MACHINE LEARNING-BASED MULTIMEDIA ANALYTICS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Daniel Mas Montserrat

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

June 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. Edward J. Delp, Chair

School of Electrical and Computer Engineering

Dr. Jan P. Allebach

School of Electrical and Computer Engineering

Dr. Fengqing M. Zhu

School of Electrical and Computer Engineering

Dr. Qian Lin

School of Electrical and Computer Engineering

Approved by:

Dr. Dimitrios Peroulis

Head of the School Graduate Program

A les meves àvies, la Roser i la Nuri.

To my grandmothers Roser and Nuri.

ACKNOWLEDGMENTS

I would like to start by thanking my doctoral advisor Professor Edward J. Delp for giving me the opportunity of being part of the Video and Image Processing Laboratory (VIPER). This thesis has been possible thanks to his guidance and support during my time at Purdue University both as a visiting undergrad and as a Ph.D. student. I would also like to thanks Dr. Qian Lin and Professor Jan P. Allebach for their mentoring during my stays at HP Inc. And to Professor Fengqing Maggie Zhu for her insightful feedback and suggestions.

I would like to thank all of VIPER Lab former members whom all helped me in one way or another during their time at the VIPER Lab: Dr. Neeraj J. Gadgil, Dr. Khalid Tahboub, Dr. Joonsoo Kim, Dr. Yu Wang, Dr. Chichen Fu, Dr. Shaobo Fang, Dr. Javier Ribera Prat, Dr. David Joon Ho, Dr. Soonam Lee, Dr. Jeehyun Choe, Dr. Dahjung Chung, Blanca Delgado Parra, He Li, Chang Liu, Dr. Yuhao Chen, and Dr. David Guera Cobo. I would also like to thank all its current members, without whom this would have been a much more boring journey: Sriram Baireddy, Emily R. Bartusiak, Enyu Cai, Alain Chen, Di Chen, Qingshuang Chen, Jiaqi Guo, Mridul Gupta, Shuo Han, Hanxiang (Hans) Hao, Jiangpeng He, Janos Horvath, Han Hu, Runyu Mao, Ruiting Shao, Zeman Shao, Changye Yang, Yue Han, Liming Wu, Justin Yang, Sri Kalyan Yarlagadda, and Yifan Zhao.

I would also like to thank my colleagues at Stanford University, especially Dr. Alex Ioannidis, Professor Nilah Ioannidis, Professor Carlos D. Bustamante, and Mr. Javier Blanco for their guidance and friendship.

I would like to give thanks to all my family, to my sister Mariona, my parents Roser and Daniel, and my grandmothers, Roser and Nuri. Thanks for supporting me my whole life. Without their unconditional and endless support and love, this journey would have not been possible. I want to give a special thanks to the family Tomàs Monsó and my friends Oriol, Pau, and Victor and Jorge, Peca, Mikel, Sandra, Stefano, Pol, Albert, and Xavi.

The material of chapter 2 and chapter 3 is based on research sponsored by HP Inc. The material of chapter 4 and chapter 5 is based on research sponsored by DARPA and Air Force Research Laboratory (AFRL) under agreement number FA8750-16-2-0173. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA and Air Force Research Laboratory (AFRL) or the U.S. Government.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

NOMENCLATURE

6DoF	6 Degrees of Freedom
AI	Artificial Intelligence
AFRL	Air Force Research Laboratory
AP	Average Precision
AUC	Area Under the Curve
CNN	Convolutional Neural Network
DARPA	Defense Advanced Research Projects Agency
FC	Fully Connected
GAN	Generative Adversarial Network
GPU	Graphic Processing Unit
GRU	Gated Recurrent Unit
HSV	Hue, Saturation and Value
LSTM	Long-Short Term Memory
mAP	Mean Average Precision
NGA	National Geospatial-Intelligence Agency
PR	Precision Recall
RGB	Red, Green and Blue
ReLU	Rectified Linear Unit
ROC	Receiver Operating Characteristic
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
VAE	Variational Autoencoder

ABSTRACT

Mas Montserrat, Daniel Ph.D., Purdue University, June 2020. Machine Learning-Based Multimedia Analytics. Major Professor: Edward J. Delp.

Machine learning is widely used to extract meaningful information from video, images, audio, text, and other multimedia data. Through a hierarchical structure, modern neural networks coupled with backpropagation learn to extract information from large amounts of data and to perform specific tasks such as classification or regression. In this thesis, we explore various approaches to multimedia analytics with neural networks. We present several image synthesis and rendering techniques to generate new images for training neural networks. Furthermore, we present multiple neural network architectures and systems for commercial logo detection, 3D pose estimation and tracking, deepfakes detection, and manipulation detection in satellite images.

1. INTRODUCTION

1.1 Multimedia Analytics And Machine Learning

Multimedia has become a key element of our daily lives. Multimedia consumption is a major form of entertainment and its production is a big industry. New multimedia content is now generated faster than ever before. Text, image, audio, video, and animation have been the main multimedia formats of the previous decades. Recently, new formats are becoming more popular and accessible due to the inexpensive price of smartphone devices and easy access to the internet. Some examples include Virtual Reality (VR) and Augmented Reality (AR), that complement the traditional mediums of multimedia.

In this scenario, a new set of tools to analyze, create, and improve all types of multimedia are required. These tools need to be able to process and extract information from large quantities of data in a fast and accurate way. Artificial Intelligence (AI) systems, specifically Machine Learning (ML) and Deep Learning (DL) algorithms are well fitted for such tasks as they can learn statistical patterns from training data to perform specific pre-defined tasks. In this thesis we study the use of neural networks, the basis of deep learning, in order to extract information from different formats of multimedia, including images and videos. Neural networks are composed of multiple layers of linear and non-linear operations, providing end-to-end systems that differ from traditional machine learning systems where hand-crafted feature extractors, designed by domain experts, are combined with statistical methods. Figure 1.1 represents two different approaches to image classification: machine learning, composed of two different steps of feature extraction and classification, and deep learning, composed by a neural network-based end-to-end method.

Neural networks have proved of being highly accurate in many fields of engineering and science, in many cases surpassing the performance of expert-designed ad-hoc methods. An area that has highly benefited from a large amount of data available on the internet and the new machine learning algorithms, such as neural networks, is computer vision. Nowadays, many tasks like image classification or object localization can be performed more accurately than humans by neural networks. The success of deep learning, specifically neural network-based methods, has been fueled by two main aspects: First, the in-

crease of the number of available data, specifically labeled data, and second, the hardware improvement of graphical processing units (GPUs) allowing a highly parallelized processing. Modern GPUs enable that, through gradient-based optimization methods coupled with back-propagation, neural networks are capable of learning from huge amounts of labeled data.

This thesis presents some examples of how machine learning algorithms can be used to analyze images and videos to extract meaningful information and perform tasks including object detection, pose estimation and tracking, satellite image manipulation detection, and deepfakes detection.

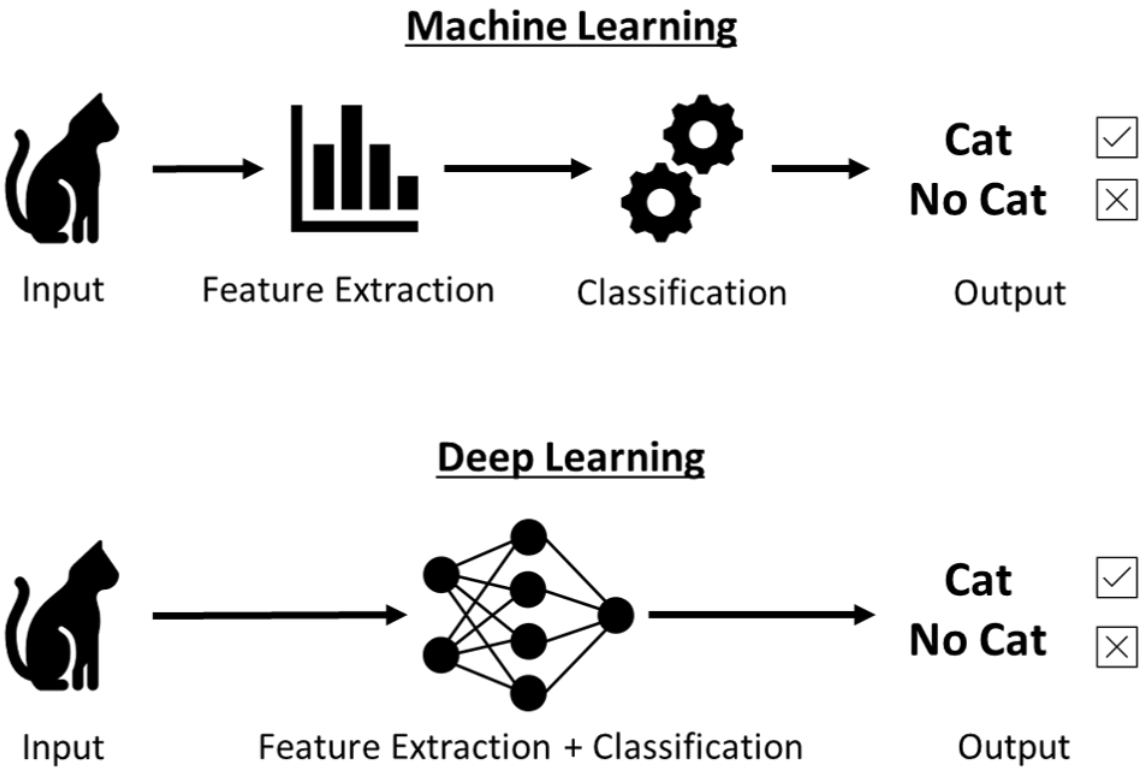

Fig. 1.1. Traditional machine learning systems rely on hand-crafted features and statistical classifiers that learn from data while deep learning systems learn in an end-to-end manner to extract and classify features.

1.2 Object And Logo Detection And Segmentation

Object detection consists of localizing and classifying the elements (objects) of interest present in an image or video. Object segmentation consists of assigning a category for each pixel of every element detected in an image or video. Figure 1.2 shows a representation of image classification and object detection and segmentation. Object detection and segmentation are steps present in many applications, including smart home, autonomous driving, robotics, surveillance, biomedicine, media forensics, photography, entertainment, and multimedia analytics. For example, object detection methods are used for detecting vehicles and pedestrians in self-driving cars, detecting, tracking and identifying people in surveillance applications, or detecting and classifying cancerous cells in medical imaging. While a large number of methods have been presented in the last decades to perform such tasks, convolutional neural networks, a deep learning method, have proved to be highly effective. By learning a set of hierarchical convolutional filters combined with non-linear operations, convolutional neural networks are able to successfully detect and identify objects within images and videos.

In this thesis, we use convolutional neural networks for object detection and segmentation in order to detect commercial logotypes. Logotypes, or logos, are an important form of multimedia. They can be found in commercial products, billboards, webpages, TV advertisements, and clothes. Their abundance and their importance in our social and economic life generate a need for tools to detect and classify them in an automated manner. The detection of logos in images or videos can be highly useful when studying the presence of a product or brand on the internet or TV. The analysis of how often and where logos are located can help companies to improve the placement of their logos and to study their consumers in order to augment the visibility of the brand.

In chapter 2 we present multiple approaches to generate images containing logos and to train convolutional neural networks to detect logos in the wild. Additionally, we show how the same techniques can be applied to other applications such as detecting human heads for sleep analysis.

Fig. 1.2. Image classification is the task of assigning one label per image. Object detection is the task of localizing and classifying the objects composing an image. Object segmentation consists of estimating a pixel-level segmentation mask for each object composing the image.

1.3 Pose Estimation And Tracking

Pose estimation is the task of inferring the position and pose of an object in the 3-dimensional world from 2D RGB images, depth information files, videos, or other multimedia formats. The pose of an object can be represented with 6 parameters: 3 rotation angles and 3 coordinates. 6 Degrees of Freedom (DoF) pose estimation and tracking is an important step for applications in augmented reality, robotics, surveillance, videogames, product maintenance, and design, and autonomous driving and navigation. For example, in order to properly place a virtual object in an augmented reality scenario, the pose and location of the objects in the scene need to be properly estimated. In order to properly grab and manipulate an object, a robot needs to detect the target object in the scene and accurately predict its pose and position in the 3D world. With the increasing number of products (e.g. smart glasses) and applications (e.g. interactive videogames) that rely on augmented reality and virtual reality technologies, techniques to infer the pose of the physical world in a fast and accurate way are needed.

6D Pose estimation has been commonly performed by extracting visual features from images and matching them to 3D models of the objects of interest. However, these approaches tend to fail when the target objects do not contain discernible visual features, they are occluded, fast-moving, or in challenging illumination conditions. Recently, convolutional neural networks have been successful in estimating and tracking the pose of objects, even if they are textureless, highly occluded, or in unfavorable illumination settings. Figure 1.3 presents a diagram of pose estimation with a neural network. While depth information can be highly informative when estimating 6D pose, devices that capture images with depth information are not highly available (e.g. consumer cameras or smartphones). Therefore, we focus on pose estimation and tracking from RGB color images and videos (without depth information). The shape of the target object can be known beforehand or, if unknown, can be estimated through 3D shape estimation techniques during the pose estimation process. We focus on pose estimation when the 3D model of the object is known as it is common in many applications (e.g. augmented reality for industrial product support and maintenance).

In chapter 3, we present multiple approaches on how to infer the location and pose of objects in the physical world from RGB images by using multiple convolutional neural networks.

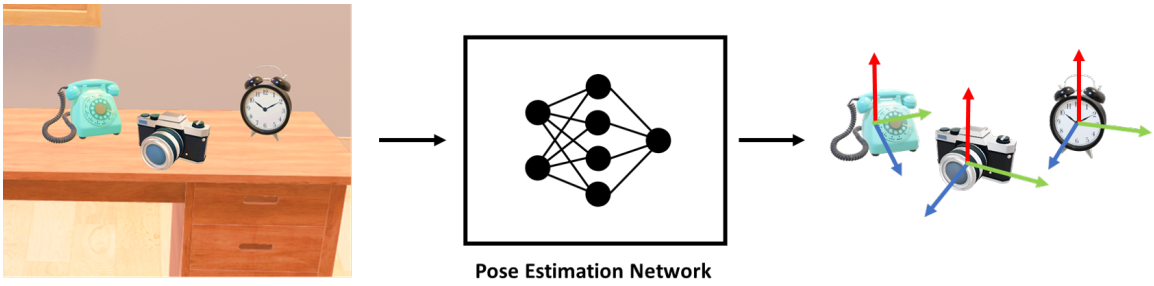

Fig. 1.3. Convolutional neural networks can be used to estimate the pose of multiple objects present in an image.

1.4 Media Forensics

Media forensics is the field that studies the integrity of multimedia and explores how manipulations can be detected. Altered and manipulated multimedia is increasingly present and widely distributed via social media platforms. Advanced and highly accessible video manipulation tools enable the generation of highly realistic-looking altered multimedia. Furthermore, machine learning techniques enable the automation of manipulating multimedia and social media allows for the uncontrolled spread of manipulated content at a large scale. In this thesis, we focus on manipulations within images and videos. These manipulations vary largely in nature. Common manipulations include copying elements of the image and duplicating them (copy-move), taking elements of other sources and placing them on the image or video (splicing), or face manipulations such as swapping faces or expression manipulation. Deepfakes, or manipulations performed with deep learning techniques, are becoming increasingly realistic and difficult to detect. Figure 1.4 presents how deepfakes can be generated with autoencoders.

We focus on two types of manipulations: deepfake manipulations performed within faces and splice manipulations performed in satellite images. Human facial manipulations are among the most common deepfake forgeries. Through face swaps, an individual can be placed at some location he or she was never present at, and by altering the lip movement and the associated speech signal, realistic videos can be generated of individuals saying words they actually never uttered. This type of manipulation can be damaging when used to generate graphic adult content or fake news that can alter public opinion. In fact, many images and videos with deepfake forgeries are already present on adult content web sites, news articles, and social media. Satellite imagery can be easily manipulated. Through simple manipulation techniques, an element of the image (such as an airplane or vehicle) can be removed or inserted. These types of manipulations can spread misinformation if used improperly. Furthermore, they can damage downstream applications that rely on satellite images, such as meteorological prediction algorithms, agricultural planning systems, or surveillance technologies.

In chapter 4, we show how convolutional and recurrent neural networks can be used to detect deepfakes manipulations within images and videos. In chapter 5 we show how generative networks can be used to detect manipulation within satellite imagery.

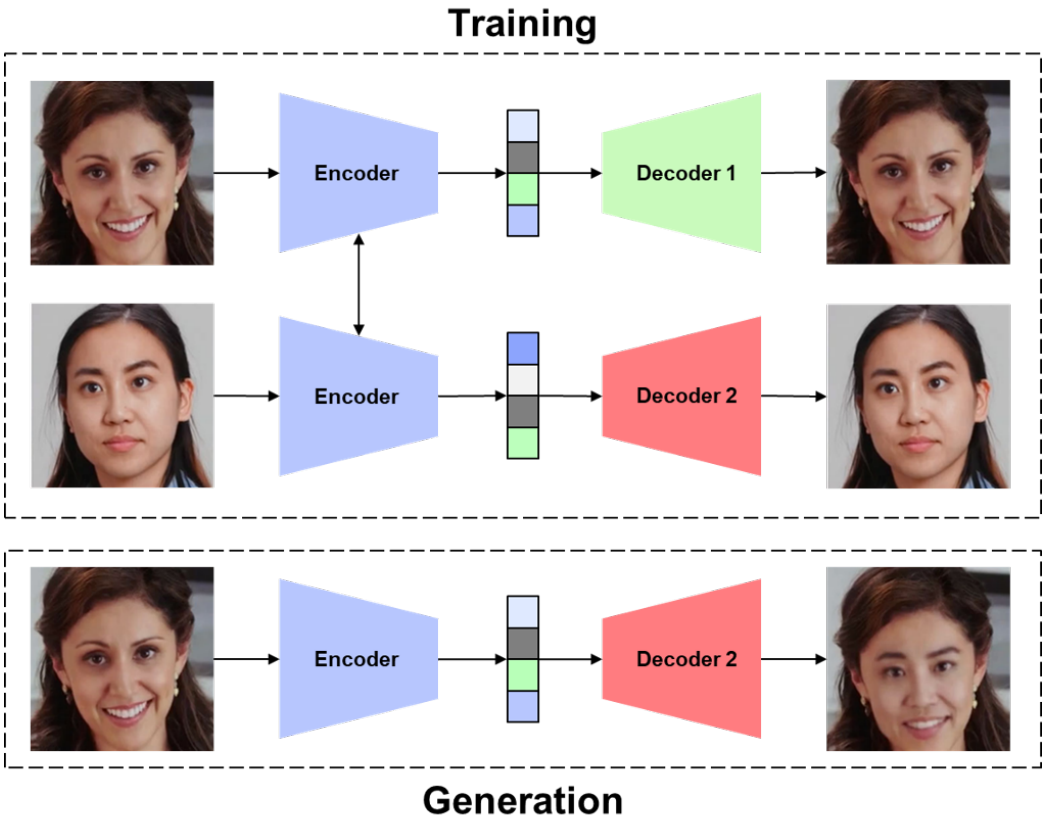

Fig. 1.4. Deepfake generation with autoencoders.

1.5 Contributions Of This Thesis

- We present multiple methods for image synthesis and data augmentation.
- We propose multiple methods to automatically detect logos in the wild.
- We introduce a method to generate training samples from 3D models of target objects.

- We show how neural nets can be used to detect, segment and estimate and track the pose of objects.
- We present a technique to detect deepfake face manipulations within videos.
- We introduce a method based on generative ensembles to detect manipulations within satellite images.

1.6 Publications Resulting From This Work

1. **D. Mas Montserrat**, Q. Lin, J. Allebach, and E. J. Delp, "Training object de- tection and recognition CNN models using data augmentation", Proceedings of the IS&T International Symposium on Electronic Imaging, January 2017, Burlingame, CA.
2. **D. Mas Montserrat**, Q. Lin, J. Allebach, and E. J. Delp, "Logo detection and recognition with synthetic images", Proceedings of the IS&T International Symposium on Electronic Imaging, January 2018, Burlingame, CA.
3. J. Choe, **D. Mas Montserrat**, A. J. Schwichtenberg, and E. J. Delp, "Sleep analysis using motion and head detection", Proceedings of the IEEE Southwest Symposium on Image Analysis and Interpretation, April 2018, Las Vegas, NV.
4. **D. Mas Montserrat**, Q. Lin, J. Allebach, and E. J. Delp, "Scalable logo detection and recognition with minimal labeling", Proceedings of the IEEE International Conference on Multimedia Information Processing and Retrieval, pp. 152-157, April 2018, Miami, FL.
5. **D. Mas Montserrat**, J. Chen, Q. Lin, J. Allebach, and E. J. Delp, "Multi-View Matching Network for 6D Pose Estimation", Proceedings of the IEEE Computer Vision and Pattern Recognition Workshops, June 2019, Long Beach, CA.
6. Q. Lin, **D. Mas Montserrat**, J. Allebach, and E. J. Delp, "Selecting training symbols for symbol recognition", WO2019152017A1, 2019-08-08, PCT/US2018/016211.

7. J. Chen, **D. Mas Montserrat**, Q. Lin, E. J. Delp, J. P. Allebach, “Extra FAT: A photorealistic dataset for 6D object pose estimation”, Proceedings of the IS&T International Symposium on Electronic Imaging, January 2020, Burlingame, CA.

8. **D. Mas Montserrat**, H. Hao, S. K. Yarlagadda, S. Baireddy, R. Shao, J. Horvath, E. Bartusiak, J. Yang, D. Guera, F. Zhu, E. J. Delp, “Deepfakes Detection with Automatic Face Weighting”, Proceedings of the IEEE Computer Vision and Pattern Recognition Workshops, June 2020, Seattle, WA.

2. OBJECT DETECTION AND SEGMENTATION

2.1 Overview

Object detection, recognition, and segmentation are some of the most important tasks in computer vision since they are key steps for many applications including multimedia analytics, biomedical imaging, smart home, smart office, surveillance, autonomous driving, and robotics. First, we will define the terms of image classification, object detection, and object segmentation. Image classification is the task of assigning a label or category (also referred to as class) to an image. Such category (commonly from a finite and predefined set) typically consists of a semantic representation of the image. Object detection is the task of estimating the locations (typically represented as bounding boxes) and categories of the objects composing the image. In this scenario, the object detection system will output a bounding box and a label for each object detected. The task of object segmentation consists of providing a pixel-level category for the objects detected within the image. An object segmentation system will typically output a label and a segmentation mask for each element detected within the image. While object detection is used in many applications, in this chapter we focus on one scenario: detecting commercial logos in the wild. Additionally, we briefly discuss how the same set of techniques can be applied in a different application: Videosomnography (section 2.6).

A logotype, or logo, is a graphic mark or symbol used to represent and identify a concept or thing (such as a company or a commercial product). We are constantly exposed to visual symbols, especially to commercial logos. Logos are a crucial part of marketing and advertisement. They are found on billboards, web-pages, magazines, commercial products, clothes, and on most objects people interact with on a daily basis. These symbols can represent a company, a product, an institution (e.g. Purdue University), a sports team, a nationality (e.g. USA flag), etc. They are an important element of our social and economical life therefore companies spend large amounts of money to design meaningful and appealing logos. These logos are usually simple and contain a small number of colors (typically between one and four). Logos might or might not contain text and if they do

is typically text with a small number of characters. Figure 2.1 presents some examples of different commercial logos.

Fig. 2.1. Examples of commercial logo images.

Logo detection is an important part of the validation of product placement, online brand management, consumer analysis, and contextual ad placement (placing relevant ads on webpages, images, and videos) [1]. In the era of big data and social media, large amounts of multimedia data are generated daily on the internet. Placing products and logotypes in images and videos of social media and TV has become a standard marketing technique. Analyzing the presence of brands and logos in social media, television or printed media can be a tedious task if done manually. This creates the need for analysis tools to locate and identify products and logos within images and videos in an automated manner.

New advances in artificial intelligence (AI), specifically machine learning (ML) and deep learning (DL) provide new methods to automatically identify objects within an image. These methods have provided state-of-the-art results when applied to the task of logo detection and recognition. Deep learning methods use artificial neural networks that learn from a large amount of labeled (groundtruth) data instead of using complex engineered handcrafted visual features. Therefore, obtaining large amounts of data/label pairs is important to obtain accurate results. The same techniques used for common object detection and recognition can be used for detecting logos within images. Several API services using

AI technology are capable to detect commercial logos [2–5]. One example where a logo detection system has been used as a tool for brand analysis in social media is presented in [6], where beer brand logos are detected in images extracted from Twitter [7] and are combined with male/female face detection to study the presence of the brands on the internet and its relation with the gender of the consumers.

Our work makes use of deep learning methods, in particular, the Faster R-CNN (Region-based Convolutional Neural Network) [8]. This network is composed of three main parts: a feature extractor, a region proposal network, and a classifier. Such a network, described in detail in the following section, learns from training datasets to detect multiple objects in a scene. However, available datasets containing logo images are usually limited in the number of logos (classes) and the number of images [9–11]. This can be a problem as methods based on deep learning typically require large amounts of training images (around 5,000 training samples per class [12]) to have a good performance. Also, real-world applications require to work with a large variety of logos (classes) and the possibility of easily adding new ones. Data augmentation, image synthesis, and bootstrapping methods can provide useful and scalable alternatives to the tedious and expensive task of manually collecting and labeling large amounts of images. In this chapter, we present multiple methods to automatically obtain new images containing logos and show how can they be used to train multiple Convolutional Neural Networks (CNNs). Specifically, we will present two different techniques of image synthesis and a method of bootstrapping to obtain new training samples.

Data augmentation methods typically apply linear and nonlinear transformations on the training data to create new samples. Transformations can include color changes, spatial rotations, warping, and other deformations. This set of transformations do not change the labels of the training samples. Image synthesis methods consist of creating new images from scratch or by combining other images. One or multiple labels can be assigned to the generated images. Several methods have been presented to synthesize images to train object detectors [13, 14]. Bootstrapping techniques can be used to obtain new training images from the internet in an automated way. Weakly labeled images can be obtained

from the web using image search engines. Typically, the images obtained by the search engines contain the logos used in the search query. Some images might contain different logos or not contain any at all. The locations of the logos within the images are unknown. Therefore, the search term from the search query can be considered as a noisy label. In addition, a localization method must be included to obtain noisy locations of the logos.

In section 2.2 we present a background on neural networks for image classification, object detection, and object segmentation. Specifically, we describe a family of Region-based Convolutional Neural Networks (R-CNN) composed by R-CNN [15], Fast R-CNN [16], Faster R-CNN [8], and Mask R-CNN [17]. Additionally, we will review previous work for data generation and image synthesis and previous methods and datasets for logo detection.

In section 2.3, our first image synthesis method used to train Faster R-CNN is presented. In this method, simple image transformations are applied to objects and logo images and placed on top of background images. Then Faster R-CNN is trained using the generated images and evaluated with logo datasets containing real images.

In section 2.4, SynthLogo, a dataset generated with a complex image synthesis method, is presented. Our method is based on the technique used in SynthText [18]. SynthText is a dataset containing images with text created by blending rendered text into background images. In this method, images are created by applying transformations to images of logos and then blending them into background images using their depth and segmentation information. Then, we use the synthesized images to train the Faster R-CNN (Region-based Convolutional Neural Network) [8] and a variant named PVANet [19]. Both networks are composed of a feature extractor, a region proposal network, and a classifier. The networks are able to accurately locate and classify multiple logos in an image.

In section 2.5 we present a two-step approach. In this method, we combine bootstrapping and image synthesis to train a logo detection system to automatically localize and label logos in images. While recent work [20] also uses bootstrapping techniques, it does not include the ability to automatically localize the logos. Our method starts by first training the Faster R-CNN [8] model, with the synthetic images of SynthLogo (described in section 2.4). We then use the regions containing logos in the ground truth of the synthetic dataset to

train a second-stage classifier, the DenseNet [21] network, described in section 2.5.1. With the second-stage classifier, we can achieve a significant increase in classification accuracy. We can further boost the performance of our logo detection system by using weakly labeled data collected from the web. We detect and locate logos with Faster R-CNN on the weakly labeled images and use the detections to train the DenseNet classifier. During inference, we first use the Faster R-CNN algorithm to get the proposed logo locations. We then crop the pixels from the original image in the proposed locations and classify them using the second-stage classifier. While Faster R-CNN architecture is capable of classifying each region, it does so by using low resolution features obtaining bad classification results. By using the second-stage classifier, we can achieve much higher logo detection results. The main advantage of this method is that it is highly automated and requires minimal human supervision. While we conducted our experiments using logo images, the detection system described in this chapter can be used to train other objects such as text, signs, and flags. The only step that involves manual interaction is the acquisition of logo (or object) images for the image synthesis processes.

In section 2.6, we show how the same set of techniques used to detect logos can be used in different applications. Specifically, we briefly describe how object detection networks can be used to detect heads of children in order to analyze their sleep patterns in a video-somnography (VSG) application. Videosomnography includes video-based methods used to analyze if an individual is sleeping or awake. Manually assessing if the individual is sleep or awake can be a highly time-consuming task that needs to be performed by trained technicians (also referred to as coders). However, by using video analysis techniques, this process can be automated. We present an automated VSG sleep detection system that makes use of motion analysis to accurately determine sleep/awake states in infants. We show how Faster R-CNN can be trained to detect the head of the individual within the video. Then, the size of the detected head, combined with a motion index computed with the difference between frames, can be used as a proxy measure that indicates if the individual is awake or asleep.

2.2 Related Work

2.2.1 Background In Object Detection And Segmentation

Before the popularization of deep learning, the main approach for object detection and other image processing and computer vision tasks was the use of handcrafted visual features like SIFT [22] and texture descriptors [23], combined with statistical classifiers, such as k-Nearest Neighbor (k-NN) [24] or Support Vector Machines (SVM) [25].

In the last several years, deep learning methods have shown to provide higher accuracy compared to traditional approaches [8, 16, 26–28]. This improvement has been possible mainly by advances in hardware (e.g. more powerful GPUs) and the availability of large labeled datasets (e.g. ImageNet [26] contains more than 14M images). Deep learning methods have demonstrated impressive results in speech recognition, object recognition, and detection and in other domains such as drug discovery and genomics [29–33]. One deep learning approach that has achieved high accuracy in classification and detection is the Convolutional Neural Network (CNN) [12, 28, 29]. Convolutional Neural Networks are the core element of deep learning methods. These networks can be used in many different settings. The networks can typically be divided into two elements: a feature extraction subnet and a task-specific subnet (i.e. decision subnet in a classification problem). A convolutional neural network combines convolutional filter layers and non-linearity layers to learn and extract features (feature extraction subnet) and fully-connected layers to classify them (decision subnet) [28, 34]. Methods using CNNs are the leading approaches in image classification competitions (e.g. ImageNet [26]) and object detection competitions (e.g. Pascal VOC [35] and MS COCO [36]).

The feature extraction subnet can contain many convolutional layers and each layer contains multiple filters. When the network is trained to perform object classification or detection in natural images, the filters of the first convolutional layers are able to detect simple features such as color or edges. Filters in deeper layers learn more complex features (e.g. some layers can detect complex shapes such as faces, wheels, and animals). Between convolutional layers, non-linear layers such as the Rectified Linear Unit (ReLU) or Max-

pooling are included. ReLU layers compute the maximum between 0 and their input value. Max-pooling layers perform a non-linear down-sampling. The down-sampling process consists of partitioning the input image into non-overlapping rectangles and selecting the maximum value inside each rectangle. The outputs of the convolutional layers are usually known as feature or activation maps. In the scenario of object detection and classification, the convolutional layers are able to encode a representation of the objects to be detected or classified. These representations can be analyzed by observing which filters are activated when presented with visuals elements.

The decision subnet can contain multiple fully connected layers. Fully connected layers consist of a set of matrix multiplications followed by non-linear operations (typically a ReLU). The size of the last fully connected layer output is equivalent to the number of classes. The network outputs a probability or confidence value for each class. The decision subnet usually requires a fixed input size, therefore it is required an initial step of cropping or resizing the input images before the feature extraction subnet. This subnet is able to combine the features or visual representations extracted by the feature extraction subnet and produce an estimation of the object or class present in the picture.

