UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

FROM TRADITIONAL MULTI-STAGE LEARNING
TO END-TO-END DEEP LEARNING FOR
COMPUTER VISION APPLICATIONS

TESIS DOCTORAL

ANA ISABEL MAQUEDA NIETO
INGENIERO DE TELECOMUNICACIÓN
2018

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE
TELECOMUNICACIÓN

FROM TRADITIONAL MULTI-STAGE LEARNING
TO END-TO-END DEEP LEARNING FOR
COMPUTER VISION APPLICATIONS

TESIS DOCTORAL

ANA ISABEL MAQUEDA NIETO
INGENIERO DE TELECOMUNICACIÓN

2018

DEPARTAMENTO DE SEÑALES, SISTEMAS Y
RADIOCOMUNICACIONES

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

**From traditional multi-stage learning to end-to-end deep
learning for computer vision applications**

Tesis Doctoral

Autor:

Ana Isabel Maqueda Nieto

Ingeniero de Telecomunicación
Universidad Politécnica de Madrid

Directores:

Narciso García Santos

Doctor Ingeniero de Telecomunicación
Universidad Politécnica de Madrid

Carlos Roberto del Blanco Adán

Doctor Ingeniero de Telecomunicación
Universidad Politécnica de Madrid

2018

TESIS DOCTORAL

From traditional multi-stage learning to end-to-end deep learning for computer vision applications

Autor: Ana Isabel Maqueda Nieto

Directores: Narciso García Santos y Carlos Roberto del Blanco Adán

Tribunal nombrado por el Mgfc. y Excmo. Sr. Rector de la Universidad Politécnica de Madrid,
el día de de 2018.

Presidente D.

Vocal D.

Vocal D.

Vocal D.

Secretario D.

Realizado el acto de defensa y lectura de la Tesis el día de de 2018
en

Calificación:

EL PRESIDENTE

LOS VOCALES

EL SECRETARIO

Acknowledgments

First, I have a lot to thank my advisors, Narciso García and Carlos Roberto del Blanco, for offering me the opportunity to join the Grupo de Tratamiento de Imágenes. They have inspired me along these four years in my research, and friendly guided me through this thesis.

I am very grateful to Davide Scaramuzza for having me in his lab, the Robotics and Perception Group in Zurich, during my PhD stay. It was a very rewarding experience, where I had the pleasure to work with great researchers. I would particularly like to thank Antonio Loquercio and Guillermo Gallego for all the help, and their great work as co-authors.

I would like to thank all the current and past colleagues at the Grupo de Tratamiento de Imágenes for all the help, fruitful discussions, football and basketball matches, and fun table talks. I would particularly like to thank Fernando, Tomás, Susana, Dani, Adrián, Rafa, César, Carlos Cuevas, Julián and Francisco for all the enjoyable time I have spent with you.

Finally, I would like to thank my family and Fede for their unconditional love, always supporting me, and also putting up with me in the hardest and more stressful stages of this trip.

Resumen

La reaparición de las Redes Neuronales Profundas, esta vez en la era del *big data*, e implementadas sobre hardware de alto rendimiento que reduce el tiempo de computación, ha cambiado el paradigma del aprendizaje automático, especialmente en el campo de la visión artificial. Mientras que los sistemas tradicionales basados en aprendizaje máquina emplean múltiples etapas y características diseñadas a mano para facilitar el proceso de aprendizaje, las Redes Neuronales Convolucionales aprenden automáticamente las características que maximizan dicho proceso de extremo a extremo, es decir, desde las propias imágenes hasta la salida deseada. El propósito de esta tesis es mostrar cualitativamente la diferencia entre los sistemas multi-etapa que se basan en aprendizaje máquina tradicional y los sistemas de aprendizaje profundo de extremo a extremo, utilizando para ello diferentes aplicaciones como contexto.

En primer lugar, se ha desarrollado un sistema de reconocimiento de gestos dinámicos de manos, donde dos de los aspectos clave son descriptores de imagen y video, y el diseño del sistema completo formado por múltiples etapas. Estos descriptores han sido diseñados para lidiar con las dificultades de los sistemas basados en visión, como los cambios de iluminación, las variaciones intra-clase e inter-clase y transformaciones que pueden sufrir los gestos. Las diferentes etapas del sistema resuelven pasos intermedios que son necesarios para aplicar con éxito los descriptores anteriores. Dado que el sistema propuesto de reconocimiento de gestos ha sido pensado para una interfaz hombre-máquina, este comprende etapas de detección y seguimiento para localizar el objeto de interés, y una etapa de reconocimiento para categorizar el gesto realizado.

En segundo lugar, se han propuesto varios sistemas basados en aprendizaje profundo, o redes neuronales profundas, para hacer frente a las debilidades presentes en el aprendizaje tradicional. A diferencia del enfoque anterior, estos sistemas no involucran múltiples etapas, ni diseño de características. La arquitectura de estas redes depende de la tarea que se quiere resolver, de su complejidad y de la cantidad de datos disponibles. Siguiendo estas directrices, se han abordado aplicaciones más comunes como la detección de vehículos y el reconocimiento de gestos de la mano, y otras más novedosas en las que la visión puede jugar un papel importante, como las aplicaciones de robótica.

Palabras clave

Aprendizaje profundo, sistemas de extremo a extremo, sistemas con múltiple etapas, características aprendidas, características diseñadas a mano, reconocimiento de gestos, detección de vehículos, navegación autónoma de robots.

Abstract

The renaissance of Deep Neural Networks in the era of big data, along with the use of high-performance hardware that reduces computational time, have changed the paradigm of machine learning, specially in the field of computer vision. Whereas systems based on traditional machine learning rely on multiple stages and hand-crafted features to get the insight of the problem, Convolutional Neural Networks automatically learn the features that maximize the learning accuracy directly from raw images in an end-to-end manner. The purpose of this dissertation is to show the gap between traditional multi-stage learning systems and end-to-end deep learning systems, addressing different applications for a qualitative comparison.

