as a reference for the worst scenario for this particular case-of-study.

- Test 1 ($Green+Red$): in this test, the dataset contains olives from both the Green and Red dataset. Red olives are distributed consistently among the training, the validation and the test set (with a $8 : 1 : 1$ ratio).

- Test 2 ($Green|TRed$): the dataset contains all the olives belonging to the Green dataset, which are splitted into a training and validation set, while the test set contains only red olives.

- Test 3 (*Green*|*VRed*|*TRed*): the difference between the previous dataset and the current one is that a subset of red olives is used both for validation and test purposes, while the training set contains only Green olives.

- Test 4 (*G2R*|*TRed*): in this test, the training and the validation set images are picked from the *Green2Red* dataset (i.e. green olives after the proposed preprocessing steps), while for the testing phase we used only red olives.

- Test 5 (*G2R*|*VRed*|*TRed*): in the fifth test we train the networks on the *Green2Red* dataset, while the red olives are used in both validation and test set.

- Test 6 (*G2R+Red*|*VRed*|*TRed*): in the final test, we split olives belonging to the *Red* dataset among the training, the validation and the test set. Like in *Test 1*, red olives are consistently distributed with the same ratio.

Each test has been created to answer to a specific question. In particular, test 1 to 3 should answer to the question: *are the two dataset similar enough to allow networks mainly trained on the most numerous dataset to perform well on the less numerous one?* On the other hand, test 4 and 5 show how the proposed approach tackle the problem of low sample number datasets in our specific scenario. All datasets have been augmented with scale, random rotations, PCA transformation, flipping and translation.

The cross-dataset data augmentation and the tests on the neural networks (pre-trained nets and the *ad hoc* ones) are implemented in Matlab using the Neural Network Toolbox. Both the source code and the datasets are publicly available on the authors web pages.

4.2.3 Network Architectures

To assess the efficacy of the proposed cross-category feature transfer framework, we compare the classification accuracy achieved by the current state-of-the-art pre-trained networks for image classification alongside with the results achieved by two custom networks developed and trained from scratch for the specific task of defective fruits detection. In particular, we compare the results using the following architectures: AlexNet, GoogleNet, ResNet-50, VGG-16, VGG-19 and Inception-v3. All the architectures are desctibed in section 1.5. All the networks have been trained on the ImageNet database (or on a subset of it).

Furthermore, we propose two novel network structures to tackle our specific case-of-study. A discussion about the reasons that lead us to develop the *ad hoc* networks can be found in Section 4.2.4, while now we provide some insight on the network architectures. The first proposed architecture counts 6 computational blocks and will be referred as *FruitDef*. The first 4 blocks are convolutional blocks. In particular, odd blocks contain a convolutional layer followed by a *max-pooling* layer (which halves the previous layer dimension) and finally an activation layer (*ReLU*). The even blocks are pretty much the same, but we use *average-pooling* as pooling layer. The fifth block contains only 2 layers: a convolutional layer and a *ReLU* layer. While the last block is a fully connected

	Test 1 Green+Red | TRed	Test 2 Green|TRed	Test 3 Green|VRed|TRed	Test 4 G2R|TRed	Test 5 G2R|VRed|TRed	Test 6 G2R+Red|VRed|TRed
FruitDef	80.157	59.175	59.461	85.591	86.387	98.782
FruitDefV2.0	82.479	62.899	63.227	86.555	88.866	98.739
AlexNet	89.748	95.582	96.301	97.991	97.629	96.504
VGG16	90.555	95.537	94.139	98.396	99.581	97.699
VGG19	94.685	88.730	89.859	98.744	98.814	98.773
GoogleNet	96.892	96.093	98.372	99.650	99.650	99.028
ResNet50	96.683	97.350	96.373	98.605	98.117	99.744
Inception-v3	97.301	98.465	98.601	99.930	99.841	99.744

Table 4.2: Performance achieved by several networks in terms of classification accuracy. Test 4 to 6 show the results on the dataset created using the proposed data augmentation approach.

layer with a *softmax* layer which yields the classification result. The first 3 convolutional layers counts 32 filters of dimension 4×4, while the last two have 64 filters of the same dimension.

The second network (*FruitDefV2.0*) counts 8 macro-blocks. The last two blocks are the same as the simpler version ones, while the first 6 blocks are convolutional blocks. Differently from the network introduced before, only block 2, 3 and 6 contain a pooling layer (average, max and average respectively). In block two, we use two *dropout* layers (at the beginning and at the end of the block) with respectively 35% and 45% dropout percentages. Each block but the last one contains an activation layer (*ReLU*) right after the pooling/convolution layer. Both networks are trained from scratch using as datasets the ones introduced in Section 4.2.2.

4.2.4 Experimental results

Figure 4.11 shows a schematic representation of the tests that were conducted in order to assess the performance of the proposed approach. The *D*s blocks represent a dataset, while *C*s blocks represent the classifier learned on it. Without loss of generality, a possible strategy is to use a numerous dataset (*i.e. a*) to train a classifier. This classifier can be used as it is in the *strongly related* but less numerous dataset (*i.e. b*). On the other hand, the same network can be re-trained (using transfer learning) in order to improve the performance on the less numerous dataset.

Table 4.2 shows classification accuracies obtained with the different datasets on different network architectures. The results we are most interested in are the ones yield by the *ad hoc* networks, which, as you can see, perform slightly worse than the pre-trained networks. In an industrial application like the one requested for our case-of-study, run-time performance are central in the development of the network to be used. Hence, even

	Green	Red
FruitDef	99.868	94.143
FruitDefV2.0	99.736	94.214
AlexNet	100	94.948
VGG16	96.513	94.948
VGG19	100	91.428
GoogleNet	99.861	86.807
ResNet50	99.650	96.190
Inception-v3	97.350	94.413

Table 4.3: Performance achieved on the base datasets. The performance achieved on the Green dataset are the best-case-scenario performance (thanks to the high number of samples in the dataset).

if the pre-trained networks are able to achieve state-of-the-art performance classification-wise, they are too slow and not practical to use in a real world scenario. This is due to both the fact that the input images are much bigger than the original image dimension (about 3 times bigger) and the network structure of the pre-trained networks are much more complex than the *ad hoc* networks we proposed. The adoption of simpler networks whose input image dimension is equal to the original dimension of the images allows us to perform the classification task an order of magnitude faster (about $0.1ms$) on the machine the tests were performed (3.5 Ghz Intel quad core with nVidia 980 GTX).

The best performance are achieved by the most complex networks, like the Inception-v3. On the other hand, even simpler and faster pre-trained networks achieve similar performance. Simpler networks (architecture-wise) are the one that get the most benefit from

	Green	Green2Red
FruitDef	88.421	93.684
FruitDefV2.0	90.336	99.370

Table 4.4: Transfer learning test on custom networks.

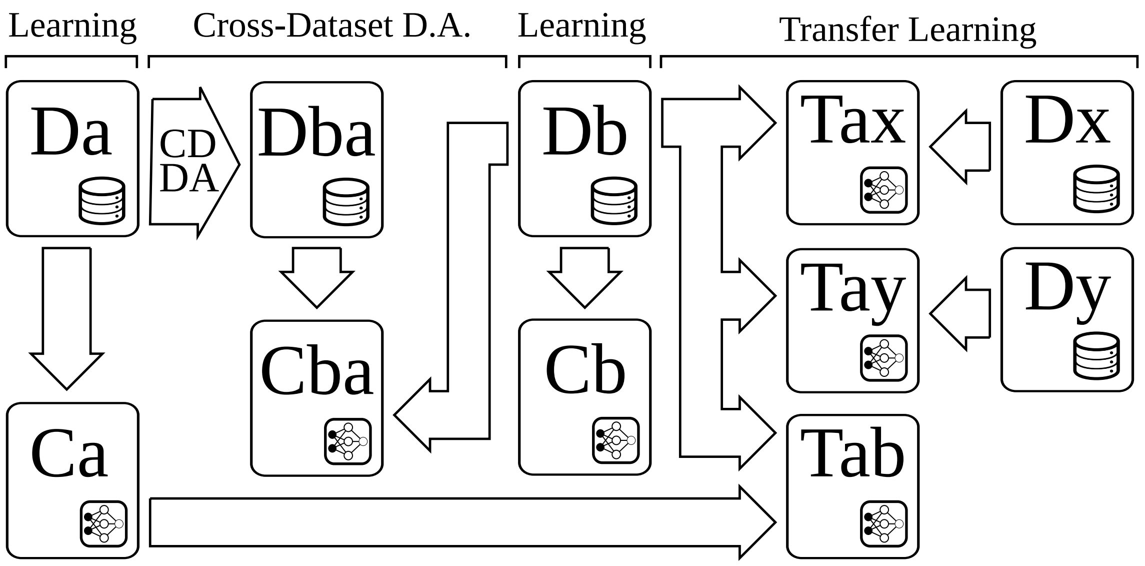


Figure 4.11: An overview of the possible strategies for data augmentation, including both traditional methods and cross-dataset data augmentation. *D* blocks represent dataset used to train a network, while a *C* block is a classifier obtained after the optimization process.

the proposed approach. All pre-trained networks show a performance gain using a CDDA approach, even if the mean gain is lower overall with respect to the gain on the custom networks.

Finally, we compare the performance of *ad hoc* networks on two last scenarios. The test involves training from scratch the same networks directly on a dataset of objects of interest (in opposite to general purpose dataset) followed by a transfer learning step on a more specific problem. In particular, we trained the custom networks on both the Green dataset and the Green dataset plus the Green2Red dataset. Then, transfer learning is performed using the olives in the Red dataset. Performance achieved are shown in Table 4.4.

4.3 Conclusions

In this chapter I studied a real-world problem, the classification of fruit images for an industrial sorter. The solutions consist in training small custom CNNs that achieve almost state-of-the-art accuracy while dramatically reducing the inference time.

I also worked with my colleagues on a new data augmentation approach in which the additional samples are created by mapping elements from a different pool instead of being generated from the dataset itself. While traditional data augmentation uses a function to generate unseen samples from the available ones, we take advantage of a different dataset which we assume can be mapped to the former by means of an image processing function. We observed that such approach tends to outperform the classical data augmentation, probably because it has better chance to preserve features that are relevant for the classification but not easily embeddable into the generator function. Again, the advantage is particularly visible in small custom CNNs, where the training phase is sensitive to the even distribution of different sample categories.

This data augmentation technique has been used to mitigate the scarcity of images of the Pink variety, allowing us to classify Edremit and Pink olives with the same network. Since the two species differ in term of shape and color, but not on the nature of their defect to classify, we investigated a mapping approach instead of the generative one. Results show that, especially for small networks, the former gives better results in almost all the tests.

5 Filter Pruning for Embedded Vision

Another project I worked on during my PhD consisted in creating a lightweight CNN to locate both 1D and 2D signal peaks respectively in line scan and area scan images. The final goal is to synthesize these networks on FPGA hardware, available on commercial smart cameras. For this reason, we designed light architectures able to perform advanced image processing, recognition of specific elements and computation of descriptors. The advantages are many: the possibility of offering a complete hardware capable of providing semantic information, the ability to reduce the weight of data transferred from the CNN-Camera to the processing systems, and in general the reusability of the framework for different tasks. After training the networks we exploited model compression techniques to reduce their size. We also introduced a novel pruning algorithm which does not require manual tuning of parameters and allows to greatly reduce the number of floating point operations (FLOPs) computed. In a throughout experimental section we show that the trimmed network achieves better results with respect to a network with the same pruned architecture, but trained starting from random weights. Moreover, the resulting performance is better than the one obtainable with other state-of-the-art trimming methods. Finally, by using other compression techniques, such as quantization, the resulting model can even be further reduced. This filter pruning technique can be used not only to train networks aimed to be implemented in embedded systems, but can also be used as a general method to reduce the number of parameters of a model without degrading its accuracy.

5.1 Motivation

As the number of applications exploiting intelligent algorithms keeps rising, not only for common devices as watches and smartphones, but also for drones and sensors, there is a need to create smaller and optimized networks. One of the common solutions is to implement Convolutional Neural Networks on Field Programmable Array (FPGA) [9, 91, 122], semiconductor devices that are based around a matrix of configurable logic blocks connected via programmable interconnects. Thanks to their programmable properties, they can be used for a large range of applications, not only in the industrial field but also for medical, automotive and communication purposes. However, they have a limited number of logic gates available, and only thanks to the most recent compression algorithm and efficient network structures it is lately possible to transfer CNNs on FPGA supports. The small size of these devices allows in particular to insert them in cameras and efficiently preprocess the images before they are even stored in the memory for processing.

5.2 Relevance Based Filter Pruning

We propose a global filter pruning method based on the idea that kernels can be ranked by means of a *relevance metric* computed according to the output after the activation function. Less relevant filters are iteratively removed in a *prune one and re-train* fashion, to allow the network to adjust to the reduced channel. Conforming with this process, we call our method ReFT (Relevance-based Filter Trimming).

More formally, given a convolutional layer, its input can be represented as a 2D tensor $I \times H$ containing a signal of I single $H-$dimensional feature vectors[1]. The convolutional layer contains K different $S \times H$ kernels that are convolved with the input to produce an $I \times K$ output tensor. This operation requires approximately $S \times H$ multiply-add operations for I filter shift for a total of $SHIK$ operations. Our goal is to define a *relevance function* to be computed over the K kernels in order to iteratively remove the kernel with smaller relevance. Each kernel removal will result in a reduction of the multiply-add operation in the order of SHI. Furthermore, it will reduce the size of the output of the layer of a factor proportional to the input I.