The weights and parameters of the convolutional and fully connected layers are learned from training samples using backpropagation [29] in combination with gradient-based methods such as Stochastic Gradient Descent (SGD) [30] or Adaptive Moment Estimation (Adam) [37]. The learning process starts by assigning random values to the weights and parameters of the network. Then, two different stages, propagation, and weight update are repeated over a fixed number of iterations. First, an input image is propagated forward through the network until it reaches the output layer. Then, the output of the last layer is compared with the ground truth value using a loss function. The loss function is a function that generates an error measure. If the predicted output is close to the desired output, the error measure will be small. If the predicted output differs a lot from the desired one, the error will be large. The error value is backpropagated through the whole network using an optimization method (for example SGD). The optimization method updates the values of the weights and parameters of the network in order to minimize the loss function.

The training error, computed with the loss function, represents how well the network fits the training data. Typically, the training error underestimates the testing error. Testing error is the error that results when the network is used for a new observation that was not seen in the training process. If the gap between test and training error is large, we say that the network is overfitting. That means that the network has learned the training data but is not able to generalize to new examples.

A common practice in deep learning is to first train the CNN with a large generic dataset (e.g. ImageNet) and then use the weights and parameters obtained in the training process as an initialization. This process is known as fine-tuning or transfer learning.

In our work, we make use of two common CNN models: the Zeiler & Fergus (ZF) net [38] and the VGG16 (Visual Geometry Group) net [39]. The ZF Network has 5 pairs of convolutional and ReLU layers followed by 2 fully connected layers. The 1st convolutional layer has 96 filters with size 7×7, the 2nd convolutional layer has 256 filters of size 5×5, the 3rd, 4th, and 5th convolutional layers have 256, 384 and 384 filters respectively. Each filter has a kernel size of 3×3. VGG16 is a deeper model containing 5 sets of layers. Each set contains two convolutional and ReLU layers followed by a max-pooling layer. The number of filters in the convolutional layers is 65, 128, 256, 512, and 512 respectively. All the filters have a size of 3×3. VGG16 ends with 3 fully connected layers.

Vanilla CNN architectures are good for image classification but they can have problems while localizing multiple objects inside an image. The Region-Based Convolutional Neural Network (R-CNN) [15] is a network able to locate and classify several objects in images of any size by combining CNNs and external region proposal methods. A region proposal method is a method that finds a set of regions, typically defined with bounding boxes, that might contain objects of interest. Typical region proposal methods are Selective Search [40] and EdgeBoxes [41]. Selective Search splits the image into several regions of interest by using similarity measures based on color and visual features. EdgeBoxes finds regions of interest using object contours information. In the R-CNN, each region of interest is cropped and resized to 227×277 pixels. Then, the resized image is used as input of a CNN consisting of five convolutional layers and two fully connected layers. The CNN assigns

to each region of interest a class and a confidence score. However, this approach can be extended to any region proposal method and any image classification CNN architecture.

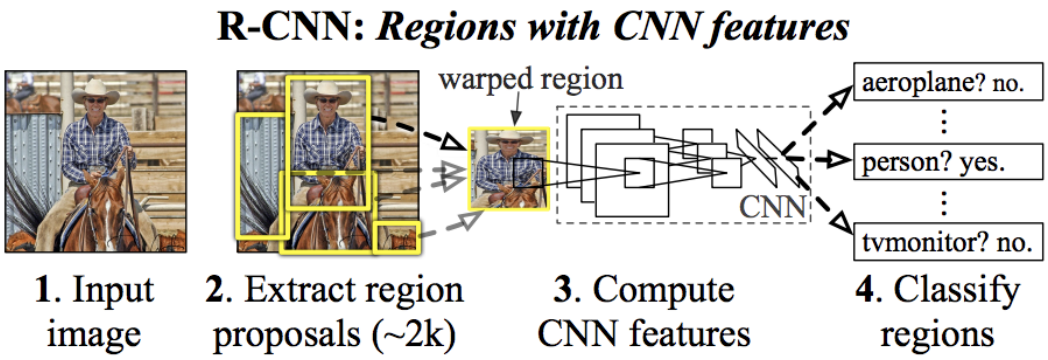

Fig. 2.2. R-CNN architecture. Figure adapted from [15].

The R-CNN, and many other object detection methods, processes the bounding boxes generated by region proposals methods using Non-Maximum Suppression (NMS) [15]. NMS rejects a bounding box if it has a large overlap with another bounding box with higher confidence. If the overlap is higher than a threshold, the bounding box is rejected. Typically, the threshold is a hyper-parameter assigned during the training process.

The main drawback of R-CNN is that it is computationally intensive since every image is processed as many times as the number of regions of interest detected. Previous work such as the Spatial Pyramid Pooling Network (SPPnet) [27] addresses this problem by using pooling. SPPnet starts with a CNN (e.g. VGG16 or ZF) followed by a spatial pyramid pooling layer and fully connected layers. The spatial pyramid pooling layer uses max-pooling for each region of interest using grids of multiple sizes. The regions of interest are computed using Selective Search. The images are processed only one time by the CNN. Then, spatial pyramid pooling is used to generate the output of the last convolutional layer (the last feature map) that is later classified by the fully connected layer.

The Fast R-CNN [16] is a network with the same structure as SPPnet but substitutes the spatial pyramid pooling with an RoI (Region of Interest) pooling layer. The RoI pooling

layer is a simplified version of the spatial pyramid pooling, where instead of using a grid with multiple sizes, only one size (typically 7×7) is used. Fast R-CNN also introduces a more effective method for training the CNN and adds a bounding box regressor. The network is trained using multiple regions per image instead of using only one as it is done in SPPnet. The bounding box regressor is a layer that fine-tunes the locations of bounding boxes where objects of interest might be located (initially provided by Selective Search).

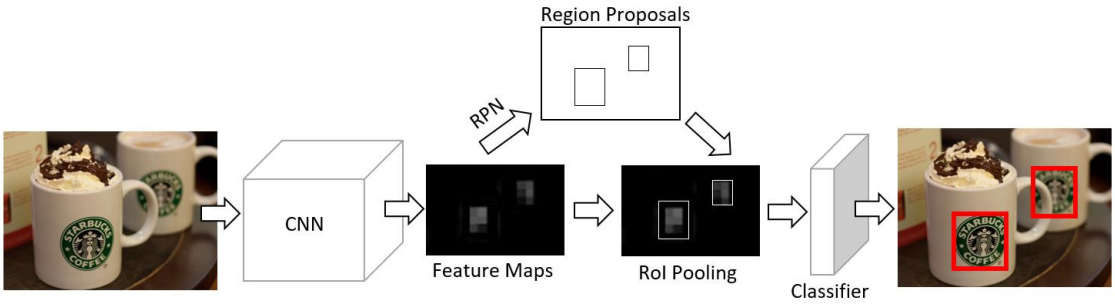

Fig. 2.3. Faster R-CNN combines a feature extractor, a Region Proposal Network, and a classifier.

In our work, we use Faster R-CNN, an improved version of the Fast R-CNN. The network, shown in figure 2.3, combines the Fast R-CNN with an RPN (Region Proposal Network). The RPN is a neural network that uses the output of the last convolutional layer of the CNN, the feature map, to generate regions of interest. The RPN consist of a 3×3 sliding window that outputs a set of 9 bounding boxes containing regions of interest (also referred to as anchors). Each bounding box has a different size and a different aspect ratio. A fully connected layer assigns a binary class (foreground or background) to each bounding box. Figure 2.4 presents the anchor system. Following the steps in the Fast R-CNN, each region of interest is applied to the RoI pooling layer and is later classified by a fully connected layer. In the classification step, a confidence score is assigned to each bounding box. The confidence value ranges from 0 to 1 where the confidence of 1 represents that the network is almost certain that the class assigned is correct. A threshold

is usually set to a high value (e.g. 0.7) in a deployment stage and all the bounding boxes below the threshold are discarded. The network can be trained end-to-end and provides an almost real-time performance. With the addition of RPN, there is no need to use external region proposals methods. The Faster R-CNN is the basis of several 1st-place entries in the ImageNet and MS COCO competitions [8]. It is also used in commercial systems such as Pinterest [42].

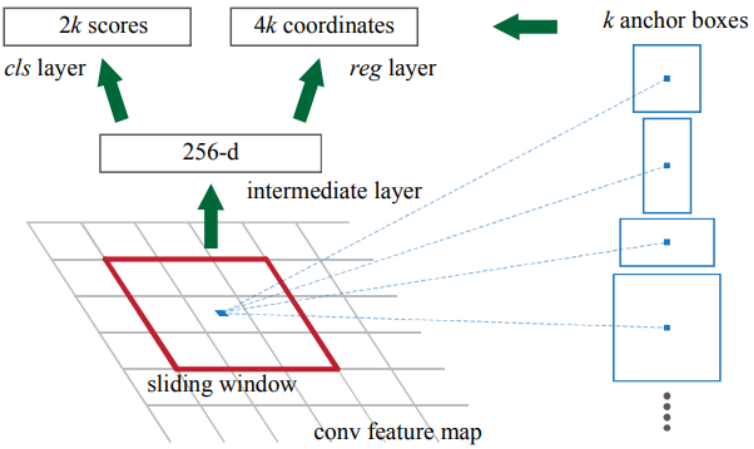

Fig. 2.4. Anchors in Faster R-CNN. Figure adapted from [8].

In our work, we also use PVANet [19], shown in figure 2.5. PVANet is a lightweight version of Faster R-CNN. The model is smaller with only 5 convolutional layers and 3 fully connected layers. While Faster R-CNN only uses the features of the last convolutional layer to localize and classify, this network combines the features of the last 3 convolutional layers. This method is faster than Fast R-CNN but can produce lower accuracy in some scenarios.

Other methods for image detection such as You Only Look Once (YOLO) [43] and YOLOv2 [44] provide real-time performance by compromising accuracy. Recent methods such as Single Shot Multibox Detector (SSD) [45] provide real-time performance and good accuracy but seem to perform poorly detecting small objects since it resizes the input images to a fixed size (typically of 300 $\times$ 300 pixels) and resolution is lost. Both methods

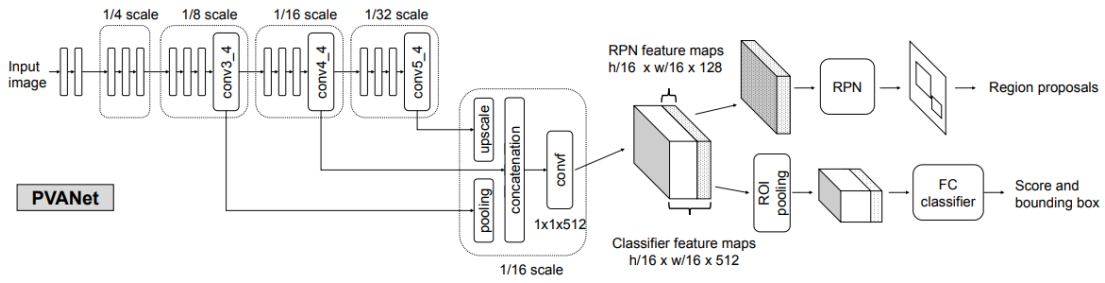

Fig. 2.5. PVANet architecture. Figure adapted from [19].

divide the image into a fixed number of regions and predict bounding boxes and probabilities for each region using fully connected layers.

Mask R-CNN [17] is an extension of Faster R-CNN designed to perform object segmentation. Faster R-CNN and Mask R-CNN [17] have been so widely used, adapted, and transformed that they are referred to as a meta-architecture in the recent literature. A major appeal of these networks is that their capabilities can be extended by adding new layers while sharing most of the weights for each task is trained to perform. Figure 2.6 presents Mask R-CNN architecture.

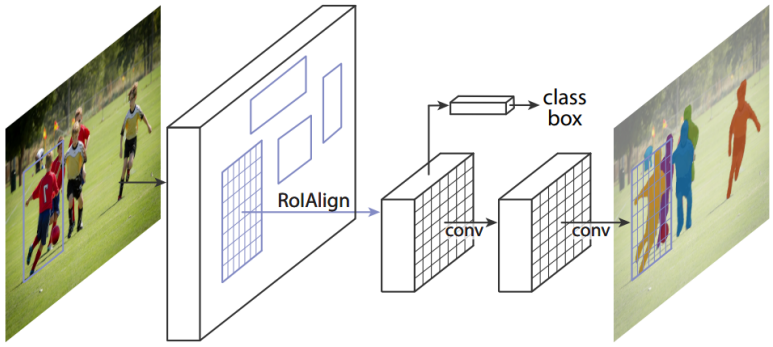

Fig. 2.6. Mask R-CNN architecture. Figure adapted from [17].

Mask R-CNN [17], and all its variations, share three key components: a backbone or feature extractor, a region proposal method, and some task-specific networks (typically referred to as heads). The backbone (previously referred to as "feature extraction subnet") consists of a set of convolutional layers and non-linearities. The backbone does not contain any fully-connected layer, only using convolutional or sliding-based operations, therefore allowing to have any input image size. A widely used backbone is ResNet [46]. ResNet is a network formed by many residual blocks. A residual block is a set of convolutions combined with an identity additive mapping from the input to the output. This identity connection solves the vanishing gradient problem found in many architectures (gradients collapsing to 0 during the training process). This structure allows us to have larger networks and a better training process due to a better flow of the gradient information. Figure 2.7 shows an example of a residual block. When working with a network based on Mask/Faster R-CNN, is a common practice and highly advised to first train the backbone network for a simpler task such as classification with lager datasets like ImageNet. Many pre-trained weights are publicly available on the internet. If the network is trained from scratch, many problems might arise as the region proposal methods will fail to find regions of interest and the task-specific heads will not generate a loss signal to train the network.

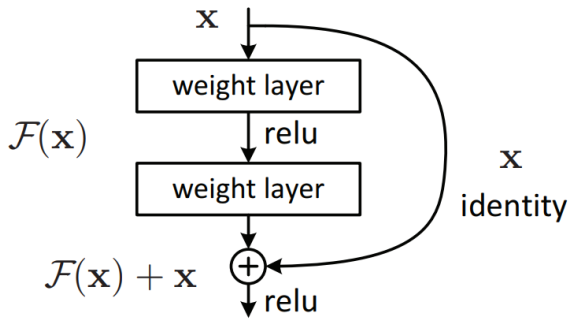

Fig. 2.7. Diagram of a Residual block in ResNet architectures. Figure adapted from [46].

The features extracted from the backbone network are used to perform specific tasks like classification or segmentation. In order to select which regions of the feature map contain useful features, a region proposal method is needed. As previously described,

many methods have been proposed for such tasks. Selective search [40] is used in Fast R-CNN [16]. Selective search looks for regions of interest within an image and then map them to the last feature map. Faster R-CNN [8] introduced the Region Proposal Network (RPN). The RPN is a convolution window that finds regions of interest from a feature map. The RPN removes the need for external methods (i.e. Selective Search) and allows them to train the network in an end-to-end fashion. The main problem of RPN is that regions of interest are estimated using only the last feature map of the backbone. Objects of multiple sizes can not be properly detected by using the features of only one layer of the backbone. In order to solve that, a system of pre-defined windows with multiple scales and aspect ratios named anchors, are estimated. The anchor system allowed to find objects of different scales and aspect ratios, but as only the features of the last convolutional layer are used, some spatial information is lost, damaging the performance of the detection of small objects. Mask R-CNN [17] substitutes the RPN by a Feature Pyramid Network (FPN) [47]. The feature pyramid network follows the same philosophy as the RPN but instead of using the features of the last convolutional layer, it uses the features of multiple layers. The anchor system is still used, but as multiple layers are used to estimate regions, a smaller number of anchors is required. The FPN is a faster approach and gives higher detection accuracy. Figure 2.8 shows multiple approaches to region proposal. The RPN follows the single feature map approach (b) while the FPN uses multiple feature maps (d).

As in Faster R-CNN, after the regions of interest are detected, a non-linear max-pooling is performed to the features inside the region of interest. The max-pooling is performed in a fixed size grid obtaining always the same number of features. The number of extracted features might change on the application (e.g. classification, segmentation...). These features are used as input for the task-dependent networks or heads. In Fast and Faster R-CNN, only one head is used (the classifier network combined with a bounding box regressor to refine the region of interest location). Mask R-CNN includes two extra heads. One head consists of a set of convolutional and deconvolutional layers to perform segmentation, and the other head is a human pose keypoint estimator.

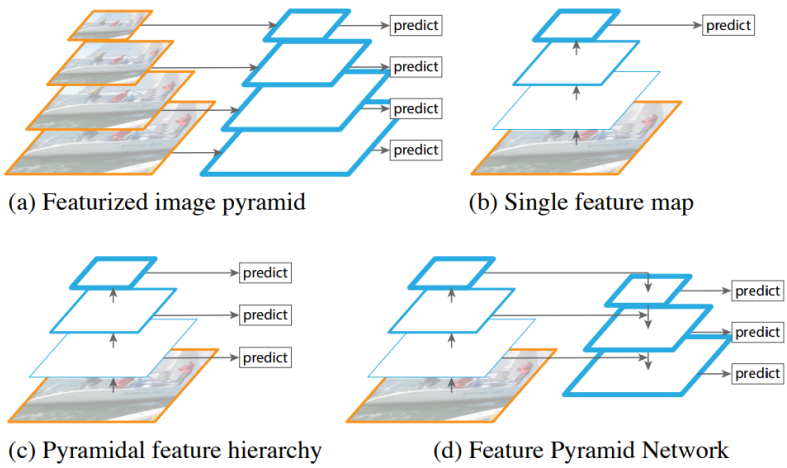

Fig. 2.8. Different approaches to region proposals. Figure adapted from [47].

2.2.2 Previous Works In Logo Detection

Typically, logo detection is performed by adapting object detection methods to the domain of commercial logos [1, 9] (i.e. treating each logo as a different object or class). Several previous works have been presented for logo detection and recognition using handcrafted visual features [9] and based on deep learning [1, 48]. The work presented in [1] makes use of Fast R-CNN with VGG16 and selective search for logo detection with and without localization obtaining a mean average precision of 74.4%. The work presented in [48] uses Faster R-CNN obtaining a 81.1% of accuracy. The work presented in [49] also uses Faster R-CNN with VGG16 and a smaller network named ZF [38] combined with data augmentation techniques obtaining a mean average precision of 85.4% on logo recognition with localization. The work presented in [50] uses an approach similar to R-CNN trained with a large number of images obtaining an accuracy of 96%. The works presented in [20, 51] apply bootstrapping techniques for logo detection. Several commercial API services enable detection of logos [2–5] in real scenes. The work presented at [52] uses an object detector network named YOLOv2 [44]. Faster R-CNN and R-CNN, described in section 2.2.1, are adapted and used in our work. The work presented in [20] applies bootstrapping techniques for logo detection.

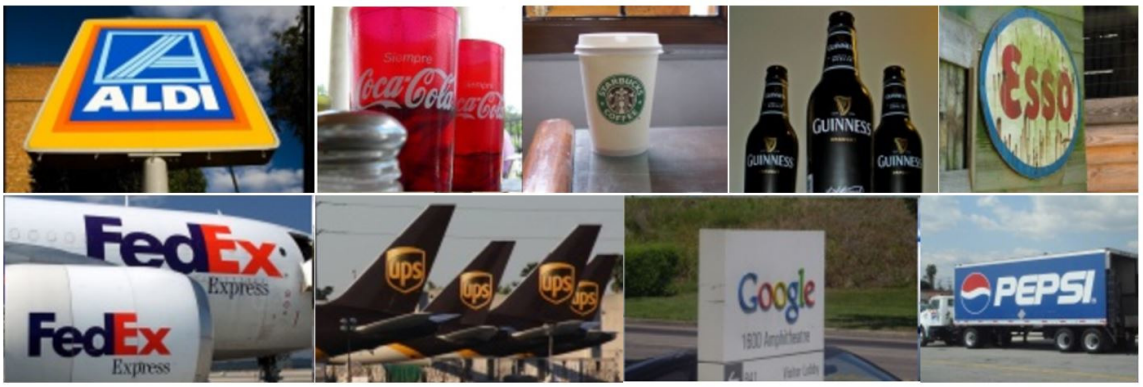

Fig. 2.9. Image samples from FlickrLogos-32.

There are several available labeled datasets with images containing logos in the wild. FlickrLogos-32 [9] dataset is the most used in recent works for training and testing. This dataset contains 32 different brands (classes), each with various versions. The dataset is composed of 8240 images mined from the Flickr image search engine [53]. The dataset is divided into a training set with 1280 images (40 per class) containing one or more logos and 3000 images with no logo content, and a testing set with 960 images (30 per class) containing genuine logos and 3000 without logo content. The images without logo content are also referred to as background images or distractors. The ground truth consists of a label (class) and a binary segmentation mask assigned to each image that contains a logo. A new version of the dataset, FlickrLogos-47, is also publicly available. FlickrLogos-47 contains the same images as FlickrLogos-32 but the labeling has been improved. In FlickrLogos-32, the logos composed by a symbol and text, only the symbol is treated as a logo, while in FlickrLogos-47 each part is treated as a different class (e.g. adidas-text, adidas-symbol). In our work, we use FlickrLogos-32 for evaluation purposes. Figure 2.9 and figure 2.10 presents some image examples of the dataset.

Other datasets include FlickrLogos-27 [54], which is composed of 27 different logos with a total of 810 annotated images (30 images per class) and 4207 distractor images. BelgaLogos [10] is a dataset with 37 different logos composed of 10,000 images with a

binary label (1 if the logo is present and 0 otherwise) and 1321 of them contain bounding boxes indicating the location of the logos. MICC-Logos [55] contains a total of 720 images with 13 different logos. TopLogo-10 [11] is a dataset containing 700 labeled images of 10 different clothing brands. Logos-32plus [56] is a dataset containing a total of 7830 images with logos from the same corpus of FickrLogos-32. Logos in the Wild Dataset [57] is a new dataset containing the largest available number of training samples with a total of 11,054 images with 871 different brands. LOGO-Net [48] is a dataset with 160 different logos with a total of 73,414 labeled images. However, LOGO-Net is not currently available to the public. WebLogo-2M [20] is a dataset with 194 logos present in 2,190,757 labeled images. This dataset does not contain bounding boxes but only the label of the logos present in the image. It has been labeled in an unsupervised manner so the labels might be incorrectly assigned. Currently, Logos in the Wild Dataset (LitW) [57] is the largest manually labeled dataset. LitW contains 11,054 labeled images and 871 logos. Later in the chapter, we introduce the SynthLogo [49] dataset. SynthLogo is a dataset that contains 280,000 images synthetically created with 604 different logos.

2.2.3 Previous Works In Data Augmentation And Image Synthesis

Data augmentation techniques, such as random cropping, flipping or color changes [26], have been used for object detection using handcrafted visual features [9, 58] and for deep learning [13]. Typical data augmentation techniques used in deep learning include image cropping, flipping, and color changes [26] to create the augmented or synthesized images. More complex techniques can include noise addition, geometric transformations, or image compression. These techniques, usually help the network to avoid overfitting and to generalize better obtaining a higher accuracy. The method presented in [58] combines multiple transformations to the training set. After the data augmentation process, an accuracy increase of 3.5% in 2010 ImageNet competition was reported.

Image synthesis has been used to create new training samples for deep learning methods. One common approach is to use image compositing by adding foreground images

(objects to be detected) into background images. This approach is used in [18] for text localization. The work presented in [11, 14] uses this approach for logo detection. Flickr-BelgaLogos [59] is a public dataset synthetically created using image compositing with the logos extracted from BelgaLogos.

Synthesized images from 3D virtual simulations have been used for training neural networks for self-driving vehicles [60] or other tasks based on reinforced learning [61]. Methods such as [62] use Recurrent Neural Networks (RNN) to generate new training samples using information extracted from a training dataset. Generative Adversarial Networks (GANs) are networks able to generate new images that resemble the images used in the training process. The work presented in [63] uses this approach to create new training samples. In the following chapter, techniques based on photorealistic rendering will be introduced.

Bootstrapping techniques have been used in many deep learning methods for image classification [64] and logo detection [20]. A bootstrapping method aims to increase the number of training samples by automatically assigning labels (with a CNN, for example) to unlabeled data and then using the weakly labeled data to train CNNs.

2.3 Object Detection And Image Synthesis

2.3.1 Proposed Approach

In this section, we describe our method for synthesizing images and how they can be used to train a CNN. In order to show the flexibility of this approach, we apply the method in the scenario of object detection (specifically in the detection of toys and logos).

Our approach is based on the work described in [13,58] which blends images of objects with real-world background images. The images of objects and logos undergo several transformations as described in the following sections. Images of objects are essential to data augmentation. For logo detection, the images are extracted from the FlickrLogos-32 [9] dataset. We selected FlickrLogos-32 for our training and evaluation purposes because

it contains a decent number of labeled images and is the one commonly used in previous works of logos in the wild detection.

FlickrLogos-32, as previously described, consists of 32 different brands (classes) each with various versions. The dataset contains 8240 images mined from Flickr [53]. Figure 2.9 shows samples from the FlickrLogos-32 dataset. The dataset is divided into training and testing parts. Training data is comprised of 1280 images (40 per class) containing genuine logos and 3000 images with no logo content. The 3000 images are used as background images (distractors). Testing data includes 960 (30 per class) images containing genuine logos and 3000 background images. Each image with a genuine logo is provided with the true class and a binary segmentation mask indicating the logo location.

In the case of object (toys) detection, high-quality images are captured using a high-definition camera contained in the HP Sprout [1]. The HP Sprout is a desktop computer released in November 2014 that has a projector, an HD camera, a 3D camera, a touch mat, and a LED desk lamp [65].

Fig. 2.10. Different logo versions for Adidas (left), Corona (center) and Starbucks (right).

[1]HP Sprout, HP Inc ®

Logo Extraction

Logo images are extracted from the FlickrLogos-32 training set using the binary segmentation masks and labels provided. Each image may include more than one logo. In total, we obtain between 60 and 80 images per brand. Images smaller than 20×20 pixels are discarded since they are very difficult to detect after data augmentation. Figure 2.11 shows examples of extracted logos from the FlickrLogos-32 dataset.

Fig. 2.11. Six different logos from FlickrLogos-32.

The same brand can have different versions of logos. Figure 2.10 shows the inter-class variation of three different brands. We do not make a distinction between logo versions and we assign one class per brand.

Object Capture

A total of 20 high-quality images are captured for every object in a different pose. The HP Sprout is used in the image acquisition process. The images are captured using the top HD camera with white light projected to the touch mat and the LED desk lamp turned off. In this work, no 3D information is captured or used.

Multiple poses are captured. Figure 2.12 shows six examples of the same object with different poses. As presented in the following sections, the number of poses used in the data augmentation process will affect the detection and precision performance. Intuitively,

if more poses are available for training, the network will be more resistant to rotations and change of poses. In this work, a set of 15 different toys was used. Figure 2.13 shows examples of different toys. Some figures have minor differences between them (set of small red and black toys). Despite that, the network is able to accurately differentiate them, as presented in the next sections.

Fig. 2.12. Captures of different poses

Fig. 2.13. Examples of six different toys

Synthetic Data Generation

We want to generate new training images containing the objects of interest (logos or toys). To accomplish this, we start by randomly selecting a background image from the MIT-Places dataset [66]. The MIT-Places dataset contains 205 scene categories and a total of 2.5 million images recorded at various locations around the world. We utilize the testing data within this dataset during the process of synthetic images generation. The testing data contains 41,000 images. Figure 2.14 shows samples from the MIT-Places dataset. We assume that the background image does not contain any object that we are trying to detect. This is a reasonable assumption since the MIT-Places dataset is focused on real-world or natural scenes. Then, several object or logo images are randomly selected (from 1 to 9 images). For each object or logo image, a set of transformations and deformations are used as described below.

Fig. 2.14. Examples from MIT-Places dataset

Geometric Transformations

First, the images are randomly rotated with a degree selected uniformly between -40 and 40. Then, a random homographic projection is used as defined by the matrix 2.1. Where the parameters h_{11} and h_{12} are randomly selected between -0.001 and 0.001. Next, the image is randomly resized such that the new size is 0.1 to 0.25 times the size of the background image. The parameters are manually selected in order to make the synthesized images

look as real as possible. Therefore, extreme resizes and highly deforming homographic transformations are unwanted.

$$\mathrm{H} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & h_{11} & h_{12} \end{pmatrix} \tag{2.1}$$

Geometric transformations aim to model different poses of objects and logos. By rotating and resizing the training images, the network is able to be scale and rotation invariant. By using perspective projections combined with capturing multiple poses in the object capture step, the network can be pose change invariant.

Color Transformation

After applying geometric transformations, a small color variation is performed to the images. Following the steps in [26, 58] we compute the eigenvalues and eigenvectors of the RGB values of the image. Each of the three eigenvectors is a 3D vector. We then find a randomly chosen weight (uniform distribution between -0.1 and 0.1) for each eigenvector and calculate the weighted sum. The weighted sum is a 3D vector and is added to the RGB vector of each pixel. This color transformation aims to model small color variations that objects or logos may present in the real world caused by different lighting conditions.

Blurring And Noise Addition

In this step, we use Gaussian blurring with a variance randomly selected between 0.001 and 0.1 and kernel size of 3×3. Then, we randomly select a noise model between Gaussian, Salt & Pepper, Poisson, and Speckle noise. A small amount of noise is added to the image.

Gaussian noise is commonly generated by capture devices during image acquisition. In order to model Gaussian noise, we add different random RGB values to each pixel in the image. The random values are sampled from a random variable with normal density

function with mean 0 and a variance selected randomly between 1.2 and 2.4. The variance range is selected empirically to introduce a reasonable amount of noise.

Salt & Pepper can originate as analog-to-digital converter errors or transmissions errors. We model Salt & Pepper noise by changing the value of each pixel of the image with probability 0.03. The pixel will be changed either to white, (255, 255, 255) in RGB value, or to black, (0, 0 ,0) in RGB value, both cases with a probability of 0.015.

Poisson noise, or also known as shot noise, can be modeled by a Poisson process. In order to generate Poisson noise, a random variable is created for each pixel. This random variable has a Poisson distribution (Equation 2.2) with mean λ, where λ is equivalent to the value of the pixel. A random sample is extracted from every random variable. The sample is used to replace each original pixel. After this process, each pixel of the image has been replaced by a random value from a Poisson distribution. This process modifies the image in a non-linear way. Because the variance of the Poisson distribution is equal to its mean λ, the darker pixels will not suffer much change while the brighter pixels will have more variation. Finally, we make a weighted average with the original image and the distorted image with weights 0.8 and 0.2 respectively. This average weight aims to avoid images too distorted.

$$P(x) = \frac{e^{-\lambda}\lambda^{x}}{x!} \tag{2.2}$$

Speckle noise is a granular multiplicative noise. We generate Speckle noise by multiplying each pixel of the image by a random value. The random values are extracted from a random variable with normal density function with mean 1 and a variance of 0.2.

After blurring and noise addition, the noisy images are clipped to range between 0 to 255 before they are added to background images as presented in the next section.

Image Blending

Finally, the object or logo images are blended into the background in a random position ensuring that there is no overlap between various objects or logos. The blending process

consists of substituting the pixels of the background image with the pixels of the foreground image. More complex blending techniques, such as Poisson Image Editing [67], were discarded for simplicity and because they can produce undesired artifacts or distortions to the foreground image. Figure 2.15 shows examples of generated images. Two synthetic datasets are generated using the process described above. The first dataset contains 16,000 images with logos extracted from FlickrLogos-32 and the second one contains 25,000 images with 15 different toys.

Fig. 2.15. Examples of generated images using logos (left) and toys (right).

