Vision-based detection of aircrafts and UAVs

THÈSE N^{O} 7589 (2017)

PRÉSENTÉE LE 5 MAI 2017

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

LABORATOIRE DE VISION PAR ORDINATEUR

PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Artem ROZANTSEV

acceptée sur proposition du jury:

Dr D. Gillet, président du jury
Prof. P. Fua, Prof. V. Lepetit, directeurs de thèse
Prof. A. Zisserman, rapporteur
Dr H. Jégou, rapporteur
Prof. B. Merminod, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2017

Acknowledgements

There are many people, without whom this PhD thesis will not be possible. I would like to first thank Pascal Fua, for the great opportunity to join his Computer Vision Laboratory at EPFL. I am very thankful for the experience that I have acquired during the years of my PhD. I would also like to thank Vincent Lepetit for his time, patience and helpfulness especially in the beginning of my PhD, when I had little knowledge about computer vision and lots of questions. Further, I am very thankful to Mathieu Salzmann for the useful discussions that helped me during the last year of my studies. I am also grateful to the committee members, Denis Gillet, Andrew Zissermann, Hervé Jégou and Bertrand Merminod for accepting to evaluate this thesis.

I would also like to give thanks to all the present and past members of CVLab: Timur, Ksenia, Agata, Carlos, Pierre, Jan, Roger, Alberto, Isinsu, Andrii, Amos, Bugra, Amaury, Sandro, Anne, Pablo, Eduard and Kwang for the great time I have spent in the lab. In particular I would like to thank my officemate Timur for being a great source of fun stories of his adventures in Russia, US and other countries. I am also very grateful to Horesh and the whole playfulvision (now SecondSpectrum) team, with whom I visited Italy and Poland and watched (filmed) different volleyball tournaments with world-class teams competing with each other. These trips were really fun and I got 'in the field' computer vision experience. I would like to give special thanks to Arianne and Josiane for their kindness and assistance. I want to particularly thank Arianne for the support and creativeness in helping me to find solutions to any problem that I encountered.

I would further like to thank Rafah and Ajay with whom we started our PhD journey, shared a lot of fun and exciting moments, visited different countries, sang in the karaoke bars and who supported me during tough times. I also want to thank all my friends at EPFL: Roman, Marwa, Farah, Silvia, Katerina, Jose, Fay, Jean, Ersi, Beril, Pinar, Elena, Helen, Nico and who made this PhD experience extremely entertaining with boat trips, concerts, hikes, ski weekends and parties. I would not like Lausanne as much as I do now, without all these people. I am also very grateful to Dima (known as Dimaleks) for showing me this opportunity of doing PhD at EPFL and for amazing skiing and surfing trips.

I would like then to give thanks to my Lausanne friends. First, I want to thank my volleyball teams with whom we played in competitive championships. In particular I would like to thank Laetitia and Max for bringing in so much fun in the trainings, for wine tasting, skiing trips and great summer beach volley. I would also like to thank the whole Moschimö team with whom we went the whole path from almost being kicked out of the league to reaching the finals and getting the fifth place among the top mixed teams in Canton Vaud (Switzerland). Further, I am

also very grateful to Fred and the whole Powerball team for great time spent at trainings and matches. Last but not least I would like to thank Simone and the IC team, who gave me the opportunity to support the I&C department in the EPFL competition. Second, I want to thank Katia and Franziska Perrollaz, from whom I rented the studio, where I lived for almost all my PhD for the warmth, support and readiness to solve any problem that I encountered.

I am also very thankful to Renata for bringing a lot of love, happiness, smiles, and creativeness to my life, for the great trips and amazing cocktails, for her kindness, support and all the exciting moments that we share. I would also like to thank Kostya and Lena for their support, help in various situations and for our amazing trips in Italy, the UK and Finland. Finally, I would like to give special thanks to my parents for their unconditional support in the toughest times, for their love, kindness and readiness to help in all possible and impossible situations. Without them this PhD will be absolutely impossible.

Abstract

Unmanned Aerial Vehicles are becoming increasingly popular for a broad variety of tasks ranging from aerial imagery to objects delivery. With the expansion of the areas, where drones can be efficiently used, the collision risk with other flying objects increases. Avoiding such collisions would be a relatively easy task, if all the aircrafts in the neighboring airspace could communicate with each other and share their location information. However, it is often the case that either location information is unavailable (e.g. flying in GPS-denied environments) or communication is not possible (e.g. different communication channels or non-cooperative flight scenario). To ensure flight safety in this kind of situations drones need a way to autonomously detect other objects that are intruding the neighboring airspace. Visual-based collision avoidance is of particular interest as cameras generally consume less power and are more lightweight than active sensor alternatives such as radars and lasers. We have therefore developed a set of increasingly sophisticated algorithms to provide drones with a visual collision avoidance capability.

First, we present a novel method for detecting flying objects such as drones and planes that occupy a small part of the camera field of view, possibly move in front of complex backgrounds, and are filmed by a moving camera. In order to be solved this problem requires combining motion and appearance information, as neither of the two alone is capable of providing reliable enough detections. We therefore propose a machine learning technique that operates on spatio-temporal cubes of image intensities where individual patches are aligned using an object-centric regression-based motion stabilization algorithm.

Second, in order to reduce the need to collect a large training dataset and to manual annotate it, we introduce a way to generate realistic synthetic images. Given only a small set of real examples and a coarse 3D model of the object, synthetic data can be generated in arbitrary quantities and further used to supplement real examples for training a detector. The key ingredient of our method is that the synthetically generated images need to be as close as possible to the real ones not in terms of image quality, but according to the features, used by a machine learning algorithm.

Third, though the aforementioned approach yields a substantial increase in performance when using Adaboost and DPM detectors, it does not generalize well to Convolutional Neural Networks, which have become the state-of-the-art. This happens because, as we add more and more synthetic data, the CNNs begin to overfit to the synthetic images at the expense of the real ones. We therefore propose a novel deep domain adaptation technique that allows efficiently combining real and synthetic images without overfitting to either of the two. While most of the adaptation techniques aim at learning features that are invariant to the possible difference of the images,

coming from different sources (real and synthetic). Unlike those methods, we suggest modeling this difference with a special two-stream architecture. We evaluate our approach on three different datasets and show its effectiveness for various classification and regression tasks.

Key words: Computer vision, unmanned aerial vehicles, object detection, motion compensation, synthetic data generation, machine learning, deep learning, domain adaptation.

Résumé

Les véhicules aériens sans pilote sont de plus en plus populaires pour une grande variété de tâches allant de l'imagerie aérienne à la livraison d'objets. Avec l'expansion des domaines où les drones peuvent être utilisés efficacement, le risque de collisions avec d'autres objets volants augmente. Il serait relativement facile de prévenir ces collisions, à condition que tous les aéronefs dans un espace aérien voisin communiquent entre eux et partagent leurs informations de localisation. Cependant, il existe un grand nombre de cas où les informations d'emplacement ne sont pas disponibles (par exemple, vol dans des environnements privés de GPS) ou la communication n'est pas possible (par exemple, différents canaux de communication ou scénario de vol non coopératif). Pour assurer la sécurité des vols dans ce genre de situation, les drones ont besoin d'un moyen de détecter de façon autonome d'autres objets qui entrent dans l'espace aérien voisin. L'évitement de collision visuel est d'un intérêt particulier car les caméras consomment généralement moins d'énergie et sont plus légères que les alternatives de capteurs actifs comme le radar et le laser. A la lumière de ceci, nous présentons trois algorithmes, dont le premier est une technique d'apprentissage capable de détecter des objets volants basés sur une seule caméra embarquée. Les deux autres permettent de réduire la taille de l'ensemble de données nécessaires à l'entrainement d'un détecteur, car la collecte de vidéos réelles et leur annotation manuelle est une procédure complexe et fastidieuse.

D'abord, nous présentons une nouvelle méthode pour détecter des objets volants tels que des drones et des avions qui occupent une petite partie du champ de vision de la caméra, se déplacent éventuellement devant des fonds complexes et sont filmés par une caméra mobile. Pour résoudre ce problème, il faut combiner des informations de mouvement et d'apparence, car aucune des deux seuls n'est capable de fournir des détections suffisamment fiables. Nous proposons donc une technique d'apprentissage qui opère sur des cubes spatio-temporels d'intensités d'image où les patchs individuels sont alignés à l'aide d'un algorithme de stabilisation de mouvement par régression.

Deuxièmement, afin de réduire le besoin de collecter un grand ensemble de données d'entraînement et son annotation manuelle, nous introduisons un moyen de générer des images synthétiques réalistes. Basé sur un petit ensemble d'exemples réels et un modèle 3D grossier de l'objet, les données synthétiques peuvent être générées en quantités arbitraires et utilisées pour compléter des exemples réels dans l'entrainement d'un détecteur. L'ingrédient clé de notre méthode est que les images générées par cette synthèse doivent être aussi proches que possible des vraies images, non en termes de qualité d'image, mais selon les caractéristiques qui sont utilisées en entrée par notre algorithme d'apprentissage.

Troisièmement, bien que l'approche mentionnée ci-dessus montre une augmentation substantielle de la performance pour les détecteurs Adaboost et DPM, elle n'apporte pas beaucoup d'amélioration pour les réseaux neuronaux convolutifs. Cela se produit en raison de la spécialisation de ce dernier à un grand nombre d'images synthétiques que nous ajoutons à l'ensemble d'entraînement et à son manque de généralisation aux images réelles. Nous proposons donc une nouvelle technique d'adaptation de domaine qui permet de combiner efficacement des images réelles et synthétiques sans sur-adaptation à l'une ou l'autre. Alors que la plupart des techniques d'adaptation visent des caractéristiques d'apprentissage qui sont invariantes à la différence possible des images, venant de sources différentes (réelles et synthétiques). Contrairement à ces méthodes, nous suggérons de modéliser cette différence avec une architecture spéciale à deux flux. Nous évaluons notre approche sur trois ensembles de données différents et montrons son efficacité pour diverses tâches de classification et de régression.

Mots clés : vision par ordinateur, véhicules aériens sans pilote, détection d'objets, compensation de mouvement, génération de données synthétiques, apprentissage automatique, adaptation de domaine, machine learning, deep learning.

Contents

List of Figures

List of Tables

Introduction

1 Motivation

We are headed for a world in which the skies are occupied not only by birds and planes but also by Unmanned Aerial Vehicles (UAVs) [1, 2], or simply 'drones'. Their role is becoming increasingly important in a broad variety of tasks including surveillance [3, 4], environment mapping [5], goods [6] and medical supplies [7] delivery. Some of these drones will be able to communicate with each other to avoid collisions using for example the Global Positioning System (GPS) and radio communication, but not all of them. Furthermore, systems like GPS are only reliable when UAVs are flying in the open areas and are useless indoors. Therefore, drones need a way to autonomously navigate and avoid collisions with other objects that may appear in the neighboring airspace. Among the broad variety of possible sensors, which can be used to solve this task, cameras are the most promising ones, as they are cheaper and consume less power then alternatives like radars or lasers. This stresses the need for vision-based drone collision avoidance methods to be developed.

In general there are two classes of objects that may pose threat to a UAV: static and moving ones. Preventing collision with static obstacles has been thoroughly studied in the world of flying machines as a sub-problem of a more general task, which is accurate position estimation and navigation from single or multiple cameras [8, 9, 10, 11, 12, 13, 14, 15]. The second class of objects, on the other hand, has received far less attention [16, 17]. It is nevertheless becoming increasingly important given the recent explosive growth of interest in consumer UAVs. In this thesis we, therefore, focus on developing inexpensive algorithms for visual-based moving objects detection from an on-board camera.

A related problem of detecting moving objects from a moving camera has been well-studied and successfully tackled in the automotive world. Various algorithms have been proposed for human [18, 19, 20] and vehicle [21, 22, 23] detection. This in turn gave rise to a number of commercial products [24, 25] that are currently used as driving assistants, with the ultimate goal of completely autonomous driving [26, 27].

However, it is not possible to simply extend these algorithms to the world of aircrafts and drones, as flying object detection poses several unique challenges that are not present in pedestrian and

(a) (b)

Figure 1: Detecting a small flying object against a complex moving background. **(a)** It is almost invisible to the human eye and hard to detect from a single image. **(b)** Yet, our algorithm can find it by using appearance and motion cues.

car detection cases. In the remainder of this chapter we first discuss these challenges. We then introduce our approach to solving them and present an outline of the thesis.

2 Challenges

2.1 Fast motions and small sizes

One of the major obstacles to reliably detecting drones is the high speed and complex motions that they can undertake. This produces large amounts of motion blur in the parts of the image where the drones are seen. Such motion blur can only be neglected when UAVs fly close to the camera of the observing drone, as enough appearance information can be extracted from the video frame. However, due to the specifics of collision avoidance problem, drones need to be detected in advance, when they are still at a safe distance, far away from the observing UAV. This increases the complexity of the detection task, as motion blur now has significant influence on the drone appearance. In general there are three types of approaches that can be used to detect small fast-moving objects. They can be appearance-based [28, 18, 29, 30], motion-based [31, 32, 33, 34], or hybrid [19, 35].

Appearance-based algorithms do not achieve good accuracy for our problem, because they rely on the information extracted just from one frame of the video sequence. As it was mentioned before, due to severe motion blur and small object sizes appearance is not enough for reliable detection. Thus, motion information should also be taken into account.

On the other hand, purely motion-based methods [31, 32, 33, 34] are prone to errors, as images acquired by a moving camera are noisy, which complicates the task of detecting small objects and feature backgrounds that are hard to stabilize because they are non-planar and change rapidly. More importantly these methods will give no positive response for the observed drone A, when it is hovering above the ground, or is on a collision course with the observing drone B, as depicted by Fig. 2. In both of these cases drone A will be static in the camera view of drone B. Furthermore, since there may be other moving objects in the scene, such as a person in the left most image in Fig. 1(a), motion by itself is not enough and appearance should be also taken into account.

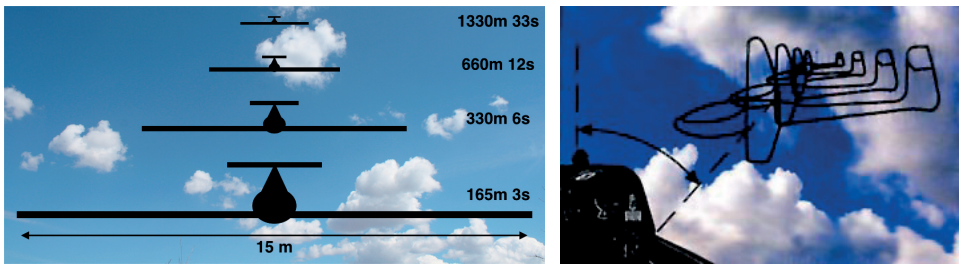


Figure 2: Collision courses. [LEFT] The apparent size of a standard glider and its 15 m wingspan flying towards another aircraft at a relatively slow speed (100 km/h) is very small 33s before impact, but the glider completely fills the field of view only half a minute later, 3s before impact. [RIGHT] An aircraft on a collision course is seen in a constant direction but its apparent size grows, slowly at first and then faster.

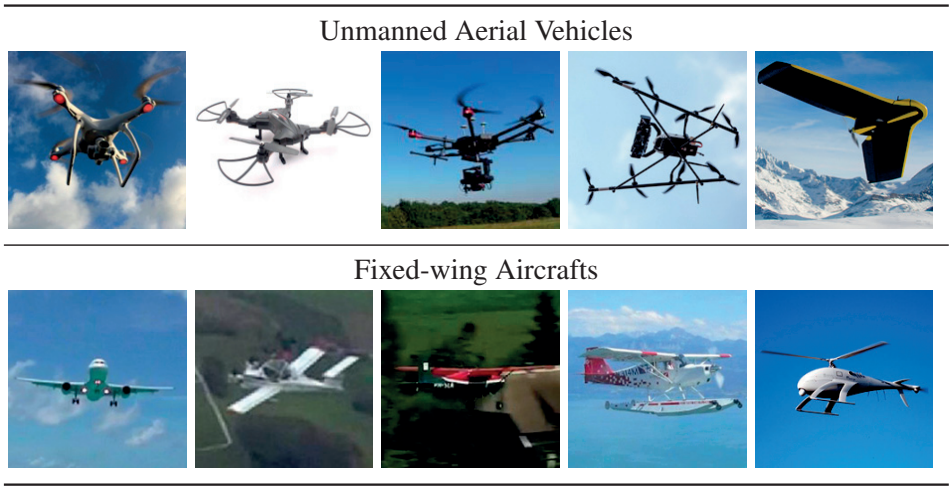


Figure 3: Flying vehicles appearance variation. [TOP] Different types of Unmanned Aerial Vehicles. [BOTTOM] Various fixed-wing aircrafts and a helicopter seen from different perspectives and in front of different backgrounds.

Finally, hybrid methods combine motion and appearance information, but unfortunately, most of them [19, 35] were developed primarily for pedestrian detection. As people typically occupy a relatively large portion of the camera's field of view and travel at moderate speeds, these methods are not directly applicable to the task of small fast-moving object detection.

2.2 In-class appearance variation

Another challenge that is common to all detection problems, is the extreme variety of shapes of objects that need to be detected by the algorithm. Fig. 3 depicts a few examples of shapes and appearances of various aircrafts and UAVs in different environments.

For a broad variety of object categories, such as cars, animals, flowers, people, etc. there exist

publicly available datasets [36, 37, 38] with a large number of examples of these objects seen from different perspectives, in various environments and lighting conditions. In our case, however, no such dataset exist and building one is a very time consuming process, because it is hard to collect real-world footage and then to manually annotate it. Furthermore, the resulting collection of images/videos need to be constantly updated as, over time, new drone models will be introduced, potentially with significant variation in shape from the ones that already exist in the database.

One common way of tackling the problem of limited training data is to augment the training set with perturbed versions of real images. This introduces variability, but does not solve the problem of having a limited number of poses in which the object is seen. To address this issue a number of methods [39, 40, 41, 42] propose supplementing a small set of real examples with realistic synthetic images. The advantage of this category of approaches is that synthetic data can be relatively easily generated in large quantities and no additional manual annotation is required. Most of the existing methods, however, focus on generating visually appealing images, which may not exhibit all the properties of the real ones, which is crucial for effective detector training. Furthermore, to achieve their top performance these methods typically require a large number of parameters to be manually adjusted.

3 Contributions

We now briefly list our contributions. We start with a learning-based technique to tackle the small fast moving object detection problem. We then introduce two complementary approaches, where the first one aims at generating realistic synthetic images and the second one allows effectively combining them with real data to achieve state-of-the-art performance with Deep Neural Networks, which have recently proved to be very successful in solving classification and detection tasks [43, 44].

3.1 UAV detection

We propose a novel method that combines appearance and motion features for training a detector. To this end, our approach takes as an input not just a single image, but rather a short sequence of consecutive video frames. As it is often done in object detection we use a sliding window method to locate the drone in the image, however, instead of using image patches we work with spatio-temporal cubes of image intensities that allow capturing both appearance and motion information. This, however, significantly increases the feature variation, as similar objects traveling in different directions will create very different spatio-temporal feature representations. Thus, in order to simplify the work for a detector we need to decrease this diversity. We tackle this problem with an object-centric motion compensation algorithm that aligns individual patches of the spatio-temporal cubes so that the drone is always in the center. Apart from the reduction of feature diversity, this makes the detector more focused on the drone itself and less dependent on the background. As shown in Fig. 1(b) our approach enables the successful detection of drones

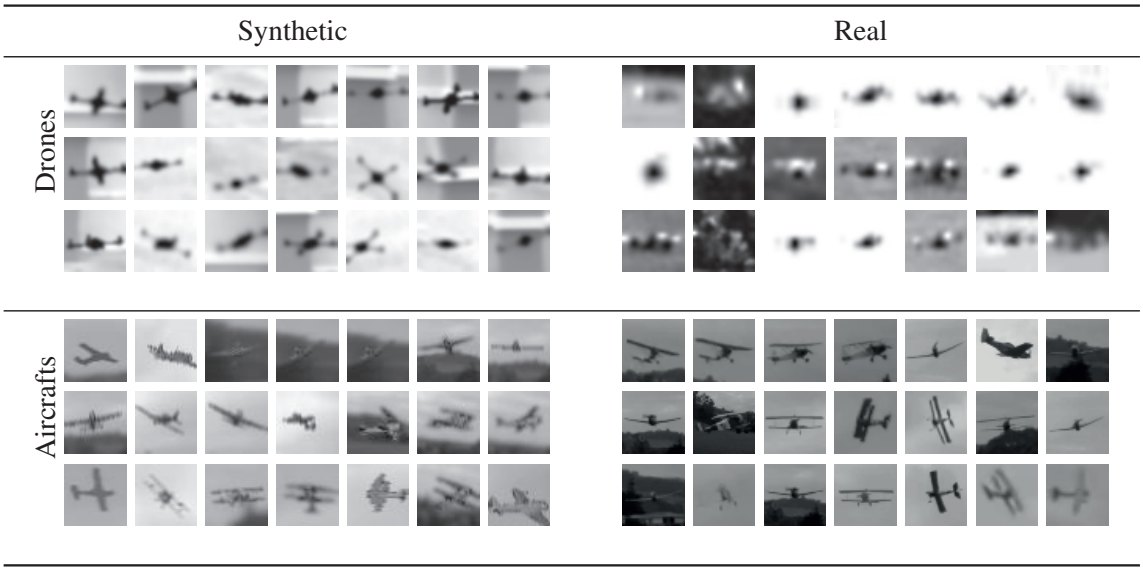


Figure 4: Synthetic and real examples of drones and fixed-wing aircrafts.

in complicated outdoor environments. Part of this work appears in [45, 46].

As the task of UAV detection is relatively new, there are no publicly available datasets for the evaluation of our approach. Therefore, we have collected a couple of datasets that reflect various challenges common to the problem of detecting drones in outdoor environments. The first dataset comprises videos from the camera mounted on a drone that records other UAVs flying around. The second dataset is a collection of videos from the publicly available sources, recorded by a hand-held camera and filming different types of aircrafts flying in various environments.

3.2 Synthetic data generation

We then move to the problem of increasing the diversity of drone shapes and poses that are present in the training dataset. This issue is critical for training an unbiased detector, capable of reliably working in outdoor environments. As mentioned before, collecting and annotating a large dataset with different types of objects in various environments and lighting conditions is very time consuming. We, therefore, propose to use synthetic data to augment the small number of real images of drones and aircrafts that we have.

Starting from a small set of real images, our algorithm estimates the rendering parameters required to generate synthetic images that are similar to the real ones, given a coarse 3D model of the target object. These parameters can then be reused to generate an unlimited number of training images of the object of interest in arbitrary 3D poses, which can further be added to the training set to increase classification accuracy. Our key insight is that the synthetically generated images should be similar to the real ones, not in terms of image quality, but rather in terms of features used during the classifier training. We show that generating images in such a way yields significantly

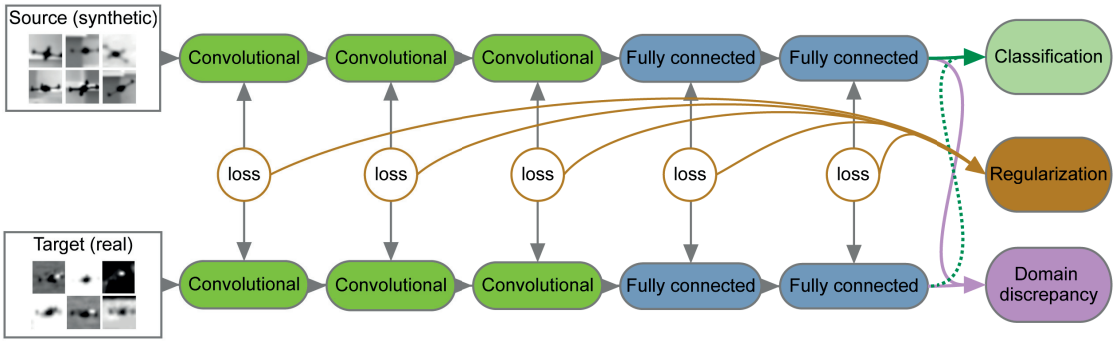


Figure 5: Our two-stream architecture that allows effectively combining real and synthetic images for accuracy improvement. The optimization is done by simultaneously minimizing the loss function that consists of the classification, regularization and domain discrepancy components. Briefly, *Classification* loss is computed based on available labeled images. The *Regularization* loss term penalizes the large differences between the corresponding parameters of the networks (streams). Finally the *Domain discrepancy* loss is required to make the final feature representation of real and synthetic data similar to each other, as we are using the same classification layer for both of the streams.

better performances than simply perturbing real images [47, 48] or even synthesizing images in such a way that they look realistic [41], as is often done when only limited amounts of training data are available. Fig. 4 shows some examples of real and synthetic images generated by our framework. Part of this work appears in [49].

3.3 Domain Adaptation for Deep Neural Networks

The aforementioned approach allows us to arbitrary increase the size of the training set with very few real examples. In general this results in having a stronger classifier, that is less sensitive to differences in flying object shapes and view-points from which the drone can be seen. This method, however, still suffers from one drawback, which is common for all learning-based approaches that rely on a vast amount of synthetic examples during training. They all tend to overfit to the synthetic images at the expense of the real ones. As a result, they may be over-influenced by any bias that might have been introduced by the synthetic data generation algorithm. This is particularly true for Convolutional Neural Networks (CNNs), which reach state-of-the-art [30, 43, 50, 44] thanks to their ability to fit the data, which makes them vulnerable to bias.

We developed a technique that lets us leverage the large amount of synthetic images to train a UAV detector. Our method not only avoids overfitting to either real or synthetic data, but also significantly reduces the dependency on real data collection, as one only needs to collect 5-10% of labeled training samples to achieve good results. Part of this work appears in [51].

Our approach, falls into the category of Domain Adaptation algorithms, which aim at transferring the knowledge from one domain to the other. This essentially means that a classifier trained on images of a given type can be efficiently used to classify images of another. In the context of Deep Learning most of these methods [52, 53, 54, 55, 56, 57] aim at finding domain invariant feature representation, which will essentially lead to an effective classifier transfer from one domain to another. Unlike the existing work, however, we propose to model the difference between domains and use it to boost the resulting classification accuracy.

To do so we propose a novel two-stream architecture, where each of the *streams* corresponds to a convolutional neural network of the same structure. Each of these networks (streams) operates on either synthetic or real data. For simplicity we refer to the first one as stream S and the latter as stream T. The parameters θ^S and θ^T of these streams, however, are not completely independent but are related to each other, so that $\theta^T \simeq \phi\left(\theta^S\right)$, where $\phi(\cdot)$ is a linear function. This allows stream T to learn the feature representation that is optimal for classifying real images. The learning process is, however, regularized by stream S that operates just on synthetic samples, which have a significantly larger variation in pose and appearance. Furthermore, the linear function $\phi(\cdot)$ makes the whole system flexible and allows modeling such domain shifts as illumination or contrast changes. Fig. 5 depicts the deep learning architecture that we use.

4 Outline

We begin by introducing our approach to drone detection with object-centric motion compensation in Chapter 1. We further present two datasets that we have collected for the evaluation of our method and show that due to effective leveraging of motion and appearance information, our approach outperforms state-of-the-art algorithms for the small fast moving object detection problem.

In Chapter 2 we tackle the problem of augmenting the small set of real images with synthetic data and introduce a way to generate synthetic images so that they are close to the real ones from the perspective of the detector. This lets us increase the spectrum of UAV poses that are present in the training dataset leading to a stronger detector.

In Chapter 3, we investigate, how the aforementioned approach can be improved in the context of Deep Neural Networks to avoid overfitting to a large number of synthetic samples. We introduce our domain adaptation algorithm that instead of learning a set of domain invariant features, uses the difference between real and synthetic images to improve classification accuracy.

Finally, we close this thesis in Chapter 4 with concluding remarks and the future work discussion.

1 Flying Objects Detection

In this chapter we present a novel approach for detecting flying objects such as UAVs and aircrafts when they occupy a small portion of the field of view, possibly moving against complex backgrounds, and are filmed by a camera that itself moves. Solving such a difficult problem requires combining both appearance and motion cues. To this end we propose a regression-based approach for object-centric motion stabilization of image patches that allows us to achieve effective classification on spatio-temporal image cubes and outperform state-of-the-art techniques.

1.1 Introduction

One of the main challenges for outdoor drone detection is their fast speeds and small sizes, which make the appearance-based methods impractical. As mentioned before purely motion-based approaches also do not allow us to achieve desired accuracy, as they are incapable of detecting hovering UAVs and may also detect other moving objects in the scene, such as humans or cars. Therefore, we propose solving the UAV detection problem by combining appearance and motion information inside the detector.

More formally, we detect whether an object of interest is present and constitutes danger by classifying 3D descriptors computed from spatio-temporal image cubes of image intensities, which we will refer as st-cubes for simplicity. The latter are formed by stacking motion-stabilized image windows over several consecutive frames, which give more information than using a single image. What makes this approach both practical and effective is a regression-based motion-stabilization algorithm. Unlike those relying on optical flow, it remains effective even when the shape of the object to be detected is blurry or barely visible, as illustrated by Fig. 1.1. This arises from the fact that learning-based motion compensation focuses on the object and is more resistant to complicated backgrounds, compared to the optical flow method as shown in Fig. 1.1.

St-cubes have been routinely used for action recognition and pose estimation purposes [58, 59, 60, 61] using a monocular camera. By contrast, most current detection algorithms work either on a single frame, or by estimating the optical flow from consecutive frames. Our approach can

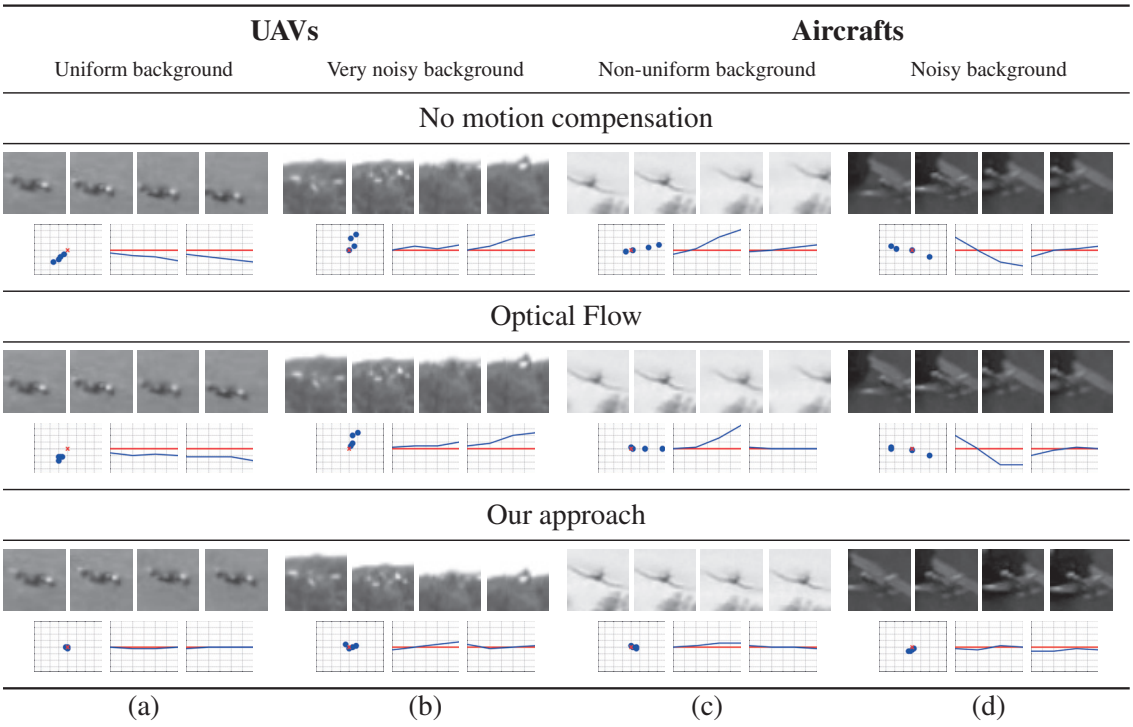


Figure 1.1: Motion compensation for four different st-cubes of flying objects seen against different backgrounds. [TOP] For each one, we show four consecutive patches before motion stabilization. In the leftmost plot below the patches, the blue dots denote the location of the true center of the drone and the red cross is the patch center over time. The other two plots depict the x and y deviations of the drone center with respect to the patch center. [MIDDLE] The same four st-cubes and corresponding graphs after motion compensation using an optical flow approach, as suggested by [35]. [BOTTOM] The same four st-cubes and corresponding graphs after motion compensation using our approach.

therefore be seen as a way to combine both the appearance and motion information to achieve effective detection in a very challenging context. In our experiments we show that this method allows to achieve higher accuracy, comparing to either appearance or motion-based methods individually.

