POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea

Convolutional neural networks for predicting the state of electrical switches

Relatore
prof. Enrico Magli
. . . . . .

Candidato
Davide Taricco
. . . . . .

Anno Accademico 2019-2020

Alla mia famiglia

Summary

The aim of this master thesis is to expose the work done during my five months of internship at the Blue Reply company in Turin where I worked on a commission project from one of the largest energy companies in the world. It has consisted in the development of an API based on a machine learning algorithm that is able to receive as input an image of an eletrical switch, to detect the QR codes which are present and to extract the information in them, to crop the area of interest and to query the appropriate convolutional neural network previously trained to give as output a prediction about the state of the switch. All the work done is the beginning of a project that aims to increase the safety at work. Indeed, the application has the goal to support the operators which have to intervene on the switches for repair or maintenance operations. To do that they must be sure that there are not risks. The issue of safety at work always covers a primary role, indeed several awards are given each year to the safest companies, and thanks to new technologies it is possible to aim to improve it more and more. A key role in the project is done by the neural networks which are one of the most important tool of the deep learning. They, as well as the object detector also present in the thesis, are increasingly common today and the challenge is to understand how to use them intelligently to solve practical problems. We have already used terms like machine learning and deep learning that belong to the area of artificial intelligence and since they will often be used during the thesis, then let's clarify for a moment what they mean so as not to confuse terminology and focus later on the main topics. There are many ways to present these terms and their relationships, in my case I chose to start with the definition more general to go into the most specific. They are taken from the free ebook *"Deep Learning vs Machine Learning: choosing the best approach"* avaiable on MATLAB.

1. **Artificial Intelligence (AI).** It is a computer system trained to perceive its environment, make decisions, and take actions. AI systems rely on learning algorithms, such as machine learning and deep learning, along with large sets of sensor data with well-defined representations of objective truth. Her date of birth is fixed in the 50s.

2. **Machine Learning (ML).** We use machine learning as shorthand for traditional machine learning, the workflow in which you manually select features and then train the model. When we refer to it we exclude deep learning. Common machine learning techniques include decision tree, SVM, and ensemble methods. We start talking about this branch of artificial intelligence at the end of the 50s.

3. **Deep learning.** It is that research field of machine learning inspired on the neural pathways of the human brain. Deep refers to the multiple layers between the input and output ones. In deep learning, the algorithm automatically learns what features are useful. Common techniques include convolutional neural networks (CNNs), recurrent neural networks, and deep Q networks. The deep learning is more accurate than humans on image classification. The first results date back to the 80s.

The thesis work is structured as follows. As first thing neural networks will be introduced, explaining what they are, how they work and giving a brief historical mention. The main attention will be placed on the convolutional neural networks, in particular on architecture known as ResNet. These notions are given both to understand how we arrived at the current situation, which remains however in continuous evolution as this is a hot topic of these years, both to provide basic information to understand how they work. After these explanations, we will move on to the work done and discuss in detail: the develpment of the machine learning algorithm, the creation of different datasets, the object detection implementation which is another important tool of deep learning, the training of neural networks and the evaluation of the results which have been obtained. Precisely their goodness has been evaluated not only taking into consideration classic performance measures such as accuracy and confidence but also through an explanatory model that has been implemented and included in an internal API. This last point is

very important because neural networks act like black-boxes, therefore it is important to understand why certain predictions are made and especially if they can be considered reliable.

Acknowledgements

Il primo pensiero va alla mia famiglia che mi ha supportato e sopportato in questo mio percorso scolastico e di vita, non facendomi mai mancare nulla sotto tutti i punti di vista, senza di loro oggi non sarei qui. A mia madre Marinella, mio padre Biagio e mia sorella Federica: GRAZIE.
In secondo luogo volevo ringraziare i miei amici con i quali mi sono svagato e divertito in questi anni di studio e ai miei compagni di corso che hanno reso le lezioni molto più divertenti.
Infine colgo l'occasione per ringraziare l'azienda Blue Reply per avermi dato la possibilità di svolgere il tirocinio e il mio relatore Enrico Magli per avermi seguito durante questo mio lavoro di tesi.

Davide

Contents

List of Tables

List of Figures

Chapter 1

Neural networks

1.1 General principles and history

Artificial Neural Networks (ANNs) or connectionist systems are computing systems that are inspired by, but not identical to, biological neural networks that constitute animal brains.[1] Such systems aim to perform tasks by considering examples, generally without being programmed with task-specific rules. For example, in image recognition, they might learn to identify images which contain a specific class such as cars, cats, people just analyzing example images that have been manually labeled to indicate whether the class is present or not and to use the results to identify it in other images. They are able to do this without any prior knowledge of class or its characteristics. An ANN is based on a set of connected units or nodes called artificial neurons or more simply neurons that represent the counter part in the natural brain. Instead the synapses, namely the connections in the biological brain, are represented in the neural networks as arrows and they allow the transmission of the signals between the neurons. An artificial neuron that receives a signal then processes it and can send an output one to the neurons connected to it, in fact neurons usually have a threshold that establishes if the signal can be sent or not. In ANN implementations, the signal at a connection is a real number and the output of each node is computed by applying some non-linear function of

[1] From Wikipedia, link: https://en.wikipedia.org/wiki/Artificial_neural_network.

the sum of its inputs. Furthermore the connections have an associated weight that adjusts as learning process which increases or decreases the importance of the signal at a connection. Typically, neurons are aggregated into layers. Different layers can apply different transformation on their input. A signal travels from the first level, the input layer, to the last one, the output layer, after passing through some internal steps, the hidden layer, usually composed by more levels. It's important to stress that this model of biological brain is very coarse. Indeed, for example, in our brain there are many several types of neurons each with different properties and the synapses are not just a single weight, but they are a complex non-linear dynamical system. Moreover the exact timing of the output spikes in many system is known to be important, suggesting that the rate code approximation may not hold. 1.1.

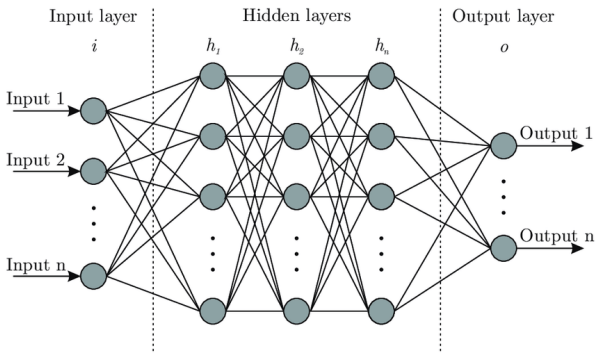

Figure 1.1. A simple example of an ANN with three hidden layers.

Originally, the artificial neural networks should have solved problems in the same way that as a human brain. However, over time, attention has been moved to performing specific tasks, leading to deviations from biology. Some common fields where the ANNs are used are: computer vision, speech recognition, machine translation, social network filtering, playing board and video games. Our problem of recognizing the state of electrical switches belongs to the field of the computer vision, precisely at the image classification, that is the task of assigning an input image one label from a fixed set of categories. Although this job of recognizing a visual concept is relatively simple for a human to perform, it is not the same for

a computer and the problem is therefore worth considering. In fact, it may incur into several challenges, here are the most common:

1. **Viewpoint variation.** An object can be oriented in many ways with respect to the image.
2. **Scale variation.** It does not refer only of the image size, but also at the different dimension that a object can have.
3. **Deformation.** Many objects are not rigid bodies and can be deformed in extreme ways.
4. **Occlusion.** An object can be partially cover.
5. **Illumination conditions.** The effects of illumination are drastic on the pixel level.
6. **Background clutter.** It may happen that the object of interest camouflages itself with the surrounding environment.

A good image classifier must be invariant to the cross product of all these variations. Before seeing how the artificial neural networks work, let's look at some historical hints which explain of how we got to today.

The first big step to the born and the development of the ANNs has been made in 1943 by Warren Sturgis McCulloch, a neurophysiologist, and Walter Pitts, a mathematical, in their pubblication: *"A logical calculus of the ideas immanent in nervous activity"*. They tried to create a first model of artificial neuron and, as result, it was able to calculate some simple boolean functions. From 1943 to the end of the 50s very little happened; the only thing to note is that in 1949 a Canadian psychologist Donald Olding Hebb tried to explain the complex model of the brain and to extrapolate the first hypotheses of learning of the neural networks. This theory became known as Hebbian learning. An important step happened in 1958 with the first skeleton of neural network presented by Frank Rosenblatt, American psychologist and computer scientist. The proposed scheme was a precursor of the actual neural networks and it took the name of perceptron. It was a network with an input and an output layer and with an intermediate learning rule. The

perceptron is an algorithm for supervised learning of binary classifiers. It is a type of linear classifier that makes its predictions based on a linear predictor function combining a set of weights with the feature vector.[2] The Rosenblatt's theories aroused great interest in the scientific community for over a decade and in the 1965 a first functional network with many layers were published by Ivakhnenko and Lapa as the Group Method of Data Handling. However in 1969 Marvin Minsky and Seymour Papert showed the limits: the perceptron was only capable of learning linearly separable patters, it was not able to calculate XOR function.[3] It was necessary wait for another decade to saw something to innovative. In 1986 David Rumelhart introduced the third layer of the neural networks which took the name of hidden. This new scheme allowed the construction of models for training the multi-layers perceptron networks (MLPs). He proposed the backpropagation algorithm which is still the basis of the ANNs. In 1992, max-pooling was introduced to help with least shift invariance and tolerance to deformation to aid in 3D object recognition. In 2012 Andrew Yan-Tak Ng and Jeffrey Adgate Dean, two computer scientists, created a network that learned to recognize higher-level tasks, such as cats, only from watching unlabeled images. Unsupervised pretraining and increased computing power from GPUs (Graphics Processing Units) and distributed computing allowed the use of larger networks, particularly in image and visual recognition problems, which became known as deep learning.

Let's see how an artifical neural network works in the next section.

1.2 How a neural network works

Let's start focusing on how a single neuron works in a neural network. Consider, for example, the neuron in the figure 1.2. The node takes two numerical inputs x_1 and x_2 with the respective weights w_1 and w_2. Additionaly, there is another

[2]From Wikipedia, link: https://en.wikipedia.org/wiki/Perceptron.

[3]Exclusive OR is a logical operation that outputs true only when inputs differ (one is true, the other is false).

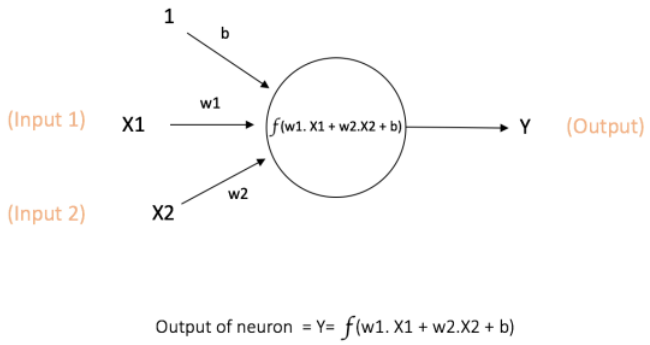

Figure 1.2. A simple neuron in a neural network.

input 1 with weight b called bias. The bias is similar to the intercept term from regression, but the key difference is that in neural networks, every neuron has its own bias term while in the regression, the model has a singular intercept term. The output y from the node is computed applying the function f that takes the name of activation function. The goal of f is to introduce non-linearity into the output of a neuron. This is fundamental because most real world data is non linear and we want that the neurons are able to learn this features. Every activation (or non-linearity) function takes a single number and performs a certain fixed mathematical operation on it.[4] The most common activation functions that we may encounter in practice are:

1. **Sigmoid.** It maps a real number in the interval [0,1] and it can be represented as $\sigma(x) = \frac{1}{1+e^{-x}}$. It has been frequently used historically since it has a nice interpretation, but it has recently fallen out of favor and it is rarely ever used nowadays. The drawbacks are principally two. The first one is that the sigmoid saturates and kills gradients. When the saturation happens at either tail of 0 or 1, the gradient at these regions is almost zero causing a slowing of the learning. During backpropagation, this local gradient will be multiplied to the gradient of this gate's output for the whole objective[5].

[4]According with Neural Networks Part 1: Setting up the Architecture (Stanford CNN Tutorial), link: http://cs231n.github.io/neural-networks-1/.