2.3.2 Experimental Results

We describe several experiments here where we train the Faster R-CNN using ZF [38] and VGG16 [39] models, presented in previous sections, with FlickrLogos-32 dataset and synthetic data. In all the experiments the Mean Average Precision (mAP) is computed using the Pascal VOC 2010 [35] procedure. mAP is defined later in this section. In the Pascal VOC 2010 procedure, each predicted bounding box is compared with all the ground truth bounding boxes of the same class for every image. If the Intersection over Union (IoU) overlap (Equation 2.3) between the predicted bounding box B_p and some ground truth bounding box B_{gt} is 50% or larger, the prediction is considered as a True Positive (TP), if not, is considered as a False Positive (FP).

$$IoUOverlap = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \tag{2.3}$$

To compute the mAP, the Precision (Equation 2.4) and Recall (Equation 2.5) are required [68, 69]. Precision is the ratio between the True Positives (TP) and the sum of True Positives (TP) and False Positives (FP). Recall is the ratio between True Positives (TP) and the number of ground truth bounding boxes (N_{bbox}).

$$Precision = \frac{TP}{TP + FP} \tag{2.4}$$

$$Recall = \frac{TP}{N_{bbox}} \tag{2.5}$$

For each class, a Precision/Recall curve is obtained by varying the threshold parameter from 0 to 1. The Average Precision (AP) is defined as the area under the curve. The previous process is repeated obtaining the AP for each class. The Mean Average Precision (mAP) is the average of the AP.

In the following experiments, we start the training process using pre-trained models with MS COCO for VGG16 and ImageNet for ZF.

Logo Recognition

In the first presented experiment, we train a Faster R-CNN with the VGG16 model. The network is trained using various combinations of synthetic images and FlickrLogos-32 images: using only synthetic images, using only FlickrLogos-32 images, and using both synthetic and FlickrLogos-32 images. The Faster R-CNN is trained for 100,000 iterations and we evaluate it every 10,000 iterations using the testing set from FlickrLogos-32. In Table 2.1 we present the best results for each combination of training data.

Our use of the Faster R-CNN instead of the Fast R-CNN appears to provide a significant improvement. The use of synthetic data together with the original data (FlickrLogos-32) provides an increase of 1.3% respect to only using original data. In Figure 2.16 we can see some examples of detected logos. The use of synthetic data without any original image has

Table 2.1.
Mean Average Precision (mAP) of different methods for logo recognition

Training data	FL32	Synthetic	FL32 + Synthetic	Previous Work [1]
mAP	84.11%	65.55%	85.40%	74.40%

poor performance. We believe this is caused by the loss of information of the background while synthesizing images (e.g. a Corona logo is more likely to be found in a bottle or a Starbucks logo is more likely to be found in a cup of coffee).

Fig. 2.16. Examples of logos detected in the wild

Object Recognition

While deploying applications using the HP Sprout the GPU memory can be a limiting factor. In this set of experiments, we focus on the previously described ZF model since it has a smaller size than VGG16 and can be used with the HP Sprout GPU. Since we only have synthetic data for toys, a small test dataset was manually labeled using LabelMe [70] for evaluation purposes. The dataset contains 35 labeled images containing up to 15 different toys. The images are captured using the HP Sprout camera with good lighting conditions. Figure 2.17 shows some examples of testing images.

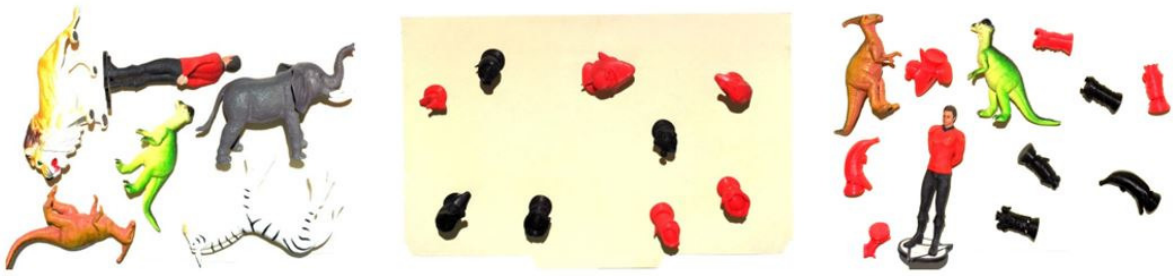

Fig. 2.17. Examples of toys testing set

First, we train the Faster R-CNN with VGG16 and ZF using the synthetic dataset containing objects (toys). We train several networks using a different number of images from synthetic toys dataset for each one: 25,000 images (100% of the synthetic dataset), 12,500 images (50% of the synthetic dataset), and 6,250 images (25% of the synthetic dataset). Following the previous experiment, the network is trained over 100,000 iterations and it is evaluated every 10,000 iterations. We present the best results using ZF and VGG16.

Note that the VGG16 model has a larger mAP than ZF. VGG16 is formed by more layers and it allows the network to learn more complex features. Using only 12,500 images in the training process seems to provide better performance. The use of a large number of images for training might cause some overfitting and therefore a decrease in performance. In Figure 2.18, some detection results are presented. Notice that the network is able to

Table 2.2.
Performance (mAP %) using different amount of synthetic data using 20 different poses

Model	6,250 images	12,500 images	25,000 images
VGG16	94.32%	97.42%	96.36%
ZF	92.91%	92.41%	93.41%

differentiate each of the individual red and black toys despite having minor differences between them.

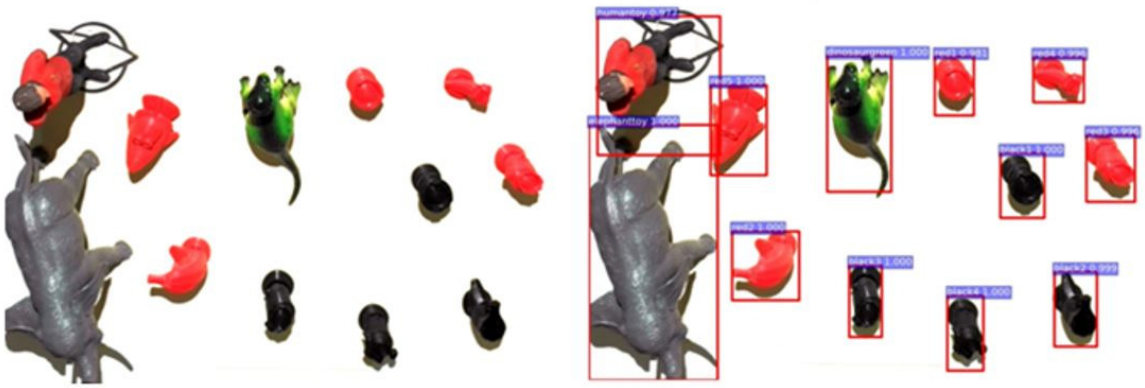

Fig. 2.18. Original images (left) and objects detected (right) using ZF model

In the last experiment, we analyze the effect of the number of images used in the synthetic generation process. We generate two extra toys datasets with 25,000 images using 10 and 5 clean images per class. We train ZF with the new datasets using a different number of synthetic images (100%, 50%, and 25% of the dataset) and we compare it with the original dataset made using 20 object images per class.

The results presented in Table 2.3 indicate that the number of original object images used for data synthesis is related to the amount of data required to train the network to achieve a good performance. If a low number of images is used in the synthesis process, the synthesized images will contain less information and fewer images will be required

Table 2.3.
Performance (mAP %) of ZF model using different amount of images for data synthesis and training

Num. poses per object	6,250 images	12,500 images	25,000 images
20	92.91%	92.41%	93.41%
10	87.54%	90.43%	88.08%
5	83.32%	80.83%	80.87%

in the training process. In the example of toys recognition, if more points of view (more images) are used for image synthesis, the synthetic images will contain more information and the average precision will increase.

2.4 SynthLogo Dataset

2.4.1 Dataset Generation

In this section, we present our second and more complex method for synthesizing images. Because our approach is based on the synthesis pipeline used in SynthText [18], we name our synthetic dataset as SynthLogo. The image synthesis process consists of estimating the depth and segmentation information of background images and blend logo images accordingly. Each step is described in the following sections.

Logo Images Acquisition

The image synthesis starts by obtaining multiple logo images. Our dataset contains a total of 604 different logos. These 604 logos include the ones that form FlickrLogos-32, FlickLogos-27, BelgaLogos and MICC-Logos, and some extra classes including logos from popular brands of food, drinks, clothing, technology, transportation, finance, etc. The images are obtained from Google Search. We search for PNG images with alpha layer so

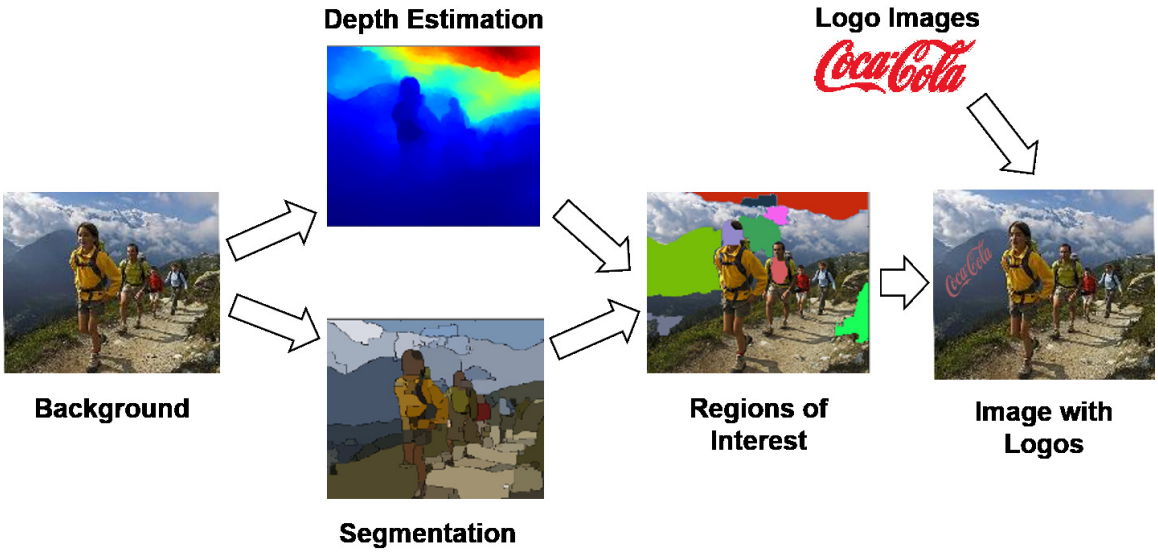

Fig. 2.19. Image synthesis pipeline.

we can separate the part of the image that corresponds to the logo from the background. Between 3 to 10 different images are obtained for each logo. We manually check them to remove outliers or undesired images. The same brand can have different versions of logos and some can contain both text and symbols. We do not make a distinction between logo versions or text or symbol part and we assign one unique class per brand. Figure 2.1 shows some example of logo images used.

Background Images

We use the same 8,000 background images as used in SynthText. The images are obtained from Google Search and manually checked so they do not contain text and they do not contain any logo from our corpus. Figure 2.20 shows some example of background images used.

Then, depth information is estimated, and the background image is segmented as explained in the following sections. Pre-computed values of depth estimation and background

segmentation are available together with the original background images and code [71]. We choose to use pre-computed values.

Fig. 2.20. Examples of background images in SynthLogo.

Depth Estimation

Depth information is inferred using a CNN as described in [72]. This method starts by dividing the image into superpixels. A superpixel is a set of neighboring pixels with similar RGB values. Then, 244 $\times$ 244 patches are cropped for each superpixel and processed by a CNN. The CNN contains 5 convolutional layers and 2 fully connected layers. This CNN estimates the depth value of the superpixel located in the center of the crop. A regularization factor is added to the loss function to accomplish pair-wise smoothness in the depth prediction between neighboring superpixels.

Background Segmentation

The background images are segmented by detecting the contours of the elements forming the image as described in [73]. The contours are locally computed using brightness, color, and texture gradients. Once the contours are detected, the Watershed algorithm is used to segment the image. Watershed is a method that clusters all the pixels located in a region defined by a contour. Then, small segments are merged using color similarities.

Image Blending

Next, a segment of a background image is selected. Small segments or segments with extreme aspect ratios are discarded. Then, a logo image is randomly selected and resized so the largest size is 0.9 times the largest side of the segment. Then, the logo image is randomly rotated with a probability of 0.3 with a degree randomly selected between -90, 180 or 90 degrees. A small color jittering is randomly applied with a probability of 0.5 in the HSV color space. A total of 3 random values are selected uniformly from -10 to 10 and added to the hue, saturation, and value channels respectively. Color jittering aims to add small color variations that logos may present in the real world caused by different lighting conditions.

Then the logo is geometrically transformed using Random Sample Consensus (RANSAC) [74]. RANSAC is a method that allows us to match and project a planar surface using matching points. A planar surface is estimated within the selected segment of the background image using the depth information and a homographic projection is estimated so the plane of the segment and the plane of the logo match.

Finally, the logo image is blended into the background. The blending is done by combining the pixels of the background image with the pixels of the transformed logo image. An alpha value, α, is selected randomly between 0.5 and 1, and alpha blending is performed as specified in equation 2.6. $I_{synthetic}$ is the generated image, I_{logo} is the logo image and $I_{background}$ is the background image.

$$I_{synthetic} = \alpha I_{logo} + (1 - \alpha) I_{background} \tag{2.6}$$

The complete process is repeated 1 to 5 times per synthetic image, therefore an image will include one or more logos. A binary mask is used to check that there is no collision between logos.

SynthLogo

Using the process previously described, we create a total of 280,000 images. Each of the 8,000 background images is used several times. In Figure 2.21 some examples of synthetic images are presented.

This process allows us to create an arbitrary number of images and new logos can be easily added to the corpus by simply obtaining images from any image search engine. In table 2.5 we compare our dataset with other existing datasets. One important aspect is how easy it is to augment both the number of images and the number of logos. In the table, we mark as ”Scalable” the datasets that can add more logos and images with minor or none manual labeling efforts.

2.4.2 Experimental Results

We train Faster R-CNN+VGG16 [39] and PVANet [19] models, previously described, with the SynthLogo dataset. We start the training process using pre-trained models with MS COCO. FlickrLogos-32 is used as validation and testing set. We compute the Mean Average Precision (mAP), previously described, using the Pascal VOC 2010 [35] method. As previously stated, in this evaluation method, each ground truth bounding box is compared with all predicted bounding boxes. If predicted and true classes match, and the Intersection over Union (IoU) (Equation 2.3) between the predicted bounding box B_p and the ground truth bounding box B_{gt} is larger than 50%, the prediction is a True Positive (TP), otherwise is a False Positive (FP). Remember that the last layer of Faster R-CNN and PVANet provides a probability or confidence measure between 0 and 1. A prediction is discarded (i.e.

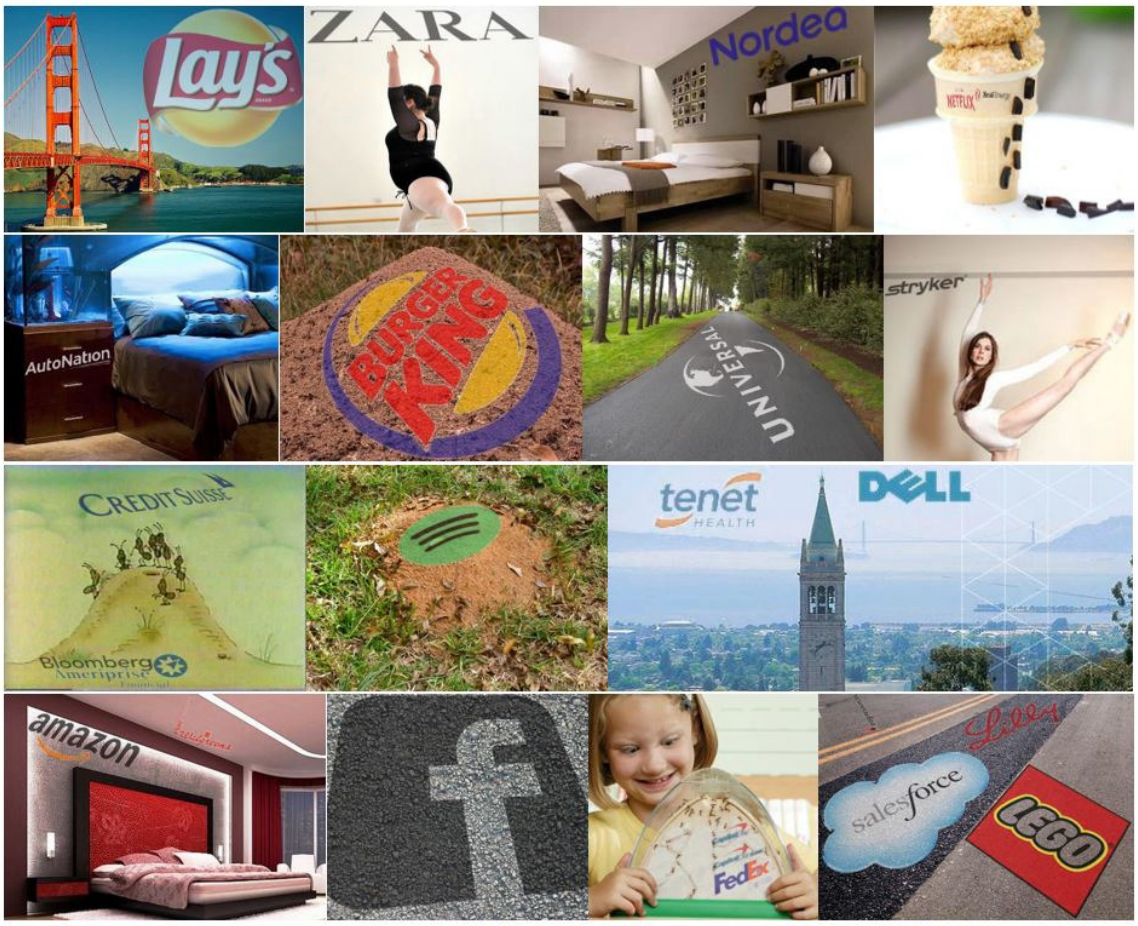

Fig. 2.21. Example of different synthetic images.

Fig. 2.22. Example of different versions of the same logo in SynthLogo.

consider as a background) if its confidence score is lower than a threshold and maintained otherwise. For each class, a Precision/Recall curve is obtained by varying the threshold parameter from 0 to 1. The Average Precision (AP) is the area under the curve. We average the AP value of each class obtaining the Mean Average Precision (mAP).

We train a Faster R-CNN+VGG16 and PVANet for 200,000 iterations and evaluate the model every 10,000 iterations with the training set of FlickrLogos-32. Then, we select the best set of weights for each model and evaluate on the testing set of FlickrLogos-32. Table 2.6 presents the best results for validation and testing of both models. We can observe that the mAP is lower than results reported in previous works where manually labeled images have been used [14,56,57] but in our work the manual effort is minimum and the number of logos can be easily increased. Some works present similar results when using an open set of logos (arbitrarily large number of logos) and suggest using Faster R-CNN as an initial stage of localization and then include CNN for classification [57]. A similar approach as ours is presented in [11] where a synthetic dataset of 463 different logos is created and then used to train Faster R-CNN obtaining a 25.0% mAP when only using synthetic data.

Table 2.4.
Mean Average Precision (mAP) of logo detection methods trained with SynthLogo and tested on FlickrLogos-32.

Model	train+val set	testing set
Faster R-CNN+VGG16	49.46%	47.66%
PVANet	40.44%	38.56%

Table 2.5.
Different datasets containing logo images.

Dataset	Number of brands	Number of annotated images	Type	Publicly Available	Scalable
FlickrLogos-32(47) [9]	32	2,240	Real	Yes	No
Logos-32plus [56]	32	7,830	Real	Yes	No
TopLogo10 [11]	10	700	Real	Yes	No
FlickrLogos-27 [54]	27	810	Real	Yes	No
MICC-Logos [55]	13	720	Real	Yes	No
BelgaLogos [10]	37	1,321	Real	Yes	No
FlickrBelgaLogos [59]	37	2,697	Synthetic	Yes	Yes
LOGO-Net [48]	160	73,414	Real	No	No
WebLogo-2M [20]	194	2,190,757	Bootstrap	Yes	Yes
Logos in the Wild [57]	871	11,054	Real	Yes	No
[14]	32	16,000	Synthetic	No	Yes
SynthLogo	604	280,000	Synthetic	No	Yes

2.5 Two-Step Logo Detection With Bootstrapping

2.5.1 Proposed Approach

In this section, we present our method for bootstrapping for logo detection. The presented bootstrapping process starts by obtaining a large number of unlabeled images from the web. Then Faster R-CNN [8], described in previous sections, is used to automatically detect logos within the images. Faster R-CNN is trained with the synthetic images from SynthLogo dataset [49] described in the previous section. The detected images are cropped, resized, and used to train the DenseNet [21] classification network, described later in this section. During testing, a two-stage process is done. First, the Faster R-CNN generates region proposals, and then, each region is classified with the DenseNet classifier.

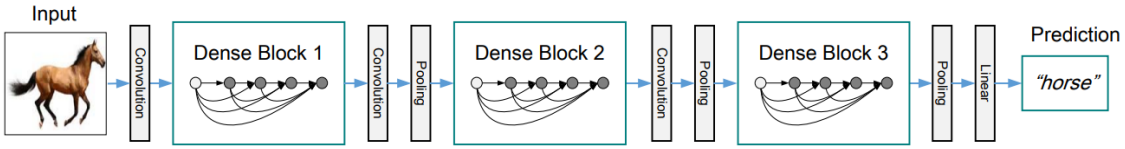

Fig. 2.23. DenseNet architecture. Figure adapted from [21].

Our Datasets

In our work, we use two different sets of training images. The synthetic images from the SynthLogo dataset are used to train both Faster R-CNN and DenseNet. The images automatically obtained from the web are used to train DenseNet.

SynthLogo, described in the previous section, includes a total of 280,000 images with 719,326 logo instances. Figure 2.22 shows some examples. During the bootstrapping process, a total of 480,000 images are automatically downloaded from the web. These images have been obtained by using search queries with the form of *logo name* + *keyword*. We select multiple keywords that define images that might contain logos. The keywords

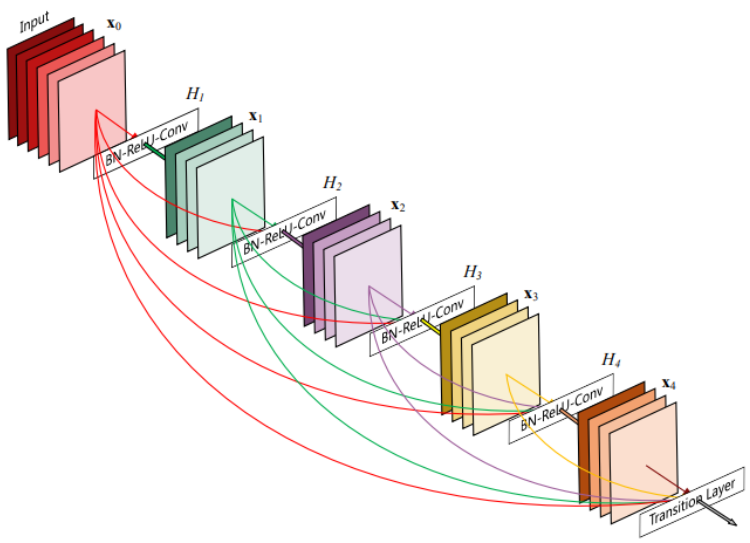

Fig. 2.24. Dense block of DenseNet. Figure adapted from [21].

include: *advertisement, billboard, commercial, merchandising,* and *building*. The number of images containing logos varies depending on the logo and the keyword used in the search query. For well-known trademarks, roughly between 70 to 80% of the first 100 search results contain one or more logos. When searching images for little known brands, the number of resulting images including logos can be small and the bootstrapping process might not be effective. In our work, we use the same 604 logos used in the SynthLogo dataset.

Two-Stage Approach

The two-stage approach of combining a region proposal and a classifier is based on the Region-Based Convolutional Neural Network (R-CNN) [15] method. As previously described, R-CNN is an architecture that crops and resizes regions of interest and classifies them with a CNN. The original work uses Selective Search [40] as a region proposal method. Selective Search computes similarity measures based on visual features to propose regions of interest. The original work uses a CNN with five convolutional layers and two

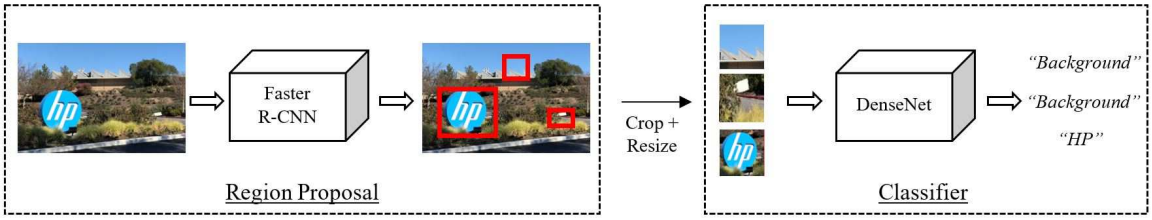

Fig. 2.25. Proposed logo detection pipeline.

fully connected layers. In this work we use Faster R-CNN [8] as logo detector and region proposal method and DenseNet [21] as a classifier. Figure 2.25 presents our two-stage method.

Many end-to-end object detectors such as YOLO [43] and YOLOv2 [44] fail to correctly detect small objects. These methods resize the input image losing information and leading to bad detection results [52]. Other methods such as Faster R-CNN [8] locates and classifies logos using features of the last convolutional layers. By doing so, small objects can be incorrectly classified. Our pipeline aims to solve these problems by cropping and classifying regions from the original image instead of using features of the last convolutional layers. The main downside of the presented two-stage process versus an end-to-end method is that it is computationally less efficient.

Region Proposal And Logo Detection

We train the Faster R-CNN with the SynthLogo dataset [49]. Faster R-CNN is used as an end-to-end logo detector during bootstrapping and as a region proposal method during testing. During bootstrapping, we assume that a correct detection occurs when a logo that matches the search query is detected with a confidence score higher than a threshold. This assumption allows us to automatically label images while tolerating some incorrectly labeled detections. The threshold used to discard detections allows us to select a trade-off between the number of detections and the noise in the labels. The smaller the threshold, the larger the number of detections, and the larger the noise in the labels. We manually

inspected the results of detections and their confidence scores in over 500 images obtained from the web. We select a threshold of 0.4 because around 80% of correct detections have a confidence score equal to or higher than 0.4. During testing, we select all detections with a confidence score higher than 0.1 as region proposals. We select a low threshold in order to obtain a high recall. The DenseNet classifier will later discard the regions of interest that do not contain any logo by classifying them as *"background."* Figure 2.26 shows some examples of detections of Faster R-CNN.

Fig. 2.26. Examples of logo detections.

Logo Classification

DenseNet [21] is used to classify each region of interest generated by Faster R-CNN. DenseNet is an architecture formed by multiple dense blocks. A dense block consists of a set of convolutional layers where the feature maps of all previous layers are treated as separate inputs and its feature maps are passed on as inputs to all following layers. This connectivity provides a better information flow and better gradient propagation providing higher accuracy than previous architectures [21]. The network used in this work is formed by 4 dense blocks with 6, 12, 24, and 16 filters respectively. The network ends with a fully-connected layer.

Faster R-CNN can process images of any size and will output regions of interest of multiple sizes and scales. DenseNet requires an input of size 224 $\times$ 224. Therefore, each region detected by Faster R-CNN is cropped and resized before classification. Typically, a crop of 224 $\times$ 224 pixels contains enough information to correctly classify a logo.

2.5.2 Experimental Results

We start our experiments by training Faster R-CNN using the SynthLogo dataset. Then, we use Faster R-CNN to detect logos in the 480,000 images obtained from the web. A total of 153,714 logos are found in 128,249 images. We crop and resize each detected logo.

We train two DenseNet models that are capable of classifying 33 (32 logos + background) and 605 (604 logos + background) classes respectively. The first model is capable of classifying the 32 logos from the FL-32 dataset. The second model is capable of classifying the 604 logos from the SynthLogo dataset. DenseNet is trained using 700,000 crops extracted from SynthLogo and 153,714 crops from the 128,249 bootstrapping images. A total of 30,000 random crops from the background images in SynthLogo are used as *"background"* class. For validation purposes, we use 19,326 crops of the SynthLogo dataset. The FL-32 dataset is used for testing purposes. The testing set of FL-32 consists of 960 images. Note that in the model of 604 logos, only 32 of them are being evaluated. We assume that the performance of the remaining 572 logos will be similar. Data augmentation techniques available in the PyTorch framework [75] are applied to the crops during training. This includes random HSV jittering, random grayscale transform, random flipping, and random crop and resize.

During testing, we observe that the region proposals of Faster R-CNN include all the logos of the FL-32 testing set. However, the network suggests several undesired regions that the classifier needs to discard. We compute the mean average precision (mAP) value of the overall system using the Pascal VOC [35] procedure. This value is equivalent to the area under the precision/recall curve. As previously described, Recall is the ratio between True Positive (TP) and the total number of ground truth bounding boxes (N_{gt}). Precision

is the ratio between TP and the number of detections. A true positive is a detection where the Intersection over Union (IoU) between the predicted bounding box (B_p) and the ground truth bounding box (B_{gt}) is larger than 50%.

Table 2.6 presents the mAP results. For the model trained to detect 604 logos, our previous experiments presented in [49] show that using Faster R-CNN trained with only with synthetic images doesnt perform well, obtaining only a 47.66% mAP. By using Faster R-CNN as an initial stage followed by DenseNet, we are able to increase the mAP to 73.96%. With the boosting method, we obtain a mAP of 76.78%, an increase of an additional 2.82% over two-stage detection and classification. When using bootstrapping images, the mAP is higher than when only using synthetic images. Bootstrap images, despite having noisy labels, are extracted from real scenes and contain real-world deformations that synthetic images are not able to capture. For the model trained to detect 32 logos, our method obtains a maximum of 80.12% of mAP. When detecting more classes, the mAP decreases because similar logos can get misclassified. Some other methods report higher mAP values (e.g. [11]) by using a large amount of manually labeled images from real scenes.

Table 2.6.
mAP results using synthetic and bootstrap images.

Method	Training Data	32 Logos	604 Logos
[49]	SynthLogo	-	47.66%
Two-Step (ours)	SynthLogo	74.85%	73.96%
Two-Step (ours)	Bootstrap	77.54%	75.12%
Two-Step (ours)	SynthLogo + Bootstrap	80.12%	76.89%

2.6 Head Detection For Sleep Analysis

2.6.1 Videosomnography

In this section, we show another example of an application that benefits from object detection methods: Videosomnography for pediatric sleep medicine. Pediatric sleep medicine is a science that studies sleep patterns of children in order to detect typical and atypical behaviors. The analysis of sleep patterns is an important aspect of medicine as abnormal sleep patterns can be an indicator that some diseases or clinical conditions might be present. In order to analyze the sleep patterns, physicians and researchers record and annotate (also referred as "code") the sleep pattern of children and analyze the sleep onset time, the total sleep duration, and the presence or absence of night awakenings. This has been commonly approached with videosomngraphy (VSG) techniques that analyze the sleep patterns from recorded videos [76, 77]. Traditional manual behavioral videosomnography (B-VSG) coding consist of the manual coding of awake and sleep states by a trained technician (also referred to as coder). However, B-VSG is highly time-consuming and requires extensive training on the side of the technicians. In this section, we present an automated VSG method (auto-VSG) that provides physicians and researchers with an automatic sleep recording tool that is more time-efficient while maintaining high levels of coding precision.

2.6.2 Auto-VSG

The proposed auto-VSG system uses the motion of the child within the video to estimate the awake/sleep state. Analyzing the motion as a proxy of the sleep state has already been used in previous works. Motion can be estimated by frame differencing [78, 79] or with motion vectors [76, 80]. However, most of the presented methods are performed in a controlled setting and fail when faced with different camera positions and lighting conditions. The auto-VSG method aims to provide robustness to the wide range of video conditions. The presented auto-VSG method includes (1) a pre-processing step consisting of a histogram equalization and an image resizing, (2) detecting the movements of the

child using background subtraction, (3) detecting and estimating the size of the child with an object detection CNN, and (4) normalizing the motion with respect of the size of the child's head in the frame. A label of awake or sleep is then assigned at each minute of the video. Figure 2.27 shows the proposed system.

