UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

BRIDGING THE GAP BETWEEN SIMULATION AND REALITY USING GENERATIVE NEURAL NETWORKS

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

NICOLÁS RICARDO CRUZ BRUNET

PROFESOR GUÍA:
JAVIER RUIZ DEL SOLAR SAN MARTÍN

MIEMBROS DE LA COMISIÓN:
FELIPE TOBAR HENRÍQUEZ
DANIELE NARDI
DORIS SÁEZ HUEICHAPAN

Este trabajo ha sido parcialmente financiado por FONDECYT 1161500 y FONDECYT 1201170

SANTIAGO DE CHILE
2021

RESUMEN DE LA TESIS PARA OPTAR
AL TÍTULO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA
POR: NICOLÁS RICARDO CRUZ BRUNET
FECHA: 2021
PROF. GUÍA: JAVIER RUIZ DEL SOLAR SAN MARTÍN

BRIDGING THE GAP BETWEEN SIMULATION AND REALITY USING GENERATIVE NEURAL NETWORKS

Los simuladores son herramientas fundamentales para el desarrollo de algoritmos. Sin embargo, la mayoría de los simuladores no son representaciones certeras del mundo real lo que implica que algoritmos entrenados únicamente en simulación tienden a tener bajo rendimiento al ser usados en la realidad. El objetivo de esta tesis es resolver el problema de transformación de imagen-a-imagen con particular enfoque en la transformación de imágenes de simulación-a-realidad para disminuir la brecha que existe entre ambos. Con este fin se proponen dos metodologías basadas en modelos generativos. Estos modelos consisten en redes neuronales, los cuales son usados para reducir la brecha entre simulación y realidad de SimRobot, el simulador más popular de la Standard Platform League de la Robocup. Los resultados demuestran que ambos métodos son capaces de generar ambientes realistas los cuales pueden ser usados para entrenar y evaluar detectores basados en redes convolucionales. FeatureGan también es evaluado en la tarea más difícil de transformar imágenes de videojuegos-a-realidad. Una red de segmentación semántica es entrenada de forma exitosa usando imágenes creadas por el modelo generativo y entrega buenos resultados al ser evaluada en una base de datos de imágenes reales.

ABSTRACT OF THE THESIS TO BE ELIGIBLE FOR
THE TITLE OF MASTER OF ENGINEERING SCIENCES IN ELECTRICAL ENGINEERING
BY: NICOLÁS RICARDO CRUZ BRUNET
DATE: 2021
SUPERVISOR: JAVIER RUIZ DEL SOLAR SAN MARTÍN

BRIDGING THE GAP BETWEEN SIMULATION AND REALITY USING GENERATIVE NEURAL NETWORKS

Simulators are fundamental tools to develop algorithms. However, most simulators are not accurate representations of reality and algorithms developed exclusively in simulation tend to under-perform when deployed into real environments. The objective of this thesis is to solve the image-to-image translation task, with a particular focus on simulation-to-reality image translation and the ultimate goal of reducing the gap that separates simulation from reality. To this end, two methodologies based on generative models are proposed. Both approaches use neural network models to reduce the simulation-to-reality gap of SimRobot, the most popular simulation environment of the Robocup Standard Platform League. The results show that both methodologies are able to produce very realistic environments that can be reliably used to train detectors based on convolutional neural networks. Additionally, FeatureGan is also tested in a more challenging problem: the video game to reality image translation task. A semantic segmentation network is successfully trained using the realistic images provided by the generator and achieves good performance on a test dataset composed of samples collected from reality.

A toda mi familia.

Agradecimientos

Agradezco a toda la gente que hizo posible este trabajo. Primero a mi familia por apoyarme en todas las cosas que quise hacer y en particular a mis padres por darme todas las oportunidades.

A mi hermana por constantemente alegrarme el día con su particular sentido del humor.

A mis amigos del colegio con los que sigo teniendo contacto hasta el día de hoy, 9 años después. Vicho, Medalla, Tapioca, Arturo, Rai, Gata, Tamaro, Monroe, Kriss.

A toda la gente que conocí en la universidad y a través de la carrera. A los amigos que hice, la gente con la que trabajé, compañeros y profesores que hicieron de la universidad un lugar interesante y valioso. En particular, al Daniel, la Sole que me obliga a hacer deportes violentos, la Cata, Roberto, Nico (el otro Nico) y la Mariana.

A la gente del laboratorio de robótica: José Miguel por aceptarme en el equipo, Pablo Cano y Kenzo Lobos por intentar enseñarme a programar con dudosos resultados, Francisco Leiva por tratar de enseñarme a mantener un formato con aún más dudosos resultados y a toda la gente con la que compartí las competencias Robocup.

Al profesor Javier Ruiz-del-Solar por ser un excelente guía durante la carrera, por mantener la robótica viva en el departamento y por todo el apoyo durante el magister.

También a Doris Saéz por la confianza y por las oportunidades que me dio.

Este trabajo fue parcialmente financiado por FONDECYT 1161500 y FONDECYT 1201170 y el programa de GPUs de NVIDIA.

Table of Contents

Table Index

Illustrations Index

Chapter 1

Introduction

1.1 Motivation

1.1.1 General overview

Simulators are fundamental tools to develop robotic applications since they provide an environment to create and test algorithms without having to use a real robot. However, developing solutions in simulation is problematic since there is a mismatch between the samples generated in simulation and those collected in a real environment. This mismatch is commonly referred to as the simulation-reality gap and usually results in the sub-optimal operation of algorithms developed in simulation when deployed in real-world conditions. Furthermore, if an algorithm is properly tuned to work in a real environment, for example by training it with real data, then its behaviour cannot be accurately evaluated in a simulator because the performance of the algorithm could be vastly different in a simulated environment compared to a real environment, given the almost noise-free nature of simulators.

Recently, several methodologies have been proposed to bridge the gap between reality and simulation [33][3], most notably in the image processing area. Among these approaches, the ones based on generative models [8][28] usually require large training sets of aligned real-simulated samples. Given an image A in the Dom_A domain and another image B in the Dom_B domain, the pair of images (A, B) is defined as aligned if the geometric disposition of the objects in both images is similar, meaning that both images share a common scene layout. Achieving large datasets of aligned images is very time consuming and requires a lot of manual labeling.

In this thesis two methodologies are proposed to perform image-to-image translation such that the generated realistic image $\hat{A}$ is aligned with the input image B. The generator can then be used for the real-time generation of realistic images in simulation environments

The first of these methodologies consist on a supervised approach. The main challenge of a supervised methodology is training the generative neural network in charge of outputting realistic images. This requires a massive amount of training data, i.e. pairs of aligned

(simulated, real) images, which are difficult and time consuming to obtain. To address this issue, the semantic segmentation of the images is used as an intermediate domain between simulation and reality. More specifically, real images are segmented, and then the generator is trained to transform segmented images into realistic ones. Once the generator is trained, rendered simulated samples are first segmented and then fed to the generator, which produces aligned realistic images.

In the here-proposed approach, the training image pairs composed of (segmented, real) image pairs are obtained using the following procedure: First, a large amount of real images of the target environment is acquired. Then, the relevant objects in each real image are extracted using an instance segmentation neural network (e.g. a Mask R-CNN in this case). Using these objects a large number of segmented images can be constructed which are of similar nature than the segmented images generated in simulation. The generator is trained using this database

However, training the instance segmentation network also requires a large amount of training data of the target domain. In this work, this training process is addressed by using a small number of labeled data together with transfer learning, active learning mechanisms and domain randomization, thus avoiding the use of a manual labeling.

While this methodology allows to generate realistic images from simulated images, it still requires some manual labeling and the approach is complex to implement and replicate, particularly for complex domains, since it involves several steps which require different neural networks.

The second of the proposed methodologies addresses this issue by using an end-to-end training method called FeatureGan which was developed for this thesis. FeatureGan has the advantage that it can be trained using unaligned images in an unsupervised manner, and achieves consistently good results in the simulation-to-reality image translation task. During training, FeatureGan uses a feature-Loss function to ensure that the generator, which is based on a U-net or Resnet [29] architecture [50], produces a realistic output image $\hat{A}$ that is aligned to the input simulated image B. A classic GAN loss function is also used to steer the generator into producing images that are part of the target domain Dom_A. To further improve the quality of the generated images, a modified class based feature-Loss function is developed, which takes into account the ground truth information provided by the simulator and a feature pyramid discriminator is used.

As a proof of concept, both methodologies are tested in the soccer robotics domain, specifically in the RoboCup Standard Platform League (SPL). In the SPL, a large part of the development process of the required vision, self-localization and strategy modules are done in simulation, since the robots are fragile and therefore are used only when strictly necessary. The simulators used in the SPL (e.g., SimRobot [31]) suffer from the simulation-to-reality gap. This thesis shows that by using the proposed methodologies, the simulation-reality gap is significantly reduced, allowing to generate a realistic simulated environment. This allows to train and test vision algorithms directly in simulation. Both FeatureGan as well as the supervised approach are compared to the standard CycleGan implementation [71] as a benchmark. Then, to prove the generality of FeatureGan, this approach is also tested in the video game domain to drastically improve the visual fidelity of these rendered environments.

Finally, FeatureGan is also tested in the real-to-real image translation task. All the figures presented in this work are original and produced by the author unless stated otherwise.

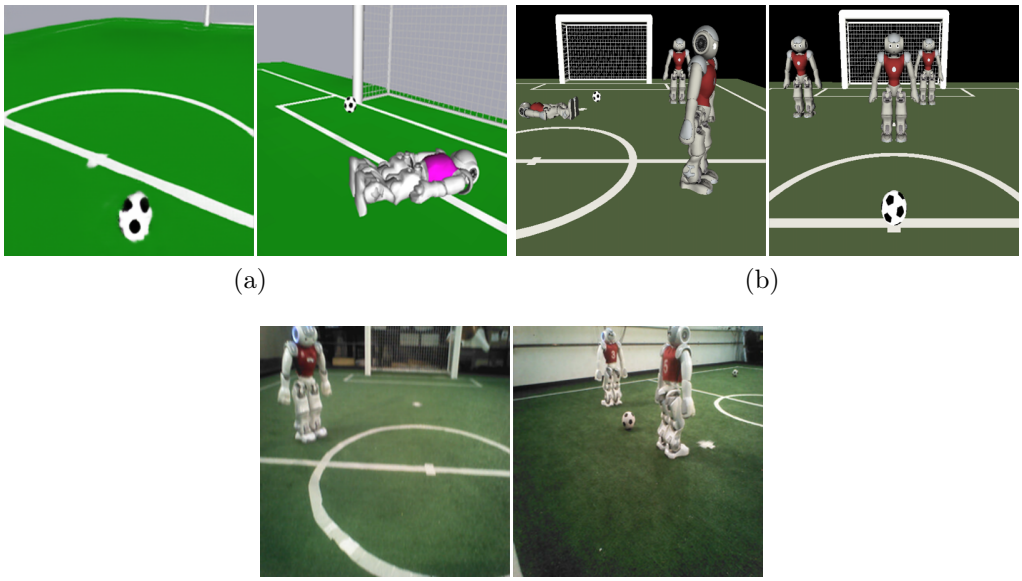
1.1.2 Robotic soccer and the Robocup

The Robocup is an annual event where teams of different backgrounds compete against each other in a series of robotic challenges with the overall objective of developing and improving the software and hardware of robots. One of the most popular competitions on the Robocup is the robotic soccer challenge, in which teams participate in a game of soccer played autonomously by robots. By iteratively improving the quality of play, it is expected that "By the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup" [48]. This is a daunting challenge given the extreme complexity of soccer. To properly play the game the robot needs to correctly sense its environment, detect relevant objects in the field, self-locate, correctly navigate the field, take decisions in a dynamic environment and coordinate with teammates. In this thesis we mainly focus on the SPL league where two teams of five Nao robots compete in a soccer game without any external processing or sensing, meaning that all relevant information and processing must be done in the robot itself in real-time.

1.1.3 SimRobot simulator

To play a soccer game, several algorithms need to be developed and tested. While the process can be performed on a real robot, simulation offers several advantages: First, constantly using the real robot damages the hardware. This damage usually comes in the shape of degradation in the joints, gears and motors which are difficult and expensive to repair or replace. Second, collecting statistics and designing tests is an easier process in simulation than in reality since the environment can be changed according to the needs of the user and ground truth information is available. Third, better debugging tools are usually available in simulation, which makes the development process far easier. Programs such as IDEs can also be used to further streamline and facilitate the development process. Finally, simulators offer an environment to quickly and reliably generate databases that can be used to fine-tune parameters or train machine learning algorithms. Given such advantages simulators are the defacto development tool for robotic applications. In the SPL league, several simulators are currently in use by different teams, the most popular one by far being the Simrobot simulator [31] developed by Bhuman. It is described as a single user, single workstation simulator that is able to simulate a 3D environment. This simulator makes use of standard dynamics and visual libraries such as the rigid body dynamics (ODE) and the OpenGl library. While SimRobot is useful for prototyping algorithms, its simplistic rendering means that the final stages of algorithm development must be done in a real environment since the visual simulation does not accurately represent reality. This is apparent given the quality of the simulated images that are projected to the simulated robot's camera, examples of which can be seen in Fig. 1.1 a), b). These are in stark contrast with the images collected from a real environment shown in Fig. 1.1 c). Given the mismatch between these samples and samples collected in a real environment, algorithms which were completely developed in simulation tend to under-perform when deployed in reality. To tackle this problem, teams

usually prototype in simulation and then fine-tune the algorithms in a real environment. This finetuning process is very time consuming for the teams and requires extensive use of the robot which translates in damaged hardware. Additionally, certain teams do not have the required number of robots (five in each team) to test a game in real conditions which means that certain parameters will not be accurate even when tuned in a real environment. Ideally, all the development would be done in simulation, with only minor testing on reality. However this is not possible with current simulators.



(a) (b)

(c)

Figure 1.1: a) images from SimRobot 2016, b) images from SimRobot 2018.

1.1.4 Video Game simulation

Video-games are currently one of the biggest entertainment industries on the planet. Much of the development of traditional computational vision algorithms and custom hardware has been spearheaded by the video-game community. This includes fundamental milestones such as the creation of the first graphic processing units and frameworks such as Unreal engine which is widely used by the robotics community and has recently found its way into movie productions.

In the last 10 years video-games have seen a staggering improvement in the quality of rendering. This is mainly due to the advancements in lighting techniques as well as better models and textures. However achieving a visual presentation which is close to reality in video-games is still not possible given the hardware constraints and the necessity of running in real time. The difference between the rendered images in video games and samples collected from similar environments in the real world is a form of simulation-to-reality gap. Solving this gap would translate in more immersive experiences and would also provide excellent environments to develop applications such as autonomous driving software.

1.2 General objectives

The main objective of this thesis is to propose a methodology to decrease the simulation-to-reality gap in the robotic soccer environment using generative neural networks to transform simulated images into realistic images. When training and testing machine learning models with images generated by using the proposed methodologies, the performance of the model should be identical or similar to the performance achieved when testing and training the same model with samples collected from a real environment.

1.3 Specific objectives

- Achieve real time simulator operation.
- Avoid using the robot to limit hardware damage.
- Achieve alignment between the original rendered simulated images and the generated images provided by the proposed methodology.
- Avoid common generator problems such as mode collapse and artifacts.
- Reduce manual labeling to a minimum.
- The methodology must trainable and deployable on consumer-grade hardware.

1.4 Hypothesis

The main hypothesis is that it is possible to generate realistic images from rendered images in real time by using a generative network. It is also hypothesised that by using the generated realistic images, object recognition systems can be trained and will achieve similar quality as those trained using real samples.

1.5 Main Contributions

The main contributions of this work are two different methodologies to generate realistic looking simulated environments. The first method uses a reliable supervised training process which uses a generative cascade refinement neural network model to generate realistic images with high consistency and little to no artifacts. However, it requires labeled image pairs to be trained and achieves low texture resolution. The second method uses a generative neural network to produce very realistic images from unlabeled, unpaired data. The generative model is trained using a combination of a feature-Loss and a GAN loss. The training process is unsupervised meaning that it might become unstable and artifacts may appear in certain circumstances. Once implemented, both methods can contribute greatly to streamline and expedite the prototyping and testing of algorithms. Furthermore, both methodologies contribute to offload most of the development from the robot to the simulator, which can be invaluable in certain circumstances.

Chapter 2

Related Work

In the last decade, robots have seen a major increase in their capabilities. In particular vision algorithms have become increasingly reliable since the introduction of machine learning based detectors and classifiers. The shift from heuristic methodologies to machine learning approaches has been ubiquitous in the robotics community, and the standard platform league of the Robocup is no exception. While some previous attempts to making CNN based detectors were made, these did not operate in real time. In 2017 the UChile robotics team introduced the first CNN based detector that was used in a real game (see appendix 2). The approach is based on an heuristic region proposal followed by a classifier which estimates the class of the extracted region. The detector was able to operate in real time on a Nao V5 robot. The performance of the detectors was further improved in [34] by substituting the RGB images by grayscale images which greatly increases the speed of the detectors with only minor penalties to accuracy (see appendix 1). Since then several other teams also transitioned to using machine learning approaches, amongst them B-Human [49], the team with most wins in the league.

While detectors that use CNN classifiers achieve far better results than heuristic approaches in challenging conditions, training the models is very time consuming since databases need to be created in accordance to the problem requirements. Simulators are a fundamental tool in the robot development pipeline which theoretically offer a cheap solution to this problem as they provide reliable virtual environments to develop algorithms. Recently, simulators have been used to train and test machine learning algorithms [39][33][3][69][51][55][65], which are then deployed to operate in real world conditions. One of the main challenges of this approach is the simulation-reality gap: given the mismatch between samples obtained from simulation and samples obtained in real conditions, algorithms which were developed in simulation tend to under perform in real environments. Furthermore, this also means that simulation is not a representative testing environment for algorithms developed using real data. Given this, performance metrics calculated in simulated environments tend to be misleading.

Several approaches have been proposed to reduce the gap between the development environment (simulation) and the operational environment (reality). Such techniques usually involve domain transfer in order to generate realistic samples in simulation [5]. When the

transformation involves changing the domain of images it is commonly referred as the image-to-image translation task. This problem has received particular attention by the computational vision community with a wide range of proposed solutions. In [33], a collision avoidance system for indoor service robots based on multimodal DRL was proposed. In this case, the reality-gap is reduced by corrupting and post-processing the simulated depth images so they resemble their real-world counterparts. In addition, both real and simulated images are decimated (and subjected to an anti-aliasing filter) to reduce their dimensionality to 60x80 pixels. The collision avoidance system was trained in simulation and then successfully transferred to reality. Other approaches involve generative models. Isola et al. [28] proposed an unsupervised architecture based on GAN networks [16] that is able to translate an image from an input domain to a target domain. Pix2pix uses an encoder-decoder structure and is trained using a combination of a GAN loss and an L1-norm loss. Chen and Koltun [8] reported that a supervised approach based on a feature-Loss was able to achieve state of the art results in the image-to-image translation task. This work uses a custom generator architecture called a cascade refinement network (CRN) which works by generating increasingly higher resolution features to produce spatially consistent output images. The input to CRN is a semantic map where each class is represented by one channel in a one-hot encoded vector. One of the main advantages of using a supervised approach is its very stable training process and its high resilience to artifacts. Recently, image-to-image translation has also become mainstream in video-games to improve visual fidelity, in the form of deep learning super-sampling (DLSS) [12]. DLSS is a neural network generative model which produces an image which appears to be of high resolution from a low resolution image. The model is trained using aligned samples of low resolution and high resolution images from the same game. Then, at inference time the model is used to improve the quality of the game with minimal cost in performance. To further improve the speed of this methodology, Nvidia introduced machine learning oriented processing units called Tensor cores to consumer GPU's.

While domain transfer using CNN models offer state of the art results aligned image pairs of $(input, target)$ samples needs to be available for training purposes. An alternative approach is projecting both simulated and real samples to an intermediate domain[3][25]. This is particularly useful when there are no aligned image pairs $(input, target)$ but aligned pairs to an intermediate domain Dom_{inter}, $(input, inter)$ and $(target, inter)$ are available. This is known as the zero-pair image-to-image translation problem. In [39], DRL based visual navigation for humanoid, biped robots in robotic soccer is proposed. Real and simulated images were segmented and then down-sampled in order to make them look similar, thus reducing the simulation-reality gap. The system was trained using the down-sampled, segmented images generated in the simulator, and then directly deployed in reality, where the robot is able to navigate between static and moving obstacles (robots).

These approaches work excellently when aligned samples from the input to the target domains are available, or when aligned samples to an intermediate domain are accessible. However, if this is not the case, training a network to generate aligned samples using a dataset of unaligned samples is still possible. Zhu et al. [71] proposed CycleGan which uses a cycle consistency loss to preserve the layout of the input image across the different domains in the cycle while a GAN-Loss is used to perform the domain transformation. Liu et al. [38] proposed an unsupervised image to image framework based on GANs and the assumption of a shared latent space. This network was later used in [4] to perform image-to-

image translation from an input image corresponding to the SPL soccer field rendered using unreal engine to realistic images. However samples generated using this approach failed to significantly improve the results over basic data augmentation over the rendered image.

A third approach is domain randomization [58]. "Domain randomization involves randomizing non-essential aspects of the training distribution in order to better generalize to a difficult-to-model test distribution" [64]. The main goal of this approach is that the real data distribution will be within the distribution of the training data. One of the main of domain randomization is to select which parameters to randomize and to what degree [3]. Domain randomization is a promising research direction where different robotic problems are being addressed [64][52][63][45]. However, a simulator based in domain randomization is not suitable to accurately represent how the robot will perform in real environments since it is not a good approximation of reality. This means that the applications of domain randomization are constrained to training machine learning models in simulation and other parts of development must still be done in a real world environment.

Chapter 3

Technical background

3.1 Simulators

Simulators are environments which mimic a certain system. This work tackles virtual simulators, which are computer programs that imitate the features of real-world environments. This kind of simulators are ubiquitously used in robotics, autonomous driving as well as in other areas of engineering for developing and testing algorithms. One of the advantages of developing in simulators rather than in a real environment is that simulators offer a controlled environment which is completely deterministic. Among other things that means that:

Observation 1 The location of each object in the scene can be known with certainty at all times.

Observation 1 is a fundamental property of simulators that is difficult to achieve using probabilistic models such as neural networks. Observation 1 also means that experimental repeatability is easily achievable given that all the relevant parameters of a given configuration are recorded.

Additionally, developing in simulation offers some major advantages in the form of a risk-free environment for the operators and the autonomous system, meaning that there is no wear of the hardware. This is important as this means that the simulated environment can be used to obtain massive amounts of data without any cost and in a fraction of the time.

3.2 Neural network basics

3.2.1 Convolutional Neural Networks

A convolutional neural network is a particular type of neural network commonly used for image processing. The input to the network is sequentially convolved with several kernels to extract increasingly semantically rich features. The kernels, commonly referred as filters are

learned using an optimization process such as backpropagation. One of the main advantages of convolutional neural networks over fully connected networks are their decreased number of tunable parameters and faster inference speeds which allows to design deeper and more powerful networks.

3.2.2 Supervised Training

Supervised learning is the most common way of training neural networks. This method requires an input x and a corresponding label y. Then, the network is tasked with learning a model f such that $f(x) = y$. The function f can be found by making a prediction $f(x) = \hat{y}$ and comparing it with the real label y. The resulting error is then used by the learning algorithm to update the model. This optimization process allows to minimize the error and consequently the model becomes better at performing predictions.

3.2.3 Unsupervided Training

Unsupervised training is the process of training a neural network using only unlabeled data. While in supervised learning 3.2.2, both the input x and the corresponding label y were available, in the case of supervised learning only the input data x is available. Given this, the model must be optimized without using any labels

3.2.4 Common vision problems

In computer vision, one of the main focus has been the identification of objects in an image. Several types of algorithms allow to solve this problem, however this thesis mainly focus on convolutional neural networks (CNNs)

Classification

The classification problem involves determining the presence or absence of one or more classes of objects in an image. If the case of one object, this is a binary classification problem where the CNN returns a probability (if softmax is used) between zero and one which indicates its confidence that the relevant object is present in the image. For several object a vector is used rather than a single number where each element j of the vector corresponds to the probability that the object of class c_j is present in the image. Current algorithms based on very deep CNNs achieve close to human performance.

Detection

Detection is the task of determining the position and class of an object in an image. This is achieved by assigning to each relevant object a bounding box and a class label. Note that different objects of the same class will correspond to different detections, meaning that detection not only determines class but also differentiates between instances of the same object class.

Semantic segmentation

Semantic segmentation corresponds to assigning a class label to each pixel of the image. This provides relevant information on the shape of the objects in the scene. All the objects in the image of the same class share the same label.

Instance segmentation

Instance segmentation corresponds to a mix between detection and semantic segmentation. Instance segmentation gives information about the class, the shape and the position of each individual object in the image by assigning an unique label to all of its pixels.

3.2.5 Receptive field

The receptive field corresponds to the region over an input tensor to which a neuron has access. This region is defined by a size and its center. The receptive field can be increased linearly by stacking more convolutions into the network, therefore making it deeper as explained by Szegedy et al. [60], or multiplicatively by using down-sampling methods such as strided convolution or some sort of pooling. Receptive fields are a fundamental consideration in the design of neural networks since they determine how spatially spread are the features extracted by the neural network. Neural networks with large receptive fields will have global knowledge of the input image and therefore will be able to find complex spatial relations. On the other hand, smaller receptive field gives the network access to only local spatial information which can be useful if there is a need to analyze details on the image such as specific textures or small geometry. It is important to note that not all pixels in the receptive field of a neuron are equally important. As described by Luo et al. [40], typically the relevance of pixels in the receptive fields follow a Gaussian distribution with the maximum located at the center of the receptive field. Given this, pixels at the borders of a receptive field have almost no influence to the output since the Gaussian distribution decays exponentially. This also means that the effective receptive field is much lower than the real receptive field.

3.2.6 Pooling layers

Pooling layers are commonly found in CNNs as they provide two key features to the network. First, pooling operations offer invariance to small translations in the input tensor. This means that if a relevant object in the input is shifted by a small number of pixels, the output of the pooling layer would not change.

Pooling is also used to decrease the spatial size of the features in the network, effectively compressing the information which in turns drastically reduces the computational cost of subsequent layers in the network. The decreased spatial size also means neurons in the subsequent layers will have larger receptive fields and as such the extracted features will be more spatially spread.

Two of the most popular pooling layers are max-pool and avg-pool. Max-pool outputs the maximum value over a region. By using max-pool over an input tensor the most salient features of each region are preserved. In terms of signal processing this is the equivalent of

a high pass filter, meaning that max-pool preserves the high frequency (details) features of the input.

Avg-pool on the other hand averages the values over a defined region. By using avg-pool over an input tensor, the high frequency features over a region are lost, meaning that in terms of signal processing it acts as a low pass filter where the general structure of the input is preserved while the details are lost.

3.2.7 Spatial information

As networks become deeper, the number of feature channels is usually increased in order to extract more features. As the number of feature channels grows, performing convolutional operations over these tensors can become prohibitively expensive in terms of computational cost. To avoid this problem sub-sampling is commonly applied. This can be done either by applying operations such as pooling or by applying strided convolutions, which in turn increases the receptive filed as explained in Sec. 3.2.5. However, sub-sampling results in a loss of information, which translates to decreased spatial accuracy most notably at the deepest layers of the network. This has a twofold effect. First is that the spatial relations from the input image are mostly lost. These relations are important since the relative position of features contribute a great amount of information on the nature of the objects in the input image. For example, the relative position of the eyes, the nose and the mouth are a fundamental aspect in recognizing a human face in an image. By losing the relation between these features any image with random positioned eyes, mouth and nose would be classified as a face. The spatial relations between features in the image can be preserved to a limited extent by using overlapping windows as proposed by Hinton et al. [20]. On the other hand, the loss of spatial information translates in a loss of detail (high frequency information) in the extracted features. This mean that small objects and small features in the input image are lost in the deeper layers of the network and therefore very deep neural networks perform poorly when faced with task that require processing small features. This is a problem since increasing the depth of the network is usually associated with improved results given the higher level of abstraction of the features that the deeper layers are able to extract. There is then a trade-off between the level of abstraction that the network can achieve and the capacity of the network to correctly maintain the spatial information of the input image throughout the network.

3.2.8 Generalization

An important feature of neural networks it's their ability to generalize to unseen samples of the same domain. This means that from a training dataset the network is able to learn relevant feature extractors that can be used to accurately predict the labels on a test dataset. The generalization error is defined as the difference in the performance metrics between the training and validation/test datasets. Zhang et al. [68] presented several insights to the problem of generalization. First, deep neural networks tend to generalize even if they generally have enough parameters to completely memorize the training dataset. Second, the deep neural networks tend to generalize when they can and fit by brute force when they must. Finally, regularization can improve the generalization capacities of the network it is

not essential since the networks can achieve similar performance without any regularization techniques.

3.2.9 Normalization

Normalization is a fundamental tool to train neural networks since it provides a more stable training process. It is not completely understood why normalization is so beneficial and several theories have been put forward. One of the most commonly accepted ideas was put forward by Ioffe and Szegedy [26] and states that normalization reduces the internal covariate shift. Covariate shift refers to the change in the activations that results from modifying the weights of the network. Indeed, since weights in a layer are continuously updated during training, this means that all subsequent layers will receive drastically different inputs throughout the training process. Normalization helps to stabilize the activations and thus make training easier. However, [54] shows in an experimental setup that there is little to no relation between the amount of internal covariate shift and the implementation of batch-normalization in a network. Indeed, networks with high covariate shift and with batch-normalization still outperform networks with low covariance shift and no batch-normalization. Instead they put forward the idea that normalization layers make the optimization landscape smoother. In particular it makes the loss function have a smaller L-Lipschitz factor as defined in Eq. 3.1:

$$|\mathrm{f}(x_1) - \mathrm{f}(x_2))| \leq \mathrm{L} \left\| x_1 - x_2 \right\|. \tag{3.1}$$

In practice a smaller L-Lipschitz factor means that the rate of change of the function is reduced. Santurkar et al. [54] also point at the gradients of the loss having a smaller L-Lipschitz factor as another reason for the increased performance of normalized over non-normalized networks. The authors also point out that these beneficial properties are not exclusive to batch-normalization but are shared by almost all normalization layers. Two normalization layers are relevant to this thesis.

Layer normalization

The first one, layer normalization [2] (see Fig. 3.1 a)), normalizes the input tensor across channels, meaning that the values of the mean μ (see Eq. 3.2) and the variance σ^2 (see Eq. 3.3) are the same for all channels in the batch. Eq. 3.4 shows the layer normalization process:

$$\mu = \frac{1}{c} \sum_{j=1}^{c} x_j, \tag{3.2}$$

$$\sigma^2 = \frac{1}{c} \sum_{j=1}^{c} \left(x_j - u_{\mathrm{i}} \right)^2, \tag{3.3}$$

$$\widehat{x_j} = \frac{x_j - u}{\sqrt{\sigma^2 + \varepsilon}} \cdot \gamma + \beta, \tag{3.4}$$

where ε is a sanity parameter and γ and β are called the element-wise affine parameters which are learnable parameters of gain and bias that are applied to each element after normalization. This means that for an input tensor of size $[c, w, h]$ there are $c \times w \times h$ values of γ and β.

Instance normalization

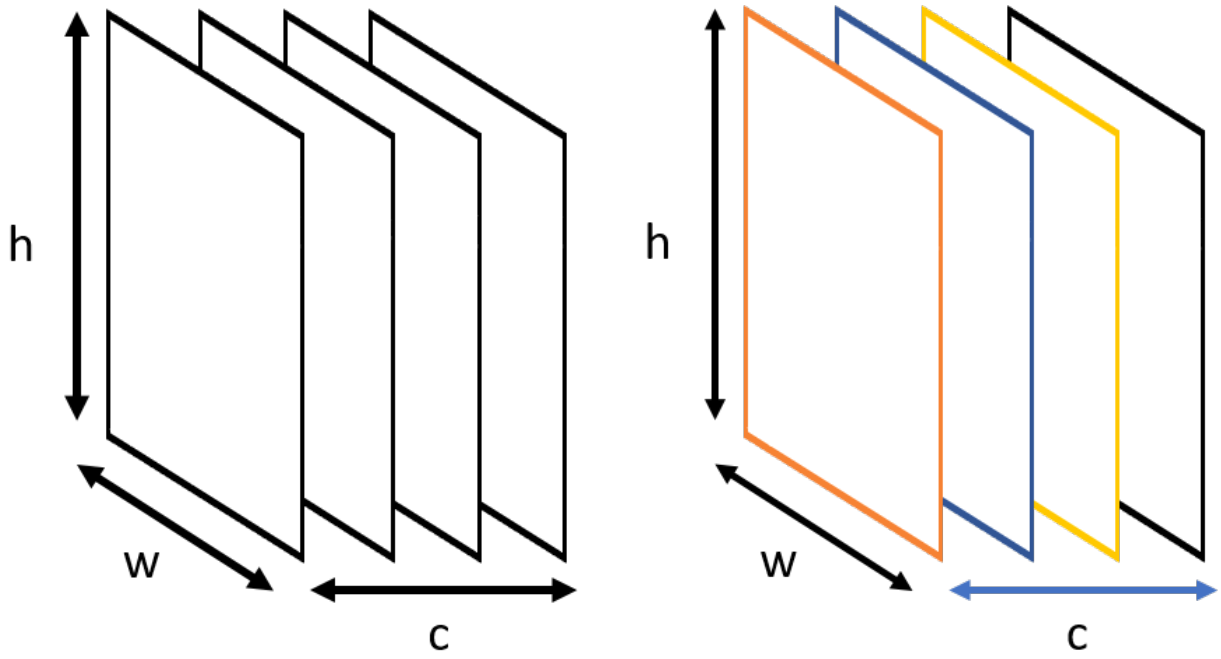
The second, called instance normalization [66] (see Fig. 3.1 b)), normalizes each channel independently, meaning that the values of mean μ_j (see Eq. 3.5) and the variance σ_j^2 (see Eq. 3.6) are different for each channel j. Eq. 3.7 shows the normalization process:

$$\mu_j = \frac{1}{w \cdot h} \sum_{l=1}^{w} \sum_{k=1}^{h} x_{j,l,k}, \tag{3.5}$$

$$\sigma_j^2 = \frac{1}{w \cdot h} \sum_{l=1}^{w} \sum_{k=1}^{h} (x_{\mathrm{i},l,k} - \mu_j), \tag{3.6}$$

$$\widehat{x_j} = \frac{x_j - u_j}{\sqrt{\sigma_j^2 + \varepsilon}} \cdot \gamma_j + \beta_j, \tag{3.7}$$

where γ_j and β_j correspond to channel-wise parameters meaning that for an input tensor of size $[c, w, h]$ there are c values of γ and β. This means that by just modifying these two parameters the statistics of each feature can be changed in unison. However, in practice the affine parameters of instance normalization are not normally used.