1.2 Related Work

Approaches for detecting moving objects can be classified into three main categories: those that rely on appearance in individual frames, those that rely primarily on motion information across frames, and those that combine the two. We briefly review all three types in this section. In the results section, we will demonstrate that we can outperform state-of-the-art representatives of each class.

Appearance-based methods rely on Machine Learning and have proved to be powerful even in the presence of complex lighting variations or cluttered background. They are typically based on Deformable Part Models (DPM) [29], Convolutional Neural Networks (CNN) [62], or Random Forests [28]. Among them the Aggregate Channel Features (ACF) [18] algorithm is considered as one of the best.

These approaches work best when the target objects are sufficiently large and clearly visible in individual images, which is often not the case in our applications. For example, in the images of Fig. 1, the object is small and it is almost impossible to make out from the background without motion cues.

Motion-based approaches can themselves be subdivided into two subclasses. The first comprises those that rely on background subtraction [63, 64, 31, 32] and determine objects as groups of pixels that are different from the background. The second includes those that depend on optical flow [33, 65, 34].

Background subtraction works best when the camera is static or its motion is small enough to be easily compensated for, which is not the case for the on-board camera of a fast moving aircraft.

Flow-based methods are more reliable in such situations but still critically dependent on the quality of the flow vectors, which tends to be low when the target objects are small and blurry. Some methods combine both optical flow and background subtraction algorithms [66, 67]. However, in our case there may be motion in different parts of the images, for example people or tree tops. Thus motion information is not enough for reliable flying object detection. Other methods that combine optical flow and background subtraction, such as [68, 69, 70, 71] still critically depend on optical flow, which is often estimated with [34] and thus may suffer from the low quality of the flow vectors. In addition to optical flow dependence, [70] makes an assumption that camera motion is translational, which is violated in aerial videos.

Hybrid approaches combine information about object appearance and motion patterns and are therefore the closest in spirit to what we propose. For example, in [19], histograms of flow vectors are used as features in conjunction with more standard appearance features and are fed to a statistical learning method. This approach was refined in [35] by first aligning the patches to compensate for motion and then using the differences of the frames, which may or may not be consecutive, as additional features. The alignment relies on the Lucas-Kanade optical flow algorithm [65]. The resulting algorithm works very well for pedestrian detection and outperforms most of the single-frame methods. However, when the target objects become smaller and harder to see, the flow estimates become unreliable and this approach, like the purely flow-based ones, becomes less effective.

1.3 Detection Framework

Our detection pipeline is illustrated by Fig. 1.2 and comprises the following steps:

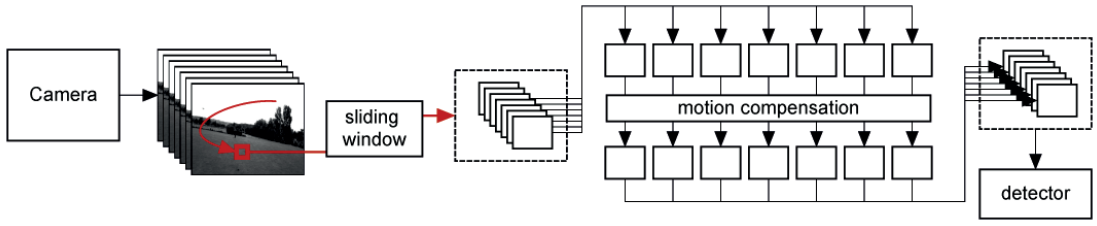


Figure 1.2: Object detection pipeline with st-cubes and motion compensation. Provided a set of video frames from the camera, we use a multi-scale sliding window approach to extract st-cubes. We than process every patch of the st-cube to compensate for the motion of the aircraft and then run the detector. (best seen in color)

- Divide the video sequence into N-frame overlapping temporal slices. The larger the overlap is, the higher the precision but only up to a point. Our experiments show that making the overlap more than 50% increases computation time without improving performance. Thus, 50% is what we used.

- Build st-cubes from each slice using a sliding window approach, independently at each scale.

- Apply our motion compensation algorithm to the patches of each of the st-cubes to create stabilized st-cubes.

- Classify each st-cube as containing an object of interest or not.

- Since each scale has been processed independently, we perform non-maximum suppression in scale space. If there are several detections for the same spatial location at different scales, we only retain the highest-scoring one. As an alternative to this simple scheme, we have developed a more sophisticated learning-based one, which we discuss in more details in Section 1.6.4.

In this section, we introduce two separate approaches—one based on boosted trees, the other one on Convolutional Neural Networks—to deciding whether or not an st-cube contains a target object and will compare their respective performance in Section 1.5. We will discuss motion compensation in Section 1.4.

More specifically, we want to train a classifier that takes as input st-cubes such as those depicted by Fig. 1.3 and returns 1 or -1, depending on the presence or absence of a flying object. Let (s_x, s_y, s_t) be the size of our st-cubes. For training purposes, we use a dataset of pairs $(b_i, y_i), i \in [1, N]$, where $b_i \in \mathbf{R}^{s_x \times s_y \times s_t}$ is an st-cube, in other words s_t image patches of resolution $s_x \times s_y$ pixels. Label $y_i \in \{-1, 1\}$ indicates whether or not a target object is present.

1.3.1 3D HoG with Gradient Boost

The first approach we tested relies on boosted trees [72] to learn a classifier $\psi(\cdot)$ of the the form $\psi(b) = \Sigma_{j=1}^{H} \alpha_j h_j(b)$, where $\alpha_{j=1..H}$ are real valued weights, $b \in \mathbf{R}^{s_x \times s_y \times s_t}$ is the input st-cube, $h_j : \mathbf{R}^{s_x \times s_y \times s_t} \rightarrow \mathbf{R}$ are weak learners, and H is the number of selected weak learners, which controls the complexity of the classifier. The α's and h's are learned in a greedy manner, using the Gradient Boost algorithm [72], which can be seen as an extension of the classic AdaBoost to real-valued weak learners and more general loss functions.

In standard Gradient Boost fashion, we take our weak learners to be regression trees $h_j(b) = T(\theta_j, \text{HoG3D}(b))$, where θ_j denotes the tree parameters and HoG3D(b), the 3-dimensional Histograms of Gradients (HoG3D) computed for b. HoG3D was introduced in [60], and can be seen as an extension of the standard HoG [73] with an additional temporal dimension. It is fast to compute and proved to be robust to illumination changes in many applications, and allows us to combine appearance and motion efficiently.

At each iteration j, the weak learner $h_j(\cdot)$ with the corresponding weight α_j is taken as the one that minimizes the exponential loss function:

$$(h_j(\cdot), \alpha_j) = \underset{h(\cdot),\alpha}{\operatorname{argmin}} \sum_{i=1}^{N} e^{-y(\psi_{j-1}(b_i) + \alpha h(b_i))} . \tag{1.1}$$

The tests in the nodes of the trees compare one coordinate of the HoG3D vector with a threshold, both selected during the optimization.

1.3.2 Convolutional Neural Networks

Since Convolutional Neural Networks (CNN) [47] have proved very successful in many detection problems, we have tested it as an alternative classification method. We use the architecture depicted by Fig. 1.4, which alternates convolutional layers and pooling layers. Convolutional layers use 3D linear filters while pooling layers apply max-pooling in 2D spatial regions only. The last layer is fully connected and outputs the probability that the input st-cube contains an object of interest. We use the hyperbolic tangent function as the non-linear operator [74].

We take the input of our CNN is a normalized st-cube

$$\eta = \frac{b - \mu(b)}{\sigma(b)} , \tag{1.2}$$

where $\mu(b)$ and $\sigma(b)$ are the mean and standard deviation of the pixel intensities in b, respectively. Note, that this normalization step is important because network parameters optimization fails to converge when using raw image intensities.

During training, we write the probability that an st-cube η contains an object of interest ($y = 1$) or

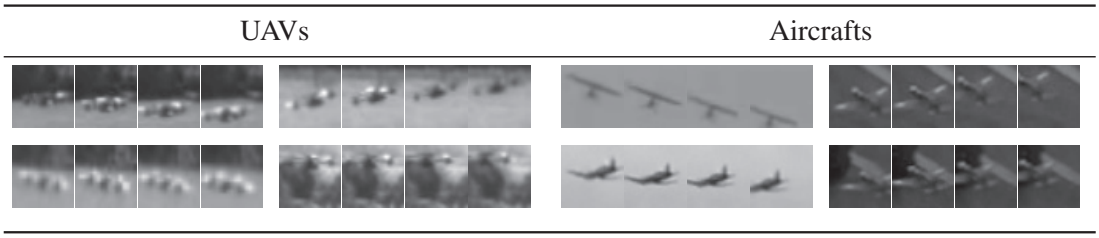


Figure 1.3: Sample st-cubes of the UAVs and aircrafts. Each row corresponds to a single st-cube and illustrates different possible motions that an aircraft could have.

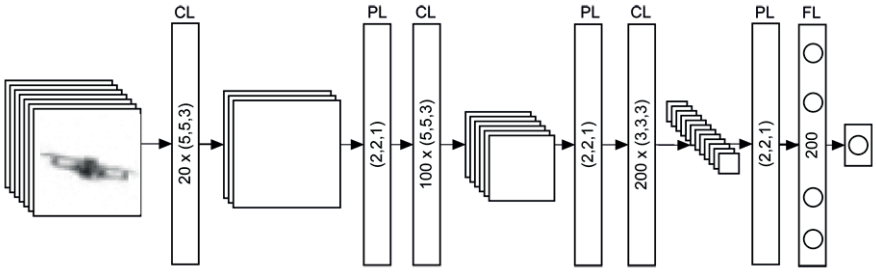


Figure 1.4: The structure of the Convolutional Neural Network, which we used for flying object detection. **CL**, **PL** and **FL** correspond to Convolution, Pooling, and Fully-connected layers respectively.

is a part of the background ($y = 0$) as

$$P(Y = y \mid \eta) = \frac{e^{\mathrm{CNN}(\eta)[y]}}{e^{\mathrm{CNN}(\eta)[0]} + e^{\mathrm{CNN}(\eta)[1]}}, y = \{0, 1\}, \tag{1.3}$$

where $\mathrm{CNN}(\eta)[y]$ denotes the classification score that the network predicts for η as being part of class y and $e^{(\cdot)}$ denotes the exponential function. We then minimize the negative log-likelihood

$$\mathscr{L}(W, bias) = -\sum_{k=1}^{N} \log P(Y = y_k \mid \eta_k) \tag{1.4}$$

with respect to the CNN parameters. Here (η_k, y_k) are pairs of normalized st-cubes and their corresponding labels from the training dataset, as defined in Section 1.3. To this end, we use the algorithm of [75] combined with Dropout [76] to improve generalization.

We tried many different network configurations, in terms of the number of filters per layer and the size of the filters. However, they all yield similar performance, which suggests that only minor improvements could be obtained by further tweaking the network. We also tried varying the dimensions of the st-cube. These variations have a more significant influence on performance, which will be evaluated in Section 1.5.

1.4 Motion Compensation

Neither of the two approaches to classifying st-cubes introduced in the previous section accounts for the fact that both the gradient orientations used to build the 3D HoG and the filter responses in the CNN case are biased by the global object motion. This makes the learning task much more difficult and we propose to use motion compensation to eliminate this problem. Motion compensation will allow us to accumulate visual evidence from multiple frames, without adding variation due to the object motion. We therefore aim at centering the target object, so that when present in an st-cube, it remains at the center of all its image patches.

More specifically, let I_t denote the t-th frame of the video sequence and (i, j) some pixel position in it. The st-cube $b_{i,j,t}$ is the 3D array of pixel intensities from images I_z with $z \in [t - s_t + 1, t]$ at image locations (k, l) with $k \in [i - s_x + 1, i]$ and $l \in [j - s_y + 1, j]$, as depicted by Fig. 1.3. Correcting for motion can be formulated as allowing patches $m_{i,j,z}, z \in [1, s_t]$ of the st-cube $b_{i,j,t}$ to shift horizontally and vertically in individual images.

In [35], these shifts are computed using optical flow information, which has been shown to be effective for pedestrians occupying a large fraction of the patch and moving relatively slowly from one frame to the next. However, as can be seen in Fig. 1.3, these assumptions do not hold in our case and we will show in Section 1.6 that this negatively impacts performance. To overcome this difficulty, we introduce instead a learning-based approach to compensate for motion and keep the object in the center of the $m_{i,j,z}$ patches of the st-cube even when the target object's appearance changes drastically.

More specifically, we treat motion compensation problem as a regression task: given a single image patch, we want to predict the 2D translation that best centers the target object. By rectifying all the image patches in an st-cube with their predicted translation, we can then align the images of the object of interest together.

1.4.1 Boosted tree-based regressors

One way to predict the translation for an input patch m, is to train two different boosted trees regressors [77] $\phi_x(m)$ and $\phi_y(m)$, one for each 2D direction (horizontal and vertical).

As for detection, we use regression trees $h_j(m) = T(\theta_j, \text{HoG}(m))$ as weak learners, where $\text{HoG}(m)$ denotes the Histograms of Oriented Gradients for patch m. The difference is that we minimize here a quadratic loss function instead of an exponential one

$$L(r, \phi_*(m)) = (r - \phi_*(m))^2, \tag{1.5}$$

where m is the input patch, r the corresponding expected 2D vector, and $\phi_*(m) = [\phi_x(m), \phi_y(m)]^\top$ the 2D vector predicted by the 2 regression trees.

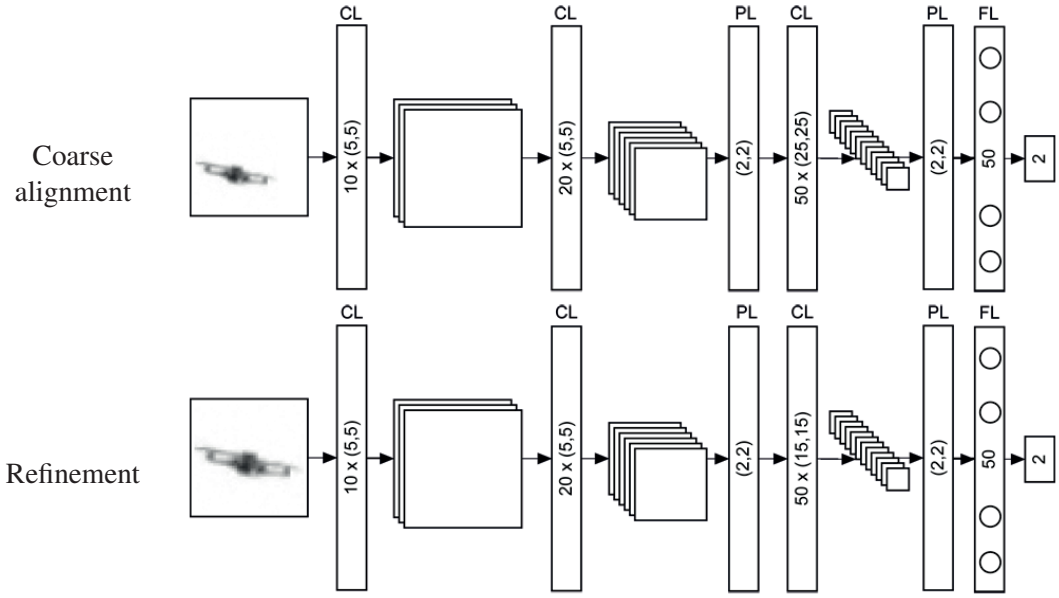


Figure 1.5: Structure of the CNNs used for motion compensation. [TOP] The first network uses extended patches to correct for the large displacements of the aircraft. [BOTTOM] The second network is applied after rectification by the motion predicted by the first network, and is designed to correct for the small motions.

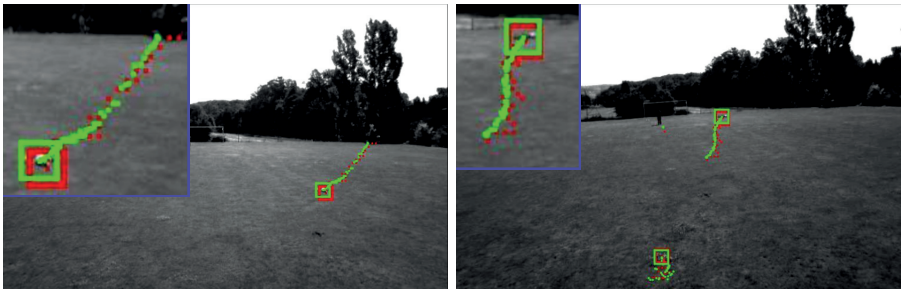


Figure 1.6: Combining multiple detections in several images of a video sequence. The red square and dots depict the positions of the original detection across the 50 frames preceding two different images. The green square and dots illustrate the position of the same detections after refinement. They are superposed and form much smoother trajectories. (best seen in color)

We then apply these regressors in an iterative way: we obtain a first estimate of the shift of the target object—if present—from the center of the patch. We translate it according to this estimate, and we re-apply the regressors. We iterate until both shift estimates drop to 0 or the algorithm reaches a preset number of iterations. In practice, 4 to 5 iterations are enough to achieve good accuracy.

(a) UAV dataset (b) Aircraft dataset

Figure 1.7: Sample image patches containing aircrafts or UAVs from our datasets.

1.4.2 CNN-based regressors

Another possible approach is to use a Convolutional Neural Network (CNN) to solve the regression task. CNNs are more flexible, as features are learned directly from the training data, in contrast to the hand-designed HoG features we need to use with our boosted tree-based regressors.

We trained two separate CNNs whose structure is depicted by Fig. 1.5. Note that there is no pooling layer after the first convolutional one. This is because pooling layers are typically used not only to reduce computational complexity but also to achieve invariance to small motions. In our case, such invariance would be counter-productive because these motions are precisely what we are trying to estimate. Furthermore, the computational complexity remains manageable even without the first pooling layer. We trained the first CNN using examples involving large 2D translations (coarse-CNN) and the second smaller ones (fine-CNN). In practice we use the latter to refine the predictions of the former. As when using boosted-trees, we use CNN-regressors iteratively until convergence, as described at the end of Section 1.4.1. We first correct for large displacements by applying several times coarse-CNN and we then apply fine-CNN, which is trained to compensate for small shifts of the object, for a couple more iterations.

In fact, we also tried training two different boosted-tree regressors such as those discussed in Section 1.4.1. Unlike in the case of the CNN regressors, it produced no significant improvement. This likely happens because our boosted trees motion compensation algorithm is based on HoG, where histograms are computed over the bins of fixed size. This, in fact, introduces invariance to small deviations of objects, which makes it hard to achieve high localization precision.

1.4.3 Motion Compensated st-cubes

Once the regressors have been trained, we use them to compensate for motion and build the st-cubes that we will use as input for classification, as depicted by Fig. 1.2. Fig. 1.1 illustrates several st-cubes of a drone from the testing dataset and after motion compensation, using either optical flow from [35] or our approach. Note that the latter tends to keep the target object much closer to the center, especially when the background is non-uniform and noisy or under lighting changes.

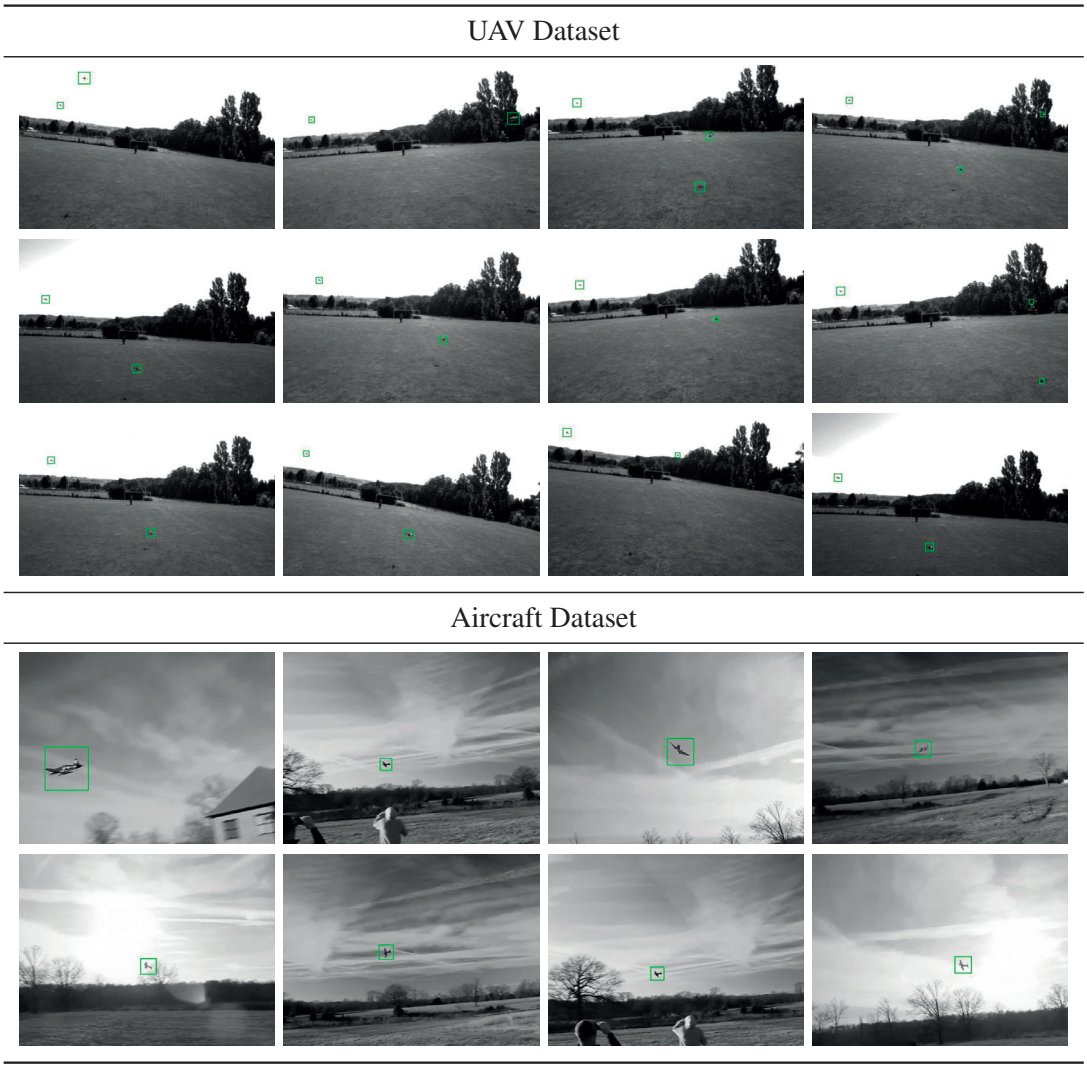


Figure 1.8: An object's apparent size can change enormously depending on its pose and distance to the camera. We therefore use a sliding window approach at different resolutions. The green boxes denote detections by our algorithm, which successfully handles background, lighting, scale, and pose changes.

Part of the difficulty in detecting fast moving flying objects is that they can appear anywhere in the 3D environment and that their apparent size can vary enormously. This makes it necessary to scan the whole image at different scales using a sliding window approach to avoid missing anything, which is computationally expensive.

Fortunately, our motion compensation scheme frees us from the need to evaluate every image position. When there is a target object, our algorithm automatically shifts the patch so it is in the center. As a result, instead of having to test windows centered at every pixel location, we only have to check non-overlapping ones because the algorithm will automatically shift their location to center the target object when one is present. This also makes its unnecessary to use heuristics

such as non-maximum suppression, as all the detections that arise from a single object will be shifted to the same position. The duplicates can therefore easily be removed, leaving us with just a single detection per object, as illustrated by Fig. 1.6.

As discussed in Section 1.3, we process each scale independently. We then perform non-maximum suppression in scale-space as a final step.

1.5 Designing the Optimal Approach

The two key components of our pipeline are motion compensation and classification of the st-cubes, both of which can be implemented using either CNNs or hand-designed features. In this section, we test the various possible combinations and justify the parameter choices we made for the final evaluation of our whole approach against several baselines, as described in Section 1.6.

Since the problem of detecting small flying objects has not yet received extensive attention from our community, there is not yet any standard dataset that can be used for testing purposes. We therefore built our own, one for UAVs and one for planes. We first describe them and then describe our testing protocol and the metrics we used for evaluation purposes. Finally, we perform the above-mentioned comparisons and demonstrate that the best results are obtained by using the CNN approach of Section 1.4.2 for motion compensation and the HoG3D descriptors of Section 1.3.1 for actual detection.

1.5.1 Datasets

To evaluate the performance of our approach, we built two separate datasets. They feature many real-world challenges including fast illumination changes and complex backgrounds, such as those created by moving treetops seen against a changing sky. We now briefly present each of these datasets:

- **UAV dataset** comprises 20 video sequences of 4000 752×480 frames each on average. They were acquired by a camera mounted on a drone filming similar ones while flying indoors and outdoors. The outdoor sequences present a broad variety of lighting and weather conditions. All these videos contain up to two objects of the same category per frame. However, the shape of the drones is rarely perfectly visible and thus their appearance is extremely variable due to changing altitudes, lighting conditions, and even aliasing and color saturation due to their small apparent sizes. Fig. 1.7(a) illustrates some examples of the variety of appearance of a drone present in this dataset.
- **Aircraft dataset** consists of 20 publicly available videos of radio-controlled planes. Some videos were acquired by a hand-held camera from the ground and the rest was filmed by a camera on board of an aircraft. These videos vary in length from hundreds to thousands of frames and in resolution from 640×480 to 1280×720. Fig. 1.7(b) depicts the variety

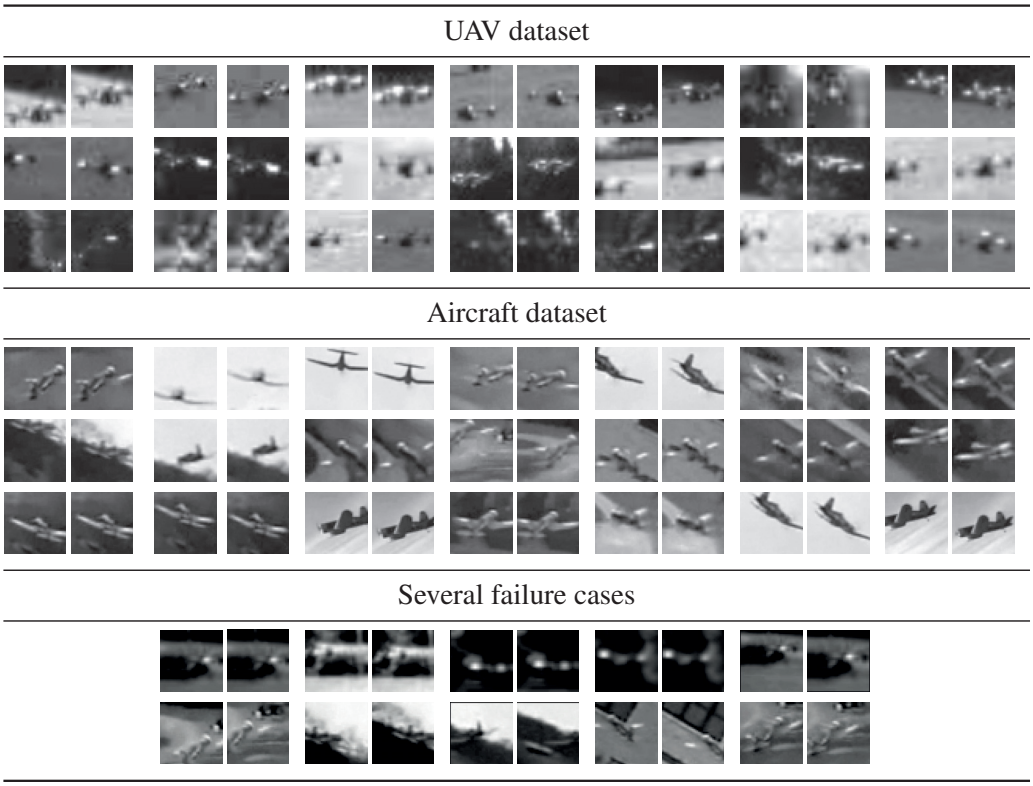


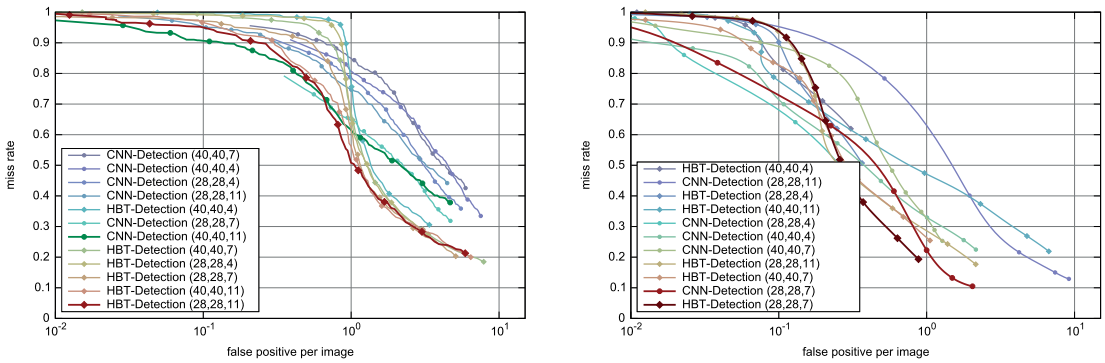
Figure 1.9: Examples of motion compensation. The first image in each pair shows the middle patch of the original st-cube, coming from the sliding window. The second image corresponds to the same patch after applying our motion compensation algorithm. Failure cases are often due to motion estimation failures, which happen when the appearance of the object is heavily corrupted by noise.

of plane types. The aircrafts may also appear under different angles, which makes the problem more complex. Fig. 1.8 shows some examples of the pose variation that a plane could have throughout the video sequence.

1.5.2 Training and Testing

In all cases, we used half of the data to train regressors and detectors and the rest for testing. We manually supplied 8000 bounding boxes centered on a UAV and 4000 on a plane.

We used the Boosted trees implementation of [78] for both regression and detection. To compute the HoG3D and HoG descriptors, we used the publicly available implementations [60] and [79], respectively. We used Theano [62] to build the CNN models for both regression and detection tasks. In both of these cases we used the method described in [75] for optimization. The structures of the CNNs for detection and motion compensation are depicted by Figs. 1.4 and 1.5 respectively. Here the parameters of each layer—the numbers of filters per layer and their dimensions—are



(a) UAV dataset (b) Aircraft dataset

Figure 1.10: Influence of the st-cubes sizes on the performance of Boosted trees (HBT-Detection) and CNN (CNN-Detection) detectors with CNN-based motion compensation method, as described in Section 1.5.2.3. The plots are colored according $MR|_{FPPI=1}$ criterion (introduced in Section 1.5.2.2). Here blue corresponds to the higher $MR|_{FPPI=1}$, while red to the lower one. The darker lines on both plots correspond to the best performing examples of two different types of machine learning algorithms, according to the same criterion. The evaluation was performed on the validation subsets of the UAV and Aircraft datasets. (best seen in color)

given in the figures in the format $N \times (k_x, k_y, k_t)$, where N and (k_x, k_y, k_t) are the number of filters and their sizes respectively.

1.5.2.1 Training the Motion Regressors

To provide labeled examples where the aircraft or UAV is not in the center of the patch but still at least partially within it, we randomly shifted the ground truth bounding boxes by a translation of magnitude up to half of their sizes. This step was repeated for all the frames of the training database to cover the variety of shapes and backgrounds in front of which the aircraft might appear.

Applying large translations to the training data allows us to run the detection to only non-overlapping patches without missing the target, as explained at the end of Section 1.4.3. This procedure allows us to generate as much training data as needed for both Boosted trees (HBT-Regression) and CNN regressors (CNN-Regression), which is important for performance especially as the latter is known to require large amounts of training data.

