POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

Robust place categorization with deep self supervised methods

Supervisor
prof. Barbara Caputo

Candidates
Valerio Sofi
ID number: 248415

December 2019

Abstract

Deep learning in the last years has allowed to reach very high performances in the task of image classification in Computer Vision, through the widely used Convolutional Neural Networks. While the traditional algorithms are very accurate when the test samples are similar to the training data, their performances are significantly lower when test and training data have a different visual appearence and belong to different domains. This work introduces a solution to this problem in the context of scene recognition, starting from the recently introduced JiGen approach that has shown very good improvements in the object recognition context, and shows that with some variations also in this context the accuracy can be improved, both in Domain Generalization and Domain Adaptation settings. The experiments are performed on the COLD and the BDD100K datasets.

Acknowledgements

In this section I want to thank all the people who helped me in these years and who supported me, both for the studies and for the development of this work. Since not all of them are comfortable with English, this part will be in Italian in order to be easily understood from everyone.

Vorrei ringraziare prima di tutto la mia famiglia, che ogni giorno mi supporta e che non mi ha mai fatto mancare nulla, permettendomi così di seguire la strada che più mi piaceva senza alcun intoppo.

Ringrazio poi molto la professoressa Barbara Caputo, che con il suo corso di Machine Learning mi ha fatto appassionare al settore e che mi ha permesso di fare la tesi presso il suo laboratorio, dandomi i mezzi che mi hanno permesso di svolgere questo lavoro e seguendomi durante lo svolgimento dello stesso. A questo proposito ringrazio anche la professoressa Tatiana Tommasi, che mi ha dato diversi consigli che si sono rivelati molto utili durante il lavoro, e i ragazzi del laboratorio tra cui soprattutto Silvia Bucci, che è stata sempre disponibile ad aiutarmi e chiarire tutti i miei dubbi.

Infine ringrazio tutti i miei amici, che mi sono sempre stati vicino in questi anni, chi dall'università e chi da prima, e che mi hanno supportato ogni volta che ho avuto problemi o che avevo bisogno di consigli. Un ringraziamento speciale va in particolare as Alessandro, Camilla ed Elisa, che fin dal liceo mi sono sempre stati vicini e con cui ho condiviso tanti momenti felici in questi anni, garantendomi la serenità e il supporto necessari per proseguire il mio percorso sia di studio che di vita. Un ringraziamento speciale va anche a Giulia, che è sempre stata disponibile per un caffè e per scambiarci consigli su qualsiasi cosa. Ringrazio anche Alessandra, Edoardo, Patrizio, Alessandro, che mi hanno accompagnato durante gli ultimi anni di studi e con cui ho condiviso molti momenti sia in ambito di studio che di divertimento.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computers are becoming more and more powerful over the years, such that today they can perform in a few seconds operations that only some years ago required so much time that they were considered impossible. This improvement led to a revolution in the deep learning field, because it allowed to train deeper and deeper neural networks, that are becoming able to solve harder and harder tasks. These networks have reached great performances in many different tasks such as object classification [1], semantic segmentation [2], motion tracking [4] and human pose estimation [3], and these are just some examples of the huge variety of fields where neural networks are reaching important goals.

This work focuses on the problem of semantic place categorization, which consists in identifying the semantic category of a certain place. This problem has a huge variety of applications, especially in robotics given the ever increasing focus on long-term mobile robot autonomy and rapid improvements in visual sensing capabilities and cost [9]. Another important application where this task is highly relevant is the self-driving context, where for a car for example is fundamental to understand if it's travelling on a urban road or in a highway. But there are more and more applications that are emerging in the last years. A place to be recognized can be either an indoor place, such as an office or a corridor, or an outdoor place such as a road, a mountain and so on. The scene recognition task is very challenging, due to the complexity of the concepts to be recognized and the variability of the conditions in which the images can be captured [10]. For example, often scenes from the same category may look different, while scenes from different categories may look similar, and different scenes could share the same objects, as shown in figure 1.1.

We can observe that this problem is often more challenging than the common object recognition task. For example, considering an object to be recognized, it's not so difficult to find a particular set of parts disposed in a certain way for each object category that define that category. Let's think about for example to a horse, which can be of different sizes, different colors etc. but it's always characterized by four legs, a tail, a head and a certain body shape. Another example can be a car, which can vary a lot in its appearances (for example, a 40 years old car is quite different from a car that is produced nowadays), but it's always characterized by a certain number of wheels disposed in a certain way and other well-recognizable elements. A scene instead is a very complex environment that can vary a lot according to the number of objects that it contains, the location, the weather, the time of the day, the season and so many other factors. An example of this is shown in

figure 1.2.

Figure 1.1: Examples of scenes exhibiting high degrees of class similarity. Images annotated with "living" belong to the scene category "living room", while scenes annotated with "dining" belong to the scene category "dining room". Picture taken from [10].

Figure 1.2: Example of variability for an urban road according to the time of the day and weather conditions. Pictures a, b and c are taken in a sunny environment, and c in particular is characterized by the road covered by snow. Pictures d, e and f are taken during night, and are very different from the previous ones. Images taken from the BDD100K dataset [45].

In general, we call *domain* the condition in which a particular image is taken (e.g. all the images taken in a sunny condition belong to the domain *sunny*). The problem related to the complexity and variety of the scene recognition task is the following: if a network is trained with images belonging to a certain set of domains, that we call *source domains*, at test time it will be very accurate if we ask it to recognize an image belonging to one of those domains and traditional approaches [5]-[8] obtain great results in this case, but what if the domain in which the network will operate (i.e. the *target domain*) has never been seen during training? What happens for example if we train a network to recognize sunny images and we test it during night? The answer to this question is that if the two domains

are quite different, of course the performances of the network will decrease significantly and it will struggle to recognize properly the classes. This issue is not a significant problem in simple scenarios where the target domain is known, and so we know exactly in which domain the network will be tested: for example, if we consider a robot that has to recognize the rooms of a certain laboratory and that will be used always in the same laboratory, it doesn't matter if it's able to recognize the same room classes also in other laboratories or not, the only important thing is that it works in that one. It becomes a problem instead in more complex scenarios, where we don't know a priori where the network will be tested and so it must be able to generalize through different domains. The brute-force approach to solve this issue is of course to collect a dataset containing millions of images, that covers all the classes that the network is asked to classify and for each class all the possible domains in which that class could appear. The issue of this solution is that collecting data is expensive, and the more data the network receives the more time it takes to be trained. Let's take for example the case of figure 1.2 where we have to recognize an urban road: it's impossible even to imagine which are all the possible domains in which the scene could appear, we should consider all the possible weather conditions, in different times of the day (because the light condition at night is completely different from the one during the day, but also during the day we can imagine very different light conditions such as sunrise, noon, sunset and so on). For each of these environmental conditions we should then consider other variations such as traffic conditions, different types of roads and also the differences on the cities where the road is located (the roads of an American city are usually quite different from the ones of an European city or an Asian city). Then, given the previous considerations, we can state that it's impossible to cover in the training dataset al the possible domain shifts that we could face in a complex real world scenario.

To address this problem, the recent algorithms exploit the so called *domain adaptation* technique [11]-[13], which consists in developing a model that has to be effective in the domain where it will be tested (target domain). The issue in this case is that usually few labeled samples from the target domain are available, because labeling the images is often a very expensive operation, so DA attempts to transfer useful knowledge from a larger set of source data, which can be for example data previously collected from other sources or derived from datasets available on the web. Domain adaptation can be used in different contexts, either if some labeled samples from the target domain are available or if only unlabeled samples are available. The problem of this solution is that in any case it requires some prior knowledge of the target domain at training time, that means to have some target data available either labeled or unlabeled, but unfortunately target data are not always available as we saw before in the example of the urban road, because in complex scenarios like that it's not possible to collect data for every possible target domain because the possibilities are too many and are unpredictable.

The alternative to DA then is the so called *Domain Generalization*, whose main idea is «to learn a domain agnostic model applicable to any unseen target domain» [46]. In other words, the purpose of DG is to find a model that can learn certain features from the source domains such that they allow it to generalize to the target domains without exploiting any target domain prior. Some domain generalization approaches are described in [14] and [15], while [46] developed a deep learning framework for domain generalization by designing a convolutional neural network architecture with novel layers performing a weighted version of batch normalization.

This work, after giving a brief introduction to neural networks and particularly to CNNs, explores Domain Generalization and Domain Adaptation solutions for the place categorization task starting from the recently introduced JiGen approach [37], which will be described in detail in chapter 3, that gave great results in the context of object detection through different domains.

Chapter 2

Neural Networks

2.1 Introduction

A neural network is a predictive model directly inspired by how the biological nervous system operates and processes information. The brain can be imagined as a collection of neurons wired together, where each neuron receives some output generated from other neurons, does some kind of calculation and then performs one of these two actions: if the calculation exceeds some threshold the neuron fires, otherwise it doesn't. This behaviour can be reproduced with an artificial neural network, which is composed by an arbitrary number of artificial neurons that receive some input and perform similar calculations over them, as can be seen in figure 2.1.

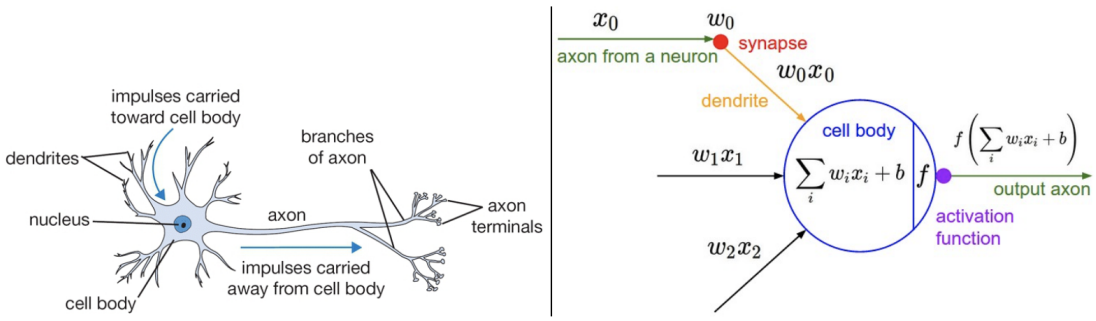


Figure 2.1: Picture taken from [16], representing on the left a cartoon drawing of a biological neuron and on the right it's mathematical model.

Neural networks allow to solve a huge variety of problems, and they are more and more used especially in the last years, because the CPUs and the GPUs are more and more powerful and allow to train bigger networks. Unfortunately most neural networks are "black boxes", which means that given a result that the network produced, it's not so easy to understand how and why it computed that result. So they can be the right choice in some contexts, but not in the ones where the result has to be explainable.

2.2 Artificial Neural Networks

2.2.1 Simple perceptron

In order to start understanding a neural network, the first phase is to analyze the perceptron, which is a simple type of artificial neuron. A perceptron takes several inputs and produces a binary output. Let's consider the example in figure 2.2: the perceptron takes three inputs (x_1, x_2, x_3) and generates a single output.

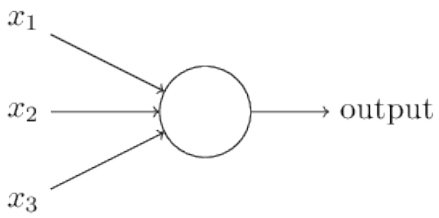


Figure 2.2: Schema of a perceptron, taken from [17]

For computing the output, at each input is assigned a weight (w_1, w_2, w_3) that defines how much that particular input will contribute to the output, that is computed by performing the sum of the weighted inputs, as shown in equation 2.1.

$$\sum_j w_j x_j \tag{2.1}$$

Once the result is computed, the perceptron's output is 0 or 1 according to whether it's less than or greater than some threshold value, which is a parameter of the neuron:

$$\text{output} = \begin{cases} 0, & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1, & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases} \tag{2.2}$$

The same condition can be expressed in a different way, by moving the treshold to the other side of the inequality and by replacing it by the so called *bias*:

$$\text{output} = \begin{cases} 0, & \text{if } \sum_j w_j x_j + b \leq 0 \\ 1, & \text{if } \sum_j w_j x_j + b > 0 \end{cases} \tag{2.3}$$

The bias can be then seen as an input that states how easy is to get the perceptron to output a 1. At this point, we can create a network composed by several layers containing perceptrons, as shown in figure 2.3, and use it to solve a problem. In order to work and to make the learning possible, the condition is that applying a small change to some weights/biases will cause only a small corresponding change in the output from the network, so that given an output we can make some small changes to the weights and biases to obtain a result closer to the one that has to be reached. Unfortunately, this doesn't happen with a network composed by simple perceptrons, because a small change could change the output of a perceptron from 0 to 1, and this change could affect considerably the final output. To avoid this issue, we must introduce an activation function.

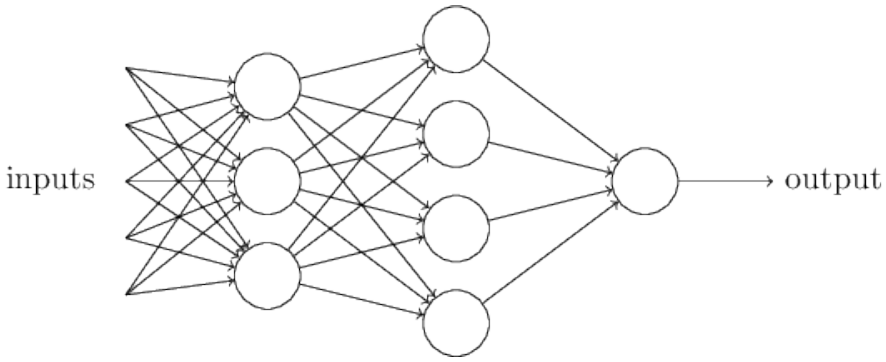


Figure 2.3: Schema of network composed of three layers of perceptrons, taken from [17]

2.2.2 Activation function

In the previous section we saw that the output values of a simple perceptron could be only 0 or 1 ($x_j \in \{0, 1\}$), and this caused the issue described before. A way to solve the problem is to introduce an activation function, for example the *sigmoid*, that allows the inputs of the neurons to be a real number ($x_j \in [0, 1]$, $x_j \in \mathbb{R}$). The sigmoid is a function defined in equation 2.4:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.4}$$

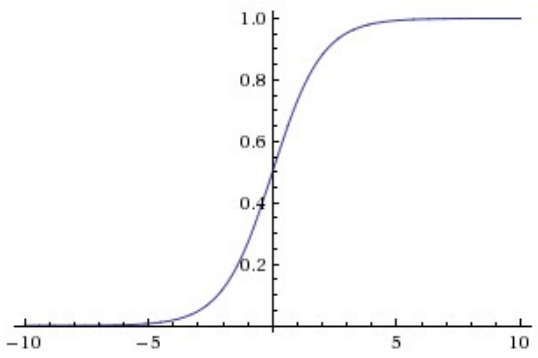
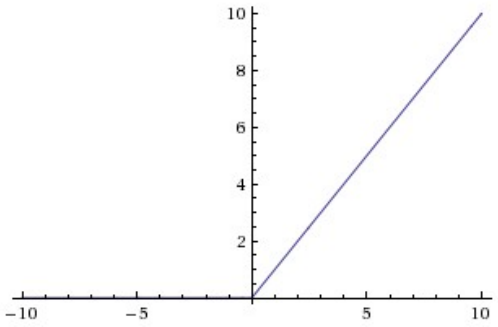


Figure 2.4: Graph of a sigmoid, image from [16]

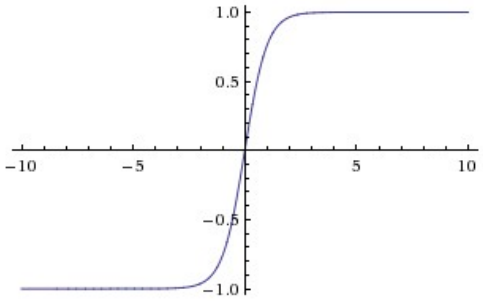
In this case, considering a neuron which receives as input a vector $x \in \mathbb{R}^m$ and performs a dot-product with the weights vector $w \in \mathbb{R}^m$, the output of each neuron is then rewritten in the form:

$$y = \sigma(\sum_j w_j x_j + b), \text{ with } y \in [0,\ 1] \tag{2.5}$$

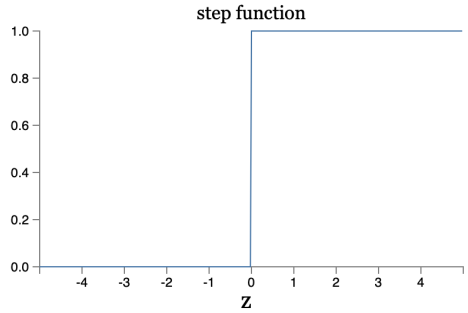
Other commonly used functions are the *tanh*, the *ReLU* and the *step function* (figure 2.5).



(a) *ReLU*



(b) *tanh*



(c) *Step function*

Figure 2.5: Graphs of the most common activation functions, images from [16] and [17]

More in general equation 2.5 can be rewritten in 2.6, where f is the activation function adopted:

$$y = f(\sum_{j=1}^{m} w_j x_j + b), \text{ with } y \in [0, 1] \tag{2.6}$$

2.2.3 Neural networks architecture

As mentioned earlier, a neural network is composed of different layers: an *input* layer, an *output* layer and optionally n *hidden* layers. An example with two hidden layers is shown in figure 2.6. The layers are connected such that the output of layer l_i is the input of l_{i+1}. This example in particular shows a fully connected network, because each neuron of layer l_i is connected with every neuron of layer l_{i+1}, but there are also other networks where not all the neurons are connected. Another clarification is that in this kind of networks each layer takes as input only the outputs of the previous layer, which means that there are no loops in the network: such networks are called *feedforward* neural networks. There are also the so called *recurrent* neural networks which have loops, but they have not been used in this work.

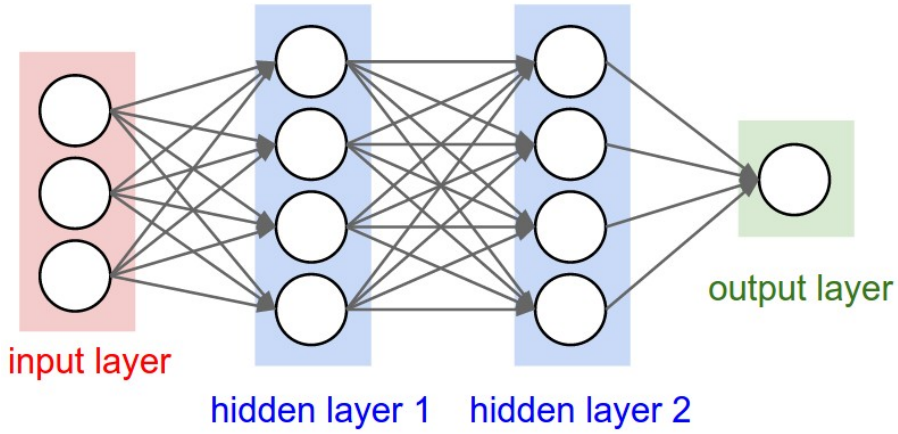


Figure 2.6: Schema of a neural network with two hidden layers, taken from [16]

2.2.4 Training process

The process of training a network consists in learning the weights and the biases for each neuron, such that the difference between the output produced from the network and the desired one is minimized. The training begins with a random initialization of the weights and the biases: it's important to initialize the weights to small values while the biases may be initialized to zero or to small positive values. After the initialization, the process evolves by alternating two phases: forward propagation and back-propagation [20]. The forward propagation is the phase where the information flows forward through the network, starting from the input layer up to the output layer, passing at each hidden layer in the network. As soon as the last value is computed, the output generated is evaluated by a loss function that computes how far is the predicted result from the desired one. The loss function in general has a form like this, assuming a training set $x = \{x_i, y_i\}_{i=1}^{m}$:

$$C(w,b) = \frac{1}{2}\sum_{i=1}^{m} |f_{w,b}(x^i) - y^i| \tag{2.7}$$

where $f_{w,b}(x^i)$ is the predicted value and y^i is the true value. The loss can never assume a negative value, and is very close to zero when the predicted value is almost equal to the target one.