[5]The backpropagation algorithm will be better explained later as well as the gradient descent

Therefore, if the local gradient is very small, it will kill the gradient and almost no signal will flow through the neuron to its weights and recursively to its data. Additionally, we have to take attention when initializing the weights of sigmoind neurons to prevent saturation. Indeed, if they are too large then most neurons would become saturated and the network would barely learn. The second disadvantage is that sigmoind outputs are not zero-centered. This is undesirable since neurons in later layers in the neural network would be receiving data that is not zero-centered. This causes complication on the dynamics during the gradient descent, because if the data coming into a neuron is always positive, then the gradient on the weights w will during backpropagation become either all be positive, or all negative. This could introduce undesirable zig-zagging dynamics in the gradient updates for the weights.

2. **tanh.** It maps a real number in the interval [−1,1] and it can be expressed in relation of sigmoid function as $tanh(x) = 2\sigma(2x) - 1$. Similarly to a sigmoid neuron its activation saturate, but unlike the latter its output is zero-centered. It is preferred to the previous non-linear function.

3. **ReLU.**[6] It takes a real number and thresholds it a zero, that is it replaces negative values with zero. It can be represented as $f(x) = max(0, x)$. There are several pros and cons to using the ReLU. The main advantages are that it greatly accelerates the convergence of stochastic gradient descent compared to the two previous functions. It is due to its non-saturating form. Moreover, while sigmoind and tanh are expensive to implement, it can be implemented by simply thresholding a matrix of activations at zero. The drawback is that ReLU units can be fragile during training and can die. This can happen if the learning rate is set too high. For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the

one.

[6]The acronym ReLU stands for Rectified Linear Unit.

gradient flowing through the unit will forever be zero from that point on. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold. This function presents some variants, more flexible, like **leaky ReLU** that instead of mapping the value in zero when $x < 0$, it maps them in a curve with a negative slope. It presents the form $f(x) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x)$ where α is a small constant and it tries to fix the dying ReLU problem.

4. **Maxout.** It is a generalization of the ReLU and its leaky version. It can be expressed as $f(x) = \max(w_1^T x + b_1, w_2^T x + b_2)$. The maxout neuron therefore enjoys all the benefits of a ReLU units and does not present its drawbacks. However, the number of parameters that we have to compute for every single neuron it is double respect the ReLU neurons and this leading to a high number of parameters.

1.3.

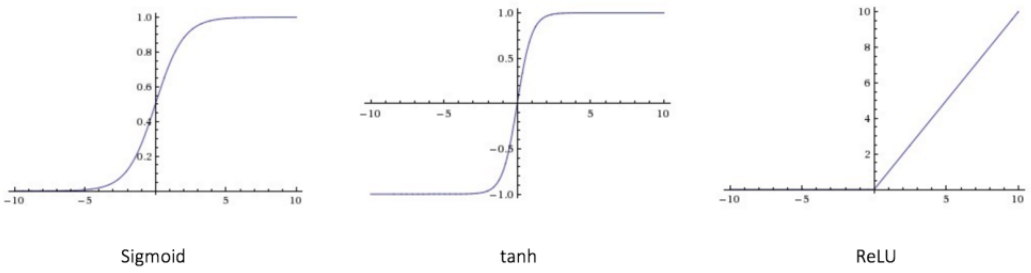

Figure 1.3. From left to right: sigmoind, tanh, ReLU.

Now that we have seen how a single neuron works, we can shift our attention on the whole neural network. The first type of artificial neural network developed has been the feedforward neural network. It is a multi-layers[7] network where nodes from adjacent layers have weighted connections between them. As we looked before, we have three types of layers:

[7]In reality there exist some feedforward neural network whitout hidden layer, like the single layer perceptron, or with only one, but the multi-layers networks are the most common.

1. **Input layer:** provides information from the outside world to the network. No computation is performed.

2. **Hidden layer:** perfoms computations and transfer information from the input nodes to the output one. An ANN usually have more than one hidden layer.

3. **Output layer:** transfers information from the network to the outside world. Unlike the other layers, they usually do not have an activation function, this is because the last output layer is often used to represent class scores which are numbers with real values.

One of the main reason for why the neural networks are organized into layers is that this structure makes them very simple and allow to evaluate them efficiently using matrix vector operations. In a feedforward network the information moves in only one direction, obviously in forward, from the input nodes to the output ones passing to the hidden layers. In this process, for every node in the hidden layer, we get the weighted sum of the input and we plug the result into the activation function. This output is passed at the next layer. At the end, after the output layer, we have a predection. This propagation is a generalization of the work of a single node shown previously. It is important to note that there are not cycles or loops in the network which instead can be found in the recurrent neural networks in which the connections between the nodes form a cycle.

Now that we have understood what the various layers of a neural network do, we can focus on how a MLP is trained and so it learns. The learning process takes the name of backpropagation algorithm, also said BackProp. It is a supervised algorithm, which means that it learns from labeled training data, therefore learning is guided. In simple words the BackProp is like learning from mistake. The supervisor, that guides the training, corrects the ANN whenever it makes mistakes. These corrections happen going to modify the value of the weights associated on the connections between the nodes of adjacent layers. The goal of learning is to assign the correct weight for every edges. We have to remember that given an input vector, these weights determine what the output vector is and that, in a supervised algorithm for the training set, we know the expected label. Now let's

look at the backpropagation algorithm in more detail. Initially all the connections weights are randomly assigned. For every input in the training phase, the ANN is activated and its output is observed. This output is compared with the desired one and the error is propagated back to the previous layer. This error is noted and the weights are adjusted accordingly. The most common methods to adjust the weights are the gradient descent method, the SGD and the Adam method. According to Wikipedia, the gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. It is based on the observation that if the multi-variable function $F(x)$ is defined and diffentiable in a neighborhood of a point a, then $F(X)$ decreases if one goes from a in the direction of the negative gradient of F at a, that is $-\nabla F(a)$. Following that, if $a_{n+1} = a_n - \gamma \nabla F(a_n)$ then $F(a_n) \geq F(a_{n+1})$, where γ is the size of steps taken to reach the minimum and it is known as learning rate. If it is too high we can cover more ground each step, but we risk overshooting the lowest point since the slope of the hill is constantly changing. If it is too low we can confidently move in the direction of the negative gradient since we are recalculating it so frequently. It is a common practice to use a decadent learning rate, which means that we can start with a high rate to speed up the first steps to then move on a lower rate for having a better precision. The Stochastic Gradient Descent (SGD) is a variant of gradient descent algorithm, but instead of performing computations on the whole dataset, which are redundant and inefficient, it only computes on a small subset or random selection of data examples. At the end, Adam is an algorithm for gradient-based optimization of stochastic objectie functions. It combines the advantages of two SGD extensions, knew as Root Mean Square Propagation and Adaptive Gradient Algorithm, and computes individual adaptive learning rates for different parameters[8]. All three optimization algorithms aim to find the best weights to minimize the Mean Square Error (MSE) between the true output and the predicted one. This process is repeated until the output error is below a predetermined threshold. Once time that this process is finished the artificial neural network is ready to work with new inputs. This ANN had learned from several examples, the labeled data, and from its mistakes, the

[8]It will be taken up and deepened in the chapter 3: *Training and result*, section: *The training phase.*

error propagation. 1.4.

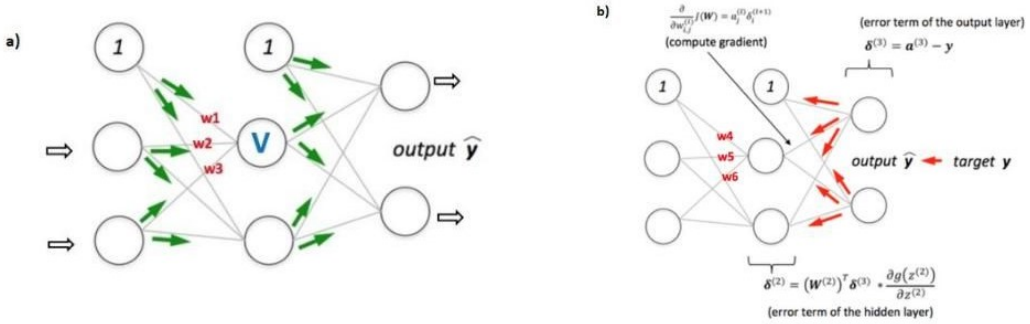

Figure 1.4. The picture a) shows the forward step. The picture b) shows the backpropagation with the weights adjustament.

Before concluding this section it is important to note one more thing. It is that neural network with at least one hidden layer are universal approximators. This means that given any continuous function $f(x)$ and some $\epsilon > 0$, there exists a neural network $g(x)$ with one hidden layer and a reasonable choice of non-linearity such that $\forall x, |f(x) - g(x)| < \epsilon$. In other words, ANN can approximate any continuous function[9]. This could raise the question of why multiple layers are used if one hidden layer is sufficient to approximate any function. As shown in the Stanford's notes cited in the footnote n.4 of this section, the answer is that the fact that a two-layer neural network is a universal approximator is, while mathematically cute, a relatively weak and useless statement in practice. In one dimension $\sum_i c_i 1(a_i < x < b_i)$ where a, b, c are parameter vectors is also a universal approximator, but noone would suggest that we use this functional form in machine learning. Indeed, neural networks work well in practice because they compactly express nice, smooth functions that fit well with the statistical properties of data we encounter in practice, and they are also easy to train using our optimization algorithms, for example gradient descent. Similarly, the fact that deeper networks

[9]According with the free ebook Neural Networks and Deep Learning, link: http://neuralnetworksanddeeplearning.com/chap4.html.

with multiple hidden layers can work better than a single-hidden-layer networks is an empirical observation, despite the fact that their representational power is equal.

1.3 Convolutional neural network

For the task of the image recognition, it is usual to use a special class of deep neural networks known as Convolutional Neural Networks (CNNs) or more simply ConvNets. They were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. In this process, in fact, the single cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. Subsequently, the receptive fields of different neurons partially overlap in such a way as to cover the entire visual field. CNNs are regularized versions of multi-layer perceptrons that are usually fully connected, i.e. each neuron in a layer is connected to all node in the next one. This could be a problem. We consider an image, it is nothing but a matrix of pixel values. So we can flatten the image transforming the image matrix into a vector and pass it to a multi-layer perceptron for our classification purposes. Taking a simple image 200×200, this means that we need 40 thousand hidden units and we have to calculate almost 2 bilion parameters. This requires enormous computing and memory capabilities and, besides being a waste of resources, it may cause overfitting. The latter occurs when a model with high capacity fits the noise in the data instead of the underlying relationship. If we also consider that in a image recognition problem we have many pictures, it easy to understand that is impossible to rely on this approach. The intuition behind the CNNs is to take advantage of the spatial and temporal dependecies in a image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to undestand the sophistication of the image better.[10] Always considering the example of the

[10] According with Towards Data Science, link: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

image 200×200, if we apply a filter with size 10×10 we have to calculate only 4 milion parameters, if we apply 100 filters the parameters to learn go down to 10 thousand. They are only the 0.0005% of the initial ones. 1.5.

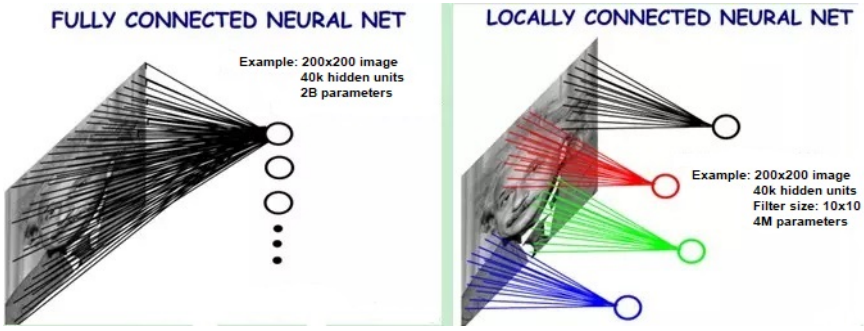

Figure 1.5. The picture on the left shows how a neuron fully-connected is connected with an input image. The picture on the right shows how filters in a ConvNet are connected with an input image.

Convolutional neural networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. They use relatively little pre-processing compared to other image classification models. This means that the CNNs learn the filters that in traditional algorithms were designed by hand. This independence from prior knowledge and human effort in designing features is a great advantage. In particular, unlike a regular neural network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height and depth. It is important to note that the depth in this case refers to the third dimension of an activation volume, not to the depth of the full neural network, which can refer to the total number of layers in a network. In addition the neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner.[11] As we described above, a CNN is a sequence of layers, and every layer of a ConvNet trasforms the image volume into an output one through a differentiable function. The three main type of layers used to build the CNN architectures are the convolutional, the pooling and the fully-connected. We now look the individual layers in the details.