The apparent size of the objects in the UAV and Aircraft datasets varies from 10 to 100 pixels. To train the regressor, we used 40×40 patches containing the UAV or aircraft shifted from the center.

The CNN-based regressor relies on convolutions of the original patch with filters from different network layers, which may produce artifacts close to the patch borders and degrade performance when the object is only partially visible. To reduce the influence of such artifacts, we extend the

input patch by 25% in both the horizontal and vertical directions. This needs to be done only for the coarse alignment CNN, as depicted by the top row of Fig. 1.5. It is not required for the refinement CNN that only estimates small motions.

Fig. 1.9 depicts some examples of motion compensation. Note that even though both aircrafts and drones appear in front of changing backgrounds, the motion compensation algorithm correctly estimates the object location within the patch. Fig. 1.9 also illustrates some cases when the motion compensation system is unable to correctly predict the location of the object in the patch. This typically occurs when the patches are very noisy and the object is almost not visible.

To handle the wide range of flying objects apparent sizes, we use a multi-scale sliding window detector. Fig. 1.8 shows the same UAV and plane appearing at various distances from the camera throughout the video sequence.

1.5.2.2 Evaluation Metrics

In our experiments we consider an object to be correctly detected if there is 50% overlap between the detected bounding box and the ground-truth bounding box.

We report precision-recall curves. Precision is computed as the number of true positives detected by the algorithm divided by the total number of detections. Recall is the number of true positives divided by the number of the positive test examples. Additionally we use the *Average Precision* measure, which we take to be the integral $\int_0^1 p(r)dr$, where p is the precision, and r the recall.

We also report the log-average miss-rate (MR) with respect to the average number of false positive per image (FPPI). The miss-rate is computed as the number of true positives missed by the detector, divided by the total number of true positives; FPPI is computed as the total number of false positives, divided by the total number of images in the testing dataset:

$$\begin{aligned} MR &= 1 - N_d/N_{tp}, \\ FPPI &= N_{fd}/N_f, \end{aligned} \tag{1.6}$$

where N_d, N_{fd}, N_{tp}, N_f are the number of true and false detections, the number of positively labeled examples and the number of frames in the test set, respectively.

1.5.2.3 Motion Compensation Performance Analysis

Prior to evaluating the detection accuracy of the methods we need to apply motion compensation to the st-cubes. Thus we need to evaluate, which motion compensation method performs best. To this end, we created a validation dataset by selecting one video from each dataset. These videos are then used to generate data, using the method introduced in Section 1.5.2.1. We use the validation set to tune the parameters and then perform the comparison against competing approaches on the test set.

We compare HBT-Regression and CNN-Regression in terms of Root Mean Square Error (RMSE). More formally, we are given a validation set of pairs $\left(X_i, S_i^a\right), i \in 1..N$, where X_i is a patch and $S_i^a \in \mathbf{R}^2$ corresponds to the true shift of the object from the center of the patch. Let also $S_i^p \in \mathbf{R}^2 : S_i^p = \phi(X_i)$ be the prediction of the shift of the object, obtained by the motion compensation system. Then the RMSE is computed using the following equation:

$$\texttt{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(S_i^p - S_i^a\right)^2}. \tag{1.7}$$

Note that S_i^p and S_i^a do not depend on the size of the patch.

Table 1.1 depicts the results of this comparison. CNN-Regression outperforms HBT-Regression on both datasets. For reference we also provide $\texttt{RMSE}_0$, which is computed as:

$$\texttt{RMSE}_0 = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(S_i^a\right)^2}. \tag{1.8}$$

$\texttt{RMSE}_0$ reflects the case when no motion compensation is applied.

	RMSE	
method	UAV dataset	Aircraft dataset
No motion compensation ($\texttt{RMSE}_0$)	0.1474	0.1451
HBT-Regression	0.0939	0.0805
CNN-Regression	**0.0669**	**0.0749**

Table 1.1: Performance of motion compensation methods.The valuation was performed on the validation subsets of the UAV and Aircraft datasets.

We therefore used the CNN-Regression algorithm to produce a number of aligned st-cubes of sizes ranging from $(s_x, s_y, s_t) = (28, 28, 4)$ to $(s_x, s_y, s_t) = (40, 40, 11)$, some of which we used for training and others for testing. For patches smaller than 40×40, we simply upscale them to 40×40 before applying the motion compensation regressors. The choice of s_t controls the trade-off between detecting far away objects using large values and closer ones using smaller ones. This is because, when the object is very close, the apparent motion may become too large for our motion compensation scheme. We found that increasing s_t beyond 11 did not bring any improvement in performance, while decreasing it below 4 left us with too little motion information.

As described above we have used the same video sequences to select the most appropriate size s_t for the st-cube. Fig. 1.10 summarizes our experiments, in terms of log-average miss-rate curves. The legend of the plot describes the set-up used during the experiments. The numbers in brackets correspond to the (s_x, s_y, s_t) dimensions of the st-cube. The order of the curves in the legend is

designed in the way that the highest curve is highest in terms of $MR|_{FPPI=1}$ measure. The lowest curve corresponds to the best performing set-up. For the different detection algorithms we show the best performing results by making the curves darker.

The classifiers of Section 1.3.1 rely on boosted trees operating on HoG3D descriptors [60]. We computed them using the default parameters, that is, 24 orientations per bin of size $4 \times 4 \times 2$ pixels. The Boosted trees detector uses 1500 trees of depth 2. We will further refer to this method as HBT-Detection.

For the CNNs of Section 1.3.2, we tried different network configurations, with variations of the number and size of filters in the convolutional layers and varying numbers of fully connected layers. In the end, they all ended up yielding very similar results. The final configuration that we used is illustrated by Fig. 1.4. We will refer to this method as CNN-Detection.

As depicted by Fig. 1.10, HBT and CNN detectors perform similarly on the plane dataset but the former clearly outperforms the latter on the UAV dataset when we allow a single false positive per frame on average. This may seem surprising but similar behaviors have been reported by [80] where the top four methods rely on decision forests while the Deep learning approach ranks only sixth. In our case, this may be attributable to the size of the training database not being large enough to take full advantage of the power of CNNs. Furthermore, for tasks that require as few false positives as possible, the CNNs win.

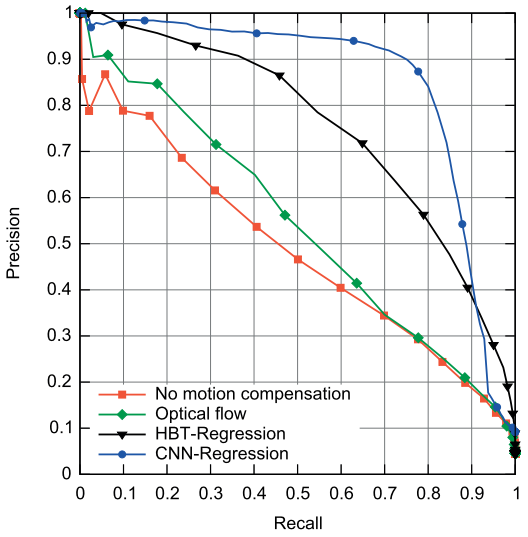
In any event, these experiments suggest that the optimal dimension of the st-cube depends on the task at hand. The apparent size of the UAVs is small, which favors large temporal dimension. As can be seen in Fig. 1.10(a), the best results are obtained for $s_t = 11$. By contrast, the Aircraft dataset comprises examples of planes flying at many different distances from the camera. In this case, $s_t = 7$ is optimal for both HoG3D descriptors and CNNs.

1.5.2.4 Detection-Based Evaluation

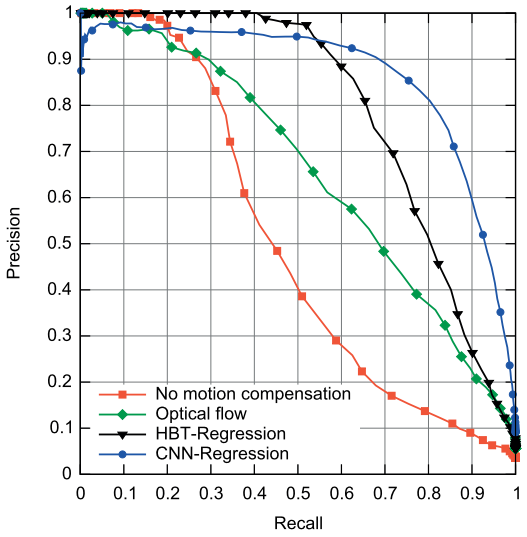
Another way to evaluate our motion compensation algorithm is to compare the detectors, trained on the data, processed with either HBT-Regression or CNN-Regression methods. This measures the influence our motion compensation algorithm has on the accuracy of the detector, which is what we are interested in. We have chosen HBT-Detection method for detection task, as it is faster to train and it showed better accuracy on validation set, based on experiments, depicted by Fig. 1.10. We compared our two methods described in Section 1.4 with an optical flow based method [35], which is probably the best available.

Fig. 1.11 illustrates the results of this comparison. We also provide the performance of the same detector, trained and tested on the data without motion stabilization for reference.

Our methods are able to correctly compensate for the UAV motion even in the cases where the background is complex and the drone might not be visible due to image saturation and noise.



(a) UAV dataset



(b) Aircraft dataset

	Average Precision	
	UAV dataset	Aircraft dataset
HBT-Detection with:		
No motion compensation	0.485	0.497
Optical flow	0.540	0.652
HBT-Regression	0.751	0.789
CNN-Regression	**0.849**	**0.864**

Figure 1.11: Comparison of motion compensation methods on the test subsets of our datasets. For all the motion compensation algorithms we have used the same HBT-Detection approach, as it proved to be more accurate, comparing to CNN-Detection. Unlike the optical flow-based algorithm, our regression-based ones properly identify the shift in object position and correct for it, even when the background is complex and the object outlines are barely visible. This yields a better precision/recall. Table in the bottom of the figure depicts the Average Precision score for the methods presented above.

Fig. 1.1(b,d) illustrates this hard situation with an example. On the contrary, the optical flow method is more focused on the background, which decreases its performance. Fig. 1.1(c) shows an example of a relatively easy situation, where the aircraft is clearly visible, but the optical flow algorithm fails to correctly compensate for its movement, while our regression-based approach succeeds.

Fig. 1.1(a) illustrates another situation, where the object is not in the center of the patch for the middle image of the st-cube. Optical flow methods will align other patches of the st-cube with respect to the middle one, which will result in object being shifted from the center in all the st-cube patches. By contrast, our motion compensation algorithm does not require any reference frame, leading to higher accuracy.

Using motion compensation for alignment of the st-cubes results into a higher performance of the detectors, as in-class variation of the data is decreased. Fig. 1.11 shows that we can achieve at least 15% improvement in average precision on both datasets using our motion compensation algorithm.

Our CNN-based motion compensation algorithm performs best. It yields about a 10% increase in accuracy, compared to the boosted trees method. Such difference in performance most likely lies in the nature of the features used by these machine learning techniques. The boosted trees regressor is using HoG features, which might not be perfectly suited for the problem, while the filters in the CNN are learned directly from the data. As the CNN obtains better accuracy, for our further experiments we will use the CNN-based motion compensation.

1.6 Comparing against Competing Methods

In this section, we compare the performance of the pipeline of Section 1.3, optimized as described in Section 1.5, against several state-of-the-art algorithms on the two challenging datasets introduced in Section 1.5.1. For these experiments, we therefore use st-cubes whose sizes are $(28, 28, 11)$ for UAVs and $(28, 28, 7)$ for planes, which are those we determined to yield the lowest miss-rates when we use HoG3D descriptors for detection and CNNs for motion compensation.

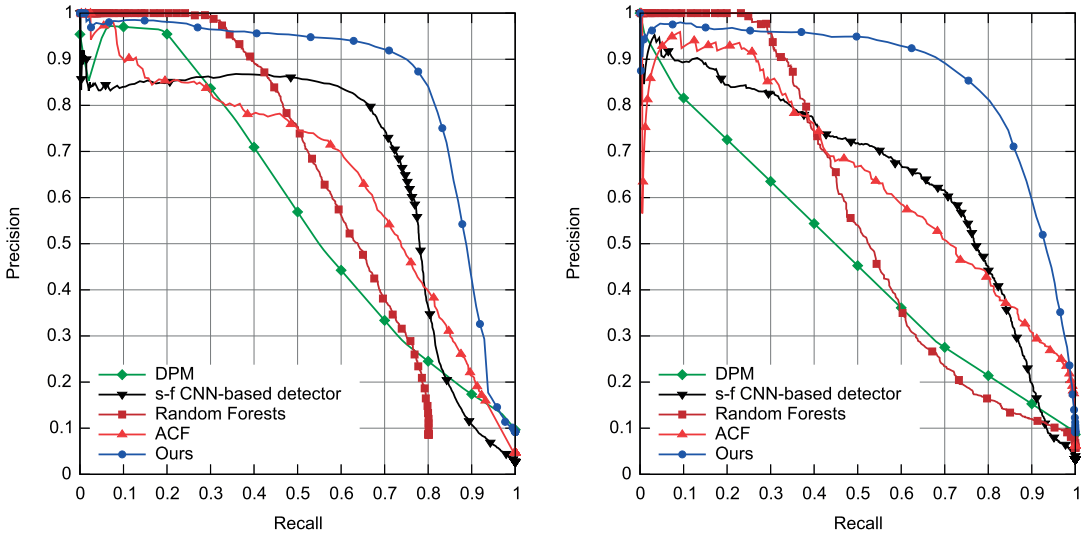
We first list the algorithms we use as baselines and show that ours outperforms them consistently both for plane and UAV detection. We then demonstrate that motion compensation does not significantly degrade performance in cases when it is not strictly needed, such as when two aircrafts are on a collision course.

1.6.1 Baselines

To demonstrate the effectiveness of our approach, we compare it against state-of-the-art algorithms. We chose them to be representative of the three different ways the problem of detecting small moving objects can be approached, as discussed in Section 1.2.

- **Appearance-Based Approaches** rely on detection in individual frames. We will compare against Deformable Part Models (DPM) [29], single-frame based Convolutional Neural Networks (s-f CNN-based detector) [62], Random Forests [81], and the Aggregate Channel Features method (ACF) [18], the latter being widely considered to be among the best.

  Since our algorithm considers st-cubes, for a fair comparison with these single-frame algorithms, we proceed as follows. Similarly to our approach we divide the video sequence into a set of N-frame overlapping slices. We further extract st-cubes using a sliding window approach, but motion compensation is not applied. We then run the single frame based detector on each of the patches of these st-cubes and consider the whole st-cube b as



UAV dataset Aircraft dataset

Figure 1.12: Comparing against appearance-based approaches [81, 62, 18, 29] in terms of precision/recall. For both the UAV and Aircraft datasets, the blue curve depicts our approach and is significantly above the others.

positive if the weighted average of scores of the patches in b is positive. We use a simple Gaussian kernel G centered on the middle frame of b as a weighting function. G is defined as

$$G = \exp\left(-\frac{(i - s_t/2)^2}{2\sigma^2}\right), \tag{1.9}$$

where s_t is the filter size and σ is taken as $\sigma = 0.3\,((s_t - 1)/2 - 1) + 0.8$ as often done. We tried simply averaging over the detection scores of the set of patches in the b, but it resulted in lower accuracy, because the detectors tend to give a higher score to the middle frame, in which the object appears to be close to the patch center.

- **Motion-based Approaches** do not use any appearance information and rely purely on the correct estimation of the background motion. Among those we experimented with MultiCue background subtraction [64, 31] and large displacement optical flow [34].
- **Hybrid approaches** are closer in spirit to ours and correct for motion using image-flow. Among those, the one presented in [35] is the most recent we know of and the one we compare against. The main difference is that it relies on optical flow for motion compensation whereas we use CNNs. To ensure a fair comparison, we used the same patches to construct the st-cubes both for our method and to extract the features [35] requires.

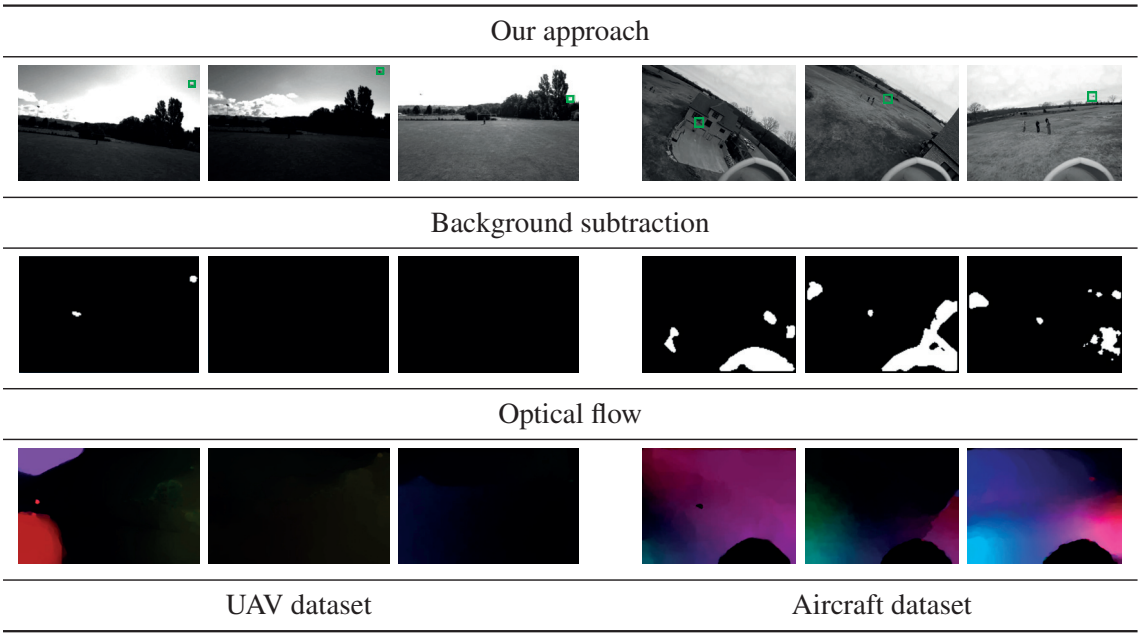


Figure 1.13: Comparing against motion-based methods [34, 64]. [TOP] Our detector detects the objects by relying on motion and appearance, as evidenced by the green rectangles. [MIDDLE] Background subtraction results of [64]. Only in the leftmost frame of the three on the left, is there a blob that corresponds to a UAV, along with one that does not. Similarly, there is a small blob that corresponds to a plane in the central frame of the three right-most ones and many large ones in the others that do not clearly correspond to anything. [BOTTOM] Optical flow computed using the algorithm of [34]. The plane and UAV generate a distinctly visible pattern in 2 or the 3 right-most images but in none of the three left-most ones. (best seen in color)

For all the motion-based [34, 64, 31] and single-frame-based [81, 62, 18, 29] approaches, the code was downloaded from publicly available sources. In particular, for ACF and Random Forests, we used the toolboxes of [82] and [78] respectively. The DPM implementation is publicly available [29]. We also used the open source BGSLibrary [31] for state-of-the-art background subtraction algorithms. To compute features, we used default parameter configurations much as we did in our own pipeline for HoG3D. For algorithms relying on Random Forest, we tried varying the number of trees, and kept the number yielding the best results, again much as we did to find the best CNN configurations in our pipeline. For [35], we did not find a publicly available implementation and reimplemented the algorithm ourselves.

1.6.2 Evaluation against Competing Approaches

We used the same video sequences to train all the methods from the three classes described above. We compare here their results against ours.

Method	Average Precision	
	UAV dataset	Aircraft dataset
Single-frame based approaches		
DPM [29]	0.573	0.470
Random Forests [81]	0.618	0.563
s-f CNN-based detector [62]	0.682	0.647
ACF [18]	0.652	0.648
Hybrid approaches		
Park [35]	0.568	0.705
Ours	**0.849**	**0.864**

Table 1.2: Average precision of detection methods on our datasets. We can see that in both cases our approach is able to reach higher detection accuracy. We achieve about 15% increase comparing to the best competing algorithms for the UAV and Aircraft datasets.

1.6.2.1 Appearance-Based Methods.

In Fig. 1.12, we compare our method with appearance-based ones on our two datasets in terms of precision/recall. Table 1.2 summarizes the results in terms of Average Precision. For both the UAV and Aircraft datasets we improve on average by $15-20\%$ over ACF [18], which itself outperforms the others.

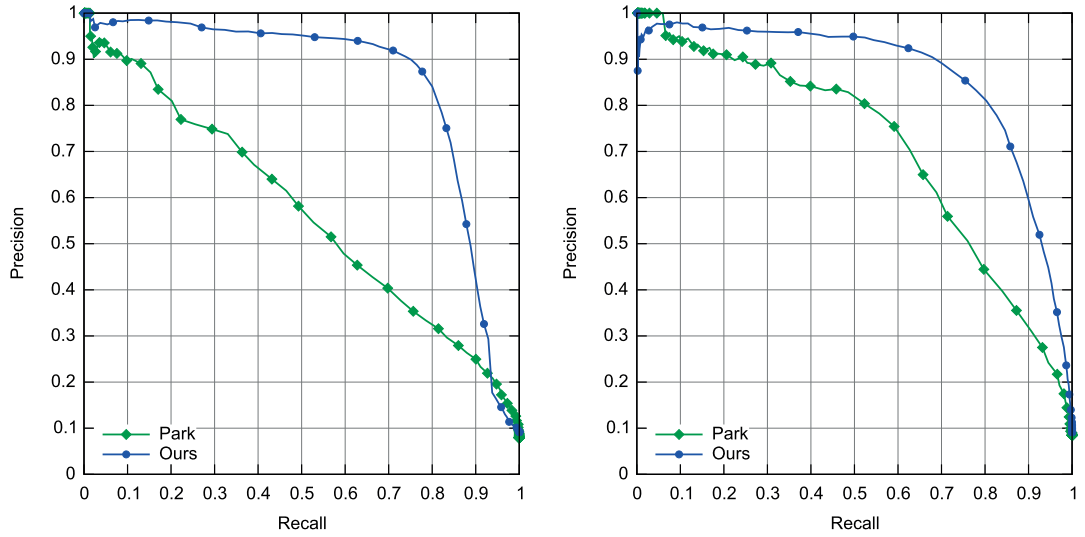
The CNN approach, provided by[62] yields scores comparable to those of the Random Forests and ACF methods. The structure of the network is the one depicted by Fig. 1.4, except for the fact that we replaced 3D convolutions by standard 2D ones. To boost CNN performance, we used Local Contrast Normalization (LCN) [83] after every convolutional layer and minimize the Hinge Loss at the final layer of the network, which was shown to be effective [84, 85].

The DPM [29] performs worst on average. This likely happens because it depends on using the correct size of the bins for HoG estimation, which makes it hard to generalize for a large variety of flying objects.

1.6.2.2 Motion-Based Methods

Fig. 1.13 depicts cases where background subtraction [64] and optical flow computation [34] algorithms, even though they are state-of-the-art, do not work well enough for detecting UAVs or planes in the challenging conditions we consider.

We did not compute precision-recall curves using these motion-based methods because it is unclear how big the moving part of the frame should be considered as an aircraft. We have tested several potential sizes and the resulting average precision values were much lower than those in Table 1.2 in all cases.



(a) UAV dataset (b) Aircraft dataset

Figure 1.14: Comparing against the hybrid method of [35]. Our approach performs better for both UAVs and Planes.

1.6.2.3 Hybrid approaches

In Fig. 1.14, we compare our method against the hybrid approach of [35], which relies on motion compensation using Lucas-Kanade optical flow method, and yields state-of-the-art performance for pedestrian detection. As shown in Fig. 1.1, optical flow motion compensation cannot achieve good performance in our case, mostly because the target object is rather small and its appearance can significantly change due to illumination and background changes.

As a result, our regression-based approach allows achieving higher performance for both the UAV and aircraft datasets. This suggests that accurate localization of the object in the patch is essential and leads to significant improvement in detection accuracy. Fig. 1.15 shows several frames to illustrate the performance of our approach.

1.6.3 Collision Courses

Motion compensation can be seen as a way to make the st-cube invariant from the motion of the aircraft, as it keeps flying object in the center, for all the patches of the st-cube. To evaluate whether enforcing this kind of invariance negatively impacts performance in the situations when it is not required, we applied our approach to the case of aircrafts on collision courses.

As shown in Fig. 1.16, if the aircraft A_1, observed from the camera of another aircraft A_2, is on collision course with A_2 then its behavior can be characterized by two important properties:

- A_1 remains at constant angle with respect to A_2

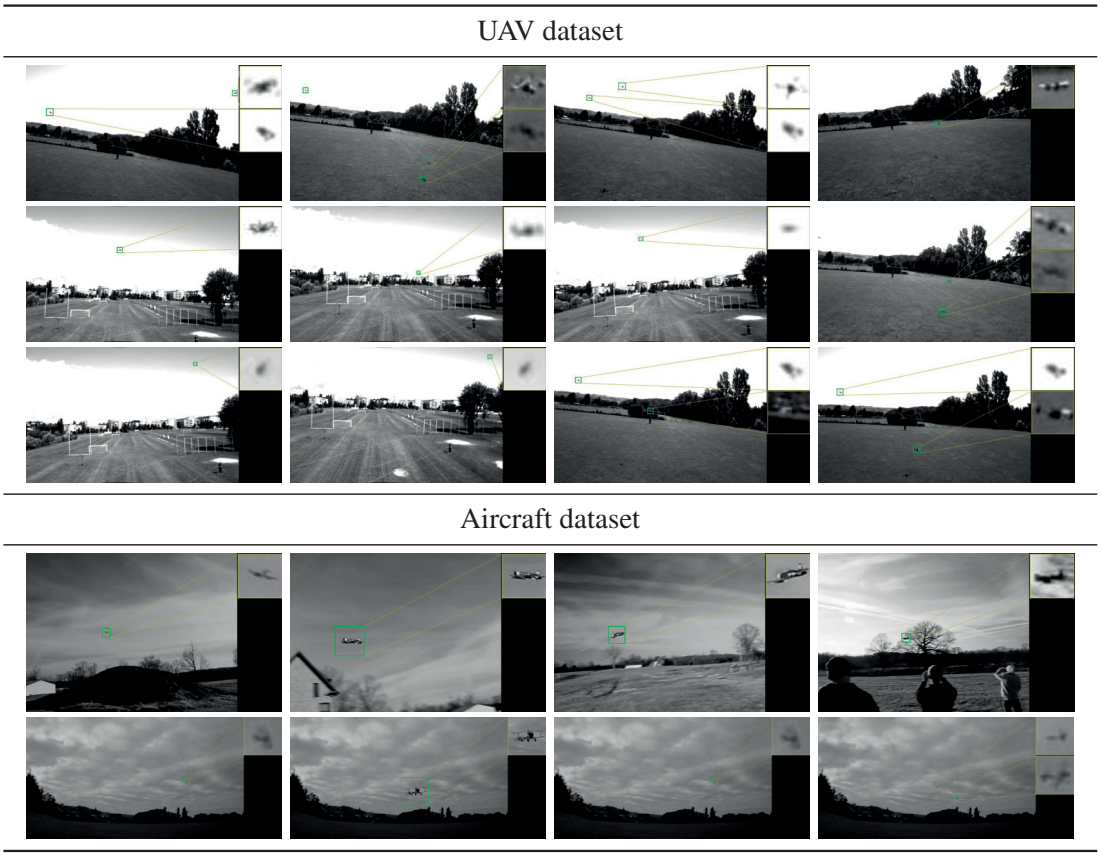


Figure 1.15: Some detection results. Thumbnails at the side of each figure show the zoomed-in versions of the detections made by our algorithm.

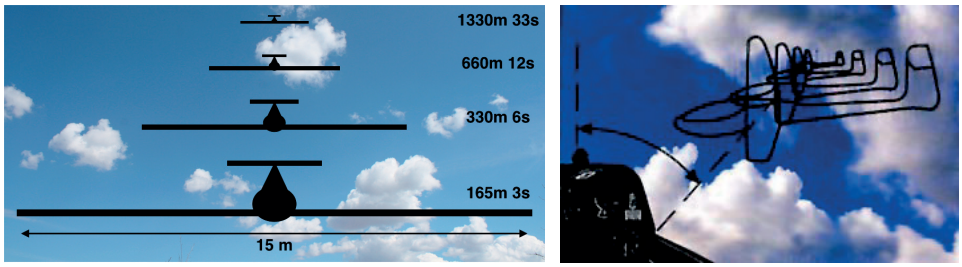
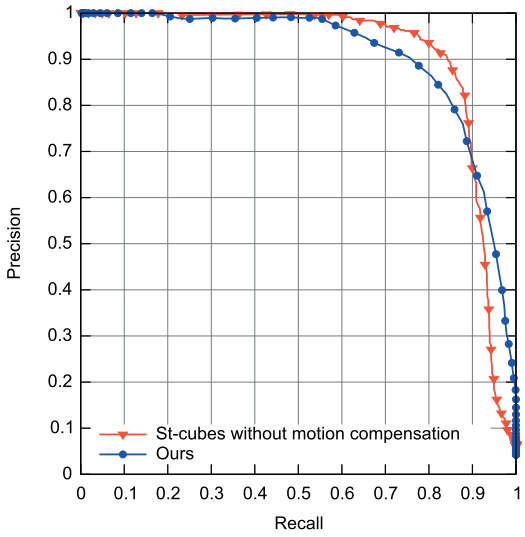


Figure 1.16: Collision courses. [LEFT] The apparent size of a standard glider and its 15 m wingspan flying towards another aircraft at a relatively slow speed (100 km/h) is very small 33s before impact, but the glider completely fills the field of view only half a minute later, 3s before impact. [RIGHT] An aircraft on a collision course is seen in a constant direction but its apparent size grows, slowly at first and then faster.

- the apparent size of A_1 increases from the point of view of A_2

These properties are invariant from the actual positions of the aircrafts in the 3D environment, the only constraint is that the paths of the aircrafts should intersect, which effectively means



st-cube	AP (Average Precision)
Without compensation	0.907
With compensation	0.904

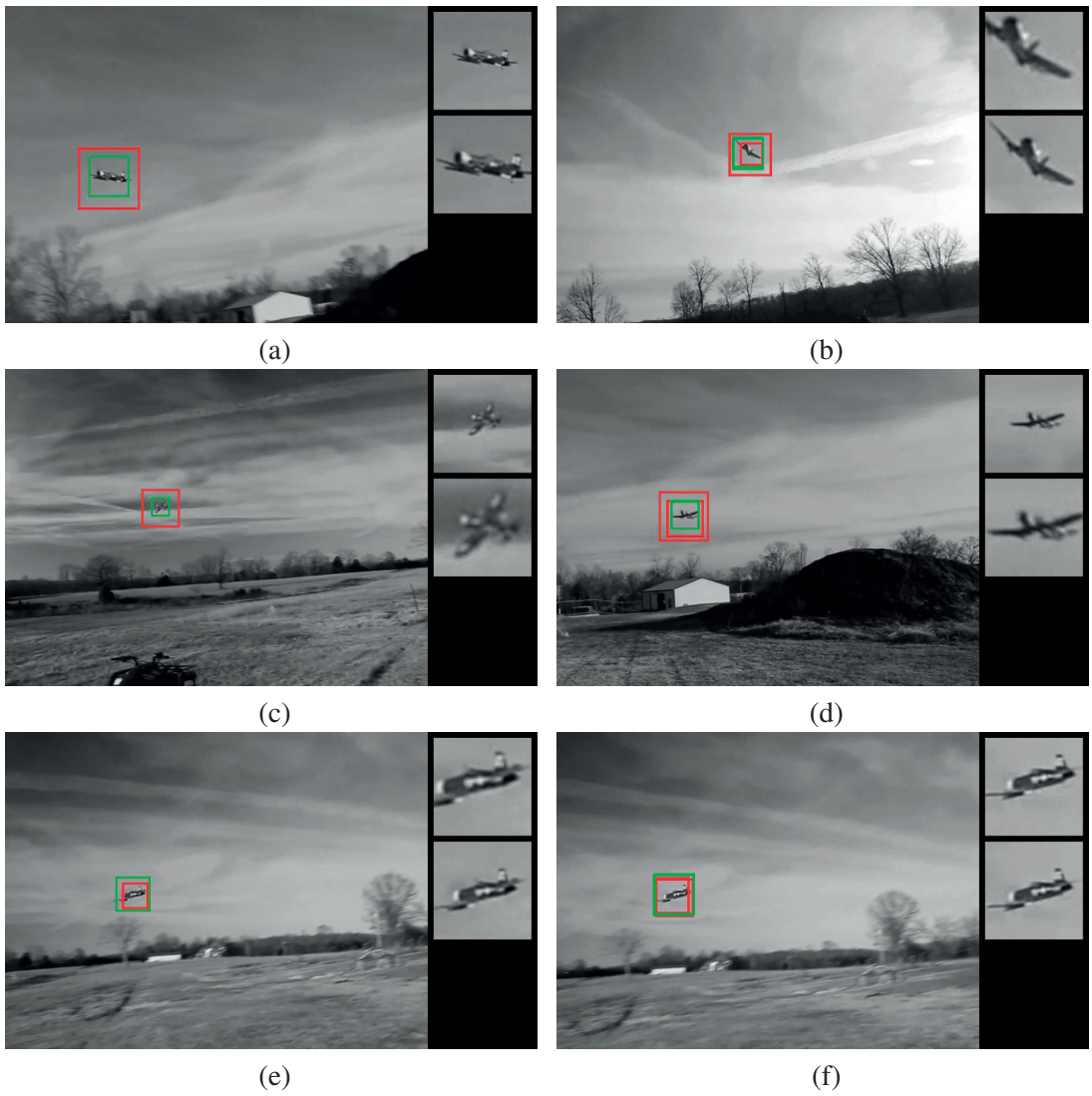
Figure 1.17: Performance for aircrafts on a collision course. [LEFT] Precision/recall with and without motion compensation. [RIGHT] Average Precision with and without motion compensation.

collision. In our case only the first property is important, which means that motion stabilization is not needed, as A_1 will always occupy the same position in the image from the camera of A_2, provided A_1 and A_2 are on collision course.