First, an expert-knowledge recognition system has been developed to deal with dynamic hand gestures. The key aspects of this system are hand-crafted image and video descriptors, and also the pipeline of the whole system. These descriptors have been designed to face difficulties of vision-based approaches such as illumination changes, intra-class and inter-class variances, and multiple scales. The design of the multiple stages of the system solve intermediate steps that are necessary to successfully apply the previous descriptors. Since the proposed hand-gesture recognition system has been designed for a human-computer interface, it comprises detection and tracking stages to localize the object of interest, and a recognition stage to categorize the performed gesture.

Second, DL approaches have been proposed for different computer vision applications. Research efforts have focused on building these types of end-to-end systems to face the weaknesses present in traditional learning. Unlike previous approach, they do not need multiple stages to perform the target task, nor feature engineering. Their architecture designs rely on the task to be solved, its complexity, and the available amount of data. These guidelines have been applied to common vision-based applications such vehicle detection, and hand-gesture recognition, but also to more challenging situations, such as robotics applications.

Key words

Deep learning, end-to-end system, multi-stage system, learned features, hand-crated features, hand-gesture recognition, vehicle detection, autonomous robot navigation.

Contents

List of Figures	V
List of Tables	XI
Glossary	XIII
1 Introduction	1
1.1 Motivation	1
1.2 Contents	3
1.3 Contributions	4
2 Relevant work	5
2.1 Introduction	5
2.2 Multi-stage learning systems	5
2.2.1 Object detection	6
2.2.2 Feature extraction	6
2.2.3 Object tracking	8
2.2.4 Supervised learning	10
2.3 End-to-end deep learning systems	11
2.3.1 Convolutional Neural Networks overview	11
2.3.1.1 CNN layers	11
2.3.1.2 Learning process	14
2.3.1.3 Regularization techniques	15
2.3.2 Recurrent neural networks overview	15
2.3.3 Image classification networks	16
2.3.4 Object recognition networks	17
2.3.5 Image segmentation networks	19
2.3.6 Object tracking networks	20
2.4 Conclusions	21
3 Multi-stage traditional learning system for hand-gesture recognition	23
3.1 Introduction	23

3.2	Related work	24
3.3	Hand-gesture recognition system	25
3.3.1	System overview	25
3.3.2	Detection	26
3.3.3	Tracking	28
3.3.4	Recognition	28
3.4	Design of spatial descriptors	30
3.4.1	Spatioqram of Local Binary Patterns (S-LBP)	31
3.4.2	Local Binary sub-Patterns (LBsP)	33
3.5	Design of temporal descriptors	35
3.5.1	Volumetric Spatiograms of Local Binary Patterns and sub-Patterns	35
3.5.2	Temporal Pyramid Matching (TPM)	37
3.6	Experimental results	38
3.6.1	Dataset	39
3.6.2	Image descriptor evaluation for hand-pose detection	40
3.6.3	Video descriptor evaluation for gesture recognition	44
3.6.4	Evaluation on the global system	46
3.7	Conclusions	49
4	End-to-end deep learning systems for different vision applications	51
4.1	Introduction	51
4.2	A structured CNN for vehicle detection at multiple scales	52
4.2.1	Related work	53
4.2.2	System description	54
4.2.2.1	Network architecture	54
4.2.2.2	Training	55
4.2.3	Experimental results	56
4.2.3.1	Dataset	56
4.2.3.2	Qualitative results	56
4.2.3.3	Quantitative results	59
4.3	Hand-gesture recognition from an image classification viewpoint	60
4.3.1	Related work	61
4.3.2	System description	61
4.3.3	Experimental results	62
4.3.3.1	Dataset	62
4.3.3.2	Qualitative and quantitative results	64

4.4	DroNet: a CNN to learn how to fly autonomously	67
4.4.1	Related work	68
4.4.2	System description	69
4.4.2.1	Learning approach	69
4.4.2.2	Drone control	71
4.4.3	Experimental results	71
4.4.3.1	Dataset	72
4.4.3.2	Regression and classification results	73
4.4.3.3	Quantitative results on DroNet’s control capabilities	74
4.4.3.4	Qualitative Results	76
4.5	Steering prediction from event-based imagery for self-driving cars	78
4.5.1	Related work	79
4.5.2	System description	80
4.5.2.1	Event-to-frame conversion	81
4.5.2.2	Preprocessing	82
4.5.2.3	Network architecture	82
4.5.3	Experimental setup	82
4.5.3.1	Dataset	82
4.5.3.2	Performance comparison on different types of images	84
4.5.4	Experimental results	85
4.5.4.1	Sensitivity analysis with respect to the event integration time	85
4.5.4.2	Results on different illumination scenarios	87
4.5.4.3	Results on the entire dataset	89
4.5.4.4	Qualitative comparison between event and standard cameras	90
4.6	Conclusions	91
5	Conclusions and future work	93
5.1	Conclusions	93
5.2	Future work	95
A	Local Binary Patterns (LBP)	97
A.1	Multi-scale Local Binary Patterns	98
A.2	Volume Local Binary Patterns (VLBP)	99
A.3	Local Binary Patterns from Three Orthogonal Planes (LBP-TOP)	100