In detail, the ReFT iterative reduction process is performed according to the pseudo-code described in Algorithm 1. Here, $Kernels$ is the full set of kernels in the network,

Algorithm 1: ReFT network reduction

```
Choose a representative dataset $\mathcal{S}$;
while $|Kernels| > t$ and $perfMetric > \epsilon$ do
    for $s \in \mathcal{S}$ do
        for $i \in Layers$ do
            for $c \in Kernels(i)$ do
                $O^i_c(s) \leftarrow output(i, c, s)$
            end
        end
    end
    $(\alpha, \beta) =_{i,c} (relevance(i, c))$;
    Remove $Kernels(\alpha, \beta)$ from the network;
    if $\sum_{s \in S} O^\alpha_\beta(s) \neq 0$ then
        Retrain a few epochs to adapt model;
    end
end
```

[1]this is actually the case for 1D convolutional layers, but the extension to 2D layers is straightforward

$Kernels(i)$ is the subset of kernels at layer i and $Kernels(i,c)$ points to a specific kernel. The value $output(i,c,s)$ refers to the output of the activation function after $Kernels(i,c)$ for data point s.

The network reduction stops when either the number of kernels left is below a given threshold t or the performance of the network (measured by the function $perfMetric$) is less than minimum acceptable value ϵ. The actual characteristic measured by $perfMetric$ could change according to the application scenario. For instance it could be a function of the average loss with respect to a validation set or a specific metric over the confusion matrix.

During its main iteration the ReFT algorithm selects the least relevant kernel, removes it from the network and performs a partial retrain if the removed kernel was not negligible (i.e. exhibiting an output value of 0 for all the samples in $\mathcal{S}$.

Of course, key to the effectiveness of the ReFT reduction is the choice of a suitable $relevance$ function. As anticipated when introducing the method, the main idea is to account for the output distribution of the kernel with respect to real data, rather than for its input weight (which is much more common in literature). The rationale is to look at the *actual* effect of the kernel, rather than at its *potential* impact expressed in an implicit way by its input weights. To this end, we define a $span$ function as:

$$span(i,c,\gamma) = q_{100-\gamma}(O_c^i) - q_\gamma(O_c^i)$$

That is the difference between percentile $100-\gamma$ and percentile γ in the distribution O_c^i over all the data points s in the dataset $\mathcal{S}$. Thus, we have that $span(i,c,0)$ represents the distance between the maximum and the minimum output for $Filters(i,c)$ over the dataset $\mathcal{S}$.

In principle, the $span$ function could be directly adopted to define relevance by choosing a specific value for γ. However, in order to mitigate the effect of outliers, we would like to use the full span only to disambiguate between kernels with similar output distributions, while adopting more conservative percentiles most of the time.

To obtain this result, we define relevance as an implicit metric by defining this pairwise partial ordering function:

$$\begin{aligned} relevance(j,k) < relevance(l,m) &\iff \\ span(j,k,2) < span(l,m,2) &\quad \vee \\ span(j,k,2) = span(l,m,2) &\quad \wedge \\ span(j,k,0) < span(l,m,0)) & \end{aligned}$$

In practice, this means that filter j,k is less relevant than filter n,m if the distance between the 98^{th} and the 2^{nd} percentile is lower or, if they are equal, the full span is lower. While it could seem counter-intuitive that the 98^{th} and the 2^{nd} can be frequently equal, this actually happens a lot due to the behaviour of clamping functions such as ReLU.

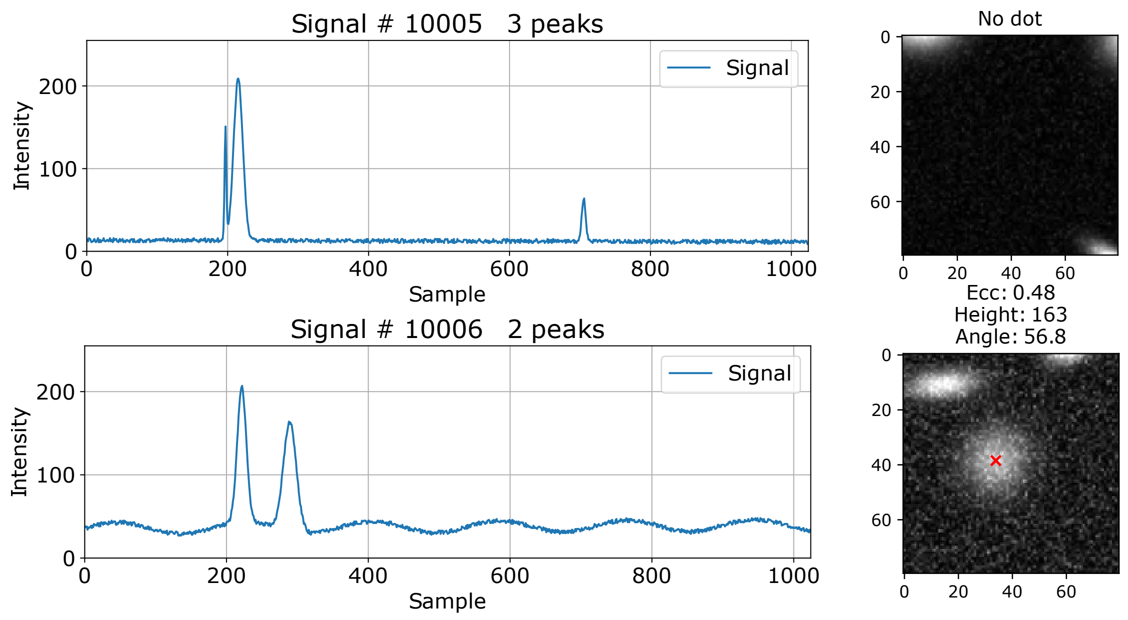


Figure 5.1: Left: two scanlines acquired by the camera for the peak detection task. Multiple or no peaks can be present in the same scanline, sometimes very close to each other. Right: two samples from the *Dots* dataset. Also in this case there might not be any dot present or multiple ones. In the bottom right picture the detected dot center is highlighted by a red cross

5.3 Applications and Experimental Evaluation

In order to have a fair comparison, we selected *property importance* methods focusing on the L1-norm of the filters [114], rank of the feature maps (HRank [68]), and layers' outputs like Apoz [75] and GM [118]. We first analyse the performances of the proposed technique (ReFT) for specific camera tasks, designed to be carried out in relatively small embedded devices. After that, we apply our pruning to VGG16 on Cifar10 to show that the proposed method is effective also when applied to more complex architectures. To assess the validity of our pruning method we also applied it to compress the network we trained to solve the fruit classification task.

5.3.1 1D and 2D Peak Detection Task

As already discussed, image processing for quality inspection in an industrial environment often involves strict requirements. For this reason, smart cameras that can pre-process frames during acquisition (for example by feeding images to a built-in CNN) may offer a substantial advantage over classical solutions. Usually, such setups require extreme and specialised network pruning approaches, in order to improve both time and memory efficiency. As a case study, we show two practical applications which significantly benefit

Model	Acc	Pos err	Height err	Params	Flops	Ratio
Peaks	0.999	0.342	0.0149	2.22K	2.412M	1.00
Peaks-ReFT	**0.993**	**0.644**	**0.0471**	0.936K	0.290M	0.42
Peaks-APoZ	0.988	0.717	0.0566	0.885K	0.295M	0.42
Peaks-L1-norm	0.984	0.806	0.0635	0.936K	0.290M	0.42
Peaks-GM	0.965	1.04	0.0784	0.936K	0.290M	0.42
Peaks-ReFT-S	0.989	1.1400	0.0578	0.936K	0.290M	0.42

Table 5.1: Average accuracy, average peak position error, average height error, number of parameters and compression ratio for Peak network with 34 pruned kernels (out of the 40 available distributed across 3 conv layers). Pruning methods have been repeated 20 times. ReFT-S is the training of Peak-ReFT with random weights. See fig.2 for standard deviation.

from the proposed pruning method, especially when implemented on a FPGA device.

The first is realized by a CNN to detect peaks in a one-dimensional light intensity timeserie. This is a typical scenario in 3D reconstruction in which planar laser beams are projected onto the object under study and observed by one or more cameras geometrically calibrated with the laser. The intersection of each laser plane with the object results in a line, usually orthogonal with the pixel arrangement of the linear camera. Therefore, each line produces a spike, or peak, whose position can be easily related with the depth of the object 3D point illuminated by the laser (See for example the signal plotted in the left column of Fig. 5.1). Our tested model is a relatively simple feed-forward Convolutional Neural Network taking in input a vector of 1024 intensity values and producing a vector of N output bins, each one containing peak probability, height, location and width of detected peaks. The network is made of 4 convolutional blocks (interleaved by ReLU activations and maxpooling) and then it splits to compute the different losses.

The second case study is the 2-dimensional extension of the linear peak detection network to detect dots in image data (see the right column of Fig. 5.1 for some examples). The network architecture is essentially equal to the corresponding 1-dimensional case, except for it takes a 2D images as input and 1-dimensional convolutions are replaced with their 2D counterparts.

Both networks provide multiple outputs, namely: the probability of the presence of a peak or dot, the information about peak position, and its height. In the 2D case, we also consider dot area and eccentricity.

Model	Acc	Pos err	Ecc err	Params	Flops	Ratio
Dots	1.000	1.11	0.073	14.2K	10.537M	1.00
Dots-ReFT	**0.999**	**3.21**	**0.073**	10.3K	0.332M	0.73
Dots-APoZ	0.997	3.73	0.137	10.2K	0.711M	0.71
Dots-L1-norm	0.791	5.91	0.153	10.3K	0.332M	0.73
Dots-GM	0.745	5.61	0.158	10.3K	0.332M	0.73
Dots-HRank	0.953	4.57	0.156	10.3K	0.332M	0.73
Dots-ReFT-S	0.994	1.25	0.101	10.3K	0.332M	1.00

Table 5.2: Average accuracy, average dot position error, average eccentricity error, number of parameters and compression ratio with 34 pruned kernels (out of 40) in the first three convolutional layers. Pruning methods have been repeated 20 times. Dot-ReFT-S is the training of Dot-ReFT with random weights. See fig.2 for standard deviation.

Synthetic datasets

Our *Peaks* dataset contains 100000 line scan acquisitions divided into training, validation and test set with ratio 80:10:10. Intensity ranges from 0 to 255 in vectors of 1024 values representing a line-camera image. *Dots* dataset was generated by approximating the intensity response with a bivariate Normal function characterized by a certain height (the intensity of the dot), eccentricity (how much the dot deviates from being circular) and angle (the major axis orientation). Signals are assumed to be acquired 80 scanlines at a time and multiple dots can be simultaneously present in each image. It consists of 100000 acquisitions with separated test and validation set of 10000 samples each.

Network training and pruning

Both the networks were trained with a learning rate (*lr*) of $1E-5$, decreased by a factor of 0.1 if the loss does not improve for 3 consecutive epochs. The validation set is used to stop before overfitting. Fine tuning after pruning is performed with $lr = 1E-5$ for a fixed number of epochs = 3. We pruned both the *Peaks* and *Dots* networks only from the first 3 blocks, which contain 16, 16 and 8 kernels respectively, leaving at least 1 kernel per block. The architecture of the network for Peaks detection can be see in figure 5.2.