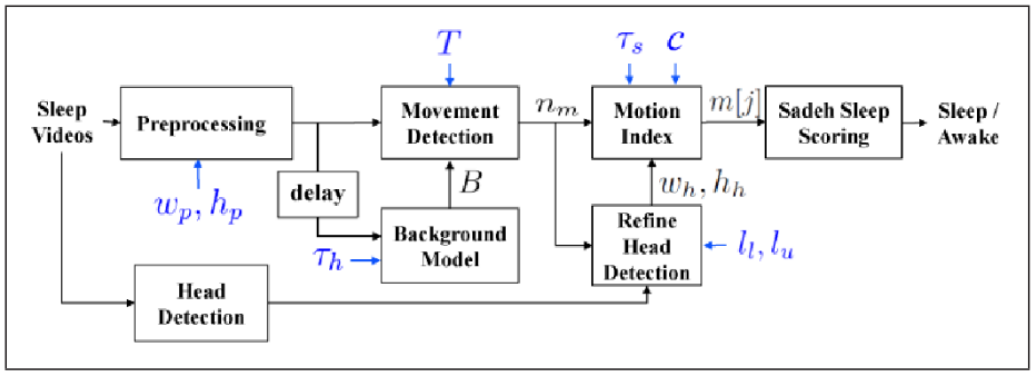

Fig. 2.27. Auto-VSG Detection System. Figure adapted from [81].

Motion Detection

We assume that during a typical sleep pattern there will be less motion during sleep than awake states. Additionally, we assume that the camera is static and the only source of motion in the video will be the children. The frames from the video are transformed to grayscale and resized to 160 $\times$ 120 pixels. Then, histogram equalization is applied in order to enhance grayscale contrasts. Next, the presented system detects the motion of the children by computing the difference between frames. Specifically, the difference between a background model (average of previous frames) and the current frame is computed. The background model is computed as in equation 2.7.

$$B_i[x, y] = \frac{1}{h} \sum_{k=i-h}^{i-1} I_k[x, y] \tag{2.7}$$

Where i is the frame index, h is the number of previous frames used to compute the background model, $I_i[x, y]$ is the ith frame and $B_i[x, y]$ is the ith background model frame.

The model frame can be seen as a moving average of the previous frames. At each frame, each pixel is considered to differ from the background model frame if their difference is higher than a threshold as specified in:

$$|B_i[x, y] - I_k[x, y]| > T \tag{2.8}$$

where T is a threshold empirically selected. In this work we select $T = 30$. Then, the number of pixels n_m, where their difference is higher than the threshold T, is averaged among h:

$$m_i = \frac{1}{h} \sum_{k=0}^{h-1} n_m[k] \tag{2.9}$$

Where $n_m[k]$ is the number of pixels "moved" within frame k. The number of moved pixels at instant i (temporal instant composed by h frames), m_i can indicate if the baby is moving and therefore if the baby is asleep or awake. However, this motion measure has the drawback that is dependent on the distance between the child and the camera. When the baby is closer to the camera, the number of moving pixels will increase, while when it is further away, the number of moving pixels will decrease. In order to address this mismatch, we detect the head of the child in order to normalize the movement with the size of the detected head. Figure 2.28 shows an example of the motion detection.

Fig. 2.28. From left to right: Input image, background model image, and moved pixels. Figure adapted from [81].

Head Detection

In order to detect the head we use Faster R-CNN, previously described. In this scenario, we select the Zeiler and Fergus (ZF) [38] network as a feature extractor because it has a small number of parameters (only 5 convolutional layers) and can lead to better generalization when the training dataset is small. The RPN uses the information provided by the features extracted from ZF to detect regions of the image where the head of a child might be located. Then, the features within those regions are cropped and used by the classifier that outputs a confidence value. Confidence of 1 represents that the network is almost certain that the region contains a head while confidence close to 0 represents that the network identifies that region of the image as a background.

We train Faster R-CNN with the Casablanca dataset [82] which consists of a total of 1466 grayscale images containing multiple heads in different lighting conditions and poses. Each image has a corresponding bounding box indicating where the head is located. Figure 2.29 shows an example of two heads detected with Faster R-CNN.

Fig. 2.29. Examples of head detections of two different childs. Figure adapted from [81].

Sleep Scoring

After detecting heads within all frames with Faster R-CNN, the detected region with the highest confidence score during the window of h frames is selected. The size of the bounding box is used to estimate the number of pixels of the head N_{max}:

$$N_{max} = W \times H \tag{2.10}$$

where W and H are the width and height of the detected bounding box in the resized image. The number of moving pixels is normalized and scaled by a constant in order to match the scale of the Sadeh Sleep Scoring method, which is commonly used for scoring the Actigraphy motion index [83]. Therefore, the estimated final score that is used as a proxy to indicate if the individual is sleeping or awake is:

$$m[j] = 400\frac{min(m_j, N_{max})}{N_{max}} \tag{2.11}$$

where $m[j]$ is the motion index at the jth time segment.

3. POSE ESTIMATION AND AUGMENTED REALITY

3.1 Overview

The main challenge when using technologies such as augmented reality, robot manipulation, or autonomous driving is to obtain a clear and detailed understanding of the geometry of the objects appearing in the image or video to properly interact with them. For example, in order to perform robot manipulation, the location of the objects in the physical world needs to be estimated to accurately grab, move, and manipulate the elements of the scene. In autonomous driving applications, the position of other vehicles and pedestrians needs to be inferred in order to safely move or stop the autonomous vehicle. To properly understand how the objects in the scene are placed, the pose and location of each element need to be accurately inferred. 6 Degree of Freedom (DoF) or 6D pose estimation techniques can be useful for such tasks. 6D pose estimation consists of inferring the 3D location coordinates and 3D rotation angles of objects in the real world from images, videos, or depth information. Figure 3.1 represents how pose estimation can be performed with a neural network.

Many solutions have been presented for 6D pose estimation working with both RGB-D (color and depth) and RGB (only color) images. However, RGB-D cameras are not highly available (e.g. in smart-phones or laptops) and currently have many limitations of depth range, resolution, and frame rate making it difficult to detect small, thin, or fast-moving objects. On the other hand, methods that use RGB-only images can easily fail to correctly estimate the pose of objects with different lighting conditions, occlusions, or objects that lack texture or distinctive visual features. In this chapter, we will focus on RGB-only techniques as they can be applied in a wider range of scenarios and is a challenging problem that still remains unsolved.

Traditional computer vision methods estimate the 6D pose by matching visual features from the image to a 3D model of the object of interest [22]. These methods fail to correctly estimate the pose of textureless objects since only a small number of visual features can be detected in the object. Recently, many methods based on deep learning have been presented. These methods use neural networks to estimate the 6D pose of objects by de-

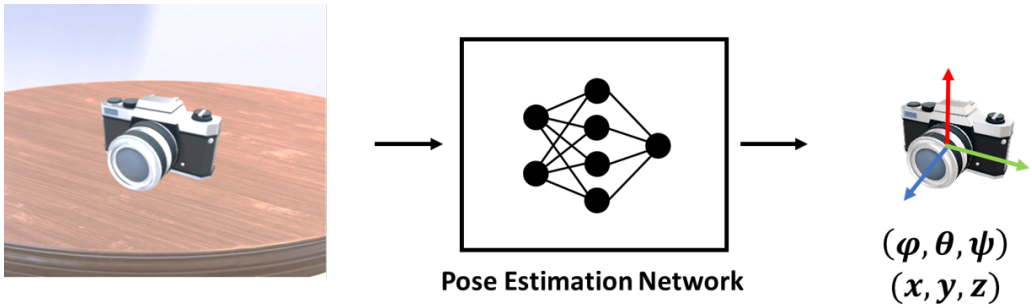

Fig. 3.1. Pose estimation task with a neural network.

tecting keypoints [84], estimating a 3 dimensional bounding box [85–87], matching the input image with rendered images [88, 89], or directly treating pose estimation as a classification [90] or regression [91, 92] problem. Textureless objects can be handled by using methods that try to directly estimate 3D coordinates from pixels or pre-defined keypoints of the objects of interest [93]. These methods typically adapt or extend object classification or detection methods to solve pose estimation. Although they can handle textureless objects, they are not highly accurate as typically the pose needs to be discretized into bins or the methods rely on learning the visual appearance of the objects at different poses.

Recently, many methods of pose refinement have been presented showing considerable improvements in accuracy. A common approach is to render an RGB image with an initial pose estimate of the target object and then match the rendered image with the input (real) image to obtain a new and more accurate pose estimate. This approach has been used using hand-crafted visual features [22] and deep learning methods [88,92]. One of these methods is DeepIM [88], a neural network that can successfully learn the SE(3) transformation of an object in two different poses even with objects from previously unseen categories.

With the increasing number of deep learning-based methods for pose estimation (mainly based on RGB-only input images), there is a need for more data to train the neural networks. Because capturing and annotating real images is highly tedious, inaccurate, and time-consuming, automated approaches are preferred. While some image synthesis methods have been presented [14, 49] to automatically generate new image samples to train

Fig. 3.2. Six rendered views of an object.

and evaluate neural networks, the synthesized images usually lack realistic appearance. A growing popular alternative, highly efficient and effective, is photorealistic image rendering. Photorealistic rendering permits to easily generate a large number of images with ground truth labels. Such images include realistic lighting, occlusions, and real-world distortions. This set of techniques can be especially useful in domains where training data is scarce or non-existent.

In this chapter, we introduce a new set of neural networks: Multi-View Matching Network (MV-Net) and Single View Matching Network (SV-Net) for 6D pose estimation, refinement, and tracking. These networks are based on DeepIM [88], a neural network that performs pose refinement by estimating the pose difference between pairs of images. MV-Net matches the input image with 6 RGB images containing the object of interest rendered from different views (Figure 3.2) to obtain an initial estimate of the pose. Then, SV-Net refines the pose estimate in an iterative manner. The same iterative refinement can be used to track the pose in a video sequence.

The main contributions of this work are as follows. First, we present the Multi-View Matching Network and the Single View Matching network, a novel extension of DeepIM, for pose estimation, refinement, and tracking, removing the need for an external initial pose estimation method. Second, we show how these networks can be combined with

Mask R-CNN to have a complete object detection and pose estimation pipeline to support AR applications. We evaluate and compare our method with previous pose estimation methods.

The rest of the chapter is organized as follows. In section 3.2 an overview of the tasks of pose estimation and an overview of the most commonly used datasets is presented. In section 3.3 Multi-View Matching Network and Single View Matching Network are described in detail. In section 3.4 the Extra FAT dataset is described.

3.2 Related Work

Many RGB-D based methods have been previously presented. A common approach is to use depth information to obtain a 3D point cloud that is then matched against a 3D model [94]. The methods presented in [94] regresses the 3D coordinates of each pixel in the input image. Then, the estimated coordinates are matched to the 3D model to obtain the 6D pose. Additionally, the Iterative Closest Point (ICP) method is used for pose refinement.

Traditional computer vision methods estimate the 6D pose of an object by matching visual features (e.g. [22]) from the image to a 3D model of the object. The main drawback of this approach is that it fails with textureless objects since only a small number of visual features can be detected on the image. Many approaches based on deep learning have been recently proposed. These methods typically treat pose estimation as a pose classification or pose regression task by extending object classification or detection networks [90–93]. Deep-6DPose [92] is a network that extends Mask R-CNN with a head that consists of a regressor that estimates a 4D quaternion encoding the object rotation. The network 3D-RCNN [90] has 4 different heads: the first one estimates the bounding box containing the object, the second one estimates the coordinate of the center of the object, the third one estimates the 3 rotation angles and the fourth one a vector that defines the shape of the object. DensePose [93] is a network that extends Mask R-CNN by adding an extra head that estimates a dense mapping from the object (in this work objects are human bodies) to a 3D surface. The method presented in [84] detects key points using neural networks and

estimates the pose by solving Perspective-n-Point (PnP). The methods presented in [86,87] make use of neural networks to regress the 3-dimensional bounding box containing the target object. The methods in [90,92] regress directly the rotation angles or quaternions and translation parameters. Some multi-view methods have been used in pose related problems. The method presented in [95] uses a multi-view convolutional neural network for shape estimation. The method proposed in [96] uses a multi-view multi-class system for pose estimation.

Recently, methods for pose refinement have been presented showing considerable improvements in accuracy [88, 97]. Such methods are typically combined with pose estimation approaches: a pose estimation method provides an initial estimate that is later refined with a pose refinement technique. A common approach is to render an RGB image with an initial pose estimate and then match the rendered image with the input image to obtain an updated pose estimate. The method presented in [98] uses a neural network to perform the pose refinement based on the object contours. DeepIM [88], the basis of our work, is capable to provide highly accurate results by applying iterative pose refinement. The regression of relative rotation and translation parameters was already studied in [97].

Deep Iterative Matching [88], or DeepIM, is a neural network for pose refinement. The network is based on FlowNetSimple (FlowNetS) [99], presented in figure 3.3. FlowNetS is a neural net that is able to estimate optical flow between two consecutive frames. In DeepIM, FlowNetS is extended to estimate the pose difference between the two images. Specifically, two fully-connected layers of dimension 256 are appended after the 10th convolutional layer of FlowNetS. Then, two linear regressors (Linear layers) are added to the FC layers to estimate the relative rotation and translation parameters respectively. Some assumptions are made:

- Both images contain the same object in different poses
- A segmentation mask is available for both images
- The pose difference between the two images is small
- The 3D model of the object to be detected is known

The key element of the DeepIM network is its iterative nature. The method starts from an initial pose estimate that is provided by some other method (e.g. [91]). With an initial pose estimate, an image is rendered with the 3D model of the object. The inputs of the network are the rendered image and the target (real) image. The network estimates the pose difference between real and rendered images. Then, the pose difference is used to update the previous pose estimate, obtaining a more accurate pose prediction. The updated pose is used to render a new image and the process is repeated, usually for a fixed number of iterations (e.g. 4 iterations).

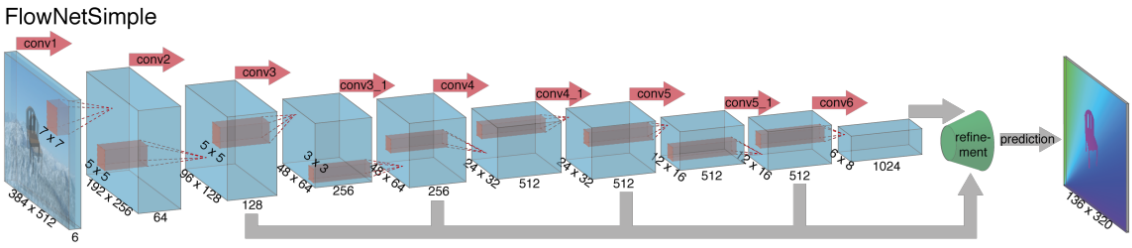

Fig. 3.3. FlowNetS architecture. Figure adapted from [99].

Many datasets consisting of real-world images for 6D pose estimation are publicly available. For example, T-LESS [100] is a dataset that contains a total of 48,900 images with 30 industrial objects which are textureless (mainly plain white or gray), and many are symmetric and highly similar among them. The YCB dataset [101] is composed of 9,240 images, captured using the Google scanner and the BigBIRD Object Scanning Rig, of 77 different common objects that are typically used for benchmarking robotic hand grasping and manipulation tasks. Figure 3.4 shows rendered objects included in the YCB dataset. The YCB-Video [91] dataset has around 134,000 video frames including a subset of 21 household objects from the YCB dataset. The Rutgers APC [102] dataset contains real images of textured products used in the first Amazon Picking Challenge. The IC-MI dataset [103] contains images of 6 objects placed in a cluttered and occluded setting. The LINEMOD dataset [104], composed of various toys and household objects, is a widely used public dataset for 6D pose estimation benchmarking. Figure 3.5 shows rendered ob-

jects included in the LINEMOD dataset. The LINEMOD OCCLUSION dataset [105, 106] is a complementary dataset of LINEMOD dataset with around 10,000 images captured under different occlusion and lighting conditions. MVTec Industrial 3D Object Detection Dataset (MVTec ITODD) [107] includes 28 industrial objects and focuses on challenging tasks such as 3D object inspection and industrial bin picking. Additionally, datasets with photorealistic rendered images, such as Falling Things (FAT) [108], have been recently presented. The FAT dataset contains synthetic images containing the 21 household object models from YCB dataset. The objects are placed in different virtual scenarios with different poses and lighting conditions.

Fig. 3.4. 3D object models in YCB dataset [101]. Figure adapted from [109].

Fig. 3.5. 3D object models in LINEMOD dataset [104]. Figure adapted from [109].

The BOP benchmark [110] was presented to unify a large variety of datasets and evaluation metrics and solve the lack of a common benchmark procedure. The BOP includes a total of 8 different datasets containing images and evaluation methodologies. Additionally,

two new datasets, the TUD Light [110] dataset and TOYOTA Light [110], datasets are introduced in the BOP dataset. The TUD Light dataset includes images of 3 objects without occlusion under different illumination. The TOYOTA Light dataset has a total of 21 different objects. The objects in TOYOTA Light are placed on top of a table with different tablecloths and 5 different lighting conditions. Figure 3.6 shows rendered objects included in the TYO-L [110], TUD-L [110], IC-MI [103], RU-APC [102], and T-LESS [100] datasets.

Fig. 3.6. 3D object models in TYO-L [110], TUD-L [110], IC-MI [103], RU-APC [102], and T-LESS [100] datasets. Figure adapted from [109].

3.3 Multi-View Matching Network

3.3.1 Proposed Method

Our proposed method has multiple stages. First, we use Mask R-CNN [17] to detect and segment the objects of interest in the input image. Next, we estimate the 6D pose of the

objects with the Multi-View Matching Network. Then, the estimated pose is refined in an iterative manner with Single View Matching Network, in a similar fashion than DeepIM. When dealing with videos, the pose of the object can be tracked with the Single View Matching Network. Figure 3.8 shows the MV-Net and SV-Net architectures. Figure 3.7 shows the complete pose estimation, refinement, and tracking pipeline.

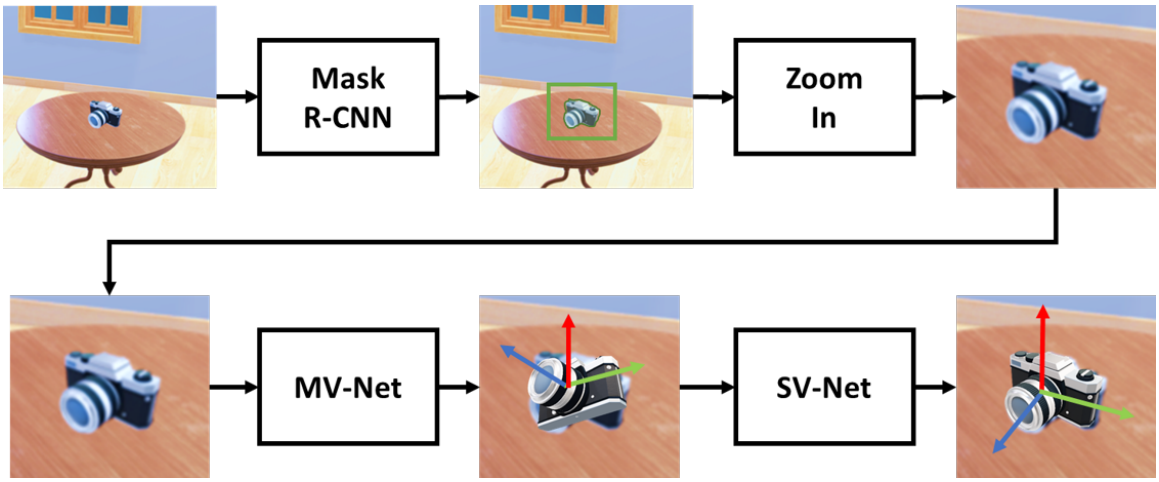

Fig. 3.7. Proposed pose estimation pipeline.

Object Detection And Segmentation

Many deep learning-based methods for object detection and segmentation are currently available [16, 17, 41, 43, 44]. Due to the modular nature of our system, we could combine the Multi-View Matching Network and Single View Matching Network with any object detection and segmentation method. We decide to use Mask R-CNN in our system as it provides state-of-the-art results on detecting and segmenting common objects.

Mask R-CNN, described in the previous chapter, is a network capable to estimate the bounding box and segmentation mask of multiple objects within an image. This network uses a Feature Pyramid Network (FPN) [47] to obtain multiple regions of interest at different scales where the objects might be located. The features inside the detected regions of interest are used to assign a category to the object and estimate its segmentation mask. In

our work we use a Mask R-CNN with a ResNet-50 [46] as a feature extractor (also referred to as backbone).

We use the estimated bounding boxes to apply a zoom-in operation to the regions where the objects have been detected. The zoom-in operation is performed by cropping and resizing the region inside the bounding box containing the object. The cropped region is expanded until it has a 4:3 ratio and it is resized so the final image has 640×480 pixels. The same zoom-in operation is performed to the estimated segmentation mask.

Initial Pose Estimation

Our pose estimation method uses two neural networks that share almost all weights, so it can be regarded as a unique two-in-one neural network. One network is trained to estimate the initial pose of the object and the other is trained to refine and track the pose estimate.

After detecting and segmenting the object with Mask R-CNN, the initial pose estimation is obtained using the network we name Multi-View Matching Network (MV-Net). This network takes as input 6 rendered images with the object in 6 different poses and the zoomed target image. Then, the Single View Matching Network (SV-Net) is used for pose refinement. This network takes as input the zoomed target image and a rendered image containing the object with the previously estimated pose. MV-Net and SV-Net are represented in figure 3.8.

Both networks are trained to estimate the relative SE(3) transformation between pairs of images containing the same object in different poses. These neural networks are composed of a set of convolutional layers, followed by several fully-connected layers, or heads. Following the work presented in [88], we use the first 10 convolutional layers of FlowNet-Simple (FlowNetS) as a starting point. Different heads are used in Multi-View Matching Network and Single View Matching Network.

Multi-View Matching Network (MV-Net) takes as input 6 rendered images with the object in 6 different poses and the zoomed target image (detected with Mask R-CNN) to

estimate the initial pose of the object. MV-Net is composed of 6 parallel branches, each of them consisting of the first 10 convolutional layers of FlowNetSimple network followed by one fully-connected layer of dimension 256. The outputs of the fully-connected layers of each 6 branches are concatenated and followed by one fully-connected layer of dimension 512. Finally, two fully-connected output layers of dimensions 4 and 3 are added to estimate the relative rotation and translation parameters respectively. The weights of every layer in the 6 branches are shared among them and with SV-Net.

Each branch has as input an 8 channel tensor that consists of the RGB zoomed target image, its segmentation mask (obtained from Mask R-CNN), and an RGB rendered image with its mask. This 8 channel input approach is the same as used in [88]. The rendered image used at each branch has a different pose. In our method, we select the 6 equidistant angles in the pitch and yaw dimensions, equivalent to the 6 views of all faces of a cube. Figure 3.2 shows the rendered images of the 6 views of an object. The same approach could be applied for a larger number of views as long as the network architecture is adapted properly and the views used are consistent during training and testing time.

The estimated rotation and translation are represented as the relative pose of the target image and the object placed with its frontal face facing the camera (the input of the 1st branch of MV-Net). The initial location of the object is inferred from the center of the bounding box. The network learns the relative pose with the same representation as in [88], where the relative rotation is expressed with a quaternion and the relative translation is expressed with an untangled representation independent from the coordinate system of the object. Both quaternion and untangled translation parameters are normalized to have zero mean and unit variance. Such representation is described in more detail later in this chapter.

Pose Refinement

Single View Matching Network (SV-Net) refines the initial pose estimate from MV-Net. SV-Net consists of one single branch from MV-Net (composed by the 10 first convolutional layers of FlowNetS) followed by one fully-connected layer of dimension 256. Then 3

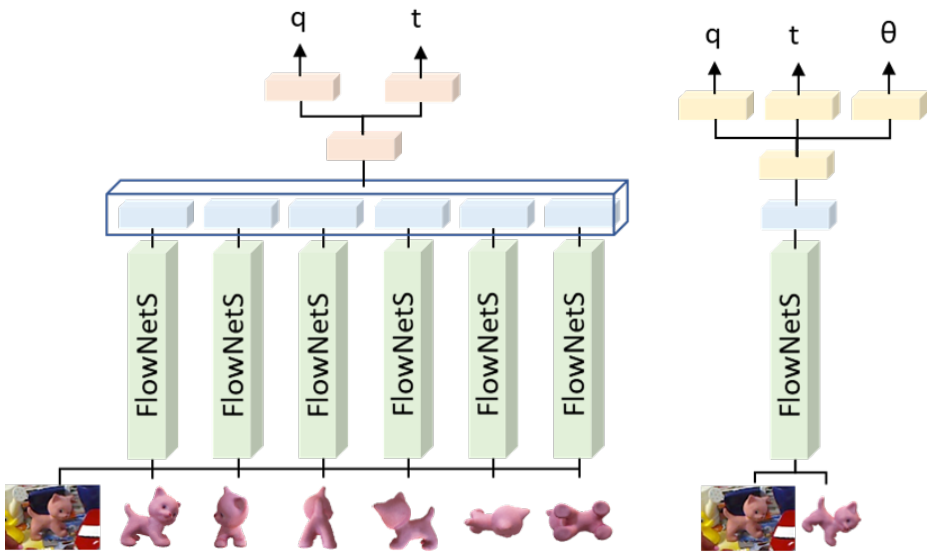

Fig. 3.8. Multi-View Matching Network (left) and Single-View Matching Network (right).

fully-connected layers of dimensions 4, 3, and 1 are used to estimate the relative rotation, translation, and angle distance respectively. The weights of the FlowNetS convolutional layers and the first fully-connected layer are shared with MV-Net. Note that without the angle distance estimation output, SV-Net is equivalent to DeepIM [88].

As in MV-Net, the input of the SV-Net consists of one 8 channel tensor composed by the target RGB+mask image, and an RGB+mask rendered image. The rendered image is obtained by rendering the object with the previously estimated pose. The refinement process starts with the initial pose estimate of MV-Net. Then, SV-Net estimates the pose difference between the rendered image and the zoomed input image. The pose difference is used to update the previous pose estimate. A new image is rendered with the updated pose and the refinement process is repeated until the estimated rotation angle, $\hat{\theta}$, between the two images, is below a threshold $T_{ref} = 2°$. If after 50 refinement iterations, the estimated rotation angle isn't below T_{ref}, the pose estimate with the lowest estimated rotation angle is selected. This policy to stop the refinement process differs from other methods (e.g. [88])

where a fixed number of refinement iterations are used. Figure 3.9 shows the refinement process.

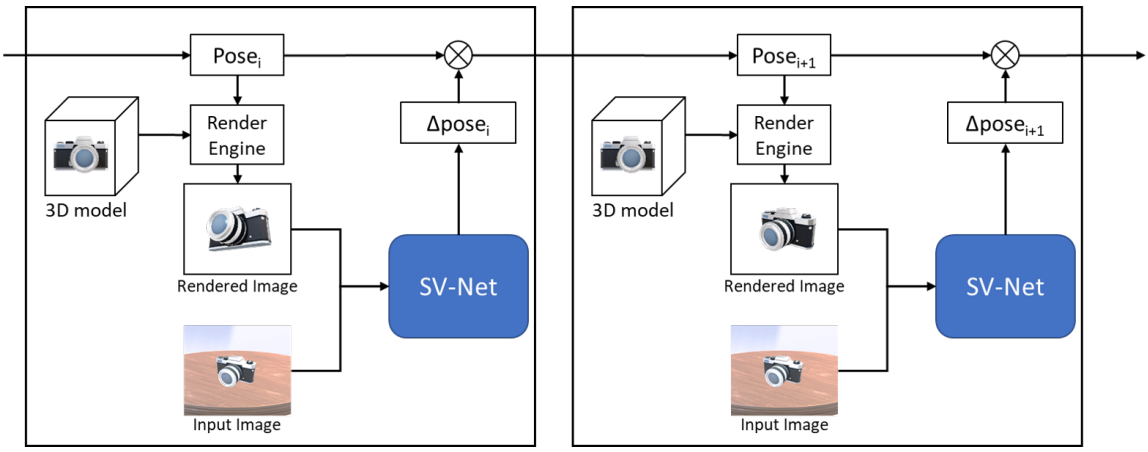

Fig. 3.9. Our refinement process of SV-Net is based on the method [88].

Pose Tracking

MV-Net and SV-Net pose estimation and pose refinement can be easily extended for pose tracking. After the pose is estimated and refined at the initial frame, the pose tracking is performed by estimating the pose difference between the current pose estimate and the next frame of the video. Then, the pose estimate is updated and the process is repeated for the following frame. Figure 3.10 shows the tracking process. Note that the only difference between pose refinement and pose tracking is the input frames: while pose refinement is performed in an individual frame, pose tracking has as input consecutive frames.

The angle distance estimate of the SV-Net can be used to assess the tracking process. If the angle distance estimate is higher than a threshold $T_{high} = 25°$, we restart the tracking process by estimating a new initial pose with MV-Net. If the angle distance estimate is lower than a threshold $T_{low} = 2°$, it is likely that the camera or object is not moving, therefore the pose estimate is not updated.

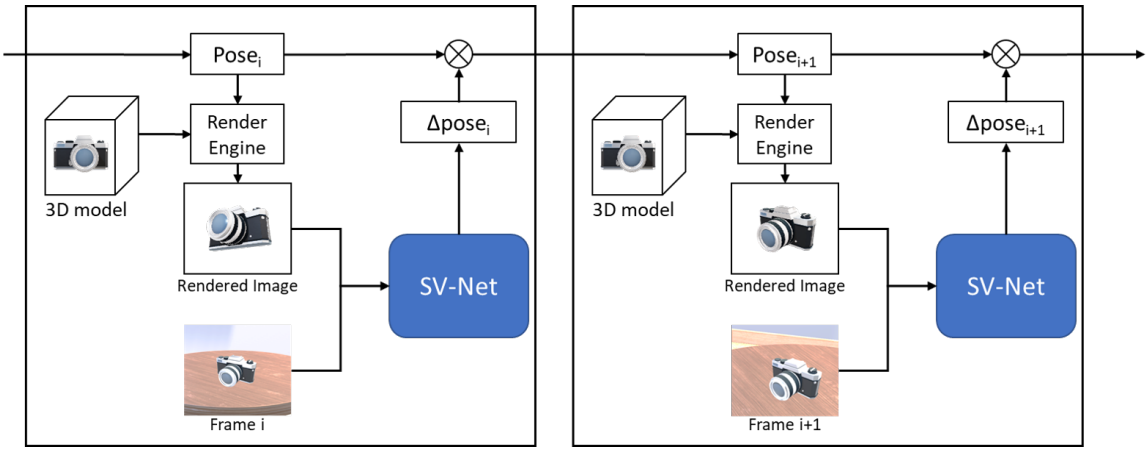

Fig. 3.10. SV-Net can track the pose of objects by performing pose refinement within consecutive frames.

Pose Transformation Representation

The representation of the SE(3) transformation used in this work is the same as the one used in [88]. This untangled pose representation is independent of the coordinate system of the object, thereby the network does not have to learn the geometry of the object. This is accomplished by modeling the pose change that needs to be applied to the camera (instead of the object). The SE(3) transformation between a source pose $p_{src} = [R_{src}|t_{src}]$ and a target pose $p_{tgt} = [R_{tgt}|t_{tgt}]$ is shown in equation 3.1.

$$\begin{aligned} R_{tgt} &= R_{\Delta} R_{src} \\ t_{tgt} &= R_{\Delta} t_{src} + t_{\Delta} \end{aligned} \tag{3.1}$$

Instead of training the neural networks to estimate directly R_{Δ} and t_{Δ}, we estimate the equivalent rotation quaternion q_{Δ} and the untangled translation v_{Δ}. $v_{\Delta} = (v_x, v_y, v_z)$ is defined in equation 3.2.

$$\begin{aligned} v_x &= f_x(x_{tgt}/z_{tgt} - x_{src}/z_{src}), \\ v_y &= f_y(y_{tgt}/z_{tgt} - y_{src}/z_{src}), \\ v_z &= \log(z_{src}/z_{tgt}) \end{aligned} \tag{3.2}$$

Where $t_{src} = (x_{src}, y_{src}, z_{src})$ and $t_{tgt} = (x_{tgt}, y_{tgt}, z_{tgt})$ are the source and target location respectively, and f_x and f_y are the focal length of the camera. As we use multiple training datasets captured with cameras with different intrinsic parameters, we assign the adequate values of f_x and f_y during training and testing. v_{Δ} and q_{Δ} represent the pose difference between the two input images when training SV-Net, and the pose difference between the input and the frontal face of the object (input of the 1st branch) when training MV-Net.