List of Figures

1.1	Multi-stage traditional hand-gesture systems can consist in detecting the hand by skin segmentation, computing a feature vector based on its shape from the previous binary mask, and using a classifier to categorize it. In contrast, a deep neural network can learn a semantic representation of the hand and categorize it in an end-to-end manner.	2
2.1	Selective Search algorithm for object detection (extracted from [1]).	7
2.2	Computation of the LBP operator: (a) 3x3 gray scale neighborhood, (b) differences between the neighbor pixels and the center one, (c) thresholded neighborhood differences, and (d) histogram of LBP (H-LBP) from the whole image. . . .	8
2.3	Temporal extensions of LBP: (a) VLBP, and (b) LBP-TOP.	9
2.4	Example of a CNN that classifies a given input image into three categories: <i>bird</i> , <i>car</i> , and <i>dog</i> . In the feature extraction module, convolutional and pooling layers are stacked to compute the shown feature maps, which encode spatial-invariant features of the image. Next, the classification stage comprises a series of fully-connected layers that map a vectorized representation of the very last feature map into another vector with the desired output dimension. The final layer is an activation function that carries out the classification task, and provides the predicted probabilities for each category.	12
2.5	In this example, the convolution between an input image $I \in \mathbb{R}^{9 \times 8 \times 4}$ and the first convolutional filter $w_1 \in \mathbb{R}^{3 \times 3 \times 4}$ produces the first channel of the filtered image $I'_1 \in \mathbb{R}^{7 \times 6 \times 1}$. The convolution between the image and the remaining $D' - 1$ filters of the layer will produce the rest of the channels of the filtered image $I' \in \mathbb{R}^{7 \times 6 \times D'}$	13
2.6	This example shows the resulting spatial regions after applying two 2×2 pooling filters with stride 2: a max pooling filter in the top, and an average pooling filter in the bottom.	14
2.7	Compact and unrolled representations of a Recurrent Neural Network with its main parameters: x_t is the input at time step t , A is the hidden state, s_t is the “memory” of the network that is computed from previous hidden states, and h_t is the output (extracted from [2]).	16
2.8	YOLO network (extracted from [3]). In this case, the final feature map is divided into a 7×7 grid, and for each cell grid 30 values are predicted: 2 bounding boxes including x , y , h , w , and c , and the probabilities corresponding to 20 classes. . . .	19

2.9	Deconvolutional neural network (extracted from [4]).	20
3.1	Gesture taxonomy along <i>form</i> and <i>flow</i> dimensions.	25
3.2	Block diagram of the detection stage.	26
3.3	Multiresolution pyramid and sliding window strategy.	28
3.4	Tracking of detected hand poses. Blue bounding boxes correspond to detections that propagates along frames, so that the tracker is able to create a trajectory of such image regions. The tracker is robust against missing detections (yellow bounding boxes), and also against false detections (red bounding boxes). The latter do not generate a trajectory because of their short length.	29
3.5	Block diagram of recognition stage.	29
3.6	Temporal window approach to segment the trajectories of cropped images in time. It works as a FIFO buffer with 1-frame enqueue and dequeue to increase the temporal resolution when localizing a gesture. A feature vector is extracted from each temporal window to feed a set of SVM classifiers and recognize the gesture within it.	30
3.7	LBP computation: (a) original image, (b) LBP image, and (c) histogram of LBPs (H-LBP).	31
3.8	(a) Orthogonal sub-lattice of $M=8$ rows and $N=8$ columns (red points) overlaid on the LBP image to select the key image coordinates. (b) Geometric visualisation of the weights of the bilinear interpolation. The spatial distribution of the LBP pattern at the red spot is measured with respect to its four closest lattice points. (c) Resulting spatial distribution for the LBP pattern type at the red spot. LBP patterns of the same type will contribute to the same spatiogram.	32
3.9	Spatigrams of Local Binary Patterns (S-LBP)	33
3.10	LBsP computation. The LBP type 00111111 is divided into the LBsP types 0011 and 1111, each one contributing to a different histogram. The two resulting histograms are concatenated to form the H-LBsP histogram.	33
3.11	Comparison of the exponential growth of the number of patterns per image with P for the LBP and LBsP descriptors. For the LBsP descriptor $n=2$ sub-patterns are considered. The left part of the figure is zoomed in to better show the difference between both descriptors. For instance, whereas LBP produces 256 pattern types for $P=8$ neighbors, LBsP just produces 16.	34
3.12	Temporal sampling procedure with $S=4$ samples, applied to a video sequence of length $L=16$. (a) Δ_e , δ_{max} and s_0 are computed. Then, a uniform sampling is applied to select the green dotted frames. Green frames are the ones that belong to the random shifting interval. (b) Random shifting is applied to the green dotted frames to obtain the remaining s_i samples (red frames), from which spatial features will be extracted.	37

3.13	Temporal Pyramid Matching. For each level l , the video sequence is partitioned into 2^l sub-sequences. S-LBsP descriptors extracted from each sub-sequence are averaged by means of the average pooling function. The resulting z_{lj} histograms are concatenated to form the final feature vector representing the video sequence.	38
3.14	Proposed hand gestures (visual examples).	40
3.15	The above figures show the response of the system while a user executes a natural sequence of hand gestures to use the mouse application. (a) shows the trajectory of the cursor in blue. Once the gesture is recognize as <i>cursor</i> , the system responds providing the location of the hand frame by frame. (b) and (c) show the recognition of a <i>left click</i> and a <i>right click</i> at the end of four temporal windows, where the video descriptor TPM-SLBsP is evaluated, that belong to the same execution sequence. Notice how the system responds even before the gestures end due to the high discriminative power of the descriptor. (d) and (e) show the recognition of the static gestures: <i>ok</i> and <i>fist</i> .	49
4.1	The proposed structured CNN for vehicle detection carries out a pixel-level classification from a single RGB image. Its architecture consists of a FCN that lacks pooling layers to preserve the spatial resolution. It contains 5 blocks of convolutional and ReLU layers to compute a compact representation of the local context of every pixel. Then, a sigmoid layer provides a probability of belonging to the <i>vehicle</i> class for every pixel. In the notation above, the number of kernels is indicated before the @, and then the kernel's size for every convolutional layer. The stride of the convolutions is 1.	54
4.2	Examples of input patches and their corresponding labels from GRAM-RTM dataset [5]: (a) original image, (b) 150×150 image patches after applying augmentation that are used as input to the network, and (c) labels corresponding to the above image patches, where bounding-box annotations have been reduced 20% regarding their original size.	55
4.3	Visual results in four different scenarios (columns) in the M-302 sequence. The first row shows the network output as a probability map, where red means high probability of belonging to the <i>vehicle</i> class, and dark blue very low one. The second one shows the binary masks extracted from the probability maps after applying a non-maxima-suppression threshold. The third row shows the segmented area (in red) overlapped with the original frame. The fourth row shows the bounding boxes computed from the segmented area: green bounding boxes are predicted by the network, and red ones are from ground truth.	57