(a) (b)

Figure 3.1: a) layer normalization, b) instance normalization.

3.2.10 Activation functions

The convolutions found in deep neural networks are linear operations. Since a combination of linear functions results in a linear function that means that stacking convolutions together is futile since the complete system could be compressed into a single convolutional layer without any performance penalty. It also means that a neural network purely composed of convolutional layers would only be able to solve linear problems. This is solved by using an activation function between convolutions that is non-linear. Four activation functions are of interest to this thesis.

Sigmoid

The sigmoid activation function (sig) is a non-linear activation function described in Eq. 3.8:

$$\text{sig}(x) = \frac{1}{1 + \text{e}^{-x}}. \tag{3.8}$$

The sigmoid outputs values between 0 and 1. It is not normally used in neural network since tanh (described in Sec. 3.2.10) offers similar behavior but usually provides better results [32].

Hyperbolic tangent

The hyperbolic tangent (tanh) is a non-linear activation function defined in Eq. 3.9:

$$\tanh(x) = \frac{2}{1 + \mathrm{e}^{-2x}} - 1. \tag{3.9}$$

Tanh is in fact a scaled version of the classic sigmoid activation function but with better properties as explained by LeCun et al. [32], such as possessing stronger gradients and since it is centered in zero it also prevents biases in those gradients. However, tanh is normally avoided since it is known to produce a vanishing of the gradient in the network. This is caused by the bounded nature of tanh which can only output values in the range $[-1, 1]$ even for extremely large or small inputs. The result is that for large changes in the input x, the change in the output $tanh(x)$ could be very small, leading to a very small gradient.

Rectified Linear Unit

The rectified linear unit (ReLU) is an activation function which solves the vanishing gradient problem by having an unbounded response to positive inputs. The definition of ReLU is shown in Eq. 3.10:

$$\mathrm{ReLU}(x) = \max(0, x). \tag{3.10}$$

ReLU has several additional properties which contribute to state-of-the-art results in deep neural network and which have contributed to make it the most commonly used activation function in the field. First, ReLU is very cheap to compute and is ideal to achieve models that operate in real time. Most importantly, ReLU contributes to make the model sparse which is a feature that biological neural networks also share. As shown in [15], model sparsity provides several advantages such as having the capacity to represent inputs with different levels of information by activating or deactivating neurons. Model sparsity also contributes to sparse representations of data, meaning that clusters of information are easier to differentiate. This property was used to create the inception module [59] which uses filters with different kernel sizes to identify clusters of information at different spatial scales.

While having a sparse representation is beneficial, neurons that constantly output negative values will not contribute to the learning process as the output of the ReLU activation function will be a constant of value zero. Since the gradient is also equal to zero, then it is very unlikely for the neuron to be reactivated. This is commonly known as the dying ReLU problem. If a large enough number of neurons are in this state, then the network will not have the necessary learning capacity to fit the target distribution. This will lead to sub-optimal results or to a collapse in the training process altogether.

A common way to deal with this problem is to set a small learning rate but this results in increased training times or can lead to the network falling into a local minimum in the optimization landscape.

Leaky Rectified Linear Unit

The leaky rectified linear unit (leaky ReLU) solves the dying ReLU problem from a design perspective rather than from a hyper-parameter perspective by adding a negative slope for

negative values which provides a gradient in this region. The mathematical definition of ReLU is shown in Eq. 3.11:

$$\text{leaky ReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ n \cdot x & \text{otherwise} \end{cases} \tag{3.11}$$

Leaky ReLU allows to train with higher learning rates and ensures that all the neurons in the network contribute to the final result. However, sparsity in the network is lost.

3.2.11 Relevant Loss Functions

L1-Loss

The L1-Loss function, also called mean absolute error loss function is a pixel matching loss which is calculated as the average of the absolute differences between a prediction $\hat{y}$ and a ground-truth y and is described in Eq. 3.12:

$$\text{L}_1\text{-Loss}(y, \hat{y}) = \frac{1}{n} \sum_{\text{i}=1}^{n} |y_\text{i} - \hat{y}_\text{i}| \,. \tag{3.12}$$

If y and $\hat{y}$ are two images, then y_i and $\hat{y}_\text{i}$ correspond to the i-th pixel of each image.

L2-norm loss

The L2-Loss function, also called mean squared error loss (MSE) function is a pixel matching loss which consists on the average of the squared differences between a prediction $\hat{y}$ and a ground-truth y and is described in Eq. 3.13:

$$\text{L}_2\text{-Loss}(y, \hat{y}) = \frac{1}{n} \sum_{\text{i}=1}^{n} (y_\text{i} - \hat{y}_\text{i})^2 \,. \tag{3.13}$$

This results in an extra level of penalization for samples that are very different from the ground truth.

Binary cross entropy loss

The binary cross entropy loss is a function tailored for classification and described in Eq. 3.14:

$$\text{BCE-Loss}(y, \hat{y}) = -\frac{1}{n} \sum_{\text{i}=1}^{n} y_\text{i} \cdot \log(\text{p}(y_\text{i})) + (1 - y_\text{i}) \cdot \log(1 - \text{p}(y_\text{i})). \tag{3.14}$$

The binary cross entropy loss is preferred over the L1 loss and the L2 loss when the

problem is based on classes, since it heavily penalizes bad estimations, but is badly suited for regression tasks.

Feature matching loss

Feature matching loss [6], [13], is a special type of loss function that is used when matching features, rather than matching pixels, is required. Given a ground-truth image y and a predicted image $\hat{y}$, features from both images are extracted by feeding each image to a pre-trained classifier neural network and recording the resulting activation maps $Map_l(y)$ and $Map_l(\hat{y})$ from different layers of the network. The feature loss is defined in Eq. 3.15:

$$\text{feature-Loss}(y, \hat{y}) = \sum_{l=1}^{L} (\gamma_l \cdot \text{L}_1 - \text{Loss}(\text{Map}_l(y), \text{Map}_l(\hat{y})), \tag{3.15}$$

where l represents the layer, γ_l represents the assigned relevance of the features from the layer l and L is the total number of layers. Features resulting from layers closer to the input are commonly selected when a comparison of high resolution spatial information is desired while features closer to the output are the common choice when a comparison of semantically rich features is desired. Using feature matching loss instead of L1-Loss or L2-Loss offers several advantages such as lighting and color invariance since the similarity of both images is estimated based on high level semantic information rather than by pixel intensity which can greatly change even for objects of the same class. This invariance has the effects of greatly reducing oscillations of the model during training.

Identity loss

The identity loss is used as a regularizer in some neural network architectures. Given a network G and an input y, the identity loss is defined in Eq. 3.16:

$$\text{idt-Loss}(y) = \text{L}_1 - \text{Loss}(y, \text{G}(y)). \tag{3.16}$$

Basically, the identity loss encourages the network to replicate the input at the output. It also encourages the model to maintain the color of the input y at the output $G(y)$

3.2.12 Transfer Learning

Transfer learning refers to using the feature extractors that were learned for a certain domain as a starting point to learn feature extractors for a second, different domain. In their paper, Pan and Yang [42] introduced a formal definition of transfer learning.

Definition 3.2.1. (Domain) "A domain Dom consists of two components: a feature space X and a marginal probability distribution P(X)"

Definition 3.2.2. (Transfer Learning) "Given a source domain Dom_S and learning task T_S, a target domain Dom_T and learning task T_T , transfer learning aims to help improve the

learning of the target predictive function $f_T(\cdot)$ in Dom_T using the knowledge in Dom_S and T_S, where $Dom_S \neq Dom_T$, or $T_S \neq T_T$."

Transfer learning is useful since it allows to drastically reduce training times as well as reducing the necessary amount of necessary data to train a model. This is possible because feature extractors which were already learned are simple re-purposed rather than learned from scratch.

3.2.13 Tracing a model

Most machine learning frameworks, such as Pytorch [44] and TensorFlow [1] use Python as the primary interface. Python is an excellent language for fast prototyping and offers great flexibility to rapidly implement code. However, it is an interpreted language and as such lacks the level of fast performance offered by other compiled, production-oriented languages such as C++. This is a major obstacle to the integration of machine learning models into simulators such as SimRobot Laue et al. [31] which are commonly written in C++ to optimize performance.

Since its 1.0 version, Pytorch integrated a just in time (JIT) compiler that is capable of recording all the native Pytorch operations of the network at runtime and re-write them in a format that can then be imported into a C++ program using the Pytorch C++ frontend. A traced model is optimized for runtime, meaning that it performs faster inference and is more memory efficient. The process of recording and saving a model is called jit-tracing and allows to easily export models which were written and trained in a Python environment to a C++ environment.

3.3 Network architectures

3.3.1 Feature Pyramid network

One of the main challenges of detection networks is to be able to recognize objects at different scales, in particular as stated in Sec. 3.2.7 detecting small objects has proven particularly challenging for very deep CNN based detectors. This is an inherent problem of CNNs since stacking additional layers into the detector usually results in better class estimation, but increasing the depth of the network also results in a loss of spatial information which means that the estimation of the position of the object in the image is negatively affected and that small objects tend to disappear in later layers of the network. A possible solution is to analyze the input image at different scales, but this is very computationally expensive. Lin et al. [37] proposed to solve this problem by merging the information provided by feature maps at different levels of the network into a single tensor. Feature maps from layers closer to the input will have high resolution features while feature maps from deeper in the network will have semantically rich features. The Feature Pyramid Network is composed of two pathways called bottom-up, which corresponds to the classical feed-forward inference of a CNN, and top-down, which consists in the up-sampling of features from higher levels of the network which are semantically rich. Lateral connections between both pathways are used to merge tensors of the same spatial dimensions by first compressing the information of the bottom-up

pathway using a 1x1 convolution and then using an element-wise addition to combine the compressed bottom-up tensor with the corresponding top-down tensor. The resulting feature tensor can then be used as input to another neural network, such as a detector. Such is the case of mask r-cnn [18], which uses pyramidal features to dramatically improve the results.

3.3.2 VGG network

In 2014, Simonyan and Zisserman [57] presented VGGNet, a network which achieved state of the art results with a very simple structure. The main insight of this architecture is the use of small convolutions of 3×3 rather than the larger convolutions used in previous approaches. By stacking the 3×3 convolutions larger receptive fields are achieved as discussed in 3.2.5. Max-pooling is used to reduce the spatial size of the features as the information flows through the network. While other more modern networks achieve better results, VGG is still very popular given its simplicity which makes it very easy to use, modify and implement.

3.3.3 Resnet

Given inputs x_{i} with associated labels y_{i}, it is possible to train a neural network F using backpropagation, such that $F(x_{\mathrm{i}}) = y_{\mathrm{i}}$. Since neural networks are universal approximators the ability of F to correctly predict a label y_{i} given an input x_{i} should depend mostly in the number of parameters of the network. However, naively increasing the number of tunable parameters in the network would make the model's computational cost too high and since data is limited it could also lead to overfitting. Given this, increasing the depth of the network is the main choice when trying to increase accuracy. Indeed, state of the art models have continuously followed a trend of becoming deeper. However, once traditional architectures achieve a certain depth, accuracy actually starts to decrease instead of increasing. It is evident that with all other things equal, a network with a higher number of layers should be at least as good as an approximator as a shallower network. To prove this, let's define a network N composed of l layers trained on a supervised classification task. Then, any deeper network M composed of $k+l$ layers should be at least as good at the classification task as N since the first k layers of M can be defined as equal to the identity and the following l layers as equal to the l layers of N. Given this, $N(x_{\mathrm{i}}) = M(x_{\mathrm{i}})$. However, in practice shallower networks outperform their deeper counterparts when using classic architectures. This indicates that an optimization problem in the training stage is responsible for the decreased performance in deeper networks. In particular, the vanishing gradient was found to be the root cause of the decrease in performance. The problem was finally solved by He et al. [17] by introducing the residual block shown in Fig.3.2. The main idea behind its design is that instead of learning a transformation from an input x to an output y by applying a function F such that $F(x) = y$, the residual block is actually learning the residual $F(x)$ defined as $F(x) = y - x$. Then, instead of completely remapping an input to an output, the residual block is only calculating a deviation from the input. This is not only easier to learn but it is also very beneficial to propagate the gradient through the network, allowing to create much deeper neural networks with more learning capacity without the gradient degrading.

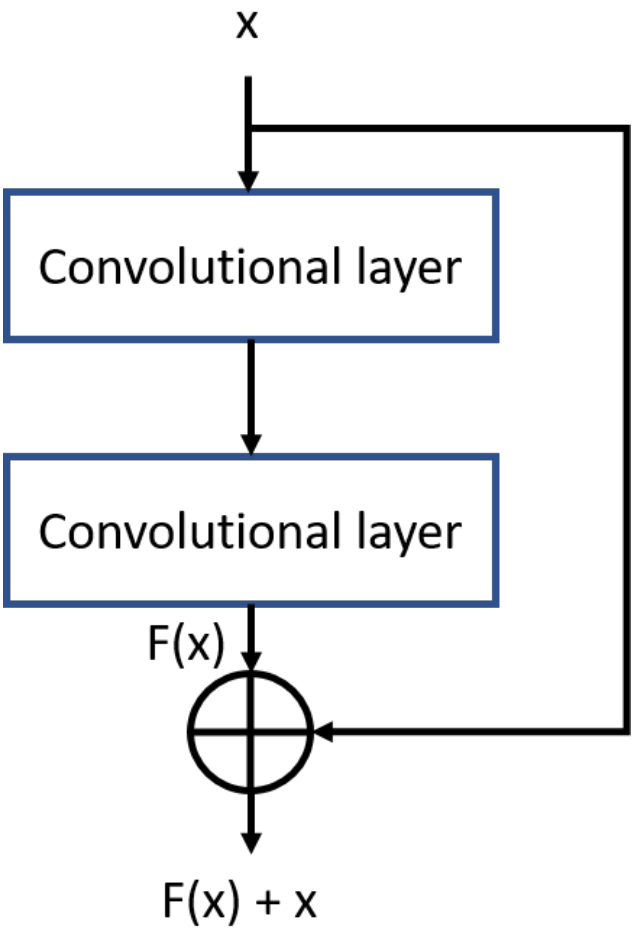


Figure 3.2: Residual block.

3.3.4 Mask R-CNN

Mask R-CNN [18] is a convolutional neural network designed to solve the instance segmentation task. The method uses a Resnet based feature pyramid network (see Sec. 3.3.1) as a backbone to extract a combination of semantically rich and spatially detailed features from an input image. Then, a region proposal network is used to scan those features at certain predefined locations and scales (called anchors) and find regions where relevant objects might be located. Once a set of regions of interest has been constructed, a second networks scans the features produced by the feature pyramid network at those locations and generates predictions for the bounding boxes, masks and classes for the objects inside the regions. By using a feature pyramid network as a backbone the network is able to accurately detect both large and small objects in an image and offer state of the art results.

3.4 Generative Neural Networks

Generative neural networks are a special type of machine learning model. Contrary to traditional discriminative models which only learn a boundary between the different classes of the problem, generative models trained on a dataset are able to infer new samples which belong to the same distribution as the one used to train it. During training, the model samples from an input distribution and given a set of parameters θ is able to generate a sample belonging to the generated distribution (see Fig. 3.3). A loss function is used to estimate how well this sample fits in the target distribution and the error is then backpropagated through the network to update the parameters θ. The model should then become increasingly better at generating new samples that closely resemble those of the target distribution, which means that after several iterations the generated distribution will match the target distribution.

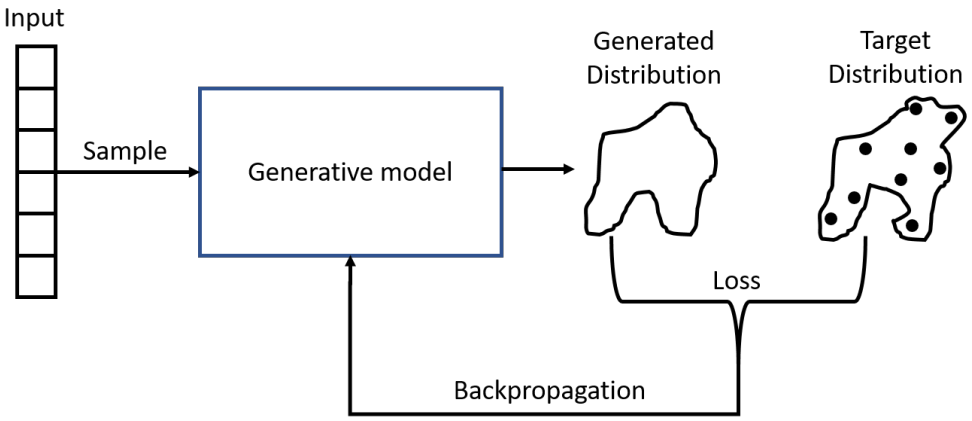


Figure 3.3: Training a generative model.

While some literature [41] puts forward the idea that the number of parameters in the generator must be smaller than the number of samples to encourage generalization, in practice even models which have a number of parameters orders of magnitude greater than the number of training samples are able to generalize well. This is in line with what was mentioned in Sec. 3.2.8. While the number of samples is not directly related to the quality of the generated distribution, the quality of the samples is, which gives way to observation 2.

Observation 2 The samples in the training dataset must accurately represent the target distribution.

Observation 2 means that not all samples are equally important which is better visualized in Fig. 3.4. Independently of the number of points, datasets with low sample variance wont accurately describe the target domain. Furthermore, generative models are prone to catastrophic forgetting [62], a phenomenon that occurs when the knowledge gained from previous samples is destroyed by trying to learn from new samples. While not easily fixable, a uniform distribution of the samples over the target distribution contributes to alleviate this problem, since it ensures that all regions of the distribution are equally weighted during training.

(a) (b) (c)

Figure 3.4: a) samples which do not accurately represent the target distribution, b) samples which accurately represent the target distribution but are not uniformly distributed, c) samples which accurately represent the target distribution.

In literature, several generative models are described, the most popular being Generative Adversarial Networks (GANs) introduced by Goodfellow et al. [16]. GANs are composed of two different agents: the discriminator and the generator. The generator, which in its traditional implementation takes a noise vector as an input has the objective of creating samples that the discriminator is unable to distinguish from the real samples of the training dataset. On the other hand, the discriminator's role is to differentiate between real samples from the training dataset and the fake samples coming from the generator network. Both networks play in a zero-sum non cooperative game described by the min-max loss function presented in Eq. 3.17:

$$\begin{aligned} \min_G \max_D \mathrm{V(D,G)} = \mathbb{E}_{x\sim p_{data}(x)} \left[\log(D(x))\right] + \\ \mathbb{E}_{z\sim z_z(z)} \left[\log(1-\mathrm{D(G}(z)))\right]. \end{aligned} \tag{3.17}$$

The loss function achieves its optimum when the generator and discriminator achieve Nash equilibrium. This happens when the generator becomes so good at generating realistic data that the discriminator is unable to differentiate between real and generated samples.

While GANs achieve state of the art result regarding the quality of the generated images, they are notably unstable and difficult to train. Common problems include the vanishing gradient problem, which happens when the discriminator becomes too good at classifying real from generated samples or when the generator becomes too good at generating images and the discriminator is unable to differentiate between both. In practice, the generator and discriminator must be in a delicate balance which is difficult to maintain. A second common problem is called mode collapse. This problem occurs when the variety of samples produced by the generator starts to collapse into a single output which is repeated for multiple inputs. This is the result of a single output being particularly likely to the discriminator and the generator learning to produce only that output. This problem is particularly difficult to solve and most of the time it's the result of a training dataset which is not representative or not

uniformly distributed.

3.5 Conditional Generative Adversarial Networks

While classic adversarial neural networks take a noise vector as an input to the generator in order to infer new samples, conditional neural networks use additional input information to allow for a much higher control of the final generated output. This additional input usually comes in the form of a one-hot encoded vector with information on the desired class of the generated output. Further control over the output can be achieved by feeding the generator with more information rich inputs such as segmented maps in order to control not only the class of the objects on a scene but also their position and shape.

3.6 Image-to-image translation

Image-to-image translation is a particular problem in which the objective is generating an image $\hat{A}$ in a target domain Dom_A from an input image B in the Dom_B domain. Some common examples of this are super-resolution [6] and style transfer [14]. Machine learning approaches have recently seen an explosion in popularity for this task with many works published on the subject. Generally, the idea consists on developing a generator G which is able to map from the distribution of Dom_B to the distribution of Dom_A such that $G(B) = \hat{A}$. Multiple methods exist to train the generator including a wide range of supervised and unsupervised approaches as well as combination of both. Most approaches require a dataset of aligned images from the Dom_A and Dom_B domains. The pair of images (A, B) aligned if the geometric disposition of the objects in both images is similar, meaning that both images share a common scene layout.

3.6.1 Generator architectures

Cascade refinement networks

Given a dataset composed of aligned pairs (A, B) of inputs samples B and targets samples A the cascade refinement neural network aims to create spatially consistent images by extracting features at different scales from the input image. The network's general architecture is presented in Fig. 3.5 and is based on analyzing an image pyramid constructed from the original input. An image pyramid consists on a set of images with different sizes, all generated by down-sampling the same original image by different factors. Starting from the smallest image in the pyramid, convolutional filters are applied in order to extract features. Since the input image is small, the convolutions can extract global features that describe the general layout of the image. The resulting feature tensor is then up-sampled by a factor of two and then concatenated with the next image in the pyramid. Convolutions are applied to the resulting tensor to generate a new set of features, which are in turn up-sampled and once again concatenated with the next image in the pyramid. This process of extraction, up-sampling and concatenation is performed by a refinement module 3.6, and a cascade refinement network has several of them. By following this process iteratively, global features extracted by the first refinement blocks are concatenated with higher resolution local features extracted by

later refinement blocks from larger, higher resolution images. In Fig. 3.5, the image pyramid is presented at the left of the image, blocks (1) and (2) correspond to refinement modules, while block 3 corresponds to the final convolution, which takes all the information extracted at different scales by previous refinement modules, and by performing a series of convolutions is able to generate a series of N images in the desired domain.

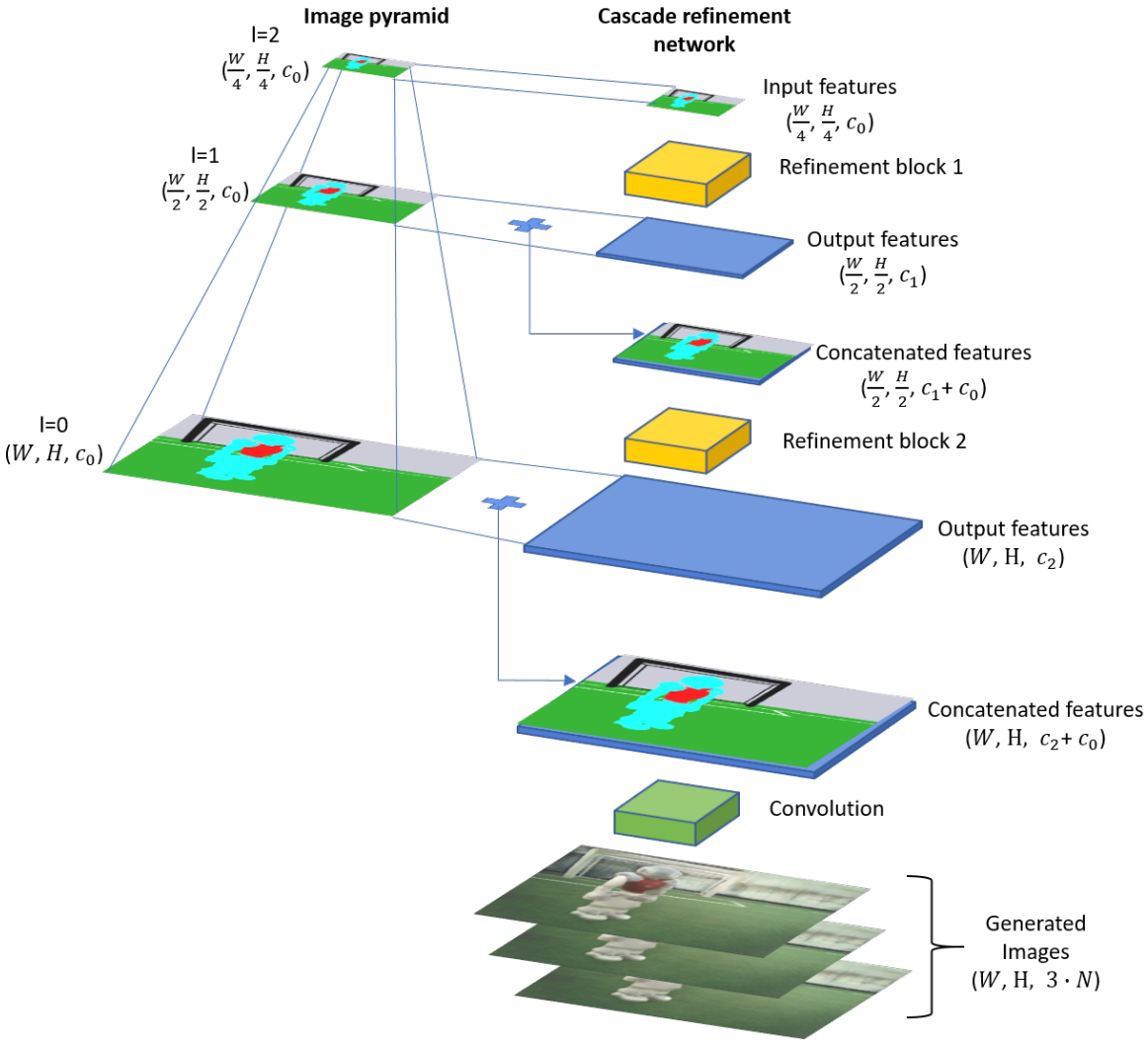


Figure 3.5: Cascade refinement network architecture.

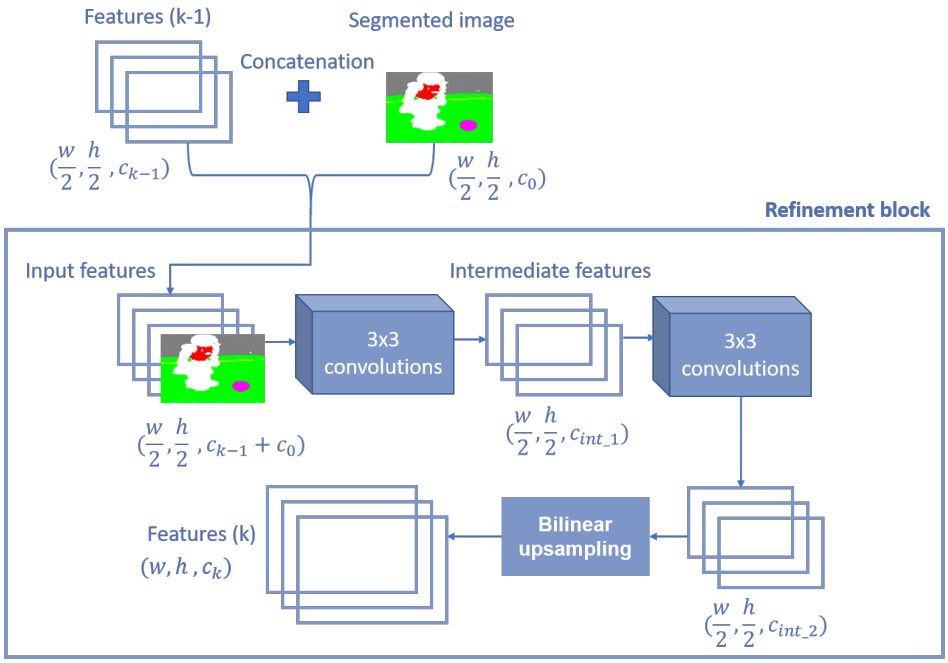


Figure 3.6: Information flow through a refinement module.

The network is trained using a variation of the feature matching loss, which was described in Sec. 3.2.11. First, the feature-Loss$(B, \hat{A}_n)$ for each image $\hat{A}_n$ with $n = 1...N$ is calculated. Then, the generator's loss is calculated as stated in Eq. 3.18:

$$\alpha \cdot \min(\text{feature-Loss}(B, \hat{A}_1), ..., \text{feature-Loss}(B, \hat{A_N})) + (1-\alpha) \cdot 1/f \cdot \sum_{n=1}^{N} (\text{feature-Loss}(B, \hat{A}_n)), \tag{3.18}$$

where α is a user tunable parameter. By generating multiple output images and using this particular loss function, the network is encouraged to diversify its outputs to cover the multiple plausible solutions for any given input. For a large α parameter the penalization of the network is given mostly by the image $\hat{A}$ which was closer to the ground-truth image. This is extremely useful when the input image is does not provide enough information to completely determine the output. This is better represented in Fig. 3.7 where the network is unable to determine the color of the robot's shirt from a class segmented image where all shirts are represented by a single value. The network then generates several plausible images which ultimately contributes to a much more stable training process.

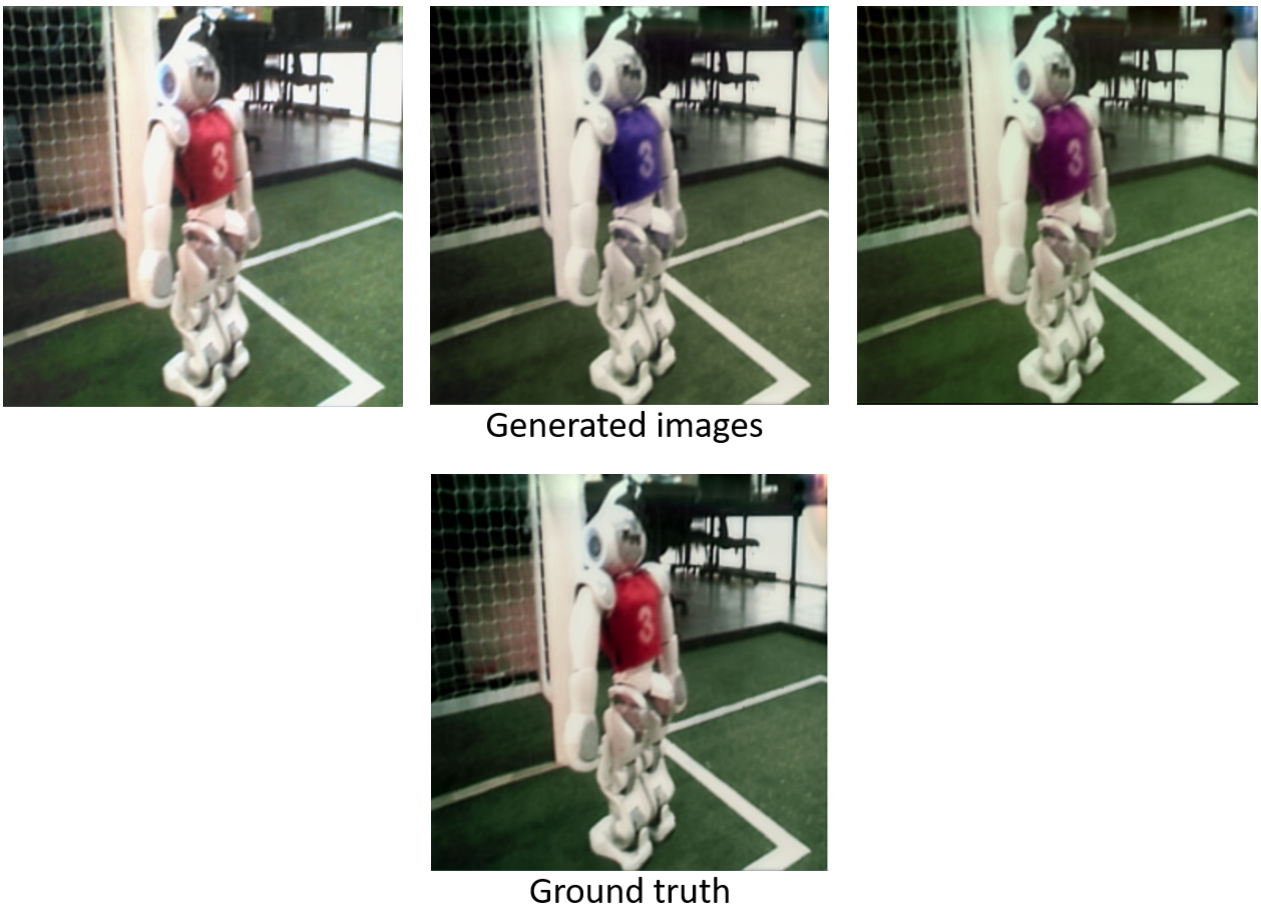


Figure 3.7: Different generated images by the Cascade Refinement Network.