After computing the loss, the back-propagation phase begins: the error that was computed by the loss function is propagated backwards up to the first layer, and the weights and the biases of the neurons are updated according to some policy. The purpose of the whole process is to find the set of weights and biases which minimize the loss function (2.7), and to do so the most common used technique is the *Gradient Descent*, which is a specific technique to solve minimization problems. This allows to obtain an update rule for the weights and the biases of each neuron:

$$w_{i,j} = w_{i,j} - \eta \frac{\partial C}{\partial w_{i,j}} \tag{2.8}$$

$$b_i = b_i - \eta \frac{\partial C}{\partial b_i} \tag{2.9}$$

where η is a small and positive parameter known as learning rate, which controls how much to change the value of the weight/bias in response to the estimated error each time the model weights are updated. Commonly η has an higher value in the first iterations of the training, because at the beginning the loss is high and we are far from the minimum, and it decreases as the loss goes down to reach the convergence. The gradients with respect to each parameter correspond then to the contribution of that parameter to the error, and so they are subtracted. Figure 2.7 Illustrates a schematic representation of the whole training process, while figure 2.8 shows an illustration of the gradient descent algorithm, which repeats step by step by computing the gradients and updating the weights in order to go in the opposite direction to the gradient, getting closer and closer to the minimum until the value of the parameter reaches a point beyond which the loss function cannot decrease.

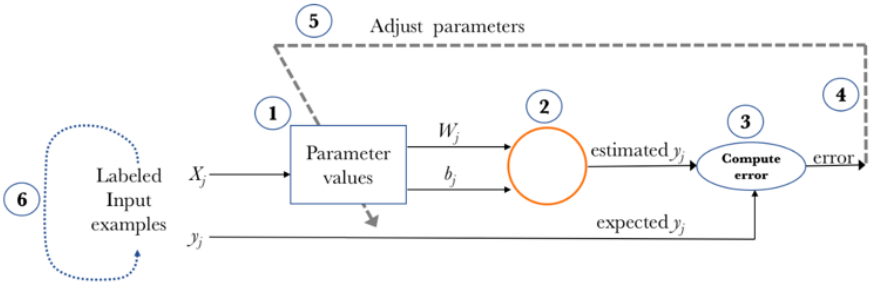


Figure 2.7: Illustration of how the gradient descent works. Image taken from [18].

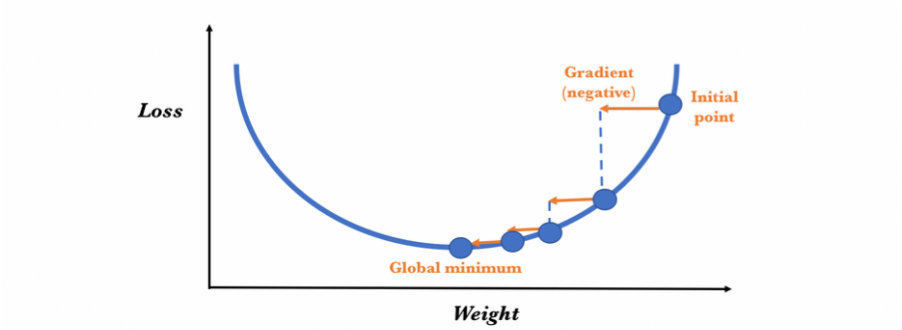


Figure 2.8: Illustration of how the gradient descent works. Image taken from [18].

The gradient descent can be applied in different ways: let's consider the cost function introduced in equation 2.7, this has the form $C = \frac{1}{n}\sum_x C_x$, which is an average over the costs for the individual training samples. This means that in this way the gradients have to be computed on each training sample, but this is not always feasible, especially when

the training samples number is huge. This case is known as *online learning*. To speed up training, it can be used the so called *Stochastic Gradient-Descent (SGD)*, where the gradients are computed only after processing batches of data. In this case the the batch size is a very important parameter, that affects considerably the performances of the algorithm. Commonly it assumes values between 16 and 256. This approach works because, provided that the batch size is large enough, the average value of the gradient of the batch will be roughly equal to the average over the whole set. In mathematical terms, considering ∇C the average overall gradient and ∇C_x the gradient over sample x, we can randomly pick m training inputs from the whole training set, calling $X = X_1, X_2, ..., X_m$ the selected samples, and considering ∇C_{X_j} the gradient over the sample X_j we can say:

$$\nabla C = \frac{\sum_x \nabla C_x}{n} \approx \frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \tag{2.10}$$

This confirms then that the overall gradient can be estimated by computing gradients just for the batch. So, in case of *SGD*, the training has the following schema: the training dataset is split randomly in m batches, which are used one by one for training and updating the weights. After the weights have been updated with batch 1, the process is repeated with batch 2 and so on until all the batches have been used. Once this process ends, it is said that an *epoch* of the training has completed, and the training can be repeated for a new training epoch.

Another technique is called *Batch Gradient Descent* which consists in using a batch that corresponds to the entire dataset, so that the weights are updated after the computation of the loss function over the whole dataset, but this is not efficient in applications that involve large datasets containing million of samples.

2.2.5 Optimizations for neural networks

The backpropagation procedure described so far can be improved with some techniques, that include the use of a better cost function known as cross-entropy and some "regularization" methods.

Starting from the cost function, the easiest way to implement it would be to use a normal quadratic cost function, but this choice sometimes presents an issue, as explained by [17]. Normally we would expect a network to learn slower when the error is small, and faster when the error is huge, but the use of the sigmoid as activation function in combination with this loss function could generate some problems. Let's consider for example a single neuron, whose expected output is 0: depending on the initialization of the weight and the bias the predicted output will converge to 0 as the training proceeds, but if the initial predicted output is very close to 1 (so the error is huge), it happens that the learning starts out very slowly and requires a lot of epochs to drive down the cost of a few points, and at some point it starts to drop fast up until reaching a value very close to zero. This behaviour is strange and is known as *slowdown* problem. The cause is that, when the learning is slow and the predicted output is very close to 1, the partial derivatives of the cost function $\partial C/\partial w$ and $\partial C/\partial b$ are very small and so it takes a lot of time to decrease the value. The derivatives are small because they depend on the derivative of the sigmoid, which is a very low value when the output is very close to zero or very close to one, as can be seen in figure 2.9.

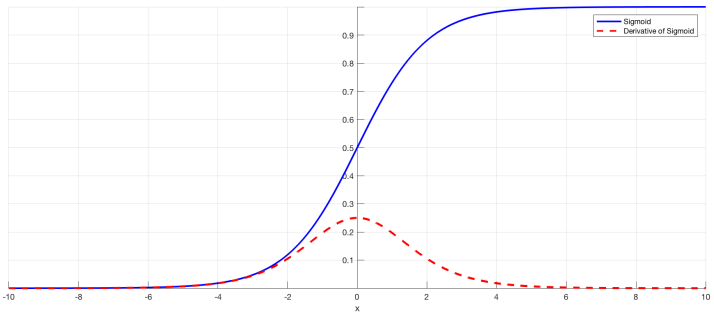


Figure 2.9: Graph of the sigmoid and it's derivative.

In the example of the single neuron defined above, it's not a problem if the output is very close to zero, because it means that it's very close to the true label, but it's a problem if the output is close to one, because it will require a lot of epochs to converge. This issue can be avoided by using the so called *cross-entropy* loss, which is defined as:

$$C = -\frac{1}{n}\sum_x [y \ln a + (1-y)\ln(1-a)] \tag{2.11}$$

where n is the size of the training data, y is the target output and $a = \sigma(z)$ is the output of the neuron, with $z = \sum_j w_j x_j + b$. It can be derived that with this cost function, the rate at which the neuron learns is controlled by $\sigma(z) - y$, that is the error in the output and so the problem has been solved.

Another major issue that can be encountered with neural networks is *overfitting*. [19] says that «the factors determining how well a machine learning algorithm will perform are its ability to:

1. Make the training error small.

2. Make the gap between training and test error small.

These two factors correspond to the two central challenges in machine learning: *underfitting* and *overfitting*. Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large».

When there is overfitting, the networks learns about peculiarities of the training set, not just recognizing the classes of the samples, and this of course is not good and generates problems. This is a major problem in neural networks, especially in the modern ones that have more and more layers and neurons, which results in having more weights and biases. A first way to deal with this problem is to use a validation set separated from the training set and the test set, so that the network is trained on the train samples and at the end of each epoch is tested on the validation set. Overfitting will occur when the accuracy on the validation saturates to 100% while the one of the test stops growing or starts to decrease. So the accuracy on the validation can be used as an overfitting alert, when it saturates the training has to be stopped. This strategy is called *early stop*. Of course, the best solution to overfitting would be to increase the training set dimension by collecting

more and more samples, but training data can often be difficult and expensive to acquire, so it's not always feasible.

Another way to reduce overfitting is to use one of the so called *regularization* techniques. The most commonly used are *L2 regularization* (known also as *weight decay)*, *L1 regularization*, *dropout* and *data augmentation*. L2 regularization consists in adding to the cost function an extra term that penalizes large weights and so makes the network to prefer small weights, as the equation below shows:

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2 \tag{2.12}$$

where C_0 is the original loss function, which can be the quadratic loss, cross-entropy or any loss function. λ is the regularization parameter and it decides how much the regularization term will impact the loss. So this operation can be viewed as finding a compromise between having small weights and minimizing the minimizing the original cost function. This technique has shown to help the network to prevent overfitting and to generalize better.

Another technique is L1 regularization, which is very similar to L2, but in this case the regularization term is a bit different:

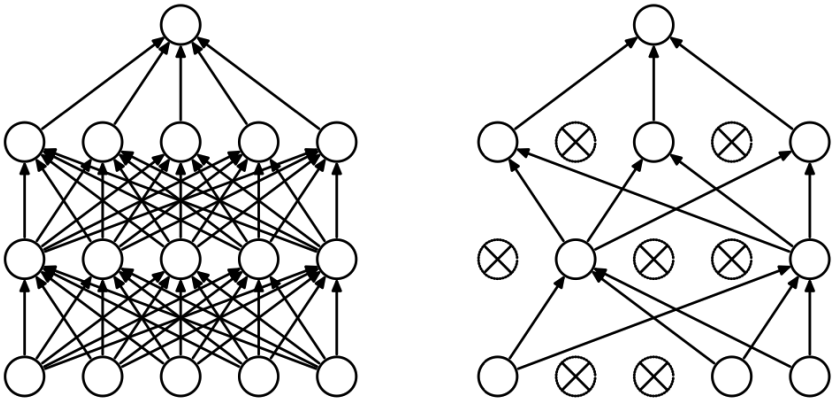
$$C = C_0 + \frac{\lambda}{n} \sum_w |w| \tag{2.13}$$

In this case, the operation is still penalizing the large weights, but the way the weights shrink is different from L2: when a weight has a large value $|w|$, L1 shrinks it much less than L2, while when the value is small, we have the opposite behaviour.

The third possible regularization technique is called *dropout*, introduced in [21] and [22], which doesn't change the loss function as the previous techniques, but the network itself. With dropout, when a batch of images has to be forward propagated through a network, a percentage of random neurons are deleted from the network ([23] found out that for the hidden layers «p=0.5 is the value that provides the highest level of regularization», where p is the probability for a neuron to be cut off from the network at each iteration). Figure 2.10 shows the comparison between a standard neural network and a network that uses dropout, where some of the neutrons are cut off.

The important thing is that the neurons are cut off only at training time, while at test time all of them are used. The explanation of the improvement given by dropout is given in [24], one of the papers that first used this technique: «This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.».

The last optimization technique presented in this section is *data augmentation*, which consists in artificially expanding the training data by performing some transformations on the samples such as random rotations, crops, horizontal flips and so on, operations that reflect real-world variation.



(a) Standard Neural Net (b) After applying dropout.

Figure 2.10: Comparison between a standard neural network and a one that uses dropout. Image taken from [21].

2.3 Convolutional Neural Networks

Convolutional neural networks [25] are a category of neural networks specialized to classify images. The name indicates that these networks use the *convolution* mathematical operation. The convolution in general is defined as:

$$s(t) = \int x(a)w(t-a)da \tag{2.14}$$

In the case of convolutional networks, the first argument (x) is the input, the second argument (w) is defined as kernel and the output (i.e s, often defined as *feature map*). CNNs are composed by different types of layers, that are described one by one in this section.

2.3.1 Convolutional layers

In a CNN, the input image is considered as a matrix of pixels, where each pixel corresponds to a cell of the matrix. Let's consider an example of a 5x5 px image, with just one color channel to simplify the case: the image will be saved as a 5x5 matrix, then a convolution is performed between this matrix and a so called *kernel*, which is a filter that recognizes a particular pattern of the image.

Figure 2.11 shows the result of the convolution by using the kernel $k = \begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{vmatrix}$

An important parameter in the convolution is the *stride*, which indicates the number of pixel shifts over the input matrix: in the example above it was equal to 1, so the kernel shifted 9 times. So in this case the first layer after the input layer generates a 1x3x3 output, because only one kernel has been used, but usually a layer has to detect more than just one feature, so more kernels can be used. For example, if it uses three kernels, the output generated will be 3x3x3. Optionally can be added also the *padding* parameter, which adds

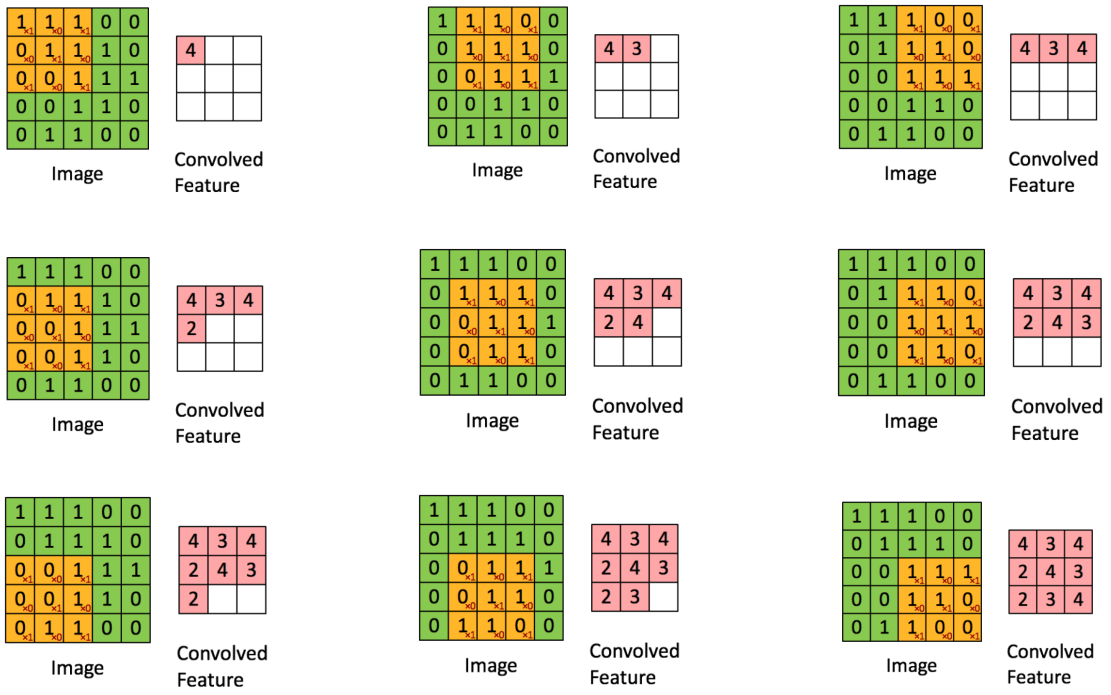


Figure 2.11: Convolution of a 5x5 matrix with a 3x3 kernel, using stride=1.

zeroes to the edge of image depending on the stride and the dimensions of the image. Given an input of size NxN, the dimensions of the feature maps can be obtained by 2.15:

$$w = \frac{N - F + 2P}{S} + 1 \tag{2.15}$$

where w is the output width, N is the width of the input matrix, F is the kernel size, P is the padding and S is the stride. An important concept in CNN is *non-linearity*, because images are non-linear by nature (e.g. transition between pixels, the colors, the borders etc.), so usually the output of the convolutional network is is given to a non linear function. One of the most used is the ReLU, defined as:

$$f(x) = max(0, x) \tag{2.16}$$

Other commonly used non-linear functions are the sigmoid, tanh and leaky ReLU.

2.3.2 Pooling layers

In addition to convolutional layers, a CNN also contains *pooling layers*, which are usually used immediately after the previous ones. These layers have the scope to simplify the output of the convolutional layers, by reducing the size of the feature maps.

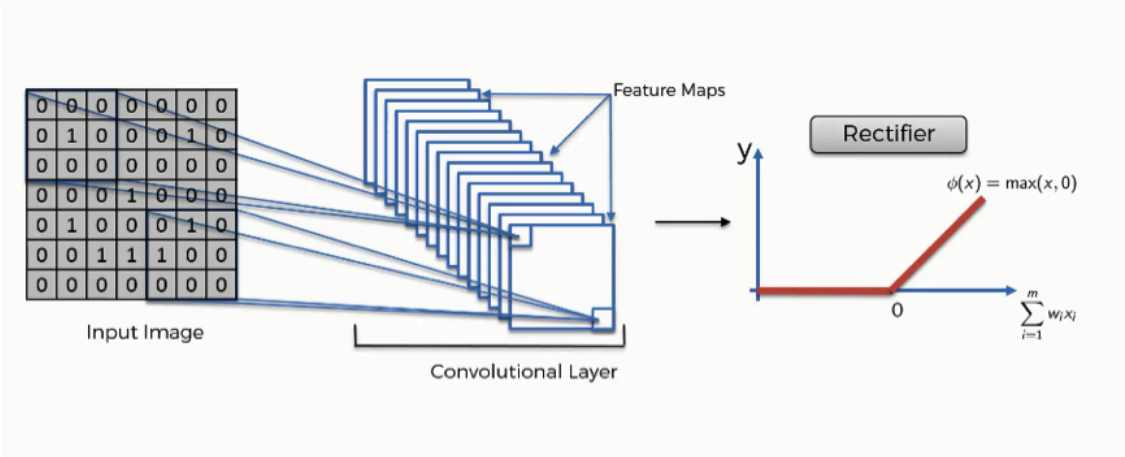


Figure 2.12: Example of usage of relu after the convolutional layer.