Training Process

Our networks are trained using the LINEMOD [104] dataset and synthetic images from Extra FAT dataset, described in the following section. These datasets include the 3D models

of multiple household objects and images containing such objects. We use as ground truth the bounding box surrounding the objects of interest, their segmentation mask, and their pose represented with a rotation quaternion, and the 3D location values.

We fine-tune Mask R-CNN using stochastic gradient descent (SGD) with a learning rate of 0.005 for 10 epochs. We use pre-trained weights from the MS COCO dataset [36].

We train MV-Net and SV-Net using the loss functions in Equation 3.3 and Equation 3.4 respectively.

$$L_m(p, \hat{p}) = \frac{1}{N}(\|1 - q^T \frac{\hat{q}}{\|\hat{q}\|}\| + \|t - \hat{t}\| + \|1 - \|\hat{q}\|\|) \tag{3.3}$$

$$L_s(p, \hat{p}) = \|1 - q^T \frac{\hat{q}}{\|\hat{q}\|}\| + \|t - \hat{t}\| + \|1 - \|\hat{q}\|\| + \|\theta - \hat{\theta}\| \tag{3.4}$$

Where $p = [q|t]$ and $\hat{p} = [\hat{q}|\hat{t}]$ are the ground truth and estimated rotation quaternion and translation parameters respectively. θ and $\hat{\theta}$ are the ground truth and estimated angle distance respectively. $\hat{\theta}$ is only estimated when training SV-Net. N is the number of views. In this work, we use $N = 6$.

We follow a simultaneous end-to-end training approach. The training process starts by estimating the initial pose with MV-Net. Then, the initial estimate is used to render a new training sample for SV-Net. We perform a total of 3 refinement iterations during training time. We compute the error between pose estimates and the ground truth pose for the initial pose and each of the 3 refined poses. Finally, we backpropagate all the losses.

In order to avoid training SV-Net with really bad initial estimates, we generate a new random rotation if the angle error of the initial estimate is higher than $25°$. By doing so we ease the training process of SV-Net as we only use examples where the pose difference is small and allow the network to learn to refine and fix small differences.

We initialize the FlowNetS convolutional layers of MV-Net and SV-Net with pre-trained weights for optical flow estimation. The fully-connected layers are initialized with random weights as in [88]. We train the networks using SGD with a learning rate of 0.001 during 48 epochs.

Our MV-Net and SV-Net simultaneous end-to-end training approach have several advantages. First, we can easily apply weight sharing between MV-Net and SV-Net and force MV-Net to learn to detect pose differences between the target and rendered images. Second, we present the training examples to the network in the same order that will be found during testing time. Therefore, because SV-Net is trained with the pose estimates produced by MV-Net, SV-Net learns to fix the specific mistakes that MV-Net produces. This differs from techniques where different methods are used for pose estimation and for pose refinement. In such techniques, the pose refinement methods might not be trained to specifically fix the pose errors that are generated by the initial pose estimation method, leading to lower accuracy.

Datasets

We use the LINEMOD (LM) dataset [104] for training and testing. This dataset contains 15 objects but we only use 13 as in previous works [88]. We use the same training and testing split as in [88]. In addition, we use two different rendered datasets for training: A domain randomized dataset and a photorealistic dataset. Figure 3.11 shows some examples of both rendered datasets.

The domain randomized dataset is generated by rendering the objects in random poses and adding background images. The background images are randomly selected from Pascal VOC [35] dataset. Additionally, we apply random blurring and color jittering to the image. The segmentation mask is dilated with a square kernel with a size randomly selected from 0 to 40 in order to mimic errors in the segmentation mask during inference. We render 5,000 training images for each object.

We the 3D models of the LM dataset in virtual environments to generate photorealistic images with Unreal Engine 4 (UE4) [111]. This includes a total of 5,000 images per object. These images are a subset of Extra FAT dataset, described in the following section.

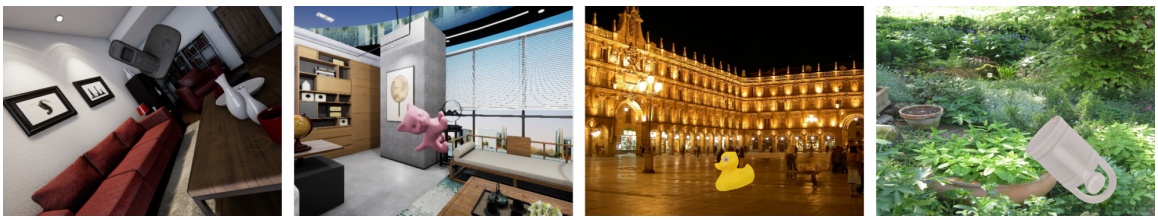

Fig. 3.11. Examples of photorealistic images (left) and domain randomized images (right).

3.3.2 Experimental Results

We evaluate our method using the ($n°$, n cm) accuracy metric with the LINEMOD testing set. This metric considers an estimate correct if the error is smaller than $n°$ and n cm and incorrect otherwise. We evaluate our method for $n = 2, 5, 10$. Table 3.1 presents our accuracy results.

Table 3.1.
Comparison with previous methods on LINEMOD dataset.

	($n°$, n cm)		
Method	(2, 2)	(5, 5)	(10, 10)
BB8 w/ ref. [87]	-	69.0%	-
PoseCNN [91]	-	19.4%	-
PoseCNN+DeepIM [88]	39.0%	85.2%	97.9%
MV-Net + SV-Net (Ours)	24.3%	73.1%	97.5%

We can observe that our accuracy results are comparable with previous state-of-the-art RGB-only pose estimation methods such as PoseCNN+DeepIM [88].

Our method shows that pose refinement methods [88] can successfully be extended for initial pose estimation. Therefore, initial pose estimation networks such as PoseCNN [91] can be replaced by highly available object detection networks [17,44] which can be trained with a large amount of training data.

3.4 Extra-FAT: 3D Pose Estimation Dataset

3.4.1 Dataset Overview

In this section, we describe the synthetic rendered dataset named Extra FAT [109]. The dataset is named Extra FAT as it is generated following a similar approach as in the FAT dataset [108]. However, Extra FAT includes more object models and more virtual scenes

than FAT. This dataset includes photorealistic rendered images containing several 3D objects from the most commonly used datasets for 6D pose estimation. Table 3.3 presents an in-depth comparison of the Extra FAT dataset with previously presented datasets.

The Extra FAT dataset includes rendered images of 640×480 pixels with the location and rotation parameters for both the virtual camera and the 3D objects and a pixel-level object segmentation mask. Figure 3.12 shows some examples of a rendered image and its segmentation mask. The set of images, segmentation mask, and location and rotation parameters can be used to train and evaluate methods for object detection, segmentation and pose estimation and tracking.

Fig. 3.12. Example of an image and a segmentation mask from Extra FAT dataset [109].

The 3D objects are placed in 5 different virtual indoor scenes that have a wide range of illumination and occlusion conditions. The virtual scenes include common indoor environments such as office spaces, living rooms, and kitchens. A total of 825,000 images are generated. The specifications for Extra FAT dataset are described in Table 3.2. Figure 3.13 shows multiple examples of rendered images.

The Extra FAT dataset includes a total of 110 different 3D object models: 15 objects from the LINEMOD dataset, 21 household objects from the YCB dataset, 14 objects from the Amazon Picking Challenge 2015 dataset, 30 objects from the T-LESS dataset, 6 objects from the IC-MI dataset, 3 objects from the TUD Light dataset and 21 objects from the

Table 3.2.
Dataset Specification. Table from [109].

Extra FAT Dataset	
Image Resolution	640 × 480
Field of view	90°
Number of frames	825k
Number of objects	110
Number of scenes	5

Fig. 3.13. Examples of rendered images with different objects and scene types. Images from [109].

TOYOTA Light. Figure 3.4, Figure 3.5 and Figure 3.6 show multiple examples of 3D models used in Extra FAT.

3.4.2 Dataset Generation

Following the generation scheme of the FAT dataset, we use the Unreal Engine 4 (UE4) [112], a widely used tool for animation and game development, to render the 3D object models in virtual indoor scenarios. Additionally, we use the open-source UnrealCV [112] plugin as a communication tool to generate photorealistic images and specify the parameters of the camera pose and object pose.

Before starting the rendering process, we manually specify some candidate locations within the virtual scenarios. The locations are selected so the 3D objects are not highly occluded by some elements of the virtual scenario. During the image generation stage, pairs of candidate pre-defined locations are selected randomly and the virtual camera and the 3D object trajectories are specified by linearly interpolating between the two locations. Figure 3.14 shows an example of multiple images rendered in the interpolated locations. Additionally, we apply a uniformly random perturbation in the location and rotation of the object and virtual camera during the interpolation stage. The distributions of Yaw, Pitch, and Roll angles are uniform indicating that the poses of objects in the dataset are representative for the general pose estimation task.

In order to always place the object within the visible range of the camera, we constrain the relative location and rotation between the camera and objects. Figure 3.15 shows the pixel coordinate (p_x, p_y) constraint, represented as:

$$\begin{aligned} -\mu_x < p_x < \mu_x, \\ -\mu_y < p_y < \mu_y, \end{aligned} \tag{3.5}$$

where (p_x, p_y) are the pixel coordinate and $\mu_x = 200$ and $\mu_y = 180$.

The relation between the 3D coordinate location in the virtual scenario and the pixel coordinate can be expressed as:

Fig. 3.14. Linear interpolation trajectory from candidate location points. Images from [109].

$$t_x = p_x \frac{t_z}{f_x}$$
$$t_y = p_y \frac{t_z}{f_y} \tag{3.6}$$

Where f_x, f_y are the focal lengths in the x and y direction. t_z, t_z , t_z are the 3D coordinates in the virtual environment. The parameter t_z is in the range $(0.3, 0.8)$ to make sure that the object is in front of the camera and not too far or too close to it.

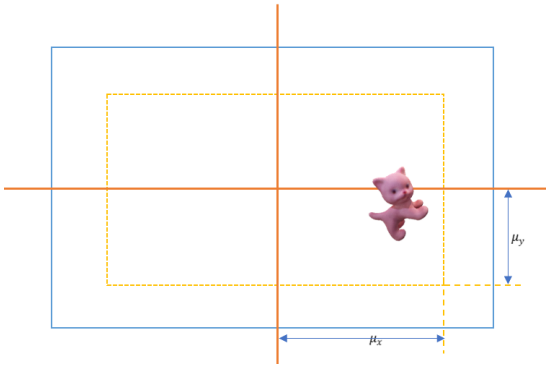

Fig. 3.15. Pixel coordinate constraint. Figure adapted from [109].

In order to avoid having objects highly occluded by elements of the virtual scenario (such as a wall or other objects), we add a constraint on the ratio of mask area to image size. Specifically, we discard images where the segmentation mask area to image size ratio is lower than a $threshold$.

$$\frac{\sum_{mask} 1}{w \times h} > threshold \tag{3.7}$$

In this work we use $threshold = 0.05$.

Table 3.3.
Comparison of different 3D datasets. Table adapted from [109].

Dataset	# obj	# frames	Type	LM	YCB	T-LESS	IC-MI	TYO-L	RU-APC	TUD-L
LINEMOD [104]	15	18k	real	✓						
LM OCC [105, 106]	15	18k	real	✓						
YCB-Video [91]	21	134k	real		✓					
FAT [108]	21	60k	rendered		✓					
T-LESS [100]	30	48.9k	real			✓				
IC-MI [103]	6	4.2k	real				✓			
TYO-L [110]	21	55.4k	combined					✓		
RU-APC [102]	24	10k	real						✓	
TUD-L [110]	6	62.3k	combined							✓
BOP [110]	89	>294k	combined	✓		✓	✓	✓	✓	✓
Extra FAT [109]	110	825k	rendered	✓	✓	✓	✓	✓	✓	✓

4. DEEPFAKES DETECTION

4.1 Overview

Manipulated multimedia is rapidly increasing its presence on the Internet and social media. Its rise is fueled by the mass availability of easy-to-use tools and techniques for generating realistic fake multimedia content. Recent advancements in the field of deep learning have led to the development of methods to create artificial images and videos that are eerily similar to authentic images and videos. Manipulated multimedia created using such techniques typically involving neural networks, such as Generative Adversarial Networks (GAN) [113] and Auto-Encoders (AE) [12], are generally referred to as Deepfakes. While these tools can be useful to automate steps in movie production, video game design, or virtual reality rendering, they are potentially very damaging if used for malicious purposes. As manipulation tools become more accessible, realistic, and undetectable, the divide between real and fake multimedia is blurred. Furthermore, social media allows for the uncontrolled spread of manipulated content at a large scale. This spread of misinformation damages journalism and news providers as it gets increasingly difficult to distinguish between reliable and untrustworthy information sources.

Fig. 4.1. Example of images from DFDC [114] dataset: original image (left) and manipulated image with the swapped face (right).

Human facial manipulations are among the most common Deepfake forgeries. Through face swaps, an individual can be placed at some location he or she was never present at.

By altering the lip movement and the associated speech signal, realistic videos can be generated of individuals saying words they actually never uttered. This type of Deepfake manipulation can be very damaging when used to generate graphic adult content or fake news that can alter the public opinion. In fact, many images and videos containing such Deepfake forgeries are already present on adult content web sites, news articles, and social media.

Image and video manipulations have been utilized for a long time. Before the advent of Deepfakes, editing tools such as Photoshop [115] or GIMP [116] have been widely used for image manipulations. Some common forgeries include splicing (inserting objects into images) [117], copy and moving parts within an image (copy-move forgery) [118], or shadow removal [119]. While research on detecting such manipulations has been conducted for more than a decade [117–125], many techniques fail to detect more recent and realistic manipulations, especially when the multimedia alterations are performed with deep learning methods. Fortunately, there is an increasing effort to develop reliable detection technology such as AWS, Facebook, Microsoft, and the Partnership on AIs Media Integrity Steering Committee with the Deepfake Detection Challenge (DFDC) [114].

Advances in deep learning have resulted in a great variety of methods that have provided groundbreaking results in many areas including computer vision, natural language processing, and biomedical applications [29]. While several neural networks that detect a wide range of manipulations have been introduced [126–131], new generative methods that create very realistic fake multimedia [132–136] are presented every year, leading to a push and pull problem where manipulation methods try to fool new detection methods and vice-versa. Therefore, there is a need for methods that are capable of detecting multimedia manipulations in a robust and rapid manner.

In this chapter, we present a novel model architecture that combines a Convolutional Neural Network (CNN) with a Recurrent Neural Network (RNN) to accurately detect facial manipulations in videos. The network automatically selects the most reliable frames to detect these manipulations with a weighting mechanism combined with a Gated Recurrent Unit (GRU) that provides a final probability of a video being real or being fake. We train

and evaluate our method with the Deepfake Detection Challenge dataset, obtaining a final score of 0.321 (log-likelihood error, the lower the better) at position 117 of 2275 teams (top 6%) of the public leader-board.

4.2 Related Work

There are many techniques for face manipulation and generation. Some of the most commonly used include FaceSwap [137], Face2Face [138], DeepFakes [135], and NeuralTextures [136]. FaceSwap and Face2Face are computer graphics based methods while the other two are learning based methods. In FaceSwap [137], a face from a source video is projected onto a face in a target video using facial landmark information. The face is successfully projected by minimizing the difference between the projected shape and the target face's landmarks. Finally, the rendered face is color corrected and blended with the target video. In Face2Face [138], facial expressions from a selected face in a source video are transferred to a face in the target video. Face2Face uses selected frames from each video to create dense reconstructions of the two faces. These dense reconstructions are used to re-synthesize the target face with different expressions under different lighting conditions. In DeepFakes [135], two autoencoders [12] (with a shared encoder) are trained to reconstruct target and source faces. To create fake faces, the trained encoder and decoder of the source face are applied on the target face. This fake face is blended onto the target video using Poisson image editing [139], creating a Deepfake video. Note the difference between DeepFakes (capital F), the technique now being described, and Deepfakes (lowercase f), which is a general term for fake media generated with deep learning-based methods. In NeuralTextures [136], a neural texture of the face in the target video is learned. This information is used to render the facial expressions from the source video on the target video.

In recent years, methods have been developed to detect such deep learning-based manipulations. In [126], several CNN architectures have been tested in a supervised setting to discriminate between GAN generated images and real images. Preliminary results are promising but the performance degrades as the difference between training and testing in-

creases or when the data is compressed. In [127–129], forensic analysis of GAN generated images revealed that GANs leave some high frequency fingerprints in the images they generate.

Additionally, several techniques to detect videos containing facial manipulations have been presented. While some of these methods focus on detecting videos containing only DeepFake manipulations, others are designed to be agnostic to the technique used to perform the facial manipulation. The work presented in [140, 141] use a temporal-aware pipeline composed by a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN) to detect DeepFake videos. Current DeepFake videos are created by splicing synthesized face regions onto the original video frames. This splicing operation can leave artifacts that can later be detected when estimating the 3D head pose. The authors of [142] exploit this fact and use the difference between the head pose estimated with the full set of facial landmarks and a subset of them to separate DeepFake videos from real videos. This method provided competitive results on the UADFV [143] database. The same authors proposed a method [144] to detect DeepFake videos by analyzing the face warping artifacts. The authors of [130] detect manipulated videos generated by the DeepFake and Face2Face techniques with a shallow neural network that acts on mesoscopic features extracted from the video frames to distinguish manipulated videos from real ones. However, the results presented in [131] demonstrated that in a supervised setting, several deep network based models [145–147] outperform the ones based on shallow networks when detecting fake videos generated with DeepFake, Face2Face, FaceSwap, and NeuralTexture.

4.3 Deepfake Detection Challenge Dataset

The Deepfake Detection Challenge (DFDC) [114] dataset contains a total of 123,546 videos with face and audio manipulations. Each video contains one or more people and has a length of 10 seconds with a total of 300 frames. The nature of these videos typically includes standing or sitting people, either facing the camera or not, with a wide range of backgrounds, illumination conditions, and video quality. The training videos have a

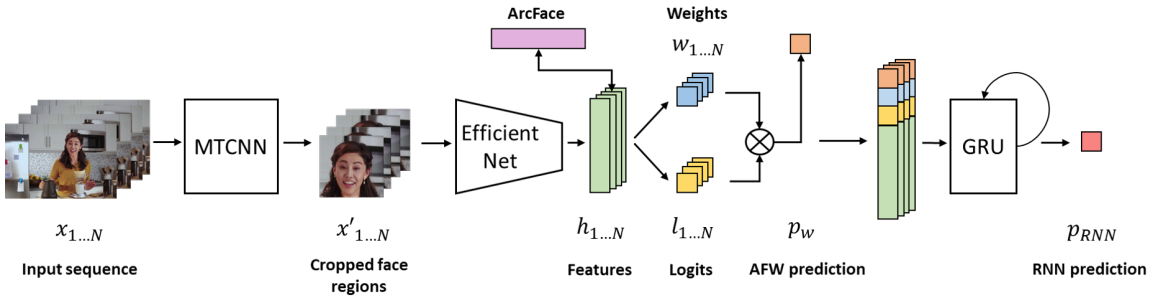

Fig. 4.2. Block Diagram of our proposed Deepfake detection system: MTCNN detects faces within the input frames, then EfficientNet extracts features from all the detected face regions, and finally the Automatic Face Weighting (AFW) layer and the Gated Recurrent Unit (GRU) predict if the video is real or manipulated.

resolution of 1920×1080 pixels, or 1080×1920 pixels if recorded in vertical mode. Figure 4.1 shows some examples of frames from videos of the dataset. This dataset is composed by a total of 119,146 videos with a unique label (real or fake) in a training set, 400 videos on the validation set without labels and 4000 private videos in a testing set. The 4000 videos of the test set can not be inspected but models can be evaluated on it through the Kaggle system. The ratio of manipulated:real videos is 1:0.28. Because only the 119,245 training videos contain labels, we use the totality of that dataset to train and validate our method. The provided training videos are divided into 50 numbered parts. We use 30 parts for training, 10 for validation and 10 for testing.

A unique label is assigned to each video specifying whether it contains a manipulation or not. However, it is not specified which type of manipulation is performed: face, audio, or both. As our method only uses video information, manipulated videos with only audio manipulations will lead to noisy labels as the video will be labeled as fake but faces will be real. Furthermore, more than one person might be present in the video, with face manipulations performed on only one of them.

The private set used for testing evaluates submitted methods within the Kaggle system and reports a log-likelihood loss. Log-likelihood loss drastically penalizes being both confident and wrong. In the worst case, a prediction that a video is authentic when it is actually manipulated, or the other way around, will add infinity to your error score. In practice, if this worst-case happens, the loss is clipped to a very big value. This evaluation system poses an extra challenge, as methods with good performance in metrics like accuracy, could have very high log-likelihood errors.

4.4 Proposed Method

Our proposed method (Figure 4.2) extracts visual and temporal features from faces by using a combination of a CNN with an RNN. Because all visual manipulations are located within face regions, and faces are typically present in a small region of the frame, using a network that extracts features from the entire frame is not ideal. Instead, we focus on

extracting features only in regions where a face is present. Because networks trained with general image classification task datasets such as ImageNet [148] have performed well when transferred to other tasks [149], we use pre-trained backbone networks as our starting point. Such backbone networks extract features from faces that are later fed to an RNN to extract temporal information. The method has three distinct steps: (1) face detection across multiple frames using MTCNN [150], (2) feature extraction with a CNN, and (3) prediction estimation with a layer we refer to as Automatic Face Weighting (AFW) along with a Gated Recurrent Unit (GRU). Our approach is described in detail in the following subsections, including a boosting and test augmentation approach we included in our DFDC submission.

4.4.1 Face Detection

We use MTCNN [150] to perform face detection. MTCNN is a multi-task cascaded model that can produce both face bounding boxes and facial landmarks simultaneously. The model uses a cascaded three-stage architecture to predict face and landmark locations in a coarse-to-fine manner. Initially, an image pyramid is generated by resizing the input image to different scales. The first stage of MTCNN then obtains the initial candidates of facial bounding boxes and landmarks given the input image pyramid. The second stage takes the initial candidates from the first stage as the input and rejects a large number of false alarms. The third stage is similar to the second stage but with a larger input image size and deeper structure to obtain the final bounding boxes and landmark points. Non-maximum suppression and bounding box regression are used in all three stages to remove highly overlapped candidates and refine the prediction results. With the cascaded structure, MTCNN refines the results stage by stage in order to get accurate predictions.

We choose this model because it provides good detection performance on both real and synthetic faces in the DFDC dataset. While we also considered more recent methods like BlazeFace [151], which provides faster inferencing, its false positive rate on the DFDC dataset was considerably larger than that of MTCNN.

We extract faces from 1 every 10 frames for each video. In order to speed up the face detection process, we downscale the frame by a factor of 4. Additionally, we include a margin of 20 pixels at each side of the detected bounding boxes in order to capture a broader area of the head as some regions such as the hair might contain artifacts useful to detect manipulations. After processing the input frames with MTCNN, we crop all the regions where faces were detected and resize them to 224×224 pixels.

4.4.2 Network Architecture

Face Feature Extraction

After detecting face regions, a binary classification model is trained to extract features that can be used to classify the real/fake faces. The large number of videos that have to be processed in a finite amount of time for the Deepfake Detection Challenge requires networks that are both fast and accurate. In this work, we use EfficientNet-b5 [152] as it provides a good trade-off between network parameters and classification accuracy. Additionally, the network has been designed using neural architecture search (NAS) algorithms, resulting in a network that is both compact and accurate. In fact, this network has outperformed previous state-of-the-art approaches in datasets such as ImageNet [148] while having fewer parameters.

Since the DFDC dataset contains many high-quality photo-realistic fake faces, discriminating between real and manipulated faces can be challenging. To achieve a better and more robust face feature extraction, we combine EfficientNet with the additive angular margin loss (also known as ArcFace) [153] instead of a regular softmax+cross-entropy loss. ArcFace is a learnable loss function that is based on the classification cross-entropy loss but includes penalization terms to provide a more compact representation of the categories. ArcFace simultaneously reduces the intra-class difference and enlarges the inter-class difference between the classification features. It is designed to enforce a margin between the distance of the sample to its class center and the distances of the sample to the centers of other classes in an angular space. Therefore, by minimizing the ArcFace loss, the clas-

sification model can obtain highly discriminative features for real faces and fake faces to achieve a more robust classification that succeeds even for high-quality fake faces.

Automatic Face Weighting

While an image classification CNN provides a prediction for a single image, we need to assign a prediction for an entire video, not just a single frame. The natural choice is to average the predictions across all frames to obtain a video-level prediction. However, this approach has several drawbacks. First, face detectors such as MTCNN can erroneously report that background regions of the frames contain faces, providing false positives. Second, some videos might include more than one face but with only one of them being manipulated. Furthermore, some frames might contain blurry faces where the presence of manipulations might be difficult to detect. In such scenarios, a CNN could provide a correct prediction for each frame but an incorrect video-level prediction after averaging.

In order to address this problem, we propose an automatic weighting mechanism to emphasize the most reliable regions where faces have been detected and discard the least reliable ones when determining a video-level prediction. This approach, similar to attention mechanisms [154], automatically assigns a weight, w_j, to each logit, l_j, outputted by the EfficientNet network for each jth face region. Then, these weights are used to perform a weighted average of all logits, from all face regions found in all sampled frames to obtain a final probability value of the video being fake. Both logits and weights are estimated using a fully-connected linear layer with the features extracted by EfficientNet as input. In other words, the features extracted by EfficientNet are used to estimate a logit (that indicates if the face is real or fake) and a weight (that can provide information of how confident or reliable is the logit prediction). The output probability, p_w, of a video being false, by the automatic face weighting is:

$$p_w = \sigma\left(\frac{\sum_{j=1}^{N} w_j l_j}{\sum_{j=1}^{N} w_j}\right) \tag{4.1}$$

Where w_j and l_j are the weight value and logit obtained for the jth face region, respectively and $\sigma(.)$ is the Sigmoid function. Note that after the fully-connected layer, w_j is passed through a ReLU activation function to enforce that $w_j \geq 0$. Additionally, a very small value is added to avoid divisions by 0. This weighted sum aggregates all the estimated logits providing a video-level prediction.

Gated Recurrent Unit

The backbone model estimates a logit and weight for each frame without using information from other frames. While the automatic face weighting combines the estimates of multiple frames, these estimates are obtained by using single-frame information. However, ideally the video-level prediction would be performed using information from all sampled frames.

In order to merge the features from all face regions and frames, we include a Recurrent Neural Network (RNN) on top of the automatic face weighting. We use a Gated Recurrent Unit (GRU) to combine the features, logits, and weights of all face regions to obtain a final estimate. For each face region, the GRU takes as input a vector of dimension 2051 consisting of the features extracted from EfficientNet (with dimension 2048), the estimated logit l_j, the estimated weighting value w_j, and the estimated manipulated probability after the automatic face weighting p_w. Although l_j, w_j, p_w, and the feature vectors are correlated, we input all of them to the GRU and let the network itself extract the useful information. The GRU is composed of 3 stacked bi-directional layers and a uni-directional layer with a hidden layer with dimension 512. The output of the last layer of the GRU is mapped through a linear layer and a Sigmoid function to estimate a final probability p_{RNN} of the video being manipulated.

4.4.3 Training Process

We use a pre-trained MTCNN for face detection and we only train our EfficientNet, GRU, and the Automatic Face Weighting layers. The EfficientNet is initialized with weights

pre-trained on ImageNet. The GRU and AFW layers are initialized with random weights. During the training process, we oversample real videos (containing only unmanipulated faces) to balance the dataset. The network is trained end-to-end with 3 distinct loss functions: an ArcFace loss with the output of EfficentNet, a binary cross-entropy loss with the automatic face weighting prediction p_w, and a binary cross-entropy loss with the GRU prediction p_{RNN}.

The ArcFace loss is used to train the EfficientNet layers with batches of cropped faces from randomly selected frames and videos. This loss allows the network to learn from a large variety of manipulated and original faces with various colors, poses, and illumination conditions. Note that ArcFace only trains the layers from EfficientNet and not the GRU layers or the fully-connected layers that output the AFW weight values and logits.

The binary cross-entropy (BCE) loss is applied at the outputs of the automatic face weighting layer and the GRU. The BCE loss is computed with cropped faces from frames of a randomly selected video. Note that this loss is based on the output probabilities of videos being manipulated (video-level prediction), while ArcFace is a loss based on frame-level predictions. The BCE applied to p_w updates the EfficientNet and AFW weights. The BCE applied to p_{RNN} updates all weights of the ensemble (excluding MTCNN).

While we train the complete ensemble end-to-end, we start the training process with an optional initial step consisting of 2000 batches of random crops applied to the ArcFace loss to obtain an initial set of parameters of the EfficientNet. This initial step provides the network with useful layers to later train the automatic face weighting layer and the GRU. While this did not present any increase in detection accuracy during our experiments, it provided a faster convergence and a more stable training process.

Due to computing limitations of GPUs, the size of the network, and the number of input frames, only one video can be processed at a time during training. However, the network parameters are updated after processing every 64 videos (for the binary cross-entropy losses) and 256 random frames (for the ArcFace loss). We use Adam as the optimization technique with a learning rate of 0.001.

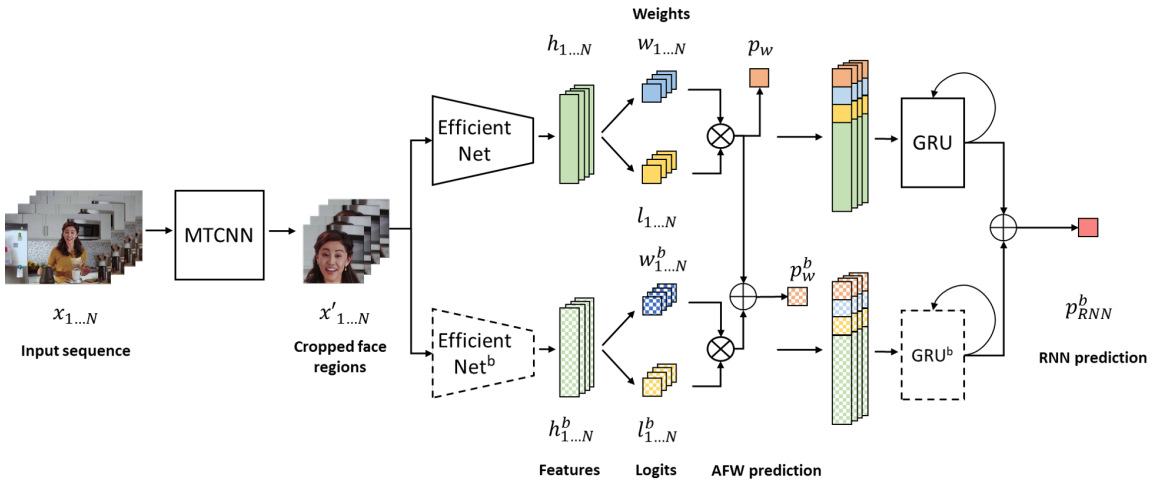

Fig. 4.3. Diagram of the proposed method including the boosting network (dashed elements). The predictions of the main and boosting network are combined at the AFW layer and after the GRUs. We train the main network with the training set and the boosting network with the validation set.

4.4.4 Boosting Network

The logarithmic nature of the binary cross-entropy loss (or log-likelihood error) used at the DFDC leads to large penalizations for predictions both confident and incorrect. In order to obtain a small log-likelihood error we want a method that has both good detection accuracy and is not overconfident of its predictions. In order to do so, we use two main approaches during testing: (1) adding a boosting network and (2) applying data augmentation during testing.