U-net

U-net [50] is a neural network architecture for image-to-image translation and is based on an encoder-decoder architecture. As in traditional neural networks, the encoder is composed of numerous instances of 3×3 convolutions with ReLu non-linearities. These convolutions are organized in blocks which are followed by a strided convolutions which perform the down-sampling process. This allows the network to reduce the spatial size and increase the semantic value of the extracted features. The decoder is composed of several instances of 3 × 3 convolutional with Relu non-linearities uses transposed convolutions to up-sample layers and iteratively increase the spatial dimensions of the features. The decoder is capable of aggregating, processing and combining the features extracted by the decoder at several levels as shown in Fig. 3.8. Much like the cascade refinement network, this architecture allows the decoder to work with information at different spatial scales and as an added benefit also at different levels of semantic richness in order to from a prediction.

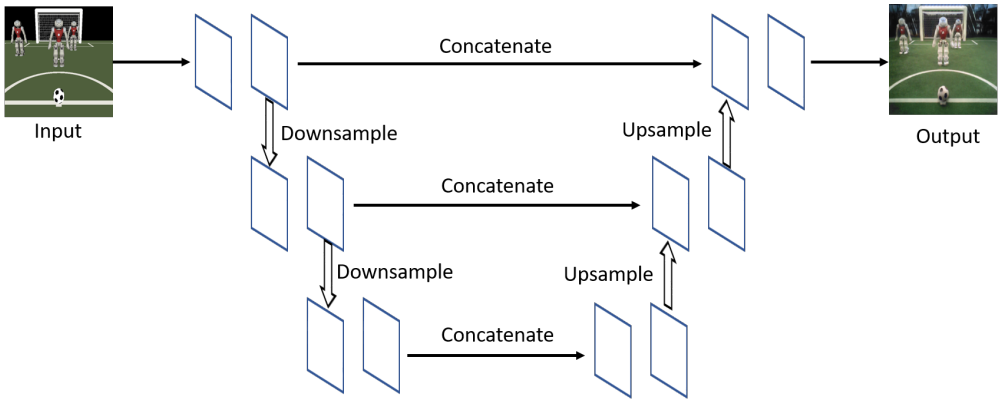


Figure 3.8: General structure of U-net.

Resnet generator

The Resnet generator [29] is a neural network to perform image-to-image translation which is depicted in Fig. 3.9. First, several strided convolutions are used to extract important features over the input image and spatially compress the information. The resulting tensor (called the encoded tensor) is the fed to a series of Residual blocks (see Sec. 3.2) which perform the image transformation. The tensor provided by the last residual block is then up-sampled to the original resolution using transposed convolutions to generate the output image. Using residual blocks rather than traditional convolutions offers some major advantages. First, a small deviation over the input tensor is learned rather than a complete mapping from input to output. Second, the identity path is fundamental in preserving the spatial information of the encoded tensor through the transformation process. Following the guidelines presented in [46] ReLu is the de-facto activation function in the network except in the last layer which uses Tanh to generate the output image since it provides a bounded output.

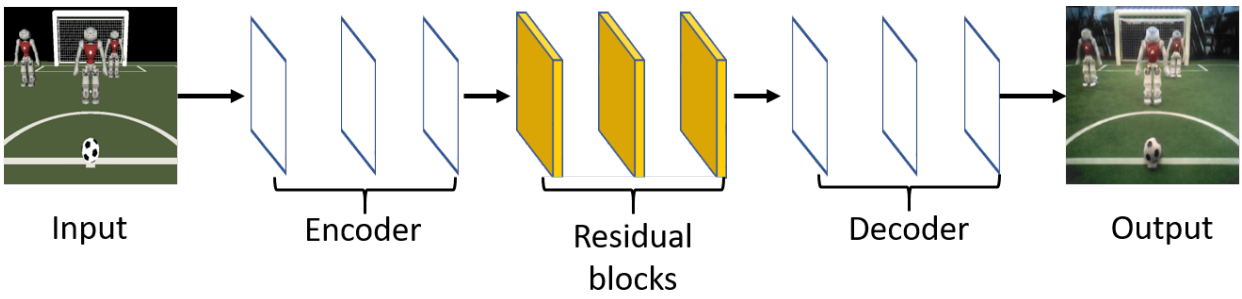


Figure 3.9: Different generated images.

3.6.2 Cycle GAN

CycleGan [71] is an image translation architecture which is composed of four neural networks, two generators and two discriminators, organized in two cycles (see Fig. 3.10). Given a domain Dom_A and a domain Dom_B represented by two datasets, CycleGan allows to generate

a sample $\hat{B}$ in the Dom_B domain from a sample A in the Dom_A domain such that the A and $\hat{B}$ are aligned. CycleGan is also able to generate aligned samples in the Dom_A domain from samples in the Dom_B domain.

While conditional adversarial networks are ideal candidates at the task image-to-image transfer, a major problem still persists at the time of constructing a training database. This is because in classical approaches a training database consisting of aligned (*input*, *label*) images is required. However, generating aligned image pairs across domains is difficult and time consuming. The CycleGan architecture addresses this problem by using unaligned image pairs to train the networks.

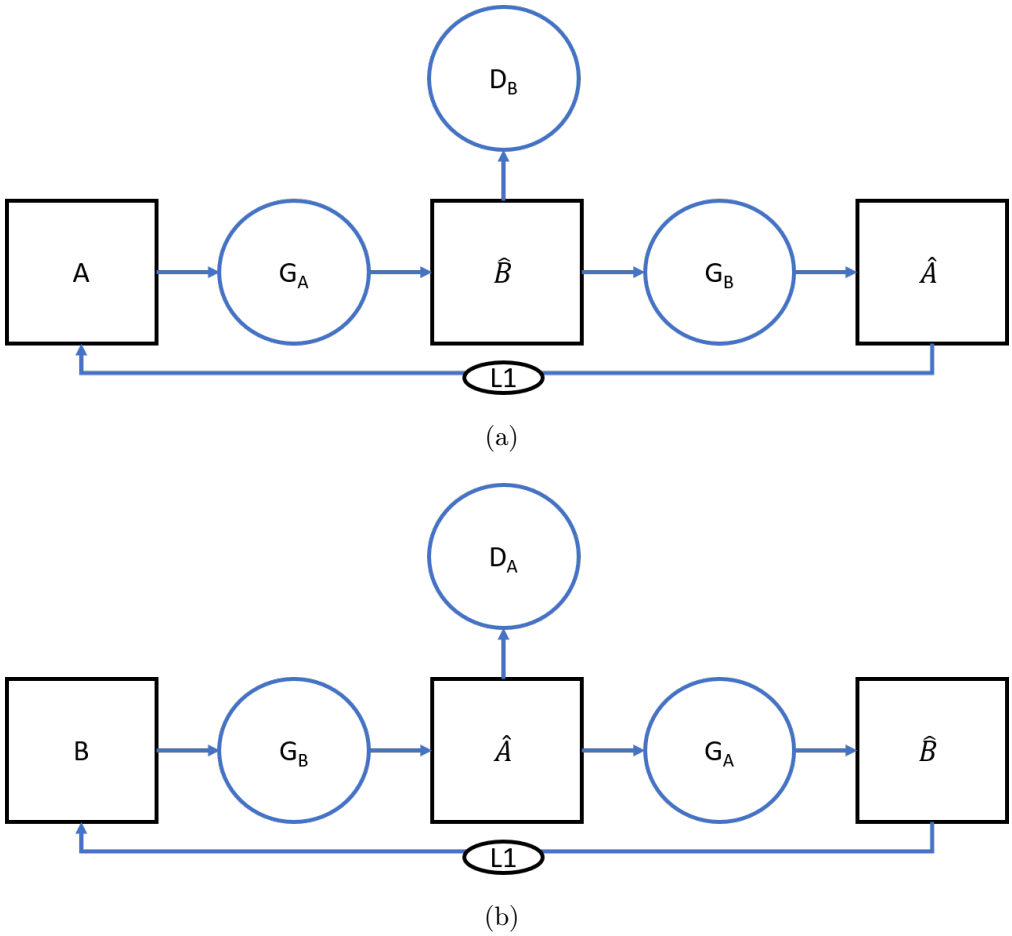


Figure 3.10: The CycleGan pipeline, a) cycle_A, b) cycle_B.

During training, the generator G_A takes an input A from the Dom_A domain and generates a sample $\hat{B}$ in the Dom_B domain. Then, $\hat{B}$ is fed to the PatchGan discriminator [27] D_B, which scores the image as real or generated and corresponds to the G_A-Loss. This is equivalent to the classical implementation of GAN and by itself does not guarantee that the input and output images are aligned. CycleGan uses the second generator G_B to transform the image $\hat{B}$ into a new image $\hat{A}$ in the Dom_A domain called the reconstructed image. Then, the the cycle_A-Loss is calculated as shown in Eq. 3.19 and is equivalent to Eq. 3.20:

$$\text{cycle}_A - \text{Loss} = \text{L1-Loss}(\hat{A}, A), \tag{3.19}$$

$$\text{cycle}_A - \text{Loss} = \text{L1-Loss}(\text{G}_\text{B}(\text{G}_\text{A}(A)), A). \tag{3.20}$$

Minimizing the cycle$_A$-Loss ensures that A and $\hat{A}$ are aligned and since $\hat{A} = \text{G}_B(\hat{B})$=$\text{G}_B(\text{G}_A(\text{A}))$ this means that the information of the layout of the input A must be preserved throughout the cycle to correctly generate the reconstructed image $\hat{A}$. By consequence, the generated image $\hat{B}$ must be theoretically aligned to A. The same process is repeated the opposite direction, forming the second cycle such that $\hat{B} = \text{G}_A(\hat{A})$=$\text{G}_A(\text{G}_B(\text{B}))$. The quality of the generated image $\hat{A}$ is estimated by the second discriminator D_A and corresponds to the G_B-Loss. Likewise, the estimation of the level of alignment between B and $\hat{B}$ is judged using the L1-Loss to which corresponds to the cycle$_B$-Loss. The authors argue that using two cycles greatly improve the quality of the results. From both cycles, the total loss to train the generators is defined as shown in Eq. 3.21:

$$\text{G-Loss} = \text{Cycle}_A - \text{Loss} + \text{Cycle}_B - \text{Loss} + \text{G}_\text{A} - \text{Loss} + \text{G}_\text{B} - \text{Loss}. \tag{3.21}$$

Depending on the problem, identity loss can also be used to improve the results. The identity loss was introduced by Taigman et al. [61] and in CycleGan [71] it is used to preserve color between domains and acts as a regularizer for the generators.

The identity loss is defined for each cycle in Eq. 3.22 and 3.23. Then, the total loss for CycleGan is presented in Eq. 3.24:

$$\text{idt-Loss} = \text{L1-Loss}(\text{G}_\text{B}(A), A), \tag{3.22}$$

$$\text{idt}_\text{B} - \text{Loss} = \text{L1-Loss}(\text{G}_\text{A}(B), B), \tag{3.23}$$

$$\begin{aligned} \text{G-Loss} = \text{cycle}_\text{A} - \text{Loss} + \text{cycle}_\text{B} - \text{Loss} + \text{G}_\text{A} - \text{Loss} + \\ \text{G}_\text{B} - \text{Loss} + \text{idt}_\text{A} - \text{Loss} + \text{idt}_\text{B} - \text{Loss}. \end{aligned} \tag{3.24}$$

Chapter 4

A supervised approach to bridge the simulation-reality-gap using a Cascade Refinement Network

As mentioned in Sec. 3.5 GANs are notoriously hard to train and are prone to produce artifacts. Given this, an alternative approach based on a generator trained with a traditional supervised approach represents and ideal candidate to perform the simulation-to-reality image translation task. The extra degree of control from a supervised training should ensure that the generated images $\hat{A}$ are correctly aligned to the input rendered images B. This section explores the necessary steps to train a supervised simulation-to-reality image translation network.

The proposed methodology consists of two main stages. The first one allows the semi-automatic generation of segmented/simulated-real image pairs using a state-of-the-art instance object segmentation network. The second stage implements the translation of segmented/simulated images onto realistic ones using a generative neural network. This network is trained using the image pairs generated in the first stage.

4.1 Generation of segmented-real image pairs using an instance object segmentation network

Supervised approaches require databases of aligned image pairs. For the simulation-to-reality image translation task this means that aligned pairs of simulated and real images are required. However, replicating the complete layout of a simulated scene in a real environment or the layout of a real environment in simulation would be prohibitively expensive in terms of time. Furthermore there is no clear way of precisely measuring the complete pose of all the objects in a real environment. The solution explored in this section uses two main ideas to solve this problem. The first one is to transform both datasets of real and simulated images to an intermediate domain were samples that come from reality are indistinguishable from samples collected from simulation. This intermediate domain corresponds to the semantic

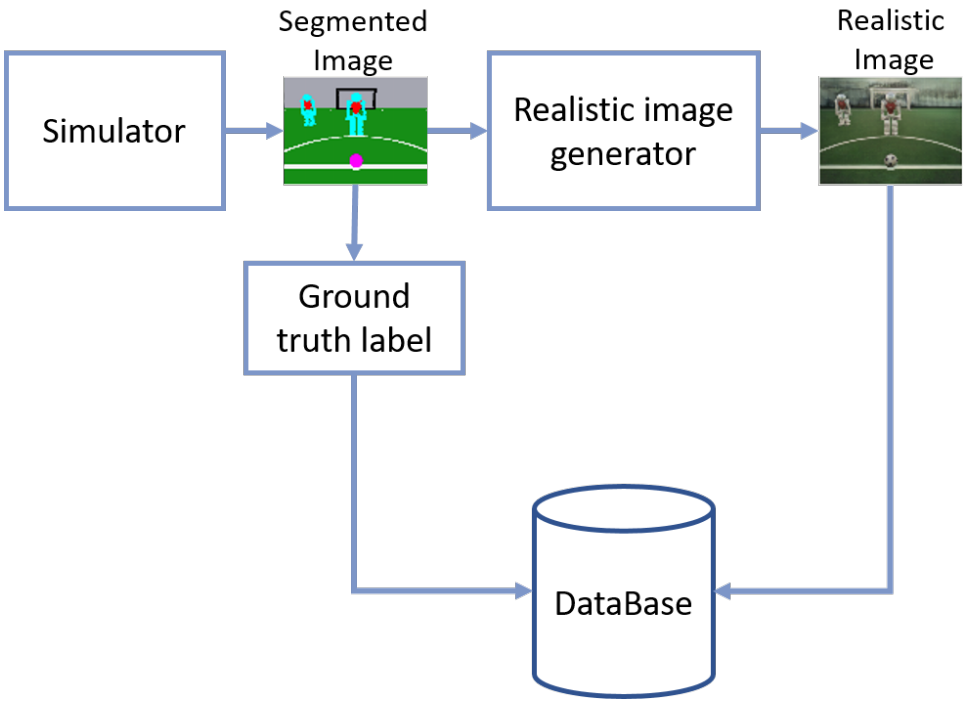


Figure 4.1: Pipeline used to generate a database containing realistic images and the ground truth (obtained from the segmented images) for the training of object recognition methods.

segmentation of the corresponding image. The second corresponds to using an active learning methodology to quickly and effortlessly label samples collected from reality.

In the proposed methodology a generator G is trained using segmented S and real A aligned image pairs to perform the segmented-to-reality image translation task. Then, to generate a realistic image $\hat{A}$ from a rendered simulated image B, the simulated image B is first processed to form the segmented image S_S which is then fed to the generator to be transformed into the aligned realistic image $\hat{A}$.

By using this method, realistic images can be obtained in situations dynamically generated by the simulator. Since the simulator also provides ground truth information, an automatically labeled database can be obtained to train vision algorithms. The realistic simulator also allows to collect metrics of the performance of the robot's algorithms which accurately represent the performance in real conditions. Fig. 4.1 shows how a database can be generated using the realistic simulator.

In the proposed methodology aligned segmented-real image pairs are generated by first obtaining real images from the target environment, and then by segmenting them using an instance segmentation neural network. In this work the Mask R-CNN [18] was selected for this task. Mask R-CNN is a CNN architecture designed to perform instance segmentation. Further detail about this network can be found in 3.3.4. The network outputs a mask for each object instance, alongside a bounding box, a label class and a confidence score. While several architectures can be used as a backbone for this model, Resnet-50 [17] was chosen due to its state-of-the-art performance. The basics of Resnet are presented in Sec. 3.3.3.

Training Mask R-CNN for a new application from scratch normally requires a large amount of labeled data. This problem is avoided by training the network using a three-step procedure.

First, Mask R-CNN is trained using the COCO dataset [36], which is a very large database of labeled images. By training the network in this dataset, Mask R-CNN learns relevant feature extractors which can then be re-purposed for other tasks.

Second, transfer learning is performed (see Sec. 3.2.12) to the target application domain. To implement this, a database of real images of the environment in which the autonomous robot will operate must be obtained. This database is called Data_A. Then, a small subset of around five images in this database are manually segmented by a human operator to form aligned pairs of real images A and segmented images S. The annotated database of real-segmented image pairs is named as Data_{AS}. Using the segmentation information, individual objects are extracted from each real image A. This dataset of objects is called Data_I and is used to perform domain randomization. Over a number of iterations some of the objects of Data_I are randomly selected. The same object can be chosen in two different iterations. Then, the scale and rotation are randomly modified and the object is placed over a background image in a random position. By using different permutations of randomly placed objects to create new images, a large database of images can be obtained from a very small Data_{I} database. Given the sheer number of possible images the network that is trained over this database cannot use brute-force to learn the complete dataset from the layout of the scene, instead being forced to generalize. While the position of the object in the scene is random, it can still be easily recognize given its color. Then, given the small total number of objects in Data_{I} the network could easily recognize them as the same object in each of the different images. This is prevented by using color jittering for each object before being placed in the image which involves randomly changing the brightness, contrast, saturation and hue of the objects which once again prevents the network from using brute force to learn the dataset. The set of images resulting from the domain randomization process is called $\text{Data}_{\text{AS-augmented}}$, and samples of the dataset are shown in Fig. 4.2.

(a)

(b)

Figure 4.2: Domain randomization, a) real image A, b) Aligned segmented image S.

Then, transfer learning for Mask R-CNN is carried out using the $\text{Data}_{\text{AS-augmented}}$ database. In order to train the network with a different number of classes than the original COCO

implementation, layers that are dependent on the number of classes are deleted and replaced. During training data augmentation is used, images are randomly flipped and the values of brightness, saturation hue and contrast are randomly changed. This process is different than the one used during domain randomization since that method used per object data augmentation, while during network training per image data augmentation is used, meaning that the statistics of the whole image are changed. This increases the network's resilience to changes in global illumination as well as to changes in the camera settings of the robot.

Third, active learning is used to improve the instance segmentation results: in each iteration, the trained Mask R-CNN network is used to perform automatic labeling of the unlabeled images from a subset $\text{Data}_{\text{A-subset}}$ of the Data_{A} database. In the first iterations, these predictions will be very unreliable since the network is trained over set of samples constructed from only a few objects of Data_{I}. The predictions of the network are manually corrected by a human, and then added to the $\text{Data}_{\text{AS-subset}}$ database. Then, domain randomization is performed as previously described and the resulting samples are added to the $\text{Data}_{\text{AS-augmented}}$ dataset. The Mask R-CNN network is then retrained using this dataset using the weights resulting from training in the COCO dataset as starting point. Thus, by following this active learning procedure (see Algorithm 1), the performance of the Mask R-CNN model is rapidly increased, while requiring minimal human interaction. The process is repeated until the Mask R-CNN network achieves a satisfactory accuracy at labeling the Data_A database.

Once this condition is met, Mask R-CNN is used to predict the segmentation of all the images in the Data_{A} database in order to obtain a large number of real-segmented pairs. After discarding badly segmented images, the relevant objects of this database are extracted and a final domain randomization step is employed to obtain the final $\text{Data}_{\text{AS-augmented}}$ database.

Algorithm 1 Active learning training process

obtain D_{real}
$\text{Data}_{\text{AS-subset}} = \emptyset$
$\text{Data}_{\text{AS-augmented}} = \emptyset$
while Mask R-CNN accuracy $\leqslant$ min-accuracy **do**
    //choose a small number of new samples
    $\text{Data}_{\text{A-subset}}(\text{j}) \subseteq \text{Data}_{\text{A}}$
    for $\text{i} \in \text{Size}(\text{Data}_{\text{A-subset}})$ **do**
        $A_{\text{i}} = \text{Data}_{A-subset}[\text{i}]$
        $S_{\text{i}} = \text{Mask R-CNN}_{infer}(A_{\text{i}})$
        $S_{\text{i}} = \text{Manual-correction}(S_{\text{i}})$
        $\text{Data}_{\text{AS-subset}}.\text{add}(A_{\text{i}},\ S_{\text{i}})$
        for *object* $\in \text{Data}_{\text{AS-subset}}$ **do**
            $\text{Data}_I.\text{add}(object)$
    $\text{Data}_{AS-augmented} = \text{Domain-randomization}(\text{Data}_I)$
    $\text{Mask R-CNN}_{train}(\text{Data}_{\text{AS-augmented}})$

4.2 Realistic image generation using a generative neural model

As explained in Sec. 3.4, generative models trained on a dataset aim to generate new samples following the distribution of such dataset. Two of the most popular approaches involve using GANs (Generative Adversarial Networks) [16] and VAE (Variational Auto Encoders) [30]. However, these approaches are prone to artifacts and suffer from unstable training processes, problems such as mode collapse and high computational costs during training. Given this, a supervised approach based on a supervised training methodology emerges as an ideal candidate.

The chosen generator is the cascade refinement network (CRN) which was described is described in length in 3.6.1 and was first introduced by Chen and Koltun [8].

The generative model is trained using the $\text{Data}_{\text{AS-augmented}}$ database without using any saturation, hue or exposure data augmentation. This allows the model to be more temporally consistent in terms of lighting. Then, once the model has achieved a high enough accuracy on the $\text{D}_{\text{RS-augmented}}$ database, the model is fine-tuned by using the $\text{D}_{\text{RS-subset}}$ database in order to be able to learn complex image features such as shadows and reflections.

Human labeling is only sparsely necessary to create the database $\text{Data}_{\text{AS-augmented}}$ and $\text{Data}_{\text{AS-subset}}$ which are used to train the generator network. Once the network is trained, no further human interaction is needed, and the network can be used to generate realistic images. This is done by segmenting the simulated image and then feeding the segmented image to the generative neural network that then outputs a realistic image. Given that all the generated realistic images have a corresponding aligned segmented image, and therefore the objects in that images are known, a database composed of realistic images with ground truth can be quickly obtained without any human labeling in order to train machine learning based algorithms (e.g. CNN based detectors).

Chapter 5

Unsupervised approach

This section presents FeatureGan, a methodology developed for this thesis to train a generator using unaligned images belonging to two different domains Dom_B and Dom_A. Once the generator is trained, it can be used to perform aligned image to image translation from Dom_B to Dom_A.

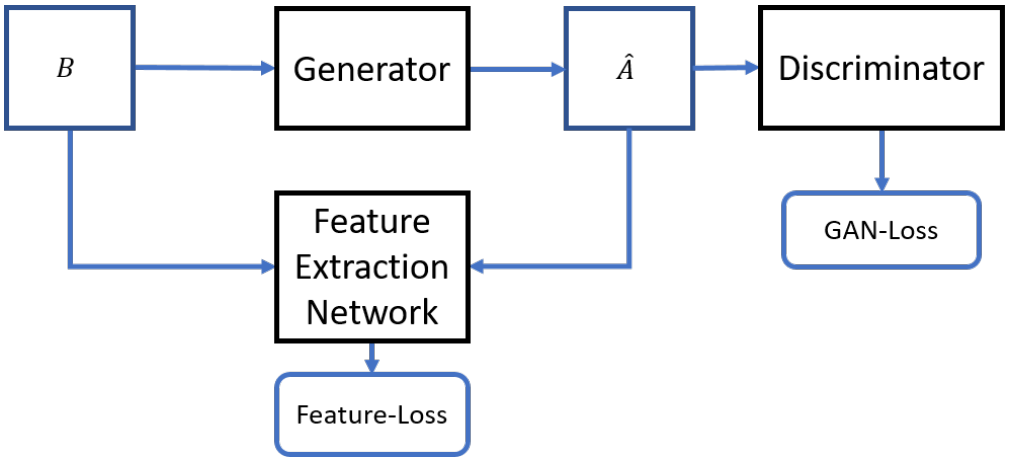


Figure 5.1: Pipeline used to train the Generator. The generator takes an image B in the domain Dom_B as input and outputs a generated image $\hat{A}$ in the Dom_A domain.

During training, FeatureGan makes use of three neural networks (see Fig.5.1). The generator network G is used to perform the domain transfer by taking an input image B in the Dom_B domain and outputting a generated image $\hat{A}$ in the Dom_A domain. The loss function GAN-Loss scores the quality of the generated image by checking how close it is to the Dom_A domain via a discriminator network D.

To preserve the layout of the input image B at the output image $\hat{A}$ FeatureGan uses a third network, the Feature Extraction Network, to calculate a feature-Loss between B and $\hat{A}$. If the Dom_B and Dom_A domains are similar enough then the feature-Loss should be minimized when B and $\hat{A}$ are aligned, and maximized when B and $\hat{A}$ are unaligned. The GAN-Loss and the feature-Loss are combined into a single loss function which is used to

train the generator G to produce realistic images in the Dom_A domain, while preserving the geometric layout of the input image B. To further improve the quality of the results, the ground truth information from the simulator is used to introduce labels into the training process by adding them as extra channels to the generator during training.

5.1 Feature Extraction Network

The feature-Loss, defined in Sec. 3.15, is used to force the generator to preserve the layout of the input image B at the output image $\hat{A}$ and by consequence encourage image alignment. Similar approaches have been used in the style transfer task [14], image to image to image translation [38] and conditional image synthesis task [8]. Given a Feature extraction network, which was pre-trained on some database, in this case imagenet [11], the alignment is measured as feature-Loss$(B, \hat{A})$. Supposing that the Dom_A and Dom_B domains are similar, then if $\hat{A}$ and B are aligned, the corresponding activation maps and should be similar and the feature-Loss should decrease. On the other hand, if $\hat{A}$ and B are not aligned the difference between the corresponding activation maps and the feature-Loss should increase.

Since classification CNN networks are trained to be invariant to changes in illumination, then it can be assumed that the illumination information is mostly lost as the data flows through the network. It follows that the activation maps of two aligned images with vastly different illuminations should be approximately equal for deep layers of the network. Given this property, the generator G is allowed to change the illumination of $\hat{A}$ with respect to B as necessary, since there is little to no penalization from the feature-Loss for doing so. The same property allows the generator to make small adjustments to the color distribution of the scene. The network of choice for feature extraction is a modified VGG-19 [57] which is explained with more detail in Sec. 3.3.2. The traditional version of the network includes max-pooling, which as shown in Sec. 3.2.6 can be interpreted as a high pass filter that preserves the detail of the input. This is useful for task such as classification, however for this application it is fundamental that two images that are similar in their layout but have a different level of detail share similar responses from the VGG-19 network. This is accomplished by using a low pass filter in the form of average pooling which replaces the max-pooling operations. The size of the pooling window determines the level of detail that is preserve from input to output. Using average pooling may also contribute to better preserve the spatial information of the features in classifiers as stated in [70] which is beneficial for this use case. Another constraint for the architecture of the network is that it must be able to preserve the relative position of the objects in the image. This is of utmost importance given that the network is used to check how well aligned both images are. However, as discussed in Sec. 3.2.6 pooling operations are used, among other things, to make the network invariant to translations which means that relevant spatial information is lost in the pooling layers. To accurately preserve the spatial relations of the input through pooling layers coarse coding is used [20] in the form of overlapping avg-pool windows. Alongside the strided convolutions, pooling operations result in a loss of high resolution, high frequency features of the input image. Since the high frequency features of the input images are lost in deeper layers of the network, the generator G will be able to add small details to the generated image without any mayor increase in feature-Loss. These additional details translate in more detailed textures and better lighting. Performing layer normalization without affine parameters (see Sec. 3.2.9)

over $\hat{A}$ and B further helps to maintain alignment in images with low contrast by making the boundaries between objects more visible. Furthermore, applying layer normalization to the input images helps to increase resilience to changes in brightness.

Recently, it has been proven that mean and variance of the data hold significant information about the nature of the input domain [23]. The objective of the feature-Loss is to compare the layout of the scene between images of different domains which is done by comparing the feature maps $\text{Map}_l(B_j)$ and $\text{Map}_l(\hat{A}_j)$. However a large mean or variance difference between $\text{Map}_l(B_j)$ and $\text{Map}_l(\hat{A}_j)$ would mean a large feature-Loss and would be the result of the difference in domains Dom_A and Dom_B rather than the different in layout between B_j and $\hat{A}_j$. Given this, the feature-Loss would penalize the generator for performing a domain change from Dom_B to Dom_A. Then, it is fundamental to reduce difference in mean and variance between $\text{Map}_l(B_j)$ and $\text{Map}_l(\hat{A}_j)$ to a minimum. As proposed in [24] this is done by applying instance normalization (see Sec. 3.2.9) to the feature maps without using affine parameters.

Finally, using instance normalization over the feature maps $\text{Map}_l(\hat{A})$ and $\text{Map}_l(\hat{B})$ can lead to a better estimation of the difference between activation maps, since all the feature channels will have similar magnitudes rather than having certain feature channels having values orders of magnitude greater than others. With these changes the original feature-Loss presented in Eq. 3.15 is modified to Eq. 5.1:

$$\text{feature-Loss}(B, \hat{A}) = \sum_{l=1}^{L} (\gamma_l \cdot \text{L}_{\text{norm}} - \text{Loss}(\text{Map-N}_l(B), \text{Map-N}_l(\hat{A})), \tag{5.1}$$

where $Map - N_l(y)$ is defined as the instance normalization without affine parameters of the features extracted by the feature extraction network from the input image y as shown in Eq. 5.2:

$$\text{Map-N}_l(y) = \text{Instance-norm}(\text{Map}_l(y)). \tag{5.2}$$

When generating images, certain regions of the generated image benefit from having a lesser or higher degree of penalization from the feature-Loss. Classes that are similar between the Dom_B and Dom_A domains can be strictly penalized to avoid artifacts and ensure alignment, while classes that are very different between the Dom_B and Dom_A benefit from a lower level of penalization which allows the generator to do more changes and generate a more realistic image. FeatureGan introduces a per class feature loss to have an extra degree of control over the penalization of the generated image $\hat{A}$. Since the ground truth for each image B is provided by the simulator, then this information can be used to separate both B and $\hat{A}$ into different images B_j and $\hat{A}_j$, with $j = 1...J$, and J the number of classes in B as shown in Eq. 5.3 and Eq. 5.4:

$$\begin{cases} \text{pix}(B_j) = \text{pix}(B) & \text{if class}(\text{pix}(B)) = j \\ \text{pix}(B_j) = 0 & otherwise \end{cases} \tag{5.3}$$

$$\begin{cases} \text{pix}(\hat{A}_j) = \text{pix}(\hat{A}) & if\ \text{class}(\text{pix}(B)) = j \\ \text{pix}(\hat{A}_j) = 0 & \text{otherwise} \end{cases} \tag{5.4}$$

Then, the feature-Loss can be independently calculated for each class by slightly modifying the original feature-Loss defined in Eq. 3.15. The per-class feature-Loss is defined in Eq. 5.5:

$$\text{feature-Loss}_j(B_j, \hat{A}_j) = \alpha_j \sum_{l=1}^{L} (\gamma_l \cdot \text{L}_{\text{norm}} - \text{Loss}(\text{Map-N}_l(B_j), \text{Map-N}_l(\hat{A}_j)), \tag{5.5}$$

where α_j is a user tunable parameter which measures how similar an object of class j in $\hat{A}_j$ must be to the corresponding object in B_j. By relaxing this parameter for a particular class, the generator will be able to introduce more changes to the class, but the preservation of the geometric layout constraint will also be relaxed. The value of norm represents how the difference between the activation maps for B and $\hat{A}$ are calculated, L1-norm (see Sec. 3.12) and L2-norm (see Sec. 3.13) are used depending on the constraints for each class.

L1-norm tends to generate images with better texture quality while L2-norm results in a higher penalization for outliers and is ideally used for classes which are prone to producing artifacts or to generate unaligned results.

5.2 A Feature Pyramid Based Discriminator Approach

5.3 The discriminator architecture

An approach consisting of multiple PatchGan discriminators which [27] operate at different scales is used. A traditional PatchGan discriminator, as proposed by Isola et al. [27], is composed of a fully convolutional network with a narrow receptive field (usually 70x70), which extracts local features from an the input-image to form a vector of predictions. The small receptive field improves generalization and results in a more stable training since regions of the image are rated independently, meaning that the generator is still rewarded for small local improvements in image quality. Furthermore, by having a small receptive field, the discriminator tends to estimate the quality of the input-image based on the quality of specific textures. This is in line with the findings presented by Isola et al. [27]. Luo et al. [40] introduced two important ideas. First, the effective receptive field of a neural network is smaller than its theoretical receptive field. Second, regions closer to the center of the input have more impact on the output than regions at the borders. These two properties further help to explain why region-based discriminators achieve better texture quality than global discriminators. Since each small region of the image is processed independently, then each region of the image has the same impact in the final loss. Furthermore, since the PatchGan discriminator uses overlapping regions, then each pixel of the input-image will be closer to the center of a receptive field. Altogether, this means that when using a PatchGan discriminator with a narrow receptive field, the relevance of each pixel in the input image contributes more equally to the total loss function, while in the case of a discriminator with a large receptive field, certain pixels contribute more to the total loss than others.