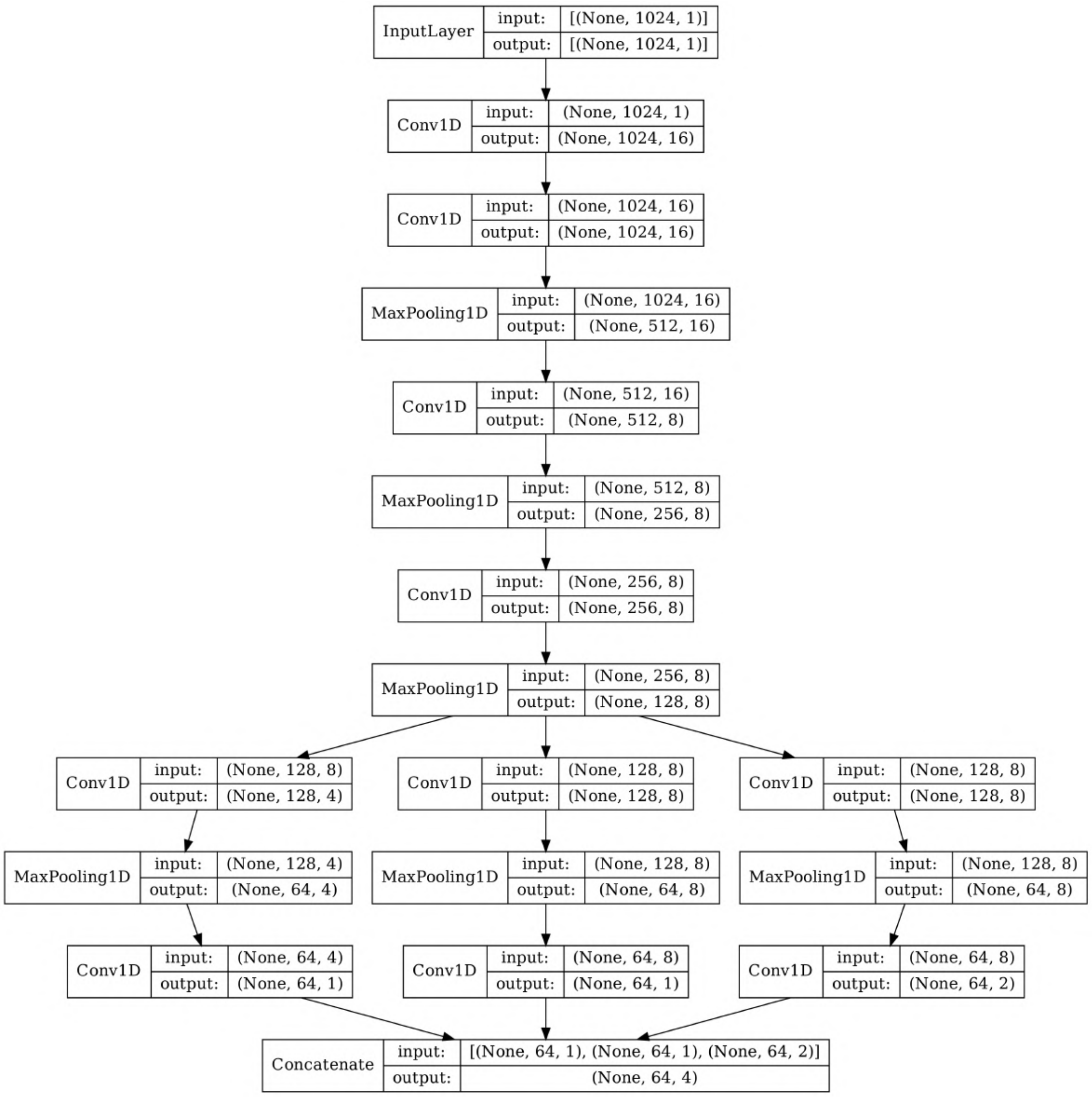

InputLayer input: [(None, 1024, 1)] output: [(None, 1024, 1)]
Conv1D input: (None, 1024, 1) output: (None, 1024, 16)
Conv1D input: (None, 1024, 16) output: (None, 1024, 16)
MaxPooling1D input: (None, 1024, 16) output: (None, 512, 16)
Conv1D input: (None, 512, 16) output: (None, 512, 8)
MaxPooling1D input: (None, 512, 8) output: (None, 256, 8)
Conv1D input: (None, 256, 8) output: (None, 256, 8)
MaxPooling1D input: (None, 256, 8) output: (None, 128, 8)
Conv1D input: (None, 128, 8) output: (None, 128, 4)
Conv1D input: (None, 128, 8) output: (None, 128, 8)
Conv1D input: (None, 128, 8) output: (None, 128, 8)
MaxPooling1D input: (None, 128, 4) output: (None, 64, 4)
MaxPooling1D input: (None, 128, 8) output: (None, 64, 8)
MaxPooling1D input: (None, 128, 8) output: (None, 64, 8)
Conv1D input: (None, 64, 4) output: (None, 64, 1)
Conv1D input: (None, 64, 8) output: (None, 64, 1)
Conv1D input: (None, 64, 8) output: (None, 64, 2)
Concatenate input: [(None, 64, 1), (None, 64, 1), (None, 64, 2)] output: (None, 64, 4)

Figure 5.2: Architecture of the network for Peaks detection. The model used for the 2D case of study, or Dots detection, has a very similar structure.

Results

Figure 5.3 shows how the performance of the 1D peak detection (top) and 2D dot detection (bottom) networks are affected by the pruning process. Our method lead to a good prediction accuracy even if a severe number of kernels are removed. L1-norm tend to perform worst, with the accuracy dropping significantly especially for the 2D dot detection network. Together with the detection accuracy, position and height are better estimated when the network is pruned with our proposed method. Apoz shows similar performance

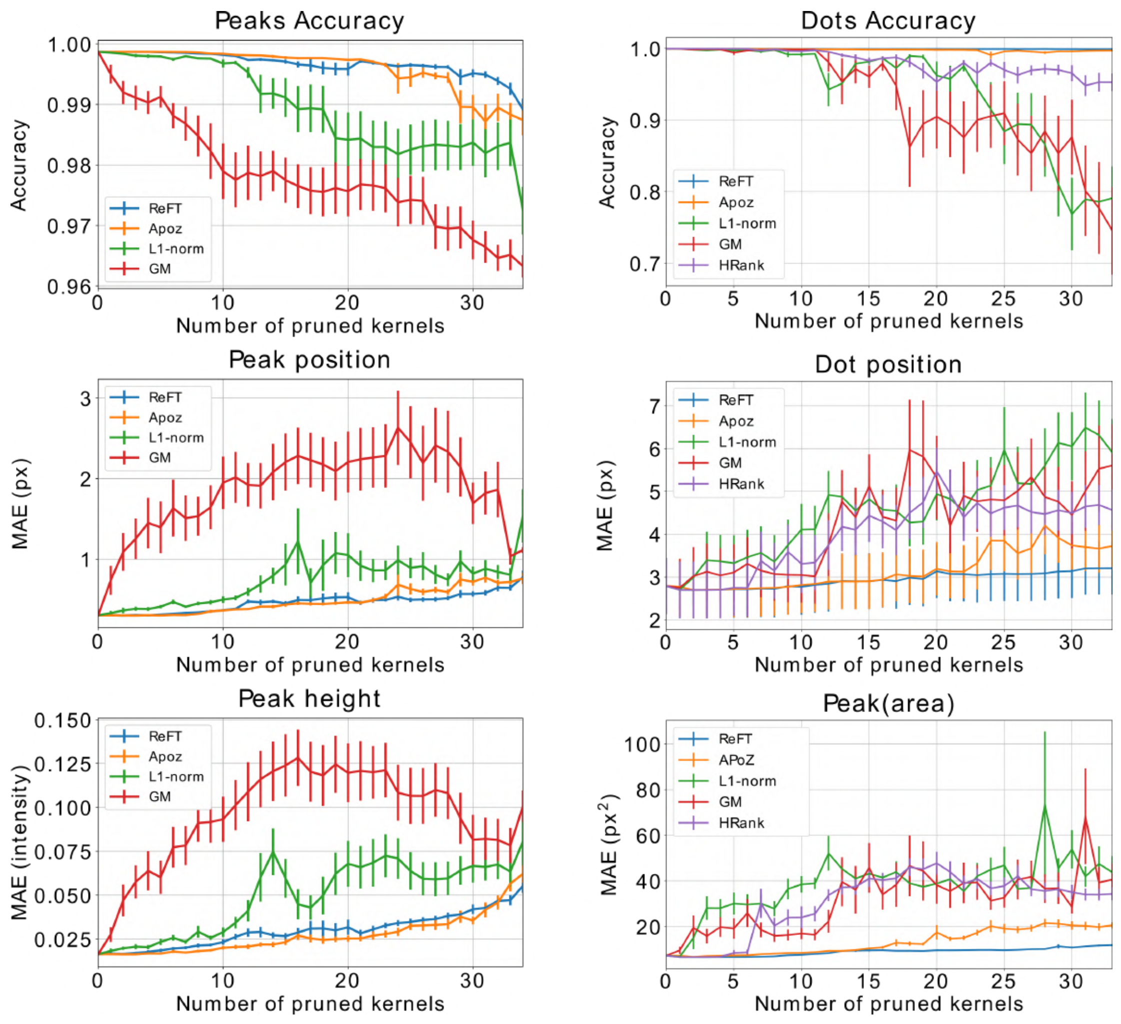


Figure 5.3: Comparison of different pruning techniques for peak and dot detection networks (1st and 2nd columns respectively). For each task we report accuracy, MAE of position error and MAE of height and area.

but higher standard deviation.

Table 5.1 and 5.2 summarize the performance of the two networks while pruning an optimal number of kernels (33 in this case). It is interesting to notice that both the models pruned with our method (Peak-ReFT and Dots-ReFT) perform better than a model with the same number of kernels trained from scratch (Peak-ReFT-S and Dots-ReFT-S). In other words, it is better to trim a complex network than training a simpler network from the start.

VGG16 on Cifar10

To demonstrate that our method is valid also on more complex CNNs we used VGG16 [178] network on Cifar10 dataset [100]. Cifar10 contains 60000 RGB images belonging to 10 classes and it is often used together with VGG16 to assess the efficiency of compressing methods.

VGG16 is a large convolutional network trained on the ILSVR2012 dataset. The network contains 13 convolutional layers with 3 fully connected on top and the activation function for each layer is a ReLU, except for the final softmax. In order to perform classification on a dataset different from the one the network has been trained on, we need to use *transfer learning*. We exploited the pretrained 13 convolutional layers, to which we attached two more fully connected layers with a small dropout rate and ReLU activations. Finally, we added one last fully connected layer to reduce the output to the correct number of classes.

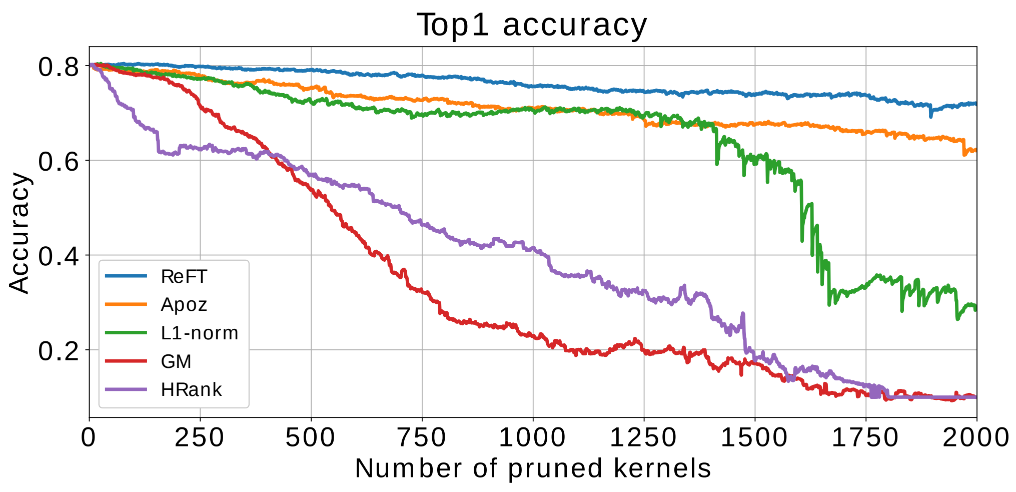
The images are resized from their 32×32 original shape to 48×48, which is the smallest input size required to be able to use all VGG16 convolutional layers. We split the images into 3 separated datasets for train, validation and test with a 60:20:20 ratio.

A fast training with learning rate $3E-5$ is performed on Cifar10 dataset, using the validation set to stop the training at a proper time. Every time the pruning algorithm removes a filter we performed a fine tuning step with 5 epochs and a small learning rate set to $1E-6$, followed by another 5 epochs with the same learning rate divided by 10, to recover any loss in accuracy. We tested HRank with the same parameters except for one during the recovery stage, as the pruned kernels are removed in a single pass, and the authors suggest a higher number of epochs (50) to recover the accuracy. The convolutional layers are frozen both at training and pruning time, such that the network can only modify the fully connected weights.

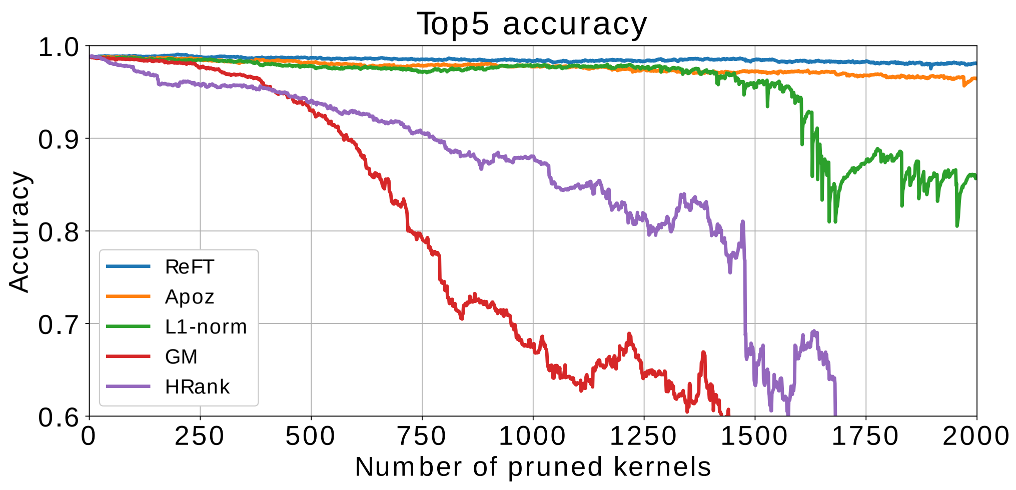
Figure 5.4 shows the top1 accuracy varying the number of pruned kernels. Our method outperforms the others, with a particularly remarkable difference when more than 1500 kernels are removed. Even though ReFT algorithm achieved fair superior performances, some credit has to be given to the other methods. In general, methods that are limited to compute metrics on the filter values (like L1-norm and GM) perform worse than ones analyzing the output of the layers (our method and Apoz). However, the latter are more computational intensive and require a representative dataset to be used, something particularly noticeable when working with images on large networks, such as Cifar10 on VGG16. HRank mitigated this problem by requiring only a small set of images to compute the output responses, but in our tests consistently performed worse than others. HRank, anyway, needs to be run only once, compensating the increased computing time during the pruning phase with the speed-up at inference time with superior performances.

5.3.2 Embedded Vision for Fruit Classification

I spent several months on searching the best architectures to classify the Edremit dataset. At the end of this year, after working extensively on pruning and studying various com-



(a) Top1 accuracy of VGG16 on Cifar10 with different pruning methods.



(b) Top5 accuracy of VGG16 on Cifar10 with different pruning methods.

Figure 5.4: Accuracy loss as the number of pruned filter increases for the different methods analysed. The speed in performance degradation is connect to the experimental setup, as the fine-tuning phase does not take into consideration the validation loss for a proper stopping condition, but rather it runs for a fixed number of epochs.

pression algorithms, I also wondered what would have happened if I applied such techniques on the olives networks. Some of the networks I trained during my PhD for this task are shown in figure 5.5 and 5.6. In particular, I wondered how the performance of our ReFT pruning algorithm would change if a different percentile was take into consideration. Figure 5.7 shows how the accuracy is affected by changing such parameter, while figure 5.8 compares our pruning algorithm on the olives dataset with other methods.