[11] According with Convolutional Neural Networks: Architectures, Convolution/Pooling Layers (Stanford CNN Tutorial), link: https://cs231n.github.io/convolutional-networks/.

1. **Convolutional layer.** It is the core building block of a ConvNet that performs most of the computational operations and its objective is to extract the high-level features from the input image. Conventionally, the first convolutional layer is responsible to capture the low-level features such as edges, color, gradient orientation. With added layers, the architecture adapts to the high-level features as well, giving us a network which has the understanding of images in the dataset, similar to how we would. It is usually indicated with CONV. Its parameters are a set of learnable filters, also known as kernels. Everyone is small spatially (along width and height), but extends through the full depth of the input volume. During the forward step, we convolve each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we produce a $2-$dimensional activation map that gives the responses of that filter at every spatial position. At the end of this process, we have an entire set of filters in each CONV layer and each of them produce a separate $2-$dimensional activation map. We have to stack these activation maps along the depth dimension and produce the output volume. As for the backward step, for a convolutional operation it is also a convolution, but with spatially-flipped filters. At the moment we have discussed about the connectivity of each neuron in the CNN to the input volume, but we have not yet discussed how many neurons there are in the output one or how they are arranged. There are three hyperparameters that control the size of the output volume: the depth, the stride and the zero-padding. The first controls the number of neurons in a layer that connect to the same region of the input volume. The stride indicates how to move the filter. When it is 1 the filter slides one pixel at a time. This leads to overlapping receptive fields between the columns and also to big output volumes. When it is 2 the filter moves 2 pixels at a time, while strides ≥ 3 are rare. The zero-padding allows us to pad the input with zeros on the border of the input volumes. Sometimes this is convenient because we can manage better the stride and we can preserve the spatial size of the input volume. We can compute the spatial size of the output volume as a function of the input volume size W, the receptive field size of the CONV neurons F, the stride

that are applied S and the amount of zero-padding uses P. The number of neurons that "fit" follow the formula $\frac{W-F+2P}{S}+1$. We have to pay attention to the fact that the number resulting from this formula can not be integer. As we said before, through the control of the stride and the zero-padding we can avoid the problem. 1.6.

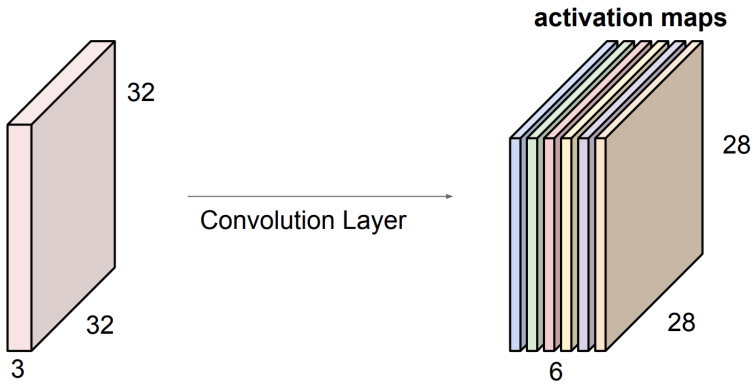

Figure 1.6. We have a $32 \times 32 \times 3$ image and suppose to have multiple filters (6) along the depth, looking at the same region in the input, everyone with a 5×5 dimension, a stride of 1 and a zero-padding of 0. We will have an output of size [28,28,6]

2. **Pooling layer.** It is common practice insert a pooling layer in-between successive convolutional layers in a ConvNet architecture. It is usually indicated with POOL and it is a form of non-linear down-sampling which serves to reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The pooling layer operates independently on every depth slice of the input and resizes it spatially. There are several non-linear functions to implement pooling among which max pooling and average pooling are the most common. The second one was often used in the past but has recently fallen out of favor compared to the max pooling operation, which has been shown to work better in practice. As for the backpropagation we have to keep in mind that the backward pass for a $\max(x, y)$ operation has a simple interpretation as only routing the gradient to the input that had the highest value in the forward step. Hence in the forward step of a pooling layer it is common

practice to keep track of the index of the max activation, in this way the gradient routing turns out to be efficient during backpropagation. The most common form is a pooling layer with filters of size 2×2 with a stride of 2. This allows to partition, in this case, the input volume into small 2×2 regions which non-overlapping themselves and to take the max over 4 numbers. The depth dimension remains unchanged while the width and height change, for example if we use the common filter described previously they are halved. The pooling layer requires two hyperparameters: the spatial extent F and the stride S. The pooling size must not have to be too larger because it risks to be destructive. The intuitive idea behind this type of layer is that the exact location of a feature is less important than its rough location relative to other features. 1.7.

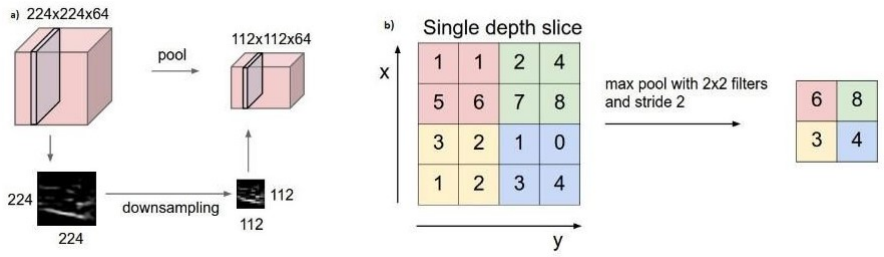

Figure 1.7. The image a) shows that applying a pooling filter the depth is preserved, while the width and height are halved. The picture b) shows how a max pooling filter works.

For completeness, we also say that in literature we find many people dislike the pooling operation and think that we can get away without it. For example, Springenberg et al. [2014] proposes a new architecture where the pooling layers are replaced by a convolutional layers.

3. **Fully-connected layer.** After several convolutional and pooling layers, the high-level reasoning in the neural network is done via fully-connected layers which could be indicated by FC. Neurons in a FC have connections to all activations in the previous layer, as seen in non-convolutional ANNs. Their activations can thus be computed as an affine trasformation, with matrix

multiplication followed by a bias offset.

In addition to these three layer it is common write the RELU activation function as a layer. As seeing before, this function is useful to introduce and increase non-linearity properties of elements. What remains to be seen is how to stack together the layers to form entire ConvNets. The most common pattern of a CNN architecture stacks a few CONV-RELU layers, follows them with POOL ones, and repeats this scheme until the image has been merged spatially to a small size. After some steps like these, it is common to transition to FC. The last fully-connected layer holds the output, such as the class scores. 1.8.

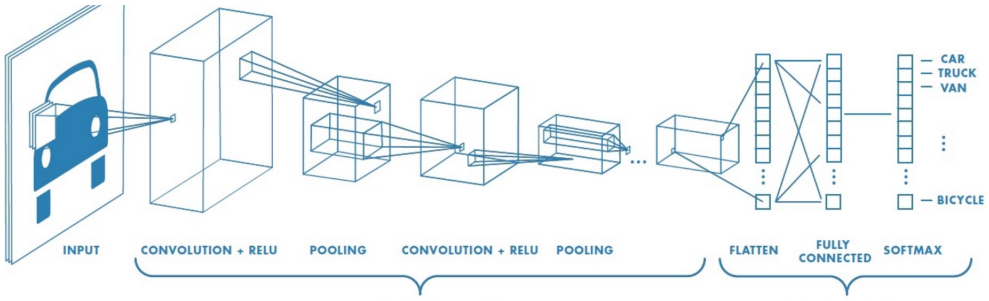

Figure 1.8. The picture shows a simple example of CNN architecture.

Chapter 2

Architecture and Datasets

2.1 ResNet

For training a convolutional neural network from scratch we need a huge amount of data that is often not available in the practical cases. For this reason it is common practice to take a pretrained ConvNet on a very large dataset, and then use it either as an initialization or a fixed feature extractor for the task of our interest. This machine learning's technique takes the name of transfer learning. *"Transfer learning and domain adaptation refer to the situation where what has been learned in one setting ... is exploited to improve generalization in another setting"* Bengio et al. [2017]. The idea behind it is to optimize the learning obtained in one task to improve, in term of speed and accuracy, the learning in another task. There are two different principal scenarios of transfer learning:

1. **ConvNet as fixed feature extractor.** This technique consist about take a convolutional neural network pretrained on a very large dataset and replace the last fully-connected layer with a newone where the number of output layers is equals to the number of different classes the we have to classify. The only level to train is the latter, while the rest of the ConvNet is treated like a fixed feature extractor for the new dataset.

2. **Fine-tuning the ConvNet.** This second strategy expects instead starting from a convolutional neural network pretrained on an appropriate dataset

to fine-tune the weights by continuing the backpropragation. It's possible to fine-tune all the layers of the ConvNet, more rare, or only a part, more common. The reason is that the earlier features of a ConvNet contain more generic feauters (e.g. shape detectors) that can be useful to many tasks, while the next levels become more specific to the details of the classes contained in the original dataset.

Both strategies are valid and for choosing the better we must consider the size of the new dataset and the similarity to the original one. In addition, we have to keep in mind that ConvNet features are more generic in early layers and more original-dataset specific in later ones. In our study case the new datasets were quite small and different from the original one (ImageNet, which contains 1.2 milion images with 1000 categories). Since the data are small and different it is likely best to only train a linear classifier, so it is preferable work with the fixed feautures extractor technique.

Once we have understood the best strategy, we need to find which is the pre-trained model that works better for the image classification problem. Indeed there are many differents models: AlexNet, VCG, ResNet, SqueezeNet, Inception, GoogLeNet, ShuffleNet, MobileNet and others. Let's see briefly some characteristics of the main convolutional neural networks[1].

1. **AlexNet.** It was one of the first deep networks work that popularized ConvNet in computer vision. It has been developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton and it is composed by 5 convolutional layers followed by 3 fully connected ones. For the non-linear part it uses the ReLU function, instead of tanh or Sigmoid which was the earlier standard for traditional neural networks. Moreover, this architecture has reduced the overfitting by using a dropout layer[2] after every fully connected layer. It was submitted

[1] According with CV-Tricks.com, link: https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/.

[2] This type of layer will be discussed in detail later.

to the ImageNet ILSVRC challenge[3] in 2012 and it significantly outperformed the second runner-up.

2. **VGGNet.** This architecture has been developed by Karen Simonyan and Andrew Zisserman and it is an improvement over AlexNet where large kernel size filters have been replaced with multiple 3×3 kernel size filters. Multiple stacked smaller size kernels are better than the one with a larger size because multiple non-linear layers increase the depth of the network which enables it to learn more complex features, and that too a lower cost. The VGG convolutional layers are followed by 3 fully connected layers. Furthermore, it has shown that the depth of the network plays a critical role. It arrived second to the ILSVRC 2014.

3. **GoogLeNet/Inception.** GoogLeNet has been the ILSVRC 2014 winner and it has been implemented by some Google researchers. Its main contribution was the development of a module called Inception which dramatically reduced the number of parameters in the network. The researchers have started to the idea that a convolutional layer with 3×3 kernel size which takes for example 512 channels as input returns an output of 512 channels, so every output channel is connected to every input one, and we have a dense connection architecture. However, most of this activations in a deep network are either unnecessary, because the value is zero, or redundant, because of correlations between them. Therefore the most efficient architecture of a deep network will have a sparse connection between the activations. There are techniques to prune out such connections which would result in a sparse weight/connection. Additionally, they also have replaced the fully-connected layers at the top of the ConvNet with a simple gloabl average pooling. This allows the elimination of a large amount of parameters that do not seem to matter much.

4. **SqueezeNet.** It has been released in 2016 by University of California, Berkeley, Stanford and DeepScale that is a private U.S. company. The goal of the

[3]It is a challenge which evaluates algorithms for object detection and image classification at large scale.

authors was to create a smaller neural network with fewer parameters that can more easily fit into computer memory and can more easily be transmitted over a computer network. It has been presentend in a paper entitled *"SqueezeNet: AlexNet-level accuracy with* $50x$ *fewer parameters and* $< 0.5MB$ *model size"*, but it is important to note that SqueezeNet is not an improvement of AlexNet, but rather a new different deep neural network. What they have in common is that both of them achieve approximately the same level of accuracy when evaluated on the ImageNet image classification validation dataset.