We therefore searched publicly available sources for video sequences in which airplanes appear to be on a collision course for a substantial amount of time. We found fourteen, which vary in length from tens to several hundreds of frames. As before, we used half of them to train the detector and the others to test it.

In Fig. 1.17, we compare our results with and without motion stabilization. As expected, even though the non-stabilized results were poor in the general case, they are much better in this specific scenario. Incorporating motion stabilization very slightly degrades performance, which could be expected because enforcing *any* kind of invariance always loses some amount of information and is penalizing when such invariance is not required. However, in this case, the loss is almost negligible.

This is significant because, in a practical on-board system, detecting aircrafts on a collision course, which present a clear and immediate danger, would probably take priority over detecting all others. The former does not require motion compensation while the latter does. However, since nothing is lost by having motion compensation on, we can detect all aircrafts, whether on a collision course or not, without performance loss in the crucial case of those that are.



(a) (b) (c) (d) (e) (f)

Figure 1.18: Scale adjustment. The red bounding box shows the original detection and the green one the position adjusted for scale and motion. The thumbnails on the right are zoomed-in versions of the detections, with the top one illustrating the original detection and the bottom one showing the one after being motion and scale are adjusted. (best seen in color)

1.6.4 Scale Adjustment

As discussed in Section 1.4.3, we must run our detection scheme at different image resolutions to accommodate rapid size changes. This additional computational burden can be reduced by compensating not only for motion but also for size, which makes it possible to reduce the number of scales the system needs to check.

More specifically, we trained a regressor $\phi_{sc}(\cdot)$ to adjust for scale so that the bounding box fits the object of interest, much in the same way as we learned a regressor to compensate for motion.

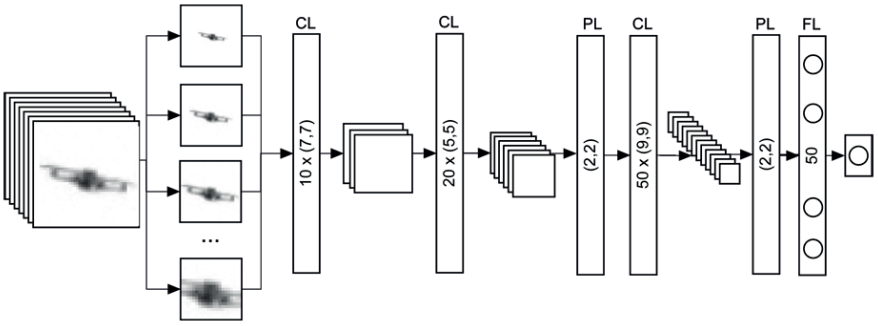


Figure 1.19: Structure of the scale adjustment Convolutional Neural Network. Several input channels contain object at different scales. The output of the CNN is a number, which characterizes the true scale of the object.

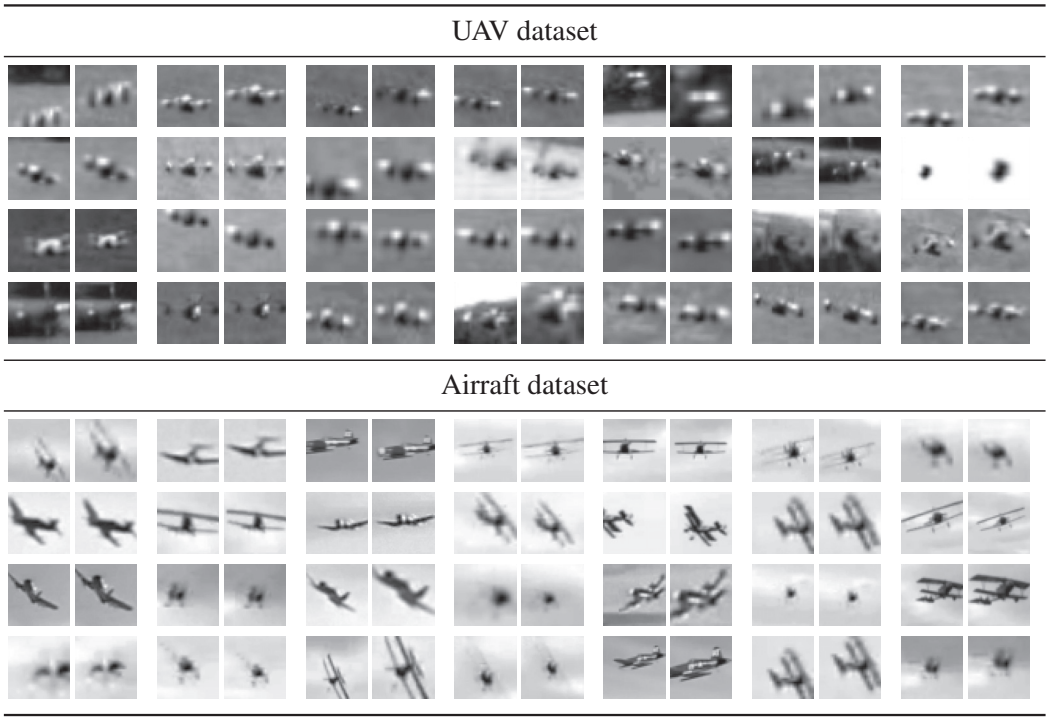


Figure 1.20: Sample results for simultaneous scale and motion compensation. The left image of each pair contains the original patch, where neither scale nor position are corrected. The right patch depicts the resulting patch after scale and motion correction.

Fig. 1.18 illustrates this process in two separate cases. Note that in the case of Fig. 1.18(b), there were originally two different detections, which were collapsed into the same one after adjustment without having to perform non-maximum suppression.

Since CNNs have proved more effective for motion compensation than HoG-based regressors, we used them to implement scale adjustment as well. We found out experimentally that using just a single patch to predict the true scale of the object is not enough. As in [86], we therefore

motion compensation + detection	0.123s
motion and scale adjustment + detection	0.193s

Table 1.3: Speed comparison of the motion and scale adjustment methods with motion compensation. We provide the time needed to process a single st-cube using an Intel® Xeon® CPU E5-2650 v2 running at 2.60GHz.

Method:		UAV dataset	
		number of scales processed per frame	average miss-rate for FPPI = 1
HBT-Detection			
	without scale adjustment	4	51%
	without scale adjustment	8	50%
	with scale adjustment	8	54%
	with scale adjustment	16	52%
	with scale adjustment	32	48%

Table 1.4: Evaluation of the HBT-Detection method on the UAV dataset with and without scale adjustment. Both method perform better when more scales are used, at the cost of increasing the computation time.

used several scales as inputs to the CNN. Fig. 1.19 illustrates its structure.

The input to this CNN is a set of images of the object at different scales, which are provided as separate channels. Its output is the estimated scale of the object. Since there is no pooling layer after the first convolutional layer, we can estimate the scale with high precision. Furthermore, this CNN can be combined with the motion stabilization one of Section 1.4 to increase the accuracy of both motion compensation and scale adjustment. The structure of the resulting composite CNN is similar to the one depicted by Fig. 1.19. However, the output of its fully-connected layer has 3 floating point values instead of only 2. The first two are the shifts from the center of the patch in the spatial domain and the last one is the estimated scale. This replaces NMS in scale space, as described in Section 1.3, and yields precise object localization. Fig. 1.20 depicts some scale-adjustment results.

Table 1.3 compares the time required to process a single st-cube using our approach with and without scale adjustment. In this case, we have used st-cubes of size $(40, 40, 4)$ and 7 scales for the scale adjustment algorithm. Note that the number of scales can be selected with respect to the desired localization quality. Thus having many scales will yield more precise estimation of the object size, at the cost of a computation time increase. In our experiments we selected 7 scales, which results in high localization precision, as depicted by Fig. 1.21, while keeping the processing time relatively low. Even though adding scale adjustment to motion compensation increases the processing time per st-cube, it reduces the overall computation time by a factor of about 4. This is because it replaces the need of doing NMS across 7 different scales, which takes

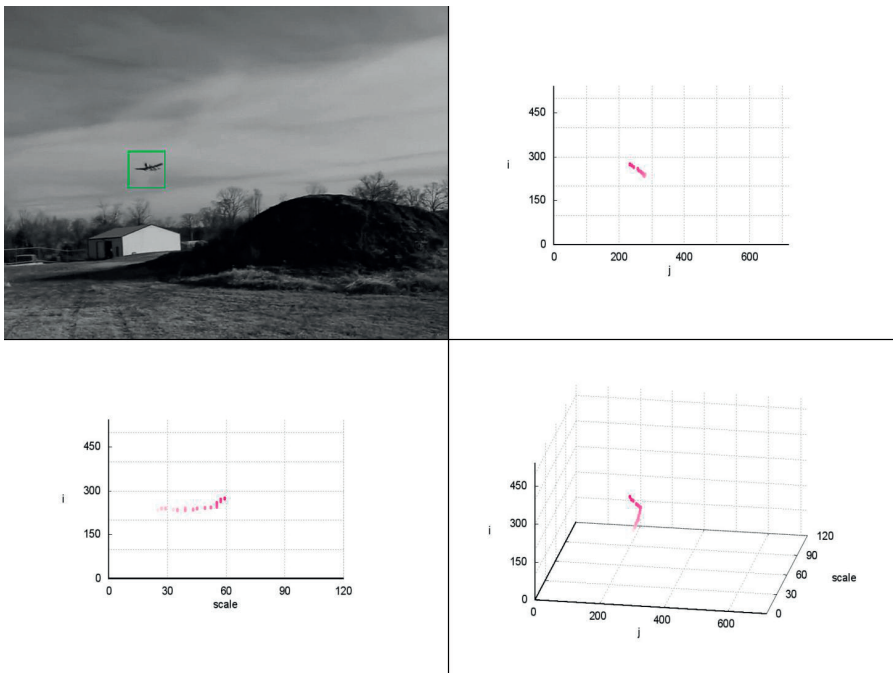


Figure 1.21: Precise estimation of the scale of the object allows us to localize it in 3-D space. [TOP LEFT] Scale and motion adjusted detection of the aircraft in one frame of a video sequence. [TOP RIGHT] Projection of the points of the 3D trajectory throughout the previous 20 frames to the image plane. [BOTTOM LEFT] Changes of object scale. [BOTTOM RIGHT] Trajectory of the object in 3D space is quite smooth due to the motion compensation algorithm, while neither tracking nor additional smoothing is applied.

$0.123 * 7 = 0.861$ seconds, by processing one st-cube while accounting for scale, which takes 0.193 seconds.

In Table 1.4, we evaluate our approach on the UAV dataset with and without scale adjustment. Even though HBT-Detection with scale adjustment allows for faster computation, its performance is slightly lower than without scale adjustment. This is mainly due to the artifacts that appear when resizing small noisy images. Greater scale numbers improve detection accuracy at the cost of increased computation time.

In the experiments of Section 1.6.2, we rely on 50% overlap between detected and ground-truth bounding boxes. Thus, it is unnecessary to localize the target objects very precisely. We therefore use our method without scale adjustment on 8 distinctive scales, which yields a good balance between accuracy and computational time.

Fig. 1.21 illustrates the performance of our detection method in combination with motion compensation and scale adjustment. Our algorithm localizes the flying object with a great accuracy and yields trajectories that are smooth both in the spatial domain and in scale space. Provided that the camera is calibrated and given the true size of the object, we can estimate its

distance to the camera, which is critical for collision avoidance purposes.

Different other examples that illustrate the performance of our motion compensation and detection approaches can be found at the following link: http://cvlab.epfl.ch/research/unmanned/detection.

1.7 Conclusion

In this chapter we showed that temporal information from a sequence of frames plays vital role in detection of small fast moving objects like UAVs or aircrafts in complex outdoor environments. We therefore developed a novel object-centric learning-based motion compensation approach that is robust to changes in the appearance of both object and background. Both CNN and Boosted trees methods allow us to outperform state-of-the-art techniques on two challenging datasets. The CNN-based method proved to be more suitable for motion compensation than the Boosted trees-based one.

2 Synthetic Data Generation

In the previous chapter we have presented a learning-based approach that combines motion and appearance information to efficiently detect drones in the challenging video sequences. The main limitation of this method lies in the complexity of data collection, as the training dataset should contain images/videos of various types of aircrafts in different environments and lighting conditions. Though we can not completely avoid the data collection process we can reduce the manual effort by augmenting the small training set of images. The standard way of doing this is by applying small deformations and adding noise to the training samples [87, 88], as done for character [47, 48], face [89], and image patch recognition [90]. This approach, however, assumes that the original training set is already diverse enough, as the range of augmented images that can be produced is limited.

A different direction is taken by [91], who suggests augmenting the small dataset of real images with synthetic ones for the task of 3D body pose estimation from a single depth camera. The major difference with our problem is that depth images do not depend on lighting, motion blur, and other artifacts that affect images from a regular camera, and are therefore comparatively simpler to synthesize. In the context of RGB images, synthetic data has been investigated for human detection and pose estimation purposes in [39, 40], but [39] does not model image-acquisition artifacts, while [40] involves considerable amounts of manual interaction, which is less desirable. It was recently shown [41] that it is possible to use a 3D car model to first extract appearance information from real images of cars, and use it to synthesize novel views. Using these generated images along with the real ones during training improves performance, however this approach does not account for other artifacts such as motion blur and is only applicable to objects with relatively simple geometry.

Furthermore, to the best of our knowledge none of these approaches offers a principled way to choose the image synthesis parameters to match the behavior of real-world cameras in the presence of noise. The relevant parameters are typically tuned by hand, which quickly becomes unmanageable when the rendering pipeline is complex. To overcome this limitation, we therefore introduce a fully automated and generic method to estimate these parameters from a small set

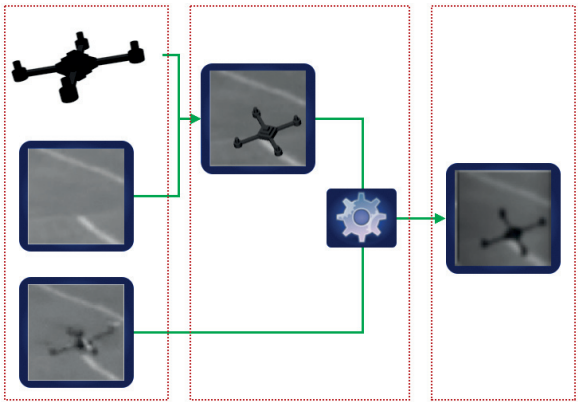


Figure 2.1: Synthetic data generation pipeline. [LEFT] Input to the system includes a simple model of the object of interest, an image of this object and a background image (it should be the same background as the image of the object has). [MIDDLE] Overlaying the model on the background yields a synthetic image. This image is then processed to maximize its similarity to a real image from the perspective of the detector. [RIGHT] The resulting synthetic image can be used for training the detector.

of available real images to maximize the performance of a detector trained using the resulting synthetic images.

To this end, we start from a small set of real seed images containing a target object and corresponding background images without it, such as the ones depicted by Fig. 2.1. Given a very coarse 3D model of the object of interest, such as that of the drone of Fig. 2.1, we estimate the 3D pose of the object, overlaid onto the background image, and then post-process the resulting composite image so that it is as similar as possible to the real one. This is achieved by automated selection of the post-processing parameters to maximize a similarity between the two images. Once these parameters are found, we can then change the position and the orientation of the object in the images to generate arbitrary large synthetic datasets with realistic imaging artifacts.

A key ingredient of our approach is the similarity function used to measure the difference between real and synthetic images. An obvious candidate would be the pixel-wise Euclidean distance. However, our goal is not to generate visually pleasant images, but rather training data that is effective for our intended purpose. We will therefore show that the best similarity depends on the target detection method. We demonstrate this for three widely used methods that are representative of the state-of-the-art: The Deformable Part Model (DPM) method [92], an AdaBoost-based detector [93], and a detector based on Convolutional Neural Networks (CNN) [94].

In short, our contribution is a novel and fully automated approach to generating synthetic training image databases that increases detection performance and outperforms the state-of-the-art techniques discussed above, irrespective of the specific detector used. We will demonstrate this in the context of drone, plane, and car detection.

In the remainder of this chapter, we first discuss the related work. In Section 2.2 we then present

the image-specific effects we want to model in our synthetic images, followed by a discussion on different similarity functions that we use to quantify the difference between synthetic and real images in Section 2.4. Finally, in Section 2.5 we demonstrate the performance of our approach on three different datasets and compare it to the state-of-the-art.

2.1 Related Work

Given the prevalence of Machine Learning based algorithms, capturing and annotating training images has become a major issue, and sometimes a severe bottleneck when such images are hard to acquire. In such cases, using Computer Graphics techniques to generate them is a very attractive alternative.

For example, Optical Character Recognition systems have long been trained using samples created by applying various deformations and adding image noise to actual samples [47, 48]. Similarly, synthetically generated image patches have been successfully used in [95, 96]. Note, however, that neither characters nor patches exhibit the full complexity of natural images and are therefore easier to synthesize. In [91], this approach was used on complete depth images generated from 3D models of people to train classifiers to recover human 3D pose from the output of a Kinect camera. This has been remarkably successful, in large part because it provides a way to create arbitrarily large training dataset. However, depth images also lack many of the imaging artefacts present in ordinary images, such as motion blur or lighting effects, which make it difficult to use such an approach for video imagery.

This was attempted in [39] by generating images of pedestrians in various poses and environments to train a pedestrian detector. The results are encouraging but the method does not take complex imaging artefacts into account. More recently, an approach to creating more realistic synthetic images by extracting people's silhouettes from real images, and superimposing them over various backgrounds was proposed [40]. However, it is very specific to pedestrian detection and requires a considerable amount of manual annotation.

Like ours, the approach of [41] relies on both real training images and a 3D model. After registering the 3D model to the images, the material and lighting properties of the different object components are estimated and used to synthesise new views of the 3D model. However, it does not take into account other artifacts such as motion blur and requires precise registration.

Furthermore, to the best of our knowledge none of the aforementioned approaches offers a principled way to choose the image synthesis parameters to match the behavior of real-world cameras in the presence of noise. The relevant parameters are typically tuned by hand, which quickly becomes unmanageable when the rendering pipeline is complex.

Of course, generic image synthesis techniques have also been used in computer vision for many other purposes, such as optimizing camera tracking algorithms [97], evaluation of algorithms [98, 99, 100, 101, 102, 103], gesture recognition and pose estimation [104, 105], or rendering virtual

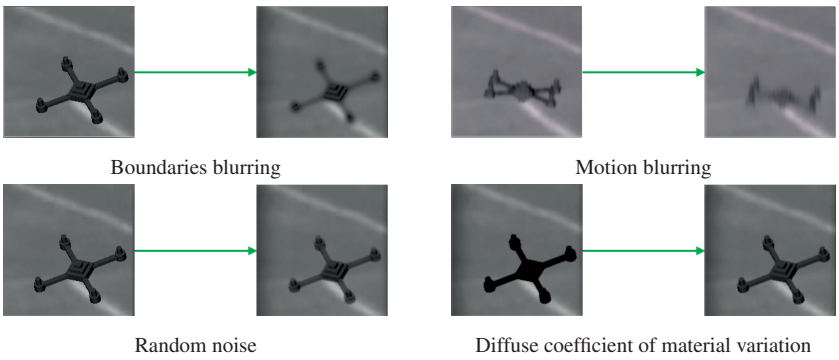


Figure 2.2: The four post-processing effects we use for increasing the similarity between the real and synthetic images.

objects that merge well with real images [106]. Some of these approaches simply project the 3D model of the object of interest on an arbitrary background image. Others add post-processing on similarly generated synthetic images in order to make them look realistic. However, to the best of our knowledge none of them estimate neither how realistic the resulting images are, nor how suitable they are for the application itself.

In this work, we will use some of the same approaches to synthesizing realistic images. This being said, visual realism is not our end goal, but rather the classification performance improvement. As such our algorithm, unlike the others, automatically optimizes the rendering parameters solely for this purpose.

2.2 Generating Synthetic Images

As illustrated by Fig. 2.1, while our pipeline is simple, it depends on many parameters that would be hard to choose by hand. We use simple CAD models, such as that of Fig. 2.1, which roughly captures the target object geometry. We assume that we are given a small set of real images featuring the target object and a corresponding set of background images without it. As we will explain, these background images can usually be extracted from the training video sequence itself. In cases where the background is not visible at any time, it is still possible to estimate it by cutting out the object from the original images and using a texture filling algorithm. This approach will be more thoroughly discussed in Section 2.5.3.

For each real image, we then compute 5 object *pose* parameters, that include 3 orientations $(\alpha^p, \beta^p, \gamma^p)$ and 2 translations (t_x^p, t_y^p), which lets us project the 3D model at the desired location. Note that as we use multi-scale detector, we do not need to vary the scale of the object.

As shown in Fig. 2.2, we then post-process the synthetic image to maximize its similarity to the real image. This involves:

- *Object boundary blurring* (**BB**). The discrete nature of the image sensor causes a mixture of the intensities of the background and the target object along its boundaries. To simulate this effect we apply Gaussian blurring along the object boundaries after the object image has been overlaid on the background image. This is controlled by the standard deviation σ^s of the Gaussian kernel used for smoothing.

- *Motion blurring* (**MB**). This mimics the blurring effect that affects on fast moving objects if the shutter time of the camera is too long. To simulate this effect we use anisotropic Gaussian blurring applied to the pixels of the object in the direction of its motion. The parameters are the two standard deviations σ_u^m and σ_v^m of the Gaussian kernel and the angle α^m of the motion.

- *Random noise* (**RN**). This emulates the shot noise added to the image by the camera. To simulate this effect we simply add independent Gaussian noise to the pixel intensities. Note this is limited to the image pixels that correspond to the inserted object, as the background images are real ones and already contain similar noise. This is controlled by the standard deviation σ^n of the Gaussian distribution used to generate the noise.

- *Material properties* (**MP**). We also vary the material properties, by changing the weight w^d of the diffuse reflection. This allows us not only to vary the color of the object, but also to introduce some diffuse lighting effects. While we do not take specularities into account, this would be a very natural extension to our approach.

We refer to these synthetic data generation parameters as *capture* parameters

$$\Theta = [\underbrace{\alpha^p, \beta^p, \gamma^p, t_x^p, t_y^p}_{\text{pose}}, \underbrace{\sigma^s, \sigma_u^m, \sigma_v^m, \alpha^m, \sigma^n, w^d}_{\text{capture}}]^\top . \tag{2.1}$$

These parameters are challenging to tune because they are heavily correlated. This is particularly true of object pose and direction of motion blur, as well of boundary blurring and motion blurring. Thus our goal is to estimate the Θ parameters for every seed real image that we use for synthetic data generation. Given the background images and the corresponding Θ parameters, we retain the capture parameters and randomize the pose ones to generate arbitrary large numbers of synthetic images that will be realistic enough to be used for training the object detector. We explain below how we recover these Θ parameters.

2.3 Optimizing the Rendering Parameters

To optimize the pose and capture parameters in Θ, we rely on a small set of real images of the target object, together with the corresponding images of the background without the target.

Starting from a background image on which we render the CAD model of the target object, we optimize the rendering parameters to reproduce the corresponding real image. This optimization is

performed on each image independently, because the same capture parameters do not necessarily apply to all of them. More formally, we consider the set of pairs of real images $\{(X_i, B_i)\}_{i=0}^{N}$, where $X_i \in \chi$ is the i^{th} image of the object and $B_i \in \chi$ is the background image for X_i. Let $d : \chi \times \chi \rightarrow \mathbf{R}^+$ be a similarity function, which we use to compare two images, and which we will define explicitly in Section 2.4. Lastly, let $S(\Theta, B_i) \in \chi$ represent the synthetically rendered image by applying the synthetic data generation process with parameters Θ to the Background B_i

To find the set of parameters Θ that best corresponds to real image X_i, we look for

$$\Theta^{(i)} = \arg\min_{\Theta} d(X_i, S(\Theta, B_i)) \tag{2.2}$$

by Simulated Annealing [107]. This approach is widely used for solving non-continuous optimization problems with a large number of parameters. In practice, we initialize the pose parameters by manually providing the object center, which could be avoided with a more sophisticated optimization algorithm. Capture parameters are initialized randomly. This optimization takes a few seconds on each of our 40×40 images.

The capture parameters in Θ depend on viewing conditions, such as lighting and weather conditions, which is why we perform the optimization in each image independently. Fig. 2.3 describes their distributions across images. Note that these distributions are absolutely not Gaussian and that it would therefore be non-trivial to describe them analytically.

2.4 Image Similarity Measures

The resulting parameters depend critically on the similarity function $d(\cdot, \cdot)$ used to evaluate how close the two images are to each other. The simplest is the Euclidean distance between the intensity values of corresponding pixels

$$d_{\text{Eucl}}(X_{\text{re}}, X_{\text{sy}}) = \sqrt{\sum_{v=1}^{H} \sum_{u=1}^{W} \left(X_{\text{re}}(u, v) - X_{\text{sy}}(u, v) \right)^2}, \tag{2.3}$$

where X_{re} and X_{sy} are the real and synthetic images respectively, and W and H denote the images dimensions.

However, since our goal is to generate synthetic images that are more effective to train a detection method, we will see this is not the best possible choice, for our purposes.

More specifically, we evaluated our approach in conjunction with three commonly used object detectors—DPM [92], an AdaBoost-based detector [93], and a CNN [94]—and we therefore introduce three different similarity functions, each one based on the image features used by one of these methods. We will show that our approach to image generation works best when relying on the distance function corresponding to the detection method.

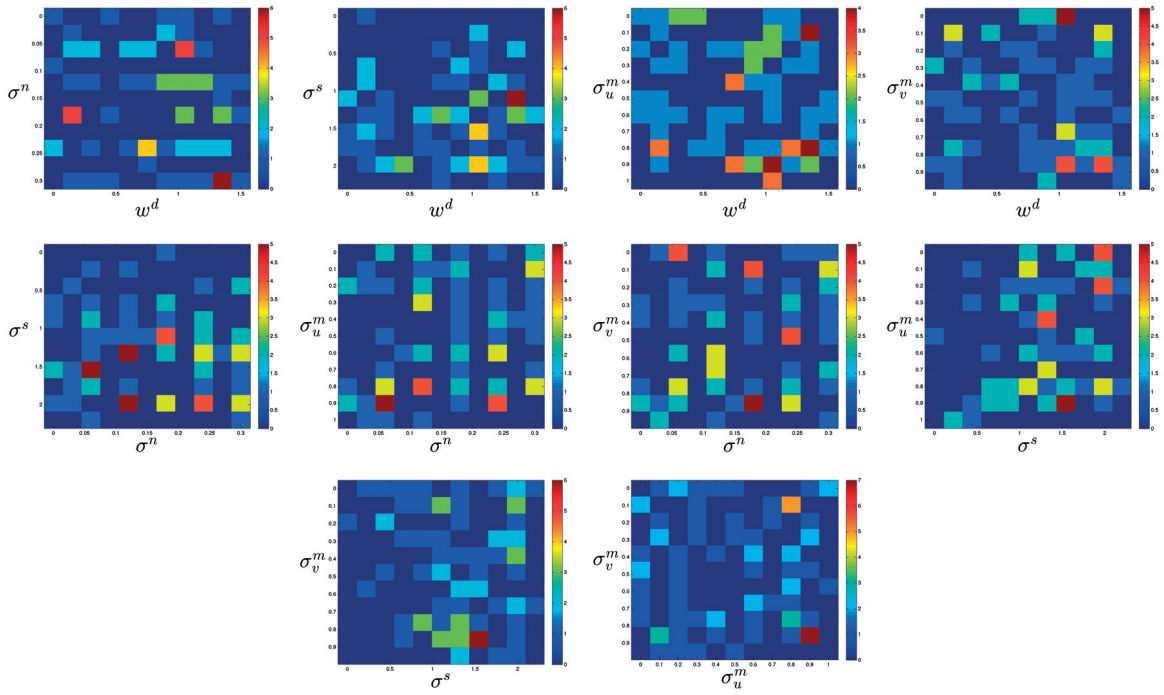


Figure 2.3: Each histogram depicts the joint distribution of different pairs of capture parameters. (best seen in color)

Since DPM relies on Histograms-of-Gradients (HoG) [73], the first similarity function we consider the distance between the HoG vectors [73] computed for the two images as the similarity function

$$d_{\mathrm{HoG}}(X_{\mathrm{re}}, X_{\mathrm{sy}}) = \sqrt{\sum_{i=1}^{L} \left(\mathrm{HoG}_i(X_{\mathrm{re}}) - \mathrm{HoG}_i(X_{\mathrm{sy}})\right)^2}, \tag{2.4}$$

where $\mathrm{HoG}_i(X)$ is the i^{th} coordinate of the HoG vector computed for image X.

We also consider an AdaBoost detector, whose weak learners rely on the image gradients proposed in [108]. We write

$$h_{R,o,\tau}(X) = \begin{cases} 1, & \text{if } E(X,R,o) > \tau, \\ 0, & \text{otherwise.} \end{cases} \tag{2.5}$$

These weak learners are parametrized by a region R, an orientation o, and a threshold τ. $E(X,R,e)$ is the normalized image gradient energy over region R in X and in orientation o. We therefore introduce the additional function:

$$d_{\mathrm{WL}}^{H}(X_{\mathrm{re}}, X_{\mathrm{sy}}) = \sqrt{\sum_{i=1}^{L} \alpha_i \left(h_i(X_{\mathrm{re}}) - h_i(X_{\mathrm{sy}})\right)^2}, \tag{2.6}$$

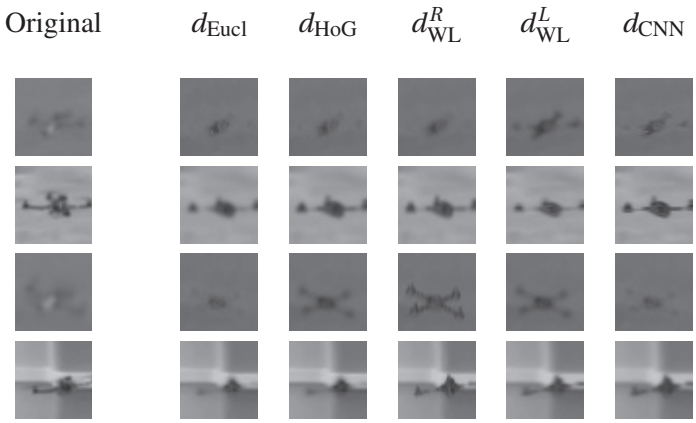


Figure 2.4: Samples of real images with corresponding synthetic ones. The Θ parameters for the synthetic images were optimized using different image similarity functions.

where L is the number of weak learners h_i with their corresponding weights α_i. We tried two different methods to build such a set:

- $d^R_{\text{WL}}(X_{\text{re}}, X_{\text{sy}})$ will denote the previous similarity function when random weak learners, each with a weight $\alpha = 1$, are used;
- $d^L_{\text{WL}}(X_{\text{re}}, X_{\text{sy}})$ will denote the previous similarity function when a set of weak learners and their weights selected by AdaBoost on the seed real images is used.