In traditional neural network design, higher abstraction features are provided by the deeper layers in the neural network, while high resolution features come from the layers closer to the input. FeatureGan introduces an improved version of PatchGan loosely based on the idea behind Feature Pyramid Networks (FPN), which were introduced by Lin et al. [37] to improve the performance of image detectors and semantic segmentation networks. As explained in Sec. 3.3.1 the feature pyramid network approach consists of upsampling the deeper feature maps generated by a CNN. The upsampled features are then combined with feature maps generated at layers closer to the input of the network. The tensors resulting from this process contain both the high resolution features from layers close to the input as well as the high level features from deeper layers of the network.

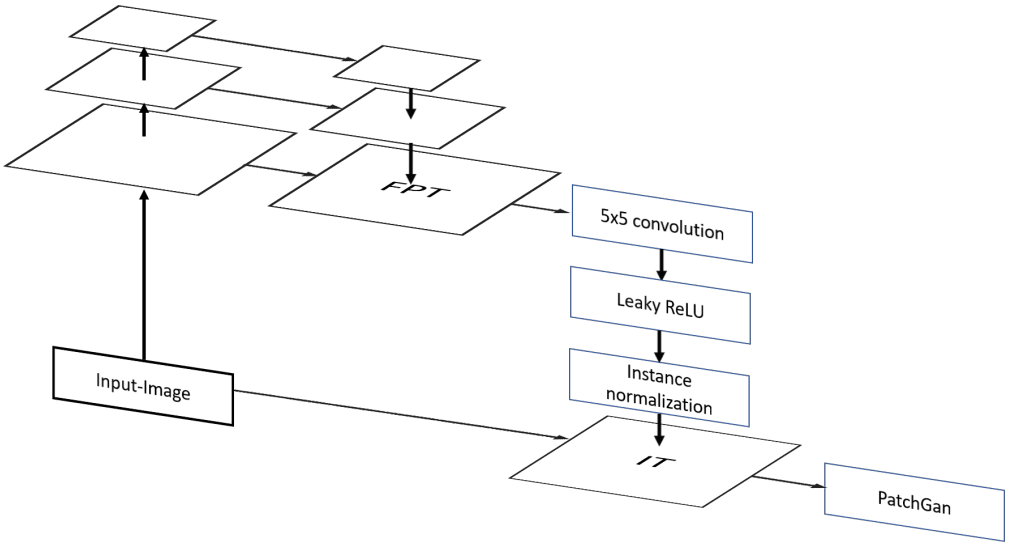
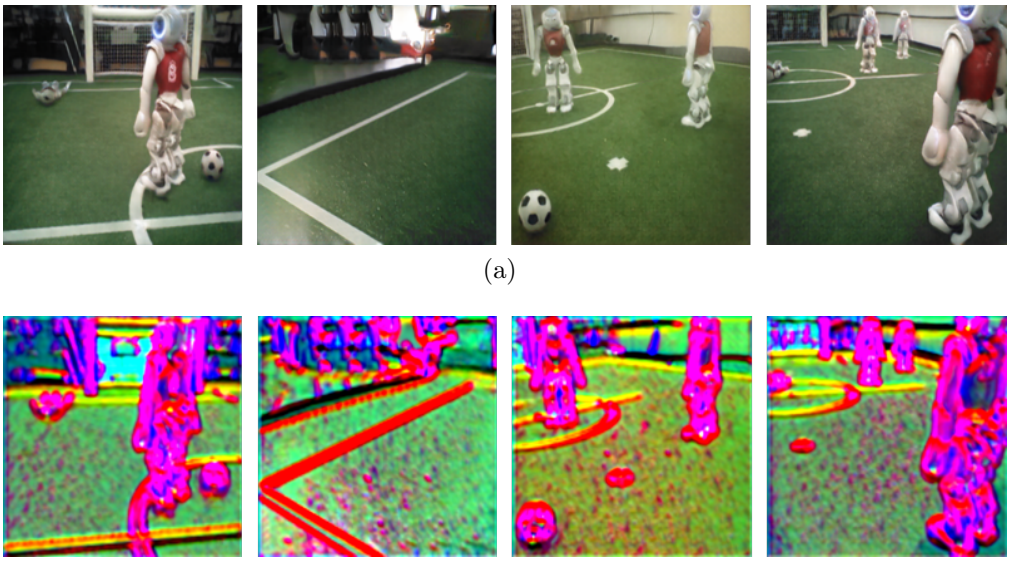


Figure 5.2: Proposed FPN discriminator.

The traditional implementation of a FPN includes a trainable CNN network. However, the extra number of tunable parameters would mean that balance between generator and discriminator would be broken since the learning capacity of the discriminator would greatly increase. To avoid this problem, the input-image is fed to a VGG-19 network which was pre-trained in imagenet [11]. Feature maps at different depths of the network are then extracted, up-sampled to the input image size and then concatenated into a single Feature Pyramid Tensor (FPT). Then a convolution with kernel size 5×5 and stride 1, followed by a leaky ReLU activation function and finally an instance normalization layer is used to compress the information of FPT into a tensor IT of N channels. As previously stated, the mean and variance of a feature channel holds significant information about the nature of the domain. The large kernel size of the convolution is designed to capture these statistics as well as spatially spread features over FPT while the small number of filters acts as a bottleneck to ensure that only the most relevant information is extracted. In Sec. 3.2.9 it is stated that instance normalization changes the mean and variance of each feature channel. Consequently, the normalization layer is placed after the leaky ReLU layer to ensure that this important information is passed through the non-linearity and is preserved in the form of features. This layer configuration is based on the same design principles that dictate the architecture of the first layers of PatchGan. While it might seem logical to choose a traditional ReLU

layer over leaky ReLU since the former could discard unnecessary information from FTP by setting the corresponding outputs to zero, empirically it was found that the results actually become worst. This is probably the result of the continuous change in the generated images. Features from FPT which were useful to differentiate between real and generated images in the first stages of training might not be useful in the later stages and vice-versa. This means that to achieve good results throughout training the sparsity configuration of the model should constantly change. However, as explained in 3.2.10 once the output of a ReLU is set to zero it is very difficult to recover, since there is no gradient flowing through that node. Given this, leaky ReLU arises as the natural choice. Then, the tensor at the output of the leaky ReLU activation function is passed through the instance normalization layer with affine parameters. The normalization brings all the features to the same magnitude as the features present in Input-Image while the affine transformation allows the network to change the statistics for each feature channel in the tensor by just tuning the affine parameters. This is a responsive way of modifying the relevancy of each channel independently. Finally, this tensor is concatenated to the Input-image and fed to the PatchGan discriminator which corresponds to the rest of the FPN discriminator. The process is shown in Fig. 5.2. As an example, the resulting compressed FPT from the 8th and 12th layers for the sim-to-real image translation task is shown in Fig. 5.3.



(a)

(b)

Figure 5.3: (a) Input images, (b) compressed FPT tensor with $N = 3$.

From these images, it is apparent that the compressed features give the discriminator important information on the shape and position of the relevant objects in the scene (robots, lines and ball), which have a high contrast with the background information. Finally, the compressed FPT is concatenated with the input-image to form the final input IT to the discriminator. To ensure that the magnitudes of the compressed FPT are similar to those of the input-image, instance normalization is performed over IT. Using IT as input to the discriminator instead of the input-image offers several advantages. First, the features of the FPT come from layers of different depth with different receptive fields, meaning that the discriminator has access to spatially local and global features simultaneously. Second,

the discriminator still works by classifying small regions and has access to the extracted feature maps as well as to the original image, therefore, its ability to generate very realistic textures is preserved. Third, in addition to texture information, each region classified by the discriminator has also access to high level features. By using both simultaneously, the discriminator can achieve better performance at classifying each individual region. Most importantly, the features extracted by the pre-trained network are fixed through the training process and are independent of the quality of the images inferred by the generator. This helps to avoid overfitting of the discriminator to the current state of the generator and to stabilize the training process. Finally, since the VGG-19 network has no tunable parameters this means that the learning capacity of the discriminator is maintained. Specifically, the total number of parameters specifically tuned to distinguish real from generated samples remains the same which helps to maintain the balance of power between the generator and the discriminator.

While local discriminators have the advantage of improved texture quality and a more stable training function, they do have some major drawbacks, mainly their inability to extract global spatial relations in the image. Isola et al. [27] found that small patch sizes lead to an increase level of artifacts in the image, which can be attributed to the lack of spatial awareness of a small region size discriminator. We also found that discriminators with larger region sizes lead to spatial consistency in terms of illumination and object placement. Following the guidelines of Wang et al. [67], we construct several discriminators to evaluate the input tensor at different resolutions. Each discriminator is run over the IT with a different level of down-sampling.

The combination of the loss GAN-Loss$_\mathrm{i}$ provided by each discriminator i results in the total discriminator loss shown in Eq. 5.6:

$$GAN - Loss = \sum_{\mathrm{i}=1}^{I} (\lambda_\mathrm{i} * \text{GAN-Loss}_\mathrm{i}), \tag{5.6}$$

where λ_i is a parameter which indicates the relative importance of the i-th discriminator, and I is the number of resolutions.

5.4 Training the discriminator

Given a discriminator D, a real image B and a generated image $\hat{A}$, a classical loss to train the discriminator is defined as:

$$\begin{aligned} \text{D-Loss} = 0.5 \cdot \text{MSE}(\text{D}(A), 1) + \\ 0.5 \cdot \text{MSE}(\text{D}(\hat{A}), 0). \end{aligned} \tag{5.7}$$

To achieve smoother probabilities and avoid extremely confident predictions, one-sided label-smoothing is used [53]. The final loss function for the discriminator is presented in Eq. 5.8:

$$\begin{aligned} \text{D-Loss} = 0.5 \cdot \text{MSE}(\text{D}(A), 0.9) + \\ 0.5 \cdot \text{MSE}(\text{D}(\hat{A}), 0), \end{aligned} \tag{5.8}$$

where the only change is a lower value as target for real samples.

In [19] the authors showed that GANs trained by using different learning rates for the generator and the discriminator converge to a local Nash equilibrium. Following this guideline, the learning rate was selected accordingly all the discriminators to be equal to two times the learning rate of the generator.

To prevent model oscillation and to avoid major changes in the discriminator, an historic record pool of generated images is kept as suggested in [56]. Samples from this image pool are randomly selected to train the discriminator.

Finally, the values of saturation, contrast, hue and brightness are slightly modified for A and $\hat{A}$ to avoid the discriminator from overfitting to the data. Bigger modifications of these values will result in generated images with inaccurate colors.

5.5 The Generator

The generator is used for domain transfer and from an input image in the simulated domain generates an output image $\hat{A}$ in the real domain. In this work two architectures for the generator were tested including Resnet [29] and U-net [50] (see. 3.6.1 and 3.8).

Empirically, U-net was found to produce the best overall results in the sim-to-real task for the SimRobot simulator. This is in line with results from other works where U-net has been found to "produce strong results in the unimodal image prediction setting when there is a spatial correspondence between input and output pairs" [72]. Given the small variance of the input and target domains, and the similarity between both U-net proves to be a perfect fit for the task.

More complex domains require a different generator that is able to more aggressively modify the input image. The Resnet generator was used with excellent results to solve the real-to-real image translation task and the video game-to-reality task.

The generator is trained using gen-Loss (see Eq. 5.9), which is built using the feature loss and the discriminator loss described by equations 5.5 and 5.6:

$$\text{gen-Loss} = \sum_{j=1}^{J} (\text{feature-Loss}_j(B_j, \hat{A})) + \sum_{\text{i}=1}^{I} (\lambda_\text{i} * \text{GAN-Loss}_\text{i}), \tag{5.9}$$

with J the number of classes and I the number of resolutions/discriminators.

5.5.1 Introducing a supervised function for image alignment.

Since FeatureGan doesn't use a Cycle-Loss as CycleGan does, the alignment between images is preserved exclusively by the feature-Loss. There is a fine balance between the feature-Loss and the GAN-Loss which determines the level of alignment between B and $\hat{A}$ as well as the level of domain change that the generator is able to perform. This balance between both loss functions is difficult to properly tune. Furthermore, since the feature-Loss is calculated based on the features extracted by a deep neural network, this means that small objects in B might not be properly replicated in $\hat{A}$ since as discussed in Sec. 3.2.7 details in the input image are lost in the deeper layers of the network. The necessity of a loss function which penalizes badly aligned images while being independent of the level of domain change is apparent. In theory, a Cycle-Loss function could be used to achieve this objective. However, a Cycle-Loss does not perform well in the sim-to-real image translation task, as shown in Sec. 6.1.2. Furthermore, using a Cycle-Loss is very computationally expensive since a new generator and a new discriminator are needed to compute the loss. Theoretically, providing the semantic map S to the generator in addition to image B would give enough information to precisely reconstruct the layout of the scene B in $\hat{A}$. This allows the generator to have simultaneous access to the class and texture information in B which reduces the number of plausible outputs $\hat{A}$ for a given input B. The class information also gives valuable information on the boundary between objects. However, there is no clear way of forcing the network to use this information and to preserve it throughout the multiple layers of the generator. Given this, a second alignment function called label-Loss is proposed based on the ideas behind the identity loss discussed in 3.2.11. The function requires a modified architecture of the generator. A version of the modified Resnet architecture is shown in Fig. 5.4.

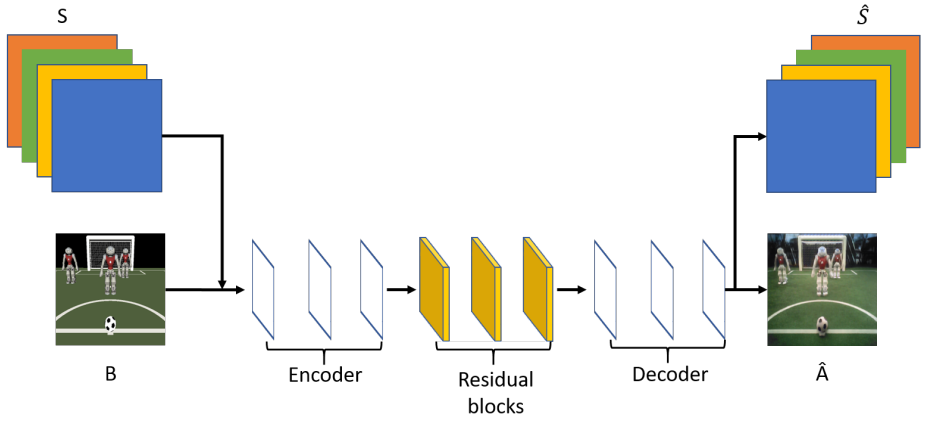


Figure 5.4: Modified Resnet architecture.

Given a set of L class labels available in the form of a semantic map, these are encoded into a tensor S composed of L one-hot encoded vectors, concatenated to the input image B and fed to the generator. The last convolutional layer of the generator then produces an output tensor of $3+L$ channels. The first three channels are passed through a tanh activation function to generate the output image $\hat{A}$ while the remaining L channels are passed through a sigmoid activation function to generate the tensor $\hat{S}$ with L channels. $\hat{A}$ is penalized as in the classic FeatureGan using a combination of the feature-Loss and the GAN-Loss. On the

other hand $\hat{S}$ is penalized using the binary cross entropy (see Sec. 3.2.11) loss between $\hat{S}$ and S as shown in Eq. 5.10:

$$\text{label-Loss}(S) = \text{BCE-Loss}(S, \hat{S}). \tag{5.10}$$

This has several effects. First, the network is actively encouraged to preserve the semantic information given by S throughout the network and since the last layer produces both the predicted semantic map and the output realistic image then the layout of the scene must be available in some form to all layers of the network. The encoder-decorder architecture prevents the network from learning the identity function to map S into $\hat{S}$ since the information is compressed. This loss also provides the encoder and the decoder with supervision to properly compress and decompress the layout of the scene. Furthermore, this loss function also takes into account small objects which the feature-Loss might not be able to properly capture. For a multi-class classification problem of N classes, Lin et al. [35] proposed in the architecture Network in Network (NiN) the novel idea of performing global average pooling over the last N channels a classifier to determine the predicted class. According to the original paper, this acts as a regularizer that encourages feature channels to correspond to categories. This falls in line with an earlier design principle of the generator, that of sparsity which states that only certain parts of the network are active for any given input. It is expected that using L channels to predict the semantic map of the input will have a similar effect of specializing certain feature extractors to certain inputs. In fact, by using global average pooling over the Predicted Semantic labels the generator would effectively become a classifier. Finally, label-Loss avoids some of the main problems of the identity-Loss. Since in label-Loss the pixels in one channel can only take binary values then the number of plausible outputs is orders of magnitude smaller than the number of outputs of a predicted RGB image $\hat{A}$. Furthermore, the binary cross entropy loss maps pixels to one of the two values meaning that pixels with high uncertainty are heavily penalized. This prevents the network from outputting the mean of all plausible solutions to approximate the target distribution which was a common problem of losses based in the L1 norm and lead to blurry images. Then, the final loss function for the generator is described by Eq. 5.11.

$$\text{gen-Loss} = \sum_{j=1}^{J} (\text{feature-Loss}_j(B_j, \hat{A})) + \sum_{\text{i}=1}^{I} (\lambda_{\text{i}} * \text{GAN-Loss}_{\text{i}}) + \text{label-Loss}(S). \tag{5.11}$$

5.5.2 Preventing training collapse

Collapse is a usual occurrence during training particularly for very complex domains where the generator is unable to correctly transform images from the Dom_B to the Dom_A domain and consequently the discriminator is able to constantly differentiate real from generated images. This section introduces a way to use the feature pyramid tensor to prevent training collapse. Some approaches such as [21] propose using a GAN-Loss over the features extracted by a feature extraction network. However, they also calculate a second GAN-Loss over the input-image. The feature pyramid discriminator combines both the input image and

the features extracted by the feature extraction network into a single tensor IT to provide semantic information of the contents of the input-image. In this section a variation of the Feature Pyramid Network is proposed where the input-image is completely discarded to avoid training collapse. In a traditional GAN-Loss the input-image is analyzed by a discriminator which provides a label which indicates if the image is real or generated. This encourages the discriminator to learn very specific, highly detailed features over the input-image A and $\hat{A}$ rather than more semantically rich features. Examples of these highly specific features might include the noise distribution of Dom_A or even small artifacts from the camera used to capture the environment. It is arguable that such features are difficult to replicate by the generator and contribute very little to generating realistic images. Traditional classifiers, such as the feature extraction network based on VGG-19, are known to discard unnecessary and noisy information while extracting increasingly semantically rich features as the depth increases. This means that the feature maps of A and $\hat{A}$ extracted from deeper layers of feature extraction network would appear increasingly similar to a discriminator. It is then theorized that training collapse can be potentially prevented by calculating a GAN-Loss over a IT composed only of the feature maps resulting from A and $\hat{A}$ and without using the input-image. The modified architecture is shown in Fig. 5.5

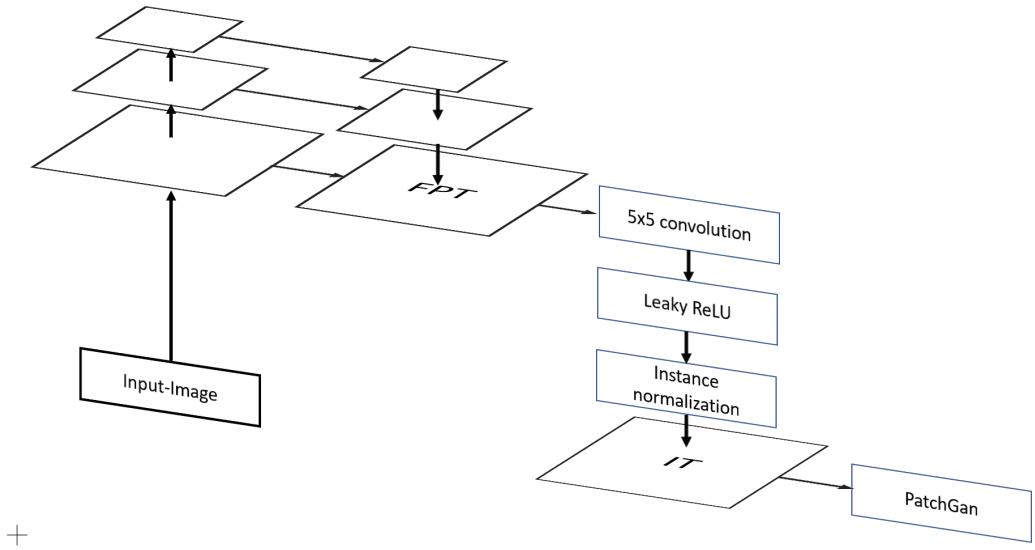
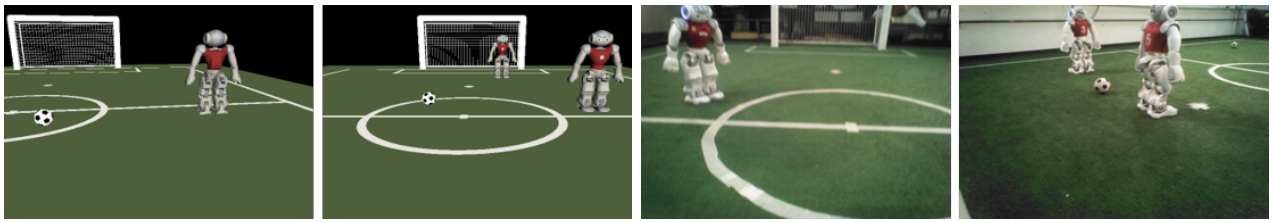


Figure 5.5: Modified FPN discriminator.

Chapter 6

Results in soccer robotics

6.1 Simulation-to-Reality Image Translation in Soccer Robotics



(a) (b)

Figure 6.1: a) simulation training samples, b) real training samples.

As a proof of concept, both FeatureGAN and CRN were used to solve the sim-to-real image translation task in soccer robotics. Both methods were used to increase the realism of the SimRobot simulator [31], widely used in the RoboCup SPL league. Additionally, CycleGan was also trained to solve this problem. Both of the proposed methods are compared to CycleGan.

To train FeatureGan, an unpaired training set was used. The dataset is composed by 3,579 images collected using the robot in a real environment, and 4,027 images collected from simulation. The learning rate for the generator was set to 0.00005 and the learning rate for the discriminator was set to 0.0001. A class based feature-Loss was used, with the selected classes being robots and goal posts, background, field, lines and shirt. α_j was set to 6 for the field class, to 4 for the robots and goal posts, 10 for the shirts, 8 for the lines and 0 for the background class, meaning that the network is free to produce unaligned backgrounds since there is no background information in the simulator as shown in the first row of Fig.6.1 (the background is defined by a single constant value). Three feature pyramid discriminators with receptive fields over the input image of 70 $\times$ 70 (local), 140 $\times$ 140 (medium) and 280 $\times$ 280 (global) were used to evaluate the quality of the generated images at different scales. λ_{i} was set to $\frac{1}{4}$ for the local discriminator, $\frac{2}{4}$ for the medium discriminator and $\frac{1}{4}$ for the global discriminator. The network was trained for 80 epochs with a constant learning rate,

and then for 150 epochs with linear decay at a resolution of 512 × 512 for three days in a Nvidia RTX 2080 ti.

The standard version of CycleGan, as presented by Zhu et al. [71] and described in 3.6.2, was trained on the same dataset as FeatureGan at a resolution of 512 × 512. A discriminator with a receptive field of 140 × 140 was used. The same generator and learning parameters were also employed to ensure a fair comparison. The identity loss was set to zero to diminish the blur of the generated images.

Samples from the real training dataset Dom_A and the simulated training dataset Dom_B are shown in Fig. 6.1. In these images, the simulation-to-reality gap is apparent, most notably in terms of illumination and in the fine detail of the textures.

Finally, the Cascade Refinement Network (CRN) based on a supervised approach was trained by following the methodology proposed in Sec. 4. The database consists of 2000 pairs of aligned $(real, segmented)$ images of size 512×512 which were obtained after performing domain randomization using 125 images of robots and 56 images of balls. The learning rate was set to a constant value of 0.0001.

Randomly selected pairs of simulated images B from the test dataset and the corresponding realistic images $\hat{A}$ generated by FeatureGan, the Cascade Refinement Network (CRN) and CycleGan [71] are shown in Fig. 6.2. A video of realistic images being generated in real time on the SimRobot simulator using FeatureGan and the different detectors being used over the generated image can be found at the following link: https://youtu.be/faJifSb2c1E.

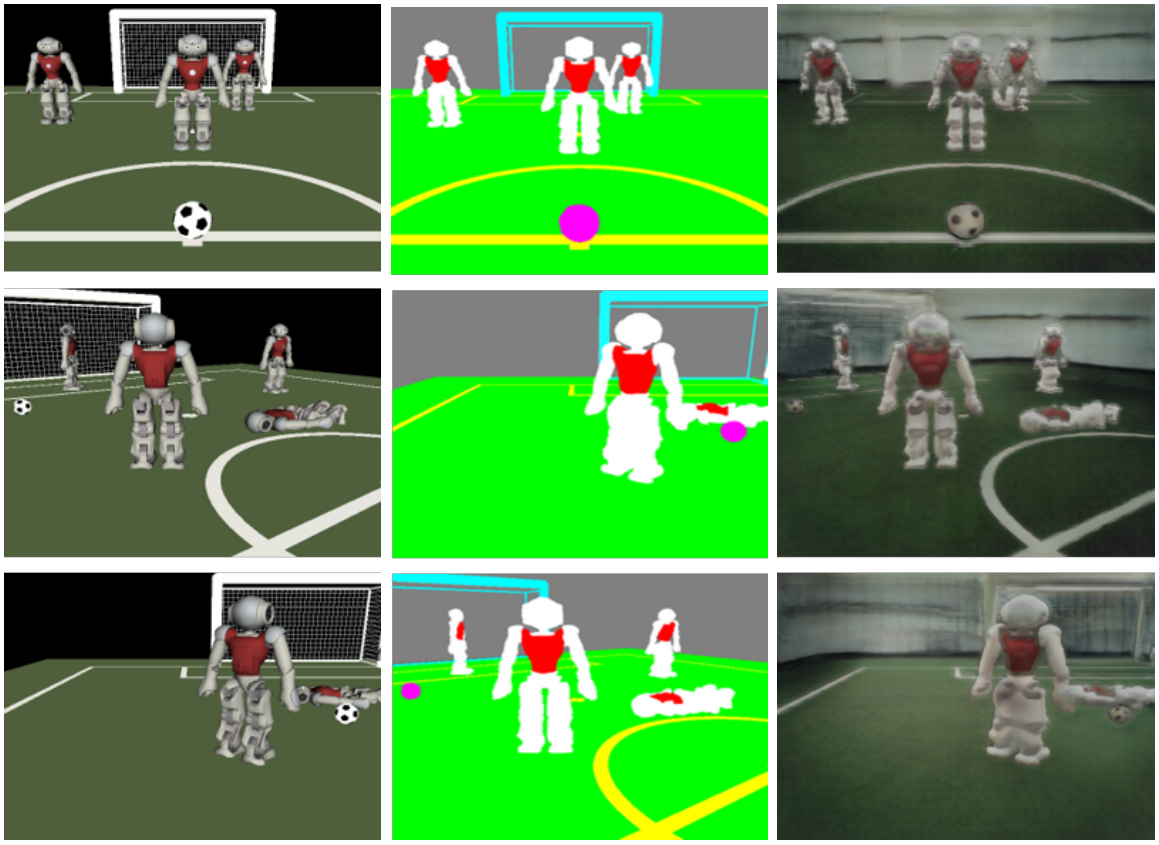
The SimRobot simulator achieved a framerate of 17 frames per second when using the generators of CycleGan and FeatureGan while a framerate of 15 frames per second was achieved using the CRN generator. All times were measured on a machine with a core i7-8700K and an RTX 2080 ti video card. This is well within the constraints of real time operation.

Figure 6.2: Input and aligned samples generated by various methodologies.

6.1.1 Visual quality analysis of CRN

From these images, it is apparent that the CRN method results in images of lesser texture and lighting quality than those produced by CycleGan and FeatureGan. This comes as a result of the chosen loss function and the use of a traditional supervised approach to go from Dom_B to Dom_A. Given an aligned pair of images (A, B) the network is trained by feeding B to the generator G which outputs a generated image $\hat{A}$. Both $\hat{A}$ and A are fed to the feature-extraction network and the resulting the loss is then calculated as the L1 norm (see Sec. 3.12) between the feature maps from generated image $\hat{A}$ and the ground truth image A as described in 3.15. However, this approach tends to produce blurry images [43]. This can be explained because there is more than one plausible output $\hat{A}$ for a given input B. As shown in Fig. 6.3, most of the information of the rendered simulated image is lost after being segmented to form the input image B. Then there is not enough information in B to determine the rotation of the robots (front-facing or back-facing) or to determine what texture to assign to the background in the generated image $\hat{A}$. There are then multiple possible solutions for $\hat{A}$ from a single input image B and the network is unable to determine which one of the solutions is correct.

The penalization of a network trained using a L1-Loss function will be minimized when all these possible solutions are averaged which results in blur in the predicted image $\hat{A}$ [28]. In other words, the network is trying to fit all plausible outputs for a given input B into a single image $\hat{A}$ by combining all the plausible solutions into one. It is important to note that by using a feature loss the network, which is invariant to changes in color and illumination, tries



(a) (b) (c)

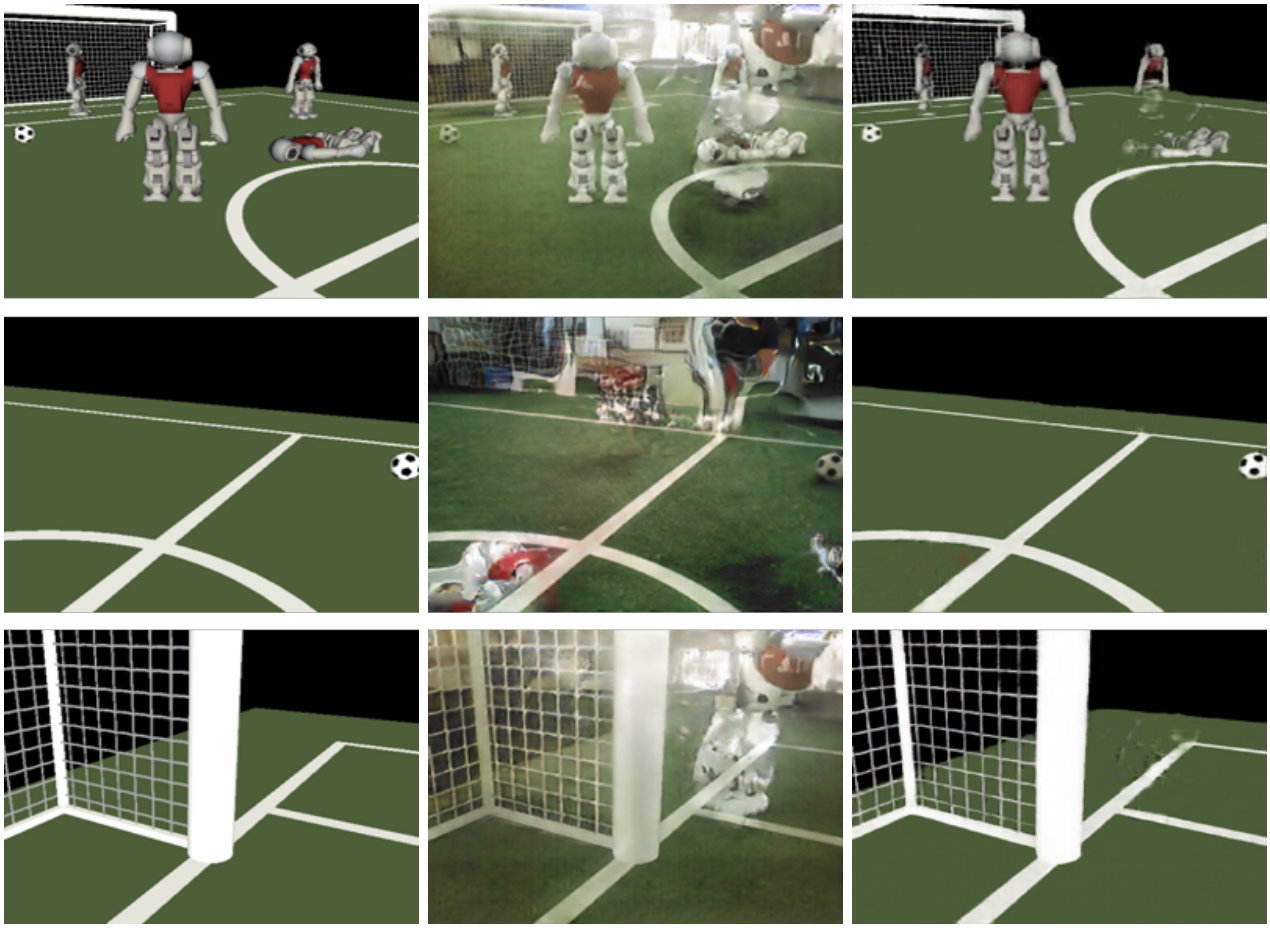
Figure 6.3: a) Simulated rendered image, b) aligned segmented image B, c) aligned generated image $\hat{A}$.

to match high level semantic features rather than specific pixels. Consequently, the network is not penalized for producing images with illuminations and colors which differ from the ground truth, which would produce additional blurring in the generated image. However, blur is still be present given that the feature extraction network is not invariant to changes in rotation or position and the input image B does not give enough information to the generator G to estimate the pose of the objects in the scene.