The boosting network is a replica of the previously described network. However, this auxiliary network is not trained to minimize the binary cross-entropy of the real/fake classification, but trained to predict the error between the predictions of our main network and the ground truth labels. We do so by estimating the error of the main network on the logit domain for both the AFW and GRU outputs. When using the boosting network, the prediction outputted by the automatic face weighting layer, p_w^b, is defined as:

$$p_w^b = \sigma\left(\frac{\sum_{j=1}^{N}(w_j l_j + w_j^b l_j^b)}{\sum_{j=1}^{N}(w_j + w_j^b)}\right) \tag{4.2}$$

Where w_j and l_j are the weights and logits outputted by the main network and w_j^b and l_j^b, are the weights and logits outputted by the boosting network for the jth input face region and $\sigma(.)$ is the Sigmoid function. In a similar manner, the prediction outputted by the GRU, p_{RNN}^b, is:

$$p_{RNN}^b = \sigma(l_{RNN} + l_{RNN}^b) \tag{4.3}$$

Where l_{RNN} is the logit outputted by the GRU of the main network, l_{RNN}^b is the logit outputted by the GRU of the boosting network, and $\sigma(.)$ is the Sigmoid function.

While the main network is trained using the training split of the dataset, described in section 5.3, we train the boosting network with the validation split.

Figure 4.3 presents the complete diagram of our system when including the boosting network. The dashed elements and the symbols with superscripts form part of the boosting network. The main network and the boosting network are combined at two different points:

at the automatic face weighting layer, as described in equation 4.2, and after the gated recurrent units, as described in equation 4.3.

4.4.5 Test Time Augmentation

Besides adding the boosting network, we perform data augmentation during testing. For each face region detected by the MTCNN, we crop the same region in the 2 previous and 2 following frames of the frame being analyzed. Therefore we have a total of 5 sequences of detected face regions. We run the network within each of the 5 sequences and perform a horizontal flip in some of the sequences randomly. Then, we average the prediction of all the sequences. This approach helps to smooth out overconfident predictions: if the predictions of different sequences disagree, averaging all the probabilities leads to a lower number of both incorrect and overconfident predictions.

4.5 Experimental Results

We train and evaluate our method with the DFDC dataset, described in section 5.3. Additionally, we compare the presented approach with 4 other techniques. We compare it with the work presented in [140] and a modified version that only process face regions detected by MTCNN. We also evaluate two CNNs: EfficientNet [152] and Xception [147]. For these networks, we simply average the predictions for each frame to obtain a video-level prediction.

We use the validation set to select the configuration for each models that provides the best balanced accuracy. Table 4.1 presents the results of balanced accuracy. Because it is based on extracting features on the entire video, Conv-LSTM [140] is unable to capture the manipulations that happen within face regions. However, if the method is adapted to process only face regions, the detection accuracy improves considerably. Classification networks such as Xception [147], which provided state-of-the-art results in FaceForensics++ dataset [131], and EfficientNet-b5 [152] show good accuracy results. Our work shows

that by including an automatic face weighting layer and a GRU, the accuracy is further improved.

Table 4.1.
Balanced accuracy of the presented method and previous works.

Method	Validation	Test
Conv-LSTM [140]	55.82%	57.63%
Conv-LSTM [140] **+ MTCNN**	66.05%	70.78%
EfficientNet-b5 [152]	79.25%	80.62%
Xception [147]	78.42%	80.14%
Ours	92.61%	91.88%

Additionally, we evaluate the accuracy of the predictions at every stage of our method. Table 4.2 shows the balanced accuracy of the prediction obtained by the averaging the logits predicted by EfficientNet, l_j (logits), the prediction of the automatic face weighting layer, p_w (AFW), and the prediction after the gated recurrent unit, p_{RNN} (GRU). We can observe that every stage increases the detection accuracy, obtaining the highest accuracy with the GRU prediction.

Table 4.2.
Balanced accuracy of at different stages of our method.

Method	Validation Accuracy
Ours (logits)	85.51%
Ours (AFW)	87.90%
Ours (GRU)	92.61%

Figure 4.4 shows some examples of correctly (bottom) and incorrectly (top) detected manipulations. We observed that the network typically fails when faced with highly-

realistic manipulations that are performed in blurry or low-quality images. Manipulations performed in high-quality videos seem to be properly detected, even the challenging ones.

Fig. 4.4. Examples of faces with manipulations from DFDC. The images in the top are incorrectly classified by the network. The bottom images are correctly classified.

We evaluate the effect of using the boosting network and data augmentation during testing. In order to so, we use the private testing set on the Kaggle system and report our log-likelihood error (the lower the better). Table 4.3 shows that by using both the boosting and test augmentation we are able to decrease our log-likelihood down to 0.321. This place the method in the position 117 of 2275 teams (5.1%) of the competition's public leaderboard.

Table 4.3.
The log-likelihood error of our method with and without boosting network and test augmentation.

Method	**Log-likelihood**
Baseline	0.364
+ Boosting Network	0.341
+ Test Augmentation	0.321

5. SATELLITE IMAGE MANIPULATION DETECTION

5.1 Overview

Satellite imagery is used in a wide range of applications such as regional infrastructure levels assessment [155, 156], agricultural crops classification [157, 158], forest characterization [159], scene classification [160, 161], soil moisture estimation [162, 163] and meteorological analysis, including precipitation prediction [164], thunderstorm detection [165] and wind speed and direction estimation [166]. These applications are possible thanks to the exponentially growing number of commercial satellites [167] (with many of those having imaging capabilities). Many image datasets captured by satellites are available to the public [168–170], such as Planet Labs or the European Space Agency image datasets [171, 172].

Editing tools like GIMP [116] or Photoshop [115] can be used to forge and manipulate satellite images in a realistic manner. Furthermore, manipulation generation can be automated by using machine learning techniques [173], removing the need for manual editing. Such manipulation methods, combined with the ease of sharing data on the internet, can difficult the institutions and companies that make use of images captured by satellites. Indeed, several instances of manipulated images have surged in recent years, including the nighttime flyovers of India during the Diwali festivals [174], the Malaysia Airlines Flight incident [175], and the images of the spliced fake Chinese bridge [176].

There is a wide range of manipulations techniques that can be used to forge satellite images. Some examples include splicing [177] (cropping and pasting regions from different image sources), copy-move [118] (cropping and pasting regions within the same image), shadow removal [119], and machine learning-based forgeries, often generated using Generative Adversarial Networks (GANs) [173]. Multiple methods to detect image manipulations have been proposed in recent years [178–180]. However, these methods are typically designed for images captured with consumer cameras and fail with images from other imaging devices, such as satellite imagery, with different compression schemes, post-processing, sensors, and color channels. Therefore, the detection of manipulations within satellite imagery still remains an unsolved problem that requires the development of new

detection techniques that are accurate regardless of the nature of the manipulations and image capturing technology.

In this work, we show how PixelCNN [181] and Gated PixelCNN [182], two generative autoregressive models, can be used to detect pixel-level manipulations. These neural networks, commonly used to generate new images, can model the distribution of a pixel given a set of previously seen pixels (neighboring pixels). These neural networks can assign a conditional likelihood value to a given pixel, and in turn, a likelihood value to a complete image. Through sampling from the pixel distribution, new images can be generated in a sequential fashion. Furthermore, manipulated pixels can be detected by selecting the pixels with a low likelihood assigned by the neural network. By averaging the likelihood estimated by an ensemble of multiple networks, the method is able to obtain a more accurate manipulation localization. Figure 5.1 presents the proposed ensemble where multiple networks process the input image and its flipped and rotated versions. Then, all predictions are averaged in order to obtain a robust prediction. Finally, we evaluate the localization precision of the presented method using a dataset composed of images with splicing forgeries, first introduced in [183].

The chapter is organized as follows. In section 5.2 we present previous work on manipulation detection and autoregressive models. In section 5.3 we describe the dataset composed by images captured by a satellite. In section 5.4 we describe the presented method. In section 5.5 we show the experimental results.

5.2 Related Work

Many techniques to detect a wide range of image manipulations have been previously presented. Some examples include techniques that detect manipulations by using embedded meta-data [125], finding double-JPEG compression artifacts [184], using neural networks with domain adaptation [185], using deepfakes detection neural networks [186] or using saturation cues [187]. Several methods have proved to accurately detect spliced objects within images captured with consumer-level cameras [177, 188]. The method pre-

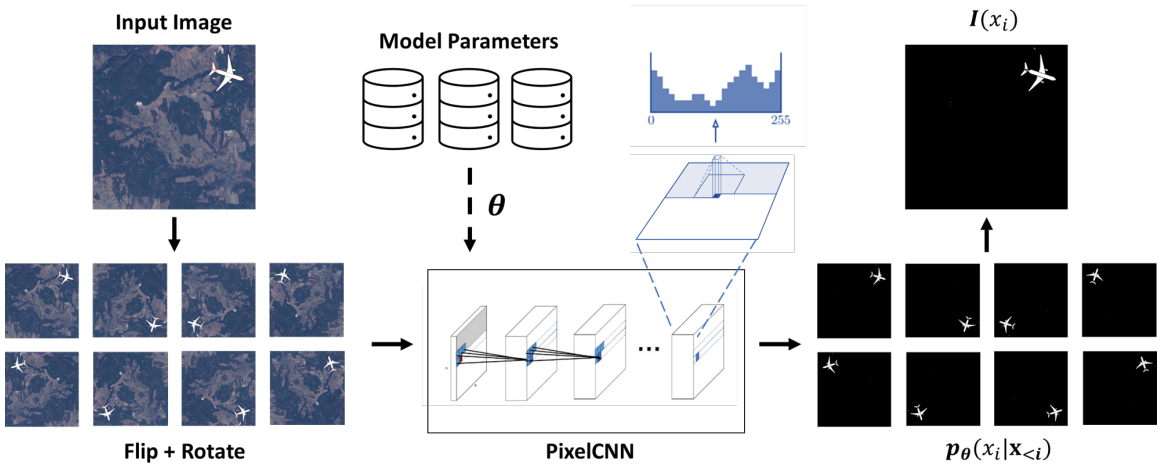

Fig. 5.1. Proposed ensemble of PixelCNNs: multiple models process the input image after applying flip and rotations. The prediction of every network is averaged obtaining a final robust and accurate likelihood estimate for each pixel of the image.

sented in [177] extracts a fingerprint from the camera model used to capture the image in order to suppress the scene content and enhance camera model-related artifacts. The method presented in [188] makes use of a feature-based technique that can detect splicing in images without any prior information of the nature of the manipulations by detecting traces left locally by processing steps within the capturing device. Unfortunately, many of these methods perform poorly when applied to satellite imagery. The image acquisition process differs between consumer cameras (including smartphone cameras) and satellites: different sensor technologies and post-processing steps such as orthorectification, radiometric corrections, and compression are used. Because of these differences, methods designed for consumer cameras do not transfer properly to satellite imagery.

Recently, multiple methods designed to detect forgeries in satellite imagery have been introduced. These include methods using hand-crafted features, like watermarking-based techniques [189], and data-driven machine learning-based approaches including supervised [190] and unsupervised [183, 191, 192] methods. While supervised methods tend to perform better, they might not generalize well to types of manipulations that were not present in the training set. Therefore, unsupervised methods, which don't make use of manipulated data during training, are preferred. The supervised method presented in [190] makes use of a conditional GAN [193] to detect and localize splicing forgeries in satellite images by estimating a forgery mask. The work introduced in [191] is based on a GAN that encodes patches from the input image into a low dimensional vector that is later used by a one-class support vector machine (SVM) to detect if a patch contains forgeries or not. The method presented in [192], named Sat-SVDD, is a kernel-based one-class classification method that detects splicing forgeries by using a modified Support Vector Data Description (SVDD) [194]. The SVDD encodes each patch from the original images (without manipulations) to a latent space within a hypersphere. During testing, the latent vectors that are placed outside the hypersphere are considered as patches containing a forgery. The method in [183] makes use of a deep belief network (DBN) [195] composed of two stacked layers of restricted Boltzmann machines (RBM) [196] parametrized with uniform distributions. The deep belief network is used to reconstruct patches extracted from the image.

Then, the reconstruction error is used to detect if manipulations are present: patches with a reconstruction error higher than a threshold are considered as forgeries.

In this work, we use generative autoregressive models, specifically PixelCNN [181] and Gated PixelCNN [182], which are described in the following sections. Many autoregressive generative models have been presented in recent years [181, 197–199]. Autoregressive models are able to estimate the distribution of an image by estimating the conditional distribution of each pixel. The distribution of each pixel is estimated given its neighboring pixels. Then, the distribution of an image can be expressed as the product of the conditional distributions. These models make use of masked convolutions in order to respect autoregressive constraints: each pixel is reconstructed only from previous pixels in a given ordering. PixelCNN and its recurrent-based counterpart PixelRNN [181], showed that autoregressive modeling can be successfully used to generate new images. Many variations have been presented such as Gated PixelCNN [182], PixelCNN++ [200], PixelSNAIL [201]. Furthermore, the same approach has been extended to video modeling in Video Pixel Network (VPN) [202], variational autoencoders in PixelVAE [203] and PixelVAE++ [204], to generative adversarial networks in PixelGAN [205] and to Markov random fields in PixelMRF [206].

Some works have studied the capability of likelihood models to detect outliers. The work presented in [207] makes use of the Watanabe-Akaike Information Criterion (WAIC) [208] to detect outliers. The work in [209] normalizes the likelihood estimate of an image with a measure of complexity to detect outliers. However, most of these approaches focus on image-level out of distribution (OoD) estimates (also referred to as anomaly detection), and likelihood methods to detect pixel-level manipulations remain unexplored.

5.3 Dataset

In order to train and evaluate our method, we use the dataset first introduced in [183]. This dataset is composed of orthorectified satellite images including regions of Slovenia taken from the Sentinel program [210]. The images have a resolution of 1000×1000 pixel.

We use a subset of the dataset consisting of 98 original images (without manipulations) for training and 500 manipulated images with their corresponding ground truth masks for testing. Each manipulated image has one spliced object randomly selected among 19 different objects, including clouds, planes, smoke, and drones. The objects are spliced with different locations, rotation angles and sizes including 16×16, 32×32, 64×64, 128×128, and 256×256 pixels. Figure 5.2 presents some examples of the dataset.

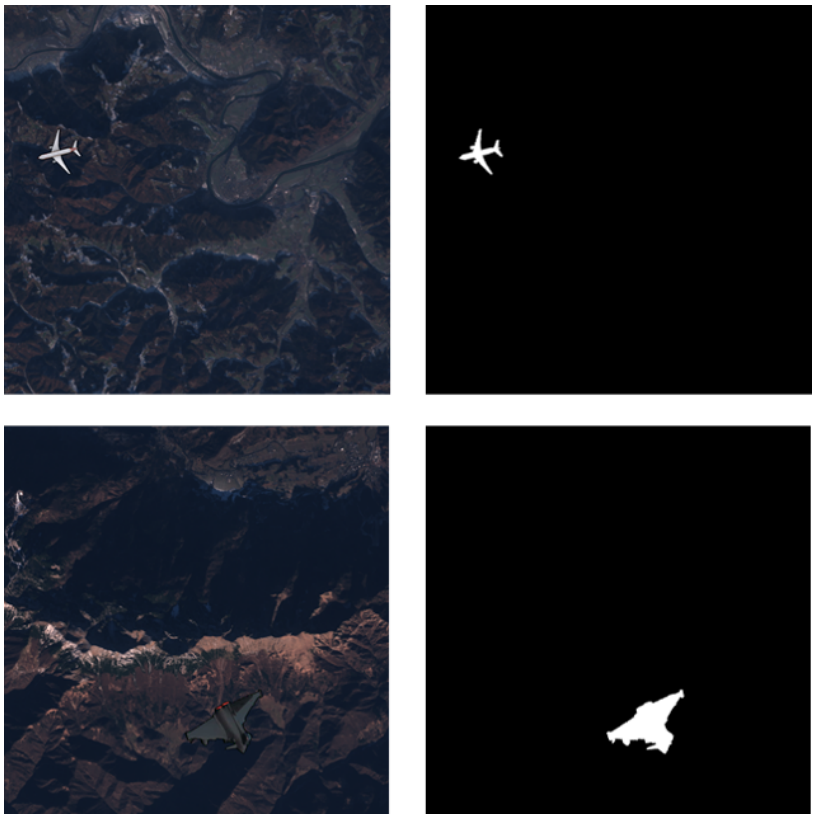

Fig. 5.2. Examples of images (left) from the dataset and its corresponding manipulation masks (right).

5.4 Proposed Method

5.4.1 Autoregressive Models

PixelCNN [181] and Gated PixelCNN [182] are neural networks composed by multiple fully-convolutional residual layers and are trained to model the distribution of an image $\mathbf{x}$ as the product of the conditional distributions of every pixel x_i:

$$p(\mathbf{x}) = \prod_{i=1}^{L} p(x_i|x_1, ..., x_{i-1}) \tag{5.1}$$

Where $\mathbf{x}$ is an image of L pixels and x_i is the ith pixel of the image. The predicted distribution of every pixel x_i is conditional to the previous pixels $x_1, ..., x_{i-1}$ in a raster scan order: row by row and pixel by pixel within every row (left to right and top to bottom).

In RGB images, each color channel (R, G, B) is modeled successively: first the red channel, then the green channel conditioned to the red, and finally the blue channel conditioned to the red and green. Therefore, the conditional probability of an RGB pixel is as follows:

$$\begin{gathered} p(x_i|\mathbf{x}_{<i}) = p(x_i|x_1, ..., x_{i-1}) = \\ p(x_{i,R}|\mathbf{x}_{<i})p(x_{i,G}|\mathbf{x}_{<i}, x_{i,R})p(x_{i,B}|\mathbf{x}_{<i}, x_{i,R}, x_{i,G}) \end{gathered} \tag{5.2}$$

The autoregressive constraints are achieved by masking the convolutions accordingly, both within spatial dimensions and within features maps. The use of convolutions allows the network to perform the likelihood predictions in parallel during training and testing but the image generation remains a sequential process.

While our method is designed for RGB images, it is common for satellites to capture multi-spectral images containing more than 3 channels. The presented approach can be easily extended to any number of channels by assigning some arbitrary order within the channels and estimating the conditional probability as follows:

$$p(x_i|\mathbf{x}_{<i}) = \prod_{j=1}^{C} p(x_{i,j}|\mathbf{x}_{<i}, x_{i,1}, ..., x_{i,j-1}) \tag{5.3}$$

Where $x_{i,j}$ is the ith pixel from the jth channel of an image with a total of K pixels and C channels.

PixelCNN, and some of its variations, models the conditional probability $p(x_i|\mathbf{x}_{<i})$ as a multinomial (categorical) distribution through a softmax layer where each channel within the image can take a value from 0 to 255. The network takes as input an image with $N \times M \times 3$ dimensions (with $N \times M = L$) and outputs a prediction with dimension $N \times M \times 3 \times 256$. While the original PixelCNN is designed to work with 8-bit images, the method can be adapted to work with images with different bit depths by properly changing the range of values that the softmax layer can take. For example, when working with 11-bit images, the softmax layer should output values from 0 to 2047. This is especially useful for satellite imagery as many datasets have bit depths higher than 8-bits.

5.4.2 Generative Ensembles

We can obtain more accurate and robust predictions by combining multiple networks within an ensemble. We average the predictions of multiple networks with different parameters and scan orderings. In order to obtain multiple model parameters, we save the parameters of the network at different epochs during the training process. The parameters θ of the network at each training epoch can be seen as an approximate proxy of posterior samples of $p(\theta|\mathcal{D})$ (the distribution of the model parameters given the training set $\mathcal{D}$). To use different scan orderings during the autoregressive modeling (the order in which neighboring pixels are observed) we can apply different masks to the convolutional filters, or equivalently, rotate and flip the input image. Figure 5.3 shows the 8 different orderings used and the transformations (flip and rotate) applied to the input image and the corresponding convolutional mask to obtain equivalent results.

The average of the prediction of multiple networks with different parameters and scan order can be understood as a Monte Carlo approximation of the marginal likelihood of each pixel $p(x_i)$, where the effect of the model parameters θ and scan ordering $\mathbf{x}_{<i}$ are smoothed out:

$$p(x_i) = \mathrm{E}_{\theta,\mathbf{x}_{<i}}[p_\theta(x_i|\mathbf{x}_{<i})] \approx \frac{1}{K}\sum_{\omega\in\Omega} p_\theta(x_i|\mathbf{x}_{<i}) \tag{5.4}$$

Where ω are samples of model parameters and scan ordering pairs $(\theta, \mathbf{x}_{<i})$ from a set Ω of size $|\Omega| = K$. In other words, the average of the prediction of K networks with different parameters and scan orderings are used to approximate the marginal likelihood $\hat{p}(x_i) \approx p(x_i)$. In order to detect manipulations, we can use the negative log-likelihood, which in turn is the information content (or Shannon information) quantity $I(x_i) = -\log p(x_i)$, approximated as:

$$\hat{I}(x_i) = -\log[\frac{1}{K}\sum_{\omega\in\Omega} p_\theta(x_i|\mathbf{x}_{<i})] \tag{5.5}$$

A pixel is considered to be manipulated if $\hat{I}(x_i) > T$, where T is experimentally selected. Ideally, the model will assign high likelihood values (and thus small information values) to pixels that have not been manipulated, and small likelihood (and high information) values to manipulated pixels.

5.4.3 Training and Testing Setup

In this work, we use a PixelCNN composed of 7 residual blocks and a Gated PixelCNN composed of 6 gated blocks. We train the networks with the Adam optimizer with a learning rate of 0.001. During the training process, we randomly rotate 0, 90, 180, or 270 degrees and horizontally flip the images. We train the network for 1000 epochs and we store the model parameters every 20 epochs.

During testing, the ensemble is composed of $K = 50$ different models. We select 50 model parameters uniformly distributed from epoch 30 to epoch 1000 of the training process. For each model parameters we use a scan ordering randomly selected from the 8 different scan orderings shown in figure 5.3.

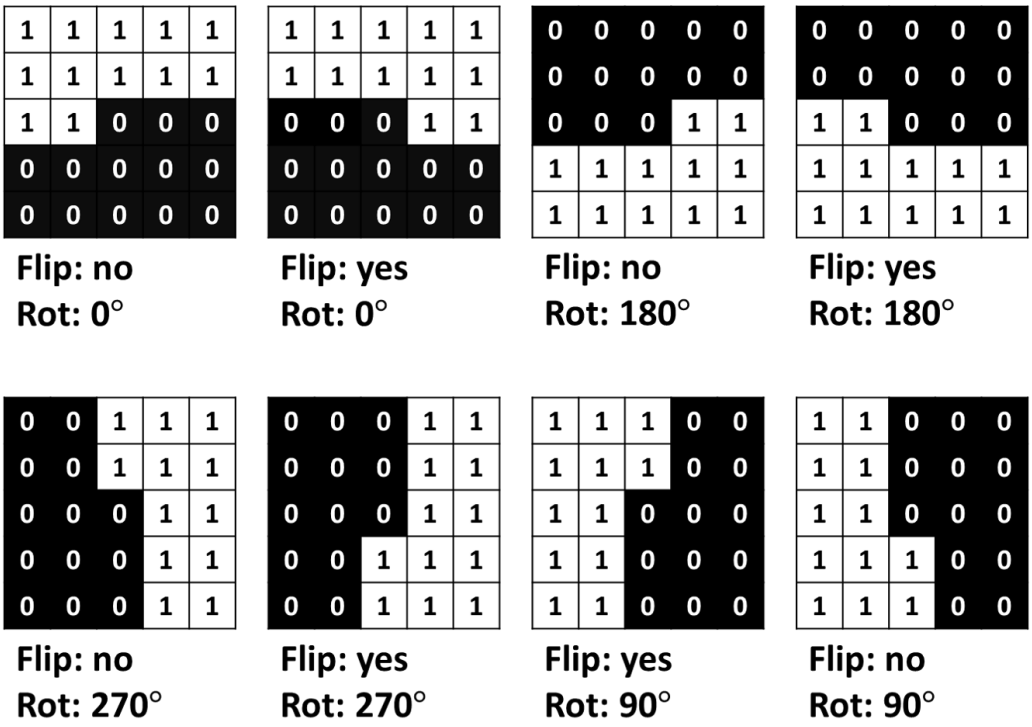

Fig. 5.3. Different convolutional masks and the respective transformation (flipping and rotation) performed to the input image to obtain the equivalent effect.

5.5 Experimental Results

We train and evaluate our method with the dataset presented in Section 5.3. The ensemble of networks is trained only with original images and no manipulated images are used during the training process. In order to evaluate the localization performance of the presented method, we compute the area under the curve (AUC) and Precision/Recall (P/R) curves by changing the threshold T used to the estimated manipulation mask. Table 5.1 presents our results compared with previous methods. Different Precision/Recall are shown for each of the different sizes of the splice objects used. For example, P/R_{32} is the AUC of the P/R curve for manipulated images with spliced objects of size 32×32.

Our experimental results show that the generative ensemble of PixelCNNs and Gated PixelCNNs outperform previously presented methods. While most of the methods fail

to detect objects smaller than 64×64, the presented generative ensembles are able to properly detect small forgeries. While methods such as [191], [192], and [183] produce estimates within patches of the input image, and therefore lacking enough resolution to detect small forgeries, PixelCNN and Gated PixelCNN process the whole image in a fully-convolutional manner and detects manipulations in a pixel-level. Figure 5.4 provides some visual examples of the estimated manipulation mask (the information content $\hat{I}(x_i)$) for the presented methods. The estimated manipulation mask shows that the method is able to properly distinguish between areas containing splicing manipulations and areas without forgeries.

We can observe that Gated PixelCNN provides more accurate results than the regular PixelCNN network, especially for objects smaller than 64×64 pixels. These results are aligned with previous works [182] which have shown that Gated PixelCNN is able to model the image distribution of the training images more accurately (with a lower negative log-likelihood score) than PixelCNN.

Table 5.1.
AUC scores (%) of the P/R curves for the localization task. The subscript ($P/R_{\times}$) denotes the manipulation size.

Method	P/R_{16}	P/R_{32}	P/R_{64}	P/R_{128}	P/R_{256}	Average
Noiseprint [177]	0.0	0.1	2.5	4.6	7.8	3.0
Yarlagadda *et al* [191]	0.0	0.3	2.5	18.3	37.8	11.7
Splicebuster [188]	0.0	0.5	7.8	31.2	48.5	17.6
Sat-SVDD [192]	0.1	1.4	18.1	34.4	55.7	21.9
UU-DBN [183]	7.5	13.3	31.7	40.5	48.8	28.4
Generative Ensemble (PixelCNN)	37.6	44.6	56.2	65.3	**75.6**	55.9
Generative Ensemble (Gated PixelCNN)	**46.3**	**53.8**	**61.1**	**65.6**	72.8	**59.9**

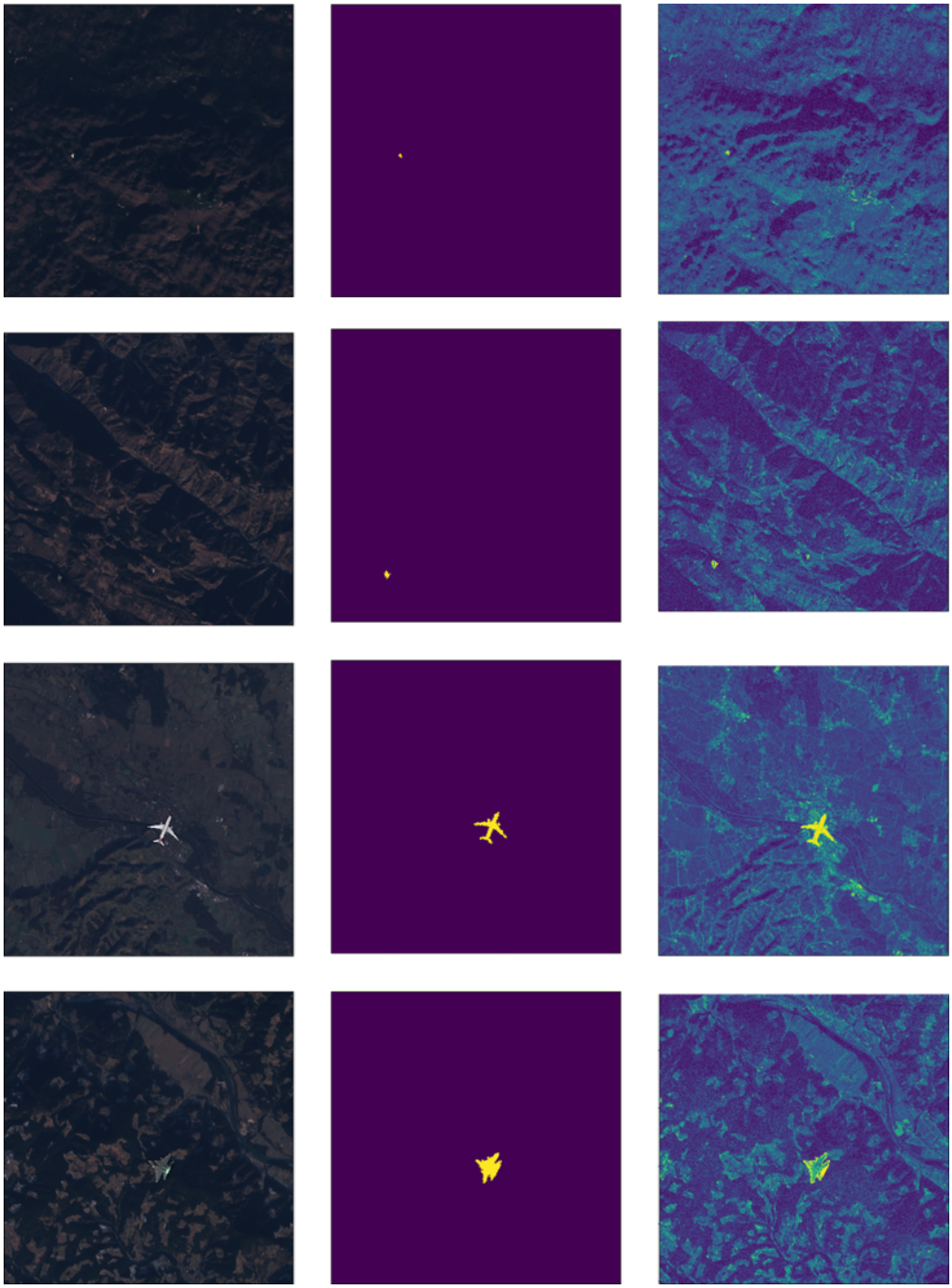

Fig. 5.4. Example of input images (left), manipulation masks (center), and estimated negative loglikelihood $\hat{I}(x_i)$ (right).

5.6 Splicing, Background, And Likelihood Analysis

We perform an analysis to study how the presented method performs when faced with images containing spliced objects and backgrounds with different visual features. To do so, we perform two experiments: First, we evaluate the ensemble of Gated PixelCNNs with subsets of the test set divided by the visual complexity of the background, and second, we evaluate the proposed method with the test set divided by the appearance of the spliced objects. Additionally, we perform both experiments using two different functions of the likelihood estimates.

In order to detect manipulations, we compare two functions of the estimated conditional likelihood. The first function is the approximate information content, described in previous sections:

$$\hat{I}(x_i) = -\log[\frac{1}{K}\sum_{\omega\in\Omega} p_\theta(x_i|\mathbf{x}_{<i})] \approx -\log \mathrm{E}[p_\theta(x_i|\mathbf{x}_{<i})] = -\log p(x_i) \tag{5.6}$$

Additionally, we evaluate another likelihood metric, which we refer to as $\hat{J}(x_i)$, where the averaging is performed in the logit space (after the $\log$ operation), instead of in the probability space (as in $\hat{I}(x_i)$). Furthermore, the standard deviation is subtracted to obtain a more robust likelihood-based metric:

$$\hat{J}(x_i) = L(x_i) - S(x_i) \tag{5.7}$$

Where $L(x_i)$, presented in equation 5.8, is the average of negative loglikelihoods, and $S(x_i)$, presented in equation 5.9, is the standard deviation of negative loglikelihoods.

$$L(x_i) = \frac{1}{K}\sum_{\omega\in\Omega} -\log p_\theta(x_i|\mathbf{x}_{<i}) \approx -\mathrm{E}[\log p_\theta(x_i|\mathbf{x}_{<i})] \tag{5.8}$$

$$S(x_i) = \sqrt{\frac{1}{K-1}\sum_{\omega\in\Omega}(-\log p_\theta(x_i|\mathbf{x}_{<i}) - L(x_i))^2} \approx \sqrt{\mathrm{Var}[\log p_\theta(x_i|\mathbf{x}_{<i})]} \tag{5.9}$$

Note that for a random variable a, $\mathrm{E}[\log a] \approx \log(\mathrm{E}[a]) - \frac{\mathrm{Var}[a]}{2\mathrm{E}[a]^2}$. Therefore, for small values of $S(x_i)$, we have that $\hat{I}(x_i) \approx \hat{J}(x_i)$. Using metrics based on averaged negative

loglikelihood estimates for out-of-distribution has been previously explored in previous work including [207].