5. **ShuffleNet/MobileNet.** Both of them architectures have been proposed in 2017 with the purpose to achieve good performances for mobile device which have very limited computing power compared to the computer.

6. **ResNet.** It has been the winner of ILSVRC 2015. Since it is the architecture that we have chosen to use, it will be explain in detail later.

To choose the best pretrained model we have to focus our attention on a list of performance measures which are:

1. **Top-1 Error.** It occurs if the class predicted by a model with highest confidence is not the same as the true class. Since the image classification problem is rather difficult, there is another useful error measure to consider called top-5 error.

2. **Top-5 Error.** It occurs when the true class is not among the top 5 classes predicted by the model (sorted in terms of confidence).

3. **Inference Time on CPU.** It is the time taken for model inference step.

4. **Inference Time on GPU.** It is the time taken for model inference step, but this time using the GPU. Although it was designed for mainly as dedicated graphics rendering of computer games, it is nowadays increasingly used in deep learning problems.

5. **Model size.** It is the physical size occupied by the pretrained model.

A good model should have these parameters as low as possible. We start to compare the accuracy of the model, comparing the top-1 and the top-5 error[4]. 2.1.

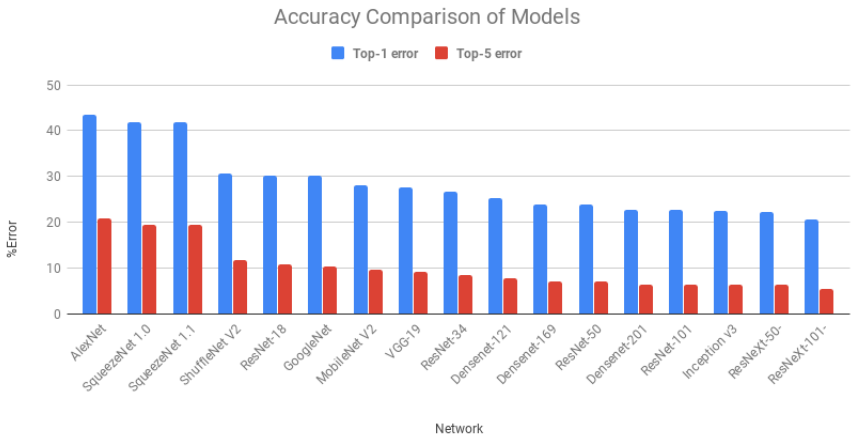

Figure 2.1. The graph shows the top-1 and the top-5 error of the different models.

It is possible to notice the trand of the top errors is similar. The best performance are achieved by GoogLeNet, ResNet, VGGNet and ResNext.

Next we will compare the inference time on CPU and GPU. One image was supplied to each model multiple times and the inference time for all iterations was averaged. Similar process was performed for CPU and GPU on Google Colab. The following two histograms shows the trends. 2.2. 2.3. For both graphs the best performances are achieved by AlexNet, SqueezeNet, ShuffleNet and ResNet. Moreover we can see how the performance provided using the GPU is definitely better than the respective using the CPU. This is due to the fact that although the GPU has a smaller number of cores than the CPU, their more logical basic design allows them to process a simpler and more identical set of calculations in parallel.

[4]These results and graphs, as the follow, can be found at the link: https://www.learnopencv.com/pytorch-for-beginners-image-classification-using-pre-trained-models/ a blog managed by Satya Mallick, an entrepreneur with a Ph.D. at the University of California, San Diego.

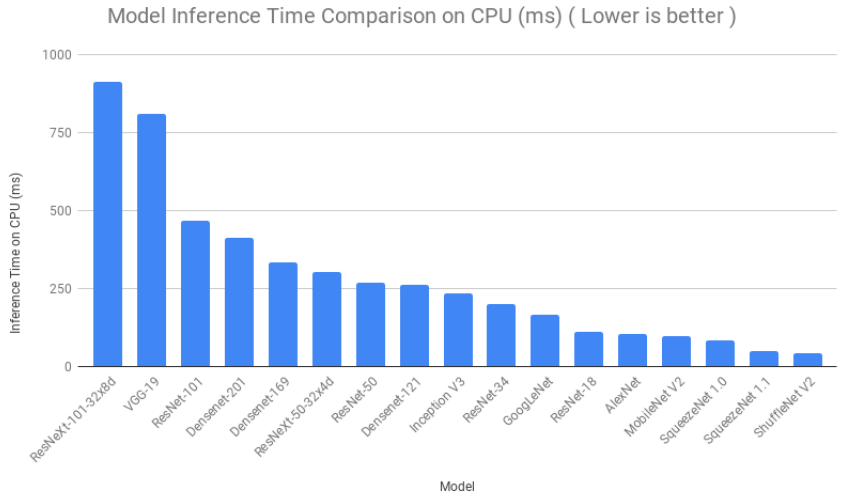

Figure 2.2. The graph shows the time taken for model inference step using the CPU.

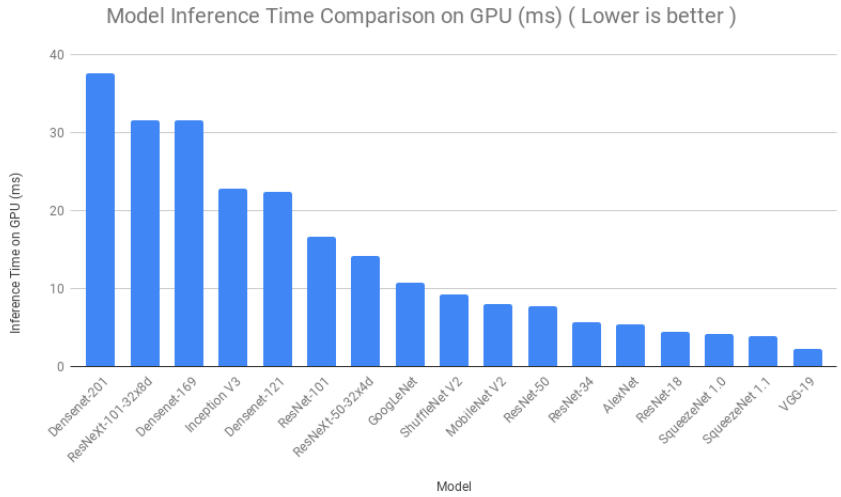

Figure 2.3. The graph shows the time taken for model inference step using the GPU.

Precisely this property of expressing simple parallel processing operations using code which for the most part concerns matrix and vector operations, that are the basic mathematics of data science, has allowed the GPU to spread through non-graphical calculations. However it cannot be definitively affirmed that using the

latter is always better than the CPU. As noted above Oracle Data Science Blog[5] there are tradeoffs to consider, between speed, reliability, and cost. As a general rule, GPUs are a safer bet for fast machine learning because, at its heart, data science model training is composed of simple matrix math calculations, the speed of which can be greatly enhanced if the computations can be carried out in parallel.

At the end, we have to compare the model size. The dimensions in MB of the different models are shown in the histogram below. 2.4.

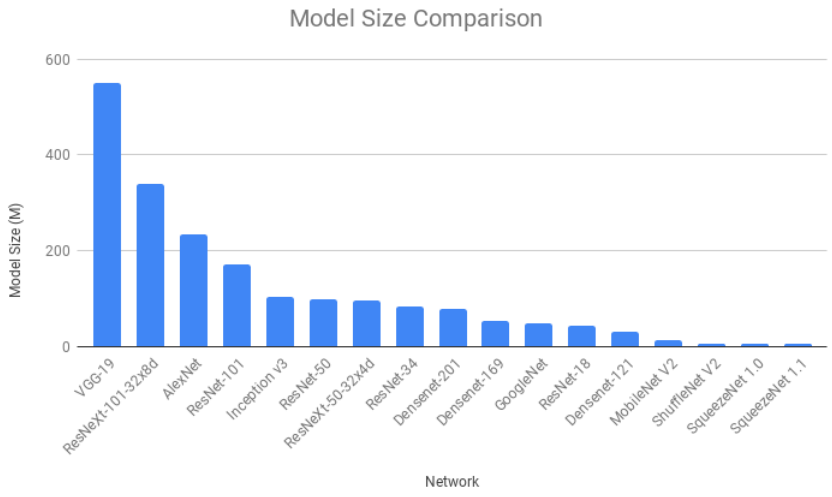

Figure 2.4. The model size (MB) of the models.

In this case the best results are achieved by SqueezeNet, ShuffleNet and MobileNet.

For choosing the best model we could squeeze all those criteria in one bubble chart which shows us the best model based on our requirements. 2.5. The x-axis represents the top-1 error, while the y-axis is the inference time on GPU in milliseconds. At the end the bubble size shows the model size. We have to find a good compromise between all the characteristics, keeping in mind that smaller bubbles are better in terms of size and bubbles near the origin are better in term of both

[5]Link: https://blogs.oracle.com/datascience/cpu-vs-gpu-in-machine-learning.

accuracy and speed. It is clear from the bubble chart that the best choise is the pretrained model ResNet50.

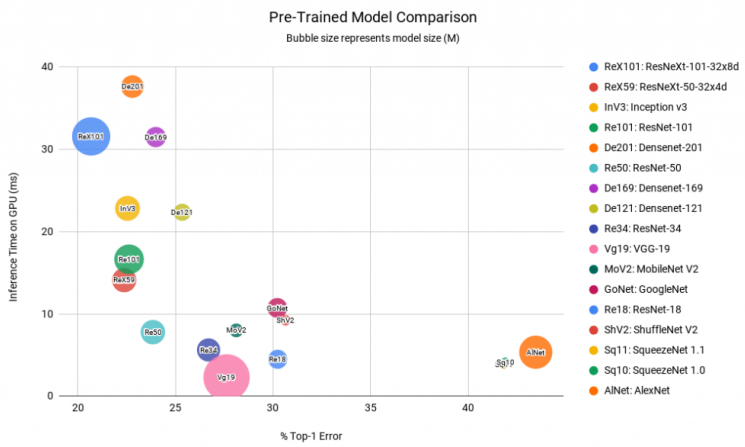

Figure 2.5. The bubble chart takes into consideration the top-1 error, the inference time on GPU and the model size.

Deep convolutional neural networks have led to a series of breakthroughs for image classification task. So, over the years the idea was that increasing the depth should increase the accuracy of the network, as long as overfitting is taken care of. Nevertheless the problem with increased depth is that the signal required to change the weights, which arises from the end of the network by comparing ground-truth and prediction becomes very small at the earlier layers, because of increased depth. It essentially means that earlier layers are almost negligible learned and this issue is called vanishing gradient problem. The second drawback with training deeper networks is, performing the optimization on huge parameter space and therefore naively adding the layers leading to higher training error. An optimal solution has been proposed in He et al. [2016] by a Microsoft Research group, that introduce the ResNet model. ResNet is a short name for Residual Network. These four researchers have tried to solve the problems using a deep residual learning framework. Instead of learning a direct mapping of x $\longmapsto$ y with a function $H(x)$, they proposed to use a residual function $F(x) = H(x) - x$, which can be reframed into $H(x) = F(x) + x$, where $F(x)$ and x represens the residual mapping and the identity fuction respectively. There are two kinds of residual

connections:

1. The identity shortcuts x can be directly add when the input has the same dimension of the output:

$$y = F(x, \{W_i\}) + x$$

where $F(x, \{W_i\})$ represent the residual mapping to be learned.

2. If x and F do not have the same size, it can perform a linear projection W_s by the shortcut connections to match the dimensions:

$$y = F(x, \{W_i\}) + W_s x$$

2.6.

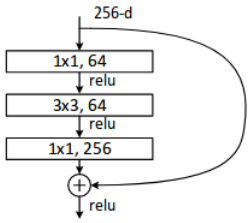

Figure 2.6. A 3 layers block in a ResNet50.