6.1.2 Visual quality analysis of CycleGan

CycleGan achieves some very crisp images with detailed textures in all the objects, a realistic background and realistic lighting conditions. However, artifacts (such as the one present in the second row of Fig. 6.2 were quite common both during training and inference over the test dataset. Images of the input rendered images B, generated realistic images $\hat{A}$ which present major artifacts and the reconstructed images $\hat{B}$ are presented in Fig. 6.4.

The artifacts appear to be a problem that derives from the CycleGan architecture when applied to the specific problem of the simulation-to-reality image translation task. As shown in Sec. 3.6.2, CycleGan operates with two cycles. The first cycle works by generating a realistic image $\hat{A}$ from an input image B such that $\hat{A} = G_B(B)$. It then generates a second



(a) (b) (c)

Figure 6.4: a) rendered image from simulation B, b) aligned images $\hat{A}$ generated by G_B, c) aligned reconstructed images $\hat{B}$ generated by G_A.

image $\hat{B}$ such that $\hat{B} = G_A(\hat{A})$. The loss for a single cycle is defined as the sum of the generator loss estimated by a discriminator over $\hat{A}$ and the cycle-Loss which corresponds to the L1-Loss between the reconstructed image $\hat{B}$ and the input image B and is used to maintain the alignment between B, $\hat{A}$ and $\hat{B}$. The second cycle works in a similar fashion but from A to $\hat{A}$ instead of from B to $\hat{B}$.

The CycleGan methodology works very well for high variance, complex environments, with several successful test cases shown in the original paper. However, this methodology does not translate well for simpler environments. Indeed, the artifacts found in Fig. 6.4 are the result of training CycleGan over a dataset of samples collected from a low variance, highly structured environments such as the SimRobot simulated environment [31]. If the generator G_A has learned a very accurate representation of the simulated scene, which is easy given the domain's low complexity and low variance, then the generator G_A can learn to generate an image $\hat{B}$ without artifacts in the simulated domain from an input realistic image $\hat{A}$ with artifacts. This is possible if the generator G_A is able to accurately detect the artifacts in $\hat{A}$ and then eliminate them in the output recreated image $\hat{B}$ by interpolating from known scene features. It follows that the generator G_A is not penalized by the cycle-Loss for generating

artifacts in the $\hat{A}$ image, since the generator G_B learns to delete such artifacts in $\hat{B}$. Then, there is nothing preventing CycleGan from falling into a mode collapse. This translates into the generator G_A outputting images that are most plausible to the discriminator, minimizing the generator loss. Then the optimal solution for CycleGan is to generate an image $\hat{A}$ that minimizes the generator loss estimated by the discriminator (for example by adding objects to the scene that are not present in the input image) and then the generator G_B can detect these artifacts and delete them from the recreated image $\hat{B}$ to replicate the input image B almost perfectly. By doing so, both the generator loss estimated by the discriminator as well as the cycle-Loss are minimized and by doing so the methodology encourages artifacts. This phenomena is shown in the images presented in Fig. 6.4, which are real results obtained from images in the test dataset and models resulting from later iterations of our training process. This artifacts translate in false positives for most heuristic based detectors and render the simulator basically unusable since it violates observation 1. Given this, it is apparent that traditional methods are not well suited for generating realistic simulators.

6.1.3 Visual quality analysis of FeatureGan

Finally, FeatureGan achieves an arguably superior image quality than CycleGan. One of the most salient difference between both corresponds to the illumination quality. In terms of global illumination, the results achieved by FeatureGan in Fig. 6.2 closely resemble those of the real images shown in Fig. 6.1 b), with several shadows and spotlights across the image as seen in see Fig. 6.5 a). Contact shadows are also present in the correct places. On the other hand, CycleGan produces results that are almost uniformly lit with very few shadows in general and sparse contact shadows (see Fig. 6.5 b)), which is not consistent with the several punctual lighting sources in the real world. This difference in the lighting quality can be attributed to two factors. In the case of CycleGan, the discriminator has a fixed receptive field of 140 $\times$ 140 which means that the discriminator has only access to spatially local information. Thus, since each observation of the discriminator is classified independently from the others, the GAN-Loss provided by the discriminator is minimized if the generator produces the most plausible texture that still guarantees alignment between the generated and input image. This results in the same texture being repeated whenever is possible, which gives the impression of uniform lighting in the scene. FeatureGan fixes this problem by using discriminators at three different scales. The global discriminator in particular has a receptive field of size 280 $\times$ 280 over the RGB image which provides spatially spread features. This allows the discriminator to check the global consistency of the lighting in the image, which in turn improves the results significantly. A second important factor to generate realistic images is the pyramid discriminator. Given that most objects in the scene (lines, goal posts, robots and balls) share somewhat similar textures at the local scale (white without much relief) and since not all of these objects cast shadows (lines don't) then it becomes fundamental for the discriminator to have access to semantically rich features, in particular to the object class. This information is provided by the features present in the compressed Feature Pyramid Tensor which means that the discriminator can accurately estimate which objects should cast contact shadows.

FeatureGan also presents only a minimal number of artifacts, which don't affect the performance of the respective detectors. Table 6.1 summarises the advantages of each methodology.

(a)

(b)

Figure 6.5: a) cropped FeatureGan images, b) cropped CycleGan images.

Table 6.1: Advantages and disadvantages of the respective methodologies.

Feature	FeatureGan	CycleGan	CRN
Image sharpness	High	High	Low
Light quality	High	Average	Low
Reliability	High	Low	Very high
Texture quality	High	High	Low

6.1.4 Training CNN based classifiers in Simulation

During our participation in the RoboCup SPL our team developed some of the first CNN based robot and ball detectors [10] [34]. To achieve real-time operation, these detectors have two main parts. In a first step, a heuristic based region proposal extracted a region of interest. Then in a second step, the region is classified by a CNN that is designed to achieve real time operation while running in a Nao humanoid robot. This procedure is used to detect the robots (teammates and opponents) as well as the ball in real-time. The detection of such objects is essential to play the game and to develop correct strategies. The architectures for the CNNs can be found in [34] and the design principles are further expanded on [10]. Training these models involves creating a manually annotated database which is time consuming.

To test the capabilities of FeatureGAN, a robot detector and a ball detector were trained using five different sets of images: (1) real images collected using a real robot in a real field, (2) images generated using FeautureGAN, taking as input the SimRobot images, (3) images

generated using the standard implementation of CycleGan [71], taking as input the SimRobot images (4) images generated using a Cascade Refinement Networks (CRN) trained using the methodology presented in Sec. 4, taking as input the SimRobot images and (5) simulated images rendered directly by SimRobot. It must be stressed that training the CRN requires aligned pairs of images (see Sec. 4), which is time consuming and involves manual labeling. For a fair comparison, all sets of images have the exact same size and no post processing over the images, such as image augmentation, was performed. All trained detectors (models) were tested using a test database composed purely of real samples collected using the real robot: 760 images for the ball detector and 960 images for the robot detector. By comparing the performance of the detectors in this real dataset the amount of simulation-reality gap can be estimated for each training dataset.

Table 6.2: Robot and ball models trained in different datasets and evaluated using real data.

Testing data type	Robot detector accuracy (%)	Ball detector accuracy (%)
Real images	90.7	98.2
FeatureGan images	89.3	95.4
CycleGan images	87.4	95.1
CRN images	83.7	95.3
SimRobot images	66.4	51.4

Table 6.2 shows the obtained results. Models trained using real samples achieve very high accuracy. Models trained using images generated by FeatureGan result in metrics that are similar to those achieved by the models trained with real data. CRN also shows a good performance while detecting the ball, but it decreases for the robot detection case. CycleGan has a slightly lower performance than our method, even while producing numerous artifacts in our tests. This is simply explained by the nature of the artifacts, which are a result of a bias in the generator network and by consequence have very little diversity. In practice, these samples are almost uni-modal and thus the classifier network can learn to classify the artifacts as outliers. In contrast, models trained using samples collected from the classical SimRobot simulator and tested with real data achieve close to random results for the ball classifier model while the robot classifier model achieves 24.3% less accuracy than the corresponding model trained using real samples. From these results, it is clear that FeatureGAN, CRN and CycleGan are bridging the gap between reality and simulation, and that best results are obtained by FeatureGAN. The use of these methods allows to fully train algorithms in simulation, which can then be deployed to a real environment with only a marginal loss in performance.

6.1.5 Testing CNN classifiers in Simulation

Robot simulators can serve as tools to evaluate the performance of robotics algorithms by offering a testing environment where they can run under real conditions. An evaluation for several simulation environments of how closely the metrics obtained for two detectors match the metrics obtained from reality is performed. To analyze this, robot and ball detectors were trained using images of the real-world, and then evaluated in five different environments: (1)

real images collected using a real robot in a real field, (2) images generated using Feauture-GAN, (3) images generated using the CycleGan, (4) images generated using CRN, and (5) simulated images rendered directly by SimRobot. Table 6.3 show the obtained results. It can be observed that simulated environments generated using FeatureGan and CRN are very good approximation of the real-world conditions, allowing to close the visual simulation-to-reality gap. The main advantage of FeatureGAN over CRN is the use of unpaired, unlabeled images to train the network. In the case of CycleGan, the robot detector tested using the generated images presented a notable decrease of 14.2% in accuracy, when compared to its accuracy in a real environment. This is a direct consequence of the artifacts produced by the generator, which translate in a large number of false positives for the Robot classifier. This resulted in false positives in the obstacle map and inaccurate path planning and behaviors. Given this, a CycleGan based simulation environment is not a good candidate for a simulation environment. Finally, testing in the classic rendered SimRobot environment resulted in sub-optimal performance for the robot model given the large simulation-reality-gap between the real and simulated domains. Surprisingly, the ball model achieved good performance in all testing environments. This can be attributed to the simplicity of the problem and argue that the CNN based model has learned to generalize even to other domains.

Table 6.3: Robot and ball models trained with real data and evaluated in different datasets.

Testing data type	Robot detector accuracy (%)	Ball detector accuracy (%)
Real images images	90.7	98.2
FeatureGan images	95.2	98.7
CycleGan images	76.5	94.3
CRN images	95.7	98.8
SimRobot images	66.2	97.0

Chapter 7

Results in CityScapes

7.1 Simulation to Reality Image Translation in CityScapes

This section examines if using FeatureGan can improve the graphics quality of video-games. In particular, it examines if by training the model to perform image-to-image translation from the rendered simulated domain to the real domain could result in more realistic looking images than those provided by the rendering engine. In particular, FeatureGan is trained using images from Grand Theft Auto 5 (GTA) [47] as the input domain Dom_B and images from CityScapes Cordts et al. [9] as the target domain Dom_A. Grand Theft Auto is an open-world game which simulates a real city environment with great accuracy. The world is populated by several agents such as driving cars, trucks, pedestrians, cyclist and others. The static elements of the world, such as building, bridges and terrain have good geometrical complexity and great variety. CityScapes on the other hand is a dataset composed of real samples taken from several cities. Importantly, both datasets have available and compatible ground truths in the form of semantic maps with 32 classes. Given the number of different objects, different textures and extremely complex lighting conditions, this problem is orders of magnitude more challenging than the simulation-to-reality problem for the SimRobot simulator that was developed in Sec. 6.

FeatureGan was trained using 24.807 images from the Grand Theft Auto 5 database as well as 23.417 samples of the Cityscapes dataset. Randomly selected samples from both datasets can be seen in Fig. 7.1.

(a) (b)

Figure 7.1: a) simulation training samples, b) real training samples.

Since video-games are simulations, the ground truth class information of all the relevant objects in the input image B can be easily obtained. The classes road, sidewalk, person, rider, car, truck, bus, train, motorcycle, bicycle, building, wall, fence, pole, traffic sign, traffic light, vegetation, terrain and sky were each assigned to a one-hot encoded vector which was then concatenated to the input image B. A class based feature-Loss was used to ensure alignment between B and $\hat{A}$. The classes terrain, sky, pole, traffic light and traffic sign which were more prone to artifacts were assigned an α_j of 2 while the rest of the classes were assigned an α_j of 6 to encourage domain transfer. The GAN-Loss was estimated using two pyramid discriminators with receptive fields over the input tensor of 70 $\times$ 70 (medium) and 140 $\times$ 140 (global). As proposed in Sec. 5.5.2 the input image was not used to generate IT. This prevents the training from collapsing. λ_{i} was set to $\frac{2}{3}$ for the medium discriminator and to $\frac{1}{3}$ for the global discriminator. A label-Loss was used to further encourage the alignment between B and $\hat{A}$ and to preserve small features in the input image. The U-net generator was trained for 10 epochs with an initial learning rate of 0.0001 and with linear decay at a resolution of 256 $\times$ 256. The learning rate for the discriminator was set to $\times$ 1.5 the learning rate of the generator during the complete training process.

Randomly selected simulated images B from the test dataset alongside the corresponding realistic images $\hat{A}$ generated by FeatureGan are shown in Fig. 7.2. A video can also be found at the following link: https://youtu.be/rR6oCqOwZrw.

(a)

(b)

Figure 7.2: (a) input images, (b) aligned images generated with FeatureGan.

7.1.1 Training Semantic segmentation models in simulation

Semantic segmentation models are a common choice when detecting the precise shape of the objects in the scene is needed. This has relevant applications for autonomous systems such as robots or self-driving vehicles. One of the most popular semantic segmentation models is DeepLab [7]. In this section, DeepLabv3+ with a mobileNet [22] backbone and a learning rate of 0.1 is trained using three different sets of images: (1) real images of a city from the CityScapes database, (2) realistic images generated by FeatureGan and (3) rendered images from the GTA database. All the datasets have 3000 samples The models were then evaluated on a the CityScapes test dataset, composed of real samples. Fig. 7.3 shows the

results of segmenting real samples of CityScapes with models trained with GTA samples and FeatureGan samples.

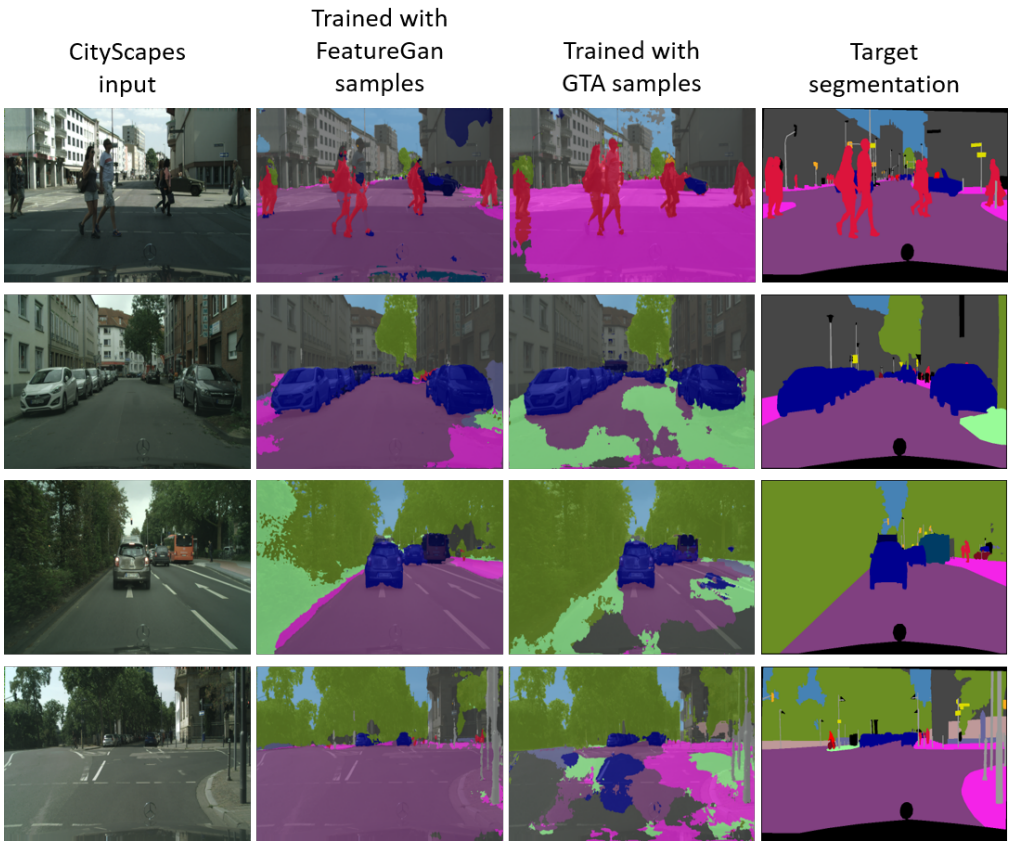


Figure 7.3: Segmentation results of DeepLabv3+ trained with different simulated datasets and applied to real samples.

Table 7.1: Semantic segmentation models trained in different datasets and tested using real data.

Training data type	Pixel accuracy (%)
Real images images	91.4
FeatureGan images	83.7
Rendered images	69.5

Table 7.1 show the results of this test. DeepLabv3+ achieves excellent results at the semantic segmentation task when trained and tested using the CityScapes dataset. However, the same architecture trained using synthetic data shows from the GTA dataset shows a significant reduction in performance of 21.9 % when tested with real samples. Training with samples generated by FeatureGan brings this difference to only 7.7 %, a massive improvement which shows that FeatureGan is able to significantly reduce the gap between simulation and reality.

7.1.2 Testing Semantic segmentation models in simulation

In this section, DeepLabv3+ is trained using real samples from the CityScapes dataset. The model is evaluated with three different sets of images: (1) real images of a city from the CityScapes database, (2) realistic images generated by FeatureGan and (3) rendered images from the GTA database. Fig. 7.4 shows some of the segmented simulated samples.

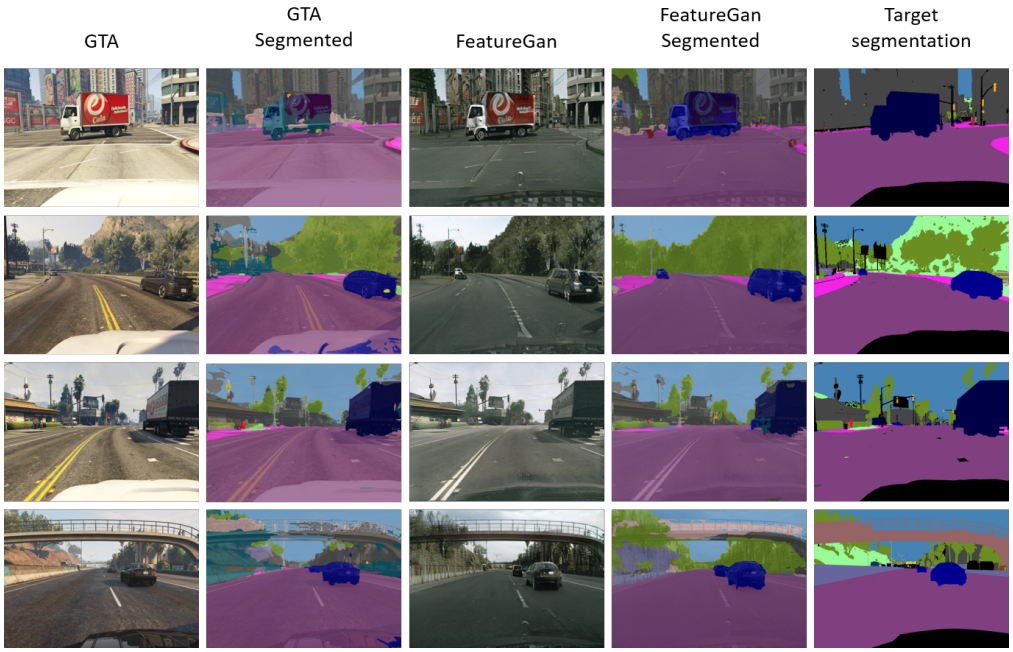


Figure 7.4: Segmentation results of DeepLabv3+ trained with real samples applied to simulated samples.

Table 7.2: Semantic segmentation model trained with real samples and tested in different datasets.

Testing data type	Pixel accuracy (%)
Real images images	91.4
FeatureGan images	84.0
Rendered images	81.0

Table 7.1 show the obtained results. Surprisingly, a model trained using real data has similar performance in the simulated data from the GTA database as well as in the generated realistic samples. A similar result was achieved in Sec. 6.1.5. A probable explanation is that the variance of the real dataset is enough to encapsulate the samples from the simulated dataset, meaning that a model trained with real data can extrapolate to simulated domains.

7.1.3 Ablation study

Importance of the Feature Pyramid Discriminator

To evaluate the importance of the Feature Pyramid Discriminator, FeatureGan was trained using a traditional PatchGan discriminator [27]. Fig. 7.5 shows the total feature-Loss and the GAN-Loss through the first three epochs of the training process. It is apparent that the training quickly collapses at around 1.6 epochs with the generator learning the identity function while the discriminator constantly outputs a value of close to 0.8 for the generated image $\hat{A}$. Interestingly, a network trained using the standard implementation of the Feature Pyramid Discriminator also collapses. It is only when using the guidelines proposed in Sec. 5.5.2 and deleting the input-image from IT that the training stabilizes and the generator actually learns to map from Dom_B to Dom_A. It appears that matching features rather than matching pixels is paramount in preventing the training process from collapsing.

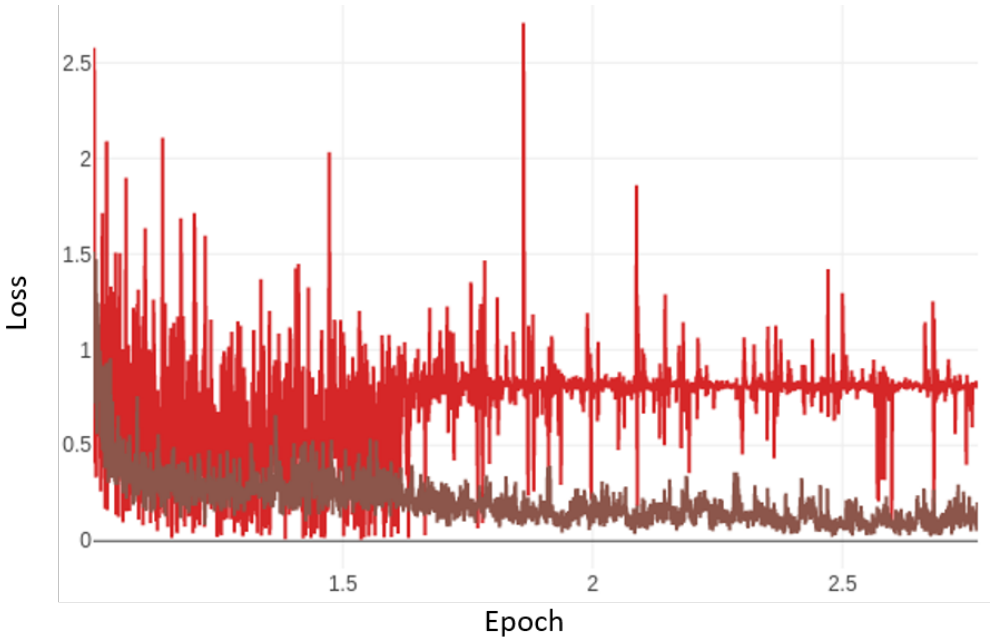


Figure 7.5: red) Gan-Loss, brown) total feature-Loss.

Importance of the feature-Loss and label-Loss

The importance of the feature-Loss in generating aligned images was evaluated by training FeatureGan without using the feature-Loss. A dataset of realistic samples was created with the resulting generator and then used to train the Deeplabv3+ semantic segmentation model. Similarly, a version of FeatureGan was trained without using the label-Loss and the resulting generator was used to produce a database with which a Deeplabv3+ model was trained. The performance of these semantic segmentation models was compared to the performance of DeepLabv3+ trained using samples produced by a generator resulting from the classic implementation of FeatureGan which was developed in Sec. 7.1.1. The semantic segmentation models were then tested with a dataset composed of real annotated samples. Results are

shown in tab. 7.3. From this, it is clear that the feature-Loss is essential to achieve proper alignment between $\hat{A}$ and B. However, the label-Loss doesn't seem to provide any benefit to the overall performance of the generator since training the network without it results in a variation in the accuracy of the semantic segmentation model of less than 1% which indicates that the class based feature-Loss is enough to ensure alignment.

Table 7.3: Semantic segmentation models trained in datasets produced by different implementations of FeatureGan and tested using real data.

FeatureGan losses	Pixel accuracy (%)
Standard implementation	83.5
Without feature-Loss	76.2
Without label-Loss	84.3

Chapter 8

Image-to-image translation in real domains

8.1 FeatureGan for real-to-real image translation

In previous chapters it was shown that FeatureGan is able to perform image to image translation in domains where ground truth data is available in the form of labels. This chapter shows that FeatureGan can also be used to perform image to image translation from one real domain to another without the need of ground truth information and discusses the implications.

8.1.1 Horse to zebra image translation task.

FeatureGAN is used to successfully train a generator to solve the horse-to-zebra transfiguration task solved in CycleGan [71]. Since no class information is available the feature-Loss was calculated over the whole image as described in Eq. 5.1 rather than by using the per-class feature-loss described in Eq. 5.5. The training database was composed of 1.067 images of horses as the input domain Dom_B and 1.334 images of zebras as the target domain Dom_A. The Resnet generator reviewed in Sec. 3.6.1 was used to transform the images from Dom_B to Dom_A. CycleGan was also trained using the same database and using the same generator architecture. Both networks were trained for 80 epochs with a constant learning rate of 0.00005 and for 80 additional epochs with decreasing learning rate.

Results obtained using FeatureGan and CycleGan are presented in Fig. 8.1 and samples of the features from the compressed Feature Pyramid Tensor are shown in Fig. 8.2.

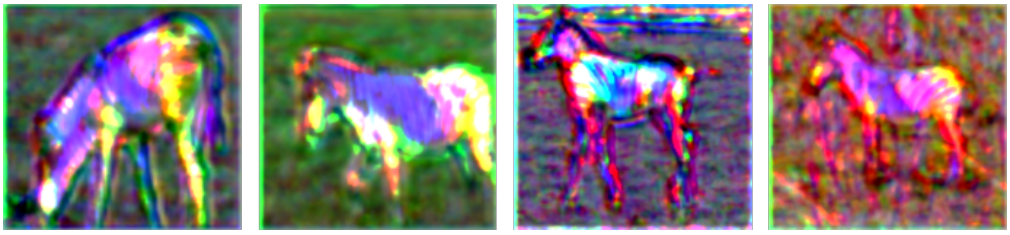
(a)

(b)

(c)

Figure 8.1: (a) input images, (b) aligned images generated with FeatureGan. (c) aligned images generated with CycleGan.

(a)



(b)

Figure 8.2: (a) input images, (b) compressed FPT tensor with N = 3.

Both methods achieve results of similar visual quality which indicates that FeatureGan is not limited to the simulation-to-reality image translation task but can also be used to solve classic problems. A video generated using FeatureGan which depicts the horse to zebra translation task is shown at the following link: https://youtu.be/KoY6hfojSQM

8.1.2 Horse to elephant image translation task.

A different problem is addressed: the horse-to-elephant image translation task. This problem is more complex than horse-to-zebra since the system needs to learn to change the shape of the relevant objects in addition to the textures. Indeed, while horses and zebras are geometrically very similar, there is no clear way of mapping the geometry of a horse to an elephant. Both FeatureGan and CycleGan are trained using the same database consisting of 2.000 images of horses and 3.634 images of elephants for 80 epochs with a constant learning rate of 0.00005 and 80 epochs with decreasing learning rate. A Resnet generator was chosen to perform the image translation.

Results obtained with FeatureGan and CycleGan are presented in Fig. 8.3 b) and c) respectively. Additionally, a video generated using FeatureGan which depicts the horse-to-elephant translation task is shown at the following link: https://youtu.be/9CAol4XoN4k. Samples of the features from the compressed Feature Pyramid Tensor are shown in Fig. 8.4.

From Fig. 8.3 b) it is clear that FeatureGan is capable of solving the horse-to-elephant task, given that the method produces visually realistic images. This is a problem that CycleGan cannot solve since according to the original paper, one of the main constraints of CycleGan [71] is that the objects involved in the transfiguration task must share similar geometries. This is reinforced by the results achieved with CycleGan, which are shown in Fig. 8.3 c). This is once again the result of the CycleGan architecture, in particular of the cycle loss.

Let's examine an image translation task that requires to go from a domain Dom_B to a domain Dom_A where the domains Dom_A and Dom_B have drastically different geometries, such as the case of the horse to elephant image translation task. Given a relevant object belonging to the Dom_B domain in the input image B, to successfully generate an image $\hat{A}$ in the Dom_A domain the geometry of the object needs to be modified. This means that the shape of the original object in B is lost in the generated image $\hat{A}$. However, that information is essential to generate the reconstructed image $\hat{B}$ and minimize the cycle loss which measures the L1 norm between B and $\hat{B}$. Given this, the problem has no evident solution for CycleGan since if the weight of the cycle loss is high then the network will learn to preserve the geometry of the input image B across all the images in the input cycle. On the other hand, if the weight of the cycle loss is decreased then the generated images $\hat{A}$ and $\hat{B}$ will not be aligned to the input image B.

While FeatureGan also measures the loss between a generated image $\hat{A}$ and the input image B, it uses a radically different approach which allows the network to do mayor geometrical changes. Indeed, while in CycleGan each pixel of the image $\hat{B}$ is compared against each pixel of B using the L1 loss function, the same is not true for FeatureGan, where each pixel is first analyzed to estimate its relevance by the Feature-Extraction network and then only the relevant pixels of $\hat{A}$ are compared to the relevant pixels of B. This means that FeatureGan uses a difference of relevant features rather than a difference of pixels to estimate the alignment of both images. In intermediate layers of the network these features come in the form of local high-level features, such as the position of the eyes, legs, etc. Then, by ensuring that the position of these high level features is preserved from B to $\hat{A}$ and since the discriminator encourages the generator to create consistent images in the Dom_A domain, then

the result is that the generator is guided to produce images in the target domain that retain as much alignment as possible while doing the necessary geometrical changes to comply with the requirements of the discriminator.

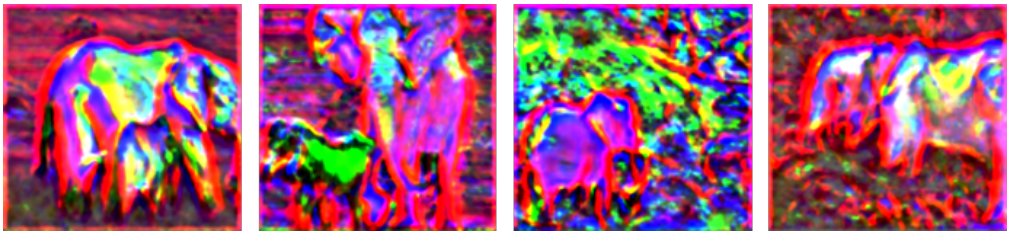
(a)

(b)

(c)

Figure 8.3: (a) input images, (b) aligned images generated with FeatureGan. (c) aligned images generated with CycleGan.

(a)



(b)

Figure 8.4: (a) input images, (b) compressed FPT tensor with N = 3.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

This work introduces two different methodologies that successfully bridge the simulation-to-reality gap by transforming rendered simulated images into realistic images using generative models. The first approach uses a cascade refinement neural network to generate realistic images from simulated images. The approach uses a supervised training regime which is very reliable but can only be used in simple environments. The second method is based on an unsupervised training regime and does not need aligned image pairs to be trained. It employs a combination of a feature-Loss and a GAN-Loss to train a generative neural network model which is then used to generate realistic images from simulated images. It can be applied to a much larger scope of problems than the first methodology, however the training might be unstable and can generate artifacts in the image.

The hypothesis of this thesis was validated in an experimental setup which proved that realistic images, which are aligned with the rendered images provided by the simulator, can be generated in real time and that vision algorithms developed in realistic simulated environments can be successfully deployed to real environments with only marginal loses in performance. Both methodologies can be easily replicated and trained in consumer-grade hardware at high resolutions. Furthermore, since these realistic simulated environments provide ground truth information they are ideal to test algorithms, quickly collect relevant metrics and generate databases. This allows to reduce or in the case of FeatureGan to completely eliminate the need of manual image labeling.

9.2 Future work

While the results of the experiments performed for this thesis validated the proposed approaches, several key areas of potential improvement were found.

- An important area in which this work can be improved is by encoding the domain information in the affine parameters of the normalization function to achieve multi-domain image to image translation. This could allow the generator to create samples

in multiple domains which could be very useful to generate a variety of simulated environments according to the experimental needs.

- The feature-Loss function needs to be revisited to perform a better estimation of the alignment between the input and output image. Feature maps from the earlier layers tend to have too much domain information which hinders the training process, while later layers lack the spatial resolution to correctly align the input and output image.
- An important area of potential improvement is sample variety. It is apparent that the variance of the realistic generated dataset is lower than that of the target real dataset. To further increase the performance of models trained in simulation, a method to match the variance of the target dataset is needed.
- Achieving temporal consistency between samples generated in simulation is a fundamental feature that must be addressed by implementing a cost function for the optical flow between consecutive frames.
- Create a generator architecture specifically designed to work with labels and which encourages the generation of aligned images.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[2] Jimmy Ba, Jamie Kiros, and Geoffrey Hinton. Layer normalization. 07 2016.

[3] Homanga Bharadhwaj, Zihan Wang, Yoshua Bengio, and Liam Paull. A data-efficient framework for training and sim-to-real transfer of navigation policies. *CoRR*, abs/1810.04871, 2018. URL `http://arxiv.org/abs/1810.04871`.

[4] Jan Blumenkamp, Andreas Baude, and Tim Laue. Closing the reality gap with unsupervised sim-to-real image translation for semantic segmentation in robot soccer, 11 2019.