The third detection method we consider is a Convolutional Neural Network (CNN) which, unlike the previous two, does not rely on hard-coded image features but learns them instead. We therefore first train a CNN on the real seed images only and consider the distance

$$d_{\text{CNN}}(X_{\text{re}}, X_{\text{sy}}) = \sqrt{\sum_{n=2}^{N} \sum_{i=1}^{L_n} \left(\text{CNN}_i^n(X_{\text{re}}) - \text{CNN}_i^n(X_{\text{sy}})\right)^2}, \tag{2.7}$$

where $\text{CNN}_i^n(X)$ is the value of the i^{th} neuron of the n^{th} layer of the Convolutional Neural Network; N is the number of layers in the CNN; L_n is the number of neurons of the n^{th} layer of CNN.

In Fig. 2.4, we show synthetic images with the corresponding real seed images. Each image was obtained by finding the rendering parameters that minimize one of the five similarity functions introduced above.

2.5 Results

In this section, we first introduce the three datasets that we used for training and testing of our algorithms. Then we compare our synthetic data generation approach with several baselines, and

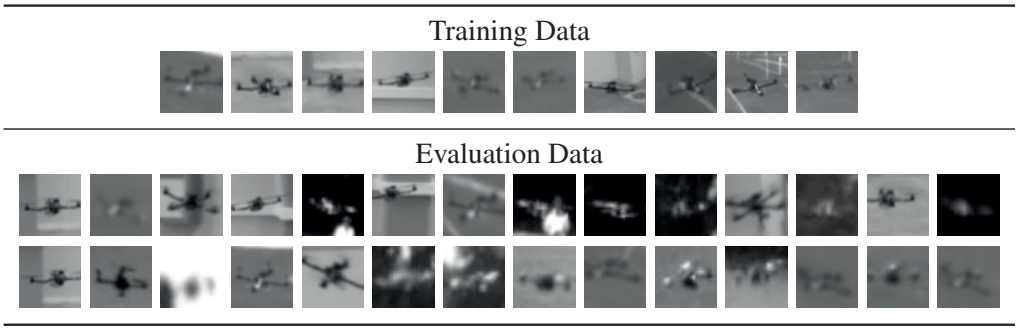


Figure 2.5: Sample real images both for training and evaluation from the UAV dataset. The evaluation images, while created using a single UAV, are very challenging as they are low-resolution while exhibiting significant lighting, background, and pose variations.

evaluate the importance of each of our rendering effects. Our next step is to show the significance of the optimization of the Θ rendering parameters. Further, we experimentally estimate the optimal ratio between synthetic and real samples used for training. We then show that our algorithm is able to generalize to multiple kinds of aircrafts. Finally, we compare our approach to a very recent one on realistic data generation on the **PASCAL VOC** dataset.

We evaluate our synthetic data generation method on three different datasets.

- **UAV Dataset.** This dataset contains challenging images that were acquired from the camera of a flying UAV. In these low-resolution images one can see another drone that flies around and appears against different backgrounds and under various lighting conditions. Even though only one drone was used to produce the images, the dataset includes many of the challenges that outdoor environments pose, such as large illumination and background changes. We use it to investigate the impact of the different effects our rendering pipeline includes.
- **Aircraft dataset.** This dataset contains images of different planes seen against changing backgrounds and under a variety of weather and lighting conditions. We use it to demonstrate that our approach generalizes to a much larger class of objects than simply drones. As in the case of the UAV dataset, we will demonstrate that regardless of the machine learning method used to detect the target objects, we can improve performance by appropriately generating our synthetic images.
- **PASCAL VOC 2007.** We use this standard Computer Vision benchmark to compare our approach to a very recent work on synthetic view generation [41]. As in [41], we restrict ourselves here to the car class, which nevertheless further demonstrates the versatility of our approach.

We will present our results in terms of both recall (r) vs precision (p) curves and *average precision* (AP). Some additional results and video sequences can be found on the webpage of the

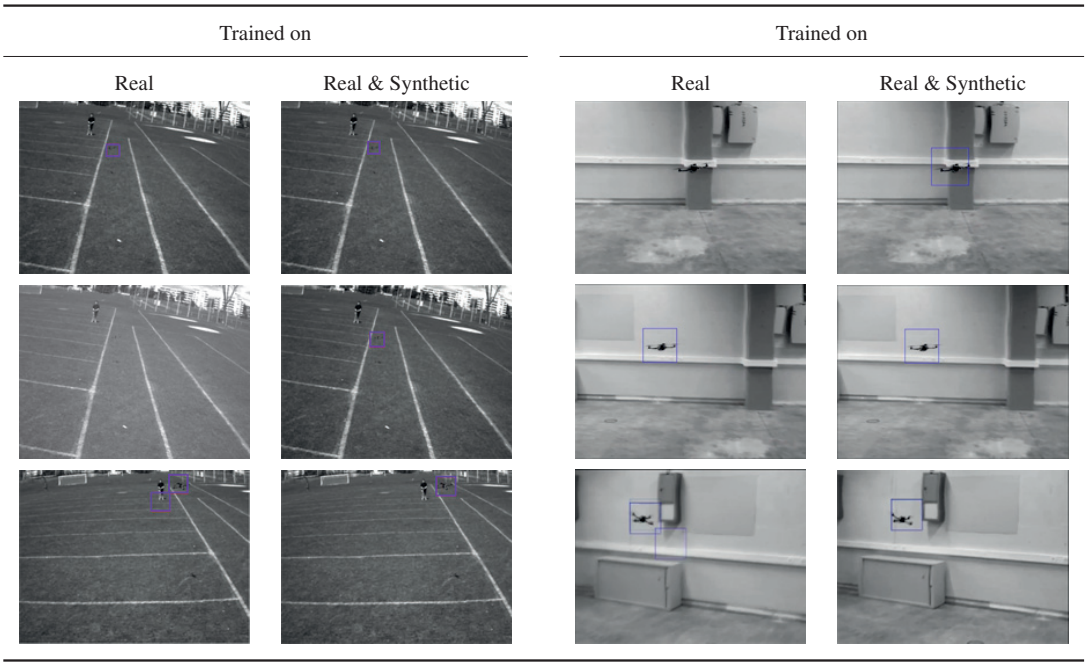


Figure 2.6: Qualitative comparison of the performance of the detectors trained just on real data versus both real and synthetic data.

project[1].

2.5.1 Gauging the Various Components of the Approach using the UAV Dataset

We created a dataset of 2000 images of UAVs in various environments and seen under different lighting conditions. Fig. 2.5 depicts some of the images. The images were captured by one UAV filming another one while they were both flying.

Fig. 2.6 depicts detections by an AdaBoost classifier trained using either real images only or both real and synthetic images. We will quantify the observed performance improvement in the remainder of this section. In Fig. 2.7, we show additional examples of detections by the detector trained on both real and synthetic data as well as some failure cases to illustrate how challenging this dataset is.

We first describe the acquisition process and then use these UAV images to test individual components of our pipeline and to evaluate overall performance.

[1] http://cvlab.epfl.ch/research/unmanned/synthetic

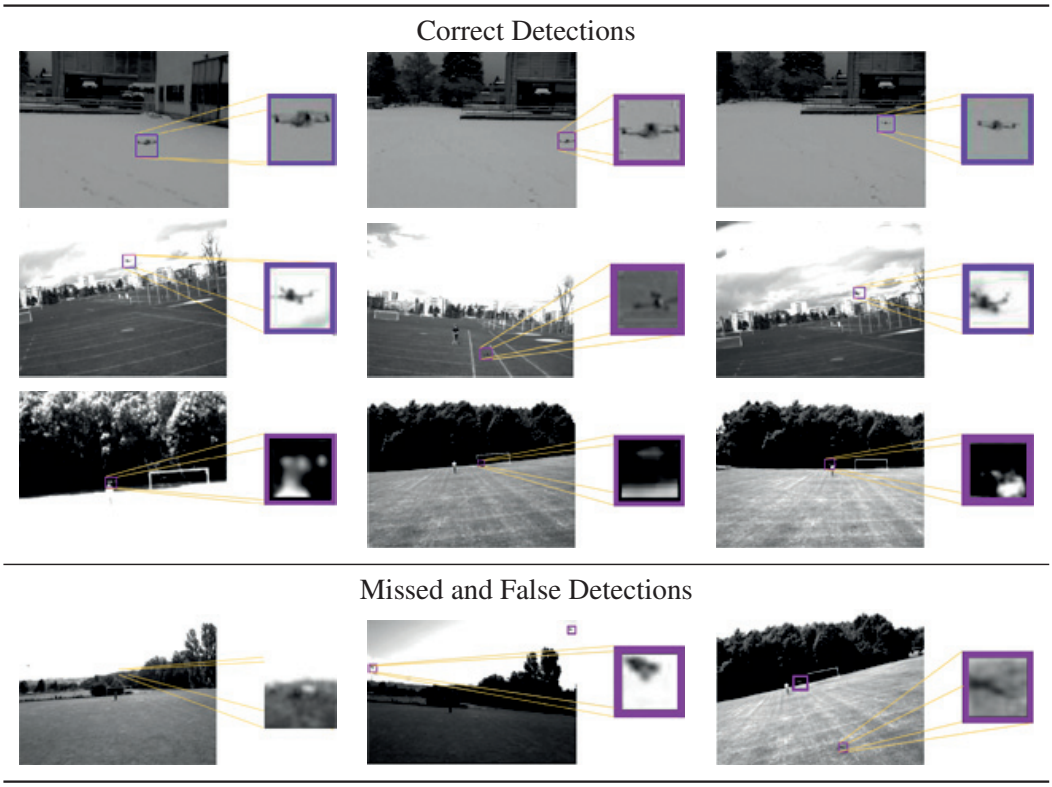


Figure 2.7: [TOP] Sample detections by the AdaBoost detector trained using both real and synthetic data. [BOTTOM] Missed or false detections to highlight the challenges that the dataset provides for the detector. (best seen in color)

2.5.1.1 Experimental Setup

To obtain the background images required to render the composite ones, we first aligned consecutive frames by computing the homographies between the frames, and kept the median intensity at each location of the aligned images.

The training and testing videos were acquired in different environments and feature different backgrounds. The CAD model of the UAV used for rendering only coarsely outlines the main geometrical structure of the real object, as illustrated by Fig. 2.1. Negative training and testing samples were obtained by randomly sampling the backgrounds of the training images. For detection, we use a sliding window approach that applies the detector at every spatial location and at different scales of the whole image. Non-maximum suppression is then applied to the response image scale-space.

The detection methods in the experiments are trained with a combination of real and synthetic data and tested on the real data only.

	Using real images only	By perturbing the real images	Our method
Detection method:	Average precision:		
DPM	0.84	0.87	**0.93**
AdaBoost	0.80	0.83	**0.92**
CNN	0.85	0.86	**0.89**

Table 2.1: Comparing average precisions for each detection method when either perturbing real training images or using our approach with the optimal number of synthetic images and the appropriate distance measure. Our approach significantly outperforms this traditional method.

2.5.1.2 Comparing against simply Perturbing the Real Images

A broadly used approach to augmenting a training set is to perturb the available images using simple image transformations [47, 48]. Table 2.1 compares the performances of all three selected detectors when being trained on images generated either in this way or using our approach. The perturbations involve combining rotation, translation, mirroring, blurring and adding noise to the original images.

Our approach significantly outperforms this simple technique. This can be explained by the fact that we generate realistic combinations of 3D poses and background that are not present in the seed images.

2.5.1.3 Relative Importance of the Various Rendering Effects

To demonstrate that correctly setting each one of the capture parameters introduced in Section 2.2 truly matters, we performed the two sets of experiments. For the first one we suppress the influence of all the introduced effects (Object Boundary blurring, Motion blurring, Random noise, Material properties) by setting the values of the corresponding capture parameters in Θ to 0. We then vary only the parameters that correspond to a single post-processing effect and repeat this experiment with different values of this parameter, and for each of the effects. Table 2.2 summarizes the results of this experiment, which show that all the classifiers benefit from the application of every single post-processing effect.

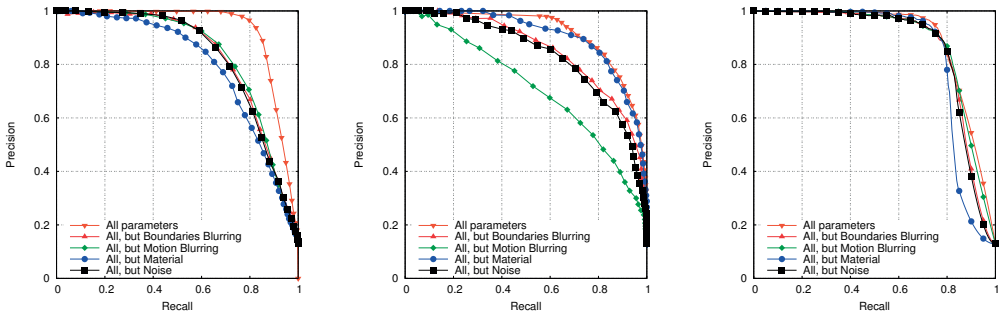
For the second experiment we proceed as follows. For each effect we set the corresponding value in the capture parameters Θ of Eq. 2.1 to 0 to cut off its influence. Further we optimize the other parameters using the appropriate similarity measure for each detection method. We then use the resulting Θ's to generate the synthetic images and train the corresponding detector on these images. Fig. 2.8 illustrates the evaluation results, which prove that correctly modeling each effect clearly has a positive influence on final performance.

Classification method	Average precision			
		Boundaries blurring:		
	No effects	$\sigma^s = 1$	$\sigma^s = 1.5$	$\sigma^s = 2$
DPM	0.78	0.77	**0.84**	0.75
AdaBoost	0.65	0.73	**0.79**	**0.79**
CNN	0.86	**0.89**	0.85	0.86
		Motion blurring:		
	No effects	$\sigma^m_u = 0.3$ $\sigma^m_v = 0.3$	$\sigma^m_u = 0.5$ $\sigma^m_v = 0.5$	$\sigma^m_u = 1$ $\sigma^m_v = 1$
DPM	0.78	**0.84**	0.81	0.79
AdaBoost	0.65	0.72	**0.79**	**0.79**
CNN	0.86	0.87	0.88	**0.89**
		Random noise:		
	No effects	$\sigma^n = 0.5$	$\sigma^n = 0.9$	$\sigma^n = 1.1$
DPM	0.78	**0.83**	0.81	**0.83**
AdaBoost	0.65	0.75	**0.79**	0.75
CNN	0.86	**0.89**	0.86	0.85
		Material properties:		
	No effects	$w^d = 0.5$	$w^d = 1$	$w^d = 2$
DPM	0.78	0.83	0.84	**0.86**
AdaBoost	0.65	0.70	**0.76**	0.66
CNN	0.86	0.88	**0.89**	0.81

Table 2.2: Influence of various post-processing effects on the detection accuracy of different detectors. More specifically, we remove the influence of all the post-processing effects by setting the corresponding capture parameter to zero and then vary the influence of only one of capture parameters to investigate the influence of the respective effect.

2.5.1.4 Importance of Optimizing over the Rendering Parameters

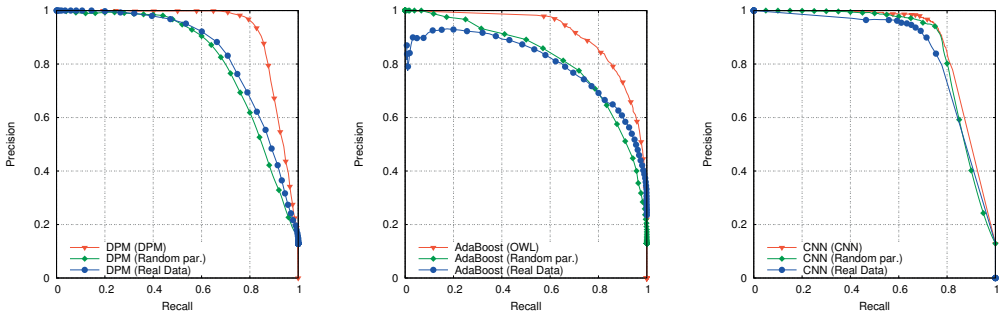
To show the importance of optimizing over the capture parameters Θ, we compare in Fig. 2.9 the final performance obtained using optimized parameters with the final performance obtained with random parameters drawn from a uniform distribution. The minimum and maximum values of the uniform distribution were taken as the minimum and maximum values of the optimized parameters. Our optimization-based approach clearly brings a significant improvement.



DPM and d_{HoG} | AdaBoost and d^{L}_{WL} | CNN and d_{CNN}

	Synthetic data generation effects:				
	no **BB**	no **MB**	no **RN**	no **MP**	All
Detection method:	Average precision:				
DPM	0.83	0.84	0.83	0.80	**0.93**
AdaBoost	0.85	0.71	0.75	0.91	**0.92**
CNN	0.88	**0.89**	0.88	0.85	**0.89**

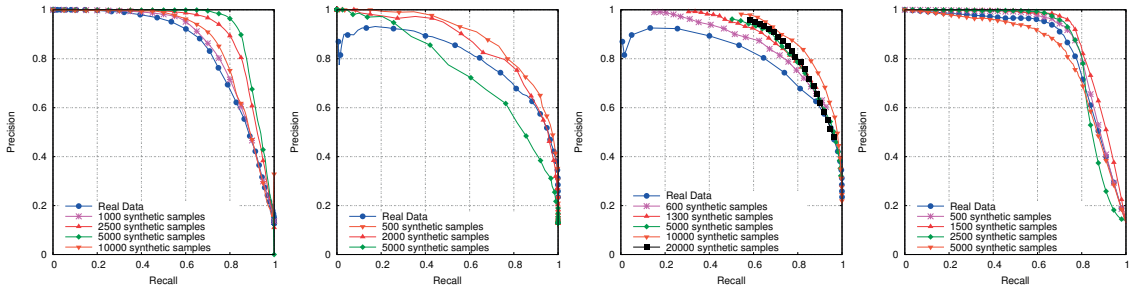
Figure 2.8: Evaluating the importance of each capture parameter. Each one clearly has a positive influence of the quality of the synthetic data. However, their respective impacts depends on the specific detection method. (best seen in color)



DPM | AdaBoost | CNN

	Using real images only	Random parameters	Optimized parameters
Classification method:	Average precision:		
DPM	0.84	0.82	**0.93**
AdaBoost	0.80	0.82	**0.92**
CNN	0.85	0.87	**0.89**

Figure 2.9: Comparison of the performances of different detectors trained on real and synthetic data generated using corresponding similarity measures with those where the capture parameters are randomly selected. The optimized parameters always yield better performance. (best seen in color)



DPM and d_{HoG} | AdaBoost and d^R_{WL} | AdaBoost and d^L_{WL} | CNN and d_{CNN}

Figure 2.10: We varied the number of synthetic images used for training, for three detection methods, using their corresponding similarity measures. Using both real and synthetic data for training phase increases performances compared to real data used alone. However using too much synthetic data may also hurt. (best seen in color)

2.5.1.5 Influence of the Number of Synthetic and Real Images

To evaluate how much we can improve the performances using synthetic images generated with our approach, we trained each of the detection methods we consider with different numbers of synthetic samples in addition to the real training samples. For each detector, the synthetic samples were generated using the parameters obtained using the appropriate similarity functions.

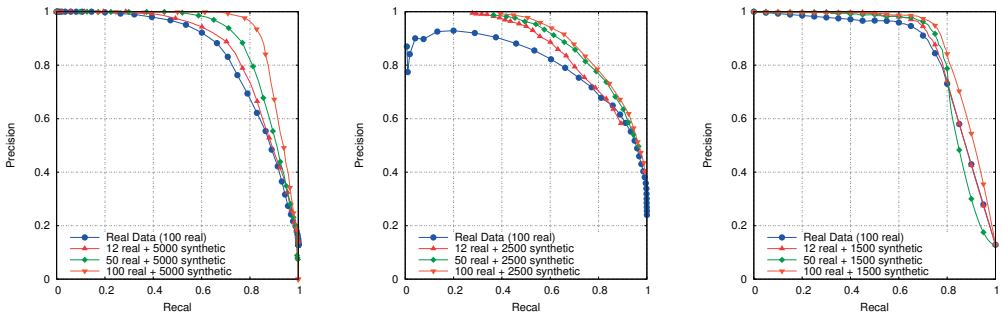
Fig. 2.10 compares the performances of these detectors when varying the number of synthetic samples. We can see that using the synthetic images significantly improves performance over using the real images alone. However, this is only true up to a point. When there are too many synthetic images, the performance eventually *decreases* because the influence of the real images gets drowned out. In practice, this means that for best performance, it makes sense to use a validation set to ascertain the optimal ratio of synthetic to real images.

From these experiments we can conclude that the best ratio of synthetic and real examples that should be used for training depends on the detection algorithm. AdaBoost achieves its highest accuracy with 100 synthetic images for each real one, DPM with 50 synthetic images for each real one, and CNN with $15-20$ synthetic images for each real one.

We also evaluated the influence of the number of seed real images on the final performances, by decreasing the number of real images used to optimize the rendering parameters. Fig. 2.11 shows the results for the AdaBoost detector. Using as few as 12 real samples is enough to generate synthetic samples that allows us to outperform a detector trained with about 8 times as many real images. Unsurprisingly, increasing the number of seed real images results in an improvement of the final performances.

		Similarity measure:				
	Using real images only	d_{Eucl}	d_{HoG}	d^{R}_{WL}	d^{L}_{WL}	d_{CNN}
Detection method:		Average precision:				
DPM	0.84	0.78	**0.93**	0.70	0.72	0.67
AdaBoost	0.80	0.72	0.85	0.89	**0.92**	0.75
CNN	0.85	0.84	0.84	0.84	0.86	**0.89**

Table 2.3: Comparison of average precisions for each detection method, when the optimal number of synthetic images is used. Each detection method performs best with the corresponding similarity function.



DPM and d_{HoG} AdaBoost and d^{L}_{WL} CNN and d_{CNN}

Figure 2.11: Performance of the DPM, AdaBoost, and CNN detectors for different numbers of seed real images. Using as few as 12 real samples is enough to generate synthetic samples that allow us to outperform a detector trained with 100 images. For each detector, the synthetic images were generated using the corresponding similarity measure.

2.5.1.6 Optimal Performance

In this section, for each detection method, we use the optimal numbers of synthetic samples as discussed in the previous section. For comparison purposes, we also estimated the optimal numbers of synthetic samples when using the Euclidean distance d_{Eucl} as similarity measure.

Table 2.3 confirms that each detection method performs best when trained using synthetic images, generated using appropriate similarity measure, as discussed in Section 2.4. In particular, using the Euclidean distance is not only ineffective, but actually yields worse results than not using synthetic images at all. Interestingly, the best performance is obtained with DPM trained with both real and synthetic images, even though CNN was better than DPM when no synthetic images were used.

Figure 2.12: The three publicly available CAD models we used for the Aircraft dataset.

(a) Real samples (b) Synthetic samples

Figure 2.13: Sample (a) real and (b) generated synthetic images from the Aircraft dataset.

2.5.2 Detecting Multiple Kinds of Aircrafts

For the Aircraft dataset, we generated synthetic data using CAD models depicted by Fig. 2.12 of three types of fixed-wing aircrafts and tested them on different real video sequences. We use 100 real images of these three aircraft types along with their corresponding background images. These images were collected by manually annotating different video sequences where the aircrafts fly in different weather conditions and appear at different angles. Sample images from this dataset are shown on Fig. 2.13(a).

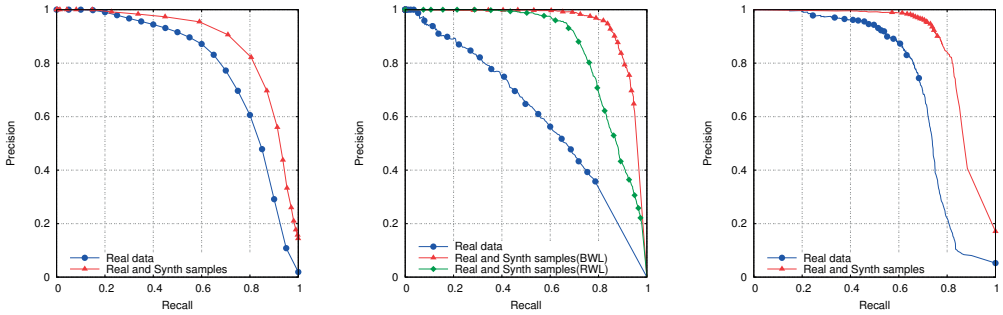
Here, we used an AdaBoost detector trained using real and synthetic images generated based on the d^L_{WL} similarity function of Section 2.4. We generated 10000 synthetic samples to supplement the real images and used them as a training dataset. Fig. 2.13(b) depicts sample synthetic images.

The test images come from 8 video sequences, one of which contains 5000 frames, while the others are made of 500 frames. These sequences show different types of aircrafts flying in different environments and weather conditions. In Table 2.4 and Fig. 2.14, we compare results using real images only against an optimal combination of real and synthetic images.

Using the detector trained on both real and synthetic images we achieve about 90% detection accuracy, as opposed to approximately 65% when using real images only. This large improvement can be explained by the fact that we have only 100 real images containing three different models, while we generated 100 images for each real seed image, which results in total in 10000 positive examples. Table 2.4 illustrate the best accuracy one can get varying the number of synthetic samples being added to the training set. Sample detections are shown in Fig. 2.15, which also depicts some failure cases.

		Similarity measure:				
	Using real images only	d_{Eucl}	d_{HoG}	d^R_{WL}	d^L_{WL}	d_{CNN}
Detection method:	Average precision:					
DPM	0.79	0.83	**0.88**	0.85	0.86	0.81
AdaBoost	0.65	0.75	0.84	0.87	**0.92**	0.73
CNN	0.72	0.75	0.85	0.70	0.83	**0.88**

Table 2.4: Comparing average precisions for each detection method when the optimal number of synthetic images is used for the Aircraft dataset. Each detection method performs best when using the corresponding similarity function.



DPM and d_{HoG} AdaBoost and $d^{L,R}_{\text{WL}}$ CNN and d_{CNN}

Figure 2.14: Comparing the accuracy of the detectors trained with real images only (blue) against those trained using both synthetic and real images (red or green). AdaBoost in particular does not perform very well with only the real images because 100 samples are not enough to train the algorithm to detect 3 different kinds of aircrafts. Nevertheless, introducing synthetic images lets us enrich the training set to the point where performance improves substantially in all cases. In the AdaBoost case, the red curve is obtained by using the d^L_{WL} similarity measure and the green one the d^R_{WL} similarity measure.

2.5.3 Comparing against another Image-Based Synthesis Approach

We compare here our approach with the one of [41], which was applied to car detection on the PASCAL VOC car dataset. Like ours, it uses a CAD model of the target object and seed real images. Its main contributions are the estimation of the material properties of every car component in a real image and the exploitation of this information to generate new synthetic views, which are then used to supplement real ones to train a DPM detector. This requires registration of the model so that it precisely fits the car in the image and the image texture can then back-projected onto the car model, so that material properties can be assigned to each visible part of the model. To generate a new view, the model is rotated in 3D and re-projected in the scene, which includes the ground plane and the background plane. This ends up making some previously invisible parts of the car visible. Material properties for these newly visible parts are estimated using a weighted sum of the properties of the parts whose material properties have

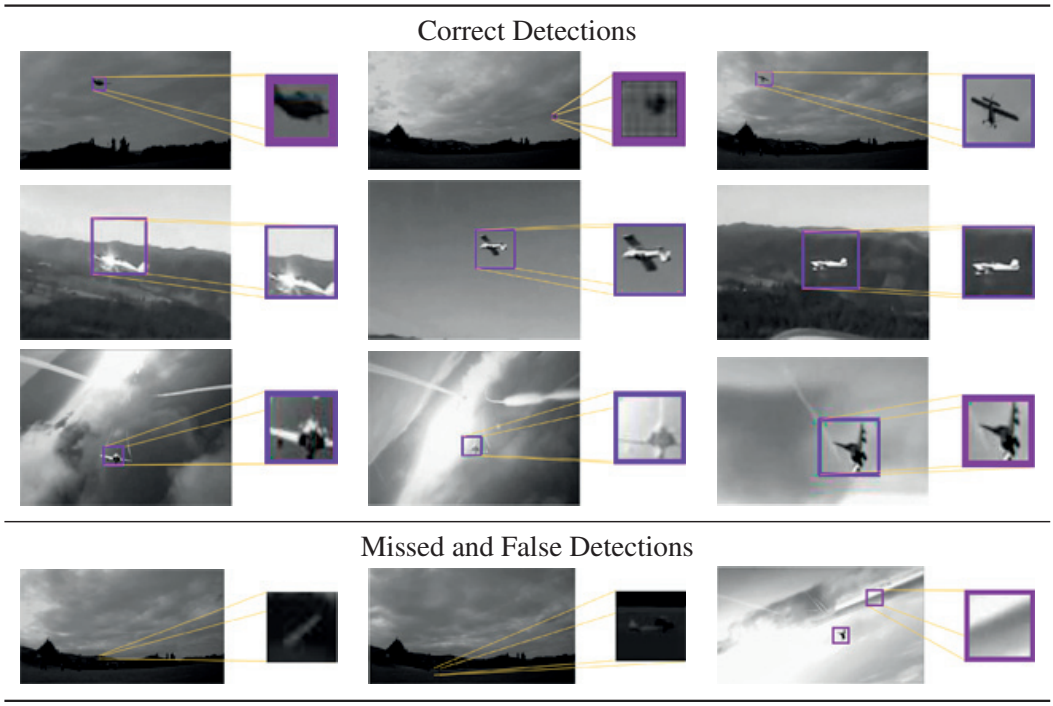


Figure 2.15: Examples of detections and some errors made by the detector, trained on both real and synthetic samples, and evaluated on the Aircraft dataset.

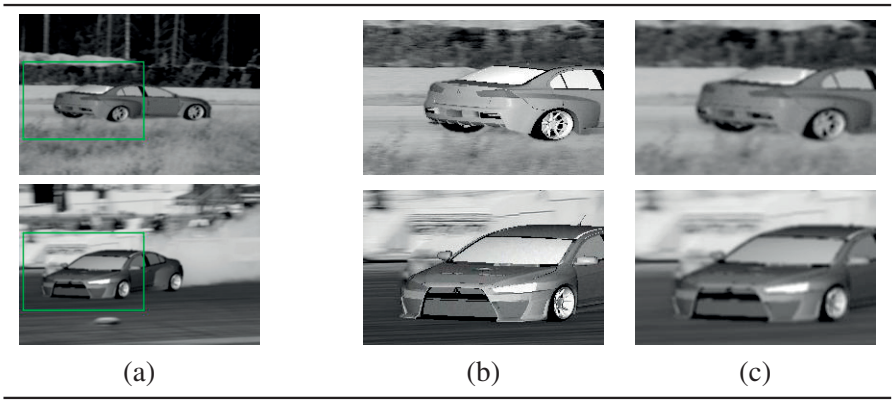


Figure 2.16: (a) Sample synthetic images of cars generated by our approach. (b, c) Patches extracted from these images emphasizing the importance of the boundary and motion blurring effects.

already been estimated.

Since, our algorithm requires background images in addition to the model and seed real images, we derived them from the seed images by cutting out the car and filling the empty space using content aware texture filling [109]. Sample synthetic images, generated by our system are shown in Fig. 2.16.