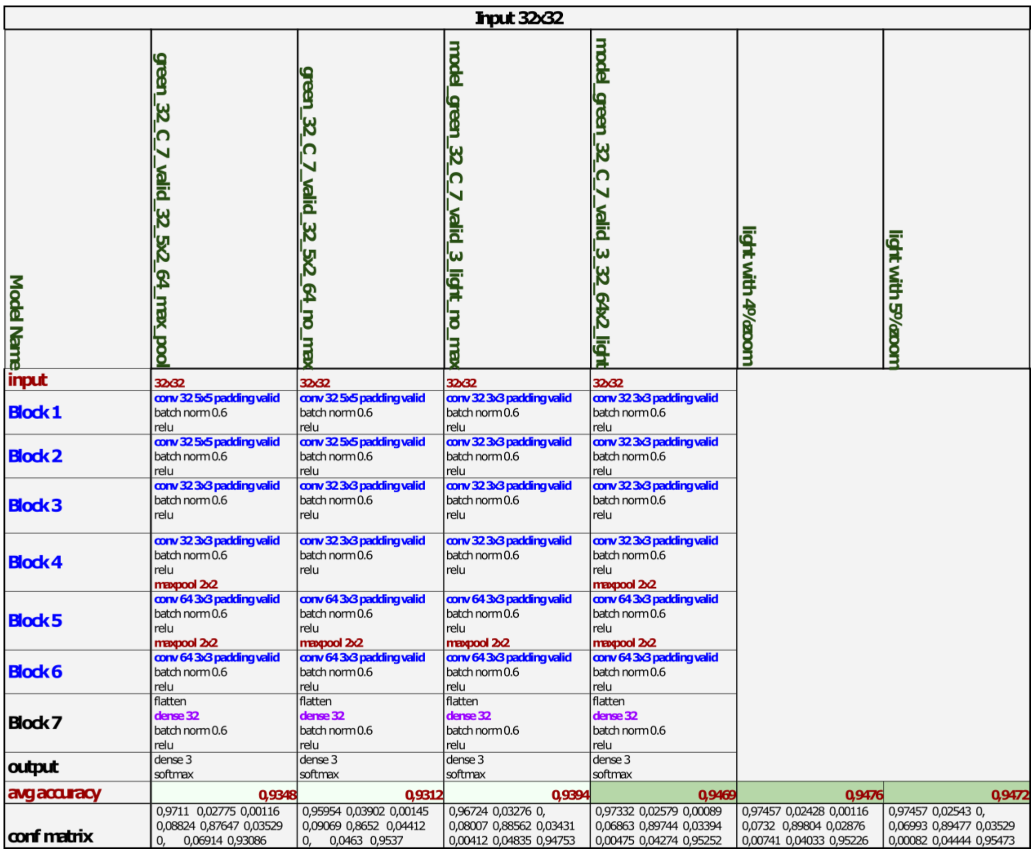
Another interesting experiment was to try knowledge distillation to train a smaller network. The models used have three convolutional blocks made of 3×3 filters, followed

	48x48 Input					
Model Name	green_48_C_7_valid_16_3	green_48_C_7_valid_16_5	green_48_C_7_valid_32_5	green_48_C_7_valid_32_5x2	green_48_C_7_valid_32_5x2_64	green_48_C_7_valid_all64_5x2
input	48x48	48x48	48x48	48x48	48x48	48x48
Block 1	conv 16 3x3 padding valid batch norm 0.6 relu	conv 16 5x5 padding valid batch norm 0.6 relu	conv 32 5x5 padding valid batch norm 0.6 relu	conv 32 5x5 padding valid batch norm 0.6 relu	conv 32 5x5 padding valid batch norm 0.6 relu	conv 64 5x5 padding valid batch norm 0.6 relu
Block 2	conv 16 3x3 padding valid batch norm 0.6 relu	conv 16 3x3 padding valid batch norm 0.6 relu	conv 32 3x3 padding valid batch norm 0.6 relu	conv 32 5x5 padding valid batch norm 0.6 relu	conv 32 5x5 padding valid batch norm 0.6 relu	conv 64 5x5 padding valid batch norm 0.6 relu
Block 3	conv 32 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 64 3x3 padding valid batch norm 0.6 relu maxpool 2x2
Block 4	conv 32 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 64 3x3 padding valid batch norm 0.6 relu maxpool 2x2
Block 5	conv 32 3x3 padding valid batch norm 0.6 relu	conv 32 3x3 padding valid batch norm 0.6 relu	conv 32 3x3 padding valid batch norm 0.6 relu	conv 32 3x3 padding valid batch norm 0.6 relu	conv 64 3x3 padding valid batch norm 0.6 relu	conv 64 3x3 padding valid batch norm 0.6 relu
Block 6	conv 32 3x3 padding valid batch norm 0.6 relu	conv 32 3x3 padding valid batch norm 0.6 relu	conv 32 3x3 padding valid batch norm 0.6 relu	conv 32 3x3 padding valid batch norm 0.6 relu	conv 64 3x3 padding valid batch norm 0.6 relu	conv 64 3x3 padding valid batch norm 0.6 relu
Block 7	flatten dense 16 batch norm 0.6 relu	flatten dense 16 batch norm 0.6 relu	flatten dense 32 batch norm 0.6 relu	flatten dense 32 batch norm 0.6 relu	flatten dense 32 batch norm 0.6 relu	flatten dense 32 batch norm 0.6 relu
output	dense 3 softmax	dense 3 softmax	dense 3 softmax	dense 3 softmax	dense 3 softmax	dense 3 softmax
avg accuracy	92,9	94,9	94	95,1	92,7	94,9
conf matrix	0,94444 0,05556 0, 0,04918 0,91803 0,03279 0, 0,01493 0,98507	0,95833 0,04167 0, 0,03279 0,96721 0, 0, 0,01493 0,98507	0,9717 0,0283 0, 0,07353 0,92647 0, 0,01681 0,05042 0,93277	0,95833 0,04167 0, 0,01639 0,98361 0, 0, 0, 1,	0,95833 0,04167 0, 0,04918 0,95082 0, 0, 0, 1,	0,97222 0,02778 0, 0,03279 0,96721 0, 0, 0, 1,

	48x48 Input					
Model Name	green_48_C_7_same_16_3	green_48_C_7_same_16_5	green_48_C_7_same_32_5	green_48_C_7_same_32_5x2	green_48_C_7_same_32_5x2_64	green_48_C_7_same_all64_5x2
input	48x48	48x48	48x48	48x48	48x48	48x48
Block 1	conv 16 3x3 padding same batch norm 0.6 relu	conv 16 5x5 padding same batch norm 0.6 relu	conv 32 5x5 padding same batch norm 0.6 relu	conv 32 5x5 padding same batch norm 0.6 relu	conv 32 5x5 padding same batch norm 0.6 relu	conv 64 5x5 padding same batch norm 0.6 relu
Block 2	conv 16 3x3 padding same batch norm 0.6 relu	conv 16 3x3 padding same batch norm 0.6 relu	conv 32 3x3 padding same batch norm 0.6 relu	conv 32 5x5 padding same batch norm 0.6 relu	conv 32 5x5 padding same batch norm 0.6 relu	conv 64 5x5 padding same batch norm 0.6 relu
Block 3	conv 32 3x3 padding same batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding same batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding same batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding same batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding same batch norm 0.6 relu maxpool 2x2	conv 64 3x3 padding same batch norm 0.6 relu maxpool 2x2
Block 4	conv 32 3x3 padding same batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding same batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding same batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding same batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding same batch norm 0.6 relu maxpool 2x2	conv 64 3x3 padding same batch norm 0.6 relu maxpool 2x2
Block 5	conv 32 3x3 padding same batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding same batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding same batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding same batch norm 0.6 relu maxpool 2x2	conv 64 3x3 padding same batch norm 0.6 relu maxpool 2x2	conv 64 3x3 padding same batch norm 0.6 relu maxpool 2x2
Block 6	conv 32 3x3 padding same batch norm 0.6 relu	conv 32 3x3 padding same batch norm 0.6 relu	conv 32 3x3 padding same batch norm 0.6 relu	conv 32 3x3 padding same batch norm 0.6 relu	conv 64 3x3 padding same batch norm 0.6 relu	conv 64 3x3 padding same batch norm 0.6 relu
Block 7	flatten dense 16 batch norm 0.6 relu	flatten dense 16 batch norm 0.6 relu	flatten dense 32 batch norm 0.6 relu	flatten dense 32 batch norm 0.6 relu	flatten dense 32 batch norm 0.6 relu	flatten dense 32 batch norm 0.6 relu
output	dense 3 softmax	dense 3 softmax	dense 3 softmax	dense 3 softmax	dense 3 softmax	dense 3 softmax
avg accuracy	95,1	94	94	96,4	95,3	95,7
conf matrix	0,94444 0,05556 0, 0, 1, 0, 0, 0, 1,	0,97222 0,02778 0, 0, 1, 0, 0, 0, 1,	0,95833 0,04167 0, 0,01639 0,98361 0, 0, 0,01493 0,98507	0,95833 0,04167 0, 0,01639 0,98361 0, 0, 0, 1,	0,94444 0,05556 0, 0,01639 0,98361 0, 0, 0, 1,	0,95833 0,04167 0, 0,01639 0,98361 0, 0, 0, 1,

	48x48 Input	
Model Name	green_48_C_7_valid_32_5x2_64_max_pool	green_48_C_7_valid_32_5x2_64_no_max
input	48x48	48x48
Block 1	conv 32 5x5 padding valid batch norm 0.6 relu	conv 32 5x5 padding valid batch norm 0.6 relu
Block 2	conv 32 5x5 padding valid batch norm 0.6 relu	conv 32 5x5 padding valid batch norm 0.6 relu
Block 3	conv 32 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding valid batch norm 0.6 relu
Block 4	conv 32 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding valid batch norm 0.6 relu maxpool 2x2
Block 5	conv 64 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 64 3x3 padding valid batch norm 0.6 relu maxpool 2x2
Block 6	conv 64 3x3 padding valid batch norm 0.6 relu	conv 64 3x3 padding valid batch norm 0.6 relu
Block 7	flatten dense 32 batch norm 0.6 relu	flatten dense 32 batch norm 0.6 relu
output	dense 3 softmax	dense 3 softmax
avg accuracy	0,9505	0,9548
conf matrix	0,98266 0,01629 0,00105 0,09091 0,89037 0,01872 0,00337 0,03928 0,95735	0,97635 0,02207 0,00158 0,07576 0,90553 0,01872 0,00449 0,02469 0,97082

Figure 5.5: Neural networks trained on the Edremit dataset with input size equal to 48x48 and transformed into grayscale images. Different hyperparameters like padding (valid/same), the type of pooling (average/max), the size of the convolutional window and the number of filters in the convolutional layers affects the achieved accuracy. All the trainings have been using Adam optimizer, with a 0.001 learning rate and data augmentation (flip, rotation and shift).



Input 32x32						
Model Name	green_32_C_7_valid_32_5x2_64_max_pool	green_32_C_7_valid_32_5x2_64_no_max	model_green_32_C_7_valid_3_light_no_max	model_green_32_C_7_valid_3_32_64x2_light	light with 4%zoom	light with 5%zoom
input	32x32	32x32	32x32	32x32		
Block 1	conv 32 5x5 padding valid batch norm 0.6 relu	conv 32 5x5 padding valid batch norm 0.6 relu	conv 32 3x3 padding valid batch norm 0.6 relu	conv 32 3x3 padding valid batch norm 0.6 relu		
Block 2	conv 32 5x5 padding valid batch norm 0.6 relu	conv 32 5x5 padding valid batch norm 0.6 relu	conv 32 3x3 padding valid batch norm 0.6 relu	conv 32 3x3 padding valid batch norm 0.6 relu		
Block 3	conv 32 3x3 padding valid batch norm 0.6 relu	conv 32 3x3 padding valid batch norm 0.6 relu	conv 32 3x3 padding valid batch norm 0.6 relu	conv 32 3x3 padding valid batch norm 0.6 relu		
Block 4	conv 32 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 32 3x3 padding valid batch norm 0.6 relu	conv 32 3x3 padding valid batch norm 0.6 relu	conv 32 3x3 padding valid batch norm 0.6 relu maxpool 2x2		
Block 5	conv 64 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 64 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 64 3x3 padding valid batch norm 0.6 relu maxpool 2x2	conv 64 3x3 padding valid batch norm 0.6 relu maxpool 2x2		
Block 6	conv 64 3x3 padding valid batch norm 0.6 relu	conv 64 3x3 padding valid batch norm 0.6 relu	conv 64 3x3 padding valid batch norm 0.6 relu	conv 64 3x3 padding valid batch norm 0.6 relu		
Block 7	flatten dense 32 batch norm 0.6 relu	flatten dense 32 batch norm 0.6 relu	flatten dense 32 batch norm 0.6 relu	flatten dense 32 batch norm 0.6 relu		
output	dense 3 softmax	dense 3 softmax	dense 3 softmax	dense 3 softmax		
avg accuracy	0,9348	0,9312	0,9394	0,9469	0,9476	0,9472
conf matrix	0,9711 0,02775 0,00116 0,08824 0,87647 0,03529 0, 0,06914 0,93086	0,95954 0,03902 0,00145 0,09069 0,8652 0,04412 0, 0,0463 0,9537	0,96724 0,03276 0, 0,08007 0,88562 0,03431 0,00412 0,04835 0,94753	0,97332 0,02579 0,00089 0,06863 0,89744 0,03394 0,00475 0,04274 0,95252	0,97457 0,02428 0,00116 0,0732 0,89804 0,02876 0,00741 0,04033 0,95226	0,97457 0,02543 0, 0,06993 0,89477 0,03529 0,00082 0,04444 0,95473

Figure 5.6: Similar to figure 5.5, in this figure we can see the networks trained with a smaller (32x32) input. Resizing the images allows to significantly reduce the number of network parameters and their related FLOPs. The last two experiments use the last architecture, but add another transformation to data augmentation, by slightly zooming in and out the inputs.