4.4	Visual results in four different scenarios (columns) in the M-302-HD sequence. The first row shows the network output as a probability map, where red means high probability of belonging to the <i>vehicle</i> class, and dark blue very low one. The second one shows the binary masks extracted from the probability maps after applying a non-maxima-suppression threshold. The third row shows the segmented area (in red) overlapped with the original frame. The fourth row shows the bounding boxes computed from the segmented area: green bounding boxes are predicted by the network, and red ones are from ground truth.	58
4.5	Examples of network predictions outside the ROI for the M-302-HD sequence. Red-doted line indicates the boundary of the ROI, so that vehicles outside it are in the upper-left region.	59
4.6	Architecture of the proposed network for hand-gesture recognition. It predicts the category of a gesture from a 320×240 grayscale image. It comprises 9 convolutional layers for feature extraction, and 3 FC layers for classification. Each FC layer is followed by a dropout unit with a 0.5 dropout ration to prevent overfitting. The last layer consists of a softmax function with 7 units corresponding to 7 gesture categories. In the notation above, first the kernel's size, then the number of filters, and eventually the stride if it is different from 1, are indicated for each convolution, and the output dimension for each FC layer.	62
4.7	Collected dataset: examples of the seven gestures embedded in the two types of background. From left to right and top to bottom: <i>fist</i> , <i>L</i> , <i>OK</i> , <i>palm</i> , <i>pointer</i> , <i>thumb down</i> , and <i>thumb up</i>	63
4.8	Examples of correct classification. The first row shows simple backgrounds, and the second one complex backgrounds. The caption of each sub-figure indicates the inferred class and its corresponding confidence score.	64
4.9	Examples of misclassification. The first row shows simple backgrounds, and the second one complex backgrounds. The caption of each sub-figure indicates the inferred class and its corresponding confidence score.	65
4.10	(a) DroNet is a forked Convolutional Neural Network that predicts, from a single 200×200 frame in gray-scale, a steering angle and a collision probability. The shared part of the architecture consists of a ResNet-8 with 3 residual blocks (b), followed by dropout and ReLU non-linearity. Afterwards, the network branches into 2 separated fully-connected layers, one to carry out steering prediction, and the other one to infer collision probability. In the notation above, first the kernel's size, then the number of filters, and eventually the stride if it is different from 1, are indicated for each convolution.	70
4.11	(a) Udacity images used to learn steering angles. (b) Collected images to learn probability of collision. The green box contains no-collision frames, and the red one collision frames.	72

4.12	Model performance: (a) Probability Density Function (PDF) of actual vs. predicted steerings on the Udacity dataset testing sequence. (b) Confusion matrix on the collision classification evaluated on testing images of the collected dataset. Numbers in this matrix indicate the percentage of samples falling in each category.	74
4.13	Testing environments: (a) <i>Outdoor 1</i> is a 90° curve with a dead end. This scenario is also tested with the drone flying at high altitude (5 m), as shown in Figure 4.14. (b) <i>Outdoor 2</i> is a sharp 160° curve followed by a 30 m straight path. A closer view of this environment can be seen in (f). (c) <i>Outdoor 3</i> is a series of 2 curves, each of approximately 60° , with straight paths in between. Moreover, DroNet is tested on scenarios visually different from the training ones, such as (d) an indoor parking lot, and (e) an indoor corridor.	75
4.14	High altitude Outdoor 1: In order to test the ability of DroNet to generalize at high altitude, the drone is flown at 5 m altitude in the testing environment <i>Outdoor 1</i> . Table 4.6 indicates that the proposed policy is able to cope with the large difference between the viewpoint of a camera mounted on a car (1.5 m) and the one of the UAV.	76
4.15	DroNet predictions. The above figures show predicted steering and probability of collision evaluated over several experiments. Despite the diverse scenarios and obstacles types, DroNet predictions always follow common sense and enable safe and reliable navigation.	77
4.16	Activation maps: spatial support regions for steering regression in city streets, on (a) a left curve and (b) a straight path. Activations on (c) an indoor parking lot, and (d) an indoor corridor are also shown. It can be observed that the network concentrates its attention to “line-like” patterns, which approximately indicate the steering direction.	78
4.17	Block diagram of the proposed approach. The output of the event camera is collected into frames over a specified time interval T , using a separate channel depending on the event polarity (positive and negative). The resulting synchronous event frames are processed by a ResNet-inspired network, which produces a prediction of the steering angle of the vehicle.	81
4.18	Different input data extracted from the DDD17 dataset for the four lighting conditions. The first row depicts grayscale images collected with a traditional camera (frames of the DAVIS). The second row shows images created by difference of two consecutive grayscale images. The third row corresponds to time-integrated event images (here displayed as single-channel images, $h^+ - h^-$, to highlight the similarity with those on the second row).	83
4.19	Maps of dataset showing (a) the whole recorded route, covering 1000 km of different roads in Switzerland and Germany; (b) and (c) training and testing frames, in red and blue respectively, for the <i>day_sun</i> subset (3 hours of driving). The training sections are different from the testing sections, corresponding to 40 and 20 seconds of recording respectively.	84