The article shows that the ResNet converges faster compared to plain counter part of it (without the residual learning) and in this case, increase the depth can really improve the accuracy of the model (obviously untill a limit, for avoiding overfitting). The same authors propose the ResNet architecture with different depth levels. The following image, in which the structure chosen by us was highlighted, is taken precisely from the article and shows the different architectures compared to each other. 2.7. The architecture is simple to read. It shows the filter size, the number of filters and the number of times which the structure is repeated. The zero-padding is not indicated instead also when there is and the stride is shown only in the first two layer. For example the first convolutional layer has 64 filters with kernel size of 7×7, stride 2 and zero-padding of 3. Instead the third

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2.7. Architectures of different depth of ResNet.

layer has 128 filters with kernel size of 1×1, stride 1 and no zero-padding, 128 filters with kernel size of 3×3, stride 2 and zero-padding of 1, 512 filters with kernel size of 1×1, stride 1 and no zero-padding and this structure is repeated 4 times. Furthermore it is possible to notice that all the ResNets end with an average pool following by a fully-connected level composed by 1000 neurons where the predictions about the labels have done with the softmax function[6]. On the left of the table is reported the output size of every layer while at the end is possible note the number of FLOPs that the specific architecture requires. FLOPs means FLoating point Operations Per Seconds and in computer science, it indicates the number of floating point operations performed in one second by the CPU.

In our study case we have develop more neural networks starting from a ResNet50. In some cases we had to implement a ternary CNN, while in the others the neural networks were a binary. The difference between the them it is only in the last line, in accordance with what we said before, because it is different the number of outputs that we required. We can see the shared part in the figure 2.8. The last fully-connected layer that has said it is composed by 1000 neurons it has been replaced by three new levels. Between one level and another it is always possible

[6]The softmax function is a function used to map a vector K of real numbers into a probability distribution consisting of K probabilities proportional to the exponentials of the input number.

```
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, num_ftrs)
model_conv.fc_bn = nn.BatchNorm2d(num_ftrs)
model_conv.dropout=nn.Dropout(0.5,inplace=True)
model_conv.fc1 = nn.Linear(num_ftrs, num_ftrs)
model_conv.fc1_bn = nn.BatchNorm2d(num_ftrs)
model_conv.dropout2 = nn.Dropout(0.4,inplace=True)
model_conv.fc2 = nn.Linear(num_ftrs, num_ftrs)
model_conv.fc2_bn = nn.BatchNorm2d(num_ftrs)
model_conv.dropout3 = nn.Dropout(0.4,inplace=True)
```

Figure 2.8. Code shared between the four neural networks.

to observe the presence of two line; the first performs the Batch normalization, while the secondone performs the dropout. They are two of the most common and useful techniques in the deep learning. Let's see what they are.

1. **Batch normalization:** according to Wikipedia it is a tool for improving the speed, performance and stability of artificial neural networks and it was introduced in 2015 paper Ioffe and Szegedy [2015] . Since any training algorithm usually works better on normalized data, it is used to normalize the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. Furthermore it reduce overfitting because it has a slight regularization effect. Similar to dropout, it adds some noise to each hidden layer's activations.

2. **Dropout:** is a regularization technique that helps avoiding co-adaptation of neurons and thus reduce overfitting. The co-adaptation effect occurs when two or more neurons begin to detect the same feauture repeatedly, this means the network is not utilising its full capacity efficiently. To avoid this problem the tool drops-out some neurons at training time, while in the test time there are all. The percentage of neurons that are turned off in the first phase is chosen arbitrarily and is indicated with a probability p that randomly determines whether a given neuron is on or off.

Like said before, there is another line yet, that represents a new fully-connected level, where the output number of features is equals to the number of classes that we need to classify.

2.2 The datasets

As previously stated in the summary, our final goal is to develop an application capable of predicting the state of an electrical switch starting from one of his photo taken. To do this, a preliminary but fundamental work it has been to create the datasets useful to train the neural networks indeed, as seen in the previous chapter, they are one of the most strong tool for doing this task. For this first step we went to the power plant and a pair of markers has been attached for each switch chosen as study case and more photos have been taken of the different possible states. As markers, different options were considered including: AZTEC code, ArUco markers and QR codes. We have choosen the latter, let's see why briefly. For the first type, the online documentation is minimal and mainly concerns the generation of them, it was therefore difficult to implement an efficient reader. As regards the ArUco, they can only register numbers inside but we had to include different information, so they were not suitable for our case. The QR codes were instead ideal for our case as they could contain within them different information and the libraries that implement the detector give excellent results, for these reasons we have selected them. The library that on Python was found to be more effective for their treatment is pyzbar which returns a list for each of them containing the position in the image and the information contained within. After collecting the photos, we had to implement the machine learning algorithm which is the base of our project. Its operation is as follows: it receives a picture in input, detects the couple of QR Codes, reads the information contained in them, recognizes the area that they delimit and cut out the photo independently. Usually the cropping of the image is done including the QR codes so as not to risk eliminating important parts if the photo was not taken frontally, however in some cases it has been necessary to perform different cuts. This is due to the fact that in certain situations the differences between states of the switches are minimal and it would have been too difficult for the network to learn the features that distinguish them. The QR codes has been made in way that every couple has unique information, in this way the mapping between the switch and his type was easy and there was not problem if there were more than two markers per photo. Furthermore, being the orientation of the switch really important in our task, we have always attached the QR codes in

way to have one on the top-left side of photo and the other in the botton-right. In this way the machine learning algorithm is able to automatically rotate the input picture if it does not arrive with the right orientation. In a first phase, the photos that are not cut by the algorithm are been cropped manually to have a greater number, moreover we have collected a set of statistics regarding the average height and width both of QR codes both of cropping area, for each type. This collection of information has been very important, in fact, starting from them, we were able to calculate two average factors for every typology of switch useful for cutting out a photo if only one QR code is detected. Specifically, the two factors are an average ratio between the height (width) of all the cropped images and the height (width) of all the respective QR codes identified in them, always for each type of switch. In this way, from this moment, if a photo is taken in which only one QR code is detected, knowing the type to which it belongs, these two factors can be used to carry out the cut correctly. If, for example, it is only revealed the top-left code, it is sufficient to multiply the height and width of the revealed QR code by the respective factors and what is obtained and the area to be cut out. The same applies if the code is found at the bottom right. This trick allowed not to take into consideration only very few photos, those in which not even a QR code is revealed and which are usually of very poor quality and they would not have been useful in training of the neural network. 2.9.

Figure 2.9. The photo on the left shows an example of image that the algorithm receives in input. The photo on the right shows the respective output.

Once the images are collected and processed as mentioned above, being the tipology of the switches totally different between them, it had been necessary to

make distinct datasets. Moreover, as already said before these differences between the switches made it necessary the implementation of different cutting functions inside the algorithm which are automatically called knowing the tipology of the switch. In this way the CNNs have been trained in the best way. All the datasets at the base of the convolutional neural networks have been splitted in a training set and a test set with the proportion $70\% - 30\%$ respectevely. This means that for every switch and for every state of its the corresponding photos are splitted following these percentages.

The first dataset is composed by 432 photos and the switch can be in three different states: on, off and extract. The difference between off and extract is that in the first case the switch is off, but the current continues to pass, in the second case the current is also interrupted. The dataset is quite balanced, indeed for the first state there are 170 photos, for the second one 163 and for the last 99. We will indicate this typology as type 1. 2.10.

Figure 2.10. Examples of photos in the first dataset. From left to right it is possible see the switch in the state: on, off, extract.

The second dataset is composed by 2260 images and also in this case the switch can be in three states: on, off and extract. However, unlike before, it is not necessary to implement a convolutional neural network able to recognize also the extracted state. Indeed, when the switch is in this state, a usually hidden QR code is visible, with different information within the usual two, and therefore the recognition can be done directly by the machine learning algorithm. For this reason the photos of this state are only 100 and they are not be taken into consideration in the following. Regarding the other two states, also in this case the dataset is

balanced, indeed for the state on there are 1108 photos, while for the state off the picture are 1052. As can be seen, in the second case the differences between states are less evident than in the first type presented. For this reason, as anticipated previously, the photos have been cropped with a different cut that excludes the QR Codes, in this way the neural network can be focused better on the relevant features. This typology will be indicated as type 2. 2.11.

Figure 2.11. Examples of photos in the second dataset. On the left, the top photo shows when the switch is on while the bottom one shows when it is off. On the right an image of when the switch is extracted.

As for the last type of switch studied which we will indicate as type 3, the situation is a bit complicated. Indeed for the prediction of the state it is necessary combined two different predictions, one deriving from a binary ConvNet and one deriving from a ternary one, so it is composed by two different typologies each with different states. From the first one we can have on or off, from the second one the output is one among: inserted, extracted, secured where inserted means that the switch is on, extracted means that it is off and secured means that the current is really detached. The final output is therefore one of the six possible combinations. Given the particular structure of the problem it was necessary to construct two different datasets, one by type. The first typology is called window and counts 2801 photos of which 1394 for the state on and 1407 for the other. The second tipology takes the name of panel and contains 2101 images of which 254 for the state inserted, 307 for the state extracted and 1540 for the state secured. As we can see, this last dataset is really unbalanced and therefore it has been

necessary to do oversampling. It is a techniques used in data analysis to adjust the class distribution of a dataset which can be implemented in different ways. We have choosen to use the random oversampling. It involves supplementing the training data with multiple copies of some of the minority classes. This is one of the earliest proposed methods, that is also proven to be robust. Specifically, once the largest class has been divided up with the percentages given at the beginning of the section, this technique has been applied to the less numerous classes, which led to having an equal number of images in the training set. 2.12.

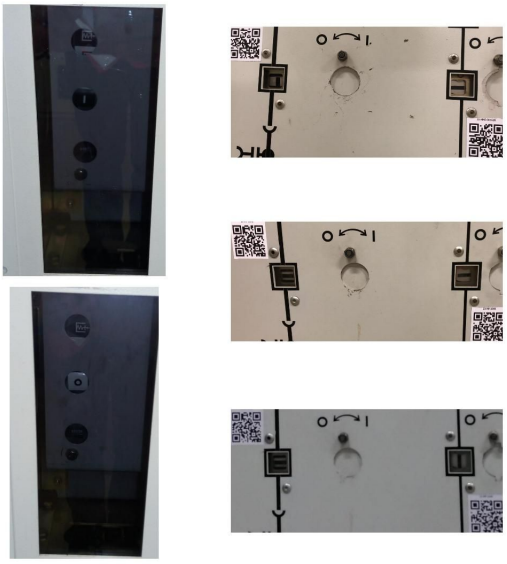

Figure 2.12. Examples of photos of windows and panels. On the left from top to bottom: on, off. On the right from top to bottom: inserted, extracted, secured.

In these cases the differences from the states both for the window and for the panel are even less obvious, indeed for the first type we classify as on if the white dash with a black background is present, and as off if the black circle on a white background is present. For the panel the prediction depends on the orientation of the two dashes. It is evident that in these situations just excluding QR codes from the cut is not enough, for this reason these types of switches have been treated in a totally different way. Indeed, recourse were made to the object detection.

A last observation before starting to talk about the training phase and the

results is that a dataset with more than 2000 photos may seem large, but we have to keep in mind that usually the neural networks have trained on milions of images to be efficient.

Chapter 3

Training and results

3.1 Object detection

Before focusing on how convolutional neural networks have been trained and seeing what results they brought, we focus our attention on object detection. As mentioned above it was used only for the window and panel types and it is fundamental because its output is the input of the ConvNets. Let's see what it is, how it works and how we have applied it in our case.

According with Wikipedia[1], object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class in digital images and videos. So, an object detection model can identify which of a known set of objects might be present and provide information about their positions within the image. To be precise, if the detection has a positive outcome, an image is supplied in output that contains one or more boxes in correspondence with the detected objects and for every of them a confidence percentage and the coordinates of location are associated. It is important to note that the object detection can be hundred of times slower than image classification and therefore, it is useful and should be used only if this last one proves to be inadequate for the problem. To be able to have a model

[1]Link: https://en.wikipedia.org/wiki/Object_detection.

that works this way we must first train it on a set of images. There are different methods to implement an object detection and therefore different way to train it. They can be split into two categories: machine learning-based approaches or deep learning-based approaches. For the first one it becomes necessary to previously define features that characterize the objects of interest and then using a technique such as support vector machine to do the classification. On the other hand, deep learning techniques that are able to do end-to-end object detection without specifically defining features, they are typically based on convolutional neural networks. These last ones are the most widespread and we have choosen to use them. Furthermore, in addition to these different basic methods, there are various techniques/architectures that can be used to implement an object detector. Let's see below what are the main ones and what was their evolution[2].

1. **Sliding window approach.** It was the approach for detecting objects most used in the classical computer vision techniques for object detection in which a sliding window is moved over the image and all the pixels inside it are cropped out and sent to an image classifier. If the image classifier identifies a known object, the bounding box and the class label are stored, otherwise the next window is evaluated. This approach is computationally really expensive to detect objects in an input image, in fact we have to try the sliding windows at different scales and several aspect ratios have to be evaluated at every pixel in the image. For this reason this approach is now used only in special cases.