Model	Top1	Top5	Params	Flops	Ratio
VGG16	0.801	0.988	15.1M	1.41G	1.000
VGG16-ReFT	**0.780**	**0.986**	11.9M	1.15G	0.789
VGG16-APoZ	0.761	0.983	11.5M	1.23G	0.762
VGG16-L1-norm	0.731	0.979	11.9M	1.15G	0.789
VGG16-GM	0.581	0.944	11.9M	1.15G	0.789

Table 5.3: Comparison of top1 accuracy, top5 accuracy, number of parameters, FLOPs and compression ratio achieved by filter pruning on VGG16 and Cifar10 after pruning 460 filters.

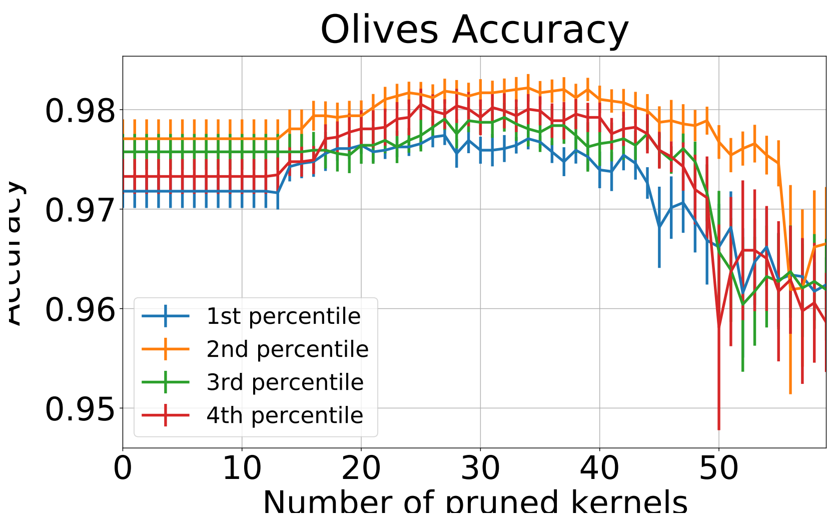


Figure 5.7: Applying ReFT on the network trained for olives classification. Different values for the percentile are used. Notice how pruning actually initially increases the overall accuracy.

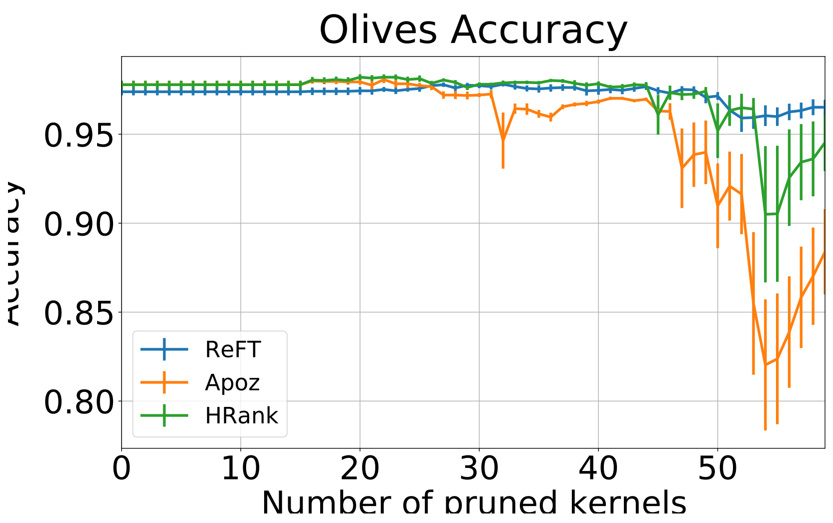


Figure 5.8: Comparison of different pruning methods on the classification model. Pruning 50 kernels out of 96 allows to compress the network by over 50%.

by a flattening layer and a dense layer (with three nodes). A ReLU activation and a maxpooling operation follow after every convolutional layer. The only difference between the teacher and the student models was the number of filters used in the convolutional layers, 64, 32 and 32 for the teacher and 32, 16 and 16 for the student. Table 5.4 shows the results.

Model	Epochs	Accuracy	Params	Flops
Teacher	10	0.98856	34.5K	12.2M
Student	5	**0.990325**	10.3K	3.44M
Student without KD	5	0.97185	10.3K	3.44M

Table 5.4: Using Knowledge Distillation on two simple networks to classify the olives. The temperature is set to 10.

Even if the experiment using Knowledge Distillation showed promising results, few consideration has to be taken into account. Firstly, increasing the number of epochs actually allows the small network to reach comparable accuracy. Secondly, many attempts have to be made to find two architectures (teacher and student) that actually allow the KD algorithm to outperform the small network trained without any help.

5.4 Conclusions

Filter pruning is a powerful tool to reduce the number of parameters of a network, not only to exploit deep learning in embedded devices, but also to optimize the model both in terms of classification speed and memory required.

We presented a simple CNN pruning method working by ranking the relevance of each convolutional kernel according to the output produced on a representative dataset. Our heuristic is simple to implement but it better preserves the network predicting power compared to similar state-of-the-art approaches while pruning a large amount of kernels.

We analysed two practical case study: peak detection in both 1 and 2 dimensional intensity signals to assess the feasibility on simple networks designed for FPGA hardware. In our tests, we almost halved the number of weights without a noticeable decrease of prediction accuracy. Even if developed to implement low-level vision tasks, our method has proven to be effective even when applied on classical datasets and network architectures like VGG16 on Cifar10.

To show the goodness of our filter pruning technique, we also applied it to some of the networks developed to solve the fruit classification task. With our method we were able to remove a considerable amount of network parameters, while keeping or even improving the initial accuracy.

6 Conclusions and Future Work

In this thesis we have seen how Neural Networks, and in particular Convolutional Neural Networks, can contribute in industrial applications. After providing a background on Neural Networks, the main focus was to present the available literature that can be exploited during the development of applications in industrial environment. Online algorithms with small inference time require small networks, while at the same time there is a low tolerance for any performance drop. The optimal way to address such problem using deep learning is to deploy small specialized networks, trained on the available data. Compression techniques also greatly help to address such challenges, by reducing the size of a network without affecting its accuracy. Another problem concerns the scarcity of the available data, as Neural Networks require at least hundreds of images during the training to converge into an optimal model. The last part of this thesis contributes in solving such problems by proposing two novel methods, the first exploiting data augmentation and the second performing a filter pruning algorithm. Such methods have actually been applied in real-case scenarios: fruit classification and 1D/2D signal detection for cameras, aimed to be implemented on FPGA. Working on tasks and data that have an actual application in real life has been stimulating and satisfying. The drawback is that developing models working on industrial machines does not always travel in parallel with the research field. Several projects I worked on during my PhD have not been presented in this thesis, as most of the times they simply involved to implement already existing techniques. Other times, instead, the models I developed could not be published because covered by industrial secrecy. Future work scenarios include further exploring existing compression algorithms, and especially trying them on the olives dataset. For what concerns the camera tasks, instead, I would like to train other networks to solve different problems (*e.g.*keypoint detection), and implement some quantization techniques to further compress our models. Finally, I would like to actually transfer the trained networks on FPGA and develop smart cameras.

Bibliography

[1] ABDELJABER, O., AVCI, O., KIRANYAZ, M. S., BOASHASH, B., SODANO, H., AND INMAN, D. J. 1-d cnns for structural damage detection: Verification on a structural health monitoring benchmark data. *Neurocomputing 275* (2018), 1308–1317.

[2] ABDELJABER, O., AVCI, O., KIRANYAZ, S., GABBOUJ, M., AND INMAN, D. J. Real-time vibration-based structural damage detection using one-dimensional convolutional neural networks. *Journal of Sound and Vibration 388* (2017), 154–170.

[3] AIOLLI, F. A preliminary study on a recommender system for the million songs dataset challenge. In *IIR* (2013), Citeseer, pp. 73–83.

[4] AKTER, Y. A., AND RAHMAN, M. O. Development of a computer vision based eggplant grading system. In *2017 4th International Conference on Advances in Electrical Engineering (ICAEE)* (2017), IEEE, pp. 285–290.

[5] AVCI, O., ABDELJABER, O., KIRANYAZ, S., HUSSEIN, M., AND INMAN, D. J. Wireless and real-time structural damage detection: A novel decentralized method for wireless sensor networks. *Journal of Sound and Vibration 424* (2018), 158–172.

[6] AVCI, O., ABDELJABER, O., KIRANYAZ, S., AND INMAN, D. Structural damage detection in real time: implementation of 1d convolutional neural networks for shm applications. In *Structural Health Monitoring & Damage Detection, Volume 7*. Springer, 2017, pp. 49–54.

[7] BENGIO, Y., COURVILLE, A., AND VINCENT, P. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence 35*, 8 (2013), 1798–1828.

[8] BIANCO, S., CADENE, R., CELONA, L., AND NAPOLETANO, P. Benchmark analysis of representative deep neural network architectures. *IEEE Access 6* (2018), 64270–64277.

[9] BIREM, M., AND BERRY, F. Dreamcam: A modular fpga-based smart camera architecture. *Journal of Systems Architecture 60*, 6 (2014), 519–527.

[10] BLALOCK, D., ORTIZ, J. J. G., FRANKLE, J., AND GUTTAG, J. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033* (2020).

[11] BOLLE, R. M., CONNELL, J. H., HAAS, N., MOHAN, R., AND TAUBIN, G. Veggievision: A produce recognition system. In *Proceedings Third IEEE Workshop on Applications of Computer Vision. WACV'96* (1996), IEEE, pp. 244–251.

[12] BUCILUǍ, C., CARUANA, R., AND NICULESCU-MIZIL, A. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (2006), pp. 535–541.

[13] CAO, S., LIU, Y., ZHOU, C., SUN, Q., PONGSAK, L., AND SHEN, S. M. Thinnet: An efficient convolutional neural network for object detection. In *2018 24th International Conference on Pattern Recognition (ICPR)* (2018), pp. 836–841.

[14] CARROLL, J. D., AND CHANG, J.-J. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika 35*, 3 (1970), 283–319.

[15] CARROLL, J. D., PRUZANSKY, S., AND KRUSKAL, J. B. Candelinc: A general approach to multidimensional analysis of many-way arrays with linear constraints on parameters. *Psychometrika 45*, 1 (1980), 3–24.

[16] CAVALLO, D. P., CEFOLA, M., PACE, B., LOGRIECO, A. F., AND ATTOLICO, G. Non-destructive automatic quality evaluation of fresh-cut iceberg lettuce through packaging material. *Journal of Food Engineering 223* (2018), 46–52.

[17] CEN, H., LU, R., ZHU, Q., AND MENDOZA, F. Nondestructive detection of chilling injury in cucumber fruit using hyperspectral imaging with feature selection and supervised classification. *Postharvest Biology and Technology 111* (2016), 352–361.

[18] CHEBOTAR, Y., AND WATERS, A. Distilling knowledge from ensembles of neural networks for speech recognition. In *Interspeech* (2016), pp. 3439–3443.

[19] CHELLAPILLA, K., PURI, S., AND SIMARD, P. High performance convolutional neural networks for document processing. In *Tenth international workshop on frontiers in handwriting recognition* (2006), Suvisoft.

[20] CHEN, D., MEI, J.-P., WANG, C., FENG, Y., AND CHEN, C. Online knowledge distillation with diverse peers. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2020), vol. 34, pp. 3430–3437.

[21] CHEN, D., MEI, J.-P., ZHANG, Y., WANG, C., WANG, Z., FENG, Y., AND CHEN, C. Cross-layer distillation with semantic calibration. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2021), vol. 35, pp. 7028–7036.

[22] CHEN, G., CHOI, W., YU, X., HAN, T., AND CHANDRAKER, M. Learning efficient object detection models with knowledge distillation. *Advances in neural information processing systems 30* (2017).

[23] CHEN, H., WANG, Y., XU, C., YANG, Z., LIU, C., SHI, B., XU, C., XU, C., AND TIAN, Q. Data-free learning of student networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), pp. 3514–3522.

[24] Chen, T., Frankle, J., Chang, S., Liu, S., Zhang, Y., Wang, Z., and Carbin, M. The lottery ticket hypothesis for pre-trained bert networks. *arXiv preprint arXiv:2007.12223* (2020).

[25] Chen, Y., Jiang, H., Li, C., Jia, X., and Ghamisi, P. Deep feature extraction and classification of hyperspectral images based on convolutional neural networks. *IEEE Transactions on Geoscience and Remote Sensing 54*, 10 (2016), 6232–6251.

[26] Cheng, J.-H., Nicolai, B., and Sun, D.-W. Hyperspectral imaging with multivariate analysis for technological parameters prediction and classification of muscle foods: A review. *Meat science 123* (2017), 182–191.

[27] Choudhary, T., Mishra, V., Goswami, A., and Sarangapani, J. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review 53*, 7 (2020), 5113–5155.

[28] Chowdhury, M. T., Alam, M. S., Hasan, M. A., and Khan, M. I. Vegetables detection from the glossary shop for the blind. *IOSR Journal of Electrical and Electronics Engineering 8*, 3 (2013), 43–53.