Given an input of size NxN, the dimension of the output of the pooling is given by the formula:

$$w = \frac{N - F}{S} + 1 \tag{2.17}$$

There are different types of pooling, the most commons are the *average pooling* and the *max pooling*. An example of max pooling is shown in figure 2.13.

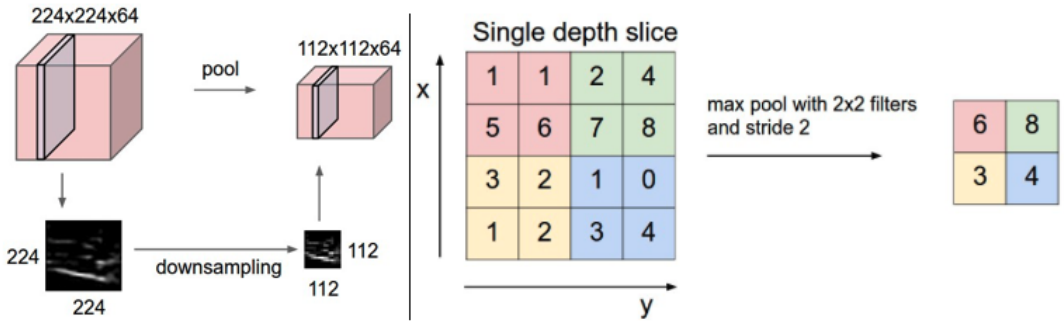


Figure 2.13: The pooling layer receives a 64x224x224 input from the convolutional layer and performs a max pooling operation with 2x2 filters and stride=2. Image taken from [25]

Max pooling, as stated in [17], can be seen «as a way for the network to ask whether a given feature is found anywhere in a region of the image. It then throws away the exact positional information. The intuition is that once a feature has been found, its exact location isn't as important as its rough location relative to other features. A big benefit is that there are many fewer pooled features, and so this helps reduce the number of parameters needed in later layers.».

Average pooling is very similar, with the difference that instead of taking the max, the average value is taken.

2.3.3 Fully Connected (FC) Layers

A FC is the classical layer of a neural network, whose neurons are connected to all the neurons of the previous layer. In CNNs they are used in the final part of the network, after the last pooling layer, and there can be only one or more than one. To be given to the FC, the w x h x d output of the pooling layer must be flattened to obtain a 1 x 1 x n array. The only parameter of the fully connected layer is the number of the neurons, K, which states how many outputs the layer will generate. The last FC has always K = number of classes.

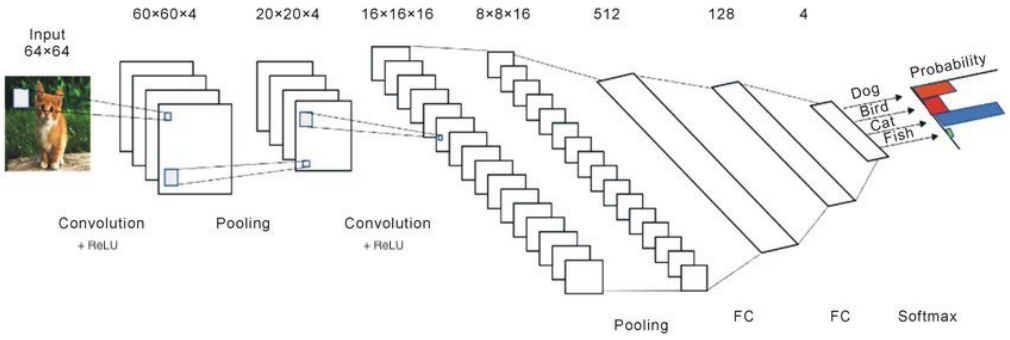


Figure 2.14: Example of a simple cnn. Image taken from [26].

Figure 2.14 shows a complete example of a very simple CNN, that receives in input images of size 64x64 px. This simple network is composed by two convolutional layers (both of them including a ReLU operation) and two pooling layers. After the last pooling layer, there are two fully-connected layers. The softmax at the end allows to obtain as output for each class the probability $p \in [0, 1]$ that the image belongs to that class.

2.3.4 Batch Normalization Layers

In 2015 [27] introduced a special kind of layer, called Batch Normalization Layer, that is frequently used in the modern networks after each convolutional layer. This layer has the scope to simplify the training of a CNN by reducing the so called *internal covariance shift*, defined in the paper as «the change in the distribution of network activations due to the change in network parameters during training. To improve the training, we seek to reduce the internal covariate shift. By fixing the distribution of the layer inputs x as the training progresses, we expect to improve the training speed». Batch Normalization is based on the idea that the training of a network converges faster if the inputs are normalized (linearly transformed to have zero means and unit variances), as discovered by [28]. This layer then is responsible of the normalization of its inputs: during the training phase, for each batch, all the elements are subtracted by the batch mean and divided by the batch variance. For example, considering a batch B, composed by $x = \{x_i, ..., x_m\}$, the normalized value $\widehat{x}$ is given by equation 2.20 [28]:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.18}$$

$$\delta_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{2.19}$$

$$\widehat{x} = \frac{x - \mu_B}{\sqrt{\delta_B^2 + \epsilon}} \tag{2.20}$$

where ϵ is a small constant added for numerical stability. The final output of batch normalization is given by 2.22:

$$\widehat{y} = \gamma x + \beta \equiv BN_{\gamma,\beta}(x) \tag{2.21}$$

where $E[x]$ and $Var[x]$ use all the input data.

where γ and β are two learnable parameters. As [28] says, «The normalization of activations that depends on the mini-batch allows efficient training, but is neither necessary nor desirable during inference; we want the output to depend only on the input, deterministically.» For this reason, once the network has been trained, the normalization used at test time is:

$$\widehat{y} = \gamma\frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} + \beta \tag{2.22}$$

where $E[x]$ and $Var[x]$ refers to the whole population, not just the single batches. [28] discovered also that batch normalization allows to use higher learning rates and makes the training more resilient to parameter scale.

2.4 Transfer learning

Transfer learning is very popular technique in the last years in the computer vision field, because it allows to build very accurate models in less time and with less data. This technique consists in exploiting a model that was already trained to solve another problem to solve the current one, instead of training a new network from scratch. Transfer learning is expressed through the use of pre-trained models, which are models that were trained on very large datasets containing millions of images (e.g. ImageNet [30]). [29] presents a comprehensive review of the performances of several pre-trained models on computer vision problems using data from the ImageNet challenge. The pre-trained models are adapted to the new problem through the so called fine-tuning process: the CNN is trained on the new dataset by keeping the weights that were learnt before, by removing the last FC layer (the classifier) and replacing it with a one that has the proper number of outputs according to the number of classes of the problem to handle. There are several fine-tuning strategies that can be used: retrain the entire model, train some layers and leave the others frozen or freeze the convolutional base and retrain only the classifier. The choice of one of these strategies must be taken according to the dataset size and also its similarity to the dataset that was used to pre-train the network, as figure 2.15 shows:

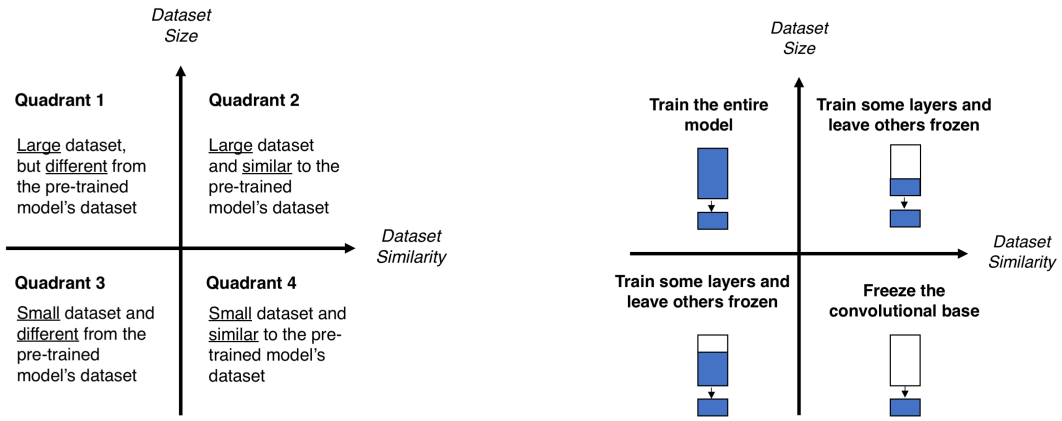


Figure 2.15: Usage policies of pre-trained convolutional neural networks. Image taken from [31].

2.5 Popular Convolutional Neural Networks

As said before, it's a common practice in the last years to use a model that was already tested in other contexts and that proved to be good. The advantage can come both from the weights that were learnt in the pre-training phase (if a pre-trained network is used), but also from the structure of the network itself. Today a lot of models are available, and some of the most used among these are breefly described in this section.

2.5.1 AlexNet

The AlexNet architecture was proposed in 2012 by [32], when it competed in the ImageNet Large Scale Visual Recognition Challenge where it reached top-1 and top-5 error rates of 37.5% and 17.0% which was considerably better than the previous state-of-the-art. The network structure is shown in figure 2.16:

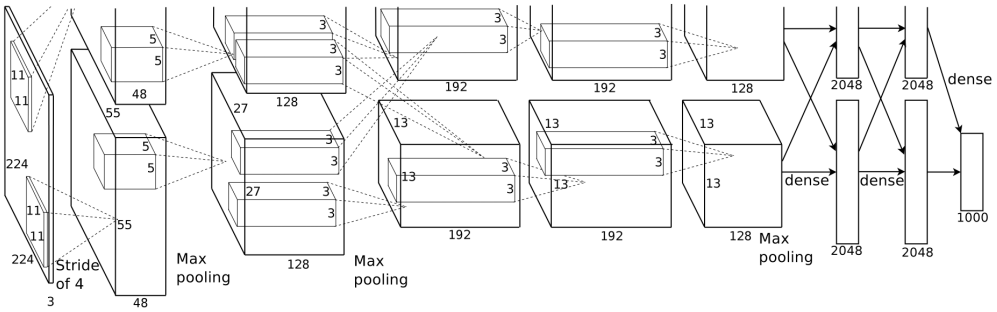


Figure 2.16: Architecture of the AlexNet, image taken from [32].

«The neural network, which has 60 million parameters and 650,000 neurons, consists

of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax.» [32]. In the fully-connected layers it uses the *dropout* regularization method.

2.5.2 VGGNet

The VGGnet was developed in 2014 by [33], in the context of object recognition and detection. The network was proposed in different configurations, as shown in image 2.17

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Figure 2.17: Different configurations of VGG. Image taken from [33].

The network won the ImageNet Challenge 2014, where it allowed the team to reach the first and the second places in the localization and classification tasks respectively.

The convolutional layers make use of filters with a very small receptive field: 3x3, with stride=1 and padding=1. The max pooling layers use filters 2x2 with stride=2. All the configurations shown in 2.17 follow the same design and «differ only in the depth: from 11 weight layers in the network A (8 conv. and 3 FC layers) to 19 weight layers in the network E (16 conv. and 3 FC layers). The width of conv. layers (the number of channels) is rather small, starting from 64 in the first layer and then increasing by a factor of 2 after each max-pooling layer, until it reaches 512.» [33]. The configuration that achieved the best results is D.

2.5.3 GoogLeNet

GoogLeNet [34] is a 22-layers deep network that won the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14) by reaching a 6.7% top 5 error. This network has 12x less parameters (only 5 million, it removes FC layers completely) than the AlexNet and is 2x more accurate. This network makes use of the so called *Inception Layers*: the main idea of these layers is to cover a bigger area but at the same time also keep fine resolutions for small informations on the images. This goal is reached by using a series of filters with different sizes, as shown in figure 2.18, while figure 2.19 shows the details of the parameters for each layer.

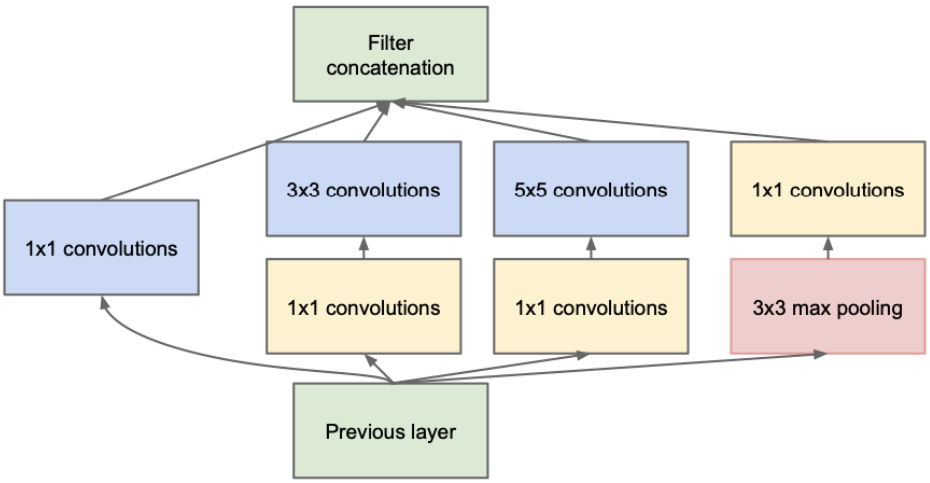


Figure 2.18: Inception module. Image taken from [34].

2.5.4 ResNet

Residual networks [35] were first presented in 2015 and won the 1st place on the ILSVRC 2015 classification task. These networks were built to face the *vanishing gradient* problem [36]: if a network depth is very large, the gradient is back-propagated in many layers and it could then became very small, making the training process very difficult. For this reason, usually increasing the number of layers in a normal network allows to obtain an improved

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figure 2.19: Inception module. Image taken from [34].

accuracy, but at some point the vanishing gradient problem will make it decrease, as shown in figure 2.20:

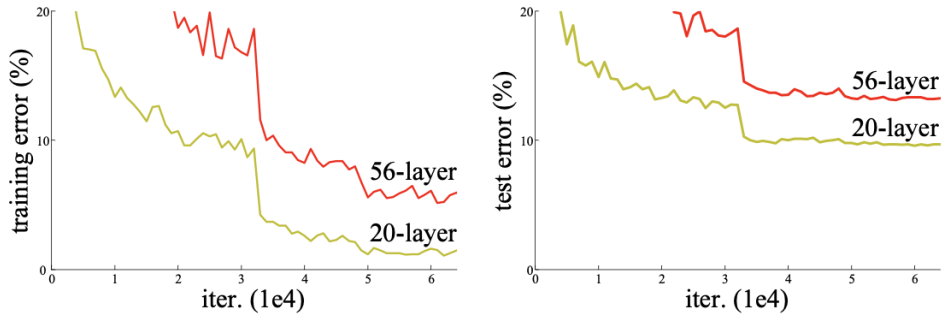


Figure 2.20: Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Image taken from [35].

The ResNet faces this problem with the so called *residual block*, shown in figure 2.21:

As can be seen in the figure, the residual block is characterized by the so called *skip connection* identity mapping, that is responsible of adding the output from the previous layer to the layer ahead. So, considering a layer:

$$y = F(x, \{W_i\}) + x \tag{2.23}$$

where x and y are the input and output vectors of the layers considered, and $F(x, \{W_i\})$

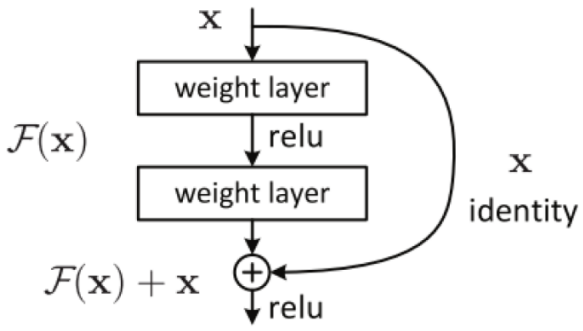


Figure 2.21: Illustration of a residual block. Image taken from [35].

represents the residual mapping to be learned. Sometimes however $F(x)$ and x can have different dimensions, in this case we can perform a linear projection W_s by the shortcut connections to match the dimensions:

$$y = F(x, \{W_i\}) + W_s x \tag{2.24}$$

The design of a 34-layers residual network is illustrated in figure 2.22.

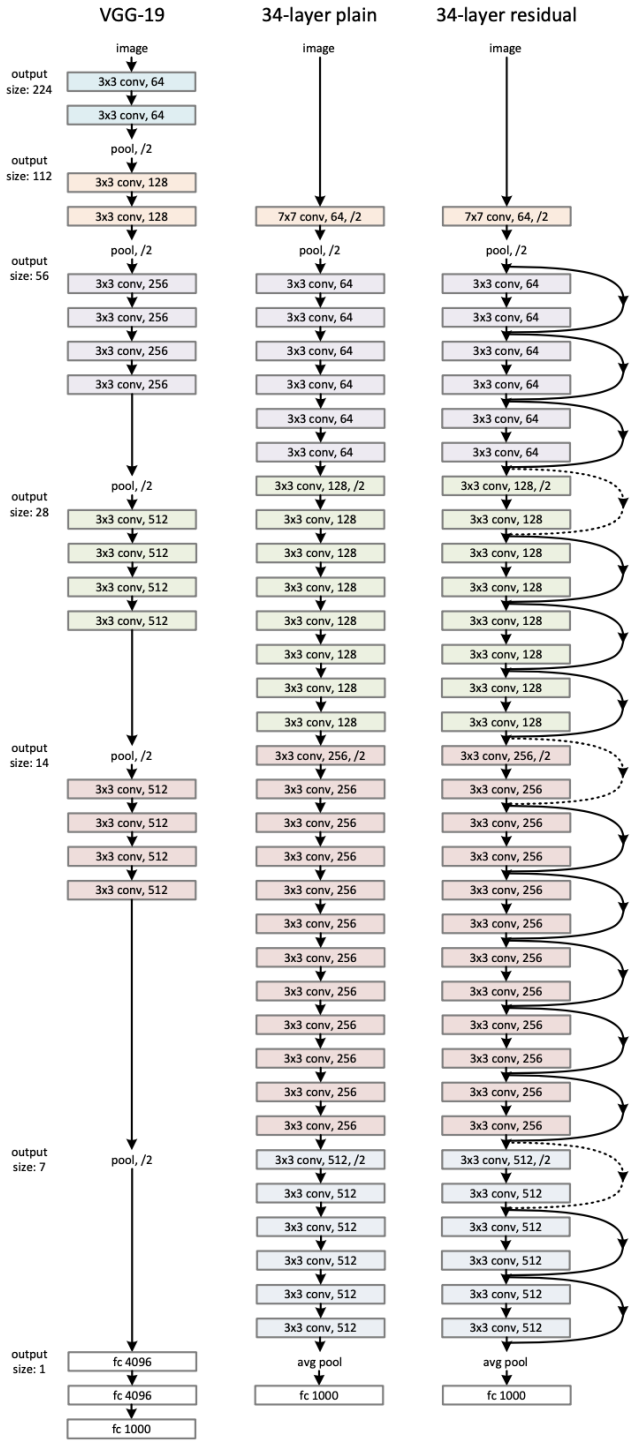


Figure 2.22: Example of a 34-layers ResNet. Image taken from [35].

Chapter 3

The JiGen approach

This chapter gives an overview over JiGen, an approach recently introduced that gave great results in the task of object recognition across different domains. This approach was the starting point of this work, and it was adapted to obtain good results also in the context of place recognition.