[5] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, Sergey Levine, and Vincent Vanhoucke. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. *CoRR*, abs/1709.07857, 2017. URL `http://arxiv.org/abs/1709.07857`.

[6] Joan Bruna, Pablo Sprechmann, and Yann Lecun. Super-resolution with deep convolutional sufficient statistics. 11 2015.

[7] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *CoRR*, abs/1802.02611, 2018. URL `http://arxiv.org/abs/1802.02611`.

[8] Qifeng Chen and Vladlen Koltun. Photographic image synthesis with cascaded refinement networks. *CoRR*, abs/1707.09405, 2017. URL `http://arxiv.org/abs/1707.09405`.

[9] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016. URL `http://arxiv.org/abs/1604.01685`.

[10] Nicolás Cruz, Kenzo Lobos-Tsunekawa, and Javier Ruiz-del Solar. Using convolutional neural networks in robots with limited computational resources: Detecting nao robots while playing soccer. In Hidehisa Akiyama, Oliver Obst, Claude Sammut, and Flavio Tonidandel, editors, *RoboCup 2017: Robot World Cup XXI*, pages 19–30, Cham, 2018. Springer International Publishing. ISBN 978-3-030-00308-1.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[12] dlss. Deep learning super-sampling. `https://www.nvidia.com/en-us/geforce/news/nvidia-dlss-2-0-a-big-leap-in-ai-rendering/`, 2020. Accessed: 2020-01-30.

[13] Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. *CoRR*, abs/1602.02644, 2016. URL `http://arxiv.org/abs/1602.02644`.

[14] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015. URL `http://arxiv.org/abs/1508.06576`.

[15] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. JMLR Workshop and Conference Proceedings. URL `http://proceedings.mlr.press/v15/glorot11a.html`.

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

[18] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. URL `http://arxiv.org/abs/1703.06870`.

[19] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500, 2017. URL `http://arxiv.org/abs/1706.08500`.

[20] Geoffrey E. Hinton, James L. Mcclelland, and David E. Rumelhart. Distributed represen-

tations. In David E. Rumelhart and James L. Mcclelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 77–109. MIT Press, Cambridge, MA, 1986.

[21] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A. Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. *CoRR*, abs/1711.03213, 2017. URL `http://arxiv.org/abs/1711.03213`.

[22] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL `http://arxiv.org/abs/1704.04861`.

[23] Xun Huang and Serge J. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *CoRR*, abs/1703.06868, 2017. URL `http://arxiv.org/abs/1703.06868`.

[24] Xun Huang, Ming-Yu Liu, Serge J. Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. *CoRR*, abs/1804.04732, 2018. URL `http://arxiv.org/abs/1804.04732`.

[25] Tadanobu Inoue, Subhajit Chaudhury, Giovanni De Magistris, and Sakyasingha Dasgupta. Transfer learning from synthetic to real images using variational autoencoders for robotic applications. *CoRR*, abs/1709.06762, 2017. URL `http://arxiv.org/abs/1709.06762`.

[26] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

[27] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016. URL `http://arxiv.org/abs/1611.07004`.

[28] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016. URL `http://arxiv.org/abs/1611.07004`.

[29] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016. URL `http://arxiv.org/abs/1603.08155`.

[30] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2014.

[31] Tim Laue, Kai Spiess, and Thomas Röfer. Simrobot – a general physical robot simulator and its application in robocup. volume 4020, pages 173–183, 07 2005. doi: 10.1007/11780519_16.

[32] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade (2nd ed.)*, volume 7700 of *Lecture Notes in Computer Science*, pages 9–48. Springer, 2012. ISBN 978-3-642-35288-1.

[33] F. Leiva, K. Lobos-Tsunekawa, and J. Ruiz-del-Solar. Collision avoidance for indoor service robots through multimodal deep reinforcement learning. *RoboCup Symposium*, 2019.

[34] Francisco Leiva, Nicolás Cruz, Ignacio Bugueño, and Javier Ruiz-del-Solar. Playing soccer without colors in the SPL: A convolutional neural network approach. *Lecture Notes in Computer Science*, 11374 (RoboCup Symposium), pp. 122-134., 2018.

[35] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network, 2013. URL `http://arxiv.org/abs/1312.4400`. cite arxiv:1312.4400Comment: 10 pages, 4 figures, for iclr2014.

[36] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. URL `http://arxiv.org/abs/1405.0312`.

[37] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016. URL `http://arxiv.org/abs/1612.03144`.

[38] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. *CoRR*, abs/1703.00848, 2017. URL `http://arxiv.org/abs/1703.00848`.

[39] K. Lobos-Tsunekawa, F. Leiva, and J. Ruiz-del-Solar. Visual navigation for biped humanoid robots using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 3(4):3247–3254, Oct 2018. ISSN 2377-3766. doi: 10.1109/LRA.2018.2851148.

[40] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard S. Zemel. Understanding the effective receptive field in deep convolutional neural networks. *CoRR*, abs/1701.04128, 2017. URL `http://arxiv.org/abs/1701.04128`.

[41] openAI. generative models. `https://openai.com/blog/generative-models/`, 2020. Accessed: 2020-01-30.

[42] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[43] A. Pandey and D. Wang. On adversarial training and loss functions for speech enhancement. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5414–5418, 2018.

[44] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga,

Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[45] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *CoRR*, abs/1710.06537, 2017. URL http://arxiv.org/abs/1710.06537.

[46] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015. URL http://arxiv.org/abs/1511.06434. cite arxiv:1511.06434Comment: Under review as a conference paper at ICLR 2016.

[47] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conference on Computer Vision (ECCV)*, volume 9906 of *LNCS*, pages 102–118. Springer International Publishing, 2016.

[48] RoboCup. Robocup standard platform league official website. https://spl.robocup.org/, 2020. Accessed: 2020-01-30.

[49] Thomas Röfer, Tim Laue, Andreas Baude, Jan Blumenkamp, Gerrit Felsch, Jan Fiedler, Arne Hasselbring, Tim Haß, Jan Oppermann, Philip Reichenberg, Nicole Schrader, and Dennis Weiß. B-Human team report and code release 2019, 2019. Only available online: http://www.b-human.de/downloads/publications/2019/CodeRelease2019.pdf.

[50] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL http://arxiv.org/abs/1505.04597.

[51] Andrei A. Rusu, Matej Vecerík, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *CoRR*, abs/1610.04286, 2016. URL http://arxiv.org/abs/1610.04286.

[52] Fereshteh Sadeghi and Sergey Levine. (cad)$ˆ2$rl: Real single-image flight without a single real image. *CoRR*, abs/1611.04201, 2016. URL http://arxiv.org/abs/1611.04201.

[53] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016. URL http://arxiv.org/abs/1606.03498.

[54] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In S. Bengio, H. Wal-

lach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2483–2493. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7515-how-does-batch-normalization-help-optimization.pdf.

[55] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. *CoRR*, abs/1612.07828, 2016. URL http://arxiv.org/abs/1612.07828.

[56] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. *CoRR*, abs/1612.07828, 2016. URL http://arxiv.org/abs/1612.07828.

[57] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[58] Niko Sünderhauf, Oliver Brock, Walter J. Scheirer, Raia Hadsell, Dieter Fox, Jürgen Leitner, Ben Upcroft, Pieter Abbeel, Wolfram Burgard, Michael Milford, and Peter Corke. The limits and potentials of deep learning for robotics. *CoRR*, abs/1804.06557, 2018. URL http://arxiv.org/abs/1804.06557.

[59] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL http://arxiv.org/abs/1409.4842.

[60] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. URL http://arxiv.org/abs/1512.00567.

[61] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. *CoRR*, abs/1611.02200, 2016. URL http://arxiv.org/abs/1611.02200.

[62] Hoang Thanh-Tung, Truyen Tran, and Svetha Venkatesh. On catastrophic forgetting and mode collapse in generative adversarial networks. *CoRR*, abs/1807.04015, 2018. URL http://arxiv.org/abs/1807.04015.

[63] Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907, 2017. URL http://arxiv.org/abs/1703.06907.

[64] Joshua Tobin, Wojciech Zaremba, and Pieter Abbeel. Domain randomization and generative models for robotic grasping. *CoRR*, abs/1710.06425, 2017. URL http://arxiv.org/abs/1710.06425.

[65] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. *CoRR*, abs/1702.05464, 2017. URL http://arxiv.org/abs/1702.05464.

[66] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016. URL `http://arxiv.org/abs/1607.08022`.

[67] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. *CoRR*, abs/1711.11585, 2017. URL `http://arxiv.org/abs/1711.11585`.

[68] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *CoRR*, abs/1611.03530, 2016. URL `http://arxiv.org/abs/1611.03530`.

[69] Fangyi Zhang, Jürgen Leitner, Michael Milford, and Peter Corke. Sim-to-real transfer of visuo-motor policies for reaching in clutter: Domain randomization and adaptation with modular networks. *CoRR*, abs/1709.05746, 2017. URL `http://arxiv.org/abs/1709.05746`.

[70] Bolei Zhou, Aditya Khosla, Àgata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. *CoRR*, abs/1512.04150, 2015. URL `http://arxiv.org/abs/1512.04150`.

[71] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017. URL `http://arxiv.org/abs/1703.10593`.

[72] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A. Efros, Oliver Wang, and Eli Shechtman. Toward multimodal image-to-image translation. *CoRR*, abs/1711.11586, 2017. URL `http://arxiv.org/abs/1711.11586`.

Chapter 10

Appendix

Autonomous Robots manuscript No.
(will be inserted by the editor)

Deep Learning Applied to Humanoid Soccer Robotics: Playing Without Using Any Color Information

Nicolás Cruz · Francisco Leiva · Javier Ruiz-del-Solar

Received: date / Accepted: date

Abstract The goal of this paper is to describe a vision system for humanoid robot soccer players that does not use any color information, and whose object detectors are based on the use of convolutional neural networks. The main features of this system are the following: (i) real-time operation in computationally constrained humanoid robots, and (ii) the ability to detect the ball, the pose of the robot players, as well as the goals, lines and other key field features robustly. The proposed vision system is validated in the RoboCup Standard Platform League (SPL), where humanoid NAO robots are used. Tests are carried out under realistic and highly demanding game conditions, where very high performance is obtained: a robot detection accuracy of 94.90%, a ball detection accuracy of 97.10%, and a correct determination of the robot orientation 99.88% of the times when the observed robot is static, and 95.52% when the robot is moving.

Keywords Soccer Robotics · Deep Learning · Convolutional Neural Networks · Robot Detection · Ball Detection · Robot Orientation Determination

1 Introduction

Robotic soccer promotes robotics and artificial intelligence research by offering a formidable challenge: "By the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup" (RoboCup, 2020a). Soccer is a real-time, distributed decision-making problem, where players need to perceive and understand the environment, make collective decisions, and execute these decisions with the final objective of winning the match; i.e. scoring goals against the opponent team and avoiding goals from it.

The perception of the environment is one of the key abilities for playing soccer; without an adequate vision system it is not possible to determine robustly the position of the ball and the pose of the other players, to identify key field features (e.g. goals and field lines) and to self-localize, which are essential abilities to play properly. Given that the soccer environment has a predefined physical setup, and that robots used in RoboCup soccer leagues normally have limited processing capabilities, most of the current vision systems used in soccer robotics are based on the use of color information. However, the use of color information has some drawbacks such as (i) the need for calibration of the camera and tuning of the color-segmentation' parameters to achieve a properly color segmented image and/or the calibration of perception algorithms employed due to the fact that color perception depends on the environmental illumination, and (ii) the need of a soccer field with predefined colors (e.g. lines need to be white, field/carpet needs to be green).

Currently, there are different robot soccer leagues, which use different kinds of real or simulated mobile robots (RoboCup, 2020a). In this work we are interested in playing soccer with real humanoid robots. We choose to work in the RoboCup Standard Platform League (SPL) given that it uses a standard platform,

Nicolás Cruz
Department of Electrical Engineering, Universidad de Chile

Francisco Leiva · Javier Ruiz-del-Solar
Department of Electrical Engineering, Advanced Mining Technology Center, Universidad de Chile
Tel.: +56-2-2977 1000 & +56-2-2978 4207
Fax: +56-2-6720162
E-mail: jruizd@ing.uchile.cl

the NAO humanoid robot (RoboCup, 2020b), which allows to compare and share developments with other teams, and to focus on the cognitive aspects of the problem.

The RoboCup SPL started in 2008, and the first vision systems used by the competing teams were based on those developed in the former Four-Legged League, which used SONY-AIBO four-legged robots as its standard platform. In both leagues, the first generation of vision systems was based on the segmentation, detection, and analysis of colored objects of interest: the ball, lines, beacons, goals, players and the field/carpet. Year by year, the restriction of having colored objects in the field was relaxed: (i) the number of colored beacons (used for the robot's self-localization) was first reduced from six to four, then to two, and then beacons were removed in 2008, (ii) the goals were first colored and solid, then composed by three colored cylinders (goalposts and crossbar) and a white net, and finally composed by three white cylinders (goalposts and crossbar) and a white, gray or black net (since 2015), (iii) the ball used to be orange, and since 2016, black and white. However, still most of the teams use color information to detect some field features (the lines, goal posts and penalty marks), the players, and the ball.

Recently, Convolutional Neural Networks (CNNs) have been used for detecting the robots and/or the ball (Albani et al., 2017; Speck et al., 2017; Cruz et al., 2018; Javadi et al., 2018; Menashe et al., 2018; Gabel et al., 2019; Speck et al., 2019; Felbinger et al., 2019; Kukleva et al., 2019; Poppinga and Laue, 2019; Teimouri et al., 2019). Most of these CNN-based detectors require object proposals, which are currently obtained using color information of the field/carpet (green) and the lines (white). There have also been efforts to use end-to-end trained CNNs to detect all field's objects without relying on object proposals (Szemenyei and Estivill-Castro, 2019a,b). However, considering the limited processing capabilities of the NAO's CPUs (the NAO v4 and v5 models are powered with an ATOM Z530 1.6 GHz CPU), these vision systems are still unable to run in real-time while playing soccer, and, at the same time, to obtain the required performance for highly competitive matches.

Therefore, to the best of our knowledge, color-free vision systems have not been used in real robot soccer games, at least not in the SPL. Some of the main reasons underlying this are the following: (i) the challenge of achieving real-time operation when using limited computational resources, (ii) the problem of training deep detectors without having very large databases, which are difficult to create when real-world soccer conditions are taken into account, and (iii) the challenge of developing efficient and reliable color-free object proposal generators.

We believe that using color-free vision systems in soccer robotics is relevant, because this eliminates the constraint of having objects on the field with specific colors (e.g. the lines), and because it eliminates the need for calibration of the vision systems (before and/or during the games), making it possible to play soccer under variable lightning conditions (e.g. indoors near big windows or outdoors).

The goal of this paper is to propose a color-free vision system for humanoid soccer robotics, which will be validated in the SPL. The main features of this system are (i) real-time operation in humanoid robots (specifically in the NAO v5 robots that are part of the official platform for the SPL), and (ii) the ability to detect the ball position, the robots' pose, the lines, and key field features very robustly. In fact, as it will be shown in Section 5, the proposed ball, robots and robots' orientation detectors are highly performant; they achieve very high detection rates, measured under realistic RoboCup SPL game conditions.

To the best of our knowledge, the proposed system is the first color-free vision system for humanoid soccer robotics that is able to run in real-time, with a performance that allows its use in robotics world championships. It is very important to stress this point, because in images acquired under real-world conditions, the objects are much difficult to detect than in standard databases. For instance, ball perception is prone to have image blurring produced by the fast movement of the ball and the unstable walking of the robots. The proposed vision system was used by our team, UChileRT, in the RoboCup 2018 Word Competition, and its robot detector in the RoboCup 2017 Word Competition.

The main technical contributions of this paper are the following: (i) the proposal of a vision framework that combines concepts of deep learning and cascade classification to obtain, at the same time, high detection rates and fast processing, (ii) the use of a training methodology based on bootstrap and active learning, and (iii) the proposal of a method that is able to accurately determine the orientation of an opponent humanoid robot player by using a combination of heuristics and CNNs.

A preliminary version of this work was presented in Leiva et al. (2019). In this extended version a much deeper explanation of the proposed color-free vision system and its main modules is provided, as well as a better description of the design and training of the CNN-based detectors. The structure of all of the proposed CNN detectors is explicitly described, as well as new detection results in real soccer fields. In addition, in this extended

version the proposed framework is also validated in a different domain (detection of human soccer players in thermal images) to show its applicability beyond soccer robotics.

The paper is organized as follows: related work is presented in Section 2; the proposed color-free vision system is described in Section 3. Section 4 describes the design and training of the proposed CNN-based detectors. The experimental validation and results are shown in Section 5; and finally, conclusions and suggestions for future work are presented in Section 6.

2 Related work

Since 2016, CNNs have been used for detecting the robots and/or the ball in the SPL and humanoid RoboCup leagues (Albani et al., 2017; Speck et al., 2017; Cruz et al., 2018; Javadi et al., 2018; Menashe et al., 2018; Gabel et al., 2019; Speck et al., 2019; Felbinger et al., 2019; Kukleva et al., 2019; Poppinga and Laue, 2019; Teimouri et al., 2019).

In Albani et al. (2017), the first CNN-based robot detector for the SPL league was proposed. In this system, robot proposals are first computed by using color-segmentation based techniques, and then, a CNN is used for validating the robot detections. Different architectures with three, four, and five layers are explored. In the reported experiments, the 5-layer architecture was able to obtain 100% accuracy in the SPQR NAO image data set, also proposed in Albani et al. (2017). However, evaluating a detector using this dataset is different from evaluating it in real game conditions, which have much harder requirements. The detector was able to run at 11-19 fps on a NAO robot when all non-related processes (such as self-localization, decision-making, and body control) were disabled. Because of the latter, this detector could not be used to play soccer in real soccer games.

In Javadi et al. (2018), the performance of three well-known CNN architectures (namely LeNet, GoogLeNet, and SqueezeNet) was analyzed in the task of detecting humanoid robots. In this study, however, no real-world deployment was presented.

In Poppinga and Laue (2019), a proposal-free robot detector based on CNNs was presented. The proposed network has an adaptable architecture, it is multi-scale, and uses separable convolutional blocks (Howard et al., 2017). The authors also proposed a novel training procedure inspired in the generator-discriminator adversarial learning paradigm, which allowed training the networks using real and simulated images at the same time. The trained detectors were able to detect robots under realistic conditions, and the obtained detection time was 9.0 ms for a single image. In case that a similar approach is used to detect other objects (e.g. the ball), it is not clear that those detectors would be able to run simultaneously in real-time.

In Cruz et al. (2018), we presented a CNN-based robot detector, capable of operating in real-time. The system was based on the classification of color-based robot proposals generated by B-Human's robot perceptor (Röfer et al., 2017). This was modeled as a binary classification problem, where proposals could be labeled as robots or non-robots. The system processed robot proposals in ∼1 ms while playing soccer, with an average accuracy of ∼97%. Although this detector achieved a very high performance, it possessed some drawbacks. While the CNN classifier was robust to noise and variations of the illumination, the same did not apply to the color-based robot proposal generator. Adverse environmental conditions could lead the algorithm to produce an excessive amount of object hypotheses, or none at all. The second drawback derived from the CNN ∼1 ms inference time. While such a network is deployable on a NAO robot, it is much slower than alternative algorithms based on heuristics or shallow classifiers, and can be prohibitively slow when too many robot proposals are generated. In this paper we address both problems by changing the robot proposals generation approach, and by further reducing the inference times while maintaining the detection accuracy.

In Speck et al. (2017), the first CNN-based ball detector for the RoboCup humanoid league was proposed. The detector used two CNNs, which were able to obtain a localization probability distribution for the ball over the horizontal and vertical image axes, respectively. Several non-linearities were tested, with the soft-sign activation function generating the best results. Processing times in the robot platforms were not reported in that work, and the obtained accuracy was about 80%.

In Teimouri et al. (2019), a CNN-based ball detector for the humanoid league was presented. The proposed architecture is multi-scale and uses separable convolutional blocks (Howard et al., 2017). The detector is not color-free, because the ball proposals are generated considering the white and green patterns of the soccer field. The obtained performance of the detector is 70.9%, and it decreases with variable lighting conditions and blurred images.

In Menashe et al. (2018), ball detection using different machine learning methods is analyzed. The system considers several heuristic stages used for generating the ball proposals, and a final classification stage implemented using either a SVM or a CNN based classifier. The performance of both systems is analyzed, but

the analysis was focused on the transferability between different soccer environments.

In Felbinger et al. (2019), a genetic design approach for optimizing the hyper-parameters of a CNN designed to detect the ball is presented. The focus is not on the real-world deployment, but on the genetic based design of the network. Nevertheless, an average runtime of 8 ms was obtained in the NAO robots.

Some other authors have proposed CNN based ball detectors that requires a GPU for running in real-time (Gabel et al., 2019; Speck et al., 2019; Kukleva et al., 2019). Obviously, these detectors cannot be used in robots that just rely on CPU-based processing (such as the NAO robots).

In a different research line, some authors have proposed end-to-end trained CNNs to detect all field's objects without using object proposals (Szemenyei and Estivill-Castro, 2019a,b). In Szemenyei and Estivill-Castro (2019a), the use of two networks, one to perform semantic segmentation of the images, and a second one to propagate class labels between consecutive frames, is proposed. Authors reported that the fully neural vision pipeline runs at 6 frames per second, which from our point of view is not enough for playing soccer at a competitive level. In Szemenyei and Estivill-Castro (2019b), ROBO, a new CNN model inspired in the popular Tiny YOLO (Redmon and Farhadi, 2017), is proposed. ROBO is able to detect all relevant objects in the soccer field. The processing time of the different versions of the CNN, which consider different levels of pruning, range from 2.3 frames per second to 13 frames per second, which obtains a Mean Average Precision (mAP) of about 83%.

We believe that the limited processing capabilities of humanoid robots currently used in robotic soccer, are not sufficient to use end-to-end trained CNNs to reliably detect all field objects in real time while playing soccer.

3 Playing Soccer without Color Information

In this section we describe the proposed vision system. Section 3.1 broadly explains the general characteristics and functioning of the vision framework, while Sections 3.2 - 3.9 describe the operation of each of its main modules.

3.1 The General Framework

As already mentioned, the main feature of the proposed vision system is that it manages to detect the ball, the robot players, their orientations, and key features of the field without using any color information, i.e. the whole processing is performed using grayscale images rather than on a color segmented image. Removing the color segmentation step from the pipeline offers several advantages such as reduced operation times, reduced points of failure for the vision modules, easier pre-game calibration, and larger range of valid camera parameters since our object and field feature detectors are more resilient to changes in illumination than color-based approaches.

The key design components that allow the robust detection of all these objects in real-time (using robots with processing limitations) are the following: (i) custom-made object proposals generators for each kind of object, which are based on the characteristics of the soccer problem, (ii) CNN-based object detectors using a light CNN architecture specially designed for this application (Cruz et al., 2018), (iii) a cascade classification methodology inspired in Viola and Jones (2001), which implements the detection of some objects (e.g., the ball) using a two-stage classification cascade of CNN-based detectors, where the first stage discards, very quickly, non-objects that are very different from the objects being detected, and the second stage performs the final classification, and (iv) the use of the detection results of some object detectors for constraining the search of the others. In summary, we follow a pragmatic approach that combines classical algorithms widely used in robot vision with modern CNN-based classifiers.

The proposed vision framework is illustrated in Fig. 1. While the detection of lines and field features is done by using a set of rules and heuristics commonly employed in the SPL community (modules in yellow), the detection of the ball, the robot players and their orientation is done by means of object proposals (modules in green) and their subsequent classification using CNNs (modules in blue). The ball and robot orientation detectors are implemented as a two-stage cascade of classifiers.

3.2 High Contrast Regions Detection

Given the environmental conditions in which RoboCup soccer matches take place (soccer field and players' characteristics), an appropriate heuristic to speed up the process of finding the soccer ball and other players is to search for them in high contrast regions of the images. Accordingly, the grayscale input images are scanned using 16x16 pixels windows to find those regions. Any window laying outside the field boundary (determined using a priori knowledge of the field dimensions and the pose of the robot's camera) is automatically discarded. Windows containing body-parts

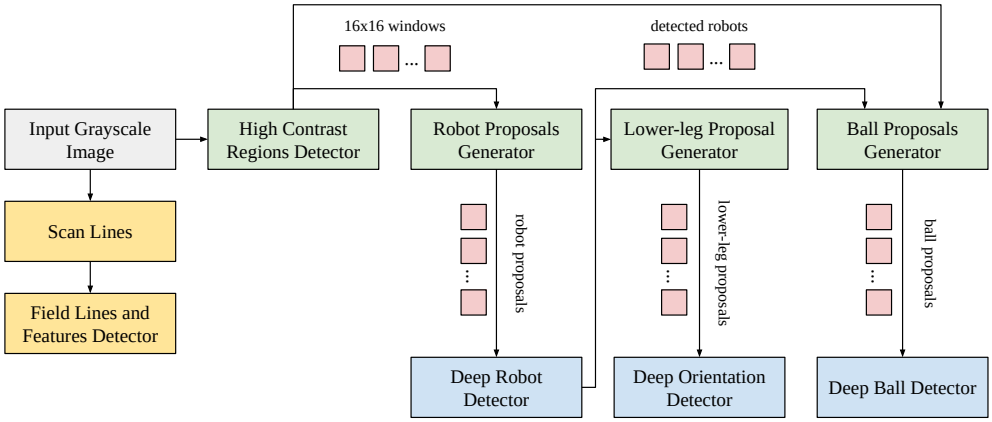


Fig. 1 Block diagram of the proposed vision system.

of the observer robot are also discarded. Over each of the remaining windows, a threshold for binarization is estimated using Otsu's method (Otsu, 1979). Only windows with a corresponding binarization threshold that is greater than a predefined value are considered to have high-contrast properties. Since the value utilized to select those windows may leave out image regions containing objects of interest, a dilation operation is applied on the selected windows. That is, all adjacent windows to any window considered to have high contrast properties, according to its binarization threshold, is also considered to have high contrast.

3.3 Robot Proposals Generator

The robot proposal generation applies vertical scan lines (y direction) over all the image's x-coordinates where high contrast regions were detected. The scan lines search for vertical abrupt contrast changes. Depending on the y coordinate of contrast changes found by the scan lines, a check is performed to see if enough of these detections have roughly the same y coordinate across the x-direction. If this is the case, the midpoints (in image coordinates) of all the sets of detections that fulfill this condition are considered to be the midpoints of the bottom segments of the bounding boxes containing the observed robot players. Then, by performing geometric sanity checks using a priori information of the other robot players (such as their height), the proposal generator provides a set of bounding boxes which may contain other robots' bodies. These sanity checks are similar to some of the rules used in Röfer et al. (2017), but adapted to be applied on a grayscale image. Moreover, all the rules that only rely on color information (such as checking for a player's jersey color, or counting colored pixels to get specific features) are not utilized. This approach is more robust to changes in lighting conditions, since it relies on local contrast information rather than on heuristic color segmentation. However, it may produce a much larger set of proposals since it has less filtering steps than the original pipeline proposed in Röfer et al. (2017).

3.4 Deep Robot Detector

The obtained robot proposals are then fed to a CNN that classifies the proposals as robots or non-robots. This CNN is based on the architecture proposed in Cruz et al. (2018), which will be described in Section 4.1. Using grayscale image regions allows the network to process in real-time a large number of robot proposals, since the reduction of input channels from 3 (color space) to 1 (grayscale) greatly reduces the CNN's inference time. The trained robot detection CNN will be called *RobotNet* in the experiments reported in Section 5. The team of each robot in the image is determined by analysing the region corresponding to the robot's shirt, which can be estimated given that the robot's position in the image is known, and using bounds over the standard RGB image to determine the color of the shirt. This approach works well since the shirts have a very high color saturation following the official rules of the SPL. It is important to note that this analysis does not require the color segmented image and its computational cost is very small given that only a small region of the image is analysed.

3.5 Lower-leg Region Proposals Generator

Inspired on the work presented in Mühlenbrock and Laue (2018), we propose an improved orientation determination method, which makes use of CNNs in order to

achieve much better prediction accuracy than the original approach. The proposed method uses the bounding boxes of the detected robots as inputs, finds the regions that contain the lower-legs of each detected robot, and determines the body orientation of each robot by analyzing each lower-leg region. The lower-leg of each robot is characterized by two lines: the so-called *major* and *minor* lines. "The major line is defined from toe to toe and from heel to heel, while the minor line is defined as a side line of a foot" (Mühlenbrock and Laue, 2018). Examples of both lines in different robot poses are shown in Fig. 2.

Fig. 2 Major and minor lines depiction.

As a first step, the set of points that compose the robots' lower silhouette is calculated (Mühlenbrock and Laue, 2018). Then, a region corresponding to the robot's feet is extracted and its Contrast-Normalized Sobel (CNS) image (Müller et al., 2012) is analyzed by using vertical scan lines. Over each scan line pixel an horizontal median filter is applied and its response is compared to a threshold. Pixels with a filter response below the threshold are considered as part of the lower silhouette. Then, by iterating for each scan line, the subset of points that make up a closed convex region can be obtained by using Andrew's convex hulls algorithm (Andrew, 1979). For each consecutive pair of points of the convex set, a line model in field coordinates is calculated. Each line model is then validated with the set of points of the lower silhouette, by using a voting methodology akin to the RANSAC algorithm (Fischler and Bolles, 1981). The line with the highest number of votes is selected as the major line. Once the linear model has been chosen, the minor line may be generated by iterating over the remaining pairs of convex points. This line must fulfill a series of conditions such as a minimum and maximum length, and to be approximately orthogonal to the major line in order to be accepted as valid. Finally, the so-called "lower-leg" proposal is built based on the major and minor lines.

3.6 Deep Robot Orientation Detector

While the major and minor lines can be used to calculate a rotation, the uncertainty on the direction of the robot means that there could be an error of 180 degrees in the orientation estimation. Indeed, a major or minor line can correspond to both the front or the back of the robot. To solve this problem, the robot orientation is determined using a two-stage classification cascade of CNNs, where the first CNN discards low-quality lower-leg regions, and the second CNN determines the robot orientation.

Thus, for each lower-leg proposal, a CNN that measures its quality, *OriBoostNet*, is first applied. Proposals with too much motion blur or that do not correspond to the robots' feet are discarded. This results in a reduction on the number of wrong orientation estimations, since outliers' region proposals are discarded.

If a proposal is not discarded in the first stage of the cascade, it is then analyzed by a second CNN, *OriNet*, which classifies the lower-leg proposal as a *side*, *front* or *back* region. Examples of the proposed regions and their labels are shown in Fig. 3.

Fig. 3 Lower-leg proposals and labels depiction.

After the lower-leg proposals are classified, a consistency check is carried out by imposing that no more than one region of each class must exist for any given robot. This further reduces the number of incorrect orientation estimations. The rotation determination is performed by applying the inverse tangent from two points belonging to the major or minor lines. Then, by using the classes (side, front, back) assigned to each line, the direction of the line can be determined in order to tackle the symmetry problem and to estimate the correct robot orientation.

Finally, the temporal consistency of the orientation estimation is verified; the resulting orientation is added to a buffer that stores the last 11 estimations, and a circular median filter is applied over it. Moreover, in order to avoid invalid results, we consider that the orientation estimation as valid only for a small period of time if no new samples are added to the buffer.

3.7 Ball Proposals Generator

Our ball proposal generator is inspired on the hypotheses generator proposed in (HTWK, 2018). The main differences between both approaches are the following: (i) we only use grayscale images, (ii) we use a different method to estimate high contrast regions (see Section 3.2), and (iii) we use the robot detections to improve the generation of proposals.

The proposal generator uses both the detected high contrast regions and the detected robots' bounding boxes to generate ball hypotheses. The high contrast image regions are utilized because of the soccer ball's high contrast properties (black and white pattern), whilst the robot detections are utilized to discard some of the regions in which a ball detection would be highly unlikely. This way, the detected robot's bounding boxes are used to filter out any high contrast region that would lie on a detected robot's body, keeping those regions lying on the robot's feet.

The filtered image regions are then scanned in a pixel-wise fashion, and the radius that the soccer ball would have for all of the traversed image coordinates is calculated (considering prior knowledge of the field, the ball size, and the robot's camera pose). These radii are used to set the support region of *Difference of Gaussians* (DoG) filters, which are constructed and applied for every image coordinate where a ball radius was calculated. Only the highest filter responses are considered as a ball proposal. This process follows the same principles that the blob search performed to find keypoints in the SIFT algorithm (Lowe, 2004).

3.8 Deep Ball Detector

The ball detection is carried out using a two-stage cascade of CNNs-based classifiers, where the first CNN discards region containing objects that are very different from balls, and the second CNN takes the final decision.

In order to speed up the detection process, the number of detected balls by the first CNN, *BoostBallNet*, is limited to a maximum of five, and then, they are sorted based on their confidence. Then, the second CNN, *BallNet*, analyzes the sorted ball hypotheses to detect the ball. Once the second CNN detects a ball, the remaining hypotheses are discarded.

3.9 Field Lines & Special Features Detection

The field lines and features detection is based on the algorithm proposed in Röfer et al. (2017). The main difference with respect to the original approach, is that in the proposed framework no color information is used. To do this, a set of vertical and horizontal scan lines are used, which save transitions from high-to-low and low-to-high pixel's values. This allows the detection of a set of points which are then fed to the algorithm described in Röfer et al. (2017), in order to associate them with lines and other field features, such as the middle circle, the corners, and line intersections. More details about the algorithm can be found in Röfer et al. (2017).