[29] Conneau, A., Schwenk, H., Barrault, L., and Lecun, Y. Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781* (2016).

[30] Courbariaux, M., Bengio, Y., and David, J.-P. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024* (2014).

[31] Courbariaux, M., Bengio, Y., and David, J.-P. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems* (2015), pp. 3123–3131.

[32] Deng, J., Dong, W., Socher, R., jia Li, L., Li, K., and Fei-fei, L. Imagenet: A large-scale hierarchical image database. In *In CVPR* (2009).

[33] Deng, L., Li, G., Han, S., Shi, L., and Xie, Y. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE 108*, 4 (2020), 485–532.

[34] Denil, M., Shakibi, B., Dinh, L., Ranzato, M., and De Freitas, N. Predicting parameters in deep learning. *arXiv preprint arXiv:1306.0543* (2013).

[35] Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems* (2014), pp. 1269–1277.

[36] Du, J. Understanding of object detection based on cnn family and yolo. In *Journal of Physics: Conference Series* (2018), vol. 1004, IOP Publishing, p. 012029.

[37] Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research 12*, 7 (2011).

[38] Egmont-Petersen, M., de Ridder, D., and Handels, H. Image processing with neural networks—a review. *Pattern recognition 35*, 10 (2002), 2279–2301.

[39] Engelbrecht, A. P. A new pruning heuristic based on variance analysis of sensitivity information. *IEEE transactions on Neural Networks 12*, 6 (2001), 1386–1399.

[40] Frankle, J., and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635* (2018).

[41] Fukuda, T., Suzuki, M., Kurata, G., Thomas, S., Cui, J., and Ramabhadran, B. Efficient knowledge distillation from an ensemble of teachers. In *Interspeech* (2017), pp. 3697–3701.

[42] Gao, M., Wang, Y., and Wan, L. Residual error based knowledge distillation. *Neurocomputing 433* (2021), 154–161.

[43] Gao, S., Cheng, M.-M., Zhao, K., Zhang, X.-Y., Yang, M.-H., and Torr, P. H. Res2net: A new multi-scale backbone architecture. *IEEE transactions on pattern analysis and machine intelligence* (2019).

[44] Garg, I., Panda, P., and Roy, K. A low effort approach to structured cnn design using pca. *IEEE Access 8* (2019), 1347–1360.

[45] Gasparetto, A., Ressi, D., Bergamasco, F., Pistellato, M., Cosmo, L., Boschetti, M., Ursella, E., and Albarelli, A. Cross-dataset data augmentation for convolutional neural networks training. In *2018 24th International Conference on Pattern Recognition (ICPR)* (2018), IEEE, pp. 910–915.

[46] Gila, D. M., Puerto, D. A., Garcia, J. G., and Ortega, J. G. Automatic classification of olives for oil production using computer vision. In *2015 IEEE International Conference on Industrial Technology (ICIT)* (2015), IEEE, pp. 1651–1656.

[47] Girish, S., Maiya, S. R., Gupta, K., Chen, H., Davis, L. S., and Shrivastava, A. The lottery ticket hypothesis for object recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 762–771.

[48] Glorot, X., and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (2010), JMLR Workshop and Conference Proceedings, pp. 249–256.

[49] GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A., AND BENGIO, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680.

[50] GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A., AND BENGIO, Y. Generative adversarial nets. *Advances in neural information processing systems 27* (2014).

[51] GOU, J., YU, B., MAYBANK, S. J., AND TAO, D. Knowledge distillation: A survey. *International Journal of Computer Vision 129*, 6 (2021), 1789–1819.

[52] GU, J., AND TRESP, V. Search for better students to learn distilled knowledge. *arXiv preprint arXiv:2001.11612* (2020).

[53] GUO, Q., WANG, X., WU, Y., YU, Z., LIANG, D., HU, X., AND LUO, P. Online knowledge distillation via collaborative learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 11020–11029.

[54] GUO, Y. A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752* (2018).

[55] GUO, Y., YAO, A., AND CHEN, Y. Dynamic network surgery for efficient dnns. *arXiv preprint arXiv:1608.04493* (2016).

[56] HABIB, M. T., MAJUMDER, A., JAKARIA, A., AKTER, M., UDDIN, M. S., AND AHMED, F. Machine vision based papaya disease recognition. *Journal of King Saud University-Computer and Information Sciences 32*, 3 (2020), 300–309.

[57] HAMEED, K., CHAI, D., AND RASSAU, A. A comprehensive review of fruit and vegetable classification techniques. *Image and Vision Computing 80* (2018), 24–44.

[58] HAN, S., MAO, H., AND DALLY, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[59] HAN, S., POOL, J., TRAN, J., AND DALLY, W. J. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626* (2015).

[60] HANSON, S., AND PRATT, L. Comparing biases for minimal network construction with back-propagation. *Advances in neural information processing systems 1* (1988), 177–185.

[61] HARSHMAN, R. A. Models for analysis of asymmetrical relationships among n objects or stimuli. In *First Joint Meeting of the Psychometric Society and the Society of Mathematical Psychology, Hamilton, Ontario, 1978* (1978).

[62] HARSHMAN, R. A., ET AL. Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis.

[63] HARSHMAN, R. A., AND LUNDY, M. E. Uniqueness proof for a family of models sharing features of tucker's three-mode factor analysis and parafac/candecomp. *Psychometrika 61*, 1 (1996), 133–154.

[64] HASSIBI, B., AND STORK, D. G. *Second order derivatives for network pruning: Optimal brain surgeon.* Morgan Kaufmann, 1993.

[65] HE, K., ZHANG, X., REN, S., AND SUN, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (2015), pp. 1026–1034.

[66] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2016), pp. 770–778.

[67] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[68] HE, Y., LIU, P., WANG, Z., HU, Z., AND YANG, Y. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), pp. 4340–4349.

[69] HINTON, G. Coursera neural networks for machine learning lecture 6, 2018.

[70] HINTON, G., VINYALS, O., AND DEAN, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[71] HONG, D., KOLDA, T. G., AND DUERSCH, J. A. Generalized canonical polyadic tensor decomposition. *SIAM Review 62*, 1 (2020), 133–163.

[72] HOSSAIN, M. S., MUHAMMAD, G., AND AMIN, S. U. Improving consumer satisfaction in smart cities using edge computing and caching: A case study of date fruits classification. *Future Generation Computer Systems 88* (2018), 333–341.

[73] HOWARD, A., SANDLER, M., CHU, G., CHEN, L.-C., CHEN, B., TAN, M., WANG, W., ZHU, Y., PANG, R., VASUDEVAN, V., ET AL. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), pp. 1314–1324.

[74] HOWARD, A. G., ZHU, M., CHEN, B., KALENICHENKO, D., WANG, W., WEYAND, T., ANDREETTO, M., AND ADAM, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[75] HU, H., PENG, R., TAI, Y.-W., AND TANG, C.-K. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250* (2016).

[76] HUANG, G., LIU, Z., VAN DER MAATEN, L., AND WEINBERGER, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 4700–4708.

[77] HUANG, Z., AND WANG, N. Like what you like: Knowledge distill via neuron selectivity transfer. *arXiv preprint arXiv:1707.01219* (2017).

[78] HUBARA, I., COURBARIAUX, M., SOUDRY, D., EL-YANIV, R., AND BENGIO, Y. Binarized neural networks. *Advances in neural information processing systems 29* (2016).

[79] IANDOLA, F. N., HAN, S., MOSKEWICZ, M. W., ASHRAF, K., DALLY, W. J., AND KEUTZER, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360* (2016).

[80] INCE, T., KIRANYAZ, S., EREN, L., ASKAR, M., AND GABBOUJ, M. Real-time motor fault detection by 1-d convolutional neural networks. *IEEE Transactions on Industrial Electronics 63*, 11 (2016), 7067–7075.

[81] JADERBERG, M., VEDALDI, A., AND ZISSERMAN, A. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866* (2014).

[82] JAHANSHAHI, A. Tinycnn: A tiny modular cnn accelerator for embedded fpga. *arXiv preprint arXiv:1911.06777* (2019).

[83] JANSSENS, O., SLAVKOVIKJ, V., VERVISCH, B., STOCKMAN, K., LOCCUFIER, M., VERSTOCKT, S., VAN DE WALLE, R., AND VAN HOECKE, S. Convolutional neural network based fault detection for rotating machinery. *Journal of Sound and Vibration 377* (2016), 331–345.

[84] JAWALE, D., AND DESHMUKH, M. Real time automatic bruise detection in (apple) fruits using thermal camera. In *2017 International Conference on Communication and Signal Processing (ICCSP)* (2017), IEEE, pp. 1080–1085.

[85] JIN, J., DUNDAR, A., AND CULURCIELLO, E. Flattened convolutional neural networks for feedforward acceleration. *arXiv preprint arXiv:1412.5474* (2014).

[86] KANAZAWA, S., NODA, A., ITO, A., HASHIMOTO, K., KUNISAWA, A., NAKANISHI, T., KAJIHARA, S., MUKAI, N., IIDA, J., FUKUSAKI, E., ET AL. Fake metabolomics chromatogram generation for facilitating deep learning of peak-picking neural networks. *Journal of Bioscience and Bioengineering 131*, 2 (2021), 207–212.

[87] KAPACH, K., BARNEA, E., MAIRON, R., EDAN, Y., AND BEN-SHAHAR, O. Computer vision for fruit harvesting robots–state of the art and challenges ahead. *International Journal of Computational Vision and Robotics 3*, 1-2 (2012), 4–34.

[88] KARNIN, E. D. A simple procedure for pruning back-propagation trained neural networks. *IEEE transactions on neural networks 1*, 2 (1990), 239–242.

[89] KATANFOROOSH, AND KUNIN. Initializing neural network, deeplearning.ai, 2019.

[90] KESELJ, V. Speech and language processing daniel jurafsky and james h. martin (stanford university and university of colorado at boulder) pearson prentice hall, 2009, xxxi+ 988 pp; hardbound, isbn 978-0-13-187321-6, $115.00, 2009.

[91] KHORASHADIZADEH, H., MONSEFI, R., AND FOOLAD, S. Attention-based convolutional neural network for answer selection using bert. In *2020 8th Iranian Joint Congress on Fuzzy and intelligent Systems (CFIS)* (2020), IEEE, pp. 121–126.

[92] KIM, H., AND SIM, S.-H. Automated peak picking using region-based convolutional neural network for operational modal analysis. *Structural Control and Health Monitoring 26*, 11 (2019), e2436.

[93] KINGMA, D. P., AND BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[94] KIRANYAZ, S., AVCI, O., ABDELJABER, O., INCE, T., GABBOUJ, M., AND INMAN, D. J. 1d convolutional neural networks and applications: A survey. *Mechanical systems and signal processing 151* (2021), 107398.

[95] KIRANYAZ, S., GASTLI, A., BEN-BRAHIM, L., AL-EMADI, N., AND GABBOUJ, M. Real-time fault detection and identification for mmc using 1-d convolutional neural networks. *IEEE Transactions on Industrial Electronics 66*, 11 (2018), 8760–8771.

[96] KIRANYAZ, S., INCE, T., AND GABBOUJ, M. Real-time patient-specific ecg classification by 1-d convolutional neural networks. *IEEE Transactions on Biomedical Engineering 63*, 3 (2015), 664–675.

[97] KIRANYAZ, S., INCE, T., AND GABBOUJ, M. Personalized monitoring and advance warning system for cardiac arrhythmias. *Scientific reports 7*, 1 (2017), 1–8.

[98] KIRANYAZ, S., INCE, T., HAMILA, R., AND GABBOUJ, M. Convolutional neural networks for patient-specific ecg classification. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* (2015), IEEE, pp. 2608–2611.

[99] KOLDA, T. G., AND BADER, B. W. Tensor decompositions and applications. *SIAM review 51*, 3 (2009), 455–500.

[100] KRIZHEVSKY, A., NAIR, V., AND HINTON, G. Cifar-10 (canadian institute for advanced research).

[101] KRIZHEVSKY, A., NAIR, V., AND HINTON, G. Cifar-100 (canadian institute for advanced research).

[102] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems 25* (2012), 1097–1105.

[103] LAM, A., KUNO, Y., AND SATO, I. Evaluating freshness of produce using transfer learning. In *2015 21st Korea-Japan Joint Workshop on Frontiers of Computer Vision (FCV)* (2015), IEEE, pp. 1–4.

[104] LEBEDEV, V., GANIN, Y., RAKHUBA, M., OSELEDETS, I., AND LEMPITSKY, V. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553* (2014).

[105] LECUN, Y., BENGIO, Y., AND HINTON, G. Deep learning. *nature 521*, 7553 (2015), 436–444.

[106] LECUN, Y., AND CORTES, C. MNIST handwritten digit database.

[107] LECUN, Y., DENKER, J. S., AND SOLLA, S. A. Optimal brain damage. In *Advances in neural information processing systems* (1990), pp. 598–605.

[108] LEE, J., KIM, T., PARK, J., AND NAM, J. Raw waveform-based audio classification using sample-level cnn architectures. *arXiv preprint arXiv:1712.00866* (2017).

[109] LEE, K., LEE, K., SHIN, J., AND LEE, H. Overcoming catastrophic forgetting with unlabeled data in the wild. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), pp. 312–321.

[110] LEE, N., AJANTHAN, T., AND TORR, P. H. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340* (2018).

[111] LEE, S., AND SONG, B. C. Graph-based knowledge distillation by multi-head attention network. *arXiv preprint arXiv:1907.02226* (2019).

[112] LEMAIRE, C., ACHKAR, A., AND JODOIN, P.-M. Structured pruning of neural networks with budget-aware regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), pp. 9108–9116.