3.1 Introduction

The JiGen approach, presented in a CVPR paper [37] in 2019, is based on the consideration that «human adaptability relies crucially on the ability to learn and merge knowledge both from supervised and unsupervised learning: the parents point out few important concepts, but then the children fill in the gaps on their own. This is particularly effective, because supervised learning can never be exhaustive and thus learning autonomously allows to discover invariances and regularities that help to generalize». This means to find a way to merge supervised and unsupervised learning in an method that will have the ability to generalize through different domains, and for this purpose JiGen proposes to perform two kind of operations: a supervised learning by training the model to recognize the labels, and an unsupervised learning that makes the model learn how to solve jigsaw puzzles on the same images. As the authors say, «this secondary task helps the network to learn the concepts of spatial correlation while acting as a regularizer for the classification task», and their experiments show that the method outperforms previous domain generalization and adaptation solutions.

3.2 Algorithm overview

As told in the introduction, the main peculiarity of JiGen is the use of a secondary unsupervised task to help the model to generalize better, and this task consists in the recovery of an original image from its shuffled parts, also known as solving jigsaw puzzles. The classification task and the jigsaw task share the same network backbone, so the approach can be used on every kind of network without performing specific architectural changes. The only change to make is to add to the network the jigsaw classifier, which replaces the

last FC layer of the network (the standard classifier). Figure 3.1 shows with an illustration how the method works:

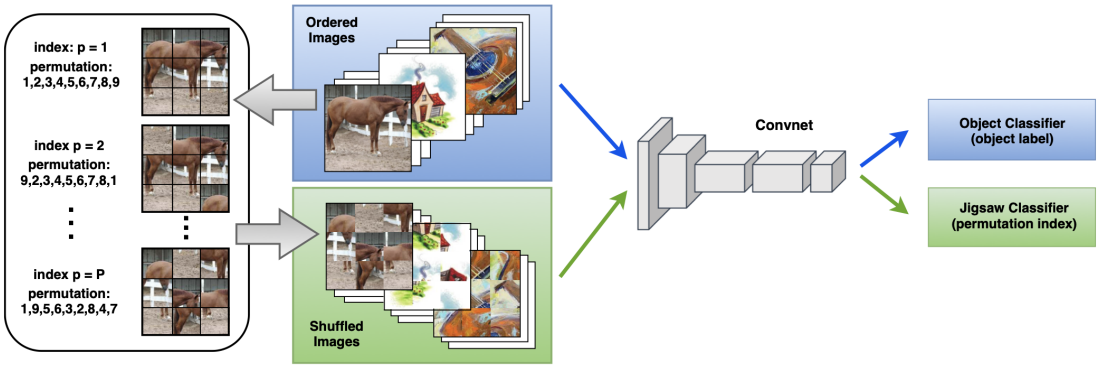


Figure 3.1: Illustration of the JiGen approach. Starting from images of multiple domains, a 3×3 grid is used to decompose them in 9 patches which are then randomly shuffled and used to form images of the same dimension of the original ones. By using the maximal Hamming distance algorithm in [38] a set of P patch permutations is defined and an index is assigned to each of them. Both the original ordered and the shuffled images are fed to a convolutional network that is optimized to satisfy two objectives: object classification on the ordered images and jigsaw classification, meaning permutation index recognition, on the shuffled images. Image taken from [37].

Considering a batch of images that the network receives in input, some images will be given ordered to be classified normally and some others will be instead given shuffled, and the proportions of the split in these two groups are given by the parameter β which will be explained in details later. Considering the shuffled ones, each image is divided in n patches and shuffled in one of the P possible permutations, where P is a subset of the $n^2!$ possible permutations of the tiles. The network will then receive both shuffled and ordered images, that will go through the same convolutional network and at the end will be classified by the object classifier or by the Jigsaw classifier. The object classifier receives and classifies only the ordered images, while the jigsaw classifier classifies both ordered and shuffled images (the ordered images are given with a permutation index $p_i = 0$, while the shuffled ones with the permutation index chosen for that permutation class). The final purpose of this method, as said in [37], is the following: «starting from the samples of multiple source domains, we wish to learn a model that can perform well on any new target data population covering the same set of categories.».

From a mathematical point of view, by following the same notation of the paper, let let us consider to have S domains, with the i-th domain containing N_i labeled instances $\{(x_j^i, y_j^i)\}_{j=1}^{N_i}$, where x_j^i is the j-th image and $y_j^i \in \{1, ..., C\}$ is its class label. The primary goal of the network corresponds to minimize the loss $\mathcal{L}_c(h(x|\theta_f, \theta_c), y)$, which is referred to the error between the true label y and the predicted value $h(x|\theta_f, \theta_c)$, where h is the deep model function parametrized by θ_f and θ_c. These parameters define the feature embedding space and the final classifier, respectively for the convolutional and fully connected parts of the network.

Together with the primary task, the network has also to perform the secondary unsupervised task of the jigsaw puzzles. For this purpose, the images are split in n x n grids of patches, which are then shuffled considering one of the $n^2!$ possible permutations. Out of the $n^2!$ permutations only a subset S is considered by following the Hamming distance based algorithm in [38], and to each permutation selected an index is assigned. At this point, a classification task can be defined: considering K_i labeled instances $\{(z_k^i, p_k^i)\}_{k=1}^{K_i}$ where z_k^i indicates the recomposed samples and $p_k^i \in \{1, ..., P\}$ the related permutation index, the loss to minimize is $\mathcal{L}_p(h(z|\theta_f, \theta_c), p)$, where the final fully connected layer dedicated to permutation recognition is parametrized by θ_p and θ_f is the same as before. The complete loss is given by:

$$\underset{\theta_f,\theta_c,\theta_p}{\operatorname{argmin}} \sum_{i=1}^{S} \sum_{j=1}^{N_i} \mathcal{L}_c(h(x_j^i|\theta_f, \theta_c), y_j^i) + \alpha \mathcal{L}_p(h(z_k^i|\theta_f, \theta_c), p_k^i) \tag{3.1}$$

where $\mathcal{L}_c$ and $\mathcal{L}_p$ are both cross-entropy losses, and α is the weight given to the loss of the jigsaw task. This parameter is very important and must be chosen properly, because it influences considerably the final accuracy. At training time, the jigsaw classifier receives both the shuffled images and the normal ones, while at test time only the normal classifier is used.

3.3 Implementation details

The algorithm has two parameters related to how the jigsaw task is defined (n and P) and three parameters related to the learning process (α, η, β). As mentioned before, n is the dimension of the grid n x n used to define the image patches, and P is the size of the chosen subset of the $n^2!$ possible permutations of the tiles. In the paper was found that the optimal values for these two parameters are $n = 3$ and $P = 30$. The others parameters are α, which as said before is the weight given to the jigsaw loss, *eta* assigned to the entropy loss when included in the optimization process for unsupervised domain adaptation, and β which defines the percentage of images that are given shuffled to the network inside each batch. For example, a $\beta = 0.6$ means that for each batch 60% of the images are ordered and 40% are shuffled.

Finally, JiGen is trained with SGD solver, 30 epochs, batch size 128, learning rate set to 0.001 and stepped down to 0.0001 after 80% of the training epochs. A simple data augmentation protocol is used, by randomly cropping the images to retain between 80 - 100% and randomly applying horizontal flipping.

3.4 Results

The JiGen algorithm was tested on different datasets: PACS [39], VLCS [40] and Office-Home [41]. In particular, figure 3.2 shows the results with Domain Generalization for PACS, which covers 7 object categories and 4 domains (Photo, Art Paintings, Cartoon and Sketches). The results are obtained by training the model considering three domains as source datasets and the remaining one as target.

	PACS	art_paint.	cartoon	sketches	photo	Avg.
	CFN - Alexnet					
	J-CFN-Finetune	47.23	**62.18**	**58.03**	70.18	59.41
	J-CFN-Finetune++	51.14	58.83	54.85	73.44	59.57
	C-CFN-Deep All	59.69	59.88	45.66	85.42	62.66
	C-CFN-JiGen	**60.68**	60.55	55.66	82.68	**64.89**
	Alexnet					
[26]	Deep All	63.30	63.13	54.07	87.70	67.05
	TF	62.86	66.97	57.51	**89.50**	69.21
[28]	Deep All	57.55	67.04	58.52	77.98	65.27
	DeepC	62.30	69.58	64.45	80.72	69.26
	CIDDG	62.70	69.73	64.45	78.65	68.88
[25]	Deep All	64.91	64.28	53.08	86.67	67.24
	MLDG	66.23	66.88	58.96	88.00	70.01
[14]	Deep All	64.44	72.07	58.07	87.50	70.52
	D-SAM	63.87	70.70	64.66	85.55	71.20
	Deep All	66.68	69.41	60.02	89.98	71.52
	JiGen	**67.63**	**71.71**	**65.18**	89.00	**73.38**
	Resnet-18					
[14]	Deep All	77.87	75.89	69.27	95.19	79.55
	D-SAM	77.33	72.43	**77.83**	95.30	**80.72**
	Deep All	77.85	74.86	67.74	95.73	79.05
	JiGen	**79.42**	**75.25**	71.35	**96.03**	80.51

Table 1. Domain Generalization results on PACS. The results of JiGen are average over three repetitions of each run. Each column title indicates the name of the domain used as target. We use the bold font to highlight the best results of the generalization methods, while we underline a result when it is higher than all the others despite produced by the naïve Deep All baseline. *Top*: comparison with previous methods that use the jigsaw task as a pretext to learn transferable features using a context-free siamese-ennead network (CFN). *Center* and *Bottom*: comparison of JiGen with several domain generalization methods when using respectively Alexnet and Resnet-18 architectures.

Figure 3.2: Results of JiGen on the PACS dataset and comparison with previous methods. Image taken from [37].

Chapter 4

Algorithm overview

4.1 Introduction

For the development of the code used in this work, the starting point was the algorithm of JiGen of [37], which was described in chapter 3. The algorithm was adapted to work with the place recognition datasets that I used in the experiments, but unfortunately I wasn't able to obtain in this context the same improvements that were obtained in the paper with the PACS dataset, so we had to make some changes in order to obtain good results in this context. This chapter explores the algorithm used and the main ideas behind it.

4.2 The algorithm

4.2.1 Data preparation

The first phase in an image classification problem is to analyze the dataset and to decide how to give the images to the algorithm. In our case the images come from two datasets: COLD and BDD100K (see Chapter 5). All the images used are resized in order to match the size 225x225 px, which is required by the pretrained neural network used, and then normalized. The pictures of COLD in particular are characterized by a thin green line at the bottom, that causes troubles with the secondary task and makes it too easy (as will be explained later, the secondary task performs rotations on the images and has to recognize the original from the rotated ones, but having a regular pattern such as the green line in this case makes the problem too easy and the network doesn't learn the proper features). So in this case instead of a normal resize is performed a random crop to exclude the green line. Another important step is to decide how to divide the images in training, validation and test set. Each dataset contains images from three domains, at each run two domains are considered known (source domains) and the remaining one is considered unknown (target domain). The test set is composed by the images of the target domain, while the source domains are split in training set (90% of the images) and validation set (10%).

4.2.2 Convolutional Neural Network

The main component of an algorithm for image classification is the neural network used, which can be either built from scratch or it can be used an already existing one, by adapting it to the problem and to the number of classes of the study case (see Chapter 2). The algorithm of JiGen uses different pre-trained networks to perform the experiments: an Alexnet and a Resnet-18, both of them pre-trained on Imagenet [30], which is a large visual database, containing more than 14 million images, designed for use in visual object recognition. For this work, it was initially used the same Alexnet used there, then we tried to search if there was a network pre-trained on a dataset specific for the place recognition task, and we found Places365. This dataset [42] contains over 1.8 million images from 365 scene categories, belonging to three macro classes: indoor, nature and urban. Figure 4.1 shows some examples of classes and images for each of the three macro classes:

Figure 4.1: Image samples from various categories of the Places Database (two samples per category). The dataset contains three macro-classes: Indoor, Nature, and Urban. Image taken from [42].

The authors offer different networks* pre-trained on the dataset, among which we choose the CaffeNet, described in [43], which is a variation of the standard Alexnet. The caffenet has the structure illustrated in figure 4.2.

It is composed from 5 convolutional layers, each of which is followed by the ReLU operation (see Chapter 2). The first two layers are followed also by a max pooling layer and a normalization layer, while the last layer is followed only by a pooling layer. After the convolutional layers there are two two FC layers with 4096 neurons each, each of them followed by a dropout layer, and finally one last FC layer that has 1000 outputs, that in this work is replaced with the rotations classifier or with the classifier adapted for the number of classes of this problem, which is 4.

*The pre-trained networks are available at the page: https://github.com/CSAILVision/places365

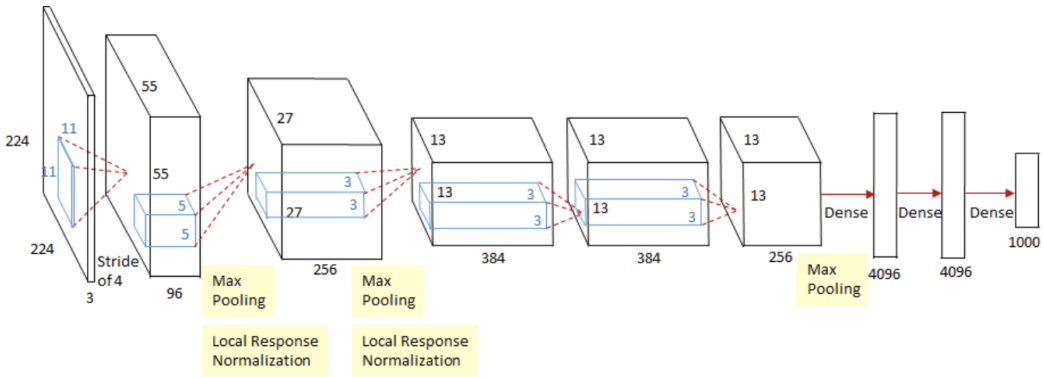


Figure 4.2: Illustration of the structure of the CaffeNet.

This network on our problem has proved to perform better than the Imagenet version, so we kept on the experiments using this one.

4.2.3 The rotations approach

The network, following the example of the JiGen [37] architecture, is characterized by two tasks:

- **Primary task**, that is the supervised classification task. Given an image and a set of classes, this task has to determine the class to which the image belongs.
- **Secondary task**, that corresponds to the unsupervised rotation task. Given an image, a rotation is performed in one of the 3 possible combinations (90°, 180° and 360°) and then the network has to determine if the image has been rotated or not. This task is completely unsupervised because the images that it receives don't need to be labeled, so it has the advantage that it use any image coming from any source.

By following the same notation as [37], let us consider to have S domains, with the i-th domain containing N_i labeled instances $\{(x_j^i, y_j^i)\}_{j=1}^{N_i}$, where x_j^i is the j-th image and $y_j^i \in \{1, ..., C\}$ is its class label. The primary task will correspond to minimize the loss $\mathcal{L}_c(h(x|\theta_f, \theta_c), y)$, which corresponds to the error between the true label y and the predicted value $h(x|\theta_f, \theta_c)$, where h is the deep model function parametrized by θ_f and θ_c. These parameters define the feature embedding space and the final classifier, respectively for the convolutional and fully connected parts of the network.

In addition to the primary task, the network is also asked to perform the secondary task of the rotations. For this secondary task, considering K_i labeled instances $\{(z_k^i, p_k^i)\}_{k=1}^{K_i}$ where z_k^i indicates the rotated samples and $p_k^i \in \{1, ..., P\}$ the related permutation index ($p_k^i = 0$ if the image is not rotated, $p_k^i \in \{1, 2, 3\}$ if the image is rotated). In this case the loss to minimize is $\mathcal{L}_p(h(z|\theta_f, \theta_c), p)$, where the final fully connected layer dedicated to permutation recognition is parametrized by θ_p and θ_f is the same as before. The complete loss is given by:

$$\underset{\theta_f,\theta_c,\theta_p}{\operatorname{argmin}} \sum_{i=1}^{S} \sum_{j=1}^{N_i} \mathcal{L}_c(h(x_j^i|\theta_f,\theta_c), y_j^i) + \alpha \mathcal{L}_p(h(z_k^i|\theta_f,\theta_c), p_k^i) \tag{4.1}$$

where $\mathcal{L}_c$ and $\mathcal{L}_p$ are both cross-entropy losses, and α is the weight given to the loss of the rotation task. This parameter is very important and must be chosen properly, because it influences considerably the final accuracy. At training time, the rotation classifier receives both the rotated images and the normal ones, which are sent with a rotation index $p_k^i = 0$, while the normal classifier receives only the non rotated images otherwise the classification task would be tougher. At test time instead, only the normal classifier is used.

4.2.4 Domain Generalization and Domain Adaptation

The rotation task in the algorithm is used with two different approaches. The first one is the Domain Generalization approach, where we assume that at training time we have no target domain sample available, so the images that are rotated and used for the secondary task belong all to the source domains. This approach works because, by learning to recognize which images are rotated and which are not, the model learns some features that proved in the experiments to be useful also in the classification task for images of a different domain, unknown at training time, which means that these features are domain invariant and so allow to generalize. The second approach is Domain Adaptation, that can be useful when at training time we have some unlabeled samples of the target domain in which the network will be tested. In this case these images can be used for the secondary task, so that the model can peek into the target domain and improve further the accuracy. In this case, the secondary task can either use only images of the target domain, or images from both source and target.

4.2.5 Implementation details

The algorithm receives as input two important parameters: α and β. Alpha as already mentioned before is the weight of the rotation loss $\mathcal{L}_p$, so an $\alpha = 1$ means that for the overall loss the rotation loss has the same importance as the classification one. In the JiGen paper [37] they tried several combination and found for their setting an optimal value of $\alpha = 0.9$, in this work I tried all the values in the range $\alpha = \{0.9, 0.6, 0.3, 0.1\}$. The other important parameter is β, which is a data bias that regulates the ratio between the number of images that are used for the main task and the ones used for the secondary task. For instance, $\beta = 0.6$ means that for each batch 60% of the images are normal and 40% are rotated. In the JiGen paper they found the best value of beta as 0.6, in this work I run the experiments by trying all the values in the range $\beta = \{0.9, 0.6, 0.3\}$.

For the other details/parameters I followed [37]: the model is trained with an SGD solver, 30 epochs, batch size=128 and learning rate=0.001, which is stepped down to 0.0001 after 80% of the training epochs. Data augmentation is performed by randomly cropping the images to retrain between 80 - 100% and randomly applied horizontal flipping. As mentioned before, the network used is the CaffeNet pretrained on places365, whose last fully connected layer is removed and substituted with a layer which can be either the classifier or the rotations.

4.2.6 Training process

Each run of the algorithm is repeated three times, and the final result is the average over the three repetitions (split 0, split 1, split 2). The images of the source domains are divided into training and validation, the network is trained for 30 epoch and on each epoch is tested on the validation set. After the end of epoch 30, we keep the weights that generated the best result on the validation set, and use them to test the network onto the test set (target domain). This is due to the fact that at training time the accuracy on the test set cannot be used directly to choose the best configuration, because it would make the model dependant on the test set. For this reason the model considered optimal is the one that gains the highest score on the validation set.