2. **R-CNN object detector.** As we have seen in the previous chapters the convolutional neural networks are the tool most used nowadays for the image classification task thanks to their excellent performance. For this reason being every object detector an image classifier at its heart, the development of a CNN based object detector became inevitable. However using the ConvNets, the detector remains slow and computationally very expensive also it allows to build a multi-class object detector that can handle different aspect ratios in

[2]Further details can be found at the links:
https://www.learnopencv.com/faster-r-cnn-object-detection-with-pytorch/ and
https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/.

addition to various scales. To avoid these problems have been introduced the Region-based Convolutional Neural Network (R-CNN) which use an object proposal algorithm called Selective Search. The idea behind this technique is to train a machine learning model that could propose locations of building boxes that contained objects. These bounding boxes are called region (or object) proposals and they are merely lists of bounding boxes with a small probability of containing an object, but they do not know or care which object is contained in the bounding box. Thanks to the Selective Search the region proposal algorithm outputs a list of a few hundred boxes ($\simeq 2000$) at different location, scales and aspect ratios where most of these do not contain any objects. This method is competitive because to evaluate the image classifier in a few hundred bounding boxes generated by this technique is much cheaper than evaluating it in hundreds of thousands or even milions of bounding boxes as the case of the sliding window approach. However when this method was proposed by Girshick et al. [2014] the speed of execution was still too slow, $18 - 20$ seconds per image on a GPU. For this reason, starting to this model, in subsequent years a new architecture has been developed and it takes the name of fast R-CNN.

3. **Fast R-CNN object detector.** Contrary to what happens into a R-CNN, where for every region proposal it is calculated a feature map and each bounding box is independently classified by the image classifier, the idea behind the fast R-CNN architecture is to calculate a single feature map for the entire image. In particular, for each region proposal a region of interest (RoI) pooling layer extractes a fixed-length feature vector from the feature map. Each of them is used for two purposes: classify the region into one of the classes and improve the accuracy of the original bounding box using a bounding box regressor. A further improvement was achieved with the faster R-CNN.

4. **Faster R-CNN object detector.** The slowest part in fast R-CNN is the Selective Search. Faster R-CNN replaces it with a very small convolutional neural network called Region Proposal Network (RPN) to generate regions of interest. This technique is similar to the one just described but, while in the previous method even though the computation for classifying the region

proposals was shared the part of the algorithm that generates them, it did not share any computation with the part that performes image classification. In this improvement instead, the two parts can use the same feature map and therefore share the computational load. A convolutional neural network is used to generate a feature map of the image which is simultaneously used for training a region proposal network and an image classifier. It is this sharing that increases the speed of execution of the classifier.

5. **Single Shot Detector (SSD).** It is the method we have adopted which shares common aspects with the previous one and that, also if in terms of precision it does not reach the faster R-CNN results, it gains in speed. Unlike the R-CNNs it runs a convolutional neural network on input image only once and calculates a feature map. In order to handle the scale, SSD predicts bounding boxes after multiple convolutional layers. Since each convolutional layer operates at a different scale, it is able to detect objects of various scales. This algorithm is usually recommended unless we need exceptional accuracy performance.

Now that we have seen the different object detection techniques/architectures we can create our object detectors but, before doing that, we need of the data that will be used for training. To train a robust detector, we need a lot of pictures which should differ a lot from each other. In our case the main differences come from the device that takes the photo, the angle and the different lighting conditions. Following the indications of the guide *"TensorFlow Object Detection API tutorial"*[3], we have splitted the datasets with the ratio of $90\% - 10\%$, i.e. 90% of the images are used for training and the rest 10% is maintained for testing. It is important to note that this not be in contrast with what was said previously, where the division of them was affirmed with the ratio of $70\% - 30\%$. In fact the datasets on which CNNs are trained and tested always follow that distribution; this new division, always made on the same images, is useful for training the object detectors. So, different datasets are then created. The next step consists into label the images and therefore we need some kind of image label software. We have used LabelImg

[3]Link: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html.

that is a great tool for labeling pictures which allows to create the bounding boxes directly on the image and to annotate them. Contrary to what one might think we must do this procedure not only for the training but also for the test set. Labeling generates a set of new .xml files, one for each image, that have to be converted before into a csv file and later into a TFRecords file which serve as input data for training of the object detector. The last thing we need to do before training is to create a label map and a training configuration file. The first one serves to map every box that have an id to a name and it consists in a simply text file. About the second, like the neural networks, we have two choices: to create a training job from scratch or to use one of the pretrained models provided by TensorFlow. Also in this case, since the images available to us were not many, we have chosen to start from a model already trained, precisally the SSD Inception v2 pretrained on COCO dataset which is a large-scale object detection, segmentation and captioning dataset that counts more than 330 thousands images and 1.5 million object instances. We chose this model because it provides a relatively good trade-off between performance and speed. They are measured respectively in terms of mAP[4], a popular metric in measuring the accuracy of object detectors which compute the average precision value for recall value over 0 to 1, and in terms of running time in milliseconds (ms) per 600×600 image. We have to note that this last one depends highly on one's specific hardware configuration and should be treated more as relative timining in many case. Along with the configuration file we also download the latest pretrained neural network for the model. At this point we can start the training that can be considered finished when the loss function stabilizes at a value between 2 and 1. We have to note that, obviously, lower value of the loss function is better, however very low value should be avoided as the model may end up overfitting the dataset and therefore to perform poorly when applied to images outside the dataset. Both detectors have required $6 - 8$ hours each to be trained using the GPU. A last thing to notice is that it is a common practice use a threshold value below which we can discard detection results. We have opted for a value of 0,65 that is quite hight and it allows to contrast the false positive at the

[4]It stands for mean Average Precision.

expense of false negative. The firsts one are objects that are wrongly identified, or areas of the image which are erroneously identified as object when they are not; the second ones are genuine objects that are missed because their confidence was low.

As anticipated at the beginning of the section, in our case we have developed two different object detectors, one for the window and one for the panel, but their operation is almost identical. Given an input image, if the detection is successful, what we are doing is to take advantage of the indications of the box that revealed the object and that always appear in an array format composed of four numbers representing a bounding rectangle that surrounds its position. They are ordered as follows:

[top, left, bottom, right]

where the top value represents the distance of the rectangle's top edge from the top of the image, in pixels and the left value represents the left edge's distance from the left of the input image. The other values represent the bottom and right edges in a similar manner. What remains to be done is to crop this box and feed it to the appropriate convolutional network. The difference between the two detection, in addition of course to what they have to look for which as we have seen is totally different from one typology to another, is the fact that for the window, if everything went well, three boxes always come back, while for the panel two must return. The fact that object detection must give exactly this number of outgoing boxes is an advantage for us because it allows us to verify the correctness of the output through the use of a control value. In the case of failure in the detection we have set an error message that requires a better quality photo, usually it is enough that it is less blurred. For the detection in the window, in the event of a positive result in the survey, the boxes are arranged vertically and we have to cut the one in the middle; instead regarding the panel we need both boxes to make the prediction and it was arbitrarily chosen to always send the left box to the neural network and then the right one. The final prevision on the state of the switch is obtained by combining the predictions made on these three boxes. 3.1.

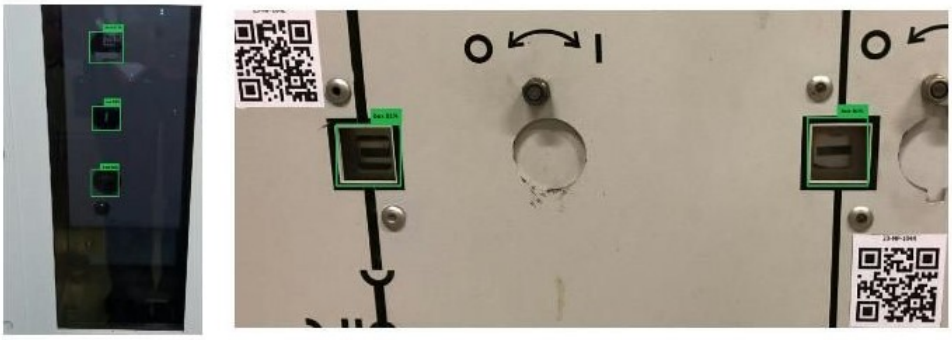

Figure 3.1. Two examples of object detection correct: on the left there is the window while on the right there is the panel.

The evaluation of the performances of an object detector is not trivial indeed there are two distinct tasks to measure: determining whether an object exists in the image, that is a classification problem, and determing the location of it, that is a localization issue. About the first task, we can associate a confidence score with each bounding box detected and to assess the model at various level of confidence. To have a more robust classifier we have imposed a threshold value of 0.65 that the confidence score have to exceed to be considered reliable. A measure that responds to our requests is the Average Precision (AP). To understand it, it is necessary to introduce the concepts of precision and recall of a classifier.

1. **Precision.** It measures how accurate is our predictions, i.e. it serves for calculating the proportion of positive identifications that are actually correct.

2. **Recall.** It measures how good we find all the positives, i.e. it serves for calculating the proportion of actual positives are identified correctly.

It relates the two measures by going to take the average value of the precision across all recall values. As for instead the task of object localization, we must first determine how well the model predicted the location of the object. This can be done using the metric known as Intersection over Union (IuO). It evaluates the overlap between two boundaries indeed it measures how much our predicted boundary overlaps with the ground truth, that is the real object boundary made during the manual labeling. Sometimes it is defined an IoU threshold in classifying

which establishes wheter the prediction is a true positive or a false positive, this means that the prediction can be considered correct if it exceeds this value. We have imposed it at 0.5. Now that we've defined Average Precision and seen how the IoU affects it, we can move on to using the mAP, previously introduced, which is calculated by taking the mean AP over all classes and/or the IoU threshold value. In our case the detector of the window has achieved a mAP of 0.9, while the panel one is 1.

3.2 The training phase

As said in the previous chapter, the choice of the neural network fell on a ResNet50 to which they were added four new fully-connected level. However, before starting the training phase we have to fix some parameters: number of epoches, optimizer, learning rate and loss function.

1. **Number of epoches.** 1 epoch corresponds to 1 forward pass plus 1 backward pass for all training sample. Therefore to fix this number means to define the number times that the ConvNet will work through the entire training dataset. It is important that the total number of epoches is not too low to allow the model to reach its maximum predictive level and is not too high to avoid overfitting. In our case it was enough to set this number to 25.

2. **Optimizer.** It is an optimization algorithm which helps us to minimize the loss function. This kind of algorithm can be of two types: of the first order (like gradient descent) or second order, less used than the first one but with the advantage of not neglect or ignore the curvature of surface. We had opted for using a first order algorithm that take the name of Adam[5] optimizer which is an improvement of the stochastic gradient descent. It is a method that computes adaptive learning rate for each parameter. In addition to storing an exponentially decaying average of past squared gradients, Adam also keeps an exponentially decaying average of past gradients. Then it uses estimations of

[5]Adam stands for adaptive moment estimation.

first and second moments of gradient to adapt the learning rate for each weight of the neural network, where with moment of grade n we mean the expected value of the variable to the power of n, $m_n = E[X^n]$. The first moment is the mean, and the second moment is the uncentered variance of the gradients. Adam works well in practice and compares favorably to the other adaptive learning-method algorithms as it converges really fast and the learning speed of the model is quiet fast and efficient. It also rectifies every problem that is faced in other optimization techniques such as vanishing learning rate, slow convergence or high variance in the parameter updates which leads to fluctuating loss function[6].

3. **Learning rate.** This hyperparameter controls how much to change the model in response to the estimated error each time the model weights are updated. If it is too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process. As seen in chapter 1 during the explanation of the gradient method and the stochastic gradient method a good solution is to start with a wider learning rate in the first steps, then move on to a lower value below to get a better solution. Following this way we had opted for a learning rate of 0.001 that decayed by a factor of 0.1 every 7 epoches.

4. **Loss function.** It is important for the neural network because it is used to measure the inconsistency between predicted value and actual label. Our choice is relapsed on the cross entropy loss. It combines two criterion, the LogSoftmax (which is the log of the softmax function) and the NLLLoss (which is the negative likelihood loss). The main advantage is that it avoids the problem of learning slowing down, indeed the rate at which the weights learn is controlled by the error in the output. The cross entropy loss increases as the predicted probability diverges from the actual label, so the larger the error, the faster the neural network will learn. The cross entropy loss is useful

[6]According with Towards Data Science, link: https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f.

when training a classification problems with C classes.