Test set	Training set	Avg. Precision		
	Number of components:	$N = 3$	$N = 4$	$N = 5$
VOC 2007	Side	16.2	18.4	16.7
	Side+Synth [41]	30.2	31.4	33.2
	Side+Synth (Our method)	**35.1**	**37.9**	**38.0**
	Full	51.7	53.4	50.7
	Full+Synth [41]	50.2	**53.1**	50.9
	Full+Synth (Our method)	**52.1**	52.9	**55.3**

Table 2.5: Comparing the performances of the DPM detector on the PASCAL VOC car dataset when trained with the method of [41] and our method as a function of N, the number of DPM components. The performance of the detectors trained using only the real images is also given for reference.

The images of the Pascal VOC dataset being in color, for a fair comparison against [41] that exploits this fact, we extended our approach to color images by simply optimizing on the Θ parameters on the three RGB channels independently, which yields three sets of parameters for every image. These parameters are then used to generate separate synthetic images for every channel, and finally combined in one RGB image. We vary the pose of the car model, but also the direction of the light source, which cannot be done with [41]. The results of this combination are presented in Fig. 2.17. The car in the second column of Fig. 2.17 does not look very realistic, because the same properties are applied to all the components of the car. A more sophisticated model would solve this issue, however we already obtain satisfying results using this simplistic rendering, which confirms that producing visually pleasing synthetic images is not a primary requirement. Some detections made by the 5 component DPM framework, trained on both real and synthetic data are presented in the Fig. 2.18.

Table 2.5 shows that we outperform [41] even though our approach was originally designed to generate small image patches centered on the target object. Furthermore, as shown in the previous sections, it is applicable to low-resolution images with very limited texture, for which the method of [41] is not well adapted. Furthermore, if we don't use color, the performance drops by only 1-2%, which is not very large.

2.6 Conclusion

We have shown that by properly optimizing the parameters of a very simple rendering pipeline, we can generate synthetic images that significantly improve the performance of an object detector when used for training. We believe our parameter optimization scheme is a powerful tool to manage the large numbers of parameters a more complex rendering pipeline could have.

Figure 2.17: Sample synthetic RGB images of cars, generated by our system. Images in the second column do not look very realistic, because the same properties are applied to all the components of the car. A more sophisticated model could be used to address this issue. However realism does not seem to be critical for our purposes since our simplistic model is sufficient to outperform [41].

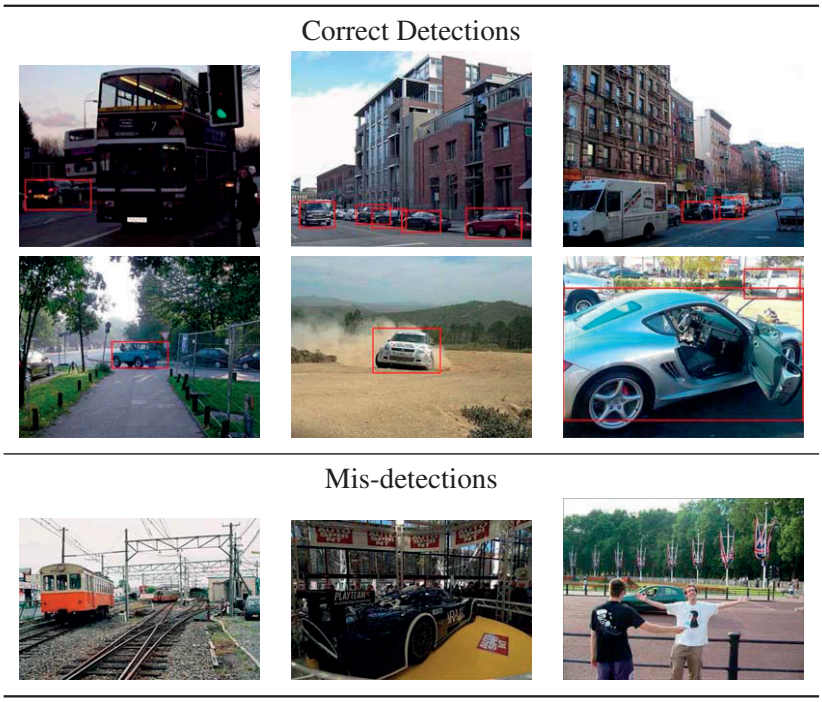


Figure 2.18: Sample detections made by the 5 component DPM, trained on the full real VOC dataset, supplemented by synthetic data, generated by our method. Last row shows the missed detections. (best seen in color)

3 Domain Adaption for Deep Networks

In the previous chapter we have presented a way to generate synthetic images that can further be used to supplement a small amount of real samples for training a detector. From the experiments in Section 2.5 we can see that our method achieves a relatively large performance improvement for AdaBoost and DPM methods, however, does not generalize well to Convolutional Neural Networks (CNNs), which have recently become the state-of-the-art in various areas of computer vision [30, 43, 50, 44]. This happens because, with the increase of the amount of synthetic data, the CNNs start to overfit to the synthetic images at the cost of the real ones. Therefore, in this chapter we introduce a novel approach that learns from a combination of real and synthetic images without overfitting to either of the two.

The problem of efficient leveraging of synthetic data can be viewed as a sub-case of a more general one, which is transferring a classification model, learned on one set of images (e.g. synthetic data, for which enough training data is available) to classify images from a related but different dataset (e.g. real images, where only very small amounts of additional annotations, or even none can be acquired). Domain Adaptation [110] and Transfer Learning [111] have long been used to overcome this problem. Recently, Domain Adaptation has been investigated in the context of Deep Learning with promising results [52, 53, 54, 55, 56, 57]. Following the classic Domain Adaptation trend these methods use the same deep architecture with the same weights for both source and target domains, which essentially means learning domain invariant features.

In this chapter, we show that imposing feature invariance is detrimental to discriminative power of the final model. Therefore instead of making features invariant to the domain shift we propose to explicitly model it. To this end, we introduce the two-stream architecture depicted by Fig. 3.1. One stream operates on the source domain and the other on the target one. This makes it possible *not* to share the weights in some of the layers. Instead, we introduce a loss function that is lowest when they are linear transformations of each other. Furthermore, we introduce a criterion to automatically determine which layers should share their weights and which ones should not. In short, our approach explicitly models the domain shift by learning features adapted to each domain, but not fully independent, to account for the fact that both domains depict the same

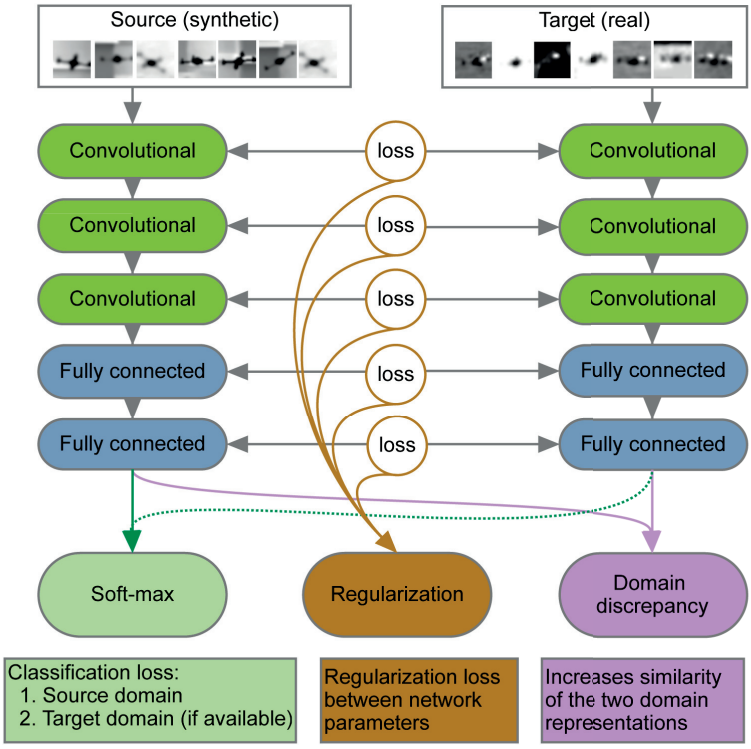


Figure 3.1: **Our two-stream architecture.** One stream operates on the source data and the other on the target data. Their weights are *not* shared. Instead, we introduce loss functions that prevent corresponding weights from being too different from each other.

underlying problem.

We demonstrate that our approach is well suited to leveraging synthetic data to increase the performance of the classifier on real images. Here, we treat the synthetic images as forming the source domain and the real images the target one. We then make use of our two-stream architecture to learn an effective model for the real data even though we have only few annotations for it. We further show that our method is more effective than state-of-the-art weight-sharing schemes on standard Domain Adaptation benchmarks for image recognition. Finally we demonstrate the effectiveness of our approach at leveraging synthetic data for facial pose estimation.

Aforementioned examples prove the generality of our method, as the first two applications involve solving a classification task and the third one – regression problem. We outperform the state-of-the-art methods in all these cases, and our experiments support our contention that specializing the network weights outperforms sharing them.

In the remainder of this chapter we first briefly review some recent trends in Domain Adaptation, with a focus on Deep Learning based methods, which are the most related to our work. In Section 3.2 we then present our approach and discuss the experimental results in Section 3.3.

3.1 Related Work

A natural approach to Domain Adaptation is to modify a classifier trained on the source data using the available labeled target data. This was done, for example, using SVM [112, 113], Boosted Decision Trees [114] and other classifiers [115]. In the context of Deep Learning, fine-tuning [52, 53] essentially follows this pattern. In practice, however, when only a small amount of labeled target data is available, this often results in overfitting.

Another approach is to learn a metric between the source and target data, which can also be interpreted as a linear cross-domain transformation [116] or a non-linear one [117]. Instead of working on the samples directly, several methods involve representing each domain as one separate subspace [118, 119, 120, 121]. A transformation can then be learned to align them [120]. Alternatively, one can interpolate between the source and target subspaces [118, 119, 121]. In [122], this interpolation idea was extended to Deep Learning by training multiple unsupervised networks with increasing amounts of target data. The final representation of a sample was obtained by concatenating all intermediate ones. It is unclear, however, why this concatenation should be meaningful to classify a target sample.

Another way to handle the domain shift is to explicitly try making the source and target data distributions similar. While many metrics have been proposed to quantify the similarity between two distributions, the most widely used in the Domain Adaptation context is the Maximum Mean Discrepancy (MMD) [123]. The MMD has been used to re-weight [124, 125] or select [126] source samples such that the resulting distribution becomes as similar as possible to the target one. An alternative is to learn a transformation of the data, typically both source and target, such that the resulting distributions are as similar as possible in MMD terms [127, 128, 129]. In [130], the MMD was used within a shallow neural network architecture. However, this method relied on SURF features [131] as initial image representation and thus only achieved limited accuracy.

Recently, using Deep Networks to learn features has proven effective at increasing the accuracy of Domain Adaptation methods. In [132], it was shown that using DeCAF features instead of hand-crafted ones mitigates the domain shift effects even without performing any kind of adaptation. However, performing adaptation within a Deep Learning framework was shown to boost accuracy even further [133, 54, 55, 56, 57, 134, 135]. For example, in [133], a Siamese architecture was introduced to minimize the distance between pairs of source and target samples, which requires training labels available in the *target* domain thus making the method unsuitable for unsupervised Domain Adaptation. The MMD has also been used to relate the source and target data representations learned by Deep Networks [54, 55] thus making it possible to avoid working on individual samples. [56, 57] introduced a loss term that encodes an additional classifier predicting from which domain each sample comes. This was motivated by the fact that, if the learned features are domain-invariant, such a classifier should exhibit very poor performance.

All these Deep Learning approaches rely on the same architecture with the same weights for both the source and target domains. In essence, they attempt to reduce the impact of the domain shift

by learning domain-invariant features. In practice, however, domain invariance might very well be detrimental to discriminative power. As discussed in the introduction, this is the hypothesis we set out to test in this work by introducing an approach that explicitly models the domain shift instead of attempting to enforce invariance to it. We show in the results section that this yields a significant accuracy boost over networks with shared weights.

3.2 Our Approach

The core idea of our method is that, for a Deep Network to adapt to different domains, the weights should be related, yet different for each of the two domains. As shown empirically, this constitutes a major advantage of our method over the competing ones discussed in Section 3.1. To implement this idea, we therefore introduce a two-stream architecture, such as the one depicted by Fig. 3.1. The first stream operates on the source data, the second on the target one, and they are trained jointly. While we allow the weights of the corresponding layers to differ between the two streams, we prevent them from being too far from each other. Additionally we use the MMD between the learned source and target representations. This combination lets us encode the fact that, while different, the two domains are related.

More formally, let $\mathbf{X}^s = \{\mathbf{x}_i^s\}_{i=1}^{N^s}$ and $\mathbf{X}^t = \{\mathbf{x}_i^t\}_{i=1}^{N^t}$ be the sets of training images from the source and target domains, respectively, with $Y^s = \{y_i^s\}$ and $Y^t = \{y_i^t\}$ being the corresponding labels. To handle unsupervised target data as well, we assume, without loss of generality, that the target samples are ordered, such that only the first N_l^t ones have valid labels, where $N_l^t = 0$ in the unsupervised scenario. Furthermore, let θ_j^s and θ_j^t denote the parameters, that is, the weights and biases, of the j^{th} layer of the source and target streams, respectively. We train the network by minimizing a loss function of the form

$$L(\theta^s, \theta^t | \mathbf{X}^s, Y^s, \mathbf{X}^t, Y^t) = L_s + L_t + L_w + L_{MMD}, \tag{3.1}$$

$$L_s = \frac{1}{N^s} \sum_{i=1}^{N^s} c(\theta^s | \mathbf{x}_i^s, y_i^s), \tag{3.2}$$

$$L_t = \frac{1}{N_l^t} \sum_{i=1}^{N_l^t} c(\theta^t | \mathbf{x}_i^t, y_i^t), \tag{3.3}$$

$$L_w = \lambda_w \sum_{j \in \Omega} r_w(\theta_j^s, \theta_j^t), \tag{3.4}$$

$$L_{MMD} = \lambda_u r_u(\theta^s, \theta^t | \mathbf{X}^s, \mathbf{X}^t), , \tag{3.5}$$

where $c(\theta^{\cdot} | \mathbf{x}_i^{\cdot}, y_i^{\cdot})$ is a standard classification loss, such as the logistic loss or the hinge loss. $r_w(\cdot)$ and $r_u(\cdot)$ are the weight and unsupervised regularizers discussed below. The first one represents the loss between corresponding layers of the two streams. The second encodes the MMD measure and favors similar distributions of the source and target data representations. These regularizers are weighted by coefficients λ_w and λ_u, respectively. In practice, we found our approach to be robust to the specific values of these coefficients and we set them to 1 in all

our experiments. Ω denotes the set of indices of the layers whose parameters are not shared. This set is problem-dependent and, in practice, can be obtained by comparing the MMD values for different configurations, as demonstrated in our experiments.

3.2.1 Weight Regularizer

While our goal is to go beyond sharing the layer weights, we still believe that corresponding weights in the two streams should be related. This models the fact that the source and target domains are related, and prevents overfitting in the target stream, when only very few labeled samples are available. Our weight regularizer $r_w(\cdot)$ therefore represents the distance between the source and target weights in a particular layer. In principle, we could take it to directly act on the difference of those weights. This, however, would not truly attempt to model the domain shift, for instance to account for different means and ranges of values in the two types of data. To better model the shift and introduce more flexibility in our model, we therefore propose not to penalize linear transformations between the source and target weights. We then write our regularizer either by relying on the L_2 norm as

$$r_w(\theta_j^s, \theta_j^t) = \left\| a_j \theta_j^s + b_j - \theta_j^t \right\|_2^2 , \tag{3.6}$$

or in an exponential form as

$$r_w(\theta_j^s, \theta_j^t) = \exp\left(\| a_j \theta_j^s + b_j - \theta_j^t \|^2 \right) - 1 . \tag{3.7}$$

In both cases, a_j and b_j are scalar parameters that are different for each layer $j \in \Omega$ and learned at training time along with all other network parameters. While simple, this parameterization can account, for example, for global illumination changes in the first layer of the network. As shown in the results section, we found empirically that the exponential version gives better results.

We have tried replacing the simple linear transformation of Eqs. 3.6 and 3.7 by more sophisticated ones, such as quadratic or piecewise linear ones. This, however, did not yield any performance improvement.

3.2.2 Unsupervised Regularizer

In addition to regularizing the weights of corresponding layers in the two streams, we also aim at learning a final representation, that is, the features before the classifier layer, that is domain invariant. To this end, we introduce a regularizer $r_u(\cdot)$ designed to minimize the distance between the distributions of the source and target representations. Following the popular trend in Domain Adaptation [136, 54], we rely on the MMD [123] to encode this distance.

As the name suggests, given two sets of data, the MMD measures the distance between the mean of the two sets after mapping each sample to a Reproducing Kernel Hilbert Space (RKHS). In

our context, let $\mathbf{f}_i^s = \mathbf{f}_i^s(\theta^s, \mathbf{x}_i^s)$ be the feature representation at the last layer of the source stream, and $\mathbf{f}_j^t = \mathbf{f}_j^t(\theta^t, \mathbf{x}_j^t)$ of the target stream. The MMD^2 between the source and target domains can be expressed as

$$\text{MMD}^2(\{\mathbf{f}_i^s\}, \{\mathbf{f}_j^t\}) = \left\| \sum_{i=1}^{N^s} \frac{\phi(\mathbf{f}_i^s)}{N^s} - \sum_{j=1}^{N^t} \frac{\phi(\mathbf{f}_j^t)}{N^t} \right\|^2 , \tag{3.8}$$

where $\phi(\cdot)$ denotes the mapping to RKHS. In practice, this mapping is typically unknown. Expanding Eq. 3.8 and using the kernel trick to replace inner products by kernel values lets us rewrite the squared MMD, and thus our regularizer as

$$r_u(\theta^s, \theta^t | \mathbf{X}^s, \mathbf{X}^t) = \sum_{i,i'} \frac{k(\mathbf{f}_i^s, \mathbf{f}_{i'}^s)}{(N^s)^2} - 2 \sum_{i,j} \frac{k(\mathbf{f}_i^s, \mathbf{f}_j^t)}{N^s N^t} + \sum_{j,j'} \frac{k(\mathbf{f}_j^t, \mathbf{f}_{j'}^t)}{(N^t)^2} , \tag{3.9}$$

where the dependency on the network parameters comes via the $\mathbf{f}_i$s, and where $k(\cdot, \cdot)$ is a kernel function. In practice, we make use of the standard RBF kernel $k(u, v) = \exp\left(-\|u - v\|^2 / \sigma\right)$, with bandwidth σ. In all our experiments, we found our approach to be insensitive to the choice of σ and we therefore set it to 1.

3.2.3 Training

To learn the model parameters, we first pre-train the source stream using the source data only. We then simultaneously optimize the weights of both streams according to the loss of Eqs. 3.2-3.5 using both source and target data, with the target stream weights initialized from the pre-trained source weights. Note that this also requires initializing the linear transformation parameters of each layer, a_j and b_j for all $j \in \Omega$. We initialize these values to $a_j = 1$ and $b_j = 0$, thus encoding the identity transformation. All parameters are then learned jointly using backpropagation with the AdaDelta algorithm [75]. Note that we rely on mini-batches, and thus in practice compute all the terms of our loss over these mini-batches rather than over the entire source and target datasets.

Depending on the task, we use different network architectures, to provide a fair comparison with the baselines. For example, for the *Office* benchmark, we adopt the AlexNet [30] architecture, as was done in [54], and for digit classification we rely on the standard network structure of [47] for each stream.

3.3 Experimental Results

In this section, we demonstrate the potential of our approach in both the supervised and unsupervised scenarios using different network architectures. We first thoroughly evaluate our method for the drone detection task. We then demonstrate that it generalizes well to other classification problems by testing it on the *Office* and *MNIST+USPS* datasets. Finally, to show that our approach also generalizes to regression problems, we apply it to estimating the position of facial

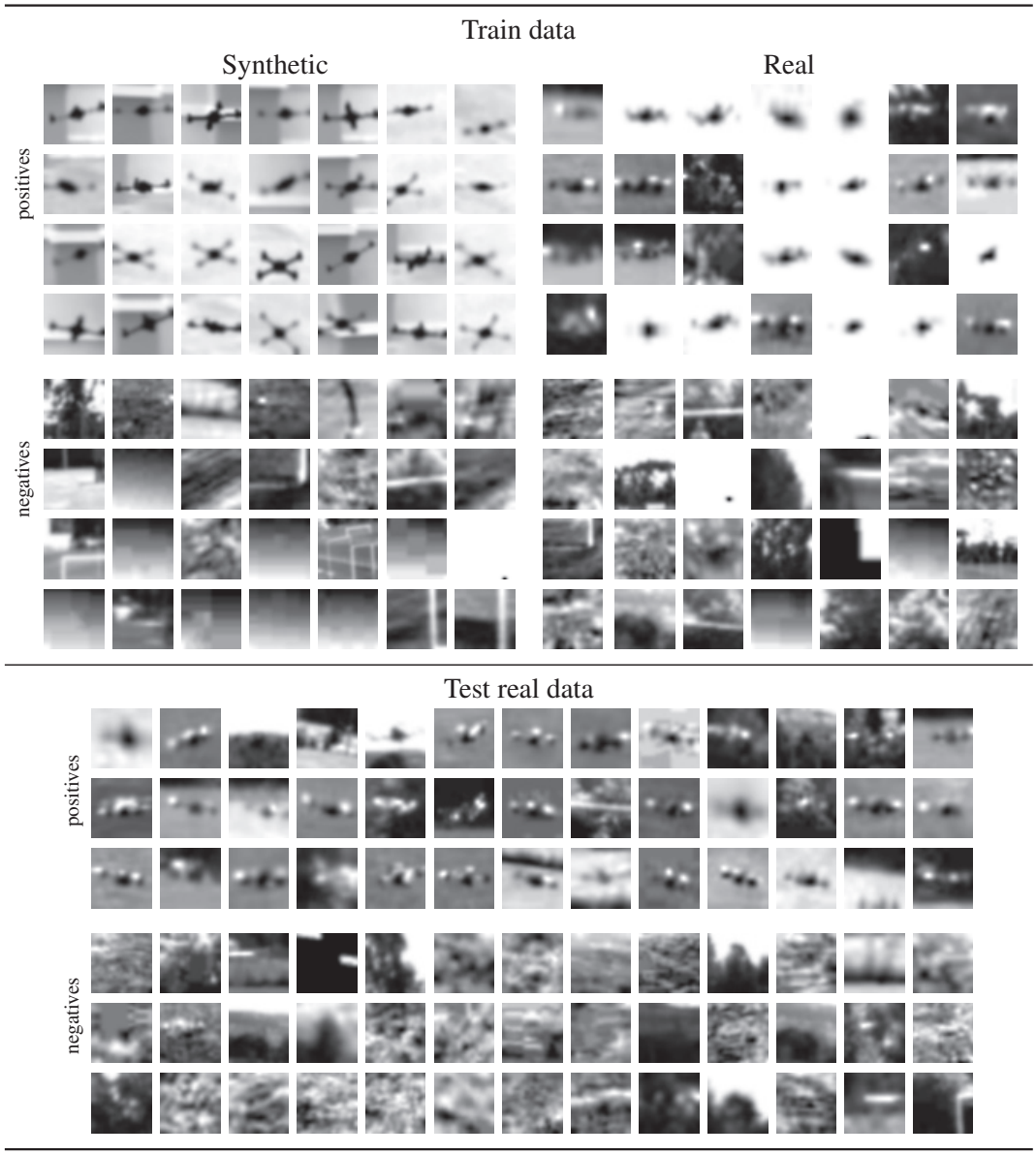


Figure 3.2: Our UAV dataset. [TOP] Synthetic and real training examples. [BOTTOM] Real samples from the test dataset.

landmarks.

3.3.1 Leveraging Synthetic Data for Drone Detection

Due to the lack of large publicly available datasets, UAV detection is a perfect example of a problem where training videos are scarce and do not cover a wide enough range of possible shapes, poses, lighting conditions, and backgrounds against which drones can be seen. However, it is relatively easy to generate large amounts of synthetic examples, which can be used to supplement

Dataset	Training			Testing	
	Pos		Neg	Pos	Neg
	(Real)	(Synthetic)	(Real)	(Real)	(Real)
UAV-200 (full)	200	32800	190000	3100	135000
UAV-200 (small)	200	1640	9500	3100	6750

Table 3.1: Statistics of our two UAV datasets. Note that *UAV-200 (small)* is more balanced than *UAV-200 (full)*.

a small number of real images and increase detection accuracy [49]. We show here that our approach allows us to exploit these synthetic images more effectively than other state-of-the-art Domain Adaptation techniques.

3.3.1.1 Dataset and Evaluation Setup

We used the approach of [49], described in Chapter 2 to create a large set of synthetic examples. Fig. 3.2 depicts sample images from the real and synthetic dataset that we used for training and testing. In our experiments, we treat the synthetic images as source samples and the real images as target ones.

We report results using two versions of this dataset, which we refer to as *UAV-200 (small)* and *UAV-200 (full)*. Their sizes are given in Table 3.1. They only differ in the number of synthetic and negative samples used for training and testing. The ratio of positive to negative samples in the first dataset is better balanced than in the second one. For *UAV-200 (small)*, we therefore express our results in terms of *accuracy*, which is commonly used in Domain Adaptation and can be computed as

$$\text{Accuracy} = \frac{\text{\# correctly classified examples}}{\text{\# all examples}} \ . \tag{3.10}$$

Using this standard metric facilitates the comparison against the baseline methods whose publicly available implementations only output classification accuracy.

In real detection tasks, however, training datasets are typically quite unbalanced, since one usually encounters many negative windows for each positive example. *UAV-200 (full)* reflects this more realistic scenario, in which the accuracy metric is poorly-suited. For this dataset, we therefore compare various approaches in terms of *precision-recall*. Additionally, we report the *Average Precision* (AP) measure.

For this experiment, we follow the supervised Domain Adaptation scenario. In other words, training data is available with labels for both source and target domains.

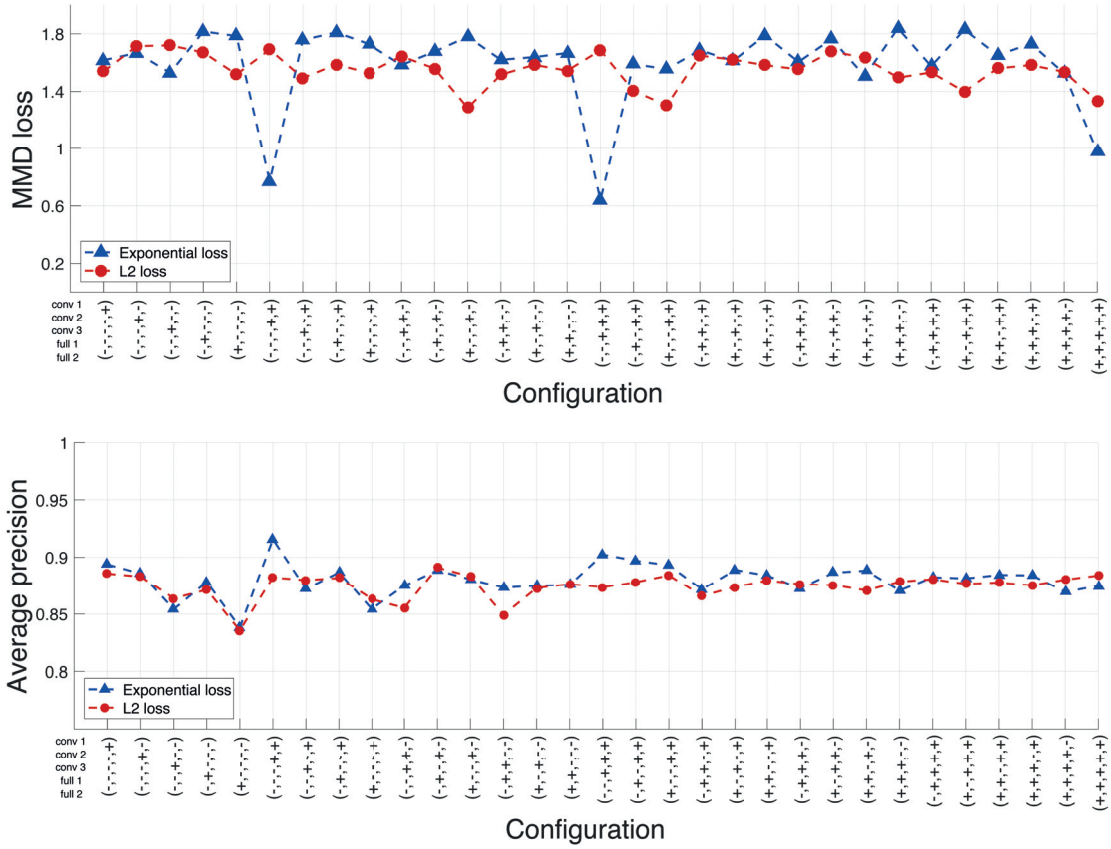


Figure 3.3: Evaluation of the best network architecture. [TOP] The y-axis corresponds to the MMD^2 loss between the outputs of the corresponding streams that operate on real and synthetic data, respectively. [BOTTOM] Here the y-axis corresponds to the AP on validation data (500 positive and 1500 negative examples). Note that low values of MMD tend to coincide with high AP values. The x-axis denotes the network configuration, where a '+' sign indicates that the corresponding network layers are regularized with a loss function and a '−' sign that the weights are shared for the corresponding layers. (Best seen in color)

3.3.1.2 Network Design

Our network consists of two streams, one for the source data and one for the target data, as illustrated by Fig. 3.1. Each stream is a CNN that comprises three convolutional and max-pooling layers, followed by two fully-connected ones. The classification layer encodes a hinge loss, which was shown to outperform the logistic loss in practice for some tasks [84, 85].

As discussed above, some pairs of layers in our two-stream architecture may share their weights while others do not, and we must decide upon an optimal arrangement. To this end, we trained one model for every possible combination. We plot the results in Fig. 3.3 **(top)**, with the + and − signs indicating whether the weights are stream-specific or shared. Since we use a common classification layer, the MMD^2 value ought to be small when our architecture accounts well for

	Accuracy
ITML [116]	0.60
ARC-t assymetric [117]	0.55
ARC-t symmetric [117]	0.60
HFA [137]	0.75
DDC [54]	0.89
Ours	**0.92**

Table 3.2: Comparison to other domain adaptation techniques on the *UAV-200 (small)* dataset.

the domain shift [54]. It therefore makes sense to choose the configuration that yields the smallest MMD^2 value. In this case, it happens when using the exponential loss to connect the first three layers and sharing the weights of the others. Our intuition is that, even though the synthetic and real images feature the same objects, they differ in appearance, which is mostly encoded by the first network layers. Thus, allowing the weights to differ in these layers yields good adaptative behavior, as will be demonstrated in Section 3.3.1.3.

As a sanity check, we used validation data (500 positive and 1500 negative examples) to confirm that this MMD-based criterion reflects the best architecture choice. In Fig. 3.3 **(bottom)**, we plot the real detection accuracy as a function of the chosen configuration. The best possible accuracies are 0.916 and 0.757 on the validation and test data, respectively, whereas the ones corresponding to our MMD-based choice are 0.902 and 0.732, which corresponds to the second best architecture. Note that the MMD of the best solution also is very low. Altogether, we believe that this evidences that our MMD-based criterion provides an effective alternative to select the right architecture in the absence of validation data.

3.3.1.3 Evaluation

We first compare our approach to other Domain Adaptation methods on *UAV-200 (small)*. As can be seen in Table 3.2, it significantly outperforms many state-of-the-art baselines in terms of accuracy. In particular, we believe that outperforming DDC [54] goes a long way towards validating our hypothesis that modeling the domain shift is more effective than trying to be invariant to it. This is because, as discussed in Section 3.1, DDC relies on minimizing the MMD loss between the learned source and target representations much as we do, but uses a single stream for both source and target data. In other words, except for the non-shared weights, it is the method closest to ours. Note, however, that the original DDC paper used a slightly different network architecture than ours. To avoid any bias, we therefore modified this architecture so that it matches ours.