Chapter 5

The datasets

5.1 Introduction

The architecture presented in chapter 4 has been tested on two datasets: COLD and BDD100k, whose details and characteristics are presented in the following sections.

5.2 COLD

COLD [44] is an acronym which stands for COsy (Cognitive Systems for Cognitive Assistants) Localization Database, it's made up of three datasets (COLD-Freiburg, COLD-Ljubljana, COLD-Saarbrücken) which contain indoor image sequences collected inside three laboratories in three different european cities: the Autonomous Intelligent Systems Laboratory at the University of Freiburg, Germany; the Visual Cognitive Systems Laboratory at the University of Ljubljana, Slovenia; and the Language Technology Laboratory at the German Research Center for Artificial Intelligence in Saarbrücken, Germany. The data were collected using three different mobile robot platforms and the same camera setup, under three different illumination conditions: cloudy weather, sunny weather and at night. The pictures were taken over several days.

5.2.1 COLD-Freiburg

The COLD-Freiburg dataset contains 26 image sequences collected by a camera mounted on an ActivMedia PeopleBot robitic platform. The acquisition was performed in two different parts of the laboratory environment (part A and part B), and for each part the robot took the pictures by following two different paths: standard and extended (figure 5.1). The data were acquired in 14 rooms belonging to 8 room categories, including regular and omni-directional images, laser range scans and odometry.

Figure 5.1 shows the structure of the laboratory and the paths that the robot followed while collecting the images. Among the 8 room categories contained in the laboratory, only four of them are selected for the experiments of this thesis that are: printer area, corridor, bathroom and office. The reason of this selection is because these are the only classes that appear in all the selected sequences of the three datasets.

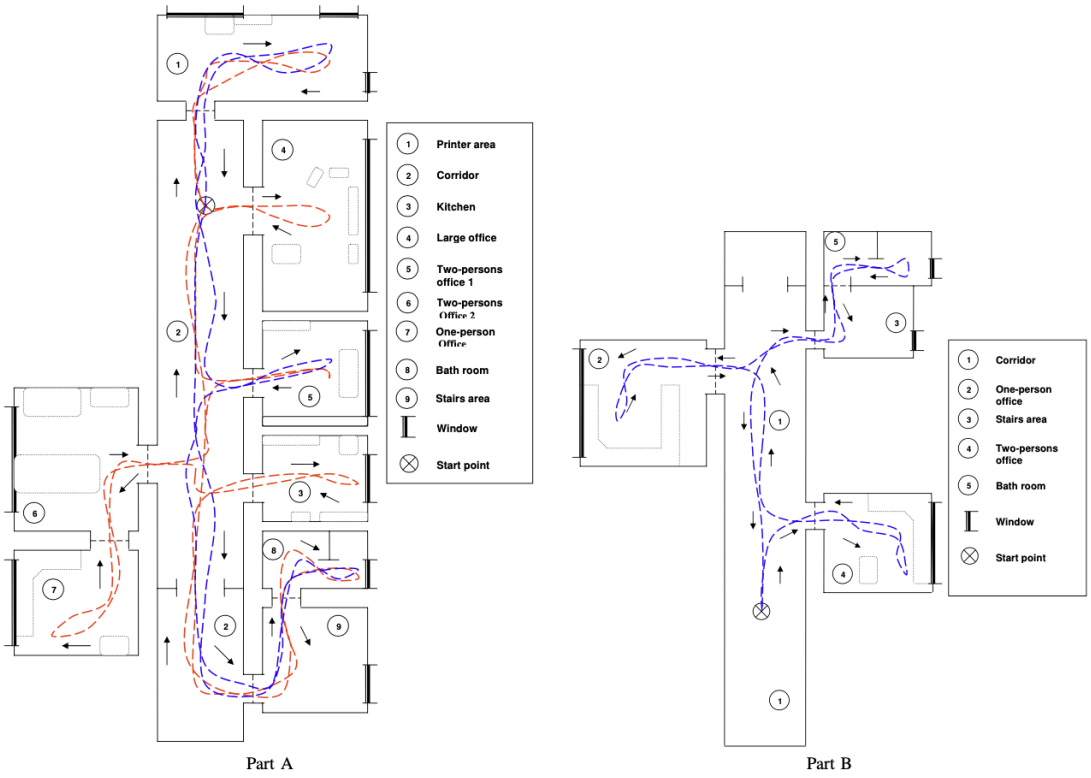
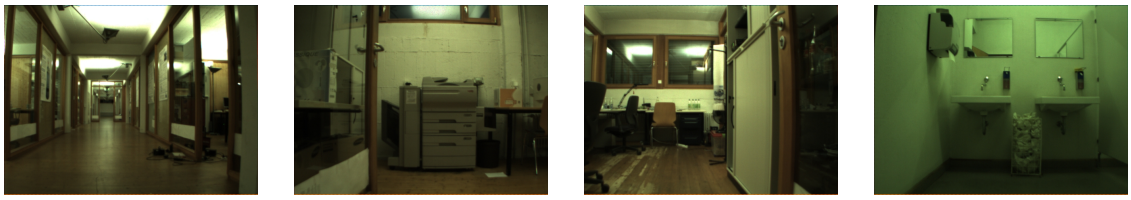


Figure 5.1: Maps of the two parts of the laboratory in Freiburg with approximate paths followed by the robot during data acquisition. The standard path is represented with blue dashes and the extended path is represented with red dashes. Arrows indicate the direction in which the robot was driving. Image taken from [44].

In figure 5.2 and 5.3 are shown some samples for each light condition and for each selected class analyzed.

(a) corridor (b) printer area (c) office (d) bathroom

Figure 5.2: Cloudy samples from COLD-Freiburg for each class



(a) corridor (b) printer area (c) office (d) bathroom

(e) corridor (f) printer area (g) office (h) bathroom

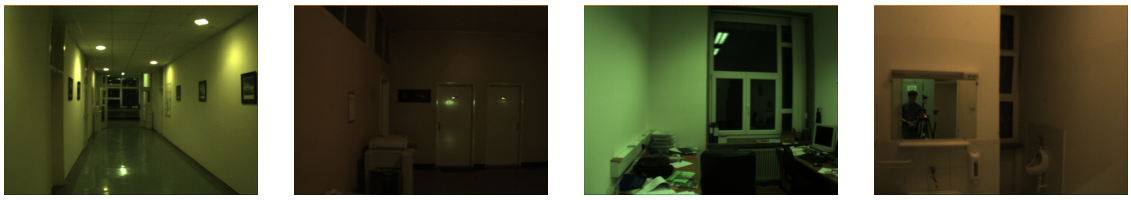
Figure 5.3: Night (row 1) and sunny(row 2) samples from COLD-Freiburg for each class

5.2.2 COLD-Ljubljana

The COLD-Ljubljana dataset is made up of 18 sequences of images taken by a camera located on an iRobot ATRV-Mini robotic platform, in this case only one part of the environment was considered (part A) and as before it followed two paths: standard and extended. The laboratory includes 6 rooms which resulted in 6 categories. Also in this case the data included come from different sources: regular and omni-directional images, and odometry.

Figure 5.4 and 5.5 show some samples of the dataset, while figure 5.6 shows the structure of the laboratory and the paths that the robot followed while collecting the images. Among the 6 room categories contained in the laboratory, were selected also in this case the same four classes: printer area, corridor, bathroom and office.

(a) corridor (b) printer area (c) office (d) bathroom



(e) corridor (f) printer area (g) office (h) bathroom

Figure 5.4: Cloudy (row 1) and night (row 2) samples from COLD-Ljubljana for each class

(a) corridor (b) printer area (c) office (d) bathroom

Figure 5.5: Sunny samples from COLD-Ljubljana for each class

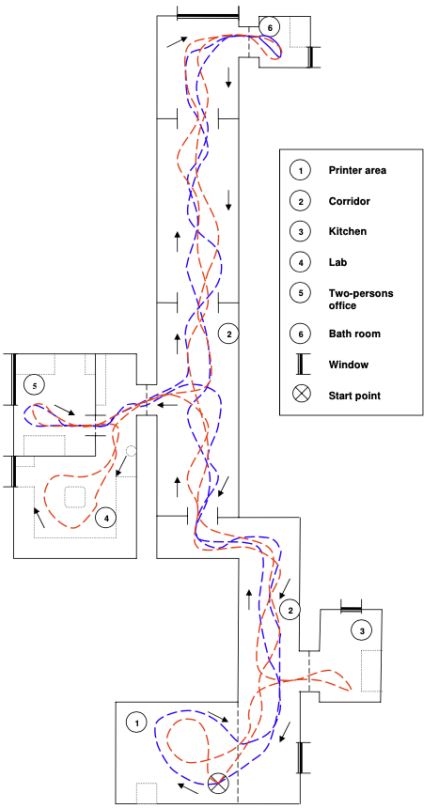


Figure 5.6: Maps of the two parts of the laboratory in Saarbrücken with approximate paths followed by the robot during data acquisition. The standard path is represented with blue dashes and the extended path is represented with red dashes. Arrows indicate the direction in which the robot was driving. Image taken from [44].

5.2.3 COLD-Saarbrücken

COLD-Saarbrücken contains 32 sequences acquired by an ActivMedia PeopleBot robotic platform at the Language Technology Laboratory at the German Research Center for Artificial Intelligence in Saarbrücken. The acquisition was performed in two different parts of the same office environment (part A and part B) and in each part the robot followed

the standard and the extended path, as in the previous cases. The sequences involve 13 rooms belonging to 9 classes. The structure of the laboratory is shown in figure 5.7, while figures 5.8 and 5.9 display some samples of the images.

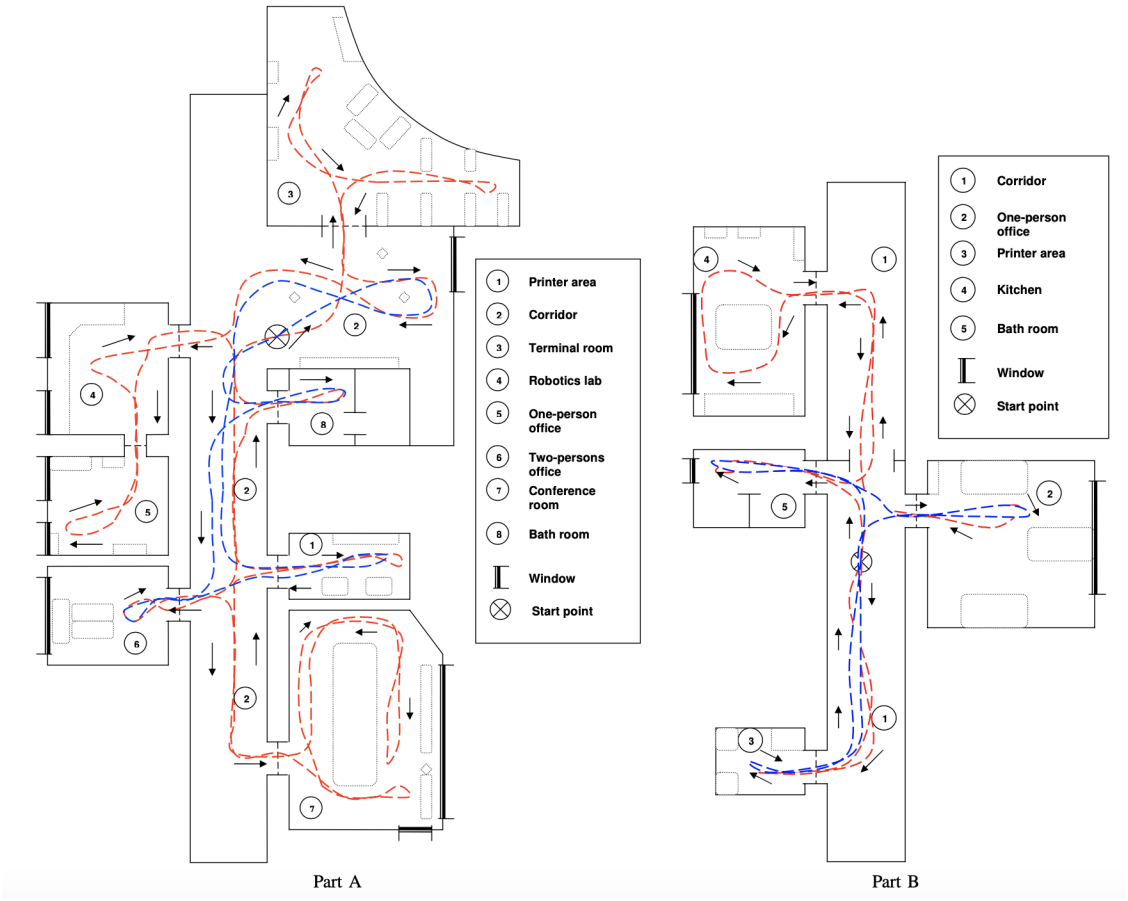
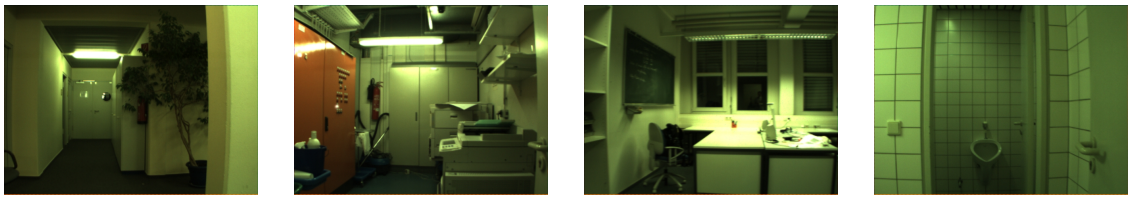


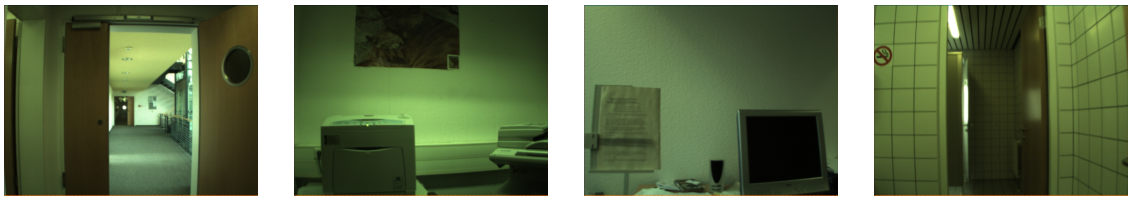
Figure 5.7: Maps of the two parts of the laboratory in Saarbrückenn with approximate paths followed by the robot during data acquisition. Image taken from [44].

(a) corridor (b) printer area (c) office (d) bathroom

Figure 5.8: Cloudy samples from COLD-Saarbrückenn for each class



(a) corridor (b) printer area (c) office (d) bathroom



(e) corridor (f) printer area (g) office (h) bathroom

Figure 5.9: Night (row 1) and sunny (row 2) samples from COLD-Saarbrückenn for each class

5.3 BDD100K

BDD100K [45] stands for "Berkley Deep Drive", it's a dataset containing 100,000 driving videos collected from more than 50,000 rides, resulting in about 120,000,000 images. The sequences were collected by driving across many regions, as shown in figure 6.4.

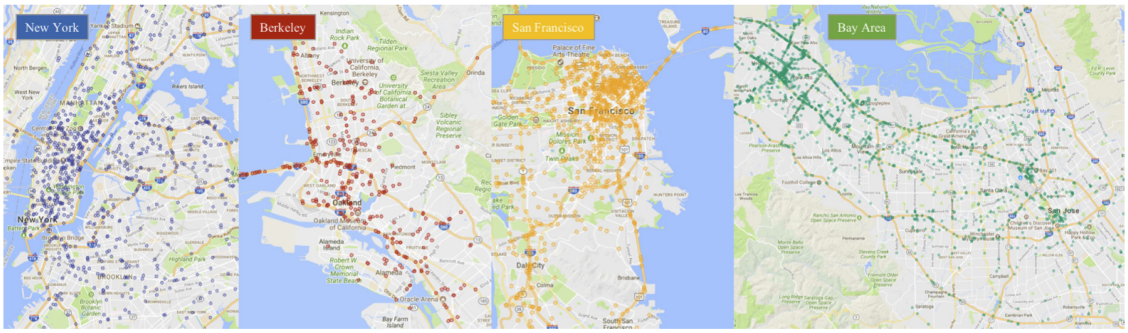
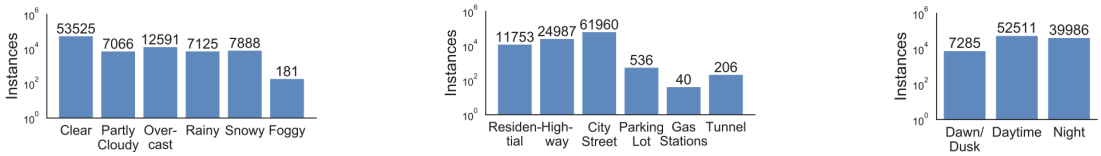


Figure 5.10: Geographical distribution of sample data in four major regions (1000 samples in each region). Each dot represents the starting location of every video clip

From each of the videos, the authors extracted and annotated the frame at the 10th second. The resulting images represent different scenes such as urban roads, residential areas and highways, in different times of the day and weather conditions, as shown in figure 5.11.

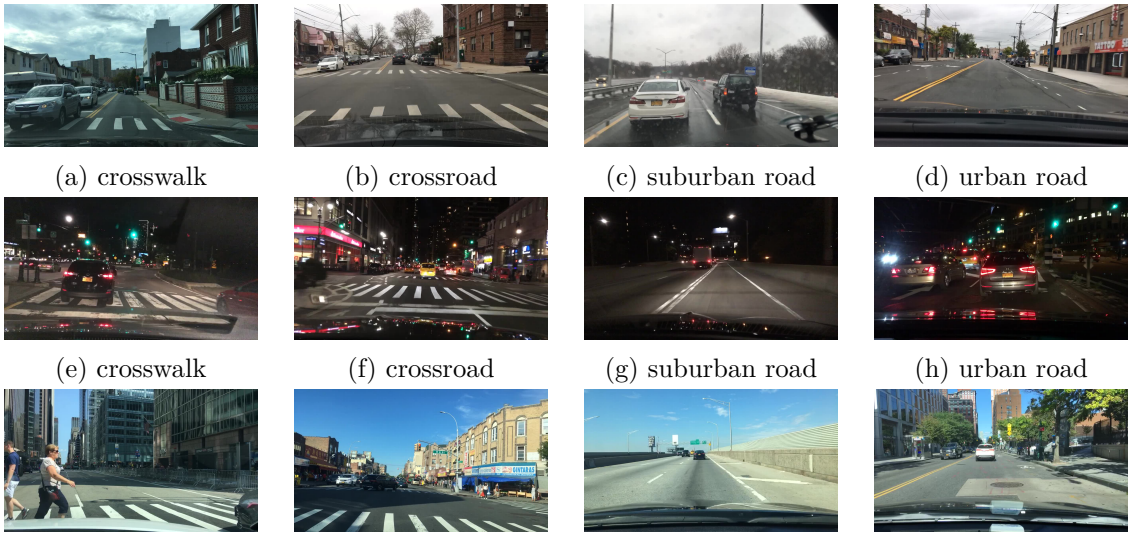
The dataset includes also bounding box annotations for 10 categories and other details, but they were not considered in this work because it's purpose is to classify the images just at scene level. For the experiments in chapter 6 the scene labels listed in 5.11 b couldn't be used, because there are too few images for the classes parking lot, gas station and tunnel



(a) weather (b) scene (c) time

Figure 5.11: Distribution of images in weather, scene, and day hours categories. Image taken from [45].

and I needed at least four classes, so by looking at the dataset I collected the 4 classes that were more frequent: urban road, suburban road (including the highways), crossroad and crosswalk. For each of these classes have been considered images in three light conditions: cloudy, night and sunny. Figure 5.12 shows a sample image for each light condition and for each class. By looking at the picture it can be noticed that the first two classes (i.e. crosswalk and crossroad) appear to be very similar, because also the crossroad has almost always one or more crosswalks. The difference is that the crosswalk class consider only a pedestrian crosswalk with no roads that cross the main one, while instead the crossroad covers the cases where there is clearly a road that crosses the one where the car is travelling.