4.20	Events collected for different durations of the interval T (cf. Figure 4.17). The scene corresponds to a <i>day_sun</i> sequence, with the car turning left in an urban environment.	86
4.21	Distribution of the relative error in steering angle prediction as a function of the ground truth steering angle and the event integration time. The performance of the network trained on large integration time (200 ms) degrades for large steering angles. In contrast, the network trained on small integration time (10 ms) predicts well large steering angles, but its performance degrades for smaller angles. The network with an intermediate integration time (50 ms) performs best for large and moderate steering angles. For small angles ($<5^\circ$), small absolute errors in the angle produce large relative errors regardless of the integration time.	86
4.22	Variation of the RMSE of ResNet50 with respect to the event accumulation time (3rd column of Table 4.7).	87
4.23	Comparison of training losses for ResNet50 with and without ImageNet initialization.	90
4.24	Steering angle regression performance on grayscale frames (first column) and on event camera data (second column). The first row shows a sunny day where visual features can be extracted from grayscale frames. However the camera saturation and the lack of temporal information makes the network predict a wrong steering angle. The second row shows a night scene, from which the network hardly predicts the correct steering angle when using grayscale frames. The proposed method accurately predicts a steering angle by combining event data and deep learning in both scenarios.	91
A.1	Local binary pattern from a pixel neighborhood: (a) 3x3 gray scale neighborhood. (b) Differences between the center pixel and its neighbors. (c) Thresholded neighborhood. The binary pattern is converted to decimal and represents one of the histogram's labels. (d) Histogram of LBP from the whole image.	97
A.2	Circularly symmetric neighbor sets for different P and R (extracted from [6]). . .	98
A.3	Temporal extensions of LBP: (a) procedure to compute VLBP _{1,4,1} , and (b) procedure to compute LBP-TOP _{4,4,4,1,1,1}	100

List of Tables

2.1	Summary of several benchmarks for image classification.	18
3.1	Comparison between LBP and LBsP in terms of number of pattern types per pixel and per image, and descriptor dimension.	34
3.2	Precision, Recall, and F1-score metrics for the best parameter configurations of the LBP descriptor.	42
3.3	Precision, Recall, and F1-score metrics for different parameter configurations of the S-LBP descriptor.	42
3.4	Precision, Recall, and F1-score metrics for different parameter configurations of the S-LBsP descriptor.	42
3.5	Comparison of the performance of different state-of-the-art image descriptors on the NTU database in terms of average accuracy.	44
3.6	Average accuracy for different parameter configurations of the VLBP descriptor. .	45
3.7	Average accuracy for different parameter configurations of the LBP-TOP descriptor.	45
3.8	Average accuracy for different parameter configurations of the VS-LBsP descriptor.	45
3.9	Average accuracy for different parameter configurations of the TPM-SLBsP descriptor.	46
3.10	Comparison of the accuracy and dimensionality among different methods using the whole system on the proposed dataset.	48
4.1	Detection results in terms of precision, recall, and F1 score for the testing sequences in GRAM-RTM dataset [5].	60
4.2	Confusion matrix for hand-gesture recognition on simple backgrounds.	65
4.3	Confusion matrix for hand-gesture recognition on complex backgrounds.	66
4.4	Comparison among the proposed network, AlexNet and VGG19 in terms of accuracy and GPU processing time per image.	66

4.5	Quantitative results on regression and classification tasks: EVA and RMSE are computed on the steering regression task, while average accuracy and F1-score are evaluated on the collision prediction task. The on-line processing time in frames per second (fps) is additionally reported, achieved when receiving images at 30 Hz from the UAV.	74
4.6	Average travelled distance before stopping: navigation results using three different policies on several environments. Recall that [7] uses only collision probabilities, while DroNet uses also predicted steering angles. <i>High Altitude Outdoor 1</i> consists of the same path as <i>Outdoor 1</i> , but flying at 5 m altitude, as shown in Figure 4.14	76
4.7	Comparison of the ResNet50 performance for the different integration times on <i>day_sun</i> subset.	87
4.8	Results for <i>day</i> subset.	88
4.9	Results for <i>day_sun</i> subset.	88
4.10	Results for <i>evening</i> subset.	88
4.11	Results for <i>night</i> subset.	88
4.12	Comparison between two state-of-the-art learning approaches using grayscale frames [8, 9] and the proposed networks that process event frames, for the whole dataset [10]. For ResNet50, both random and ImageNet initializations have been evaluated. . .	89

Glossary

HCI	Human-Computer Interaction
SVM	Support Vector Machine
BoW	Bag of Words
LBP	Local Binary Pattern
S-LBP	Spatiograms of Local Binary Patterns
S-LBsP	Spatiograms of Local Binary sub-Patterns
VLBP	Volume Local Binary Pattern
LBP-TOP	Local Binary Pattern of Three Orthogonal Planes
VS-LBP	Volumetric Spatiograms of Local Binary Patterns
TPM	Temporal Pyramid Matching
DL	Deep Learning
SGD	Stochastic Gradient Descent
DNN	Deep Neural Network
CNN	Convolutional Neural Network
FCN	Fully-convolutional network
ReLU	Rectified Linear Unit
FC	Fully-connected
LSTM	Long-Short Term Memory
ResNet	Residual network
UAV	Unmanned Aerial Vehicle
DVS	Dynamic Vision Sensor

Chapter 1

Introduction

1.1 Motivation

Research efforts on computer vision have led to the development of systems that learn from and make decisions on data, for which two strategies have been adopted. The first one consists of manually designing features that facilitate the learning process by removing redundant and useless information. Although this way of proceeding has been widely and successfully applied for the last decades, it has reached a plateau because performance barely improves. The second one automatically learns appropriate features from raw data through Deep Neural Networks (DNN), which have outperformed traditional approaches in the last years.

The reason why DNNs are changing the paradigm of machine learning is mainly three-fold. Firstly, they avoid the extremely hard work of feature engineering, since they automatically learn features. Secondly, the appearance of large, high-quality labeled data allows networks to appropriately learn complex models, which are composed of even millions of parameters. And lastly, the development of specific hardware makes it possible to speed up their learning processes, and train more complex (deeper) architectures, which leads to lower error rates. In particular for computer vision, the key aspect that has contributed to the success of DNN, is the emergence of Convolutional Neural Networks (CNN). These networks make the explicit assumption that the inputs are images, taking advantage of their spatial structure to efficiently implement the forward function, and considerably reduce the number of parameters in the network.