[113] LI, E., ZENG, L., ZHOU, Z., AND CHEN, X. Edge ai: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications 19*, 1 (2019), 447–457.

[114] LI, H., KADAV, A., DURDANOVIC, I., SAMET, H., AND GRAF, H. P. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* (2016).

[115] LI, Y. Detecting lesion bounding ellipses with gaussian proposal networks. In *International Workshop on Machine Learning in Medical Imaging* (2019), Springer, pp. 337–344.

[116] LIN, H.-H., LI, Z.-Y., SHIH, M.-H., SUN, Y.-N., AND SHEN, T.-L. Pupil localization for ophthalmic diagnosis using anchor ellipse regression. In *2019 16th International Conference on Machine Vision Applications (MVA)* (2019), IEEE, pp. 1–5.

[117] LIN, M., CHEN, Q., AND YAN, S. Network in network. *arXiv preprint arXiv:1312.4400* (2013).

[118] LIN, M., JI, R., WANG, Y., ZHANG, Y., ZHANG, B., TIAN, Y., AND SHAO, L. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 1529–1538.

[119] LIN, P.-W., AND HSU, C.-M. Lightweight convolutional neural networks with model-switching architecture for multi-scenario road semantic segmentation. *Applied Sciences 11*, 16 (2021), 7424.

[120] LIN, S., JI, R., YAN, C., ZHANG, B., CAO, L., YE, Q., HUANG, F., AND DOERMANN, D. Towards optimal structured cnn pruning via generative adversarial learning, 2019.

[121] LIN, T.-Y., MAIRE, M., BELONGIE, S., HAYS, J., PERONA, P., RAMANAN, D., DOLLÁR, P., AND ZITNICK, C. L. Microsoft coco: Common objects in context. In *European conference on computer vision* (2014), Springer, pp. 740–755.

[122] LIU, B., ZOU, D., FENG, L., FENG, S., FU, P., AND LI, J. An fpga-based cnn accelerator integrating depthwise separable convolution. *Electronics 8*, 3 (2019), 281.

[123] LIU, C., ZOPH, B., NEUMANN, M., SHLENS, J., HUA, W., LI, L.-J., FEI-FEI, L., YUILLE, A., HUANG, J., AND MURPHY, K. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)* (2018), pp. 19–34.

[124] LIU, J., CHEN, Y., AND LIU, K. Exploiting the ground-truth: An adversarial imitation based knowledge distillation approach for event detection. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), vol. 33, pp. 6754–6761.

[125] LIU, R., FUSI, N., AND MACKEY, L. Model compression with generative adversarial networks.

[126] LIU, S., LIN, Y., ZHOU, Z., NAN, K., LIU, H., AND DU, J. On-demand deep model compression for mobile devices: A usage-driven model selection framework. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services* (2018), pp. 389–400.

[127] LIU, Y., JIA, X., TAN, M., VEMULAPALLI, R., ZHU, Y., GREEN, B., AND WANG, X. Search to distill: Pearls are everywhere but not the eyes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 7539–7548.

[128] LIU, Y., AND LAPATA, M. Learning structured text representations. *Transactions of the Association for Computational Linguistics 6* (2018), 63–75.

[129] LIU, Z., LI, J., SHEN, Z., HUANG, G., YAN, S., AND ZHANG, C. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision* (2017), pp. 2736–2744.

[130] LIU, Z., SUN, M., ZHOU, T., HUANG, G., AND DARRELL, T. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270* (2018).

[131] LOTRIČ, U., AND BULIĆ, P. Applicability of approximate multipliers in hardware neural networks. *Neurocomputing 96* (2012), 57–65.

[132] MA, N., ZHANG, X., ZHENG, H.-T., AND SUN, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)* (2018), pp. 116–131.

[133] MALACH, E., YEHUDAI, G., SHALEV-SCHWARTZ, S., AND SHAMIR, O. Proving the lottery ticket hypothesis: Pruning is all you need. In *International Conference on Machine Learning* (2020), PMLR, pp. 6682–6691.

[134] MALLYA, A., AND LAZEBNIK, S. Piggyback: Adding multiple tasks to a single, fixed network by learning to mask. *arXiv preprint arXiv:1801.06519 6*, 8 (2018).

[135] MAO, H., HAN, S., POOL, J., LI, W., LIU, X., WANG, Y., AND DALLY, W. J. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv preprint arXiv:1705.08922* (2017).

[136] MATHIEU, M., HENAFF, M., AND LECUN, Y. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851* (2013).

[137] MCDANEL, B., TEERAPITTAYANON, S., AND KUNG, H. Embedded binarized neural networks. *arXiv preprint arXiv:1709.02260* (2017).

[138] McKinney, S. M., Sieniek, M., Godbole, V., Godwin, J., Antropova, N., Ashrafian, H., Back, T., Chesus, M., Corrado, G. S., Darzi, A., et al. International evaluation of an ai system for breast cancer screening. *Nature 577*, 7788 (2020), 89–94.

[139] Mellempudi, N., Kundu, A., Mudigere, D., Das, D., Kaul, B., and Dubey, P. Ternary neural networks with fine-grained quantization. *arXiv preprint arXiv:1705.01462* (2017).

[140] Melnikov, A. D., Tsentalovich, Y. P., and Yanshole, V. V. Deep learning for the precise peak detection in high-resolution lc–ms data. *Analytical chemistry 92*, 1 (2019), 588–592.

[141] Meng, Z., Li, J., Zhao, Y., and Gong, Y. Conditional teacher-student learning. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2019), IEEE, pp. 6445–6449.

[142] Mishra, A., and Marr, D. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852* (2017).

[143] Miyashita, D., Lee, E. H., and Murmann, B. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025* (2016).

[144] Mobahi, H., Farajtabar, M., and Bartlett, P. L. Self-distillation amplifies regularization in hilbert space. *arXiv preprint arXiv:2002.05715* (2020).

[145] Morcos, A. S., Yu, H., Paganini, M., and Tian, Y. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. *arXiv preprint arXiv:1906.02773* (2019).

[146] Mozer, M. C., and Smolensky, P. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in neural information processing systems* (1989), pp. 107–115.

[147] Nair, V., and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning* (Madison, WI, USA, 2010), ICML'10, Omnipress, p. 807–814.

[148] Neill, J. O. An overview of neural network compression. *arXiv preprint arXiv:2006.03669* (2020).

[149] Nesterov, Y. A method for unconstrained convex minimization problem with the rate of convergence o (1/k^ 2). In *Doklady an ussr* (1983), vol. 269, pp. 543–547.

[150] Nowak, T. S., and Corso, J. J. Deep net triage: Analyzing the importance of network layers via structural compression. *arXiv preprint arXiv:1801.04651* (2018).

[151] OH, D., STRATTAN, J. S., HUR, J. K., BENTO, J., URBAN, A. E., SONG, G., AND CHERRY, J. M. Cnn-peaks: Chip-seq peak detection pipeline using convolutional neural networks that imitate human visual inspection. *Scientific reports 10*, 1 (2020), 1–12.

[152] OLIVAS, E. S., GUERRERO, J. D. M., SOBER, M. M., BENEDITO, J. R. M., AND LOPEZ, A. J. S. *Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques - 2 Volumes*. Information Science Reference - Imprint of: IGI Publishing, Hershey, PA, 2009.

[153] PAGANINI, M., AND FORDE, J. Z. Bespoke vs. pr\^ et-\a-porter lottery tickets: Exploiting mask similarity for trainable sub-network finding. *arXiv preprint arXiv:2007.04091* (2020).

[154] PAN, H., HE, X., TANG, S., AND MENG, F. An improved bearing fault diagnosis method using one-dimensional cnn and lstm. *J. Mech. Eng 64*, 7-8 (2018), 443–452.

[155] PAPERNOT, N., MCDANIEL, P., WU, X., JHA, S., AND SWAMI, A. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE symposium on security and privacy (SP)* (2016), IEEE, pp. 582–597.

[156] PASSALIS, N., TZELEPI, M., AND TEFAS, A. Heterogeneous knowledge distillation using information flow modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 2339–2348.

[157] PENG, X., SUN, B., ALI, K., AND SAENKO, K. Learning deep object detectors from 3d models. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)* (Washington, DC, USA, 2015), ICCV '15, IEEE Computer Society, pp. 1278–1286.

[158] PHAM, H., DAI, Z., XIE, Q., AND LE, Q. V. Meta pseudo labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 11557–11568.

[159] PILIPOVIĆ, R., BULIĆ, P., AND RISOJEVIĆ, V. Compression of convolutional neural networks: A short survey. In *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)* (2018), IEEE, pp. 1–6.

[160] POLINO, A., PASCANU, R., AND ALISTARH, D. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668* (2018).

[161] QIU, J., WANG, J., YAO, S., GUO, K., LI, B., ZHOU, E., YU, J., TANG, T., XU, N., SONG, S., ET AL. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (2016), pp. 26–35.

[162] RADFORD, A. "visualizing optimization algos".

[163] RAI, R., TIWARI, M. K., IVANOV, D., AND DOLGUI, A. Machine learning in manufacturing and industry 4.0 applications, 2021.

[164] RASTEGARI, M., ORDONEZ, V., REDMON, J., AND FARHADI, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision* (2016), Springer, pp. 525–542.

[165] REN, S., HE, K., GIRSHICK, R., AND SUN, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497* (2015).

[166] RESSI, D. Feasibility of deep-learning methods for automatic fruit classification tasks. Master's thesis, Ca' Foscari di Venezia, Italy, 2016.

[167] RIGAMONTI, R., SIRONI, A., LEPETIT, V., AND FUA, P. Learning separable filters. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2013), pp. 2754–2761.

[168] ROGEZ, G., AND SCHMID, C. Mocap-guided data augmentation for 3d pose estimation in the wild. In *NIPS* (2016).

[169] ROMERO, A., BALLAS, N., KAHOU, S. E., CHASSANG, A., GATTA, C., AND BENGIO, Y. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550* (2014).

[170] ROSENBLATT, F. A probabilistic model for information storage and organization in the brain. *Artificial Intelligence: Critical Concepts 2*, 6 (2000), 398.

[171] RUIZ, J. T., PÉREZ, J. D. B., AND BLÁZQUEZ, J. R. B. Arrhythmia detection using convolutional neural models. In *International Symposium on Distributed Computing and Artificial Intelligence* (2018), Springer, pp. 120–127.

[172] RUMELHART, D. E., HINTON, G. E., AND WILLIAMS, R. J. Learning representations by back-propagating errors. *nature 323*, 6088 (1986), 533–536.

[173] RUSSAKOVSKY, O., DENG, J., SU, H., KRAUSE, J., SATHEESH, S., MA, S., HUANG, Z., KARPATHY, A., KHOSLA, A., BERNSTEIN, M., ET AL. Imagenet large scale visual recognition challenge. *International journal of computer vision 115*, 3 (2015), 211–252.

[174] SANDLER, M., HOWARD, A., ZHU, M., ZHMOGINOV, A., AND CHEN, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), pp. 4510–4520.

[175] SHORTEN, C., AND KHOSHGOFTAAR, T. M. A survey on image data augmentation for deep learning. *Journal of Big Data 6*, 1 (2019), 1–48.

[176] SIDIROPOULOS, N. D., DE LATHAUWER, L., FU, X., HUANG, K., PAPALEXAKIS, E. E., AND FALOUTSOS, C. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing 65*, 13 (2017), 3551–3582.

[177] SIMONS, T., AND LEE, D.-J. A review of binarized neural networks. *Electronics 8*, 6 (2019), 661.

[178] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *CoRR abs/1409.1556* (2014).

[179] SIVANARAYANA, G., KUMAR, K. N., SRINIVAS, Y., AND KUMAR, G. R. Review on the methodologies for image segmentation based on cnn. In *Communication Software and Networks*. Springer, 2021, pp. 165–175.

[180] SUAU, X., APOSTOLOFF, N., ET AL. Filter distillation for network compression. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)* (2020), IEEE, pp. 3129–3138.

[181] SUN, Y., GU, X., SUN, K., HU, H., XU, M., WANG, Z., TU, K., AND PAN, L. Hyperspectral reflectance imaging combined with chemometrics and successive projections algorithm for chilling injury classification in peaches. *Lwt 75* (2017), 557–564.

[182] SZE, V., CHEN, Y.-H., YANG, T.-J., AND EMER, J. S. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE 105*, 12 (2017), 2295–2329.

[183] SZEGEDY, C., LIU, W., JIA, Y., SERMANET, P., REED, S., ANGUELOV, D., ERHAN, D., VANHOUCKE, V., AND RABINOVICH, A. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015), pp. 1–9.

[184] SZEGEDY, C., LIU, W., JIA, Y., SERMANET, P., REED, S., ANGUELOV, D., ERHAN, D., VANHOUCKE, V., AND RABINOVICH, A. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), pp. 1–9.

[185] SZEGEDY, C., VANHOUCKE, V., IOFFE, S., SHLENS, J., AND WOJNA, Z. Rethinking the inception architecture for computer vision. *CoRR abs/1512.00567* (2015).

[186] SZEGEDY, C., VANHOUCKE, V., IOFFE, S., SHLENS, J., AND WOJNA, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 2818–2826.

[187] TAN, M., CHEN, B., PANG, R., VASUDEVAN, V., SANDLER, M., HOWARD, A., AND LE, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), pp. 2820–2828.

[188] TAN, M., AND LE, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning* (2019), PMLR, pp. 6105–6114.

[189] TAN, M., AND LE, Q. V. Efficientnetv2: Smaller models and faster training. *arXiv preprint arXiv:2104.00298* (2021).

[190] TAYLOR, L., AND NITSCHKE, G. Improving deep learning using generic data augmentation. *CoRR abs/1708.06020* (2017).