(a) crosswalk (b) crossroad (c) suburban road (d) urban road

(e) crosswalk (f) crossroad (g) suburban road (h) urban road

(i) crosswalk (j) crossroad (k) suburban road (l) urban road

Figure 5.12: Cloudy (row 1), night (row 2) and sunny (row 3) samples from BDD100K for each class.

Chapter 6

Experiments

6.1 Introduction

The experiments have been done by using the datasets COLD and BDD100K (see chapter 5). For the COLD database, I followed the experimental settings used in [37] and considered only a subset of the available images: for each laboratory and light condition were considered only the standard sequences 1 of part A, except for Saarbrücken Cloudy for which was used sequence 2 because the authors of the database declared that there were some acquisition issues with sequence 1, and for Saarbrücken Sunny for which was used sequence 1 of part B because part A doesn't contain sunny sequences.

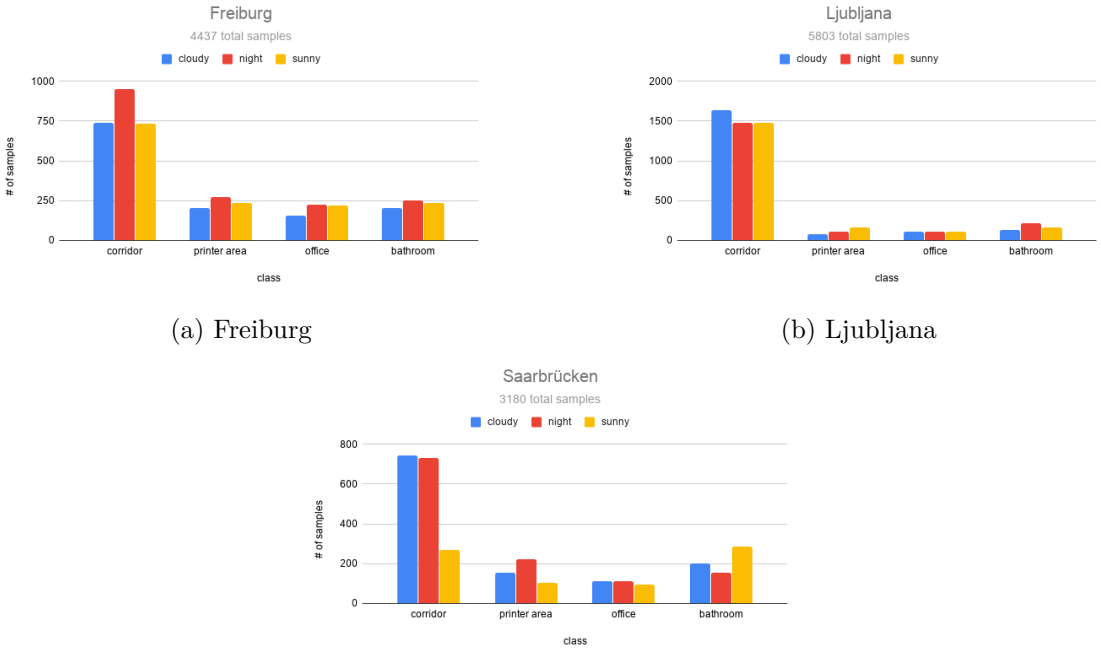
The BDD100K dataset instead was quite challenging because it didn't have the labels that I needed for the experiments, so I considered a small part of it and I added the labels by hand, collecting more or less 2000 images.

Each result presented in the tables in this chapter is the average of three runs of the same experiment (step 0, step 1 and step 2), in order to minimize the error.

6.2 COLD dataset

For the COLD dataset I considered the only four classes that were shared by all the image sequences of the three laboratories: printing area, corridor, office and bathroom. The class office collects the images from both the classes one-person and two-person office that appear in the original database labeling system.

In figure 6.1 is shown the distribution of the images for each laboratory environment, according to the different class and light condition. As can be seen, the database is really unbalanced: the most of the images belong to the class corridor, especially in the case of Ljubljana (fig. 6.1 b), while the other three classes have a much lower number of samples. This is due to the structure of the laboratories, where the corridor occupy most of the space, and so the robot spent much more time on it while taking the pictures.



(a) Freiburg

(b) Ljubljana

(c) Saarbrücken

Figure 6.1: Distribution of the images of the COLD database according to class and light condition

6.2.1 Problem introduction

The dataset presents two domain shifts: one related to the light condition (Cloudy, Night, Sunny) and the other related to the laboratory where the pictures were taken (Saarbrücken, Freiburg and Ljubljana). So two different problems can be analyzed in this case:

- **Problem 1**: the domain shift is assumed to be on the light condition, the network is trained on sequences of the same laboratory, training on two light conditions and testing on the third. The experiment is repeated for each combination of light conditions and for each laboratory (e.g. images from Lab 1, train on Cloudy and Sunny and test on Night images etc.).

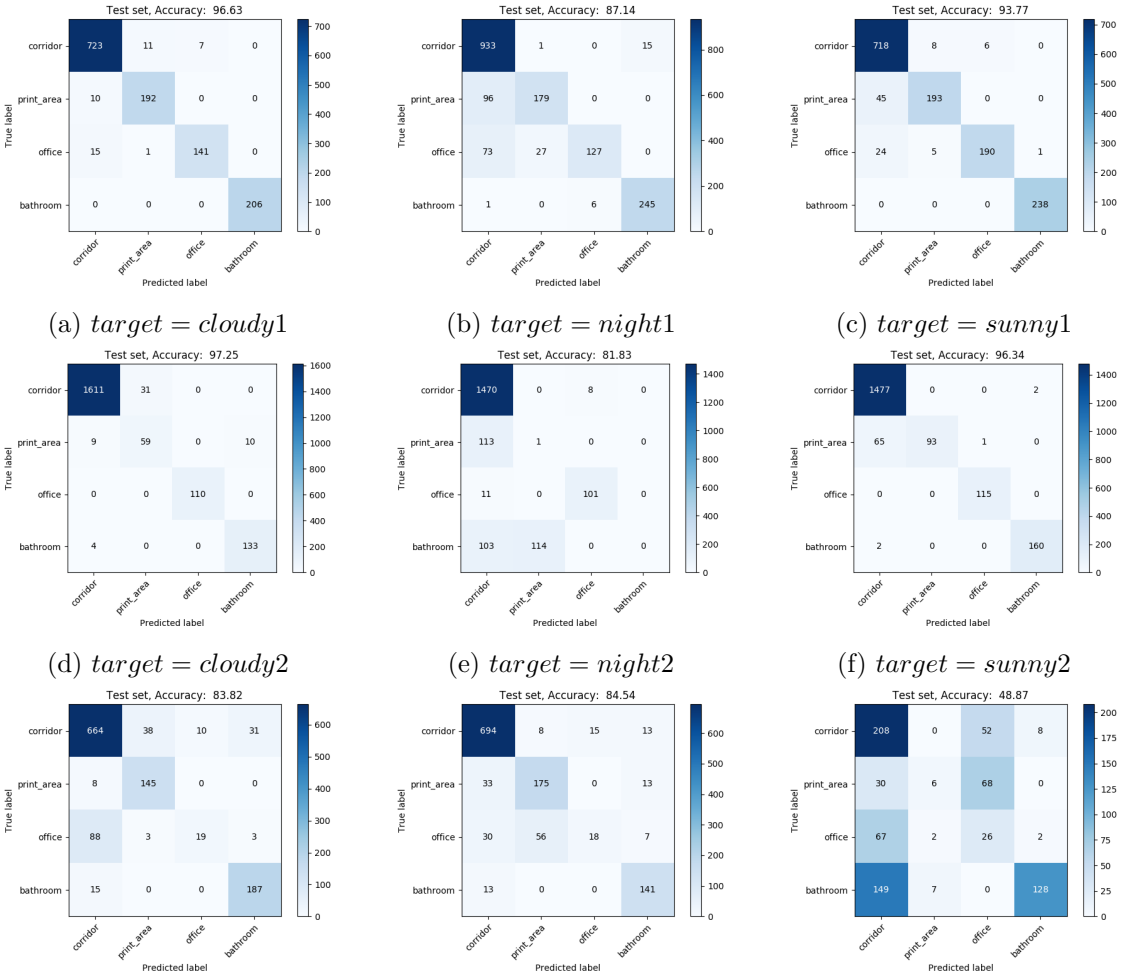
- **Problem 2**: the domain shift is the change of the laboratory, the network is trained on sequences of the same light condition, training on two laboratories and testing on the third. The experiment is repeated for each combination of laboratory and for each light condition. (e.g. Cloudy images, train on Lab1 and Lab2 and test on Lab3 images etc.).

6.2.2 Domain Generalization

The first way to handle the problem is domain generalization, where it's assumed that there are no images of the target domain available during the training phase. First of

all, I considered Problem 1 where the laboratory is kept constant and the domain shift is the light condition, and run an experiment without using the secondary task, to see how accurate is the CNN when tested on a domain that it has never seen during training. The result is shown in the first row of table 6.1, characterized by the input parameters alpha=0 and beta=1 (i.e. alpha=0 means that the secondary task will have a weight=0 in the loss, and beta=1 is saying that all the images are used only for classification). This experiment generates nine results: three results (i.e. one for each light condition) for each laboratory. The last column (i.e. *avg*) shows the average of the previous nine results. The average accuracy in this case is 85.95%, it's a good result considering that the network was tested on an unknown domain and means that in general the three domains are similar. In figure 6.2 are shown the confusion matrices for each of the nine sub-experiments: each experiment as mentioned before was run three times (step 0, step 1 and step 2), these matrices are related to the first run (step 0). The matrices show that the results are very good for Freiburg and Ljubljana, with accuracies that are above 90% when the target domains are cloudy and sunny, while it's a bit lower when testing on night images as could be expected, because it's the more relevant domain shift among the three. The results on Saarbrücken (fig. 6.2 g, h, i) instead are lower, especially in the case target=sunny, that is quite different from the others because it has a different distribution of images in the four classes: in particular, the corridor has much less pictures compared to the proportions of the other cases, and many images of the other classes are confused as corridor. The accuracy in this case is only less than 50%. After this first experiment, the secondary task was turned on and the algorithm was tested with different combinations of alpha and beta considering alpha = [0.9, 0.6, 0.3, 0.1] and beta = [0.9, 0.6, 0.3]. The results are shown in table 6.1, where each row correspond to a different configuration of the input parameters. By looking at these results, we can see that the secondary task is helping the network to classify better the images, because the average accuracy is almost always better than the one of the configuration alpha=0, beta=1. In particular, the best result is obtained with the combination alpha=beta=0.6, where 60% of the images are used for classification and the remaining 40% for the secondary task. The worst results are obtained with beta = 0.3, and the reason is that in this case only 30% of the images are used for classification, a too low percentage that causes a reduction of accuracy. I also tried some experiments with beta=0.1 and the average accuracy decreased further, giving an accuracy below 85%.

Table 6.2 shows instead the results for Problem 2. This case is similar to Problem 1, with the difference that here the light condition is kept constant and the domain shift is the change of the laboratory, so for each light condition the network is trained on images of two laboratories and tested on the third. As in the previous case the experiment is repeated for each combination of light condition and for each laboratory, generating nine results for each run plus their average. The first thing that can be noticed from this table is that the accuracy is quite lower than in problem 1: considering for example the baseline case, for problem 1 the accuracy was about 86% while now it's about 60%. This happens because the domain shift of the laboratory is much more relevant than the one on the light condition, because the rooms of the three laboratories are very different. The confusion matrices of this experiment (step 0) are displayed in figure 6.3, and show that in this case the network manages to recognize properly only the corridor, while fails in recognizing the other three classes: the reason of this behaviour is that the features that were learnt for the classes office, printer area and bathroom are limited to the source domains, because



(a) $target = cloudy1$ (b) $target = night1$ (c) $target = sunny1$

(d) $target = cloudy2$ (e) $target = night2$ (f) $target = sunny2$

(g) $target = cloudy3$ (h) $target = night3$ (i) $target = sunny3$

Figure 6.2: Confusion matrices of the baseline for Problem 1 (Step 0). The first row shows the results with target domain t_d= Freiburg (a, b, c), second row with t_d=Ljubljana (d, e, f), third row with t_d=Saarbrücken (g, h, i).

the obtained accuracy on the validation set is always very high ($> 90\%$), but they are not suitable for the target domain. In this case we can see that the best result is the one with alpha=0 and beta=1, that is the one obtained with the secondary task turned off, so here the task is not only not helping the classification, but it seems that it's going in conflict with it. The reason of this behaviour is probably that the domain shift among the source domains and the target domains is too huge and so the secondary task doesn't manage to learn features that can be helpful across all the domains.

	Freiburg			Ljubljana			Saarbrücken			avg
	Cloudy	Night	Sunny	Cloudy	Night	Sunny	Cloudy	Night	Sunny	
$\alpha = 0$, $\beta = 1$	96.99	90.62	95.85	97.25	81.68	96.33	84.92	83.31	46.53	85.94
$\alpha = 0.9$, $\beta = 0.9$	97.29	92.29	94.33	96.83	**83.03**	95.56	86.76	84.21	48.21	86.5
$\alpha = 0.9$, $\beta = 0.6$	97.58	89.49	96.17	96.56	82.53	96.21	83.79	84.81	49.98	86.35
$\alpha = 0.9$, $\beta = 0.3$	**98.11**	89.1	94.7	97.12	82.42	95.63	83.7	**85.03**	45.15	85.66
$\alpha = 0.6$, $\beta = 0.9$	97.45	92.58	94.65	97.0	82.72	96.07	86.29	83.94	49.23	86.66
$\alpha = 0.6$, $\beta = 0.6$	97.45	90.31	**97.48**	96.97	82.79	95.86	86.02	84.29	49.89	**86.78**
$\alpha = 0.6$, $\beta = 0.3$	97.63	90.59	93.63	**97.27**	82.47	95.86	84.56	84.13	47.72	85.98
$\alpha = 0.3$, $\beta = 0.9$	97.6	92.01	94.96	97.09	82.32	95.74	**86.95**	83.88	48.74	86.59
$\alpha = 0.3$, $\beta = 0.6$	97.52	**93.09**	95.1	96.75	82.35	**96.5**	85.88	83.36	49.18	86.64
$\alpha = 0.3$, $\beta = 0.3$	97.5	88.49	95.77	96.71	82.02	95.65	86.07	83.58	46.04	85.76
$\alpha = 0.1$, $\beta = 0.9$	97.7	92.5	96.24	96.95	81.83	95.94	86.46	81.91	**50.2**	86.64
$\alpha = 0.1$, $\beta = 0.6$	97.65	92.15	95.42	96.98	82.72	96.21	85.11	84.05	46.92	86.36
$\alpha = 0.1$, $\beta = 0.3$	97.06	92.33	96.1	96.92	82.35	95.81	85.8	82.37	50.07	86.53

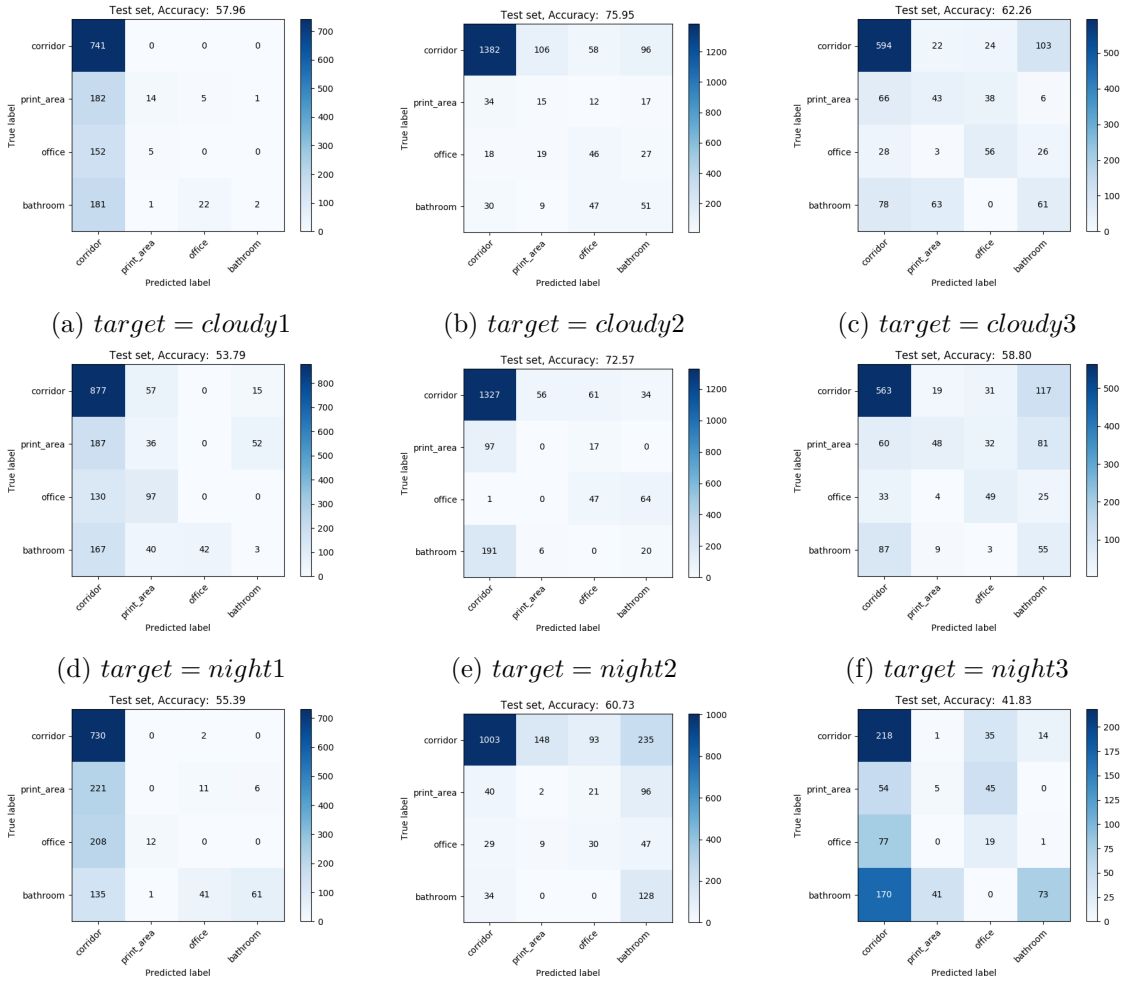
Table 6.1: Results obtained by keeping constant the laboratory and by changing the light condition. Each row represents a run with a different combination of alpha and beta, each column represents the domain that was used as target. The avg column reports the average of the nine results for each run.