We then turn to the complete dataset *UAV-200 (full)*. In this case, the baselines whose implementations only output accuracy values become less relevant because it is not a good metric for unbalanced data. We therefore compare our approach against DDC [54], which we found to be our strongest competitor in the previous experiment, and against the Deep Learning approach

	AP (Average Precision)
*CNN (trained on Synthetic only (**S**))*	0.314
*CNN (trained on Real only (**R**))*	0.575
*CNN (pre-trained on **S** and fine-tuned on **R**):*	
Loss: L_t	0.612
Loss: $L_t + L_w$ *(with fixed source CNN)*	0.655
*CNN (pre-trained on **S** and fine-tuned on **R** and **S**:)*	
Loss: $L_s + L_t$ [49]	0.569
DDC [54] *(pre-trained on **S** and fine-tuned on **R** and **S**)*	0.664
*Our approach (pre-trained on **S** and fine-tuned on **R** and **S**)*	
Loss: $L_s + L_t + L_w$	0.673
Loss: $L_s + L_t + L_{MMD}$	0.711
Loss: $L_s + L_t + L_w + L_{MMD}$	**0.732**

Table 3.3: Comparison of our method against several baselines on the *UAV-200 (full)* dataset. As discussed in Section 3.2, the terms L_s, L_t, L_w, and L_{MMD} correspond to the elements of the loss function, defined in Eqs. 3.2, 3.3, 3.4, 3.5, respectively.

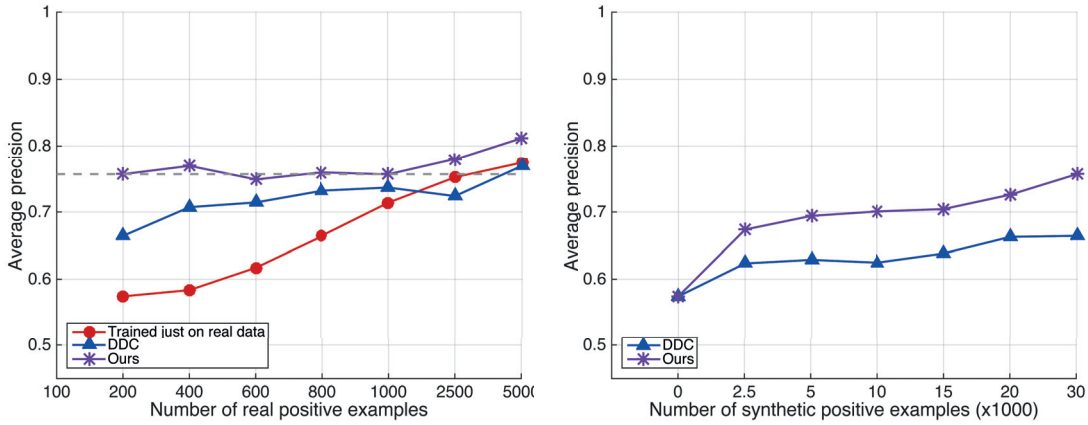


Figure 3.4: Influence of the ratio of synthetic to real data. [LEFT] AP of our approach (violet stars), DDC (blue triangles), and training using real data only (red circles) as a function of the number of real samples used given a constant number of synthetic ones. [RIGHT] AP of our approach (violet stars) and DDC (blue triangles) as a function of the number of synthetic examples used given a small and constant number of real one. (Best seen in color)

of [49], which also tackles the drone detection problem. We also turn on and off some of our loss terms to quantify their influence on the final performance. We give the results in Table 3.3. In short, all loss terms contribute to improving the AP of our approach, which itself outperforms all the baselines by large margins. More specifically, we get a 10% boost over DDC and a 20% boost over using real data only. By contrast, simply using real and synthetic examples together, as was done in [49], does not yield significant improvements. Note that dropping the terms linking

the weights in corresponding layers while still minimizing the MMD loss (*L*oss: $L_s + L_t + L_{MMD}$) performs worse than using our full loss function. We attribute this to overfitting of the target stream.

3.3.1.4 Influence of the Number of Samples

Using synthetic data in the UAV detection scenario is motivated by the fact that it is hard and time consuming to collect large amounts of real data. We therefore evaluate the influence of the ratio of synthetic to real data. To this end, we first fix the number of synthetic samples to 32800, as in *UAV-200 (full)*, and vary the amount of real positive samples from 200 to 5000. The results of this experiment are reported in Fig. 3.4(left), where we again compare our approach to DDC [54] and to the same CNN model trained on the real samples only. Our model always outperforms the one trained on real data only. This suggests that it remains capable of leveraging the synthetic data, even though more real data is available, which is not the case for DDC. More importantly, looking at the leftmost point on our curve shows that, with only 200 real samples, our approach performs similarly to, and even slightly better than, a single-stream model trained using 2500 real samples. In other words, one only needs to collect 5-10% of labeled training data to obtain good results with our approach, which, we believe, can have a significant impact in practical applications.

Fig. 3.4(right) depicts the results of an experiment where we fixed the number of real samples to 200 and increased the number of synthetic ones from 0 to 32800. Note that the AP of our approach steadily increases as more synthetic data is used. DDC also improves, but we systematically outperform it except when we use no synthetic samples, in which case both approaches reduce to a single-stream CNN trained on real data only.

3.3.2 Domain Adaptation on Office

To demonstrate that our approach extends to the unsupervised case, we further evaluate it on the *Office* dataset, which is a standard domain adaptation benchmark for image classification. Following standard practice, we express our results in terms of accuracy, as defined in Eq. 3.10. The *Office* dataset [116] comprises three different sets of images (Amazon, DSLR, Webcam) featuring 31 classes of objects. Fig. 3.5 depicts some images from the three different domains.

3.3.2.1 Unsupervised Domain Adaptation

For this experiment, we used the "fully-transductive" evaluation protocol proposed in [116], which means using all the available information on the source domain and having no labels at all for the target domain. In addition to the results obtained using our MMD regularizer of Eq. 3.5, and for a fair comparison with [56], which achieves state-of-the-art results on this dataset, we also report results obtained by replacing the MMD loss with one based on the domain confusion

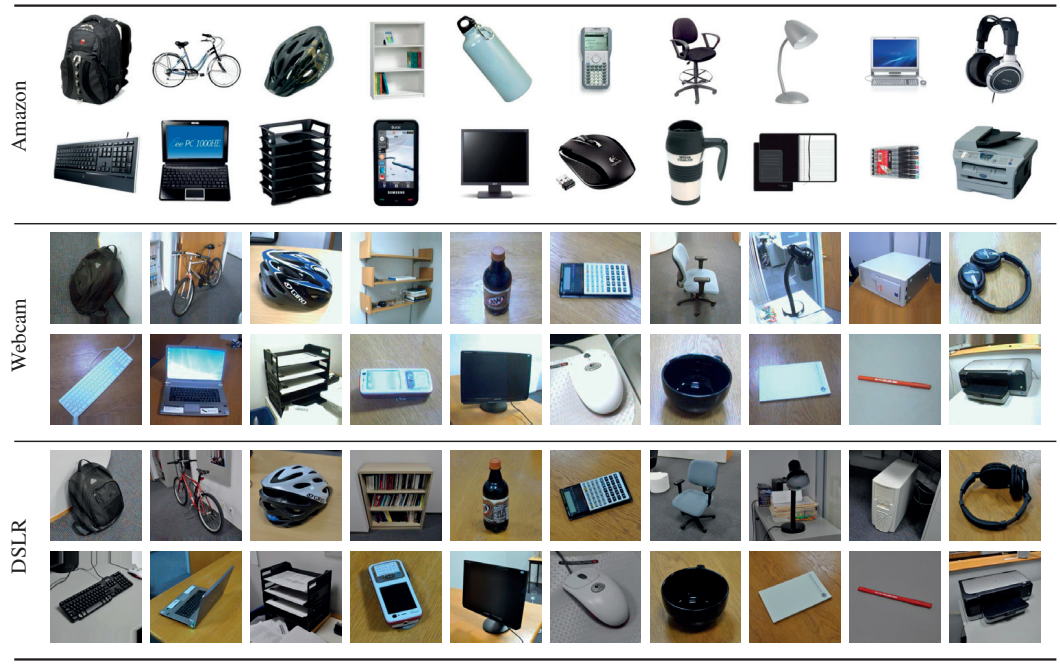


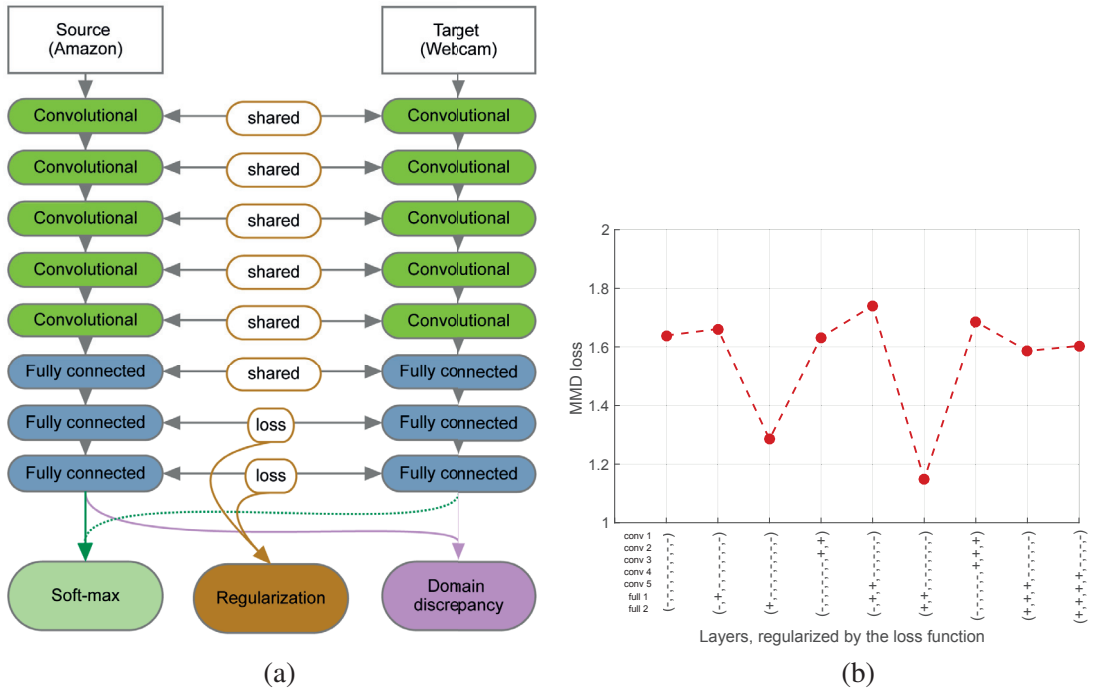
Figure 3.5: Some examples from three domains in the Office dataset.

	Accuracy			
	A → W	D → W	W → D	Average
GFK [119]	0.214	0.691	0.650	0.518
DLID [122]	0.519	0.782	0.899	0.733
DDC [54]	0.605	0.948	0.985	0.846
DAN [55]	0.645	0.952	0.986	0.861
DRCN [134]	0.687	0.964	0.990	0.880
GRL [56]	0.730	0.964	0.992	0.895
Ours	0.630	0.961	0.992	0.861
Ours (+ GRL)	**0.760**	**0.967**	**0.996**	**0.908**

Table 3.4: Comparison against other domain adaptation techniques on the Office benchmark. We evaluate on all 31 categories, following the "fully-transductive" evaluation protocol [116].

classifier advocated in [56]. We used the same architecture as in [56] for this classifier.

Fig. 3.6(a) illustrates the network architecture we used for this experiment. Each stream corresponds to the standard AlexNet CNN [30]. As in [54, 56], we start with the model pre-trained on ImageNet and fine tune it. However, instead of forcing the weights of both streams to be shared, we allow them to deviate as discussed in Section 3.2. To identify which layers should share their weights and which ones should *not*, we used the MMD-based criterion introduced in Section 3.3.1.2. In Fig. 3.6(b), we plot the MMD^2 value as a function of the configuration on the *Amazon* → *Webcam* scenario, as we did for the drones in Fig. 3.3. In this case, not sharing the



(a) (b)

Figure 3.6: Office dataset. **(a)** The network architecture that proved to be the best according to our MMD-based criterion. **(b)** The y-axis corresponds to the MMD^2 loss between the outputs of the corresponding streams that operate on *Amazon* and *Webcam*, respectively. The x-axis describes the configuration, as in Fig. 3.3.

last two fully-connected layers achieves the lowest MMD^2 value, and this is the configuration we use for our experiments on this dataset.

In Table 3.4, we compare our approach against other Domain Adaptation techniques on the three commonly-reported source/target pairs. It outperforms them on all the pairs. More importantly, the comparison against GRL [56] confirms that allowing the weights not to be shared increases accuracy.

3.3.2.2 Supervised Domain Adaptation

For this experiment, we used the evaluation protocol proposed in [116], which corresponds to the supervised scenario. This involves using a fraction of the available labeled samples in the target domain for training purposes along with all the labeled data from the source domain. As in [116], we used the labels of 20 randomly sampled images for each class for the Amazon domain and 8 labeled images per class for the DLSR and Webcam domains, when used as source datasets. For the target domain, we only used 3 randomly selected labeled images per class. The rest of the dataset was then used as unlabeled data for the calculation of the MMD loss of Eq. 3.5.

In Table 3.5, we compare our approach against other Domain Adaptation techniques on the three

	Accuracy			
	A → W	D → W	W → D	Average
GFK [119]	0.464	0.613	0.663	0.530
SA [120]	0.450	0.648	0.699	0.599
DA-NBNN [138]	0.528	0.766	0.762	0.685
DLID [122]	0.519	0.782	0.899	0.733
$DeCAF_6$ [132]	0.807	0.948	–	–
DaNN [130]	0.536	0.712	0.835	0.694
DDC [54]	0.841	0.954	0.963	0.919
Ours	**0.876**	**0.960**	**0.988**	**0.941**

Table 3.5: Comparison to other domain adaptation techniques for the supervised domain adaptation on the Office standard benchmark. We evaluate on all 31 categories, according to the supervised evaluation protocol described in [116].

commonly-reported source/target pairs. It outperforms them on all three. More importantly, the comparison against DDC confirms that allowing the weights not to be shared increases accuracy.

3.3.3 Unsupervised Domain Adaptation on MNIST-USPS

The *MNIST* [47] and *USPS* [139] datasets for digit classification both feature 10 different classes of images corresponding to the 10 digits. They have recently been employed for the task of Domain Adaptation [140].

For this experiment, we used the evaluation protocol of [140], which involves randomly selecting of 2000 images from *MNIST* and 1800 images from *USPS* and using them interchangeably as source and target domains. As in [140], we work in the unsupervised setting, and thus ignore the target domain labels at training time. Following [136], as the image patches in the *USPS* dataset are only 16×16 pixels, we rescaled the images from *MNIST* to the same size and applied L_2 normalization of the pixel intensities. For this experiment, we relied on the standard CNN architecture of [47] and employed our MMD-based criterion to determine which layers should not share their weights. We found that allowing all layers of the network not to share their weights yielded the best performance.

In Table 3.6, we compare our approach with DDC [54] and with methods that do not rely on deep networks [127, 119, 120, 140]. Our method yields superior performance in all cases, which we believe to be due to its ability to adapt the feature representation to each domain, while still keeping these representations close to each other.

method	Accuracy		
	M→U	U→M	Average
PCA	0.451	0.334	0.392
SA [120]	0.486	0.222	0.354
GFK [119]	0.346	0.226	0.286
TCA [127]	0.408	0.274	0.341
SSTCA [127]	0.406	0.222	0.314
TSL [141]	0.435	0.341	0.388
JCSL [140]	0.467	0.355	0.411
DDC [54]	0.478	0.631	0.554
Ours	**0.607**	**0.673**	**0.640**

Table 3.6: Comparison against other domain adaptation techniques on the MNIST+USPS standard benchmark.

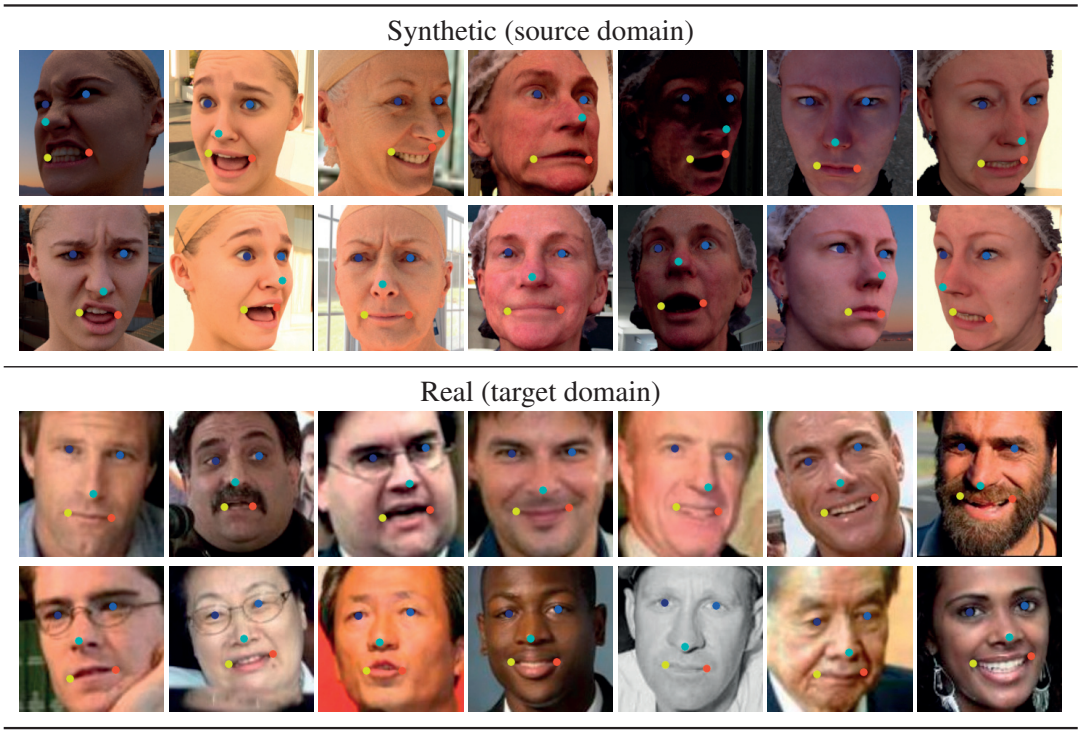


Figure 3.7: Samples images from *Source* and *Target* datasets with synthetic and real images respectively.

3.3.4 Supervised Facial Pose Estimation

To demonstrate that our method can be used not only for classification or detection tasks but also for regression ones, we further evaluate it for pose estimation purposes. More specifically, the task we address consists of predicting the location of 5 facial landmarks given 50×50 image

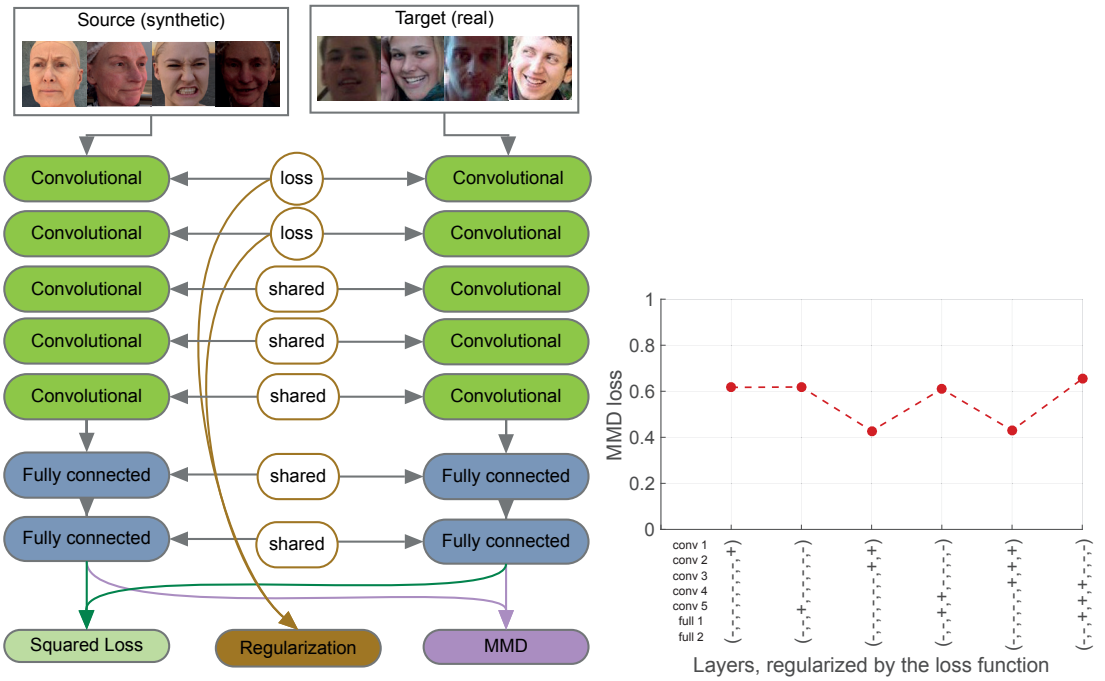


Figure 3.8: [LEFT] Network architecture for facial pose estimation. [RIGHT] Choosing the network architecture according to our MMD^2 criterion. '-' and '+' denote that the weights of the corresponding layers are shared and stream-specific, respectively.

patches, such as those of Fig. 3.7. To this end, we train a regressor to predict a 10D vector with two floating point coordinates for each landmark. As we did for drones, we use *synthetic* images, such as the ones shown in the top portion of Fig. 3.7, as our source domain and *real* ones, such as those shown at the bottom, as our target domain. Both datasets contain $\sim 10k$ annotated images. We use all the synthetic samples but only 100 of the real ones for training, and the remainder for testing.

Our architecture is depicted by Fig. 3.8. The same figure further shows the MMD^2 values corresponding to different configurations of layers with shared/non-shared weights. Here, the x-axis illustrates the network configuration, where '-' and '+' denote that the weights of the corresponding layers are shared and stream-specific, respectively. As we can see the first two-layers should be allowed to have different weights, which, as in the UAV dataset, reflects the fact that the synthetic images differ from the real ones mostly by low-level appearance variations.

In Table 3.7, we compare our Domain Adaptation results to those of DDC [54] in terms of percentage of correctly estimated landmarks (PCP-score). Each landmark is considered to be correctly estimated if it is found within a 2 pixel radius from the ground-truth. Note that, again, by not sharing the weights, our approach outperforms DDC.

	Synthetic	DDC [54]	Ours
Right eye	64.2	68.0	**71.8**
Left eye	39.3	56.2	**60.3**
Nose	56.3	64.1	**64.5**
Right mouth corner	47.8	57.6	**59.8**
Left mouth corner	42.3	55.5	**57.7**
Average	50.0	60.3	**62.8**

Table 3.7: Regression results on facial pose estimation.

3.3.5 Discussion

In all the experiments reported above, allowing the weights *not* to be shared in some fraction of the layers of our two-stream architecture boosts performance. This validates our initial hypothesis that explicitly modeling the domain shift is generally beneficial.

However, the optimal choice of which layers should or should not share their weights is application dependent. In both the UAV detection and facial pose estimation cases, allowing the weights in the first two layers to be different yields top performance, which we understand to mean that the domain shift is caused by low-level changes that are best handled in the early layers. By contrast, for the *Office* dataset, it is best to only allow the weights in the last two layers to differ. This network configuration was determined using *Amazon* and *Webcam* images, such as those shown in Fig. 3.5. Close examination of these images reveals that the differences between them are not simply due to low-level phenomena, such as illumination changes, but to more complex variations. It therefore seems reasonable that the higher layers of the network, which encode higher-level information, should be domain-specific.

Fortunately, we have shown that the MMD provides us with an effective criterion to choose the right configuration. This makes our two-steam approach practical, even when no validation data from the target domain is available.

3.4 Conclusion

In this chapter we have shown that Deep Learning approaches to Domain Adaptation should not focus on learning features that are invariant to the domain shift, which makes them less discriminative. Instead, we should explicitly model the domain shift. To prove this, we have introduced a two-stream CNN architecture, where the weights of the streams may or may not be shared. To nonetheless encode the fact that both streams should be related, we encourage the non-shared weights to remain close to being linear transformations of each other by introducing an additional loss term.

Our experiments on very diverse datasets have clearly validated our hypothesis. Our approach

consistently yields higher accuracy than networks that share all weights for the source and target data, both for classification and regression.

4 Concluding Remarks

We began this thesis by identifying the need for cheap and effective vision-based collision avoidance algorithms for Unmanned Aerial Vehicles. With the growing consumer and industrial interest in this area other means of relative positioning, such as those based on GPS paired with radio communication, are not enough to ensure flight safety in both indoor and outdoor environments. Vision-based detection and collision avoidance of UAVs, however, poses several unique challenges that are specific to our chosen application domain. These led us to develop three approaches, where the first one is the learning based technique that allows detecting small fast-moving flying objects from a single moving camera and the other two aim at decreasing the need for real data collection and its manual annotation.

In Chapter 1 we introduced a hybrid approach that combines appearance and motion information for detection of small fast moving objects in complex outdoor environments. The proposed method relies on spatio-temporal cubes (st-cubes) of image intensities extracted from a sequence of consecutive video frames, which allows the detector to efficiently combine appearance and motion cues that both play vital role in detection process. However, this led to a significant increase in the data variability, as the features computed from these st-cubes are very different for the same aircraft flying in different directions. Therefore, we introduced an object-centric regression-based motion compensation method that considerably reduces this variation, which ultimately led to a significant increase in detection accuracy.

We then showed in Chapter 2 that we can decreases the need in collecting real footage and its manual annotation by augmenting a dataset of real images with synthetic ones. This resulted in a more robust detector capable of identifying different kinds of aircrafts in various environments and weather conditions. The key component of the proposed synthetic data generation approach is the feature-based similarity measure that allows generating images that do not have to be visually pleasing, but are close to the real images in the feature space of the chosen detector. We proved the effectiveness of our approach on three different datasets, featuring UAVs, fixed-wing aircrafts and cars.

Motivated by the success of Deep Learning in various classification and detection tasks [30, 43,

50, 44] we applied the aforementioned algorithm for synthetic data generation to train a Deep Neural Network. In Chapter 3 we showed that simply combining real and synthetic images is not enough and leads to overfitting to the large amount of synthetic examples. To this end, we proposed a novel domain adaptation technique that enables us to learn useful representations on synthetic data and apply those to real images. Our technique is based on a two-stream architecture that efficiently combines large amount of synthetic images and a small number of real ones during training. Our two-stream architecture models the domain shift which is the difference between real and synthetic images and allows improving over the state-of-the-art domain adaptation techniques.

To summarize, methods presented in this work will make it possible to increase fight safety and awareness of the UAVs about surrounding flying objects both indoors and outdoors using a single camera. They will also help reduce the annotation requirements for learning-based detection and regression algorithms. The methods we have proposed are generic and widely applicable. Therefore, they have the potential to help not just in the area of flying vehicles, but also in a broad variety of other fields where manual data annotation is either time consuming or prohibitively difficult.

Limitations and Future Work

In this section we discuss the limitations of our existing approaches and ways in which they could be overcome.

1 Joint training

In Chapter 1 we showed that combining motion and appearance information is essential for precise detection and localization of small fast moving objects in complex outdoor environments. We evaluated different approaches for detection and showed that the one based on Convolutional Neural Networks achieves accuracy which is on par with the method based on boosted trees [72], which itself demonstrates state-of-the-art. We further showed that CNN-based object-centric motion compensation significantly improves detection accuracy by decreasing the in-class variation of the data. In our current implementation, motion compensation and detection algorithms are trained separately. We believe a further increase in performance can be achieved by merging both of these approaches in a unified Deep Learning architecture, which enables us to jointly optimize the parameters on both tasks.

2 Tracking

In Chapter 1 we focused on improving the quality of individual detections of flying objects. A different way of increasing the overall accuracy and improving the speed of the whole UAV relative positioning framework is linking these detections over time with the help of tracking

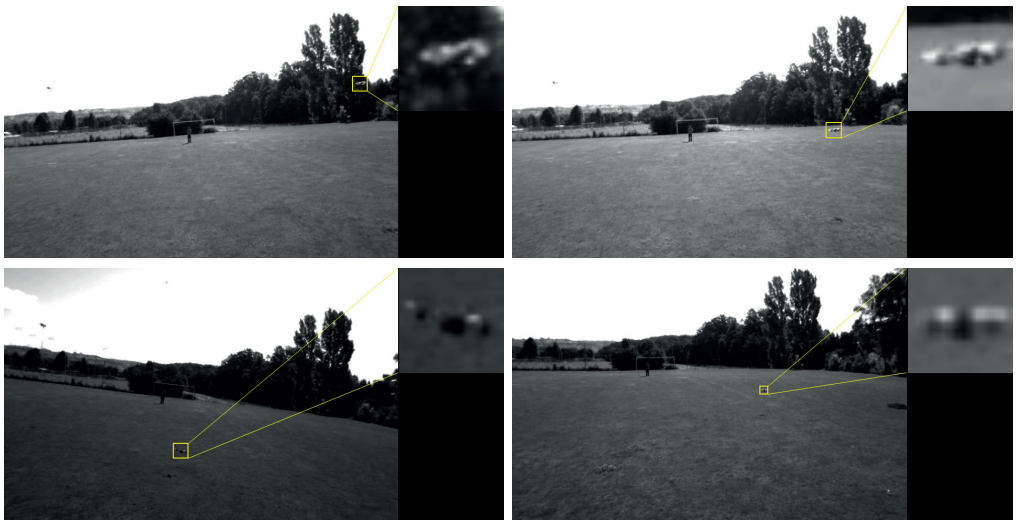


Figure 4.1: Sample images showing sharp light reflection from the UAV rotors. The thumbnails on the right of each image illustrate the zoomed in patch of the image with the drone and show the importance of modeling the specular light reflection.

approaches.

Single-view tracking has been studied well in the context of estimating trajectories of pedestrians [142, 143], vehicles [144, 145] and other objects [146, 147]. However, in the context of UAVs, this area has been relatively unexplored due to the complexity of possible motions and variety of drone appearances. Our detection approach can be particularly beneficial in this case, due to the fact that it is based on st-cubes of image intensities, cropped out of consecutive video frames. Processing each of these st-cubes with motion compensation system and the detector gives us not just the location of the drone, but also its movement direction and its short-term trajectory (tracklet). These tracklets and drone's movement orientation can then be efficiently used by tracking algorithms to increase robustness to false detections and avoid possible identity switches, when multiple drones are flying around. Therefore combining existing tracking approaches with our detector is a promising direction for future research.

3 Synthetic data generation improvement

In Chapter 2 we described our approach to generating synthetic images from a very coarse model of the object and a set of pairs of images with and without a drone. The proposed algorithm can be extended by increasing the range of effects that it is capable of modeling. One example of such an effect is the specular component of the Phong reflection model [148], modeling which could be rather important, as it may have a large influence on the appearance of the rendered UAV, e.g. in situations, when the light is reflected from the rotors of the drone. Fig. 4.1 depicts several frames from the video sequence, where specular light reflection has a high impact on the drone appearance.

Figure 4.2: Sample complex 3D UAV models.

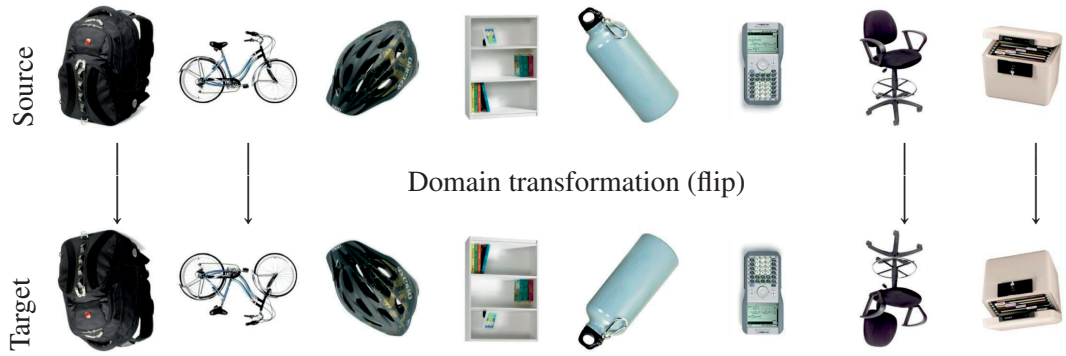


Figure 4.3: Sample pair of domains, related with a flipping transformation

Further, in our experiments we used a somewhat simplistic 3D model that roughly outlines the geometric structure of the drone. Therefore, another way of improving the quality of synthetic images is to increase the expressiveness of the underlying UAV model, for example by having different material properties for different parts of the model (body, rotors, etc.) or adding texture to it, as illustrated by Fig. 4.2. We believe that these particular improvements and, generally, application of more sophisticated Computer Graphics techniques for data generation will make synthetic images even closer to the real ones from detector's point of view, resulting in a further increase in performance.