4 Design and Training of the CNN-based Detectors

In this section we describe the design and training methodologies used to obtain the CNN based classifiers used in the proposed vision framework. Section 4.1 presents the network architecture of the classifiers, and Section 4.2 describes the active learning procedure used to train them.

4.1 Base CNN

The proposed vision system is composed of several statistical classifiers. Each of these classifiers, *RobotNet*, the robot detector, *BoostBallNet* and *BallNet*, the two cascade-stages of the ball detector, and *OriBoostNet* and *OriNet*, the two cascade-stages of the robot orientation estimation network– uses as base the same CNN architecture. The preliminary version of this architecture (*miniSqueezeNet*) was described in Cruz et al. (2018), while in this work slight variations are incorporated to achieve higher processing speeds, while maintaining accuracy.

The main component of *miniSqueezeNet* is the *extended Fire module*, which was proposed in Cruz et al. (2018), inspired by the original *Fire module* (Iandola et al., 2016) and on GoogleNet's *inception module*. This module uses a 1×1 filter placed at the beginning of each extended Fire module to compress the size of the representation into a feature tensor with less channels. This compressed representation is then fed to filters of different sizes; small filters are used to extract spatially local information, while bigger filters obtain global information which is more spatially spread out. The features obtained from these filters are then combined into a single tensor by means of channel wise concatenation and then fed to the next layer. Following this approach allows the training of performant models whose use is computationally inexpensive.

In Cruz et al. (2018) guidelines for designing CNN architectures to be used in embedded systems with low processing capabilities are proposed. The main design

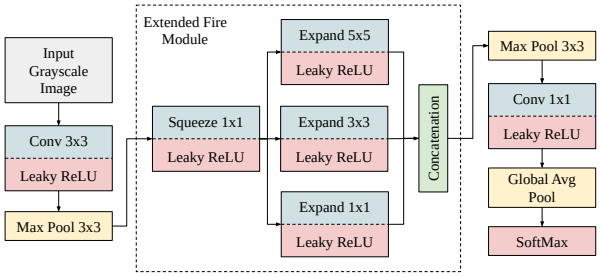


Fig. 4 Modified MiniSqueezeNet network structure.

variables are the depth of the network and the number of filters in each layer. In addition, it is proposed to use max-pooling operations implemented using non-overlapping windows to reduce the inference time. Following these guidelines and using the *extended Fire module*, the so-called *miniSqueezeNet* was designed for the detection of robots in real-time while playing soccer (Cruz et al., 2018).

In this work the *miniSqueezeNet* is further improved. First, grayscale images instead of color images are used as inputs, which reduces the number of input channels from three to one, and modifies the whole structure of the network. Second, leaky ReLU (Maas et al., 2013) instead of ReLU is used as activation function. Previously, we used ReLU in most layers, however, this sometimes resulted in the "dying ReLU" problem while training (no gradients flow backward through the neurons). The use of leaky ReLU solves this, while incurring in no accuracy losses. Further fine-tuning was performed on the networks' structure in order to estimate the correct input size and the required number of parameters. This was done by manually modifying the number of filters in accordance with the requirements of the problem.

A diagram of the new base CNN structure is presented in Fig. 4. All CNN based classifiers were developed using the Darknet library (Redmon, 2013–2016), and trained according to the methodology described in the next section. Taking into account the specific needs of the problem, variations on the number of convolutional filters were used for each of the CNN classifiers. The exact parameters used for each convolutional and maxpooling layer of the trained CNNs can be found in Table 1. Each one of this layers is then followed by batch-normalization and a leaky ReLu activation function.

4.2 Active Learning Training Methodology

The use of an appropriate methodology for the training of the classifiers, which considers realistic game conditions, is crucial to obtain high performant classifiers. We implemented an active learning procedure that selects and annotates unlabeled data obtained under realistic conditions. The training process has several stages which are described in the following paragraphs.

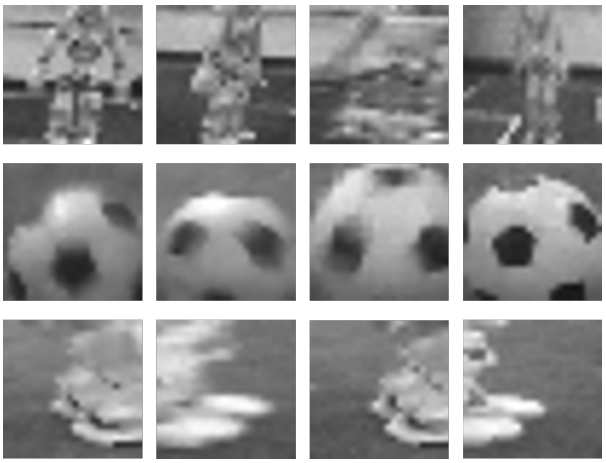


Fig. 5 First row: robot samples, second row: ball samples, third row: feet side samples.

As a first step, the different CNNs are trained using publicly available soccer-robotics datasets, e.g., the SPQR dataset (Albani et al., 2017). However, when the trained CNNs are used for processing images obtained under realistic soccer conditions, the classifiers will likely behave poorly because there is a distribution mismatch between this kind of images and the samples present in the public datasets.

To address this problem, the classifiers must be fine-tuned using the same kind of samples that would actually reach the networks during games. Examples of such images are shown in Fig. 5. To accomplish this, the vision system is deployed on the NAO robot and data is collected using the objects proposal algorithms. Each obtained proposal is classified and stored in the robot's memory with its corresponding label. To get uncorrelated data, we set a constraint for the object's hypotheses to be saved: for the robot proposals and lower-leg proposals for orientation determination, data is acquired periodically in accordance to a predefined time step; for the ball proposals, samples can only be saved if no other proposals with the same position and estimated radius were previously collected. The next stage consists of actively checking the data saved by the observer robot, and manually annotating only the samples that were incorrectly labeled. We then aggregate this data to the original data set and re-train the models.

The above process is repeated until each CNN reaches a high performance. By doing this, we are pro-

Table 1 Structure of the trained CNNs.

Layer		RobotNet	BoostBallNet	BallNet	OriBoostNet	OriNet
Conv 3× 3	Size	3×3	3×3	3×3	3×3	3×3
	Filters	12	4	10	4	12
	Stride	2	2	2	2	2
Max Pool	Size	3×3	3×3	3×3	3×3	3×3
	Stride	2	2	2	2	2
Squeeze 1×1	Size	1×1	1×1	1×1	1×1	1×1
	Filters	6	2	4	4	6
	Stride	1	1	1	1	1
Expand 1×1	Size	1×1	1×1	1×1	1×1	1×1
	Filters	6	2	4	4	6
	Stride	1	1	1	1	1
Expand 3×3	Size	3×3	3×3	3×3	3×3	3×3
	Filters	3	3	2	4	3
	Stride	1	1	1	1	1
Expand 5×5	Size	5×5	-	5×5	-	5×5
	Filters	3	-	2	-	3
	Stride	1	-	1	-	1
Max Pool	Size	3×3	3×3	3×3	3×3	3×3
	Stride	2	2	2	2	2
Conv 1×1	Size	1×1	1×1	1×1	1×1	1×1
	Filters	2	2	2	2	3
	Stride	1	1	1	1	1
Avg Pool	Size	Global	Global	Global	Global	Global

gressively aggregating correctly labeled samples to acquire enough training data for robust feature learning, but we are also aggregating samples which the models fail to correctly infer, to encourage changes in the decision boundaries of the classifiers.

After we obtain proficient models by following the described methodology, we further enhance them by switching to a bootstrap procedure. To do this, we add confidence-based constrains to collect new training data in environments where the objects we want to detect are absent. For instance, if we are getting false positives from the ball detector, we would set the NAO robot to collect data in environments were no balls are present, and we would store every high confidence detection, relabelling them afterwards as non-balls. The samples collected would then be used to re-train the ball classifiers. Likewise, if the orientation detector is labeling a front region as a back region, generating a false positive, we would set the NAO robot to collect data in an environment where only back lower-leg regions are visible to re-train the classifier. Notice that the fine tuning procedure is applied over the detector, which means that when a cascade of CNNs is utilized, a sample is stored based on the compound performance of the CNNs, being the confidence constrain only considered for the last network involved in the classification. This active learning-bootstrap procedure results in a dramatic improvement in the performance of the classifiers after only a few iterations, and also allows the fine tuning of the CNN parameters by means of using data aggregation when an abrupt domain change occurs. Since the inputs to our models have relatively low dimensionality, the space used in the NAO memory during the data collection process is very small, for instance, 1,000 robot proposal samples weight about 3 MB. This procedure, combined with the semi-supervised selection and labeling of the new samples, makes the training process extremely time-wise efficient.

5 Results

5.1 CNN Classification

All classifiers were trained using the methodology described in the previous section. Table 2 shows the obtained model complexity (number of CNN parameters), average inference time (on the NAO robot), and accuracy calculated over a balanced database with a 50% of positive and 50% negative samples for each developed CNN.

Results show that the classifiers achieve very high performance while maintaining low inference times, which proves that their use is suitable for real time ap-

Table 2 Performance of the developed CNNs (Leiva et al., 2019).

Model	RobotNet	BoostBallNet	BallNet	OriBoostNet	OriNet
Input size	24×24×1	12×12×1	26×26×1	12×12×1	24×24×1
N° of parameters	884	125	444	246	657
Inference time (ms)	0.382	0.043	0.343	0.059	0.329
Accuracy	0.969	0.965	0.984	0.962	0.984

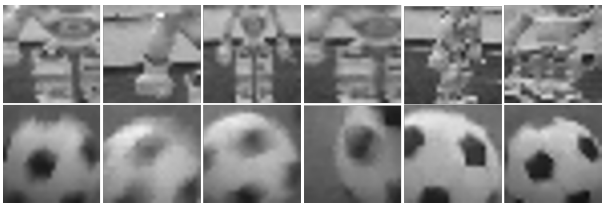


Fig. 6 Example images from our dataset. First row: robot samples, second row: ball samples.

plications, such as robotic soccer. This also validates the effectiveness of the proposed methodology for the design and training of the classifiers. Finally, this also shows that the use of color information is not necessary to detect robots or balls when using classifiers such as CNNs. In fact, the CNN used in the robot detector achieves a similar accuracy rate that the model proposed in Cruz et al. (2018), while being approximately 2.75 times faster.

5.2 Robots, Ball and Field Features Detection Systems

In order to evaluate the designed robot/ball proposal generators and classifiers, we acquire about 600 frames by a humanoid robot player under typical and challenging game conditions. Several lighting conditions were imposed while collecting these frames in order to test the robustness and reliability of our modules. Some examples of the cropped samples obtained from these frames can be found in Fig. 6. The complete database will be public after paper acceptance. The analysis of these frames allowed the extraction of empirical results in relation to the performance of the proposals generators and the CNN based classifiers, which are shown in Table 3. Examples of robot and ball detections can be found in Figures 7 and 8.

Results show that the robots and ball proposals generators achieve high recall rates, while producing an average number of proposals per frame that can be processed in real time by the subsequent classifiers. Given the recall rate of the ball proposals module and the percentage of true positives of the boosting stage, the overall detection module has a very high detection rate. In fact, our ball detector outperforms B-Human's implementation (Röfer et al., 2017), which achieves an

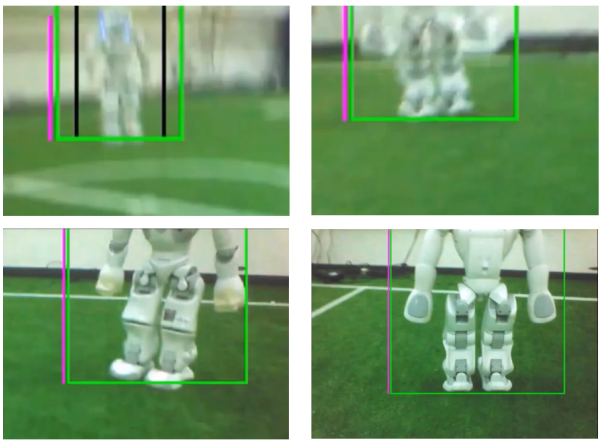


Fig. 7 Examples of robot detections, showing robots' bounding boxes.

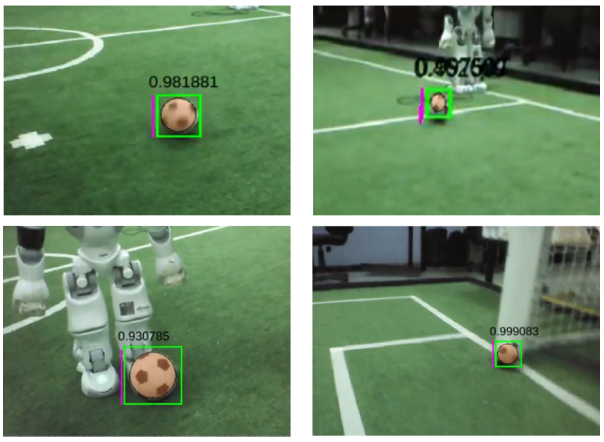


Fig. 8 Examples of ball detections, showing ball bounding boxes and confidence estimations.

overall accuracy rate of 0.697 when testing it under the same conditions. Similarly, the robot detector achieves high recall for the proposal generation and an overall very high accuracy.

Finally, the field lines and features detector was tested by comparing the difference between the real and the estimated robot pose. The estimation was obtained by using the field lines and features detected by our module. By using this approach we calculated a mean

Table 3 Performance of the robots and ball detection systems (Leiva et al., 2019).

Module	Robot Detector	Ball Detector
Proposals per Frame	3.05	10.3
Proposals Recall	0.972	0.993
Overall Accuracy	0.949	0.971

squared error of 40.07 mm, which indicates a suitable accuracy and reliability.

5.3 Robot Orientation Determination

The proposed robot orientation determination system is compared with the one proposed in Mühlenbrock and Laue (2018) (BH: B-Human), which is the only orientation determination system for NAO robots reported in the literature. We analyzed two flavors of our system: the proposed base orientation determination system (UCh), and its output after applying a circular median filtering (UChF). Some examples of the detected rotations as well as the corresponding major and minor lines are shown in Fig. 9.

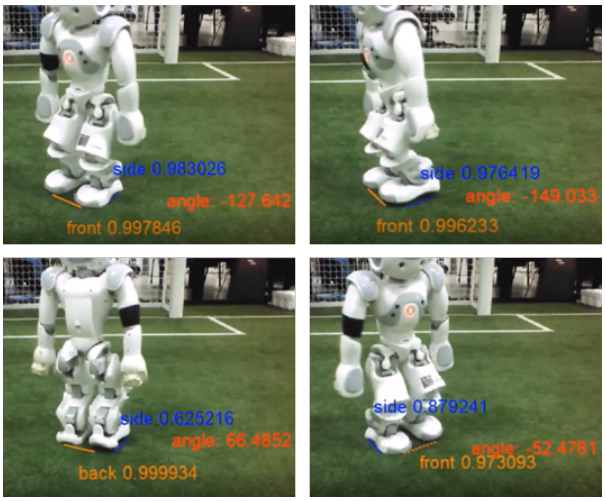


Fig. 9 Top number: confidence minor line. Middle number: estimated rotation. Bottom number: Confidence mayor line.

In the first experiment (static robot), whose results are shown in Fig. 10, the observer and the observed robot are static and placed at a distance of 120cm from each other. For each measurement the observed robot was rotated 22.5° around its axis. As in Mühlenbrock and Laue (2018), we define a *false positive* as any estimation that deviates more than a tolerance angle of 11.25° from the ground-truth. The orientation is classified as *semi perceived* when the rotation can be determined but the facing direction is unknown. The class *not perceived* corresponds to any frame where the orientation could not be calculated, while an orientation estimation is *perceived* if it does not deviate more than a tolerance angle of 11.25° from the ground-truth orientation.

In the second experiment (moving robot), whose results are shown in Fig. 11, the observed robot is moving at a speed of 12.0 cm/s, while the observer remains static. The observed robot is rotated in 45° around its axis for each measurement. We define the same classes for the orientation estimations as in the static experiment, but using a tolerance angle of 22.5°.

As shown in Fig. 10 and in Fig. 11, the proposed method outperforms the baseline system (Mühlenbrock and Laue, 2018). The orientation estimation is completely perceived 99.88% of the time in static conditions, and 95.52% of the time in the dynamic experiment. It is clear that the algorithm proposed is better at determining the facing direction of the observed robots. This results in an increased number of completely perceived orientations while sharply decreasing the number of semi perceived orientations. Also, noise filtering techniques such as the median filter and RANSAC algorithm, combined with the utilization of a CNN contribute to lowering the number of false positive estimations. Finally, the integration of the circular median filter further reduces the number of false positives.

5.4 Profiling

Table 4 shows the maximum and average execution times for the different modules of the proposed vision framework when deployed on the NAO v5 platform. The results obtained show that the proposed color-free vision system is deployable on platforms with limited processing capacity (such as the NAO robot). In addition, they prove the importance of the dimensionality reduction of CNN-based classifier inputs, and how this design decision impacts the performance of the system from a time-efficiency point of view.

5.5 Approach Comparison

To further validate our approach, we compare the proposed detectors to those included in the latest release of the B-Human soccer code (Röfer et al., 2019). The comparison is performed on a realistic simulator (Cruz and Ruiz-del Solar, 2020) able to produce images that closely match reality by using a generative model, trained with images collected from real soccer environments. Samples from the realistic simulator are

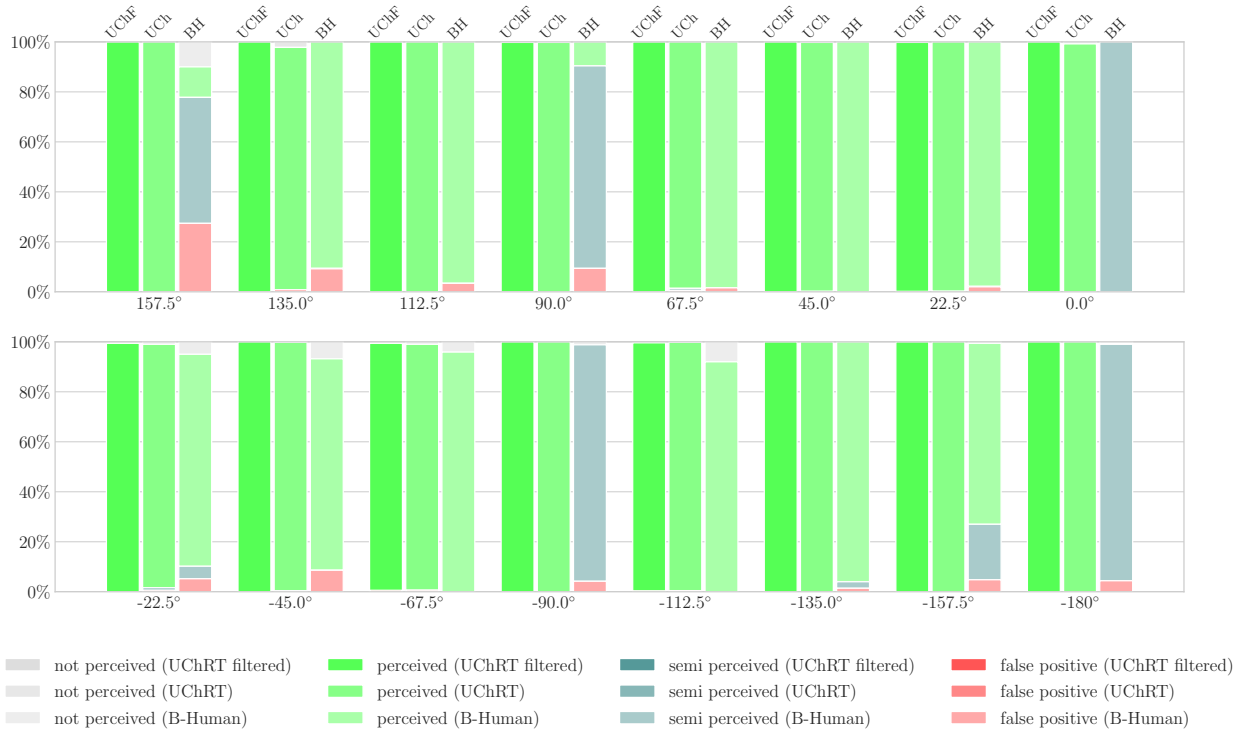


Fig. 10 Results obtained for the first experiment. Graph shows a performance comparison between raw (UCh) and filtered (UChF) estimations for our orientation detector and a B-Human system replication (BH). (Leiva et al., 2019)

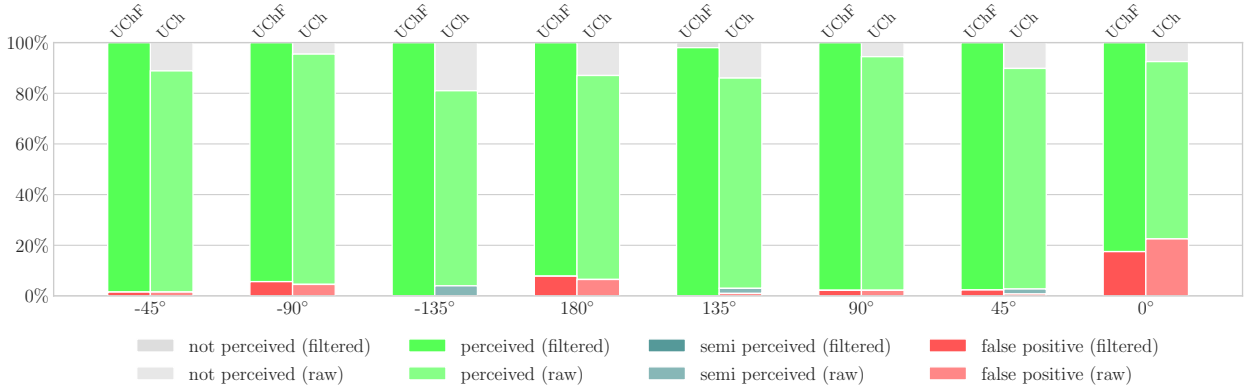


Fig. 11 Dynamic experiment results. Graph shows a performance comparison between raw (UCh) and filtered (UChF) estimations for our orientation detector. (Leiva et al., 2019)

Table 4 Vision framework profiling. Maximum (Max.) and Average (Avg.) processing time in a NAO v5 platform (Leiva et al., 2019).

Module	Max. (ms)	Avg. (ms)
High Contrast Regions Detector	2.755	1.478
Field Lines & Features Detector	2.909	1.300
Robot Proposals Generator	2.692	1.083
Robot Detector	2.417	0.939
Robot Orientation Detector	4.537	1.366
Ball Proposals Generator	2.506	1.132
Ball Detector	6.959	2.452

show in Fig. 12. The simulator is able to randomly shift the pose of the robot in the field as well as the pose of all the other objects in the scene (opponent robots and ball). Moreover, the simulator is also able to provide ground truth information to calculate precise statistics.

We use this simulator to evaluate the performance of the proposed ball and robot detector systems as well as the performance of the robot (Javadi et al., 2018) and ball detectors systems of the B-Human team. Given that teams use different pipelines to detect objects, we define a detection system as the combination of modules

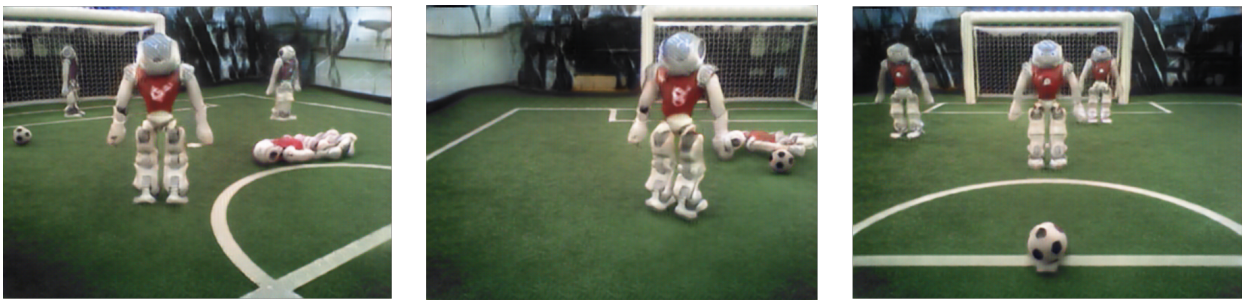


Fig. 12 Image samples generated by the realistic simulator and used to estimate performance metrics.

Table 5 Performance of the robots detection systems.

Robot detector system	Ours	B-Human
Detector recall	0.847	0.702
Detector precision	0.985	0.962
Average time (ms)	2.0	4.5

Table 6 Performance of the ball detection systems.

Ball detector system	Ours	B-Human
Detector recall	0.806	0.831
Detector precision	0.987	0.986
Average time (ms)	3.4	–

that are used to detect an object in the scene. For our framework this means a combination of the region proposal extractor and the region classifier. Other teams use different approaches such as B-Human's robot detector system which is composed of an end-to-end detection model.

All four systems were trained using a data set composed of real samples. In the case of our ball and robot detectors we use the exact same models that were used to achieve the results presented in Table 2. Table 5 presents the metrics collected for both robot detectors systems, while Table 6 presents the results of the ball detector systems. The reported recall and precision metrics correspond to the detection system as a whole, in accordance to how results are presented in (Javadi et al., 2018). The module's average times are measured on a NAO v5 robot.

From Table 5 it can be seen that our method offers better recall than its B-Human counterpart. We found that this difference is the result of a better detection rate of robots facing sideways to the camera. Furthermore, our proposed methodology is less computationally expensive when tested on a NAO v5 robot, running at more than double the average speed when compared to the B-Human approach, as reported in (Javadi et al., 2018). We attribute this to the simplicity of our CNN model, which achieves state of the art performance with fewer computations.

Table 6 shows the corresponding metrics for the ball detector systems. Both approaches are very similar and consist on a region proposal extractor followed by a CNN classifier that takes as input the proposed region in gray scale and outputs the probability that the sample corresponds to a ball. This is then followed by an estimation of the ball's position in the image. Given the similarity of the approaches, it is not surprising that both methods achieve very similar performances in terms of recall and precision. We report the average time of our proposed methodology on a NAO V5, however the average time in a NAO V5 for the B-Human detector is not reported in the literature.

The above results show that the proposed vision framework is competitive with the ones of other teams that consistently reach top spots on the SPL, such as B-Human. This speaks to the overall quality of the proposed system.

5.6 Applicability in Other Domains

We hypothesize that the effectiveness of a vision system built using the proposed approach in a different domain, would depend on the availability of exploitable patterns and regularities in that domain. Furthermore, for such system to function in real-time, its applicability would be restricted to domains in which hand-engineering computationally inexpensive proposal generators is feasible.

Such domains correspond, for instance, to structured environments in which the lighting conditions and the overall geometrical layout of the scene are stable over time. This kind of environments usually corresponds to some indoor spaces, such as industrial plants, and warehouses. Stores and hotels are also viable candidates for this kind of approach. In recent years, robots have begun to become more ubiquitous in this kind of working environments to offload some work from human operators by performing tasks such as greeting cos-

tumers, and delivering room service. Since these kind of robots are often low cost, they usually have low computational capacity, which renders them an ideal target to implement vision systems similar to those proposed on this paper.

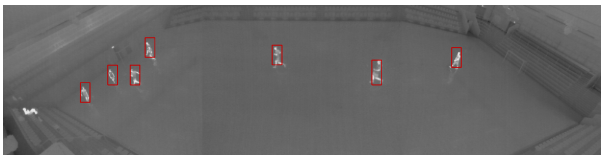
Unstructured environments may also be approachable when using images that contain information regarding their state that simplify their complexity. For instance, the problem of generating proposals for object detection in a complex indoor scene may be simplified if depth or thermal information is available, and the target objects have a known shape and size.

As a proof of concept of these ideas, we implemented a detector for human soccer players as observed by thermal cameras (see Fig. 13). The detector consists of a proposal generator similar to that described in Section 3.7, and a CNN-based classifier that has the RobotNet architecture (see Table 1).

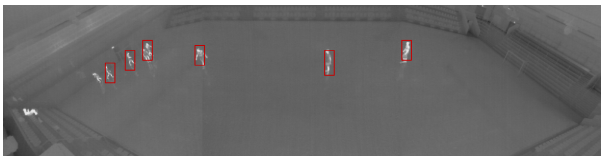
For training and evaluating this detector, the data set presented in Gade and Moeslund (2018) was utilized. This data set is constructed by stitching three simultaneously obtained thermal images from an AXIS Q1922 thermal camera, resulting in 1920×480 pixels images. These images contain between six to eight soccer players in and indoor field (Gade and Moeslund, 2018).

As a proposal generator of the human players, the ball proposal generator described in Section 3.7 was modified so that the support region of the DoG filters applied would coarsely match the silhouette of the players. Moreover, these non-square filters were applied using two different scales. Contrary to the approach adopted for ball detection, the proposal generator this time was applied over the entire image. The produced proposals are resized to 32×32 pixels, and fed to a CNN-based classifier that was trained using labeled proposals from a fraction of the data set.

Figure 13 shows the results provided by the detector over two image samples. The performance metrics for the detector are displayed in Table 7. The performance of the detector could be further improved using tracking, which integrates information over time and thus reduces the number of false negatives by propagating information between consecutive frames, as proposed in Gade and Moeslund (2018). Including more heuristics to the region proposal generator could also improve the overall detector's performance. However this falls outside the scope of this paper. Overall, these results support our hypothesis regarding the applicability of the proposed approach to domains beyond the RoboCup SPL, as suitable solutions can be obtained by constructing systems based on some of the processing pipelines of our detectors and their CNN-based classifiers.



(a)



(b)

Fig. 13 Human soccer players' detection in thermal cameras. (a) Correctly detected players, (b) a false negative detection, and two players incorrectly grouped as one.

Table 7 Performance of the human soccer players detector.

Module	Detector
Proposals per Frame	15.2
Detector System Recall	0.806
Detector System Precision	0.935

6 Conclusions

This paper describes a new vision framework that does not use any color information. This is a novel approach for vision systems designed for the RoboCup SPL, achieving very high performance while being computationally inexpensive.

The proposed vision system we present introduces four new modules: a redesigned robot detector, a visual robot orientation estimator, a brand new ball detector, and finally, a color-free field lines and features detector. All modules developed for this paper are able to run simultaneously in real-time when deployed on a NAO robot playing soccer.

Moreover, we demonstrate that CNN-based classifiers are a useful tool to solve most of the perception requirements of humanoid soccer robotics, and generally translate in an overall better performance of the corresponding modules when coupled with good region proposal algorithms, and a proper use of design and training techniques.

Furthermore, the proposed framework is successfully validated in a different domain, where human soccer players are detected using thermal images. This shows the applicability of the proposed framework beyond soccer robotics.

Acknowledgements

The authors thank Kenzo Lobos-Tsunekawa for his contributions on the development of the robot detection system, and Ignacio Bugueño for his contributions on the development of the robot orientation determination system. This work was partially funded by ANID (Chile) projects FONDECYT 1201170, PIA AFB 180004, and CONICYT-PFCHA/Magíster Nacional/2018-22182130.