The first experiment evaluates the detection performance of the proposed method with different types of background images. We divide the test set in 3 subsets composed by background images that are simple (lack texture and have small contrast), regular (images with some texture and small regions with high contrast), and complex (images highly textured and with high contrast regions). For each subset of images, we compute the AUC of the Precision/Recall curves, shown in table 5.2. The results presented in the table show that the performance decreases as the complexity of the background increases. The main reason for this decrease in the AUC score is an increase of false positives where the network detects areas with high contrast or texture as possible manipulations. Results show that the $\hat{I}(x_i)$ metric provides better manipulation detection results.

Figure 5.5 shows examples of images with simple (upper rows, 1), regular (middle rows, 2), and complex backgrounds (bottom rows, 3). For each input image (a), the ground truth manipulation mask (b) is shown. Additionally, the figure shows the estimated $\hat{I}(x_i)$ (c), its thresholded version $\hat{I}(x_i) > T_i$ (d), the estimated $\hat{J}(x_i)$ (e), and its thresholded version $\hat{J}(x_i) > T_j$ (f). T_i and T_j are selected per each image in order to obtain the optimal F-1 score. We can observe that while $\hat{I}(x_i)$ and $\hat{J}(x_i)$ have different dynamic ranges ($\hat{J}(x_i)$ tends to provide estimates with a larger dynamic range), after thresholding both estimates are similar. The figures show that the manipulations are accurately detected in images containing a simple background, however, the network struggles to properly detect manipulations in complex background images. Specifically, the proposed method properly detects the boundaries of manipulations but fails in differentiating between the inside and outside of the spliced object, as shown in the examples of the middle and bottom rows.

The second experiment consists of evaluating the proposed method with subsets of images containing different types of objects. We divide the testing set in four different subsets: images containing gray planes, gray clouds, white planes, and yellow clouds. We compute the AUC of the Precision/Recall curve, shown in table 5.3, and show some visual

Table 5.2.
AUC scores (%) of the P/R curves for the localization task with testing images containing backgrounds of different complexity.

Method	Simple	Regular	Complex
$\hat{J}(x_i)$	69.6	62.6	44.5
$\hat{I}(x_i)$	**75.1**	**66.7**	**45.2**

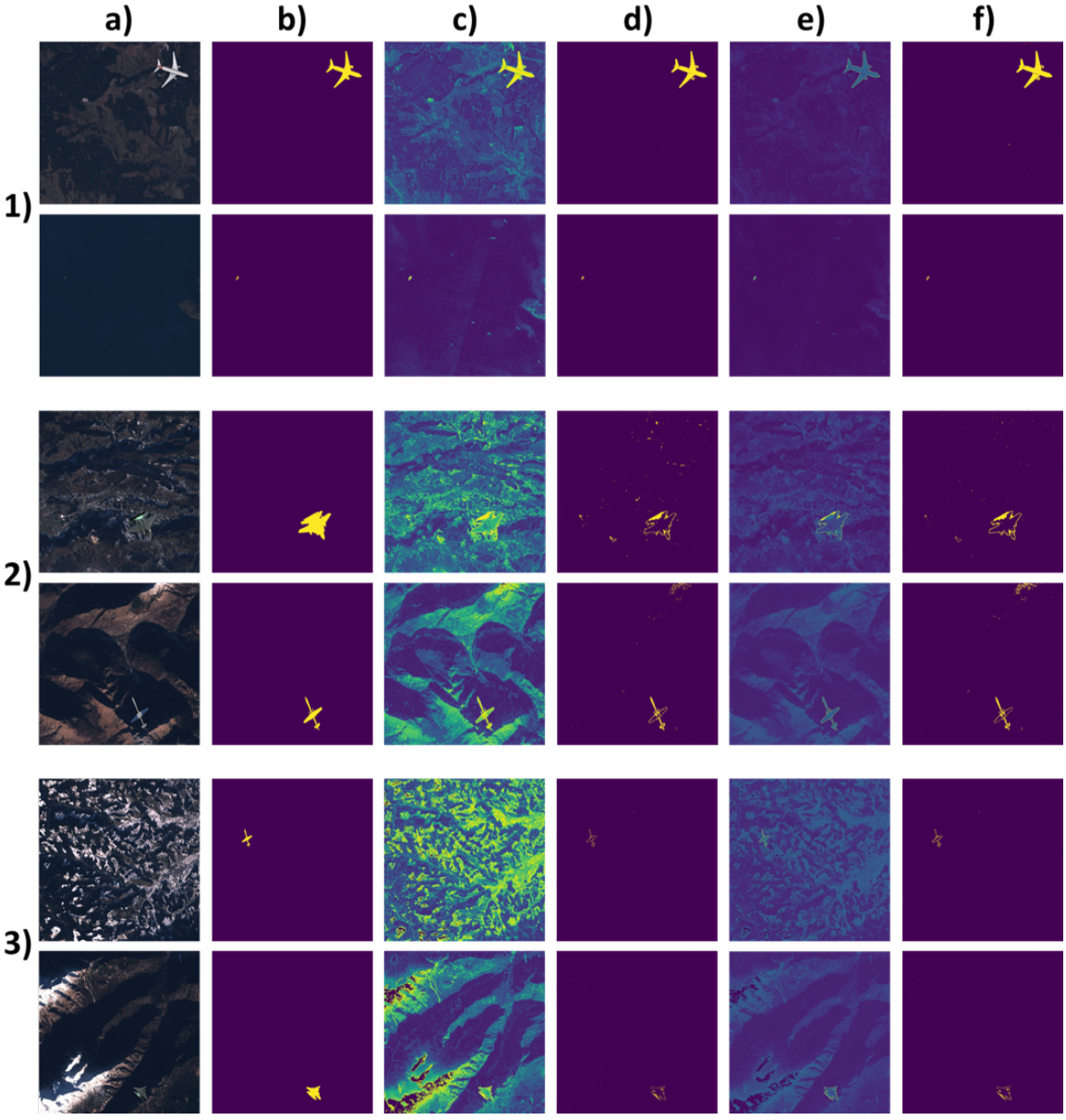

Fig. 5.5. Example of manipulated images containing backgrounds with different level of complexity: simple background (1 - upper rows), regular background (2 - middle rows), and complex background (3 - bottom rows). From left to right, the figure includes: (a) input image, (b) ground truth manipulation mask, (c) $\hat{I}(x_i)$, (d) $\hat{I}(x_i) > T_i$, (e) $\hat{J}(x_i)$, and (f) $\hat{J}(x_i) > T_j$.

examples in figure 5.6, and 5.7. The AUC values from table 5.3 show that the performance of the method varies largely depending on the type of object present in the image. Objects composed by darker colors, such as gray planes and gray clouds, tend to have a visual appearance similar to the background images. On the other hand, objects like white planes or yellow clouds are more visually distinctive and easier to detect.

Figure 5.6 and figure 5.7 show multiple examples of detected manipulations. Figure 5.6 shows examples of images containing gray clouds and gray planes. The images show that the proposed method is capable to properly detect the edges of the spliced objects, however, it tends to not properly classify the pixels inside the spliced objects as manipulated. Figure 5.7 shows examples of manipulated images containing white planes and yellow clouds which are more visually distinct from the background images. The images show how the network is capable to properly detect all the pixels of the spliced object as manipulated pixels. These visual examples are aligned with the quantitative results shown in table 5.3.

Both experiments show that autoregressive-based methods can accurately detect edges of splicing manipulations. However, the proposed method struggles to classify the pixels inside the splicing manipulation as forged pixels when faced with challenging backgrounds or spliced objects. Future work could include using the proposed method as an initial step of edge manipulation detection, combined with a post-processing step to accurately segment each pixel of the spliced object.

Table 5.3.
AUC scores (%) of the P/R curves for the localization task with testing images containing different spliced objects.

Method	Gray Planes	Gray Cloud	White Planes	Yellow Clouds
$\hat{J}(x_i)$	$\mathbf{31.9}$	47.1	66.9	89.7
$\hat{I}(x_i)$	30.4	$\mathbf{49.7}$	$\mathbf{69.1}$	$\mathbf{94.1}$

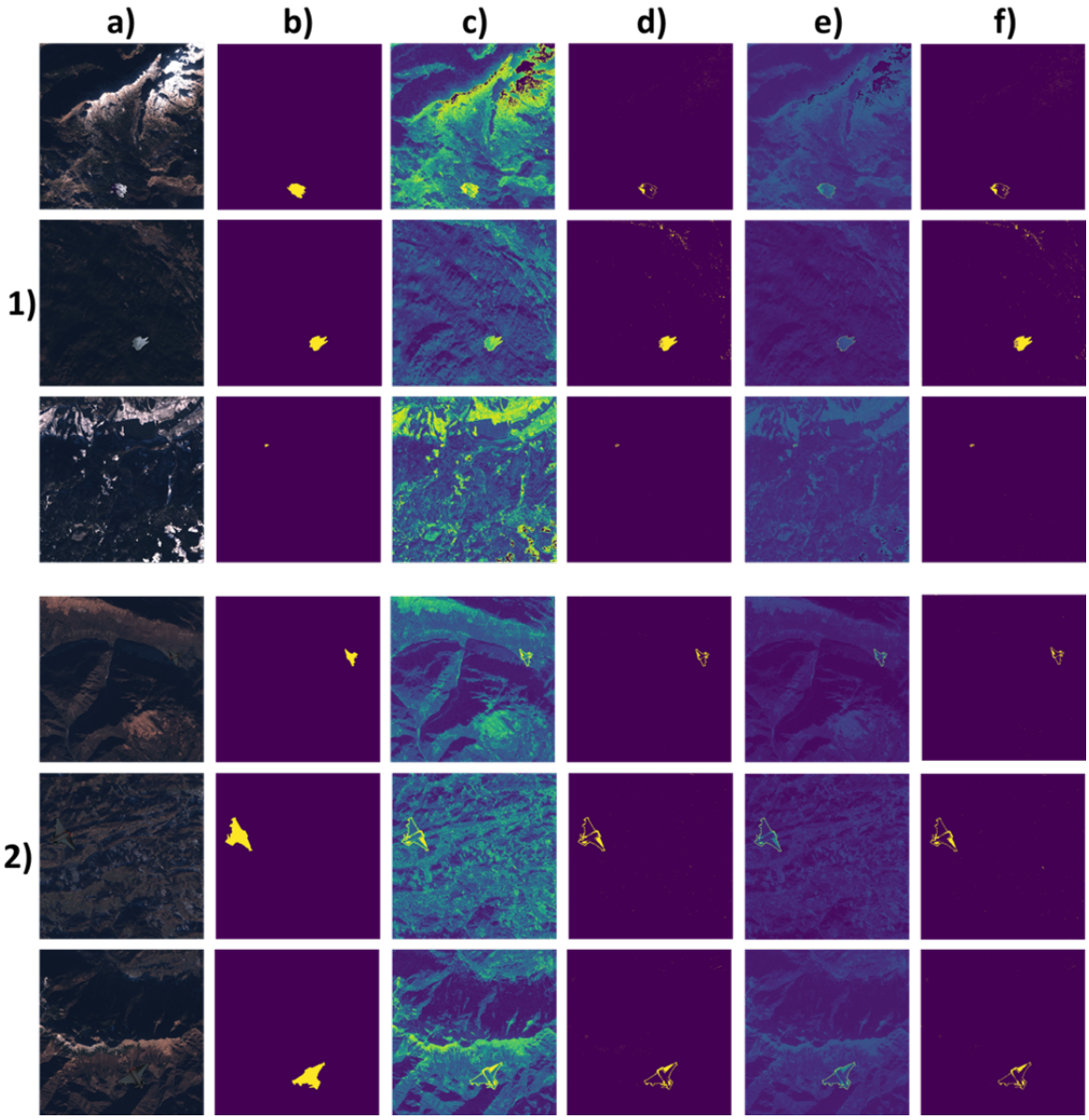

Fig. 5.6. Example of manipulated images containing (1) gray clouds, and (2) gray planes. From left to right, the figure includes: (a) input image, (b) ground truth manipulation mask, (c) $\hat{I}(x_i)$, (d) $\hat{I}(x_i) > T_i$, (e) $\hat{J}(x_i)$, and (f) $\hat{J}(x_i) > T_j$.

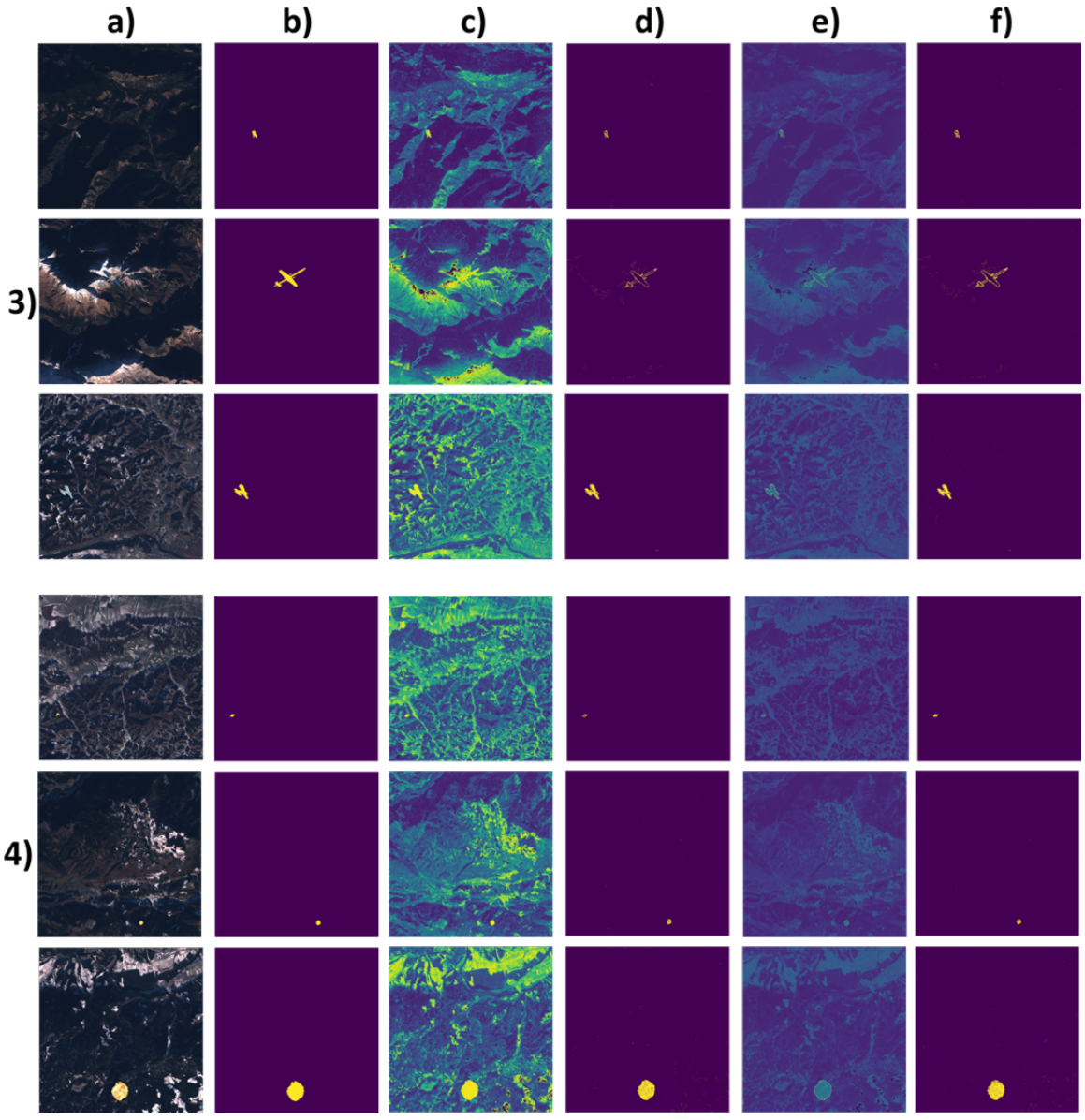

Fig. 5.7. Example of manipulated images containing (3) white planes, and (4) yellow clouds. From left to right, the figure includes: (a) input image, (b) ground truth manipulation mask, (c) $\hat{I}(x_i)$, (d) $\hat{I}(x_i) > T_i$, (e) $\hat{J}(x_i)$, and (f) $\hat{J}(x_i) > T_j$.

6. SUMMARY AND FUTURE WORK

6.1 Overview

In this thesis, we have introduced multiple problems in the field of multimedia analytics and solutions based on new machine learning methods.

In chapter 2, we stated the problem of logo detection and presented multiple solutions. We first showed that object detection networks such as Faster R-CNN can be successfully adapted to detect logos in the wild. Additionally, we showed that by combining object detection networks with image classification networks such as DenseNet, the detection accuracy can be further improved. In the scenario where the number of training samples is small or non-existent, image synthesis techniques can be applied to create new training samples. We presented two different techniques to create new images. The first technique consists of randomly splicing logo images in a background image. The second technique improves upon the first one and extracts information of the background image in order to splice logo images in a realistic manner. In combination with image synthesis, we showed that bootstrapping techniques can be used to further increase the logo detection accuracy. As logos are largely found on the internet, using weakly-labeled images from the internet is a useful approach for logo detection.

However, there is still a large gap in the accuracy of object detection methods trained with synthetic images and object detection methods trained with real images. Therefore, more research needs to be done in image synthesis techniques, in order to create realistic images with the same statistical properties as the real-world images. It is important to analyze which aspects of the image synthesis process (i.e. location of logos, distortions, and transformations applied to the foreground and background images, statistical variation...) are determining factors when it comes to train networks that generalize well. Methods that find image synthesis pipelines in a data-driven manner instead of ad-hoc hand-crafted pipelines could generate more realistic images that in turn might provide a significant increase in detection accuracy. Furthermore, semi-supervised, self-supervised, and unsupervised techniques (e.g. triplet-loss, Siamese networks...) coupled with few-shot and one-shot learning methods, should be included in the training process in order to fully

benefit of a large number of unlabeled images and videos containing logos available on the internet.

In chapter 3, we presented the problem of pose estimation. We introduced the Multi-View Matching Network (MV-Net) and the Single View Matching Network (SV-Net) to perform pose estimation and tracking. The pair of networks provides an initial estimate of the pose and then it refines the pose in an iterative manner. The same iterative process can be used to track the pose within a video. Additionally, we showed how photorealistic rendering techniques can be used to generate datasets that can be used to train the neural networks, removing the need for manually annotating the 6D pose of images.

While these techniques provide promising results, most AR applications require to work in real-time and typically in smartphone devices. These devices have memory and computing limitations and require low-weight highly optimized neural networks. Therefore, there is a need for neural networks that are highly accurate while being compact and fast. Neural architecture search (NAS) has proved to be highly successful in image classification and object detection task. Future work includes exploring NAS techniques to designed novel neural networks for pose estimation. Furthermore, neural network-based techniques to render images from 3D information could be coupled with 6D pose estimation methods in order to further increase the estimation accuracy.

In chapter 4, we presented a new method to detect face manipulations within videos. We showed that combining convolutional and recurrent neural networks achieves high detection accuracies on the DFDC dataset. We described a method to automatically weight different face regions and showed that boosting techniques can be used to obtain more robust predictions. The method processes videos quickly (in less than eight seconds) with a single GPU. Although the results of our experiments are promising, new techniques to generate deepfake manipulations emerge continuously. The modular nature of the proposed approach allows for many improvements, such as using different face detection methods, different backbone architectures, and other techniques to obtain a prediction from features of multiple frames.

Furthermore, this work focuses on face manipulation detection and dismisses any analysis of audio content which could provide a significant improvement of detection accuracy in future works. Incorporating adversarial losses during training and making use of discriminators from GANs could improve the manipulation detection accuracy. Additionally, NAS techniques could be applied to obtain more compact and accurate architectures.

In chapter 5, we presented a new method to detect manipulations within satellite images. The wide range of manipulations that can be applied to images and the large diversity of imaging technologies used in satellites makes their detection a challenging problem that still remains unsolved. We introduced an unsupervised splicing detection method. The method consists of an ensemble of generative autoregressive models that estimates the pixel distribution of the image. The method is capable to accurately detect manipulated pixels by selecting the regions of the image where the network predicts a low likelihood value. The presented method is fully unsupervised and doesn't use any prior knowledge from the applied manipulation during training. Our experiments show that the localization accuracy of our method surpasses the previous works and shows that generative models, specially autoregressive-based networks, provide a promising approach to detect pixel-level manipulations.

Future work includes exploring different generative approaches in order to detect manipulations. Methods that estimate the exact likelihood (autoregressive models, flow-based models, etc), an approximate likelihood (variational autoencoders) or an implicit likelihood (generative adversarial models) are well fitted to detect manipulations within images. By combining multiple approaches the manipulation detection accuracy might be further improved. For example, autoregressive likelihood models could be improved by including adversarial losses.

6.2 Complete List Of Publications

Following is an exhaustive list of the publications in which I have been involved during my Ph.D. studies at Purdue University:

1. **D. Mas Montserrat**, Q. Lin, J. Allebach, and E. J. Delp, “Training object de- tection and recognition CNN models using data augmentation”, Proceedings of the IS&T International Symposium on Electronic Imaging, January 2017, Burlingame, CA.

2. **D. Mas Montserrat**, Q. Lin, J. Allebach, and E. J. Delp, “Logo detection and recognition with synthetic images”, Proceedings of the IS&T International Symposium on Electronic Imaging, January 2018, Burlingame, CA.

3. J. Choe, **D. Mas Montserrat**, A. J. Schwichtenberg, and E. J. Delp, “Sleep analysis using motion and head detection”, Proceedings of the IEEE Southwest Symposium on Image Analysis and Interpretation, April 2018, Las Vegas, NV.

4. **D. Mas Montserrat**, Q. Lin, J. Allebach, and E. J. Delp, “Scalable logo detection and recognition with minimal labeling”, Proceedings of the IEEE International Conference on Multimedia Information Processing and Retrieval, pp. 152-157, April 2018, Miami, FL.

5. S. K Yarlagadda, D. Guera, **D. Mas Montserrat**, F. M. Zhu, E. J. Delp, P. Bestagini and S. Tubaro, “Shadow removal detection and localization for forensics analysis”, Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, May 2019, Brighton, United Kingdom.

6. **D. Mas Montserrat**, J. Chen, Q. Lin, J. Allebach, and E. J. Delp, “Multi-View Matching Network for 6D Pose Estimation”, Proceedings of the IEEE Computer Vision and Pattern Recognition Workshops, June 2019, Long Beach, CA.

7. Q. Lin, **D. Mas Montserrat**, J. Allebach, and E. J. Delp, “Selecting training symbols for symbol recognition”, WO2019152017A1, 2019-08-08, PCT/US2018/016211.

8. **D. Mas Montserrat**, C. Bustamante and A. Ioannidis, “Class-Conditional VAE-GAN for Local-Ancestry Simulation”, Machine Learning for Computational Biology at NeurIPS, December 2019, Vancouver, Canada.

9. J. Chen, **D. Mas Montserrat**, Q. Lin, E. J. Delp and J. P. Allebach, “Extra FAT: A photorealistic dataset for 6D object pose estimation”, Proceedings of the IS&T International Symposium on Electronic Imaging, January 2020, Burlingame, CA.

10. **D. Mas Montserrat**, A. Kumar, C. Bustamante and A. Ioannidis, “Addressing Ancestry Disparities in Genomic Medicine: A Geographic-Aware Algorithm”, International Conference on Learning Representations Workshops, April 2020, Addis Ababa, Ethiopia.

11. A. Kumar, **D. Mas Montserrat**, C. Bustamante and A. Ioannidis, “XGMix: Local-Ancestry Inference with Stacked XGBoost”, International Conference on Learning Representations Workshops, April 2020, Addis Ababa, Ethiopia **(Best Paper Award)**.

12. **D. Mas Montserrat**, C. Bustamante and A. Ioannidis, “LAI-Net: Local-Ancestry Inference With Neural Networks”, Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, May 2020, Barcelona, Spain.

13. J. Horváth, **D. Mas Montserrat**, H. Hao and E. J. Delp, “Manipulation Detection in Satellite Images Using Deep Belief Networks”, Proceedings of the IEEE Computer Vision and Pattern Recognition Workshops, June 2020, Seattle, WA.

14. **D. Mas Montserrat**, H. Hao, S. K. Yarlagadda, S. Baireddy, R. Shao, J. Horváth, E. Bartusiak, J. Yang, D. Guera, F. Zhu and E. J. Delp, “Deepfakes Detection with Automatic Face Weighting”, Proceedings of the IEEE Computer Vision and Pattern Recognition Workshops, June 2020, Seattle, WA.

REFERENCES

REFERENCES

[1] F. N. Iandola, A. Shen, P. Gao, and K. Keutzer, "Deeplogo: Hitting logo recognition with the deep neural network hammer," *arXiv:1510.02131*, 2015.

[2] "Pixel Intelligence," URL: www.developers.hp.com.

[3] "Meerkat," URL: www.meerkat.com.br.

[4] "Google Cloud," URL: www.cloud.google.com.

[5] "Logo Grab," URL: www.logograb.com.

[6] "A study on beer: logo detection and analysis on social media," URL: https://medium.com/@meerkat.cv/a-study-on-beer-logo-detection-and-analysis-on-social-media-9ab2dab0014c.

[7] "Twitter," URL: www.twitter.com.

[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *Proceedings of the Advances in Neural Information Processing Systems*, pp. 91–99, December 2015, Montreal, Canada.

[9] S. Romberg, L. G. Pueyo, R. Lienhart, and R. van Zwol, "Scalable logo recognition in real-world images," *Proceedings of the ACM International Conference on Multimedia Retrieval*, pp. 251–258, April 2011, Trento, Italy.

[10] A. Joly and O. Buisson, "Logo retrieval with a contrario visual query expansion," *Proceedings of the ACM International Conference on Multimedia Retrieval*, pp. 581–584, October 2009, Beijing, China.

[11] S. G. Hang Su, Xiatian Zhu, "Deep learning logo detection with data expansion by synthesising context," *Proceedings of the IEEE Winter Conference on Applications of Computer Science*, pp. 20–25, March 2017, Santa Rosa, CA.

[12] I. Goodfellow, Y. Bengio, and A. Courville, "Introduction," *Deep Learning*. Cambridge, MA: MIT Press, 2016, vol. 1, p. 20.

[13] 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, August 2015.

[14] D. Mas Montserrat, Q. Lin, J. Allebach, and E. J. Delp, "Training object detection and recognition CNN models using data augmentation," *Proceedings of the IS&T International Symposium on Electronic Imaging*, January 2017, Burlingame, CA.

[15] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587, 2014, Columbus, OH.

[16] R. Girshick, "Fast R-CNN," *Proceedings of the International Conference on Computer Vision*, pp. 1440–1448, December 2015, Santiago, Chile.

[17] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2980–2988, October 2017, Venice, Italy.

[18] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Reading text in the wild with convolutional neural networks," *International Journal of Computer Vision*, vol. 116, no. 1, pp. 1–20, January 2016.

[19] S. Hong, B. Roh, K.-H. Kim, Y. Cheon, and M. Park, "PVANet: Lightweight deep neural networks for real-time object detection," *arXiv:1611.08588*, 2016.

[20] H. Su, S. Gong, and X. Zhu, "Weblogo-2m: Scalable logo detection by deep learning from the web," *Proceedings of the IEEE International Conference on Computer Vision*, no. 8, pp. 20–25, October 2017, Venice, Italy.

[21] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4700–4708, June 2017, Honolulu, HI.

[22] D. G. Lowe, "Object recognition from local scale-invariant features," *Proceedings of the International Conference on Computer Vision*, pp. 1150–1159, September 1999, Kerkya, Greece.

[23] R. M. Haralick, K. Shanmugam, and I. Dinstein, "Textural features for image classification," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-3, no. 6, pp. 610–621, November 1973.

[24] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[25] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Proceedings of the Advances in Neural Information Processing Systems*, pp. 1097–1105, December 2012, Stateline, NV.

[27] H. Kaiming, Z. Xiangyu, R. Shaoqing, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *Proceedings of the European Conference on Computer Vision*, pp. 346–361, September 2014, Zurich, Switzerland.

[28] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, August 2013.

[29] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[30] L. Zhang, L. Zhang, and B. Du, "Deep learning for remote sensing data: A technical tutorial on the state of the art," *IEEE Geoscience and Remote Sensing Magazine*, vol. 4, no. 2, pp. 22–40, June 2016.

[31] D. Mas Montserrat, C. Bustamante, and A. Ioannidis, "LAI-Net: Local-Ancestry Inference With Neural Networks," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2020, Barcelona, Spain.

[32] M. Mas Montserrat, "Essays on Wealth Taxation, Avoidance and Evasion Among the Rich," February 2020.

[33] D. Mas Montserrat, C. Bustamante, and A. Ioannidis, "Class-Conditional VAE-GAN for Local-Ancestry Simulation," *Machine Learning in Computational Biology*, December 2019, Vancouver, Canada.

[34] C.-C. J. Kuo, "Understanding convolutional neural networks with a mathematical model," *Visual Communication and Image Representation*, vol. 41, pp. 406–413, November 2016.

[35] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, January 2015.

[36] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," *Proceedings of the European Conference on Computer Vision*, pp. 740–755, September 2014, Zürich, Switzerland.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Proceedings of the International Conference on Learning Representations*, May 2014, San Diego, CA.

[38] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *Proceedings of the European Conference on Computer Vision*, pp. 818–833, September 2014, Zürich, Switzerland.

[39] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Proceedings of the International Conference on Learning Representations*, May 2015, San Diego, CA.

[40] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *Visual Communication and Image Representation*, vol. 104, pp. 154–171, September 2013.

[41] P. D. Larry Zitnick, "Edge boxes: Locating object proposals from edges," *Proceedings of the European Conference on Computer Vision*, pp. 391–405, September 2014, Zurich, Switzerland.

[42] "Pinterest," URL: www.pinterest.com.

[43] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proceedings of th IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779–788, June 2016, Las Vegas, NV.

[44] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *arXiv:1612.08242*, 2016.

[45] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," *Proceedings of the European Conference on Computer Vision*, pp. 21–37, October 2016, Amsterdam, Netherlands.

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, June 2016, Las Vegas, NV.

[47] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 936–944, June 2017, Honolulu, HI.

[48] S. C. Hoi, X. Wu, H. Liu, Y. Wu, H. Wang, H. Xue, and Q. Wu, "Large-scale deep logo detection and brand recognition with deep region-based convolutional networks," *arXiv:1511.02462*, 2015.

[49] D. Mas Montserrat, Q. Lin, J. Allebach, and E. J. Delp, "Logo detection and recognition with synthetic images," *Proceedings of the IS&T International Symposium on Electronic Imaging*, January 2018, Burlingame, CA.

[50] S. Bianco, M. Buzzelli, D. Mazzini, and R. Schettini, "Deep learning for logo recognition," *Neurocomputing*, vol. 245, pp. 23–30, 2017.

[51] D. Mas Montserrat, Q. Lin, J. Allebach, and E. J. Delp, "Scalable logo detection and recognition with minimal labeling," *Proceedings of the IEEE International Conference on Multimedia Information Processing and Retrieval*, pp. 152–157, April 2018, Miami, FL.