4 Domain Adaptation

4.1 Modeling of complex domain transformations

In Chapter 3 we argue that our two-stream architecture for modeling the difference between domains is beneficial for the overall classification accuracy. We further show that weights of the different streams should not be independent from each other. More specifically the top performance is reached when the parameters of one network are linear transformations of the parameters of the other one. As depicted by Eq. 3.7 in Section 3.2.1 this transformations solely depend on two parameters for each pair of layers that are not shared.

While effective, this method is not capable of modeling such geometric domain transformations as image mirroring, flipping, rotation, etc. (sample domain transformation is depicted by Fig. 4.3).

Therefore, one of our future research directions will be extending our approach to model both appearance and geometric domain transformations by increasing the expressibility of the function that models the relationship between the corresponding parameters of two streams. One possible way to pursue this would be to introduce an additional neural network that transforms parameters of the stream that operates on the source data to parameters of the stream that operates only on the target data. This will allow the modeling of complex domain shifts including the ones, depicted in Fig. 4.3.

4.2 Automatic architecture selection

Our domain adaptation technique requires the set of layers that do not share their weights to be defined before the beginning of the training. While we have seen in Chapter 3 that MMD-criterion can be used to select the right configuration, allowing the network to make this decision automatically is an attractive alternative, as in this case we will not need to train multiple versions of the network to find the optimal configuration. This will facilitate the use of deeper architectures, such as VGG [43] and GoogleNet [50], as evaluating different possible configurations for these networks is very time consuming. Therefore, automatic architecture selection for our two-stream domain adaptation technique is a possible direction for future research.

Bibliography

[1] "Dji matrice 100." [Online]. Available: http://www.dji.com/product/matrice100

[2] "AscTec FireFly." [Online]. Available: http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-firefly/

[3] "Microdrones." [Online]. Available: https://www.microdrones.com/en/applications/areas-of-application/monitoring/

[4] "Airborne Drones." [Online]. Available: http://www.airbornedrones.co/pages/security-drones

[5] Pix4D, "Pix4D Mapper," 2012. [Online]. Available: https://www.pix4d.com/

[6] "Amazon Prime Air." [Online]. Available: https://www.amazon.com/b?node=8037720011

[7] A. Momont, "Ambulance Drone." [Online]. Available: http://www.io.tudelft.nl/onderzoek/delft-design-labs/applied-labs/ambulance-drone/

[8] G. Conte and P. Doherty, "An Integrated UAV Navigation System Based on Aerial Image Matching," in *IEEE Aerospace Conference*, 2008, pp. 3142–3151.

[9] C. Martínez, I. F. Mondragón, M. Olivares-Méndez, and P. Campoy, "On-Board and Ground Visual Pose Estimation Techniques for UAV Control," *Journal of Intelligent and Robotic Systems*, vol. 61, no. 1-4, pp. 301–320, 2011.

[10] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "PIXHAWK: A System for Autonomous Flight Using Onboard Computer Vision," in *International Conference on Robotics and Automation*, 2011, pp. 2992–2997.

[11] C. Hane, C. Zach, J. Lim, A. Ranganathan, and M. Pollefeys, "Stereo Depth Map Fusion for Robot Navigation," in *Proceedings of International Conference on Intelligent Robots and Systems*, 2011, pp. 1618–1625.

[12] S. Weiss, M. Achtelik, S. Lynen, M. Achtelik, L. Kneip, M. Chli, and R. Siegwart, "Monocular Vision for Long-Term Micro Aerial Vehicle State Estimation: A Compendium," *Journal of Field Robotics*, vol. 30, pp. 803–831, 2013.

[13] S. Lynen, M. Achtelik, S. Weiss, M. Chli, and R. Siegwart, "A Robust and Modular Multi-Sensor Fusion Approach Applied to MAV Navigation," in *International Conference on Intelligent Robots and Systems*, 2013, pp. 3923–3929.

[14] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast Semi-Direct Monocular Visual Odometry," in *International Conference on Robotics and Automation*, 2014, pp. 15–22.

[15] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, "Autonomous, Vision-based Flight and Live Dense 3D Mapping with a Quadrotor Micro Aerial Vehicle," *Journal of Field Robotics*, vol. 33, no. 4, pp. 431–450, 2016.

[16] T. Zsedrovits, A. Zarándy, B. Vanek, T. Peni, J. Bokor, and T. Roska, "Visual Detection and Implementation Aspects of a UAV See and Avoid System," in *European Conference on Circuit Theory and Design*, 2011, pp. 472–475.

[17] J. Li, D. H. Ye, T. Chung, M. Kolsch, J. Wachs, and C. Bouman, "Multi-target detection and tracking from a single camera in unmanned aerial vehicles (uavs)," in *International Conference on Intelligent Robots and Systems*, 2016, pp. 4992–4997.

[18] P. Dollár, Z. Tu, P. Perona, and S. Belongie, "Integral Channel Features," in *British Machine Vision Conference*, 2009, pp. 1–11.

[19] S. Walk, N. Majer, K. Schindler, and B. Schiele, "New Features and Insights for Pedestrian Detection," in *Conference on Computer Vision and Pattern Recognition*, 2010, pp. 1030–1037.

[20] S. Zhang, R. Benenson, M. Omran, J. Hosang, and B. Schiele, "How Far are We from Solving Pedestrian Detection?" in *Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1259–1267.

[21] X. Chen, K. Kundu, Y. Zhu, A. G. Berneshawi, H. Ma, S. Fidler, and R. Urtasun, "3D Object Proposals for Accurate Object Class Detection," in *Advances in Neural Information Processing Systems*, 2015, pp. 424–432.

[22] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3d object detection for autonomous driving," in *Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2147–2156.

[23] Y. Xiang, W. Choi, Y. Lin, and S. Savarese, "Subcategory-aware Convolutional Neural Networks for Object Proposals and Detection," in *arXiv Preprint*, 2016, vol. abs/1604.04693.

[24] "Mercedes-Benz Intelligent Drive." [Online]. Available: https://www.mercedes-benz.com/en/mercedes-benz/innovation/mercedes-benz-intelligent-drive/

[25] "Mobileeye Inc." [Online]. Available: http://us.mobileye.com/technology/

[26] "Tesla autopilot." [Online]. Available: https://www.tesla.com/autopilot

[27] "Mercedes Autonomous Driving." [Online]. Available: https://www.mercedes-benz.com/en/mercedes-benz/innovation/autonomous-driving/

[28] A. Bosch, A. Zisserman, and X. Munoz, "Image Classification Using Random Forests and Ferns," in *International Conference on Computer Vision*, 2007, pp. 1–8.

[29] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan, "Object Detection with Discriminatively Trained Part Based Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[30] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1106–1114.

[31] A. Sobral, "BGSLibrary: An OpenCV C++ Background Subtraction Library," in *IX Workshop de Visao Computacional*, 2013.

[32] D. Zamalieva and A. Yilmaz, "Background Subtraction for the Moving Camera: A Geometric Approach," *Computer Vision and Image Understanding*, vol. 127, pp. 73–85, 2014.

[33] T. Brox and J. Malik, "Object Segmentation by Long Term Analysis of Point Trajectories," in *European Conference on Computer Vision*, 2010, pp. 282–295.

[34] T. Brox and J. Malik, "Large Displacement Optical Flow: Descriptor Matching in Variational Motion Estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 3, pp. 500–513, 2011.

[35] D. Park, C. L. Zitnick, D. Ramanan, and P. Dollár, "Exploring Weak Stabilization for Motion Feature Extraction," in *Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2882–2889.

[36] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," Master's thesis, Department of Computer Science, University of Toronto, 2009.

[37] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results."

[38] O. Russakovsky, J. Deng, H. Su, J. Krause, S.Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg, and L. Fei-Fei, "Imagenet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[39] J. Marin, D. Vázquez, D. Geronimo, and A. M. Lopez, "Learning Appearance in Virtual Scenarios for Pedestrian Detection," in *Conference on Computer Vision and Pattern Recognition*, 2010, pp. 137–144.

[40] L. Pishchulin, A. Jain, A. Mykhaylo, T. Thormaehlen, and B. Schiele, "Articulated People Detection and Pose Estimation: Reshaping the Future," in *Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3178–3185.

[41] K. Rematas, T. Ritschel, M. Fritz, and T. Tuytelaars, "Image-Based Synthesis and Re-Synthesis of Viewpoints Guided by 3D Models," in *Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3898–3905.

[42] J. Papon and M. Schoeler, "Semantic Pose Using Deep Networks Trained on Synthetic RGB-D," in *International Conference on Computer Vision*, 2015, pp. 774–782.

[43] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference for Learning Representations*, 2015.

[44] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3D Bounding Box Estimation Using Deep Learning and Geometry," *arXiv Preprint*, vol. abs/1612.00496, 2016.

[45] A. Rozantsev, V. Lepetit, and P. Fua, "Flying Objects Detection from a Single Moving Camera," in *Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4128–4136.

[46] A. Rozantsev, V. Lepetit, and P. Fua, "Detecting Flying Objects Using a Single Moving Camera," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016.

[47] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," in *Proceedings of the IEEE*, 1998, pp. 2278–2324.

[48] T. Varga and H. Bunke, "Generation of Synthetic Training Data for an HMM-Based Handwriting Recognition System," in *International Conference on Document Analysis and Recognition*, 2003, pp. 618–622.

[49] A. Rozantsev, V. Lepetit, and P. Fua, "On Rendering Synthetic Images for Training an Object Detector," *Computer Vision and Image Understanding*, vol. 137, pp. 24–37, 2015.

[50] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Conference on Computer Vision and Pattern Recognition*, June 2015, pp. 1–9.

[51] A. Rozantsev, M. Salzmann, and P. Fua, "Beyond Sharing Weights for Deep Domain Adaptation," *arXiv Preprint*, 2016.

[52] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," *arXiv Preprint*, 2013.

[53] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and Transferring Mid-Level Image Representations Using Convolutional Neural Networks," in *Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1717–1724.

[54] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell, "Deep Domain Confusion: Maximizing for Domain Invariance," *arXiv Preprint*, 2014.

[55] M. Long, Y. Cao, J. Wang, and M. I. Jordan, "Learning Transferable Features with Deep Adaptation Networks," in *International Conference on Machine Learning*, 2015, pp. 97–105.

[56] Y. Ganin and V. Lempitsky, "Unsupervised Domain Adaptation by Backpropagation," in *International Conference on Machine Learning*, 2015, pp. 1180–1189.

[57] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko, "Simultaneous Deep Transfer Across Domains and Tasks," in *International Conference on Computer Vision*, 2015, pp. 4068–4076.

[58] P. Dollar, V. Rabaud, G. Cottrell, and S. Belongie, "Behavior Recognition via Sparse Spatio-Temporal Features," in *VS-PETS*, October 2005, pp. 65–72.

[59] I. Laptev, "On Space-Time Interest Points," *International Journal of Computer Vision*, vol. 64, no. 2-3, pp. 107–123, 2005.

[60] D. Weinland, M. Ozuysal, and P. Fua, "Making Action Recognition Robust to Occlusions and Viewpoint Changes," in *European Conference on Computer Vision*, 2010, pp. 635–648.

[61] B. Tekin, A. Rozantsev, V. Lepetit, and P. Fua, "Direct Prediction of 3D Body Poses from Motion Compensated Sequences," in *Conference on Computer Vision and Pattern Recognition*, 2016, pp. 991–1000.

[62] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio, "Theano: new features and speed improvements," 2012.

[63] N. Oliver, B. Rosario, and A. Pentland, "A Bayesian Computer Vision System for Modeling Human Interactions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 831–843, 2000.

[64] N. Seungjong and J. Moongu, "A New Framework for Background Subtraction Using Multiple Cues," in *Asian Conference on Computer Vision*. Springer Berlin Heidelberg, 2013, pp. 493–506.

[65] B. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *IJCAI*, 1981, pp. 674–679.

[66] Y. Zhang, S.-J. Kiselewich, W.-A. Bauson, and R. Hammoud, "Robust Moving Object Detection at Distance in the Visible Spectrum and Beyond Using a Moving Camera," in *Conference on Computer Vision and Pattern Recognition Workshops*, 2006, p. 131.

[67] S.-W. Kim, K. Yun, K.-M. Yi, S.-J. Kim, and J.-Y. Choi, "Detection of Moving Objects with a Moving Camera Using Non-Panoramic Background Model," *Machine Vision and Applications*, vol. 24, pp. 1015–1028, 2013.

[68] S. Kwak, T. Lim, W. Nam, B. Han, and J. Han, "Generalized Background Subtraction Based on Hybrid Inference by Belief Propagation and Bayesian Filtering," in *International Conference on Computer Vision*, 2011, pp. 2174–2181.

[69] A. Elqursh and A. Elgammal, "Online Moving Camera Background Subtraction," in *European Conference on Computer Vision*, 2012, pp. 228–241.

[70] M. Narayana, A. Hanson, and E. Learned-miller, "Coherent Motion Segmentation in Moving Camera Videos Using Optical Flow Orientations," in *International Conference on Computer Vision*, 2013, pp. 1577–1584.

[71] A. Papazoglou and V. Ferrari, "Fast Object Segmentation in Unconstrained Video," in *International Conference on Computer Vision*, 2013, pp. 1777–1784.

[72] J. Friedman, "Stochastic Gradient Boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[73] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *Conference on Computer Vision and Pattern Recognition*, 2005, pp. 886–893.

[74] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[75] M. D. Zeiler, "ADADELTA: an Adaptive Learning Rate Method," *arXiv Preprint*, 2012.

[76] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[77] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2009.

[78] R. Sznitman, C. Becker, F. Fleuret, and P. Fua, "Fast Object Detection with Entropy-Driven Evaluation," in *Conference on Computer Vision and Pattern Recognition*, 2013, pp. 3270–3277.

[79] A. Vedaldi and B. Fulkerson, "VLFeat: An open and portable library of computer vision algorithms," http://www.vlfeat.org/, 2008.

[80] R. Benenson, O. Mohamed, J. Hosang, and B. Schiele, "Ten Years of Pedestrian Detection, What Have We Learned?" in *European Conference on Computer Vision Workshops*, 2014, pp. 613–627.

[81] L. Breiman, "Random Forests," *Machine Learning*, 2001.

[82] P. Dollár, "Piotr's Computer Vision Matlab Toolbox (PMT)," http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html.

[83] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the Best Multi-Stage Architecture for Object Recognition?" in *International Conference on Computer Vision*, 2009, pp. 2146–2153.

[84] J. Jin, K. Fu, and C. Zhang, "Traffic Sign Recognition with Hinge Loss Trained Convolutional Neural Networks," *Transactions on Intelligent Transportation Systems*, vol. 15, pp. 1991–2000, 2014.

[85] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial Transformer Networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2017–2025.

[86] M. Oberweger, P. Wohlhart, and V. Lepetit, "Hands Deep in Deep Learning for Hand Pose Estimation," *arXiv Preprint*, vol. abs/1502.06807, 2015.

[87] C. Burges and B. Schölkopf, "Improving the Accuracy and Speed of Support Vector Machines," in *Advances in Neural Information Processing Systems*, 1997, pp. 375–381.

[88] D. Decoste and B. Schölkopf, "Training Invariant Support Vector Machines," *Machine Learning*, vol. 46, pp. 161–190, 2002.

[89] F. Fleuret and D. Geman, "Coarse-To-Fine Visual Selection," *International Journal of Computer Vision*, vol. 41, no. 1, pp. 85–107, January 2001.

[90] V. Lepetit, P. Lagger, and P. Fua, "Randomized Trees for Real-Time Keypoint Recognition," in *Conference on Computer Vision and Pattern Recognition*, June 2005, pp. 775–781.

[91] J. Shotton, R. Girshick, A. Fitzgibbon, T. Sharp, M. Cook, M. Finocchio, R. Moore, P. Kohli, A. Criminisi, A. Kipman, and A. Blake, "Efficient Human Pose Estimation from Single Depth Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 12, pp. 2821–2840, 2013.

[92] P. Felzenszwalb, D. Mcallester, and D. Ramanan, "A Discriminatively Trained, Multiscale, Deformable Part Model," in *Conference on Computer Vision and Pattern Recognition*, June 2008.

[93] Y. Freund and R. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," in *European Conference on Computational Learning Theory*, 1995, pp. 23–37.

[94] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust Object Recognition with Cortex-Like Mechanisms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 3, pp. 411–426, 2007.

[95] V. Lepetit and P. Fua, "Keypoint Recognition Using Randomized Trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 9, pp. 1465–1479, September 2006.

[96] D. Cireşan, A. Giusti, L. Gambardella, and J. Schmidhuber, "Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images," in *Advances in Neural Information Processing Systems*, 2012, pp. 2852–2860.

[97] A. Handa, R. Newcombe, A. Angeli, and A. Davison, "Real-Time Camera Tracking: When is High Frame-Rate Best?" in *European Conference on Computer Vision*, 2012, pp. 222–235.

[98] B. Horn and B. Schunck, "Determining Optical Flow," *Artificial Intelligence*, vol. 17, pp. 185–204, 1981.

[99] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of Optical Flow Techniques," *International Journal of Computer Vision*, vol. 12, pp. 43–77, 1994.

[100] M. Stark, M. Goesele, and B. Schiele, "Back to the Future: Learning Shape Models from 3D CAD Data," in *British Machine Vision Conference*, 2010, pp. 1061–10 611.

[101] J. Liebelt and C. Schmid, "Multi-View Object Class Detection with a 3D Geometric Model," in *Conference on Computer Vision and Pattern Recognition*, 2010, pp. 1688–1695.

[102] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. Black, and R. Szeliski, "A Database and Evaluation Methodology for Optical Flow," *International Journal of Computer Vision*, vol. 92, pp. 1–31, 2011.

[103] B. Kaneva, A. Torralba, and W. Freeman, "Evaluation of Image Features Using a Photorealistic Virtual World," in *International Conference on Computer Vision*, 2011, pp. 2282–2289.

[104] V. Athitsos and S. Sclaroff, "Estimating 3D Hand Pose from a Cluttered Image," in *Conference on Computer Vision and Pattern Recognition*, June 2003, pp. 432–439.

[105] L. Taycher, G. Shakhnarovich, D. Demirdjian, and T. Darrell, "Conditional Random People: Tracking Humans with CRFs and Grid Filters," in *Conference on Computer Vision and Pattern Recognition*, 2006, pp. 222–229.

[106] G. Klein and D. W. Murray, "Simulating Low-Cost Cameras for Augmented Reality Compositing," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 3, pp. 369–380, 2010.

[107] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[108] K. Levi and Y. Weiss, "Learning Object Detection from a Small Number of Examples: the Importance of Good Features," in *Conference on Computer Vision and Pattern Recognition*, 2004, pp. 53–60.

[109] T. Ruzic and A. Pizurica, "Texture and Color Descriptors as a Tool for Context-Aware Patch-Based Image Inpainting," in *SPIE Electronic Imaging*, vol. 8295, 2012.

[110] J. Jiang, "A Literature Survey on Domain Adaptation of Statistical Classifiers," University of Illinois at Urbana-Champaign, Tech. Rep., 2008.

[111] S. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, 2010.

[112] L. Duan, I. Tsang, D.Xu, and S. Maybank, "Domain Transfer SVM for Video Concept Detection," in *Conference on Computer Vision and Pattern Recognition*, 2009, pp. 1375–1381.

[113] A. Bergamo and L. Torresani, "Exploiting Weakly-Labeled Web Images to Improve Object Classification: A Domain Adaptation Approach," in *Advances in Neural Information Processing Systems*, 2010, pp. 181–189.

[114] C. Becker, M. Christoudias, and P. Fua, "Non-Linear Domain Adaptation with Boosting," in *Advances in Neural Information Processing Systems*, 2013, pp. 485–493.

[115] H. Daumé and D. Marcu, "Domain Adaptation for Statistical Classifiers," *Journal of Artificial Intelligence Research*, vol. 26, no. 1, pp. 101–126, 2006.

[116] K. Saenko, B. Kulis, M. Fritz, and T. Darrell, "Adapting Visual Category Models to New Domains," in *European Conference on Computer Vision*, 2010, pp. 213–226.

[117] B. Kulis, K. Saenko, and T. Darrell, "What You Saw is Not What You Get: Domain Adaptation Using Asymmetric Kernel Transforms," in *Conference on Computer Vision and Pattern Recognition*, 2011, pp. 1785–1792.

[118] R. Gopalan, R. Li, and R. Chellappa, "Domain Adaptation for Object Recognition: An Unsupervised Approach," in *International Conference on Computer Vision*, 2011, pp. 999–1006.

[119] B. Gong, Y. Shi, F. Sha, and K. Grauman, "Geodesic Flow Kernel for Unsupervised Domain Adaptation," in *Conference on Computer Vision and Pattern Recognition*, 2012, pp. 2066–2073.

[120] B. Fernando, A. Habrard, M. Sebban, and T. Tuytelaars, "Unsupervised Visual Domain Adaptation Using Subspace Alignment," in *International Conference on Computer Vision*, 2013, pp. 2960–2967.

[121] R. Caseiro, J. Henriques, P. Martins, and J. Batista, "Beyond the Shortest Path : Unsupervised Domain Adaptation by Sampling Subspaces Along the Spline Flow," in *Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3846–3854.

[122] S. Chopra, S. Balakrishnan, and R. Gopalan, "DLID: Deep Learning for Domain Adaptation by Interpolating Between Domains," in *International Conference on Machine Learning*, 2013.

[123] A. Gretton, K. Borgwardt, M. Rasch, B. Schölkopf, and A. Smola, "A Kernel Method for the Two-Sample Problem," *arXiv Preprint*, 2008.

[124] J. Huang., A. Smola, A. Gretton., K. Borgwardt, and B. Scholkopf, "Correcting Sample Selection Bias by Unlabeled Data," in *Advances in Neural Information Processing Systems*, 2006, pp. 601–608.

[125] A. Gretton, A. Smola, J. Huang, M. Schmittfull, K. Borgwardt, and B. Schölkopf, "Covariate Shift by Kernel Mean Matching," *Journal of the Royal Statistical Society*, vol. 3, no. 4, pp. 5–13, 2009.

[126] B. Gong, K. Grauman, and F. Sha, "Connecting the Dots with Landmarks: Discriminatively Learning Domain-Invariant Features for Unsupervised Domain Adaptation," in *International Conference on Machine Learning*, 2013, pp. 222–230.

[127] S. Pan, I. Tsang, J. Kwok, and Q. Yang, "Domain Adaptation via Transfer Component Analysis," in *International Joint Conference on Artificial Intelligence*, 2009, pp. 1187–1192.

[128] K. Muandet, D. Balduzzi, and B. Schölkopf, "Domain Generalization via Invariant Feature Representation," in *International Conference on Machine Learning*, 2013, pp. 10–18.

[129] M. Baktashmotlagh, M. Harandi, B. Lovell, and M. Salzmann, "Unsupervised Domain Adaptation by Domain Invariant Projection," in *International Conference on Computer Vision*, 2013, pp. 769–776.

[130] M. Ghifary, W. B. Kleijn, and M. Zhang, "Domain Adaptive Neural Networks for Object Recognition," in *Pacific Rim International Conference on Artificial Intelligence*, 2014, pp. 898–904.

[131] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "SURF: Speeded Up Robust Features," *Computer Vision and Image Understanding*, vol. 10, no. 3, pp. 346–359, 2008.

[132] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition," in *International Conference on Machine Learning*, 2014, pp. 647–655.

[133] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a Similarity Metric Discriminatively, with Application to Face Verification," in *Conference on Computer Vision and Pattern Recognition*, 2005, pp. 539–546.

[134] M. Ghifary, W. B. Kleijn, M. Zhang, D. Balduzzi, and W. Li, "Deep Reconstruction-Classification Networks for Unsupervised Domain Adaptation," *European Conference on Computer Vision*, pp. 597–613, 2016.

[135] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan, "Domain Separation Networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 343–351.

[136] M. Long, J. Wang, G. Ding, J. Sun, and P. Yu, "Transfer Feature Learning with Joint Distribution Adaptation," in *International Conference on Computer Vision*, 2013, pp. 2200–2207.

[137] W. Li, L. Duan, D. Xu, and I. W. Tsang, "Learning with Augmented Features for Supervised and Semi-Supervised Heterogeneous Domain Adaptation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1134–1148, 2014.

[138] T. Tommasi and B. Caputo, "Frustratingly Easy NBNN Domain Adaptation," in *International Conference on Computer Vision*, 2013, pp. 897–904.

[139] J. Hull, "A Database for Handwritten Text Recognition Research," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, pp. 550–554, 1994.

[140] B. Fernando, T. Tommasi, and T. Tuytelaars, "Joint Cross-Domain Classification and Subspace Learning for Unsupervised Adaptation," *Pattern Recognition Letters*, vol. 65, pp. 60–66, 2015.

[141] S. Si, D. Tao, and B. Geng, "Bregman Divergence-Based Regularization for Transfer Subspace Learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 7, pp. 929–942, 2010.

[142] M. Enzweiler and D. M. Gavrila, "Monocular Pedestrian Detection: Survey and Experiments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 12, pp. 2179–2195, 2009.

[143] B. Yang, C. Huang, and R. Nevatia, "Learning Affinities and Dependencies for Multi-Target Tracking Using a CRF Model," in *Conference on Computer Vision and Pattern Recognition*, 2011, pp. 1233–1240.

[144] S. Sivaraman and M. M. Trivedi, "A review of recent developments in vision-based vehicle detection," in *IEEE Intelligent Vehicles Symposium*, 2013, pp. 310–315.

[145] Q. Yuan, A. Thangali, V. Ablavsky, and S. Sclaroff, "Multiplicative kernels: Object detection, segmentation and pose estimation," in *Conference on Computer Vision and Pattern Recognition*, June 2008, pp. 1–8.

[146] W. Luo, J. Xing, X. Zhang, X. Zhao, and T.-K. Kim, "Multiple Object Tracking: A Literature Review," in *arXiv Preprint*, vol. abs/1409.7618, 2015.

[147] T. Vojir, J. Matas, and J. Noskova, "Online Adaptive Hidden Markov Model for Multi-Tracker Fusion," *Computer Vision and Image Understanding*, vol. 153, pp. 109–119, 2016.

[148] B. T. Phong, "Illumination for Computer Generated Pictures," *Commun. ACM*, vol. 18, no. 6, pp. 311–317, June 1975.

Artem Rozantsev

EPFL / IC / ISISM / CVLab
Station 14, CH-1015
Lausanne, Switzerland

+41(0) 78 947-27-21
E-mail: artem.rozantsev@gmail.com
Skype: artem.rozantsev
http://people.epfl.ch/artem.rozantsev

Interests

Computer Vision	Object detection and tracking, 3D reconstruction
Computer Graphics	Realistic synthetic data generation, realistic environment modeling
Deep Learning	Deep domain adaptation, deep pose estimation

Experience

2012-present — **CVlab, Ecole Polytechnique Federale de Lausanne (EPFL),** Switzerland, *Doctoral Assistant (PhD)*

- **Project** 'Detect and Avoid Systems for Unmanned and Manned Aviation'
  - Developed small fast moving objects detection algorithm from a single moving camera
  - Developed an automatic system for synthetic data generation
  - Designed an Deep Learning approach to efficiently combine real and synthetic data, which yields 20% accuracy increase
- **Project** 'Human 3D pose estimation'
  Developed and implemented algorithms for precise person localization
- **EU Project** http://www.mycopter.eu/
  Developed detection algorithms for visual-based aircraft detection

Tools: C/C++, MATLAB and Python

June 2016 – Sep 2016 (3 months) — **Microsoft Research**, Redmond, Washington, USA, *Research Intern*
Developed a 3D reconstruction technique for the recovery of the trajectory of small fast moving objects from multiple weakly calibrated static cameras. Part of this work has been accepted as an Oral presentation at CVPR 2017.
Tools: C/C++ and MATLAB

April 2016 – May 2016 (2 months) — **Honeywell**, Prague, Czech Republic, *Visiting Researcher*
As part of an EU project http://www.centaur-project.eu/. Improved tracking algorithms for multi-camera video surveillance.
Tools: C/C++ and MATLAB

2010–2012 — **ASTEROS**, Moscow, Russia, *Software developer*
Tools: MS SQL, Oracle Access and Identity Manager software

Education

2012–present (expected in may 2017) — **Ecole Polytechnique Federale de Lausanne (EPFL)**, Lausanne, Switzerland
Ph.D. in Computer Vision
Topic: Visual Detection and Tracking of Flying Objects in Unmanned Aerial Vehicles

2007–2012 — **Lomonosov Moscow State University (MSU)**, Moscow, Russia
Specialist (5 years).
Diploma with honors in Mathematics and Computer Science. (CGPA 4.7/5)

Summer 2010 — **Cornell University**, Ithaca, NY, USA
Business Management, Intensive case-study program with emphasis on strategic analysis. (GPA A-)

Technical Skills

C/C++	**6 years** of experience with OpenCV, OpenGL and STL libraries
Matlab	**4 years** of experience with toolboxes for computer vision and machine learning
Python	**2 year** experience with Numpy, Scipy and Theano for general purpose programming and machine learning with Neural Networks
Other	**Software**: QT, Blender

Publications

- A. Rozantsev, S.N. Sinha, D. Dey and P. Fua. 'Flight Dynamics-based Recovery of a UAV Trajectory using Ground Cameras'. Accepted to Computer Vision and Pattern Recognition conference (CVPR), 2017
- A. Rozantsev, V. Lepetit and P. Fua. 'Detecting Flying Objects using a Single Moving Camera'. Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2017
- A. Rozantsev, M. Salzmann and P. Fua. 'Beyond Sharing Weights for Deep Domain Adaptation'. arXiv preprint, 2016

- K. Sapkota, S. Roelofsen, A. Rozantsev, V. Lepetit, D. Gillet, P. Fua and A. Martinoli. 'Vision-Based Unmanned Aerial Vehicle Detection and Tracking for Sense and Avoid Systems'. In International Conference on Intelligent Robots and Systems (IROS), 2016
- B. Tekin, A. Rozantsev, V. Lepetit and P. Fua. 'Direct Prediction of 3D Body Poses from Motion Compensated Sequences'. In Computer Vision and Pattern Recognition conference (CVPR), 2016
- A. Rozantsev, V. Lepetit and P. Fua. 'Flying Objects Detection from a Single Moving Camera.' In Computer Vision and Pattern Recognition conference (CVPR), 2015
- A. Rozantsev, V. Lepetit and P. Fua. 'On Rendering Synthetic Images for Training an Object Detector'. In Computer Vision and Image Understanding (CVIU), 2015
- A. Rozantsev, C. M. Christoudias, V. Lepetit and P. Fua. 'Detection of Aircrafts on a Collision Course using Spatio-Temporal HOG'. Technical report, 2013
- V. Trofimov and A. Rozantsev. '2D soliton formation of BEC at its interaction with external potential'. In Proc. SPIE, Photonic Fiber and Crystal Devices: Advances in Materials and Innovations in Device Applications VI, 2012

Teaching

Teaching Assistant (EPFL): Computer Vision and C/C++ programming courses

Honors

Privileged Admission–Full Scholarship, Lomonosov Moscow State University
Winner of the Multidisciplinary Olympiad, Lomonosov Moscow State University

Extracurricular

Co-President of the Graduate Student Association at the doctoral school of the computer science department at EPFL
- Organized Open House events of the Computer Science department in EPFL
- Organized social events for the new Ph. D. students of the department

Academic Group Leader, Lomonosov Moscow State University

Language

English: fluent TOEFL(100)
Russian: native
French: elementary (A2/B1)

Hobbies

Volleyball, snowboard, soccer