[191] THEIS, L., KORSHUNOVA, I., TEJANI, A., AND HUSZÁR, F. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787* (2018).

[192] TUCKER, L. R. Some mathematical notes on three-mode factor analysis. *Psychometrika 31*, 3 (1966), 279–311.

[193] UNAY, D., AND GOSSELIN, B. Automatic defect segmentation of 'jonagold'apples on multi-spectral images: A comparative study. *Postharvest Biology and Technology 42*, 3 (2006), 271–279.

[194] VAN SOELEN, R., AND SHEPPARD, J. W. Using winning lottery tickets in transfer learning for convolutional neural networks. In *2019 International Joint Conference on Neural Networks (IJCNN)* (2019), IEEE, pp. 1–8.

[195] WAJID, A., SINGH, N. K., JUNJUN, P., AND MUGHAL, M. A. Recognition of ripe, unripe and scaled condition of orange citrus based on decision tree classification. In *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)* (2018), IEEE, pp. 1–4.

[196] WAN, P., TOUDESHKI, A., TAN, H., AND EHSANI, R. A methodology for fresh tomato maturity detection using computer vision. *Computers and electronics in agriculture 146* (2018), 43–50.

[197] WANG, H., HU, X., ZHANG, Q., WANG, Y., YU, L., AND HU, H. Structured pruning for efficient convolutional neural networks via incremental regularization. *IEEE Journal of Selected Topics in Signal Processing 14*, 4 (2019), 775–788.

[198] WANG, J., BAO, W., SUN, L., ZHU, X., CAO, B., AND PHILIP, S. Y. Private model compression via knowledge distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), vol. 33, pp. 1190–1197.

[199] WANG, R. J., LI, X., AND LING, C. X. Pelee: A real-time object detection system on mobile devices. *arXiv preprint arXiv:1804.06882* (2018).

[200] WEN, W., WU, C., WANG, Y., CHEN, Y., AND LI, H. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems 29* (2016), 2074–2082.

[201] WU, J., LENG, C., WANG, Y., HU, Q., AND CHENG, J. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 4820–4828.

[202] Wu, S., Li, G., Chen, F., and Shi, L. Training and inference with integers in deep neural networks. *arXiv preprint arXiv:1802.04680* (2018).

[203] Xie, Q., Luong, M.-T., Hovy, E., and Le, Q. V. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 10687–10698.

[204] Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 1492–1500.

[205] Xie, T., Zhang, R., Liang, J., Jia, Q., Fan, X., and Luo, Z. An efficient ellipse detector based on region detection and arc pruning. In *2020 IEEE International Conference on Image Processing (ICIP)* (2020), IEEE, pp. 2890–2894.

[206] Xu, K., Long, W., Sun, Y., and Lin, Y. A novel image feature extraction algorithm based on the fusion autoencoder and cnn. In *Journal of Physics: Conference Series* (2020), vol. 1646, IOP Publishing, p. 012039.

[207] Yaeger, L. S., Lyon, R. F., and Webb, B. J. Effective training of a neural network character classifier for word recognition. In *Advances in Neural Information Processing Systems 9*, M. C. Mozer, M. I. Jordan, and T. Petsche, Eds. MIT Press, 1997, pp. 807–816.

[208] Yang, T.-J., Chen, Y.-H., and Sze, V. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 5687–5695.

[209] Yang, Z., Dai, Z., Salakhutdinov, R., and Cohen, W. W. Breaking the softmax bottleneck: A high-rank rnn language model. *arXiv preprint arXiv:1711.03953* (2017).

[210] Yim, J., Joo, D., Bae, J., and Kim, J. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 4133–4141.

[211] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. How transferable are features in deep neural networks? *CoRR abs/1411.1792* (2014).

[212] You, S., Xu, C., Xu, C., and Tao, D. Learning from multiple teacher networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017), pp. 1285–1294.

[213] Yu, R., Li, A., Chen, C.-F., Lai, J.-H., Morariu, V. I., Han, X., Gao, M., Lin, C.-Y., and Davis, L. S. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 9194–9203.

[214] Zaborowicz, M., Boniecki, P., Koszela, K., Przybylak, A., and Przybył, J. Application of neural image analysis in evaluating the quality of greenhouse tomatoes. *Scientia Horticulturae 218* (2017), 222–229.

[215] Zagoruyko, S., and Komodakis, N. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928* (2016).

[216] Zeiler, M. D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* (2012).

[217] Zhai, X., Kolesnikov, A., Houlsby, N., and Beyer, L. Scaling vision transformers. *arXiv preprint arXiv:2106.04560* (2021).

[218] Zhang, C., Patras, P., and Haddadi, H. Deep learning in mobile and wireless networking: A survey. *IEEE Communications surveys & tutorials 21*, 3 (2019), 2224–2287.

[219] Zhang, D., Yang, J., Ye, D., and Hua, G. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)* (2018), pp. 365–382.

[220] Zhang, L., Song, J., Gao, A., Chen, J., Bao, C., and Ma, K. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), pp. 3713–3722.

[221] Zhang, W., Peng, G., and Li, C. Bearings fault diagnosis based on convolutional neural networks with 2-d representation of vibration signals as input. In *MATEC web of conferences* (2017), vol. 95, EDP Sciences, p. 13001.

[222] Zhang, X., Zhou, X., Lin, M., and Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), pp. 6848–6856.

[223] Zhang, Y., Phillips, P., Wang, S., Ji, G., Yang, J., and Wu, J. Fruit classification by biogeography-based optimization and feedforward neural network. *Expert Systems 33*, 3 (2016), 239–253.

[224] Zhou, G., Fan, Y., Cui, R., Bian, W., Zhu, X., and Gai, K. Rocket launching: A universal and efficient framework for training well-performing light net. In *Thirty-second AAAI conference on artificial intelligence* (2018).

[225] Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).

[226] ZHU, M., AND GUPTA, S. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878* (2017).

[227] ZHU, X., LIU, Y., LI, J., WAN, T., AND QIN, Z. Emotion classification with data augmentation using generative adversarial networks. In *Pacific-Asia conference on knowledge discovery and data mining* (2018), Springer, pp. 349–360.

[228] ZIHLMANN, M., PEREKRESTENKO, D., AND TSCHANNEN, M. Convolutional recurrent neural networks for electrocardiogram classification. In *2017 Computing in Cardiology (CinC)* (2017), IEEE, pp. 1–4.

[229] ZOPH, B., AND LE, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).

[230] ZOPH, B., VASUDEVAN, V., SHLENS, J., AND LE, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), pp. 8697–8710.

Università
Ca'Foscari
Venezia

DEPOSITO ELETTRONICO DELLA TESI DI DOTTORATO

DICHIARAZIONE SOSTITUTIVA DELL'ATTO DI NOTORIETA'
(Art. 47 D.P.R. 445 del 28/12/2000 e relative modifiche)

Io sottoscritto DALILA RESSI

natA a SAN VITO AL TAGLIAMENTO (prov. PN) il 20/08/1991

residente a MUSILE DI PIAVE in VIA MARTIRI n. 47

Matricola (se posseduta) 839745 Autore della tesi di dottorato dal titolo:

CONVOLUTIONAL NEURAL NETWORKS COMPRESSION FOR EMBEDDED INDUSTRIAL APPLICATIONS

Dottorato di ricerca in COMPUTER SCIENCE

(in cotutela con ……………………………………)

Ciclo 33

Anno di conseguimento del titolo 2020/2021

DICHIARO

di essere a conoscenza:
1) del fatto che in caso di dichiarazioni mendaci, oltre alle sanzioni previste dal codice penale e dalle Leggi speciali per l'ipotesi di falsità in atti ed uso di atti falsi, decado fin dall'inizio e senza necessità di nessuna formalità dai benefici conseguenti al provvedimento emanato sulla base di tali dichiarazioni;
2) dell'obbligo per l'Università di provvedere, per via telematica, al deposito di legge delle tesi di dottorato presso le Biblioteche Nazionali Centrali di Roma e di Firenze al fine di assicurarne la conservazione e la consultabilità da parte di terzi;
3) che l'Università si riserva i diritti di riproduzione per scopi didattici, con citazione della fonte;
4) del fatto che il testo integrale della tesi di dottorato di cui alla presente dichiarazione viene archiviato e reso consultabile via Internet attraverso l'Archivio Istituzionale ad Accesso Aperto dell'Università Ca' Foscari, oltre che attraverso i cataloghi delle Biblioteche Nazionali Centrali di Roma e Firenze;
5) del fatto che, ai sensi e per gli effetti di cui al D.Lgs. n. 196/2003, i dati personali raccolti saranno trattati, anche con strumenti informatici, esclusivamente nell'ambito del procedimento per il quale la presentazione viene resa;
6) del fatto che la copia della tesi in formato elettronico depositato nell'Archivio Istituzionale ad Accesso Aperto è del tutto corrispondente alla tesi in formato cartaceo, controfirmata dal tutor, consegnata presso la segreteria didattica del dipartimento di riferimento del corso di dottorato ai fini del deposito presso l'Archivio di Ateneo, e che di conseguenza va esclusa qualsiasi responsabilità dell'Ateneo stesso per quanto riguarda eventuali errori, imprecisioni o omissioni nei contenuti della tesi;
7) del fatto che la copia consegnata in formato cartaceo, controfirmata dal tutor, depositata nell'Archivio di Ateneo, è l'unica alla quale farà riferimento l'Università per rilasciare, a richiesta, la dichiarazione di conformità di eventuali copie;

Data 20/12/2021 **Firma** Dalila Ressi

NON AUTORIZZO

l'Università a riprodurre ai fini dell'immissione in rete e a comunicare al pubblico tramite servizio on line entro l'Archivio Istituzionale ad Accesso Aperto la tesi depositata per un periodo di 12 (dodici) mesi a partire dalla data di conseguimento del titolo di dottore di ricerca.

DICHIARO

1) che la tesi, in quanto caratterizzata da vincoli di segretezza, non dovrà essere consultabile on line da terzi per un periodo di 12 (dodici) mesi a partire dalla data di conseguimento del titolo di dottore di ricerca;
2) di essere a conoscenza del fatto che la versione elettronica della tesi dovrà altresì essere depositata a cura dell'Ateneo presso le Biblioteche Nazionali Centrali di Roma e Firenze dove sarà comunque consultabile su PC privi di periferiche; la tesi sarà inoltre consultabile in formato cartaceo presso l'Archivio Tesi di Ateneo;
3) di essere a conoscenza che allo scadere del dodicesimo mese a partire dalla data di conseguimento del titolo di dottore di ricerca la tesi sarà immessa in rete e comunicata al pubblico tramite servizio on line entro l'Archivio Istituzionale ad Accesso Aperto.

Specificare la motivazione:
□ motivi di segretezza e/o di proprietà dei risultati e/o informazioni sensibili dell'Università Ca' Foscari di Venezia.
[✓ motivi di segretezza e/o di proprietà dei risultati e informazioni di enti esterni o aziende private che hanno partecipato alla realizzazione del lavoro di ricerca relativo alla tesi di dottorato.]
□ dichiaro che la tesi di dottorato presenta elementi di innovazione per i quali è già stata attivata / si intende attivare la seguente procedura di tutela:

……………………………………………………………………………………………………;

□ Altro (specificare):

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

A tal fine:
- dichiaro di aver consegnato la copia integrale della tesi in formato elettronico tramite auto-archiviazione (upload) nel sito dell'Università; la tesi in formato elettronico sarà caricata automaticamente nell'Archivio Istituzionale ad Accesso Aperto dell'Università Ca' Foscari, dove rimarrà non accessibile fino allo scadere dell'embargo, e verrà consegnata mediante procedura telematica per il deposito legale presso la Biblioteca Nazionale Centrale di Firenze;
- consegno la copia integrale della tesi in formato cartaceo presso la segreteria didattica del dipartimento di riferimento del corso di dottorato ai fini del deposito presso l'Archivio di Ateneo.

Data …20/12/2021………… **Firma** ……[illegible]……………

La presente dichiarazione è sottoscritta dall'interessato in presenza del dipendente addetto, ovvero sottoscritta e inviata, unitamente a copia fotostatica non autenticata di un documento di identità del dichiarante, all'ufficio competente via fax, ovvero tramite un incaricato, oppure a mezzo posta.

Firma del dipendente addetto ……………………………………………………

Estratto per riassunto della tesi di dottorato

L'estratto (max. 1000 battute) deve essere redatto sia in lingua italiana che in lingua inglese e nella lingua straniera eventualmente indicata dal Collegio dei docenti.
L'estratto va firmato e rilegato come ultimo foglio della tesi.

Studente:	Dalila Ressi	matricola:	839745
Dottorato:	Computer Science		
Ciclo:	33		

Titolo della tesi[1] :

Convolutional Neural Networks Compression

for Embedded Industrial Applications

Abstract:

Convolutional Neural Networks have proved to be a powerful tool to solve a wide range of Computer Vision tasks, especially where is difficult to implement a solution in a purely algorithmic way. In the industry, the availability of powerful deep models to address classification, detection, and image segmentation now offers new possibilities for automating not only the production, but also the quality assessment of the final products. Unfortunately, industrial applications have to face some limitations, especially when dealing with the so called ”Embedded Vision” solutions where such models have to be transferred and used directly on-camera. Indeed, limited memory and computational capability pose important challenges on the architecture of the model to use. During the last years a large amount of research aimed to face such problems, proposing various compression algorithms to reduce the number of parameters of neural networks. The purpose of this thesis is to study such challenges, explore the existing literature and to give a contribute by proposing novel methods applied to some specific industrial use-case.

Firma dello studente
Dalila Ressi

[1] Il titolo deve essere quello definitivo, uguale a quello che risulta stampato sulla copertina dell'elaborato consegnato.