Another major characteristic of DNNs is the training of end-to-end systems, which learn a mapping function from input to output directly with just a single neural network. This has left behind traditional systems that require multiple stages of processing to perform intermediate tasks, such as detection, feature extraction, tracking and classification. And each of these stages are in turn comprised of multiple techniques to face specific difficulties, such as illumination changes, occlusions, dynamic scenes, different viewpoints, etc. Unlike them, deep learning systems can act as full pipelines to solve several tasks at once, going from the raw input data to the final output. This approach could involve the concatenation of multiple networks such as CNNs and Long-Short Term Memory (LSTM), which are trained simultaneously. Therefore, a traditional hand-gesture

recognition system, as the one shown in Figure 1.1, can be replaced by a single network that learns the labels of the gestures directly from input images. In addition to achieving a better performance, end-to-end learning saves a lot of time, and reduces the effort of human design.

Traditional pipeline:

End-to-end approach:

Figure 1.1: Multi-stage traditional hand-gesture systems can consist in detecting the hand by skin segmentation, computing a feature vector based on its shape from the previous binary mask, and using a classifier to categorize it. In contrast, a deep neural network can learn a semantic representation of the hand and categorize it in an end-to-end manner.

These advances may awaken interest in immature applications, such as hand-gesture recognition. A few years ago, there was great interest in Human-Computer Interaction (HCI) systems to achieve a more natural, intuitive, and user-friendly interface to interact with computers. Since hand gestures seem the most intuitive way to create this interaction, a lot of research was done to design and build robust and efficient hand-gesture recognition systems. Algorithms based on color imagery did not have much success due to the hard job of implementing a user-independent system which is robust against varying illumination conditions, cluttered backgrounds, occlusions, etc. However, other more robust approaches that make use of gloves or other sensors, such as time-of-flight, and stereo cameras, did not gain much acceptance either because they were uncomfortable, and required high-costly special hardware. Therefore, end-to-end deep learning through CNNs could solve the aforementioned drawbacks, even with a single RGB camera, which is very attractive since these sensors are present in most consumer devices, are very cheap, and minimize the learning process required by the user.

Besides, as deep learning (DL) become more mature in computer vision applications, it is spreading to more complex scenarios, such as robotics. This field presents several challenges that directly affect the learning process of a DNN: generating training data is generally expensive, and sub-optimal performance can pose a danger for users (human-robot cooperation), the environment, and the robot itself (collision avoidance). Nevertheless, research efforts are finding creative solutions to deal with the lack of training data, for instance, via simulation, and their high complexity when operating in safety-critical environments.

This thesis presents the recent evolution in the design of computer vision systems: from an expert-knowledge based design to a DL end-to-end design. Both of them present a central problem in computer vision, which is separating signal (relevant information) from noise (irrelevant information) under different challenging conditions of illumination, cluttered backgrounds, occlusions, changes in viewpoint, and dynamic environments. This thesis provides contributions to solving this problem in different computer vision applications, such as vehicle detection, hand-

gesture recognition, and steering prediction for autonomous robot navigation.

1.2 Contents

Besides this introductory chapter, this thesis contains another four chapters and an appendix, whose contents are briefly outlined next.

Chapter 2 presents the most relevant work in the literature to solve different computer vision tasks from two different viewpoints: multi-stage learning through an expert-knowledge design (Section 2.2), and end-to-end deep learning (Section 2.3). The first one explains the multiple stages in which a traditional machine learning system is divided to solve intermediate stages, such as detection, tracking, feature extraction and inference, under different conditions. The second one focuses on the change of paradigm of machine learning in the DL era. In particular, the design of CNNs, popular architectures, and mechanisms to deal with different difficulties are explained in this second part.

In Chapter 3, a robust hand-gesture recognition system relying on traditional machine learning is proposed. Its design comprises three stages: detection, tracking, and recognition, to locate and recognize a hand gesture in both spatial and temporal domains. Most posterior stages are developed to deal with the errors inherited from the previous ones. Novel LBP-based image and video descriptor are integrated into the detection and recognition stages respectively, to distinguish among different gesture categories, and increase the recognition rate. The performance of such image and video descriptors, as well as the whole system, are evaluated on several datasets and compared against state-of-the-art hand-gesture recognition systems. Results show that the proposed system pipeline improves the performance of the whole system, and the discriminative power of the proposed descriptors achieves very high recognition rates. In order to facilitate the understanding of the descriptors' design based on LBP, a further explanation of the LBP principle, as well as its most relevant spatial and temporal extensions can be found in Appendix A.

Chapter 4 presents several end-to-end systems based on deep learning for different computer vision applications. A DL approach for vehicle detection at multiple scales is proposed in Section 4.2. A simpler fully-convolutional architecture is developed to only learn local features, and provide the location of the vehicles through image segmentation. The performance of the proposed DL system is evaluated on a traffic monitoring environment, where a high detection rate is obtained. In Section 4.3, a hand-gesture recognition network is proposed to deal with hand gestures that occupy a small region of the image and are embedded in clutter, without any previous location stage. This approach is evaluated in both simple and complex scenarios, being capable of achieving above 97% in the former, and 85% in the latter. In Section 4.4 a DL approach called DroNet is proposed to safely and autonomously fly a drone in a urban environment. The proposed system transfer learning from manned vehicles such as cars and bicycles to imitate their behavior, and predict the steering angle and the probability of collision. Excellent results are obtained for both steering and probability predictions in comparison with other works in the literature. In addition, experiments on a real drone not only prove a soft and reliable behavior in the streets of city, but also its generalization power to indoor environments such as corridors and parking lots. In Section 4.5, an event camera is used along with a DL algorithm to solve the steering prediction task for

self-driving cars. This sensor naturally responds to motion in the scene, so it makes the learning process easier. Its asynchronous output is converted into synchronous frames to unlock its potential through state-of-the-art deep architectures, which are adapted to solve steering prediction from a regression viewpoint. This sensor-algorithm combination outperforms other works in the literature based on standard cameras.