	Cloudy			Night			Sunny			avg
	Frei.	Ljub.	Saar.	Frei.	Ljub.	Saar.	Frei.	Ljub.	Saar.	
$\alpha = 0$, $\beta = 1$	57.48	77.24	**62.26**	54.2	71.92	55.76	55.39	64.28	40.86	**59.93**
$\alpha = 0.9$, $\beta = 0.9$	**59.29**	76.56	58.6	**57.76**	**72.65**	53.43	**57.45**	59.9	36.79	59.16
$\alpha = 0.9$, $\beta = 0.6$	57.91	71.38	59.98	57.0	69.37	53.43	56.61	61.78	41.35	58.75
$\alpha = 0.9$, $\beta = 0.3$	57.55	74.33	59.18	55.51	68.82	**57.92**	56.23	59.93	40.15	58.85
$\alpha = 0.6$, $\beta = 0.9$	58.37	75.92	58.63	57.27	71.11	52.22	55.58	64.07	39.75	59.21
$\alpha = 0.6$, $\beta = 0.6$	57.78	68.75	60.14	55.31	70.69	55.29	54.83	63.26	36.74	58.09
$\alpha = 0.6$, $\beta = 0.3$	57.78	72.19	58.3	54.53	71.53	56.25	55.02	66.77	39.93	59.14
$\alpha = 0.3$, $\beta = 0.9$	58.52	75.7	58.99	56.68	68.75	56.47	57.12	62.59	**41.39**	59.58
$\alpha = 0.3$, $\beta = 0.6$	57.78	74.63	58.02	56.49	71.46	52.08	56.33	62.18	40.73	58.86
$\alpha = 0.3$, $\beta = 0.3$	57.17	75.36	61.52	54.0	70.07	54.33	56.56	61.24	39.0	58.81
$\alpha = 0.1$, $\beta = 0.9$	58.35	79.39	56.76	56.72	70.61	54.61	55.81	65.57	40.81	59.85
$\alpha = 0.1$, $\beta = 0.6$	57.71	**80.56**	60.64	54.41	68.87	54.28	54.37	**67.22**	39.4	59.72
$\alpha = 0.1$, $\beta = 0.3$	57.86	74.06	58.63	54.65	68.33	55.59	55.74	65.83	40.64	59.04

Table 6.2: Results obtained by keeping constant the light condition and by changing the laboratory. Each row represents a run with a different combination of alpha and beta, each column represents the domain that was used as target. The avg column reports the average of the nine results for each run.

6.2.3 Domain Adaptation

In the last section we saw that the rotations task helps the classification only in the case of problem 1, while in the second case the accuracy gets lower. To improve the results also for problem 2, I looked for a way to extract some features that can be meaningful



(a) $target = cloudy1$ (b) $target = cloudy2$ (c) $target = cloudy3$

(d) $target = night1$ (e) $target = night2$ (f) $target = night3$

(g) $target = sunny1$ (h) $target = sunny2$ (i) $target = sunny3$

Figure 6.3: Confusion matrices of the baseline for Problem 2 (Step 0)

also for the target domain, and I decided to try with Domain Adaptation. In the Domain Generalization case, the images that are used for the secondary task belong only to the source domain, because it's assumed that there are no samples available from target domain at training time. In this case instead, the secondary task receives as input images from both the source and the target domain, so that the algorithm can use them to peek into the target domain and to learn some features from the rotations that can be helpful also in classifying the target samples. Of course the images from the target domain are used only for the secondary task and not for classification, and they are sent without the label because the task is completely unsupervised. The architecture is very similar to the previous one, but this time more images will be available for the secondary task: the parameter beta in this case represents both the percentage of images used for the rotations from the source domains, and the percentage of images from the ones available from the target domain (e.g. beta=0.6 means that for the secondary task will be used 40% of the image of the

source and 40% of the ones of the target). The results of Problem 1 are shown in 6.3, and we can notice that they are quite similar to the ones with Domain Generalization, with a maximum accuracy of 86.86 with alpha=0.6 and beta=0.3. So, in this case where the domain shift is the light condition, using some images from the target does not improve significantly the accuracy of the primary task, probably because the target domain is not so different compared to the source ones, so the features that are learnt with the rotations on the source images are already sufficient to get an improvement, as we already saw in Table 6.1.

	Freiburg			Ljubljana			Saarbrücken			avg
	Cloudy	Night	Sunny	Cloudy	Night	Sunny	Cloudy	Night	Sunny	
$\alpha = 0$, $\beta = 1$	96.99	90.62	95.85	97.25	81.68	96.33	84.92	83.31	46.53	85.94
$\alpha = 0.9$, $\beta = 0.9$	97.4	86.85	94.72	96.66	**82.8**	96.57	86.73	83.91	49.76	86.16
$\alpha = 0.9$, $\beta = 0.6$	96.99	88.12	96.66	97.14	82.68	95.61	84.75	83.42	49.49	86.1
$\alpha = 0.9$, $\beta = 0.3$	97.29	84.32	95.1	96.95	82.7	95.2	83.29	81.55	48.34	84.97
$\alpha = 0.6$, $\beta = 0.9$	97.52	90.6	96.03	97.05	82.63	**96.97**	84.53	82.95	46.7	86.11
$\alpha = 0.6$, $\beta = 0.6$	97.45	89.94	96.78	96.9	82.35	95.94	85.08	83.99	51.22	86.63
$\alpha = 0.6$, $\beta = 0.3$	97.4	90.57	**97.01**	96.97	82.61	95.82	84.81	**84.65**	**51.88**	**86.86**
$\alpha = 0.3$, $\beta = 0.9$	97.14	89.23	96.71	97.12	82.16	96.21	**87.48**	83.66	47.1	86.31
$\alpha = 0.3$, $\beta = 0.6$	97.45	**92.27**	96.36	96.78	82.53	95.84	85.38	83.25	46.97	86.31
$\alpha = 0.3$, $\beta = 0.3$	**97.88**	91.43	95.33	96.42	82.67	95.82	83.62	81.17	50.07	86.05
$\alpha = 0.1$, $\beta = 0.9$	97.45	90.43	93.56	**97.36**	82.37	96.08	86.48	83.42	47.28	86.05
$\alpha = 0.1$, $\beta = 0.6$	97.32	88.67	93.18	96.63	82.3	95.77	85.47	83.66	47.59	85.62
$\alpha = 0.1$, $\beta = 0.3$	97.24	89.0	96.48	96.85	82.39	96.43	83.18	81.66	49.31	85.84

Table 6.3: Results with Domain Adaptation for problem 1, obtained by keeping constant the laboratory and by changing the light condition. Each row represents a run with a different combination of alpha and beta, each column represents the domain that was used as target. The avg column reports the average of the nine results for each run.

The situation changes significantly in the setting of Problem 2, where we noticed in Table 6.2 that the features learned only on the source domains were not meaningful for the target. Table 6.4 reports the results with Domain Adaptation, and they show a significant improvement from the ones with DG. In this case the secondary task allows to obtain an accuracy slightly higher than the baseline result, but by observing the avg results for each run where the secondary task is used, the scores are much better than the ones with DG.

The best score of 60.39% is obtained with alpha=0.6 and beta=0.6, which is much better than the 58.09% obtained with the same parameters with Domain Generalization: if we look more in details the results, there is a huge improvement especially when we test on Saarbrücken cloudy, where with DG the accuracy is 68.75%, while with DA is 80.94% .

Also in this case however the results are only slightly better than the baseline, so I tried to change the configuration in order to see if I could get a bigger improvement. After many tries, I managed to obtain a better accuracy by changing the beta impact on the percentage of images of the target used: in the previous results, a beta=0.6 meant that 60% of the source domains images are used for the primary classification task, while the remaining 40% of the source plus another 40% of the images of the target are given rotated to the secondary task. My speculation was that the more target images the secondary task

	Cloudy			Night			Sunny			avg
	Frei.	Ljub.	Saar.	Frei.	Ljub.	Saar.	Frei.	Ljub.	Saar.	
$\alpha = 0$, $\beta = 1$	57.48	77.24	62.26	54.2	71.92	55.76	55.39	64.28	40.86	59.93
$\alpha = 0.9$, $\beta = 0.9$	57.94	77.85	59.68	55.24	72.83	55.35	55.53	66.53	37.72	59.85
$\alpha = 0.9$, $\beta = 0.6$	57.43	80.56	60.8	54.65	71.79	57.35	54.69	65.97	38.29	60.17
$\alpha = 0.9$, $\beta = 0.3$	56.94	82.75	56.98	55.08	73.5	55.81	54.15	64.75	39.58	59.95
$\alpha = 0.6$, $\beta = 0.9$	57.78	75.53	60.61	54.9	72.48	56.47	56.51	66.35	38.51	59.91
$\alpha = 0.6$, $\beta = 0.6$	57.3	80.94	62.7	**55.9**	73.28	53.32	53.92	**67.12**	39.04	**60.39**
$\alpha = 0.6$, $\beta = 0.3$	57.43	**82.9**	62.7	55.84	69.72	56.03	55.11	59.03	38.69	59.72
$\alpha = 0.3$, $\beta = 0.9$	58.04	78.94	61.52	55.0	70.5	52.74	**57.03**	65.19	**40.86**	59.98
$\alpha = 0.3$, $\beta = 0.6$	57.76	80.22	59.12	54.96	**75.29**	**57.65**	54.15	63.08	40.02	60.25
$\alpha = 0.3$, $\beta = 0.3$	57.55	80.95	61.22	55.51	74.51	56.11	53.57	65.99	36.21	60.18
$\alpha = 0.1$, $\beta = 0.9$	57.73	76.53	59.45	55.12	70.4	56.88	55.53	66.32	40.81	59.86
$\alpha = 0.1$, $\beta = 0.6$	**58.47**	77.58	62.79	55.39	74.35	56.61	54.72	63.57	39.58	60.34
$\alpha = 0.1$, $\beta = 0.3$	57.2	79.83	**63.8**	55.28	69.76	57.48	55.79	62.52	39.18	60.09

Table 6.4: Results with Domain Adaptation for problem 2, obtained by keeping constant the light condition and by changing the laboratory. Each row represents a run with a different combination of alpha and beta, each column represents the domain that was used as target. The avg column reports the average of the nine results for each run.

	Cloudy			Night			Sunny			avg
	Frei.	Ljub.	Saar.	Frei.	Ljub.	Saar.	Frei.	Ljub.	Saar.	
$\alpha = 0$, $\beta = 1$	57.48	77.24	62.26	54.2	71.92	55.76	55.39	64.28	40.86	59.93
$\alpha = 0.9$, $\beta = 0.9$	57.63	80.43	60.14	55.55	74.53	57.81	54.65	68.98	34.71	60.49
$\alpha = 0.9$, $\beta = 0.6$	57.15	83.17	**62.76**	56.0	70.97	**57.98**	55.49	66.01	39.53	61.0
$\alpha = 0.9$, $\beta = 0.3$	57.15	**83.19**	59.12	**56.49**	71.87	55.73	54.44	65.2	38.03	60.14
$\alpha = 0.6$, $\beta = 0.9$	57.27	82.21	61.38	55.26	69.88	57.48	54.44	68.16	36.56	60.29
$\alpha = 0.6$, $\beta = 0.6$	57.2	78.27	61.27	54.84	**75.45**	57.84	53.9	**70.25**	37.27	60.7
$\alpha = 0.6$, $\beta = 0.3$	57.45	81.82	60.03	55.12	71.42	57.59	54.93	64.61	37.14	60.01
$\alpha = 0.3$, $\beta = 0.9$	**58.01**	82.82	60.75	56.0	71.56	56.99	**57.0**	67.95	38.38	**61.05**
$\alpha = 0.3$, $\beta = 0.6$	57.73	82.38	59.1	54.9	70.4	55.56	54.27	65.45	37.98	59.75
$\alpha = 0.3$, $\beta = 0.3$	57.38	80.8	59.32	55.33	70.45	56.3	54.46	68.7	38.69	60.16
$\alpha = 0.1$, $\beta = 0.9$	57.53	79.09	61.66	55.65	74.53	55.56	52.26	65.0	40.68	60.22
$\alpha = 0.1$, $\beta = 0.6$	57.38	79.68	62.04	55.51	71.87	56.22	55.79	63.15	**41.74**	60.38
$\alpha = 0.1$, $\beta = 0.3$	57.17	77.05	61.13	55.43	71.25	54.85	53.29	65.15	38.96	59.37

Table 6.5: Results obtained with the same setting as table 6.4, but using all the available target images for the secondary task.

gets, the better will be the accuracy of the classification, because the features learned with the rotations are more and more representative of the target domain. So I tried to remove beta for the target images, such that at each run the algorithm gets always all the target images available, while I kept it unchanged for the images of the source domains.

The results of this experiment are shown in Table 6.5, and they prove that the previous speculation was right: in fact we obtain much better results especially with alpha=0.9, beta=0.6 and with alpha=0.3 and beta=0.9, that gain more than one point compared to

the baseline score. Also in the case of alpha=0.6 and beta=0.6, where before we obtained the highest accuracy, in this case the network reaches a score of 60.7%, that is higher than the 60.39% obtained before. I tried to use this configuration also with Problem 1, but the results were quite similar to the ones presented in table 6.3, without significant improvements.

6.3 BDD100K dataset

The BDD100K dataset contains 100.000 video sequences taken in different light and weather conditions in an urban context. I considered a subset of it, containing about 2000 images, and I identified the most recurrent classes which are: crosswalk, crossroad, urban road and suburban road. Each class appears in three light conditions: sunny, night and cloudy. The distribution of the images over the different classes and light conditions is shown in figure 6.4.

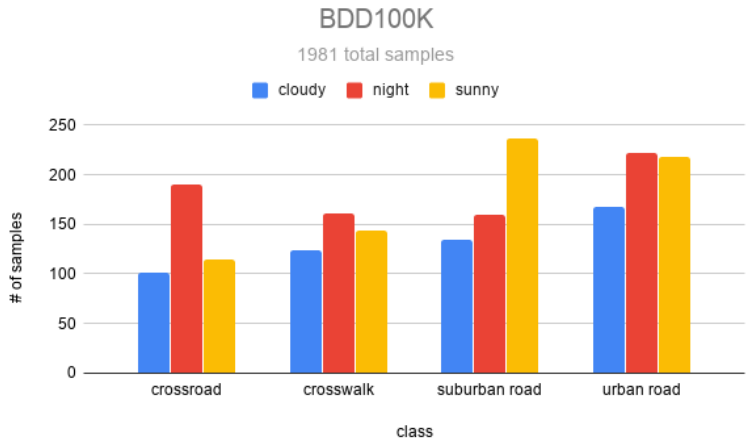


Figure 6.4: Distribution of the images of the BDD100K database subset according to class and light condition

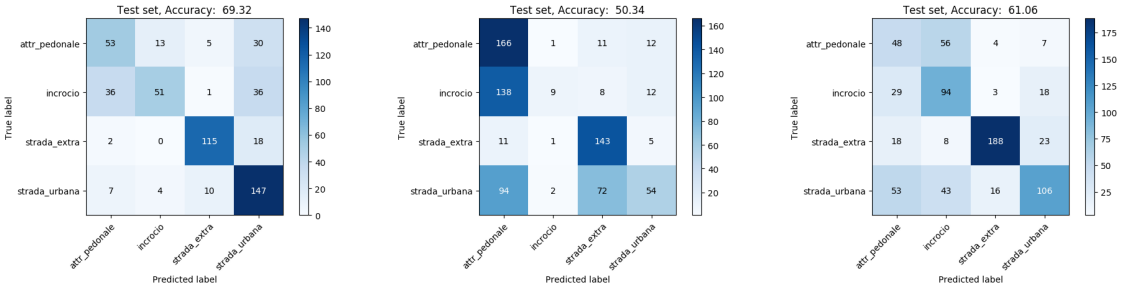
6.3.1 Problem introduction

The dataset presents only one domain shift, the light condition. Another domain shift interesting to analyze could have been the change of weather conditions, because there are also some images with snow and rain, but unfortunately they are too few to perform a good analysis.

The problem to handle then is the following: given the domain shift of the light condition, the network is trained on sequences belonging to two light conditions and tested on the third. The experiment, as well as for the COLD dataset, is repeated for each combination of light condition (e.g. train on Cloudy and Sunny and test on Night images, and so on).

6.3.2 Domain Generalization

The first way to deal with the problem is again to try to generalize through domains only by using the source images, assuming that we haven't any target sample available at training time. So we test again our network with the additional secondary task of the rotations, to see if also in this case this task helps the network to generalize better. First of all, the experiment is performed with the baseline architecture, so the secondary task is turned off by setting alpha=0 and beta=1 as already done with the previous dataset. The result is shown in the first row of table 6.6, which reports an average accuracy of 61.79% over the three possible combinations of target domains, which is not a very high value because only a very small subset of the dataset was used for finetuning the network. If more images were used, the accuracy would of course be higher, but the point of this experiment is not to reach the highest possible accuracy with the baseline, it's instead to show that there is an improvement by including the secondary task in the network, that allows to generalize better over the different domains. In figure 6.5 are displayed the confusion matrices of step 0 for each combination of target domain.



(a) *target = cloudy* (b) *target = night* (c) *target = sunny*

Figure 6.5: Confusion matrices of the baseline for BDD100K (Step 0)

The figure shows that the most difficult domain, as predictable, is the night condition, where the average accuracy is about 50%. The confusion matrix shows that the only classes that are easily recognized are the suburban road and the crosswalk, while the crossroad is almost always confused with the crosswalk and the urban road is often confused with crosswalk and suburban road. In the other cases instead the accuracy is higher, especially when the target is the cloudy domain, where the accuracy is almost 70% and the classes urban and suburban road are very well separated, while there is still a bit of confusion between crosswalks and crossroads: this behaviour is expected because the two classes are very similar, due to the fact that the crossroads that appear in the dataset contain almost always crosswalks. By looking at the three matrices it also appears that the class that is classified better is the suburban road, and also this behaviour is expected because it's the class that is most different from the others (crossroad and crosswalk are always located in an urban road). The results with the secondary task activated are listed in table 6.6, and they are almost always higher than the baseline, with a maximum value of 63.25% of accuracy with alpha=0.3 and beta=0.6.