The CNNs had been trained using the servers of Google Colab that is the free cloud service made available by Google. The most important feature that distinguishes Colab from other cloud service is that it provides GPU and it is totally free. As we saw before, the GPU is important because it allows to greatly accelerate the required calculations. Before starting with the training of the CNNs some transformations have been applied on the data both to standardize the inputs of the ConvNets both to have the phenomenon known as data augmentation. The latter is essential for every state-of-the-art result in image classification and for this reason its use is now common. It consists in the creation of altered copies of each instance within a training dataset. It is useful because it allows to increase the size of the of labeled training set and this facilitates the finetuing of the parameters. When we feed image data into a neural network, there are some features of the images that we would like the neural network to be able to recognize and learn. These features are the pixels which make up the object in the image. On the other hand, there are features of the picture that we would not like the neural network takes into consideration because they are noise. They are the pixels which form the background in the image. A very simple solution which ensure that the neural network is able to differentiate relevant features from noise is to create multiple alterations of each image, where the object in the picture is kept invariant, whilst the background is distorted. There are two different way to perform the data augmentation on the dataset. The first one is to apply all the transformations beforehand, essentially increasing the size of the dataset. The second one is to perform these transformations on a mini-batch, just before feeding it to the machine learning model. The first option is known as offline augmentation and it is preferred for relatively smaller datasets, as we would end up increasing the size of the dataset by a factor equal to the number of transformations that we performed. The second option is known as online augmentation. This method is prefered for larger datasets, as we can not afford the explosive increase in size. In this case we perform transformations on the mini-batches that we would feed to our model. The offline augmentation is the solution that we have chosen. These are all the transformations that we had applied on the training set and they are presented with the PyTorch language:

1. **RandomHorizontalFlip(p).** It flips horizontally the given PIL image[7] randomly with a given probability p.

2. **Resize($size$).** It resizes the input PIL image to the given size. If $size$ is a sequence like (h, w), output size will be matched to this. This transform is useful because generally images in the dataset have differing size, therefore they have to be resized before being used as input to the model.

3. **CenterCrop($size$).** It crops the given PIL image at the center. If the input is a single parameter it makes a square crop $(size, size)$, oterwhise if we pass a two parameters a rectangular crop $(height, width)$ is made. It allows to focus on the main elements and to neglect the rest.

4. **ToTensor().** It converts a PIL image to tensor. A tensor is a generalization of vectors and matrices and is easily understood as a multidimensional array. It is the the basic element for the CNNs which deal with images recognition.

5. **Normalize($mean, std$).** It normalizes a tensor image with mean and standard deviation. Given mean: $(m_1, \ldots, m_n)$ and standard deviation: $(s_1, \ldots, s_n)$ for n channels, this transform will normalize each channel of the input. In this way the data are of the same scale.

As previously anticipated, these transformations are applied only on the training set because they are useful not only to standardize the inputs that the model requires but also to improve the number of samples from which to learn, while through the test set we only want to evaluate the quality of our model. For this reason, the only transformations which are applied on the latter are: Resize($size$), ToTensor() and Normalize($mean, std$) which do not modify the size of the set.

All the models have given good results in terms of accuracy and in terms of confidence and they have requested a training time of between 25 and 40 minutes. The reason for this difference depends on the size of the training sets, that as we have seen vary a lot depending on the type of switch. However, at the beginning

[7]It is a format to open e manipulate images in Python.

we had some problems. The main one has been that the first collection of images was made by taking pictures at the same electrical switch with different devices and at different angles. This has given problems because, being always the same identical subject, it occurred overfitting. It has been possible to realize it thanks to the use of an explanatory model, but this will be deepened in the next dedicated section. The overfitting problem has been solved by collecting new images from different switches of the same category and retraining the models[8]. Let's start talking about accuracy. It is simply a metric for evaluating classification models and it can be defined as the fraction between the number of correct predictions and the total number of predictions. The results on the datasets have been similar. The accuracy for the type 1 has reached 100%, while for the others it went very close, indeed it reached 99.2% for type 2, 97.7% for the window and 98.7% for the panel. In all cases this level has been achieved after a low number of epochs, usually between 5 and 7. However accuracy alone, if it does not reach 100%, is not able to tell us all the useful information. Indeed, when we make a mistake we would also like to see how wrong it is, going to see the number of false positives and false negatives that have occurred. Proper to better understand the correctness of the results we have printed the confusion matrices that report the predictions on the test set. 3.2. It is possible to note that the confusion matrix of the window is ternary also if we have said that the ConvNet is binary. There is a column called N/A that we have used to indicate the cases in which the recognition failed because the object detection model did not identify the section on which to apply the classification model. In the panel's matrix the column N/A there is not present because we have always been able to recognize the boxes on which to make the recognition. Starting from the results shown by the four confusion matrices, we are able to calculate other two important performance measures of our models known as precision and recall which we have already introduced in the section on object detection. These measures are important for a classifier because the mistakes that we make are not all the same, but some weigh more than others. In our case, for example, if we consider the panel typology, to say that a switch

[8]The results, as well as the data reported in chapter: *Architecture and Datasets* section: *The datasets*, already refer to this image update.

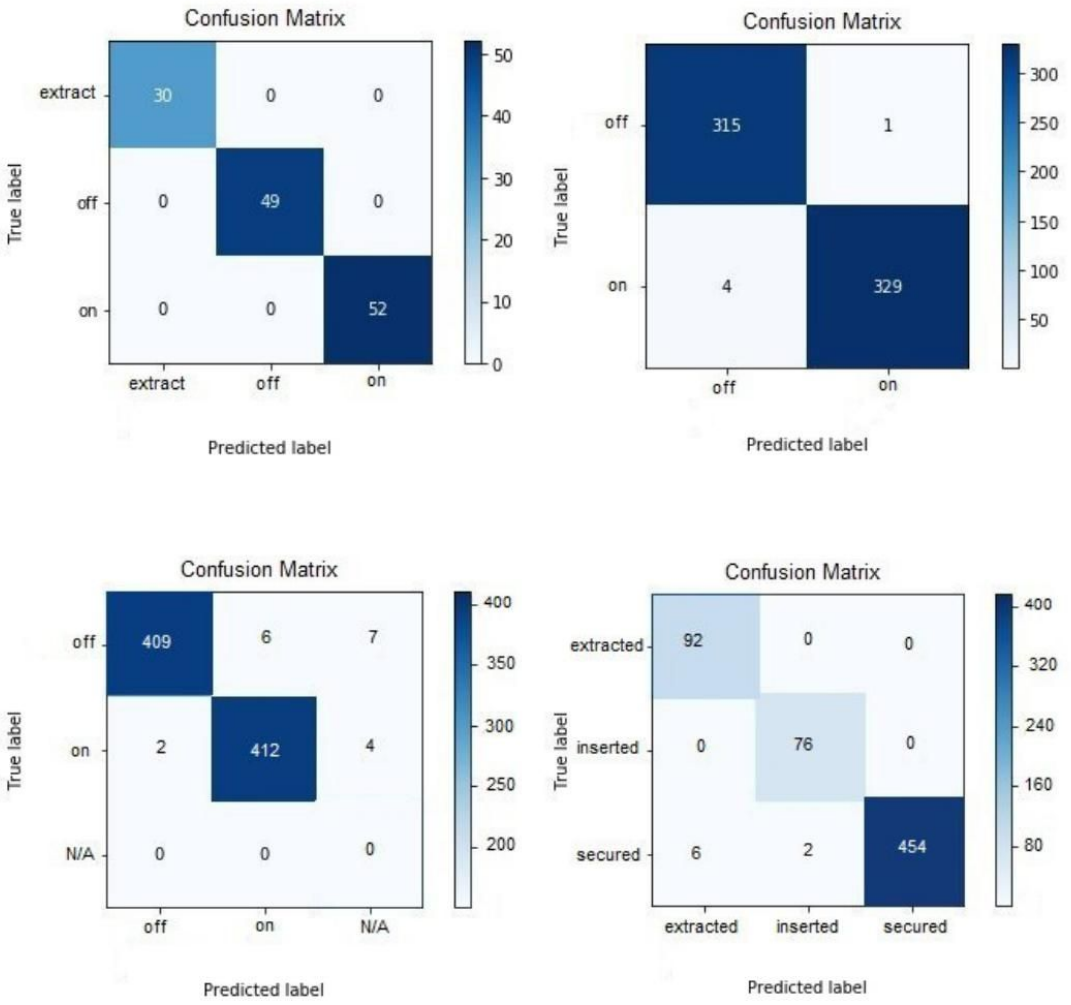

Figure 3.2. Starting from top left and moving by line, the confusion matrix of: type 1, type 2, window and panel.

is secured when it is not, it is a much more serious error than to say that it is inserted when it is extracted. Being the matrix of the type 1 without mistakes they have been calculate only for the others. Usually in the binary cases[9], there

[9]We have considered the typology window as binary, not considering the row and the column N/A.

exists the convention to read it as:

$$\begin{bmatrix} \text{true positive} & \text{false positive} \\ \text{false negative} & \text{true negative} \end{bmatrix}$$

while for the typology panel where the matrix is ternary we have to calculate the value of true positive, true negative, false positive and false negative individually for each class. For example for the state extracted they are respectively: 92, 530 $(76 + 454)$, 6, 0. As said before, the precision serves for calculating the proportion of positive identifications that are actually correct and it is defined as

$$\frac{TP}{TP + FP}$$

where TP and FP stand for true positive and false positive respectively. Instead the recall, as already mentioned previously, serves for calculating the proportion of actual positives are identified correctly and mathematically it is defined as

$$\frac{TP}{TP + FN}$$

where FN stands for false negative. We can see in the following table the results achieved from the different ConvNets. 3.1

Typology-State	Precision	Recall
Type 2	99.68%	98.75%
Window	99.55%	99.51%
Panel-extracted	93.88%	100%
Panel-inserted	97.44%	100%
Panel-secured	100%	98.27%

Table 3.1. The table shows the result in terms of precision and recall for every ConvNet and in the ternary case also for every state.

Another measure to take into consideration is the confidence. It is important because it says how reliable the estimates of the predictions are. To calculate it we

have written a function in Python that compares the true label with the outgoing one returning the corresponding probability evaluated with the softmax function. Also in this case the results have been good, the average confidence was found to be over 90% for all the CNNs. More specifically it has reached 96.43% for the type 1, 94.37% for the type 2, 91.02% for the window and 93.33% for the panel. Furthermore we have kept track of when a low confidence happened in way to verify the cause. Fortunately, it is a situation which has happened rarely and we were able to conclude that the main reason was the low quality of the single photo. Leaving aside for a moment our specific case and giving a more general look at the discourse of confidence, as shown in Guo et al. [2017] modern networks as ResNet tend to be over-confident. They discovered that recent neural networks, unlike those from a decade ago, are poorly calibrated and this depends from factors like depth, width, weight decay and Batch normalization. By poorly calibrated they mean that the difference between the accuracy and the average confidence is not negligible. This discrepancy is also called miscalibration. It is important because a model without it helps the users to better interpret the predictions of the neural network. The most interpretable cause of the miscalibration is the increase of capacity and the cross-entropy loss. Model capacity can be understood as a measurement of how much a model can memorize. If we were able to have an infinite capacity, the model could learn by heart the whole training set. It is necessary a trade-off between low and high capacity indeed if it is too low the model would not be able to learn essential features of the data, on the other hand if it is too high the model will learn too much and overfit instead of generalize. A right capacity allows the model to pick up the most representative features and will then better generalize. The architecture of ResNet have way more capacity than the older traditional neural networks and this led to better accuracy, the training set can almost be learned by heart. Regarding the cross-entropy loss, the model tries to optimize it and this force it to be right and to be very confident. Moreover, the authors have shown that using Batch normalization improves the training convergence and the final performance but also increases the miscalibration, but they are not able to explain the reason why. Opposite using the weight decay, which is an additional loss that penalizez the $L2$ norm of the weights and avoids to find extreme values that could make overfitting, the model accuracy decreases but also decrease the

miscalibration. This happens because regularization avoids overfitting and thus over-confidence. They propose different solutions to fix miscalibration that are all post-processing method[10], including a new one that takes the name of temperature scaling. Instead of compunting the softmax in the traditional way:

$$softmax(x_i) = \frac{e^{y_i}}{\sum_j^N e^{y_j}}$$

is better to use:

$$softmax(x_i) = \frac{e^{\frac{y_i}{T}}}{\sum_j^N e^{\frac{y_j}{T}}}$$

where T is called temperatue and $T > 1$. This temperature softens the probabilities indeed extreme probabilities (high confidence) are more decreased than smaller probabilities (low confidence). Another positive thing of temperature scaling is tha all logits are divided by the same value, and the softmax is a monotone function, so the accuracy remains unchanged. However back to our study case, we have not taken into consideration this change. In fact it must be considered that the over-confidence is usually only a few percentage points more than the real value, therefore to counter this problem we have introduced a threshold value more high. It is useful because if the confidence of the forecast does not pass this value a warning message appears to signal that the neural network is not sure about the result and so it is not possible make a prediction. This additional constraint allows to have an answer only when the confidence is high and in this way is possible to limit the over-confidence. The threshold values are different for each network. Indeed, for the binary cases it has been set at 0.65 while for the ternary ones the value is 0.45. These values avoid the problem of having predictions in doubt, in which the neural network foresees at random.