Chapter 5 summarizes the proposed work, and elaborates on the conclusions.

1.3 Contributions

This thesis contributes to several problems in computer vision through different systems that reflect how this field has recently evolved due to the renaissance of neural networks. The contributions of this thesis have been published in several peer-reviewed conferences and journals as detailed next:

- A robust hand-gesture recognition system designed for addressing the location and recognition of hand gestures along spatial and temporal domains under an HCI scenario was published in [11].
- Two novel LBP-based video descriptors designed for improving the performance of hand-gesture recognition systems were published in [11, 12]. The descriptors provide richer spatio-temporal information than the state of the art while having a manageable computational cost.
- A structured CNN for vehicle detection without relying on multi-scale region proposal stages was published in [13]. The CNN architecture provides the location of vehicles through image segmentation in highway environments for traffic monitoring.
- A deep neural network designed for recognizing hand gestures from a semantic viewpoint, without any previous segmentation or detection stage, was published in [14]. The network achieves above 85% recognition rate in scenarios with highly cluttered backgrounds, where the hand occupies a small region of the image.
- The first deep learning approach designed for safely and efficiently flying a quadrotor in urban environments while avoiding obstacles was published in [15]. The DL approach predicts the steering angle and the collision probability directly from images, which are used to command the quadrotor, thus coupling the long-standing separate problems of “perception/sensing” and “control/actuation”.
- The first deep learning approach designed for unlocking the potential of event cameras on a steering regression problem was published in [16]. This sensor-algorithm combination outperforms state-of-the-art systems based on traditional, frame-based cameras under a self-driving car scenario.
- The release of datasets and code associated to several of the aforementioned contributions.

Chapter 2

Relevant work

2.1 Introduction

Automating the recognition task that the human visual system does is still a challenge, since it is necessary to face all the possible conditions under which a machine must learn, while taking into account the context. The traditional way of addressing these difficulties has been building learning systems with multiple stages to carry out intermediate tasks, and therefore, make the learning process easier. Although CNNs are able to address such tasks in an end-to-end manner, they still have to face the same challenges as traditional approaches, such as varying lighting conditions and viewpoints, distortions, occlusions, and cluttered backgrounds. However, the potential that DL offers in terms of performance and generalization power has made the interest among researchers increase, who are adapting computer vision tasks to DL very fast.

This chapter presents different efforts to address vision-based learning from the two viewpoints: traditional multi-stage machine learning, and end-to-end deep learning. Section 2.2 shows the most popular techniques used in the different stages that are involved in the former. Section 2.3 introduces the approach of DNN focused on computer vision-related tasks, and presents different network proposals that solve them under an end-to-end learning.

2.2 Multi-stage learning systems

This section will present the most relevant approaches that have been used to build different modules of a traditional supervised learning approach. Although the type and number of modules depends on the application, this section focuses on the four main stages that a visual recognition system can contain to carry out the object recognition task: object detection, feature extraction, tracking and learning stages.

2.2.1 Object detection

The aim of the detection stage is to generate a set of candidate bounding boxes that are likely to contain an object of interest. This task has been widely addressed as a classification problem that uses a sliding window strategy to exhaustively extract region candidates. Other methods restricts the number of candidate regions by relaxing the category of the object, such as Binarized Normed Gradients (BING) [17] and EdgeBoxes [18]. The idea of these approaches is to generate a small set of bounding boxes to reduce the number of regions that need to be further analyzed. BING [17] consists of a cascaded Support Vector Machine (SVM) framework over edge features that is applied in a sliding window fashion. EdgeBoxes [18] proposes candidate bounding boxes also in a sliding window manner, and evaluates them by computing a scoring function based on object boundaries estimates. Previous methods have in common the use of simple image features that are fast to compute. In contrast, a multi-class classification of the candidate regions is performed using more sophisticated feature descriptors, such as the ones that will be seen in Section 2.2.2. These descriptors allow the classifier to discriminate among different categories, so that detection and recognition are performed at once.

An evolution of the previous techniques is the so called region proposal method. This approach shares the idea of proposing a small set of candidate bounding boxes regardless of the type of object, but avoiding the exhaustive search of the sliding window. In general, these methods can be divided into three categories according to their mechanism to generate candidate bounding boxes representing object proposals: measuring “objectness” [19][11], providing multiple foreground-background segmentations [20][9, 10], and merging superpixels [1][8]. An objectness function usually requires to learn multiple cues to be able to enhance the locations of the objects. For instance, the objectness measure used in [19] quantifies how likely a window contains an object (of any category) by combining saliency, color contrast, edge density, and superpixel segments. The proposed set of windows is sampled from the distribution given by the saliency cues. On the other hand, segmentation-based approaches aims at exploiting the structure of the image to generate object locations. However, since images and the different uses of objects are hierarchical, multiple partitions are required, which is a complex task. To generate a foreground-background segmentation, Constrained Parametric Min-Cuts (CPMC) [20] generates multiple seed regions that are ranked using a regressor to predict objectness based on low and mid-level features. Regarding merging superpixels, Selective Search [1] is based on a hierarchical grouping algorithm of superpixels to generate locations, as it is described in Figure 2.1, and score functions that measure color and texture similarities.

2.2.2 Feature extraction

The feature extraction stage usually consists of computing a set of local features that distinctively represent an image region based on its local structure. The resulting local descriptors are aggregated into a very compact vector representation that describes the whole image region. The aim of this stage is two-fold. The first one is to reduce the dimensionality of the input image region by effectively removing irrelevant and redundant data. The second one is to capture the most significant information related to the task contained in the image to increase the learning accuracy. In addition to being computationally efficient, it is desirable that feature descriptors have certain

Figure 2.1: Selective Search algorithm for object detection (extracted from [1]).

attributes, keeping in mind that they depend on the actual application:

- **Discriminability:** visually similar image regions should have similar descriptors, and visually different image regions should have different descriptors.
- **Invariance:** despite the transformation (i.e. rotation, scale, translation), two visually similar image regions should still have similar descriptors.
- **Robust:** visually similar image regions should have similar descriptors despite the distortions (i.e. illumination changes).