Figure 6.6 shows a comparison among the confusion matrices of the baseline and the ones of the best result for Domain generalization (alpha=0.3, beta=0.6), and we can see

	Cloudy	Night	Sunny	avg
$\alpha = 0$, $\beta = 1$	69.63	53.0	62.75	61.79
$\alpha = 0.9$, $\beta = 0.9$	**71.4**	53.68	63.54	62.87
$\alpha = 0.9$, $\beta = 0.6$	71.21	55.03	62.46	62.9
$\alpha = 0.9$, $\beta = 0.3$	69.13	**57.06**	62.75	62.98
$\alpha = 0.6$, $\beta = 0.9$	68.5	52.1	60.27	60.29
$\alpha = 0.6$, $\beta = 0.6$	70.33	55.53	61.76	62.54
$\alpha = 0.6$, $\beta = 0.3$	70.08	50.92	61.9	60.97
$\alpha = 0.3$, $\beta = 0.9$	70.77	54.4	62.98	62.72
$\alpha = 0.3$, $\beta = 0.6$	71.21	56.07	62.46	**63.25**
$\alpha = 0.3$, $\beta = 0.3$	69.63	50.56	**64.15**	61.45
$\alpha = 0.1$, $\beta = 0.9$	70.08	53.13	62.98	62.06
$\alpha = 0.1$, $\beta = 0.6$	70.39	53.09	62.7	62.06
$\alpha = 0.1$, $\beta = 0.3$	68.31	54.22	61.95	61.49

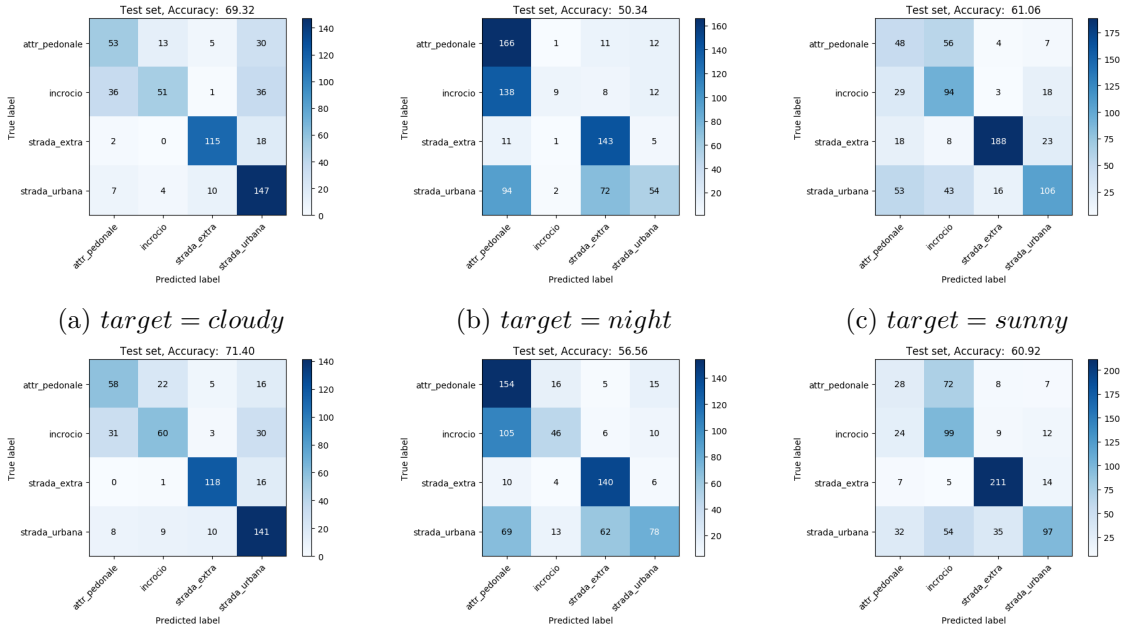
Table 6.6: Results obtained for BDD100K with Domain Generalization. Each row is a run of the experiment with a different combination of the input parameters. Each column represents the domain that was used as target.

an improvement especially in the first two cases, where the target domains are respectively cloudy and night.

6.3.3 Domain Adaptation

Also with this dataset, we can try the approach of domain adaptation to see if we can reach even better results. So the network is trained by giving the images of the source domains to the classification task, and images both from the source and the target (without the labels) to the secondary unsupervised task that performs the rotations. The results are listed in table 6.7, and we can see that they are a bit better than the ones with Domain Generalization. So also in this case we have the confirmation that performing an unsupervised task on the images of the target domain allows to learn features that help the primary classification task to be more accurate than in the case where it works only with images of the source domains.

With this dataset I tried also an approach a bit different than the one used so far. In the previous experiments, each batch contained a part of images that was used for classification and the remaining images were given rotated to be used in the secondary task, and the percentage of the split was given by the input parameter beta as seen so far. This implies that the more beta is low, the less images are used for classification and for that reason with very low values of beta the results are not good (e.g. I tried with beta=0.1 and the results were lower than the others). So I tried instead to launch an experiment where the classification task receives always all the source images, but a percentage of these is duplicated and given rotated to the secondary task according to the



(a) $target = cloudy$ (b) $target = night$ (c) $target = sunny$

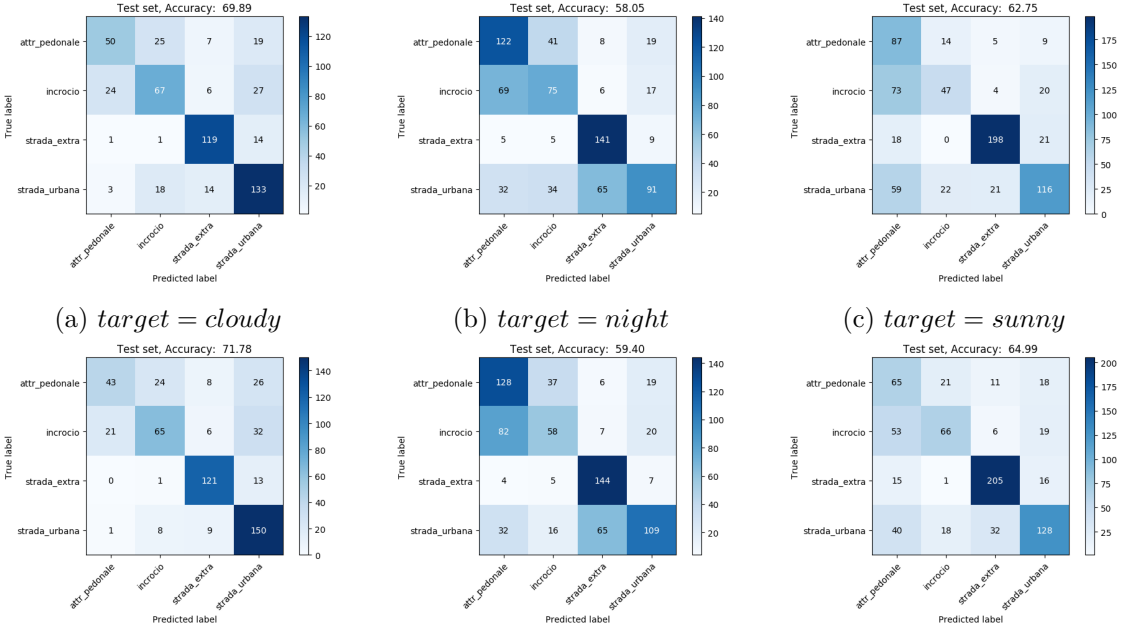
(d) $target = cloudy$ (e) $target = night$ (f) $target = sunny$

Figure 6.6: Confusion matrices at step 0 of the baseline (a, b, c) vs Domain Generalization with $\alpha = 0.3$, $\beta = 0.6$ (d, e, f).

value of beta, plus also a percentage of images from the target domain because we are in the Domain Adaptation case. With this setting, I obtained very good results (Table 6.8) that outperform the ones seen in table 6.7, with a best accuracy of 65.05% with alpha=0.9 and beta=0.1, that is the condition where the 90% of images are also used for the rotations task. Moreover, the results in the table have a predictable trend, because the highest values of the accuracy appear when we use beta = 0.3 and beta=0.1, that are the cases where we are using almost all the available images also for the secondary task, while instead in the runs where beta=0.9 (i.e. only 10% of the images are given rotated to the secondary task) we obtain an accuracy comparable to the baseline. Figure 6.7 shows a comparison between the confusion matrices of the best result of table 6.7 (63.76% with $\alpha = 0.1$ and $\beta = 0.6$) and the best result of table 6.8 (65.05% with $\alpha = 0.9$ and $\beta = 0.1$), and there is a considerable improvement in all the three cases.

	Cloudy	Night	Sunny	avg
$\alpha = 0$, $\beta = 1$	69.63	53.0	62.75	61.79
$\alpha = 0.9$, $\beta = 0.9$	70.39	55.93	63.03	63.12
$\alpha = 0.9$, $\beta = 0.6$	68.37	**58.28**	61.39	62.68
$\alpha = 0.9$, $\beta = 0.3$	68.69	58.19	61.44	62.77
$\alpha = 0.6$, $\beta = 0.9$	70.14	54.08	62.14	62.12
$\alpha = 0.6$, $\beta = 0.6$	70.08	57.42	61.81	63.1
$\alpha = 0.6$, $\beta = 0.3$	69.63	53.5	63.31	62.14
$\alpha = 0.3$, $\beta = 0.9$	**70.96**	55.44	62.56	62.98
$\alpha = 0.3$, $\beta = 0.6$	67.68	53.95	62.98	61.53
$\alpha = 0.3$, $\beta = 0.3$	70.27	54.62	63.26	62.72
$\alpha = 0.1$, $\beta = 0.9$	70.39	56.97	62.0	63.12
$\alpha = 0.1$, $\beta = 0.6$	69.89	57.92	**63.49**	**63.76**
$\alpha = 0.1$, $\beta = 0.3$	67.87	52.95	62.56	61.13

Table 6.7: Results obtained for BDD100K with Domain Adaptation. Each row is a run of the experiment with a different combination of the input parameters. Each column represents the domain that was used as target.



(a) *target = cloudy* (b) *target = night* (c) *target = sunny*

(d) *target = cloudy* (e) *target = night* (f) *target = sunny*

Figure 6.7: Confusion matrices at step 0 of DA with $\alpha = 0.1$, $\beta = 0.6$ (a, b, c) vs DA (image duplication) with $\alpha = 0.9$, $\beta = 0.1$ (d, e, f).

	Cloudy	Night	Sunny	avg
$\alpha = 0$, $\beta = 1$	69.63	53.0	62.75	61.79
$\alpha = 0.9$, $\beta = 0.9$	69.63	53.5	62.0	61.71
$\alpha = 0.9$, $\beta = 0.6$	71.02	57.19	61.2	63.14
$\alpha = 0.9$, $\beta = 0.3$	71.02	57.83	63.68	64.18
$\alpha = 0.9$, $\beta = 0.1$	71.34	**59.54**	64.29	**65.05**
$\alpha = 0.6$, $\beta = 0.9$	70.9	51.83	62.37	61.7
$\alpha = 0.6$, $\beta = 0.6$	**71.78**	54.4	62.65	62.94
$\alpha = 0.6$, $\beta = 0.3$	70.83	58.55	63.91	64.43
$\alpha = 0.6$, $\beta = 0.1$	70.33	58.5	63.77	64.2
$\alpha = 0.3$, $\beta = 0.9$	71.15	53.77	62.75	62.55
$\alpha = 0.3$, $\beta = 0.6$	69.82	51.78	64.15	61.92
$\alpha = 0.3$, $\beta = 0.3$	70.77	54.13	63.12	62.67
$\alpha = 0.3$, $\beta = 0.1$	70.14	57.24	63.91	63.76
$\alpha = 0.1$, $\beta = 0.9$	69.13	51.74	62.84	61.23
$\alpha = 0.1$, $\beta = 0.6$	70.52	55.21	62.09	62.61
$\alpha = 0.1$, $\beta = 0.3$	70.14	55.98	64.43	63.51
$\alpha = 0.1$, $\beta = 0.1$	70.14	56.34	**64.47**	63.65

Table 6.8: Results obtained for BDD100K with Domain Adaptation by duplicating the images of the source that are selected to be rotated.

Chapter 7

Conclusions

This work explored the problem of generalizing through different domains in the context of scene recognition, which is a very recurrent and challenging problem because same scenes can differ a lot according to the different domains they can belong to, as seen in Chapter 1. So while developing an algorithm that performs image classification in this context this aspect has always to be considered and handled in some way, because it's not always possible to cover all the possible domains and include them all into the training set. For this purpose, the JiGen [37] approach has been used to perform both Domain Generalization and Domain Adaptation solutions: after a brief explanation of its principles given in chapter 3, in chapter 4 we analyzed its usage in the context of semantic place categorization by using the rotations as secondary unsupervised task and by using a pre-trained neural network optimized for the context, an AlexNet pretrained on Places365 [42]. In the experiments in chapter 6 we saw that this approach allows to improve the accuracy when the model is tested on images that belong to a new and previously unknown domain, both in the Domain Generalization and in the Domain Adaptation case. The experiments were performed on subsets of the datasets COLD and BDD100K, described in chapter 5. Additional experiments that should be performed in future should consider first of all larger subsets of those two datasets, and then also other scene recognition datasets such as VPC [47] and SPED [48], to verify that this method can improve the results in every possible setting.

Bibliography

[1] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. 2016 IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778.

[2] H. Noh, S. Hong, and B. Han, “Learning deconvolution network for semantic segmentation”, in Proceedings of the IEEE international conference on com- puter vision, 2015, pp. 1520–1528.

[3] A. Toshev and C. Szegedy, “Deeppose: Human pose estimation via deep neural networks”, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 1653–1660.

[4] N. Doulamis and A. Voulodimos, “FAST-MDL: Fast Adaptive Supervised Training of multi-layered deep learning models for consistent object tracking and classification”, in Imaging Systems and Techniques (IST), 2016 IEEE International Conference on, IEEE, 2016, pp. 318–323.

[5] I. Kostavelis and A. Gasteratos, “Semantic mapping for mobile robotics tasks: A survey,” Robot. Auton. Syst., vol. 66, pp. 86–103, 2015.

[6] J.WuandJ.M.Rehg,“Centrist:Avisualdescriptorforscenecategoriza- tion,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 33, no. 8, pp. 1489– 1501, Aug. 2011.

[7] E.Fazl-ErsiandJ.K.Tsotsos,“Histogramoforienteduniformpatternsfor robust place recognition and categorization,” Int. J. Robot. Res., vol. 31, no. 4, pp. 468–483, 2012.

[8] P. Ursic, R. Mandeljc, A. Leonardis, and M. Kristan, “Part-based room categorization for household service robots,” in Proc. IEEE Int. Conf. Robot. Autom., 2016, pp. 2287–2294.

[9] Lowry, Stephanie, et al. "Visual place recognition: A survey." IEEE Transactions on Robotics 32.1 (2015): 1-19.

[10] Fornoni, Marco. “Saliency-based representations and multi-component classifiers for visual scene recognition.” (2014).

[11] S. Prasath Elango, T. Tommasi, and B. Caputo, “Transfer learning of vi- sual concepts across robots: A discriminative approach,” Idiap, Martigny, Switzerland, Tech. Rep. Idiap-RR-06-2012, 2012.

[12] S. Prasath Elango, T. Tommasi, and B. Caputo, “Transfer learning of vi- sual concepts across robots: A discriminative approach,” Idiap, Martigny, Switzerland, Tech. Rep. Idiap-RR-06-2012, 2012.

[13] S. Prasath Elango, T. Tommasi, and B. Caputo, “Transfer learning of vi- sual concepts across robots: A discriminative approach,” Idiap, Martigny, Switzerland, Tech. Rep. Idiap-RR-06-2012, 2012.

[14] K. Muandet, D. Balduzzi, and B. Schoïlkopf, “Domain generalization via invariant

feature representation," in Proc. Int. Conf. Int. Conf. Mach. Learn., 2013, pp. I-10–I-18.

[15] A. Khosla, T. Zhou, T. Malisiewicz, A. A. Efros, and A. Torralba, "Undo- ing the damage of dataset bias," in Proc. Eur. Conf. Comput. Vis., 2012, pp. 158–171.

[16] A. Karpathy. (2018). Convolutional Neural Networks (CNN / ConvNets), [Online]. Available: http://cs231n.github.io/ (visited on 11/09/2019).

[17] Michael Nielsen. (2019). Neural Networks and Deep Learning, [Online]. Available: http://neuralnetworksanddeeplearning.com/ (visited on 11/09/2019).

[18] Jordi Torres. (2018). Learning process of a neural network, [Online]. Available: https://towardsdatascience.com/how-do-artificial-neural-networks-learn-773e46399fc7/ (visited on 11/10/2019).

[19] Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, MIT Press, 2016

[20] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." Cognitive modeling 5.3 (1988): 1.

[21] Hinton, Geoffrey E., et al. "Improving neural networks by preventing co-adaptation of feature detectors." arXiv preprint arXiv:1207.0580 (2012).

[22] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." The journal of machine learning research 15.1 (2014): 1929-1958.

[23] Baldi, Pierre, and Peter J. Sadowski. "Understanding dropout." Advances in neural information processing systems. 2013.

[24] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[25] LeCun, Yann, et al. "Backpropagation applied to handwritten zip code recognition." Neural computation 1.4 (1989): 541-551.

[26] Williams, Travis & Li, Robert. (2018). An Ensemble of Convolutional Neural Networks Using Wavelets for Image Classification. Journal of Software Engineering and Applications. 11. 69-88. 10.4236/jsea.2018.112004

[27] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).

[28] LeCun, Yann A., et al. "Efficient backprop." Neural networks: Tricks of the trade. Springer, Berlin, Heidelberg, 2012. 9-48.

[29] Canziani, Alfredo, Adam Paszke, and Eugenio Culurciello. "An analysis of deep neural network models for practical applications." arXiv preprint arXiv:1605.07678 (2016).

[30] Deng, Jia, et al. "Imagenet: A large-scale hierarchical image database." 2009 IEEE conference on computer vision and pattern recognition. Ieee, 2009.

[31] Marcelino Pedro. (2018). Transfer learning from pre-trained models, [Online]. Available: https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751 (visited on 11/14/2019).

[32] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[33] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[34] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015. APA

[35] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
[36] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult", IEEE transactions on neural networks, vol. 5, no. 2, pp. 157–166, 1994.
[37] Carlucci, Fabio M., et al. "Domain Generalization by Solving Jigsaw Puzzles." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.
[38] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In Euro- pean Conference on Computer Vision (ECCV), 2016.
[39] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M. Hospedales. Deeper, broader and artier domain general- ization. In International Conference on Computer Vision (ICCV), 2017.
[40] Antonio Torralba and Alexei A. Efros. Unbiased look at dataset bias. In Conference on Computer Vision and Pattern Recognition (CVPR), 2011.
[41] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In Conference on Com- puter Vision and Pattern Recognition (CVPR), 2017.
[42] Zhou, Bolei, et al. "Places: An image database for deep scene understanding." arXiv preprint arXiv:1610.02055 (2016).
[43] Jia, Yangqing, et al. "Caffe: Convolutional architecture for fast feature embedding." Proceedings of the 22nd ACM international conference on Multimedia. ACM, 2014.
[44] Pronobis, Andrzej, and Barbara Caputo. "COLD: The CoSy localization database." The International Journal of Robotics Research 28.5 (2009): 588-594. APA
[45] Yu, Fisher, et al. "Bdd100k: A diverse driving video database with scalable annotation tooling." arXiv preprint arXiv:1805.04687 (2018).
[46] Mancini, Massimiliano, et al. "Robust place categorization with deep domain generalization." IEEE Robotics and Automation Letters 3.3 (2018): 2093-2100.
[47] J. Wu, H. I. Christensen, and J. M. Rehg, "Visual place categorization: Problem, dataset, and algorithm," in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2009, pp. 4763–4770.
[48] Z. Chen et al., "Deep learning features at scale for visual place recogni- tion," in Proc. IEEE Int. Conf. Robot. Autom., 2017, pp. 3223–3230.