Once the convolutional neural networks have been trained and we have tested their goodness on the test set, we have saved the model in a .pth file. This allowed us to download the neural network models and to integrate them into an API

[10]This means that each method takes the form of an additional model that corrects the calibration error of an original model. Fitting the calibration model is a distinct, second step, done only after the original model is fully trained.

which also includes the machine learning algorithm described previously. API is the acronym for Application Programming Interface and, according to Wikipedia, it is a communication protocol between a client and a server intended to simplify the building of client-side software. It is a simple way to manage the requests made by a client for a server and the returned data in response. Our API requires in input: a photo of the switch and the code of the QR Codes. The code inside the markers is unique and starting from that it is possible to go back to the type to which the switch belongs. Once this information is also obtained, the machine learning algorithm is simply called and after it has operated as seen in the previous chapter the correct neural network is queried, which returns the forecast or an error/warning message if something went wrong. Furthermore, at this point it has been possible to evaluate the total time of execution of the application, from when it receives a photo to when it provides a prediction. In fact it is important that this time is not too long otherwise the user experience would not be good and the application itself would not be very useful. Also in this case the results have been satisfactory, indeed the API required on average $2 - 3$ seconds to make a status prediction for types 1 and 2, while it required about $10 - 12$ for type 3. This last time is however reasonable if we think that two images are processed and each of them requires in addition to a prediction of a convolutional neural network, the use of an object detector. As last thing, we have to stress the importance of having the whole process within an API. The main advantage of having everything within a single API where multiple codes interact with each other is that if in the future we will want add a new type of switch it will be sufficient to add only the tools to predict the latter. For example it could be enough to create a new convolutional neural network and, once trained, to add the .pth file in the API and to specify the corresponding path.

3.3 The explanatory model

In addition to having trained the convolutional neural networks and to having integrated all in the API we have developed another model, the explanatory one. This kind of models are useful to describe why and how a thing works or to explain why a phenomenon is the way it is. As shown in Ribeiro et al. [2016a] separating

the explanations from the machine learning model has some advantages. The approach which they propose is that of model-agnostic, i.e. to extract post-hoc explanations by treating the original model as a black box. This involves learning an interpretable model on the predictions of the black box one, perturbing inputs and seeing how it reacts, or both. The interpretability methods are just a layer between the prediction of the machine learning models and the information learned by the humans. The main advantages of a model-agnostic explanation system are:

1. **Model flexibility.** The interpretation method is able to work with any machine learning model, such as random forests and deep neural network.

2. **Explanation flexibility.** It means that we are not limited to a certain form of explanation. In some situation it might be useful to have a linear formula, in other cases a graphic with feature importances.

3. **Representation flexibility.** The explanation system should be able to use a different feature representation as the model being explained.

The algorithm behind our model is known by the name LIME and it has been proposed in Ribeiro et al. [2016b]. LIME is the acronym of Local Interpretable Model-agnostic Explanation and it is a part of the local surrogate models, that is a class of interpretable models that are used to explain the individual predictions of machine learning ones. The overall goal of LIME is to identify an interpretable model over the interpretable representation that is locally faithful to the classifier, while our goal is to understand why the machine learning one made a certain prediction. LIME tests what happens to the predictions when we effect a perturbation on the data into the machine learning method. The algorithm creates a new dataset consisting of permuted samples and the corresponding predictions of the model. The new dataset is the training set for an interpretable model, which is weighted by the proximity of the sampled instances to the instance of interest. We have to note that the interpretable model can be anything, from a linear regression to a logist one, from a GLM to a decision tree. The learned model should be a good approximation of the machine learning one predictions locally, but it does not have to be good globally. This kind of accuracy is know as local fidelity. Mathematically, local surrogate models with interpretability constraint

can be expressed as follows:

$$\xi(x) = \arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

where $\xi(x)$ represents the explanation produced by the surrogate model. Into details, the explanation model for instance x is the model g, for example a linear regression one, that minimize loss L, for example the mean square error, which measures how close the explanation is to the prediction of the original model f, while the its complexity $\Omega(g)$ is kept low, in fact is preferable have fewer features. G is the family of possible explanations, for example all possible linear regression model. The proximity measure π_x defines how large the neighborhood around instance x is that we consider for the explanation Molnar [2018]. In substance, LIME aims to optimize the loss part, while the user has to determine the complexity, for example selecting the maximum number of features that the model g have to use. A low number of features K is useful because it simplifies the interpretation of the model. Moreover we can understand the principal features that lead to the prediction. On the other hand, a higher K can produce models with higher fidelity. There are several ways to train models with exactly K features. A good choice is with Lasso[11]. Another strategies are forward or backward selection of features. It means that we can start with the full model which contains all features or with a model with only one and then test wich feature would bring the biggest improvement when added or removed, unitil a model with K features is reached. In cases like ours where we handle with images, this last one approach is recommended.

LIME for image works in a rather particular way. Indeed, it would not have much sense to perturb individual pixel of a picture, since many more than one pixel contribute to one class. Randomly changing indidual pixels would probably not change the predictions of the classifier. For this reason, it is more convenient

[11]It is a common way to introduce sparsity into the linear regression model. When it is applied in a linear regression model, it performs feature selection and regularization of the selected feature weights, penalizing the larger ones.

to connect and join pixels which are similar between them into sets of pixels called superpixels. The variations of the image are created by segmenting it into these sets which are turning off or on where turning off means that pixels with similar colors have been replacing with a user-defined color such as gray.

This method, besides being useful for all that has already been said, allows also to understand if there is overfitting and how it affects the prediction. 3.3.

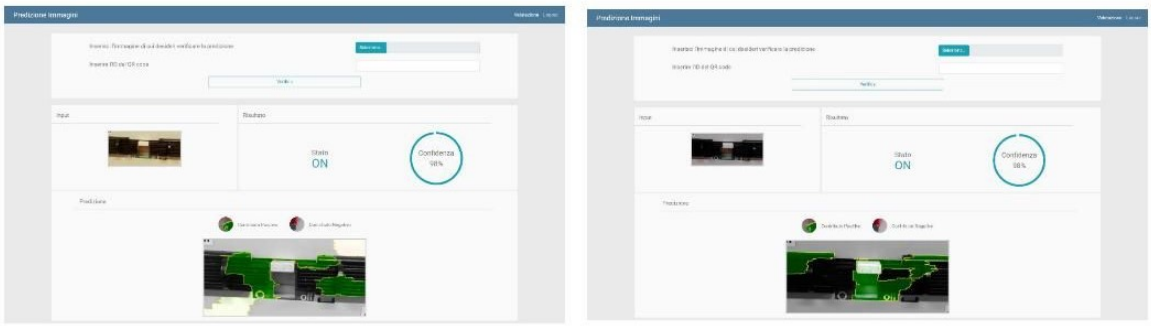

Figure 3.3. Two output examples of the explanatory model.

The photo above shows two output examples of the explanatory model, where the number of features K has been fixed on 7. The model that we have implemented is quite simple and it has been incorporated into an internal API since the information it provides does not serve the customer but only us to understand if a good job has been done. It requires in input an image that it is already cropped and the code which is present in the QR codes. Once this information has been provided the model is able to interact with the API described previously, indeed it go back to the correct convolutional neural network and query it on the perturbed image, so as to obtain a prediction. As said previously, at the beginning we have had the overfitting problem. This has shown in the picture on the left. Indeed, it is possible to notice that the main contribute to make the prediction is given by contour elements of the switch which are present in all the photos, while the lever that would be the real indicator of the state of the switch is not considered in any way. However the confidence is high and the prediction is right.This can just be explained through overfitting, it is likely that the neural network has learned

the image by heart. Instead, on the right, there is an output of the explanatory model after we had took other images to retrain the CNNs. This time is possible to notice that the positive contribution derives mainly from the lever and from the space occupied by it. There is still a part of the contribution due to parts that should not actually influence the prediction, but it is very likely that continuing to collect images and retraining the models later this problem disappers.

Chapter 4

Conclusion

Having come to the end of the exposition of what has been done during the internship and the related parts of the theory it is appropriate to get some final conclusions. First of all it is important to say that the service has satisfied the client who decided to continue the collaboration with Blue Reply and the project has therefore gone into production, that is the API developed has been integrated into the application already owned by the company. At the same time, the customer's firm has submited new types of switches to study and predict, in fact if the results will continue to be positive as those seen previously, in the end the application will be able to predict the status of 20 different types of switches. However, it has to be considered that these three types of switches that we have studied in this first phase of the project can be considered among a level simple and intermediate as regards the difficulty of the prediction and therefore, it will be hard to expect such high performances also in the continuation in which there will be switches with a higher prediction's difficulty level. Instead, an advantage for this second phase and for all the subsequent ones, is that this time we already have the general architecture, so all we should do will be integrate the existing API with the new tools developed. A thing really important to note is that for each new type of switch for which we want to make a prediction of the state through artificial intelligence algorithms it is necessary and fundamental to understand if it is actually possible to do so. First of all it is necessary to go to understand where and how to place the QR codes, in some cases it is not said that they can

be attacked as explained and done before indeed there may be physical constraints due for example walls, switch disposition or other factors. Moreover, as we saw in the thesis, sometimes it is not possible to solve the problem by developing directly a convolutional neural network because it can happen that the differences between one state and another are minimal and therefore even developing deeper networks we would not be able to make them reliable anyway. This was for example the case of type 3, in which we resorted to object detection in order to solve the problem. On other occasions, however, even this solution may not be sufficient, so precisely in order to be able to face up and solve the most disparate situations it is important to stay up to date as much as possible on the tools of artificial intelligence, which are always in continuous evolution, and on how they can interact and complete each other.

Still with the same customer company, other possibilities for collaboration are being evaluated, not only regarding the prediction of electrical switches. A project on which you are thinking, and which always concerns the increase in safety at work, is to develop an application that can understand in real time if a worker is equipped with adequate work equipment, in particular: helmet and jacket. Also for this task the best solution can be achieved using the artifical intelligence's tools, in particular in this case the object detection seems be the best choice. Obviously in this situation the development of the application should be totally different than previously seen, because the work that it have to do and its final goal are completely different, but this is the proof of how these techniques are increasingly important and more widespread for solving of the most varied problems nowadays.

Finally this thesis work has also tried to give a general overview of neural networks from a theoretical point of view, focusing in particular on convolutional ones and specifically on the architecture known as ResNet. This is important because, in order to best apply these and other tools of the artificial intelligence, it has necessary to know how they work, their strengths and weaknesses, if they can be useful for one task or not. What it has been shown in this case is how a convolutional neural network works, which factors are to be taken into account when we have to choose an architecture rather than another and how and which parameters

influence the learning of models. Furthermore, some of the most widespread measures to evaluate the performance of predictive models were shown, in particular the results obtained were exposed in terms of accuracy, confidence, precision and recall. In addition to these, it has been explained the importance of developing an explanatory model. As shown, this last one helps to better understand why a certain prediction is made, what factors influence it and consequently to understand if it is reliable or not. In particular, as seen, the explanatory model can be very useful to investigate the possible presence of the overfitting phenomenon.

Bibliography

Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. Citeseer, 2017.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1321–1330. JMLR. org, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Christoph Molnar. Interpretable machine learning. *A Guide for Making Black Box Models Explainable*, 7, 2018.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*, 2016a.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd*

ACM SIGKDD international conference on knowledge discovery and data mining, pages 1135–1144. ACM, 2016b.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.