[52] "Logo detection using YOLOv2," URL: https://medium.com/@akarshzingade/logo-detection-using-yolov2-8cda5a68740e.

[53] "Flickr," URL: flickr.com/.

[54] Y. Kalantidis, L. Pueyo, M. Trevisiol, R. van Zwol, and Y. Avrithis, "Scalable triangulation-based logo recognition," *Proceedings of the ACM International Conference on Multimedia Retrieval*, pp. 1–7, April 2011, Trento, Italy.

[55] H. Sahbi, L. Ballan, G. Serra, and A. D. Bimbo, "Context-dependent logo matching and recognition," *IEEE Transactions on Image Processing*, vol. 22, no. 8, pp. 1018–1031, August 2013.

[56] S. Bianco, M. Buzzelli, D. Mazzini, and R. Schettini, "Logo recognition using cnn features," *Proceedings of the International Conference on Image Analysis and Processing*, pp. 438–448, September 2015, Genova, Italy.

[57] A. Tüzkö, C. Herrmann, D. Manger, and J. Beyerer, "Open Set Logo Detection and Retrieval," *Proceedings of the International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, January 2018, Funchal, Portugal.

[58] M. Paulin, J. Revaud, Z. Harchaoui, F. Perronnin, and C. Schmid, "Transformation pursuit for image classification," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3646–3653, June 2014, Columbus, OH.

[59] P. Letessier, A. Joly, and O. Buisson, "Scalable mining of small visual objects," *Proceedings of the ACM international conference on Multimedia*, pp. 599–608, 2012, Nara, Japan.

[60] J. Xu, D. Vazquez, A. Lopez, J. Marin, and D. Ponsa, "Learning a part-based pedestrian detector in virtual world," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 5, pp. 2121–2131, October 2014.

[61] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Kttler, A. Lefrancq, S. Green, V. Valds, A. Sadik, J. Schrittwieser, K. Anderson, S. York, M. Cant, A. Cain, A. Bolton, S. Gaffney, H. King, D. Hassabis, S. Legg, and S. Petersen, "Deepmind lab," *arXiv:1612.03801*, 2015.

[62] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, "Draw: A recurrent neural network for image generation," *Proceedings of the International Conference on Machine Learning*, pp. 1462–1471, July 2015, Lille, France.

[63] S. C. A Radford, L Metz, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv:1511.06434*, 2016.

[64] S. E. Reed, H. Lee, D. Anguelov, C. Szegedy, D. Erhan, and A. Rabinovich, "Training deep neural networks on noisy labels with bootstrapping," *Proceedings of the International Conference on Learning Representations*, May 2014, San Diego, CA.

[65] "HP Sprout," URL: http://www8.hp.com/us/en/sprout/home.html.

[66] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," *Proceedings of the Advances in Neural Information Processing Systems*, pp. 487–495, December 2014, Montreal, Canada.

[67] P. Prez, M. Gangnet, and A. Blake, "Poisson image editing," *Proceedings of the International Conference On Computer Graphics And Interactive Techniques*, pp. 313–318, July 2003, San Diego, CA.

[68] D. M. W. Powers, "Evaluation: From precision, recall and f-factor to roc, informedness, markedness and correlation," *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37–63, December 2011.

[69] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, no. 8, p. 861874, June 2006.

[70] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, "Labelme: A database and web-based tool for image annotation," *International Journal of Computer Vision*, vol. 77, no. 1-3, pp. 157–173, May 2008.

[71] "SynthText Github," URL: github.com/ankush-me/SynthText.

[72] F. Liu, C. Shen, and G. Lin, "Deep convolutional neural fields for depth estimation from a single image," *Proceedings of the IEEE Conference of Computer Vision and Pattern Recognition*, pp. 5162–5170, June 2015, Boston, MA.

[73] C. F. P. Arbelaez, M. Maire and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, pp. 898–916, May 2011.

[74] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the Association for Computing Machinery*, vol. 24, no. 6, pp. 381–395, June 1981.

[75] "PyTorch," URL: http://pytorch.org.

[76] O. S. Ipsiroglu, Y.-H. A. Hung, F. Chan, M. L. Ross, D. Veer, S. Soo, G. Ho, M. Berger, G. McAllister, H. Garn, *et al.*, ""diagnosis by behavioral observation" home-videosomnography–a rigorous ethnographic approach to sleep of children with neurodevelopmental conditions," *Frontiers in psychiatry*, vol. 6, p. 39, 2015.

[77] M. Killen, J. Lee-Kim, H. McGlothlin, and C. Stangor, *Monographs of the Society for Research in Child Development*. Blackwell Publishing, 2002.

[78] S. Okada, Y. Ohno, K. Kato-Nishimura, I. Mohri, M. Taniike, *et al.*, "Examination of non-restrictive and non-invasive sleep evaluation technique for children using difference images," *Proceedings of the IEEE International Conference in Medicine and Biology Society*, pp. 3483–3487, July 2008, Vancouver, Canada.

[79] M. Nakatani, S. Okada, S. Shimizu, I. Mohri, Y. Ohno, M. Taniike, and M. Makikawa, "Body movement analysis during sleep for children with adhd using video image processing," *Proceedings of the IEEE International Conference in Medicine and Biology Society*, pp. 6389–6392, July 2017, Osaka, Japan.

[80] A. Heinrich, X. Aubert, and G. de Haan, "Body movement analysis during sleep based on video motion estimation," *Proceedings of the IEEE International Conference on e-Health Networking, Applications and Services*, pp. 539–543, October 2013, Lisbon, Portugal.

[81] J. Choe, D. M. Montserrat, A. Schwichtenberg, and E. J. Delp, "Sleep analysis using motion and head detection," *Proceedings of the IEEE Southwest Symposium on Image Analysis and Interpretation*, pp. 29–32, April 2018, Las Vegas, NV.

[82] T.-H. Vu, A. Osokin, and I. Laptev, "Context-aware cnns for person head detection," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2893–2901, December 2015, Santiago, Chile.

[83] A. Sadeh, M. Sharkey, and M. A. Carskadon, "Activity-based sleep-wake identification: an empirical test of methodological issues," *Sleep*, vol. 17, no. 3, pp. 201–207, April 1994.

[84] G. Pavlakos, X. Zhou, A. Chan, K. G. Derpanis, and K. Daniilidis, "6-DoF object pose from semantic keypoints," *Proceedings of the International Conference on Robotics and Automation*, pp. 2011–2018, May 2017, Singapore, Singapore.

[85] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep object pose estimation for semantic robotic grasping of household objects," *Proceedings of the Conference on Robot Learning*, October 2018, Zurich, Switzerland.

[86] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1530–1538, October 2017, Venice, Italy.

[87] M. Rad and V. Lepetit, "BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3848–3856, October 2017, Venice, Italy.

[88] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, "DeepIM: Deep iterative matching for 6d pose estimation," *Proceedings of the European Conference on Computer Vision*, pp. 683–698, September 2018, Munich, Germany.

[89] D. M. Montserrat, J. Chen, Q. Lin, J. P. Allebach, and E. J. Delp, "Multi-View matching network for 6D pose estimation," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, June 2019, Long Beach, CA.

[90] A. Kundu, Y. Li, and J. M. Rehg, "3D-RCNN: Instance-level 3d object reconstruction via render-and-compare," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3559–3568, June 2018, Salt Lake City, UT.

[91] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes," *Proceedings of Robotics: Science and Systems*, June 2018, Pittsburgh, PA.

[92] T. T. Do, M. Cai, T. Pham, and I. D. Reid, "Deep-6DPose: Recovering 6d object pose from a single rgb image," *arXiv:1802.10367*, 2018.

[93] R. A. Güler, N. Neverova, and I. Kokkinos, "DensePose: Dense human pose estimation in the wild," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7297–7306, June 2018, Salt Lake City, UT.

[94] F. Michel, A. Kirillov, E. Brachmann, A. Krull, S. Gumhold, B. Savchynskyy, and C. Rother, "Global hypothesis generation for 6d object pose estimation," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 115–124, June 2017, Honolulu, HI.

[95] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3D shape recognition," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 945–953, December 2015, Santiago, Chile.

[96] C. Li, J. Bai, and G. D. Hager, "A unified framework for multi-view multi-class object pose estimation," *Proceedings of the European Conference on Computer Vision*, pp. 945–953, September 2018, Munich, Germany.

[97] I. Melekhov, J. Ylioinas, J. Kannala, and E. Rahtu, "Relative camera pose estimation using convolutional neural networks," 2017. [Online]. Available: https://arxiv.org/abs/1702.01381

[98] F. Manhardt, W. Kehl, N. Navab, and F. Tombari, "Deep model-based 6d pose refinement in rgb," *Proceedings of the European Conference on Computer Vision*, September 2018, Munich, Germany.

[99] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. v.d. Smagt, D. Cremers, and T. Brox, "FlowNet: Learning optical flow with convolutional networks," *Proceedings of the IEEE International Conference on Computer Vision*, December 2015, Las Condes, Chile.

[100] T. Hodan, P. Haluza, Š. Obdržálek, J. Matas, M. Lourakis, and X. Zabulis, "T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects," *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*, pp. 880–888, March 2017, Santa Rosa, CA.

[101] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Yale-CMU-Berkeley dataset for robotic manipulation research," *International Journal of Robotics Research*, vol. 36, no. 3, pp. 261–268, 2017.

[102] C. Rennie, R. Shome, K. E. Bekris, and A. F. De Souza, "A dataset for improved RGBD-based object detection and pose estimation for warehouse pick-and-place," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 1179–1185, 2016.

[103] A. Tejani, D. Tang, R. Kouskouridas, and T.-K. Kim, "Latent-class hough forests for 3d object detection and pose estimation," *Proceedings of the European Conference on Computer Vision*, pp. 462–477, September 2014, Zurich, Switzerland.

[104] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes," *proceedings of the 11th Asian Conference on Computer Vision*, pp. 548–562, November 2012, Daejeon, Korea.

[105] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, "Learning 6D object pose estimation using 3D object coordinates," *Proceedings of the European Conference on Computer Vision*, pp. 536–551, September 2014, Zurich, Switzerland.

[106] A. Krull, E. Brachmann, F. Michel, M. Ying Yang, S. Gumhold, and C. Rother, "Learning analysis-by-synthesis for 6D pose estimation in RGB-D images," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 954–962, December 2015, Santiago, Chile.

[107] B. Drost, M. Ulrich, P. Bergmann, P. Hartinger, and C. Steger, "Introducing MVTec ITODD-a dataset for 3D object recognition in industry," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2200–2208, October 2017, Venice, Italy.

[108] J. Tremblay, T. To, and S. Birchfield, "Falling things: A synthetic dataset for 3D object detection and pose estimation," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 2038–2041, June 2018, Salt Lake City, UT.

[109] J. Chen, D. M. Montserrat, Q. Lin, E. J. Delp, and J. P. Allebach, "Extra FAT: A photorealistic dataset for 6D object pose estimation," *Proceedings of the IS&T International Symposium on Electronic Imaging*, January 2020, Burlingame, CA.

[110] T. Hodan, F. Michel, E. Brachmann, W. Kehl, A. GlentBuch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis, *et al.*, "BOP: Benchmark for 6D object pose estimation," *Proceedings of the European Conference on Computer Vision*, pp. 19–34, September 2018, Munich, Germany.

[111] "Unreal Engine 4," URL: www.unrealengine.com.

[112] W. Qiu and A. Yuille, "UnrealCV: Connecting computer vision to unreal engine," *Proceedings of the European Conference on Computer Vision*, pp. 909–916, October 2016, Amsterdam, Netherlands.

[113] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Proceedings of Advances in Neural Information Processing Systems*, pp. 2672–2680, December 2014, montreal, Canada.

[114] B. Dolhansky, R. Howes, B. Pflaum, N. Baram, and C. C. Ferrer, "The deepfake detection challenge (dfdc) preview dataset," *arXiv preprint arXiv:1910.08854*, 2019.

[115] E. Bailey, *Adobe Photoshop: A Beginners Guide to Photoshop Lightroom - The 52 Photoshop Lightroom Tricks You Didnt Know Existed!* CreateSpace Independent Publishing Platform, 2016, vol. 1, North Charleston, SC.

[116] The GIMP Development Team, "GIMP," https://www.gimp.org.

[117] D. Cozzolino and L. Verdoliva, "Noiseprint: a cnn-based camera model fingerprint," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 144–159, May 2019.

[118] M. Barni, Q.-T. Phan, and B. Tondi, "Copy move source-target disambiguation through multi-branch cnns," *arXiv preprint arXiv:1912.12640*, 2019.

[119] S. Yarlagadda, D. Güera, D. M. Montserrat, F. Zhu, E. Delp, P. Bestagini, and S. Tubaro, "Shadow removal detection and localization for forensics analysis," *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 2677–2681, May 2019, brighton, UK.

[120] D. Cozzolino, G. Poggi, and L. Verdoliva, "Splicebuster: A new blind image splicing detector," *Proceedings of IEEE International Workshop on Information Forensics and Security*, pp. 1–6, January 2015, rome, Italy.

[121] S. K. Yarlagadda, D. Güera, P. Bestagini, F. Maggie Zhu, S. Tubaro, and E. J. Delp, "Satellite image forgery detection and localization using gan and one-class classifier," *Proceedings of the IS&T International Symposium on Electronic Imaging*, vol. 2018, no. 7, pp. 214–1, January 2018.

[122] E. R. Bartusiak, S. K. Yarlagadda, D. Güera, P. Bestagini, S. Tubaro, F. M. Zhu, and E. J. Delp, "Splicing detection and localization in satellite imagery using conditional gans," *Proceedings of IEEE Conference on Multimedia Information Processing and Retrieval*, pp. 91–96, March 2019, san Jose, CA.

[123] J. Horváth, D. Güera, S. K. Yarlagadda, P. Bestagini, F. M. Zhu, S. Tubaro, and E. J. Delp, "Anomaly-based manipulation detection in satellite images," *Networks*, vol. 29, p. 21, 2019.

[124] M. Barni, L. Bondi, N. Bonettini, P. Bestagini, A. Costanzo, M. Maggini, B. Tondi, and S. Tubaro, "Aligned and non-aligned double jpeg detection using convolutional neural networks," *Journal of Visual Communication and Image Representation*, vol. 49, pp. 153–163, November 2017.

[125] D. Güera, S. Baireddy, P. Bestagini, S. Tubaro, and E. J. Delp, "We need no pixels: Video manipulation detection using stream descriptors," *Proceedings of the International Conference on Machine Learning , Synthetic Realities: Deep Learning for Detecting AudioVisual Fakes Workshop*, June 2019, Long Beach, CA.

[126] F. Marra, D. Gragnaniello, D. Cozzolino, and L. Verdoliva, "Detection of gan-generated fake images over social networks," *Proceedings of the IEEE Conference on Multimedia Information Processing and Retrieval*, pp. 384–389, April 2018, miami, FL.

[127] F. Marra, D. Gragnaniello, L. Verdoliva, and G. Poggi, "Do gans leave artificial fingerprints?" *Proceedings of IEEE Conference on Multimedia Information Processing and Retrieval*, pp. 506–511, March 2019, san Diego, CA.

[128] X. Zhang, S. Karaman, and S.-F. Chang, "Detecting and simulating artifacts in gan fake images," *arXiv preprint arXiv:1907.06515*, 2019.

[129] N. Yu, L. S. Davis, and M. Fritz, "Attributing fake images to gans: Learning and analyzing gan fingerprints," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 7556–7566, October 2019, seoul, South Korea.

[130] D. Afchar, V. Nozick, J. Yamagishi, and I. Echizen, "Mesonet: a compact facial video forgery detection network," *Proceedings of the IEEE International Workshop on Information Forensics and Security*, pp. 1–7, December 2018, hong Kong.

[131] A. Rossler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner, "Faceforensics++: Learning to detect manipulated facial images," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1–11, October 2019, seoul, South Korea.

[132] Z. Hui, J. Li, X. Wang, and X. Gao, "Image fine-grained inpainting," *arXiv preprint arXiv:2002.02609*, 2020.

[133] H. Le and D. Samaras, "Shadow removal via shadow image decomposition," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 8578–8587, October 2019, seoul, South Korea.

[134] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," *arXiv preprint arXiv:1809.11096*, 2018.

[135] DeepFakes. https://github.com/deepfakes/faceswap.

[136] J. Thies, M. Zollhöfer, and M. Nießner, "Deferred neural rendering: Image synthesis using neural textures," *ACM Transactions on Graphics*, vol. 38, no. 4, pp. 1–12, July 2019.

[137] M. Kowalski. Faceswap. https://github.com/MarekKowalski/FaceSwap/.

[138] J. Thies, M. Zollhofer, M. Stamminger, C. Theobalt, and M. Nießner, "Face2face: Real-time face capture and reenactment of rgb videos," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, Las Vegas, NV, June 2016, pp. 2387–2395.

[139] P. Pérez, M. Gangnet, and A. Blake, "Poisson image editing," *Proceedings of the ACM Special Interest Group on Computer GRAPHics and Interactive Techniques*, pp. 313–318, July 2003, san Diego, California.

[140] D. Güera and E. J. Delp, "Deepfake video detection using recurrent neural networks," *Proceedings of the IEEE International Conference on Advanced Video and Signal Based Surveillance*, pp. 1–6, November 2018, auckland, New Zealand.

[141] E. Sabir, J. Cheng, A. Jaiswal, W. AbdAlmageed, I. Masi, and P. Natarajan, "Recurrent convolutional strategies for face manipulation detection in videos," *Interfaces (GUI)*, vol. 3, p. 1, 2019.

[142] X. Yang, Y. Li, and S. Lyu, "Exposing deep fakes using inconsistent head poses," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8261–8265, May 2019, brighton, United Kingdom.

[143] Y. Li, M.-C. Chang, and S. Lyu, "In ictu oculi: Exposing ai created fake videos by detecting eye blinking," *Proceeding IEEE International Workshop on Information Forensics and Security*, pp. 1–7, 2018, Hong Kong.

[144] Y. Li and S. Lyu, "Exposing deepfake videos by detecting face warping artifacts," *arXiv preprint arXiv:1811.00656*, 2018.

[145] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 4700–4708, July 2017, honolulu, HI.

[146] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, June 2016, las Vegas, NV.

[147] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1251–1258, July 2017, honolulu, HI.

[148] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," pp. 248–255, August 2009, Miami, FL.

[149] M. Huh, P. Agrawal, and A. A. Efros, "What makes imagenet good for transfer learning?" *arXiv preprint arXiv:1608.08614*, 2016.

[150] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, April 2016. [Online]. Available: https://doi.org/10.1109/LSP.2016.2603342

[151] V. Bazarevsky, Y. Kartynnik, A. Vakunov, K. Raveendran, and M. Grundmann, "Blazeface: Sub-millisecond neural face detection on mobile gpus," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop on Computer Vision for Augmented and Virtual Reality*, June 2019, Long Beach, CA.

[152] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

[153] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "ArcFace: Additive angular margin loss for deep face recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2019, Long Beach, CA.

[154] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Proceedings of Advances in Neural Information Processing Systems*, pp. 5998–6008, December 2017, Long Beach, CA.

[155] P. K. Suraj, A. Gupta, M. Sharma, S. B. Paul, and S. Banerjee, "On monitoring development using high resolution satellite images," *arXiv:1712.02282*, December 2017.

[156] B. Oshri, A. Hu, P. Adelson, X. Chen, P. Dupas, J. Weinstein, M. Burke, D. Lobell, and S. Ermon, "Infrastructure quality assessment in africa using satellite imagery and deep learning," *Proceedings of the International Conference on Knowledge Discovery & Data Mining*, August 2018, london, United Kingdom.

[157] M. Rußwurm, S. Lefèvre, and M. Körner, "Breizhcrops: A satellite time series dataset for crop type identification," *Proceedings of the International Conference on Machine Learning Time Series Workshop*, June 2019, long Beach, CA.

[158] J. Brandt, "Spatio-temporal crop classification of low-resolution satellite imagery with capsule layers and distributed attention," *arXiv:1904.10130*, April 2019.

[159] A. Chauve, C. Vega, S. Durrieu, F. Bretar, T. Allouis, M. P. Deseilligny, and W. Puech, "Advanced full-waveform lidar data echo detection: Assessing quality of derived terrain and tree height models in an alpine coniferous forest," *International Journal of Remote Sensing*, vol. 30, no. 19, pp. 5211–5228, September 2009.

[160] A. Davari, V. Christlein, S. Vesal, A. Maier, and C. Riess, "GMM Supervectors for Limited Training Data in Hyperspectral Remote Sensing Image Classification," *Proceedings of the International Conference on Computer Analysis of Images and Patterns*, pp. 296–306, July 2017, Ystad, Sweden.

[161] M. Shimoni, D. Borghys, R. Heremans, C. Perneel, and M. Acheroy, "Land-cover classification using fused polsar and polinsar features," *Proceedings of the European Conference on Synthetic Aperture Radar*, pp. 1–4, June 2008, friedrichshafen, Germany.

[162] N. Efremova, D. Zausaev, and G. Antipov, "Prediction of soil moisture content based on satellite data and sequence-to-sequence networks," *Proceedings of the Conference on Neural Information Processing Systems Women in Machine Learning Workshop*, December 2018, montreal, Canada.

[163] D. D. Alexakis, F. K. Mexis, A. K. Vozinaki, I. N. Daliakopoulos, and I. K. Tsanis, "Soil moisture content estimation based on sentinel-1 and auxiliary earth observation products. A hydrological approach," *Sensors*, June 2017.

[164] V. Lebedev, V. Ivashkin, I. Rudenko, A. Ganshin, A. Molchanov, S. Ovcharenko, R. Grokhovetskiy, I. Bushmarinov, and D. Solomentsev, "Precipitation nowcasting with satellite imagery," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, August 2019, anchorage, AK.

[165] Y. Zhang, S. Wistar, J. Li, M. Steinberg, and J. Wang, "Storm detection by visual learning using satellite images," *Proceedings of the IEEE Transactions on Geoscience and Remote Sensing*, March 2016, yokohama, Japan.

[166] I. Sahoo, J. Guinness, and B. Reich, "Estimating atmospheric motion winds from satellite image data using space-time drift models," *arXiv:1902.09653*, February 2019.

[167] "UCS satellite database," https://www.ucsusa.org/resources/satellite-database/.

[168] G. Xia, X. Bai, J. Ding, Z. Zhu, S. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang, "Dota: A large-scale dataset for object detection in aerial images," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3974–3983, June 2018, salt Lake City, UT. [Online]. Available: https://arxiv.org/pdf/1906.07789

[169] S. M. Azimi, C. Henry, L. Sommer, A. Schumann, and E. Vig, "Skyscapes fine-grained semantic understanding of aerial scenes," *Proceedings of the IEEE International Conference on Computer Vision*, October 2019, Seoul, Korea.

[170] Y. Yang and S. Newsam, "Bag-of-visual-words and spatial extensions for land-use classification," *Proceedings of the ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems*, pp. 270–279, November 2009, San Jose, CA. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/1869790.1869829?download=true

[171] R. Gupta, R. Hosfelt, S. Sajeev, N. Patel, B. Goodman, J. Doshi, E. Heim, H. Choset, and M. Gaston, "xBD: A dataset for assessing building damage from satellite imagery," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 10–17, June 2019.

[172] M. Schmitt, L. H. Hughes, C. Qiu, and X. X. Zhu, "SEN12MS – a curated dataset of georeferenced multi-spectral sentinel-1/2 imagery for deep learning and data fusion," *arXiv:1906.07789*, June 2019. [Online]. Available: https://arxiv.org/pdf/1906.07789

[173] S. Nam, Y. Kim, and S. J. Kim, "Text-adaptive generative adversarial networks: Manipulating images with natural language," *Proceedings of the Advances in Neural Information Processing Systems*, December 2018, montreal, Canada.

[174] D. Byrd, "Fake image of diwali still circulating," https://earthsky.org/earth/fake-image-of-india-during-diwali/-versus-the-real-thing.

[175] A. E. Kramer, "Russian images of malaysia airlines flight 17 were altered, report finds," https://www.nytimes.com/2016/07/16/world/europe/malaysia-airlines-flight-17-russia.html.

[176] J. Edwards, "China uses gan technique to tamper with earth images," https://www.executivegov.com/2019/04/ngas-todd-myers-china-uses-gan-/technique-to-tamper-with-earth-images/.

[177] D. Cozzolino and L. Verdoliva, "Noiseprint: A cnn-based camera model fingerprint," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 144–159, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8713484

[178] A. Rocha, W. Scheirer, T. Boult, and S. Goldenstein, "Vision of the unseen: Current trends and challenges in digital image and video forensics," *ACM Computing Surveys*, vol. 43, no. 4, October 2011.

[179] V. Schetinger, M. Iuliani, A. Piva, and M. Oliveira, "Image forgery detection confronts image composition," *Computers & Graphics*, vol. 68, no. C, pp. 152–163, November 2017. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0097849317301498

[180] F. Bartolini, A. Tefas, M. Barni, and I. Pitas, "Image authentication techniques for surveillance applications," *Proceedings of the IEEE*, vol. 89, no. 10, pp. 1403–1418, October 2001.

[181] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," *Proceedings of the International Conference on Machine Learning*, pp. 1747–1756, 2016, New York, NY.

[182] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, *et al.*, "Conditional image generation with pixelcnn decoders," *Proceedings of the Advances in Neural Information Processing Systems*, pp. 4790–4798, December 2016, Barcelona, Spain.

[183] J. Horváth, D. M. Montserrat, H. Hao, and E. J. Delp, "Manipulation detection in satellite images using deep belief networks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, June 2020, Seattle, WA.

[184] M. Barni, A. Costanzo, and L. Sabatini, "Identification of cut&paste tampering by means of double-JPEG detection and image segmentation," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1687–1690, May 2010, Paris, France. [Online]. Available: https://doi.org/10.1109/ISCAS.2010.5537505

[185] D. Cozzolino, J. Thies, A. Rössler, C. Riess, M. Nießner, and L. Verdoliva, "Forensictransfer: Weakly-supervised domain adaptation for forgery detection," *arXiv:1812.02510*, December 2018.

[186] D. M. Montserrat, H. Hao, S. K. Yarlagadda, S. Baireddy, R. Shao, J. Horváth, E. Bartusiak, J. Yang, D. Güera, F. Zhu, and E. J. Delp, "Deepfakes detection with automatic face weighting," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, June 2020, Seattle, WA.

[187] S. McCloskey and M. Albright, "Detecting GAN-Generated Imagery Using Saturation Cues," *Proceedings of the IEEE International Conference on Image Processing*, pp. 4584–4588, September 2019, taipei, Taiwan.

[188] D. Cozzolino, G. Poggi, and L. Verdoliva, "Splicebuster: A new blind image splicing detector," *Proceedings of the IEEE International Workshop on Information Forensics and Security*, pp. 1–6, November 2015, Rome, Italy.

[189] A. T. S. Ho and W. M. Woon, "A semi-fragile pinned sine transform watermarking system for content authentication of satellite images," *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium*, vol. 2, pp. 1–4, July 2005, seoul, South Korea. [Online]. Available: https://doi.org/10.1109/IGARSS.2005.1525212

[190] E. R. Bartusiak, S. Kalyan Yarlagadda, D. Güera, F. M. Zhu, P. Bestagini, S. Tubaro, and E. J. Delp, "Splicing detection and localization in satellite imagery using conditional gans," *Proceedings of the IEEE International Conference on Multimedia Information Processing and Retrieval*, pp. 91–96, March 2019, San Jose, CA.

[191] S. Kalyan Yarlagadda, D. Güera, P. Bestagini, F. Zhu, S. Tubaro, and E. Delp, "Satellite image forgery detection and localization using gan and one-class classifier," *Proceedings of the IS&T International Symposium on Electronic Imaging*, vol. 2018, no. 7, pp. 214–1–214–9, February 2018, Burlingame, CA.

[192] J. Horvath, D. Guera, S. Kalyan Yarlagadda, P. Bestagini, F. Maggie Zhu, S. Tubaro, and E. J. Delp, "Anomaly-based manipulation detection in satellite images," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 62–71, June 2019, Long Beach, CA.

[193] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5967–5976, July 2017, Honolulu, HI.

[194] D. M. J. Tax and R. P. W. Duin, "Support vector data description," *Machine Learning*, vol. 54, no. 1, pp. 45–66, January 2004.

[195] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computer*, vol. 18, no. 7, July 2006.

[196] P. Smolensky, *Information Processing in Dynamical Systems: Foundations of Harmony Theory*. Cambridge, MA: MIT Press, 1986.

[197] M. Germain, K. Gregor, I. Murray, and H. Larochelle, "MADE: Masked autoencoder for distribution estimation," *Proceedings of the International Conference on Machine Learning*, pp. 881–889, July 2015, lille, France.

[198] H. Larochelle and I. Murray, "The neural autoregressive distribution estimator," *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pp. 29–37, April 2011, Lauderdale, FL.

[199] K. Gregor, I. Danihelka, A. Mnih, C. Blundell, and D. Wierstra, "Deep autoregressive networks," *Proceedings of the 31st International Conference on Machine Learning*, vol. 32, no. 2, pp. 1242–1250, June 2014, Bejing, China.

[200] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "PixelCNN++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications," *arXiv preprint arXiv:1701.05517*, 2017.

[201] X. Chen, N. Mishra, M. Rohaninejad, and P. Abbeel, "Pixelsnail: An improved autoregressive generative model," *arXiv preprint arXiv:1712.09763*, 2017.

[202] N. Kalchbrenner, A. van den Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu, "Video pixel networks," *Proceedings of the International Conference on Machine Learning*, pp. 1771–1779, August 2017, Sydney, Australia.

[203] I. Gulrajani, K. Kumar, F. Ahmed, A. A. Taiga, F. Visin, D. Vazquez, and A. Courville, "PixelVAE: A latent variable model for natural images," *arXiv preprint arXiv:1611.05013*, 2016.

[204] H. Sadeghi, E. Andriyash, W. Vinci, L. Buffoni, and M. H. Amin, "Pixelvae++: Improved pixelvae with discrete prior," *arXiv preprint arXiv:1908.09948*, 2019.

[205] A. Makhzani and B. J. Frey, "Pixelgan autoencoders," pp. 1975–1985, December 2017, Long Beach, CA.

[206] K. Migdol and J. Ventura, "PixelMRF: A Convolutional Markov Random Field for Image Generation," http://cs.uccs.edu/ jkalita/work/reu/REU2018/14Migdol.pdf.

[207] H. Choi, E. Jang, and A. A. Alemi, "WAIC, but why? generative ensembles for robust anomaly detection," *arXiv preprint arXiv:1810.01392*, 2018.

[208] S. Watanabe, "Asymptotic equivalence of bayes cross validation and widely applicable information criterion in singular learning theory," *Journal of Machine Learning Research*, vol. 11, no. December, pp. 3571–3594, 2010.

[209] J. Serrà, D. Álvarez, V. Gómez, O. Slizovskaia, J. F. Núñez, and J. Luque, "Input complexity and out-of-distribution detection with likelihood-based generative models," *arXiv preprint arXiv:1909.11480*, 2019.

[210] Sentinel Program, "Sentinel missions," Accessed 2020, https://sentinel.esa.int/web/sentinel/missions.

VITA

VITA

Daniel Mas Montserrat was born in Catalonia, Spain in 1993. In 2015, he received his Bachelor of Science in Audiovisual Systems in Telecommunications Engineering from the Polytechnic University of Catalonia. Mr. Mas joined the Ph.D. program at the School of Electrical and Computer Engineering at Purdue University, West Lafayette, Indiana in 2016 where he worked at the Video and Image Processing Laboratory (VIPER) under the supervision of Professor Edward J. Delp. While pursuing his Ph.D. at Purdue, he primarily worked on projects sponsored by HP Labs and the Air Force Research Laboratory (AFRL), the Defense Advanced Research Projects Agency (DARPA), and the National Geospatial Intelligence Agency (NGA). During his studies, he also gained research experience at HP, Qualcomm and Stanford University. His current research interests include deep learning, signal processing, and computer vision. He is a student member of the IEEE and the IEEE Signal Processing Society.