References

Albani D, Youssef A, Suriani V, Nardi D, Bloisi DD (2017) A deep learning approach for object recognition with nao soccer robots. In: Behnke S, Sheh R, Sarıel S, Lee DD (eds) RoboCup 2016: Robot World Cup XX, Springer International Publishing, Cham, pp 392–403

Andrew A (1979) Another efficient algorithm for convex hulls in two dimensions. Information Processing Letters 9(5):216 – 219, DOI https://doi.org/10.1016/0020-0190(79)90072-3, URL http://www.sciencedirect.com/science/article/pii/0020019079900723

Cruz N, Ruiz-del Solar J (2020) Closing the simulation-to-reality gap using generative neural networks: Training object detectors for soccer robotics in simulation as a case study. In: The International Joint Conference on Neural Networks 2020

Cruz N, Lobos-Tsunekawa K, Ruiz-del Solar J (2018) Using convolutional neural networks in robots with limited computational resources: Detecting nao robots while playing soccer. In: Akiyama H, Obst O, Sammut C, Tonidandel F (eds) RoboCup 2017: Robot World Cup XXI, Springer International Publishing, Cham, pp 19–30

Felbinger GC, Göttsch P, Loth P, Peters L, Wege F (2019) Designing convolutional neural networks using a genetic approach for ball detection. In: Holz D, Genter K, Saad M, von Stryk O (eds) RoboCup 2018: Robot World Cup XXII, Springer International Publishing, Cham, pp 150–161

Fischler MA, Bolles RC (1981) Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Commun ACM 24(6):381–395, DOI 10.1145/358669.358692, URL https://doi.org/10.1145/358669.358692

Gabel A, Heuer T, Schiering I, Gerndt R (2019) Jetson, where is the ball? using neural networks for ball detection at robocup 2017. In: Holz D, Genter K, Saad M, von Stryk O (eds) RoboCup 2018: Robot World Cup XXII, Springer International Publishing, Cham, pp 181–192

Gade R, Moeslund TB (2018) Constrained multi-target tracking for team sports activities. IPSJ Transactions on Computer Vision and Applications 10(1):2

Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H (2017) Mobilenets: Efficient convolutional neural networks for mobile vision applications. 1704.04861

HTWK NT (2018) Nao-team htwk: Team research report. http://www.htwk-robots.de/documents/TRR_2017.pdf?lang=en, accessed: 2020-01-30

Iandola FN, Han S, Moskewicz MW, Ashraf K, Dally WJ, Keutzer K (2016) Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size. 1602.07360

Javadi M, Azar SM, Azami S, Ghidary SS, Sadeghnejad S, Baltes J (2018) Humanoid robot detection using deep learning: A speed-accuracy tradeoff. In: Akiyama H, Obst O, Sammut C, Tonidandel F (eds) RoboCup 2017: Robot World Cup XXI, Springer International Publishing, Cham, pp 338–349

Kukleva A, Khan MA, Farazi H, Behnke S (2019) Utilizing temporal information in deep convolutional network for efficient soccer ball detection and tracking. In: Chalup S, Niemueller T, Suthakorn J, Williams MA (eds) RoboCup 2019: Robot World Cup XXIII, Springer International Publishing, Cham, pp 112–125

Leiva F, Cruz N, Bugueño I, Ruiz-del Solar J (2019) Playing soccer without colors in the spl: A convolutional neural network approach. In: Holz D, Genter K, Saad M, von Stryk O (eds) RoboCup 2018: Robot World Cup XXII, Springer International Publishing, Cham, pp 122–134

Lowe DG (2004) Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision 60(2):91–110, DOI 10.1023/B:VISI.0000029664.99615.94, URL https://doi.org/10.1023/B:VISI.0000029664.99615.94

Maas AL, Hannun AY, Ng AY (2013) Rectifier nonlinearities improve neural network acoustic models. In: in ICML Workshop on Deep Learning for Audio, Speech and Language Processing

Menashe J, Kelle J, Genter K, Hanna J, Liebman E, Narvekar S, Zhang R, Stone P (2018) Fast and precise black and white ball detection for robocup soccer. In: Akiyama H, Obst O, Sammut C, Tonidandel F (eds) RoboCup 2017: Robot World Cup XXI, Springer International Publishing, Cham, pp 45–58

Mühlenbrock A, Laue T (2018) Vision-based orientation detection of humanoid soccer robots. In: Akiyama H, Obst O, Sammut C, Tonidandel F (eds)

RoboCup 2017: Robot World Cup XXI, Springer International Publishing, Cham, pp 204–215

Müller J, Frese U, Röfer T (2012) Grab a mug - object detection and grasp motion planning with the nao robot. In: 2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012), pp 349–356, DOI 10.1109/HUMANOIDS.2012.6651543

Otsu N (1979) A threshold selection method from gray-level histograms. IEEE Transactions on Systems, Man, and Cybernetics 9(1):62–66, DOI 10.1109/TSMC.1979.4310076

Poppinga B, Laue T (2019) Jet-net: Real-time object detection for mobile robots. In: Chalup S, Niemueller T, Suthakorn J, Williams MA (eds) RoboCup 2019: Robot World Cup XXIII, Springer International Publishing, Cham, pp 227–240

Redmon J (2013–2016) Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`

Redmon J, Farhadi A (2017) Yolo9000: Better, faster, stronger. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

RoboCup (2020a) Robocup federation official website. `https://www.robocup.org/objective/`, accessed: 2020-01-30

RoboCup (2020b) Robocup standard platform league official website. `https://spl.robocup.org/`, accessed: 2020-01-30

Röfer T, Laue T, Bülter Y, Krause D, Kuball J, Mühlenbrock A, Poppinga B, Prinzler M, Post L, Roehrig E, Schröder R, Thielke F (2017) B-Human team report and code release 2017. Only available online: `http://www.b-human.de/downloads/publications/2017/coderelease2017.pdf`

Röfer T, Laue T, Baude A, Blumenkamp J, Felsch G, Fiedler J, Hasselbring A, Haß T, Oppermann J, Reichenberg P, Schrader N, Weiß D (2019) B-Human team report and code release 2019. Only available online: `http://www.b-human.de/downloads/publications/2019/CodeRelease2019.pdf`

Speck D, Barros P, Weber C, Wermter S (2017) Ball localization for robocup soccer using convolutional neural networks. In: Behnke S, Sheh R, Sarıel S, Lee DD (eds) RoboCup 2016: Robot World Cup XX, Springer International Publishing, Cham, pp 19–30

Speck D, Bestmann M, Barros P (2019) Towards real-time ball localization using cnns. In: Holz D, Genter K, Saad M, von Stryk O (eds) RoboCup 2018: Robot World Cup XXII, Springer International Publishing, Cham, pp 337–348

Szemenyei M, Estivill-Castro V (2019a) Real-time scene understanding using deep neural networks for robocup spl. In: Holz D, Genter K, Saad M, von Stryk O (eds) RoboCup 2018: Robot World Cup XXII, Springer International Publishing, Cham, pp 96–108

Szemenyei M, Estivill-Castro V (2019b) Robo: Robust, fully neural object detection for robot soccer. In: Chalup S, Niemueller T, Suthakorn J, Williams MA (eds) RoboCup 2019: Robot World Cup XXIII, Springer International Publishing, Cham, pp 309–322

Teimouri M, Delavaran MH, Rezaei M (2019) A real-time ball detection approach using convolutional neural networks. In: Chalup S, Niemueller T, Suthakorn J, Williams MA (eds) RoboCup 2019: Robot World Cup XXIII, Springer International Publishing, Cham, pp 323–336

Viola P, Jones M (2001) Rapid object detection using a boosted cascade of simple features. In: Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, vol 1, pp I–I, DOI 10.1109/CVPR.2001.990517

Using Convolutional Neural Networks in Robots with Limited Computational Resources: Detecting NAO Robots while Playing Soccer

Nicolás Cruz, Kenzo Lobos-Tsunekawa, and Javier Ruiz-del-Solar

Advanced Mining Technology Center & Dept. of Elect. Eng., Universidad de Chile
{nicolas.cruz,kenzo.lobos,jruizd}@ing.uchile.cl

Abstract. The main goal of this paper is to analyze the general problem of using Convolutional Neural Networks (CNNs) in robots with limited computational capabilities, and to propose general design guidelines for their use. In addition, two different CNN based NAO robot detectors that are able to run in real-time while playing soccer are proposed. One of the detectors is based on the XNOR-Net and the other on the SqueezeNet. Each detector is able to process a robot object-proposal in ~1ms, with an average number of 1.5 proposals per frame obtained by the upper camera of the NAO. The obtained detection rate is ~97%.

Keywords: Deep learning, Convolutional Neural Networks, Robot Detection

1 Introduction

Deep learning has allowed a paradigm shift in pattern recognition, from using hand-crafted features together with statistical classifiers, to using general-purpose learning procedures to learn data-driven representations, features, and classifiers together. The application of this new paradigm has been particularly successful in computer vision, in which the development of deep learning methods for vision applications has become a hot research topic. This new paradigm has already attracted the attention of the robot vision community. However, the question is whether or not new deep learning solutions to computer vision and recognition problems can be directly transferred to robot vision applications. We believe that this transfer is not straightforward considering the multiple requirements of current deep learning solutions in terms of memory and computational resources, which in many cases include the use of GPUs. Furthermore, we believe that this transfer must consider that robot vision applications have different requirements than standard computer vision applications, such as real-time operation with limited on-board computational resources, and the constraining observational conditions derived from the robot geometry, limited camera resolution, and sensor/object relative pose.

One of the main application areas of deep learning in robot vision is object detection and categorization. These are fundamental abilities in robotics, because they enable a robot to execute tasks that require interaction with object instances in the real-world. State-of-the-art methods used for object detection and categorization are based on generating object proposals, and then classifying them using a

Convolutional Neural Network (CNN), enabling systems to detect thousands of different object categories. But as already mentioned, one of the main challenges for the application of CNNs for object detection and characterization in robotics is real-time operation. On the one hand, obtaining the required object proposals for feeding a CNN is not real-time in the general case, and on the other hand, general-purpose object detection and categorization CNN based methods are not able to run in real-time in most robotics platforms. These challenges can be addressed by using task-dependent methods for generating few, fast and high quality proposals for a limited number of possible object categories. These methods are based on using other information sources for segmenting the objects (depth information, motion, color, etc.), and/or by using non general-purpose, but object specific weak detectors for generating the required proposals. In addition, fast and/or lightweight CNN architectures can be used when dealing with a limited number of object categories.

Preliminary CNN based object detection systems have been already proposed in the context of robotic soccer. In [1], a CNN system is proposed for detecting players in RGB images. Player proposals are computed by using color-segmentation based techniques. Then, a CNN is used for validating the player detections. Different architectures with 3, 4, and 5 layers are explored, all of them using ReLU. In the reported experiments, the 5-layer architecture is able to obtain 100% accuracy when processing images at 11-19 fps on a NAO robot, when all non-related processes such as self-localization, decision-making, and body control are disabled. In [2], a CNN-based system for detecting balls inside an image is proposed. Two CNNs are used, consisting of three shared convolutional layers, and two independent fully-connected layers. Both CNNs are able to obtain a localization probability distribution for the ball over the horizontal and vertical image axes respectively. Several nonlinearities were tested, with the soft-sign activation function generating the best results. Processing times in NAO platforms are not reported in that work. From the results reported in [1] and [2], it can be concluded that these object detectors cannot be used in real-time by a robot with limited computational resources (e.g. a NAO robot) while playing soccer, without disturbing other fundamental processes (walk engine, self-localization, etc.).

In this context the main goal of this paper is to analyze the general problem of using CNNs in robots with limited computational capabilities and to propose general design guidelines for their use. In addition, two different CNN based NAO robot detectors that are able to run in real-time while playing soccer are proposed. Each of these detectors is able to analyze a robot object-proposal in ~1ms, and the average number of proposals to analyze in the presented system is 1.5 per frame obtained by the upper camera of the NAO. The obtained detection rate is ~97%.

2 Deep Learning in Robots with limited Computational Resources

The use of deep learning in robot platforms with limited computational resources requires to select fast and lightweight neural models, and to have a procedure for their design and training. These two aspects are addressed in this section.

2.1 Neural Network Models

State-of-the-art computer vision systems based on CNNs require large memory and computational resources, such as those provided by high-end GPUs. For this reason, CNN-based methods are unable to run on devices with low resources, such as smartphones or mobile robots, limiting their use in real-world applications. Thus, the development of mechanisms that allow CNNs to work using less memory and fewer computational resources, such as compression and quantization of the networks, is an important research area.

Different approaches have been proposed for the compression and quantization of CNNs. Among them, methods that compute the required convolutions using FFT [16], methods that use sparse representation of the convolutions such as [17] and [18], methods that compress the parameters of the network [19], and binary approximations of the filters [5]. This last option has shown very promising results. In [5], two binary-based network architectures are proposed: Binary-Weight-Networks and XNOR-Networks. In Binary-Weight-Networks, the filters are approximated with binary values in closed form, resulting in a 32x memory saving. In XNOR-Networks, both the filters and the input of convolutional layers are binary, but non-binary non-linearities like ReLU can still be used. This results in 58x faster convolutional operations on a CPU, by using mostly XNOR and bit-counting operations. The classification accuracy with a Binary-Weight-Network version of AlexNet is only 2.9% less than the full-precision AlexNet (in top-1 measure); while XNOR-Networks have a larger, 12.4%, drop in accuracy. An alternative to compression and quantization is to use networks with a low number of parameters in a non-standard CNN structure, such as the case of SqueezeNet [3]. Vanilla SqueezeNet achieves AlexNet accuracy using 50 times fewer parameters. This allows for more efficient distributed training and feasible deployment in low-memory systems such as FPGA and embedded systems such as robots. In this work, we select XNOR-Net and SqueezeNet for implementing NAO robot detectors, and to validate the guidelines being proposed.

2.2 Design and Training Guidelines

We propose general design guidelines for CNNs to achieve real-time operation and still maintain acceptable performances. These guidelines consist on an *initialization step*, which sets a starting point in the design process by selecting an existing state-of-the-art base network, and by including the nature of the problem to be solved for selecting the objects proposal method and size, and an *iterative design step*, in which the base network is modified to achieve an optimal operating point under a Pareto optimization criterion that takes into account inference time and the classification performance.

Initialization

- Object Proposals Method Selection: A fast method for obtaining the object proposals must be selected. This selection will depend on the nature of the problem being solved, and on the available information sources (e.g., depth data obtained by a range sensor). In problems with no additional information sources, color-based

proposals are a good alternative (e.g., in [12]).

- Base Network Selection: As base network a fast and/or lightweight neural model, as the ones described in sub-section 2.1 must be selected. As a general principle, networks already applied in similar problems are preferred.

- Image/Proposal Size Selection: The image/proposal size must be set accordingly to the problem's nature and complexity. Large image sizes can produce small or no increases in classification performance, while increasing the inference times. The image size must be small, but still large enough to capture the problem's complexity. For example, in face detection, an image/window size of 20x20 pixels is enough in most state-of-the-art detection systems.

Sequential Iteration

A Pareto optimization criterion is needed to select among different network's configurations with different classification performances and inference times. The design of this criterion must reflect the importance of the real-time needs of the solution, and consider a threshold, i.e. a maximum allowed value, in the inference time from which solutions are feasible. By using this criterion, the design process iterates for finding the Pareto's optimal number of layers and filters:

- Number of layers: Same as in the image size case, the needed number of layers depends on the problem complexity. For some classification problems with a high number of classes, a large number of layers is needed, while for two-class classification, high performances can be obtaining with a small number of layers (e.g. as small as 3). One should explore the trade-off produced with the number of layers, but this selection must also consider the number of filters in each layer. In the early stages of the optimization, the removal of layers can largely reduce the inference time without hindering the network's accuracy.

- Number of filters: The number of filters in each convolutional layer is the last parameter to be set, since it involves a high number of correlated parameters. The variations in the number of filters must be done iteratively with slight changes in each step, along the different layers, to evaluate small variations in the Pareto criterion.

The proposed guidelines are general, and adaptations must be done when applying them to specific deep models and problems. Examples of the required adaptations are presented in Section 3.1 and 3.2 for the SqueezeNet and XNOR-Net, respectively.

3 Case Study: Real-time NAO Detection while Playing Soccer

The detection of other robots is a critical task in robotic soccer, since it enables players to perceive both teammates and opponents. In order to detect NAO robots in real-time while playing soccer, we propose the use of CNNs as image classifiers, turning the robot detection problem into a binary classification task, with a focus on real-time, in-game use. Under this modeling, the CNN based detector will be fed by object proposals obtained using a fast robot detector (e.g. the one proposed in [12]).

Since the main limitation for the use of CNNs in robotic applications is the memory consumption and the execution time, we select two state-of-the-art CNNs to address the NAO robot detection problem: SqueezeNet [3], which generates lightweight models, and XNOR-Nets [5], which produces fast convolutions. NAO

robot detectors using each of those networks are designed, implemented and validated. In both cases, the proposed design guidelines are followed, using the same Pareto criterion, with a maximum processing time of 2ms to ensure real-time operation while playing soccer.

One important decision when designing and training deep learning systems is the learning framework to be used. We analyzed the use of three frameworks with focus on deployment in embedded systems: Caffe [13], TensorFlow [14], and Darknet [15]. Even though Caffe is implemented in C++, its many dependencies make the compatibility in 32-bit systems highly difficult. Tensorflow is also written in C++ (the computational core), but it offers a limited C++ API. Hence, we chose Darknet, which is a small C library with not many dependencies, which allows an easy deployment in the NAO, and the implementation of state-of-the-art algorithms [5].

For the training and validation of the proposed networks we use the NAO robot database published in [1], which includes images taken in various game situations and under different illumination conditions.

3.1 Detection of NAO Robots using SqueezeNet

In the context of implementing deep neural networks in systems with limited hardware, such as the NAO robot, SqueezeNet [4] appears as a natural candidate. First of all, the small model size allows for network deployment in embedded systems without requiring large portions of the memory to store the network parameters. Second, the reduced number of parameters can lead to faster inference times, which is fundamental for the real-time operation of the network.

These two fundamental advantages of the SqueezeNet arise from what the authors call a *fire module* (see Figure 1 (a)). The fire module is composed of three main stages. First, a squeeze layer composed of 1x1 filters, followed by an *expand layer* composed of 1x1 and 3x3 filters. Finally, the outputs of the *expand layer* are concatenated to form the final output of the fire module.

The practice of using filters of different sizes and then concatenating their outputs is not new, and has been used in several networks, most notably in GoogLeNet [6], with its *inception module* (see Figure 1 (c)). This module is based on the idea that sparse neural networks are less prone to overfitting due to the reduced number of parameters and are theoretically less computationally expensive. The problem with creating a sparse neural network arises due to the inefficiency of sparse data structures. This was overcome in GoogLeNet by approximating local sparse structures with dense components as suggested in [7], giving birth to the *naïve inception module*. This module uses a concatenation of 1x1, 3x3, and 5x5 filters; 1x1 filters are used to detect correlation in certain clusters between channels, while the larger 3x3 and 5x5 filters detect more spatially spread out of the clusters. Since an approximation of sparseness is the goal, ReLu activation functions are used to set most parameters to zero after training. The same principle is at the core of the fire module, which concatenates the outputs of 1x1 and 3x3 filters, but eliminating the expensive 5x5 filter. While concatenating the results of several filter's sizes boost performance, it has a serious drawback: large convolutions are computationally

expensive if they are placed after a layer that outputs a large number of features. For that reason both, the fire module and the inception module, use 1x1 filters to reduce the number of features before the expensive large convolutions. The 1x1 filter was introduced in [9] as a way to combine features across channels after convolutions, while using very few parameters.

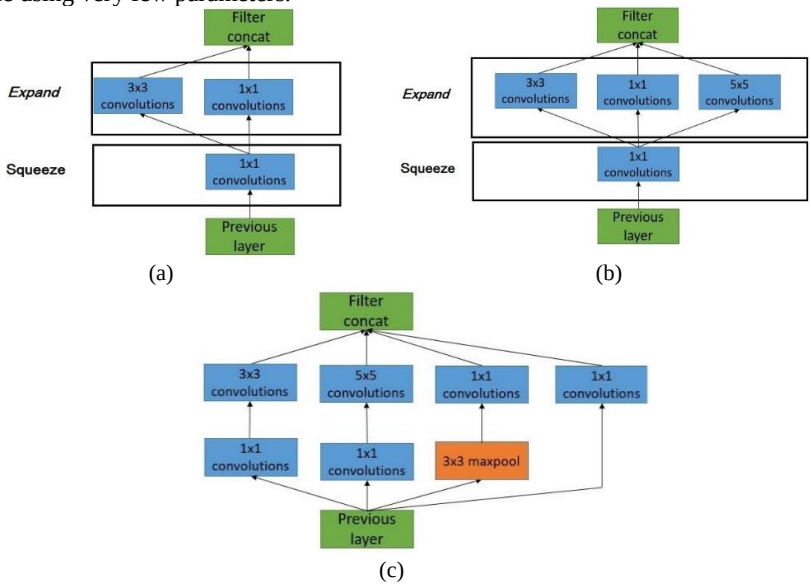


Figure 1. (a) Fire module from SquezeNet [3]. (b) Extended fire module (proposed here). (c) Inception module from GoogLeNet [6].

The main difference between the inception module and the fire module approaches to dimension reduction lies in the structure. The inception module has each of the 1x1 filter banks feeding only one of the large convolutional filters of the following layer, so there are as many 1x1 filter banks in the feature reduction layer as there are large convolutions in the next layer. However, if we assume a high correlation between the outputs of each of the 1x1 filter banks in the feature reduction layer, all filters in this layer could be condensed into only one 1x1 filter bank that feeds all the filters in the next layer. This approach was taken by the creators of the SqueezeNet. In our experiments, we found that adding a 5x5 filter bank to the expand layer of the fire module, in what we called an *extended fire module* (proposed here), can boost performance. The extended fire module was developed for this paper, and is shown in Figure 1 (b). In this modified structure one 1x1 filter bank of the squeeze layer feeds the 1x1, 3x3 and 5x5 filters, further confirming the idea that the 1x1 filter banks of the inception module are heavily correlated in some cases, and can be compressed in just 1 bank.

In order to adapt the SqueezeNet to embedded systems some changes need to be made to the vanilla architecture of SqueezeNet, in particular to the depth of the network and the number of filters in each layer. We recommend resizing the network in order to achieve optimal inference time by following the guidelines postulated in

Section 2. However, the size reduction usually comes with reduced network accuracy. To solve this problem, we propose to use the following two strategies. First, in case of reduced accuracy due to network resizing, we propose replacing the ReLu activation function with a PreLu activation function in early layers as suggested in [10]. If this approach fails to deliver extra accuracy, then replacing standard fire modules with extended fire modules can increase the quality of the network. The overall inference time can be further diminished without reducing accuracy by implementing all maxpool operations using non-overlapping windows as suggested in [11]. The proposed iterative algorithm to produce an optimal network is presented in Figure 2.

reduce image size
make maxpool windows non-overlapping
while *network can be improved according to a Pareto criteria* **do**
  resize the network in term of layers and filters following the guidelines in Section 2
  if *the accuracy is lower than desired* **do**
    replace the ReLu activation functions of initial layers by PreLu
  end if
  if *the accuracy is lower than desired and using PreLu doesn't improve accuracy* **do**
    replace fire modules by extended fire modules
  end if
end while
end optimization

Figure 2. Guidelines for real-time SqueezeNet implementation in embedded systems.

Table 1 presents execution times and classification performances achieved by different variants of the Squeeze network obtained by following the design procedure shown in Figure 2. First, the SqueezeNet, designed originally for the ImageNet database, was modified (*NAO adapted SqueezeNet*) to provide the correct number of output classes, and the size of the input was changed to match the size of the used region proposals. This network was further changed by reducing the number of filters and layers according to the guidelines in Section 2, substituting ReLu with PreLu activation function in the first convolutional layer of the network, and using maxpool operations with non-overlapping windows, giving birth to the *miniSqueezeNet2* variant. For *miniSqueezeNet3* several image input sizes were tested and 24x24 was found to have the right dimensions to achieve low inference time while preserving accuracy. To further reduce inference time, the number of filters was also diminished. Finally, in the *miniSqueezeNet4* variant the number of layers and filters was further reduced, and the remaining fire module was replaced by the newly developed extended fire module. The structure of *miniSqueezeNet4* is shown in Figure 3.

Interestingly as the inference time and number of free parameters decreases the network becomes more accurate. It is important to note that simply reducing the number of filters and layers is not a good method to achieve real-time inference, since following this simple approach will result in very poor network accuracy. Instead, by methodically and iteratively applying the proposed guidelines and testing the network, one can achieve very low inference time while retaining or even increasing accuracy. Another factor to take into account is that the network's size reduction can lead to a higher accuracy for small datasets due to the overfitting reduction, given the smaller number of tunable parameters. In the context of the RoboCup this characteristic

becomes extremely relevant since datasets are small, because the building process is slow.

Table 1. Inference times and classification results for different SqueezeNet networks.

Name of the network	Inference time on the NAO [ms]	Classification Rate [%]
NAO adapted SqueezeNet	68.4	51.25
miniSqueezeNet2	3.5	92.5
miniSqueezeNet3	1.55	96.33
miniSqueezeNet4	1.05	98.30

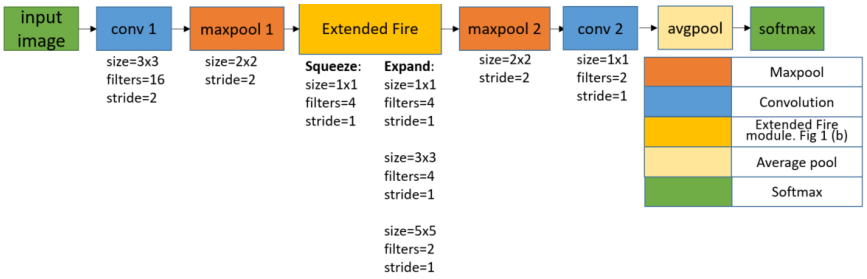


Figure 3. Diagram of the miniSqueezeNet4 network designed in Section 3.1.

3.2 Detection of NAO Robots using XNOR-Net

Since the use of deep learning approaches in robotic applications becomes limited by memory consumption and processing time, many studies have been conducted trying to compress models or approximate them using various techniques. In [4] it is stated that 70-90% of the execution time of a typical convolutional network is used in the convolution layers, so it is natural to focus the study in how to optimize or approximate those layers. From the many options that have been proposed in the last few years, XNOR-Nets [5] becomes an attractive option due to its claim to achieve a 58x speedup in convolutional operations. This speedup is produced since both the input representation in each layer, and the associated weights, are binarized. Hence, a single binary operation can replace up to 64 floating point operation (in a 64-bit architecture). However, since not all operations are binary, the theoretical speedup is around 62x, and in [5] a practical 58x speedup is achieved.

However, even if these results are promising, implementations on embedded systems need to consider the target architecture, which affects directly the speedup obtained by the binary convolutions. For example, in CPU implementations, two critical aspects are the word length and the available instruction set. In the specific case of the NAO, which uses an Intel Atom Z530, the word length is 32-bits, which halves the theoretical speedup, and the instruction set does not support hardware bit-counting operations, which are needed for an optimal implementation, since counting

bits is an important factor in XNOR layers, as they replace sums in convolutions.

Since the authors of [5] do not release their optimized CPU version of XNOR-Nets, we use our own, by implementing the binary counterparts of the popular *gemm* and *im2col* algorithms, obtaining an asymptotic speedup of 15x in the convolutional operations, with the bottleneck being the bit counting operations, which are computed by software algorithms.

The design of convolutional networks using XNOR layers for specific, real-time applications must follow the design procedure explained in Section 2. However, since the XNOR layers are approximations of normal convolutions, in each design step, both the XNOR and the full precision versions of the used CNN architecture must be considered, in order to perform the next step, since some architectures take more advantage than others of the binarization. Furthermore, it is important to remark that even though XNOR layers can substitute any convolutional layers, it is not convenient to replace the first and the last convolution layers, since binarization in those layers produces high information losses.

To validate the proposed design methodology for the specific XNOR-Net architecture, we consider as base networks the following three, as well as their binarized versions: AlexNet, the convolutional network proposed in [15] for the CIFAR-10 database (here called *Darknet-CIFAR10*), and another network for the CIFAR-10 database, also proposed in [15] (here called *Darknet-CIFAR10-v2*). The performances of these three base networks, and their binarized counterparts, are shown in Table 2. We chose *Darknet-CIFAR10-v2* for applying our design guidelines, since it achieves high classification performance, using much less computation resources than the other two networks. As a result of applying the proposed design guidelines, the *miniDarknet-CIFAR10* network shown in Figure 4 is obtained, which achieves a slightly lower classification performance than Darknet-CIFAR10-v2, but has an inference times of less than one millisecond (see last two rows in Table 2).

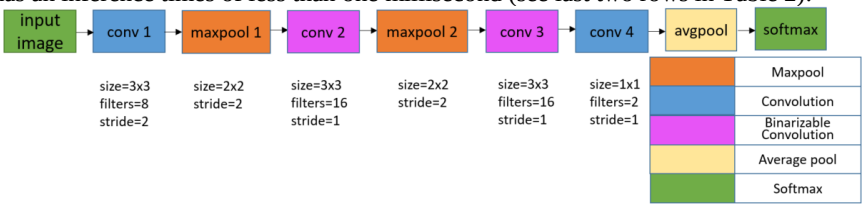


Figure 4. Diagram of the miniDarknet-CIFAR10 network designed in Section 3.2.

Table 2. Inference times and classification results for XNOR-Networks

Name of the network		Inference time on the NAO [ms]	Classification Performance [%]
Alexnet	Full precision	7400	97.2
	XNOR	1500	97.8
Darknet-CIFAR10	Full precision	4400	99.2

	XNOR	400	93.8
Darknet-CIFAR10-v2	Full precision	48	98.6
	XNOR	11.5	98.1
miniDarknet-CIFAR10	Full precision	0.9	97.6
	XNOR	0.95	96.6

3.3 Robot Detection while Playing Soccer

The two deep learning based detectors described in the two former sub-sections need to be fed using region proposals. As region proposals generator we choose the algorithm described in [12]. This algorithm scans the NAO image using vertical scanlines, where non-green spots are detected and merged into a bounding-box, which constitutes a region proposal. This algorithm runs in less than 2ms in the NAO [12], and although it shows excellent results on simulated environments, it fails under wrong color calibration and challenging light conditions, generating false detections, which is why a further classification step is needed.

The computation time of the whole NAO robot detection system (proposal generation + deep based robot detector) is calculated by adding the execution time of the region proposal algorithm, and the convolutional network inference time multiplied by the expected number of proposals. To estimate the expected number of proposals, several realistic game simulations where run using the SimRobot simulator [12], and then the number of possible robot proposals was calculated for each of the cameras. The final execution times are presented in Table 3. It is important to note that we use the average number of proposals in the upper camera, since the lower camera rarely finds a robot proposal.

Table 3. Execution time of the robot detection system.

Regions proposal time	0.85 [ms]
Selected network inference time (XNOR-Net)	0.95 [ms]
Average number of proposals (in the upper NAO camera)	1.5
Average total detection time	2.275 [ms]

3.4 Discussion

The XNOR-Net and SqueezeNet design methodologies have been validated, obtaining inference times and classification performances that allow deployment in real robotic platforms with limited computational resources, such as the NAO robot. The main common principles derived from the proposed methodologies are:

1. To select a base network taking as starting point fast and/or lightweight deep models used in problems of similar complexity - XNOR-Net and SqueezeNet seems to be good alternatives for object detection problems of a similar complexity than the robot detection problem described here.
2. To select an image/proposal size according to the problem's complexity (24x24 pixels was the choice in the described application).
3. To follow an iterative design process by reducing the number of layers and filters, following a Pareto optimization criterion that considers classification performance and inference time.

In the described NAO robot detection problem, the best detectors for each network type (XNOR-Net and SqueezeNet) are comparable, obtaining a very similar performance. While the XNOR-Net based detector achieves a marginally lower inference time (0.95 ms against 1.05 ms), the SqueezeNet based detector gives a better classification performance (98.30% against 96.6%). We also validate the hypothesis that hybrid systems that use handcrafted region proposals that feed CNN classifiers are a competitive choice against end-to-end methods, which integrate proposal generation and classification in a single network such as Faster R-CNN, since the use of the first kind of methods (handcrafted proposals + deep networks) make possible the application of the final detector in real-time.

It must be noted that while the reported network inference times are the ones of a network running in a real NAO robot, the reported classification performances correspond to the test results when using the SPQR database [1]. The performance using this database may differ from the performance in real-world conditions, since the data distribution in this database might be different from the one expected in real games.

4 Conclusions

In this paper two deep neural networks suited for deployment in embedded systems were analyzed and validated. The first one, XNOR consists on the binarization of a CNN network, while the second one, SqueezeNet, is based on a lightweight architecture with a reduced number of parameters. Both networks were used for the detection of NAO robots in the context of robotic soccer, and obtained state-of-the-art results (~97% detection rate), while having very low computational cost (~1ms for analyzing each robot proposal, with an average of 1.5 proposal per image).

With this work, we show that using deep learning in NAO robots is indeed feasible, and that it is possible to achieve state-of-the-art robot detection while playing soccer. Similar neural network structures to the ones proposed in this paper can be used to perform other detections tasks, such as ball detection or goal post detection in this same context. Moreover, since the methodologies presented in this work to achieve real-time capabilities are generic, it is possible to implement the same strategies in applications with similar hardware restrictions such as smartphones, x-rotors and low-end robot systems.

Acknowledgements

This work was partially funded by FONDECYT Project 1161500.

REFERENCES

1. Albani, D., Youssef, A., Suriani, V., Nardi, D., Bloisi, D.D.: A Deep Learning Approach for Object Recognition with NAO Soccer Robots. RoboCup Int. Symposium. July 2016
2. Speck, D., Barros, P., Weber, C., Wermter, S.: Ball Localization for Robocup Soccer using Convolutional Neural Networks. RoboCup Int. Symposium. July 2016
3. Iandola, F.N., Moskewicz, M.W., Ashraf, K., Han, S., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and<1mb modelsize. CoRR (2016), http://arxiv.org/abs/1602.07360
4. Hadjis, S., Abuzaid, F., Zhang, C., Ré, C.: Caffe Con Troll: Shallow Ideas to Speed Up Deep Learning. In: Proceedings of the Fourth Workshop on Data Analytics in the Cloud
5. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks, ECCV, 2016
6. Çatalyürek, U.V., Aykanat, C., Uçar, B.: On two-dimensional sparse matrix partitioning: Models, methods, and a recipe. SIAM J. Sci. Comput. 32(2), 656–683(Feb 2010)
7. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1–9, 2015.
8. Lin, M., Chen, Q., Yan, S.: Network in network. CoRR (2013), http://arxiv.org/abs/1312.4400
9. Paszke, A., Chaurasia, A., Kim, S., Culurciello, E.: Enet: A deep neural network architecture for real-time semantic segmentation. CoRR (2016), http://arxiv.org/abs/1606.02147
10. Scherer, D., Müller, A., Behnke, S.: Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition, pp. 92–101. SpringerBerlin Heidelberg, Berlin, Heidelberg (2010)
11. Röfer, T., Laue, T., Kuball, J., Lübken, A., Maaβ, F., Müller, J., Post, L., Richter-Klug, J., Schulz, P., Stolpmann, A., Stöwing, A., Thielke, F.: B-Human team report and code release 2016 (2016), only available online: http://www.b-human.de/downloads/publications/2016/coderelease2016.pdf
12. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093 (2014)
13. Abadi, M., Agarwal, A., et al.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), http://tensorflow.org/, software available from tensorflow.org
14. Redmon, J.: Darknet: Open source neural networks in c. http://pjreddie.com/ darknet/ (2013–2016)
15. Mathieu, M., Henaff, M., LeCun, Y.: Fast training of convolutional networks through ffts, arXiv preprint arXiv:1312.5851, 2013.
16. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up Convolutional Neural Networks with Low Rank Expansions. arXiv:1405.3866 [cs.CV]
17. Liu, B., Wang, M., Foroosh, H., Tappen, M., Penksy, M.: Sparse Convolutional Neural Networks, 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 806-814.
18. Han, S., Mao, H., Dally, W.J.: Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding (ICLR'16, best paper award)