During the last three decades, several local descriptors have been proposed to describe image and video regions by exploiting their color, appearance, structure, and motion properties. The shape and size of such regions depends on the nature of the description mechanisms. Some of the most popular image descriptors for object recognition are described below. They can be categorized into point descriptors and region descriptors. The Scale Invariant Feature Transform (SIFT) algorithm [21] is the most used keypoint descriptor. It first detects a set of keypoints over a number of scales, and then computes a descriptor for each keypoint which is invariant against scale and rotation. The SIFT descriptor samples the magnitudes and orientations of the image region gradient in a 4×4 region around each keypoint, and builds histograms with the 8 dominant edge orientations. The resulting descriptor consists of a 128-dimensional feature vector for each keypoint, which is finally normalized to unit length to gain invariance against affine illumination changes. A more efficient alternative to SIFT is the Speeded-Up Robust Features (SURF) descriptor [22]. The resulting descriptors can often be not straightforward to use due to their large number or their high dimensionality. More compact representations can be formulated by the so called Bag of Words (BoW) techniques. They consist in generating a codebook from all the descriptors, and computing some pooling function to aggregate them, creating a unique representation of the image region.

Regarding region descriptors, the Histograms of Oriented Gradients (HOG) descriptor [23] and Local Binary Patterns (LBP) [24, 6] are the most popular. HOG is similar to SIFT, but it relies on computing local region gradients over a dense grid of uniformly spaced image blocks, instead of at keypoints. Local contrast normalization on overlapping blocks is used to improve the accuracy. The resulting descriptor is a feature vector containing the elements of the normalized

blocks histograms from all of the block regions. The LBP descriptor characterizes the spatial structure of a pixel neighborhood by thresholding its intensity values with respect to the the center pixel, as shown in Figure 2.2. The thresholded values are concatenated in an 8-bit binary number that indicates the LBP of each pixel. Finally, the LBPs extracted from all the pixels are used to generate a histogram of $2^8 = 256$ labels that compactly represents the image region. The LBP descriptor is invariant against monotonic illumination changes. One of its multiple extensions allows it to be invariant against rotations [25].

Figure 2.2: Computation of the LBP operator: (a) 3x3 gray scale neighborhood, (b) differences between the neighbor pixels and the center one, (c) thresholded neighborhood differences, and (d) histogram of LBP (H-LBP) from the whole image.

On the other hand, video descriptors have been designed to extend the feature analysis to the spatio-temporal domain. A popular way to address spatio-temporal features is by combining appearance and motion cues. This is the case of the Spatio-Temporal Interest Points (STIP) [26] descriptor. Similar to SIFT, STIP extracts spatio-temporal keypoints using the 3D Harris corner detector, and describes their 3D local neighborhood by computing HOG in terms of local spatial structure, and Histograms of Optical Flow (HOF) in terms of motion. Moreover, the spatio-temporal versions of SIFT, HOG and LBP have been proposed: SIFT3D [27], HOG3D [28], and Volumes of Local Binary Patterns (VLBP) and Local Binary Patterns from Three Orthogonal Planes (LBP-TOP) [29]. SIFT3D and HOG3D extends the analysis of gradients to a 3D neighborhood. The idea behind VLBP is to use three parallel planes to compute a volume local binary pattern. However, its large dimensionality can restrict its applicability. LBP-TOP tries to reduce the dimensionality by only concatenating LBP histograms from three orthogonal planes: XY (appearance information), XT (horizontal-temporal space), and YT (vertical-temporal space), which intersect in the center pixel. An overview of both VLBP and LBP-TOP is shown in Figure 2.3.

2.2.3 Object tracking

The aim of tracking is to locate and identify all objects of interest for each frame in a video, so that the object identities are consistent throughout the video sequence. Single and multiple object tracking are challenging tasks. In addition to difficulties such as scale, translation, and rotation changes that arise because of the motion of objects and camera, and camera distortion, objects may be partially or completely occluded, or even temporarily leave the field of view. Moreover, objects whose paths intersect might confuse the model, and cause it to erroneously switch their IDs.

Figure 2.3: Temporal extensions of LBP: (a) VLBP, and (b) LBP-TOP.

A large number of object tracking algorithms have been proposed in the last decades to solve one or more of these challenges. In particular, tracking has mainly focused on finding good features, object representations, and motion models to improve the tracking quality. In general, tracking algorithms can be divided into two parts: observation model and matching/correspondence process. The former localizes all objects in the image using an object detector, as the ones described in Section 2.2.1, which usually extracts features to distinguish among different objects, as the ones seen in Section 2.2.2. The latter associates detected objects between frames to their trajectories using the previously computed features.

The most common observation models are based on appearance, motion, and location. Appearance models describe objects or distinguish them from the background by using image descriptors, such as HOG [30] and LBP [31], and other low level features, such as color, shape, and super-pixel, which can be fused [32, 33], or used individually. On the other hand, motion cues are very important since they can reduce the search space when predicting potential positions in future frames. Motion models can be linear and non-linear. Linear ones assume that motion is linear over short periods of time, and use a linear regressor to estimate the velocity [30]. Non-linear models deal with those cases where linear models fail, such as in unexpected camera movements [34]. Other methods use spatio-temporal relationships to exclude possible objects [35].

Regarding matching approaches, they can be generally categorized into two classes depending on how they link the observation to the target: Bayesian methods and data-association-based methods. The two most popular Bayesian methods are Kalman filtering and particle filtering. The Kalman filter optimally estimates the state of a linear dynamic system. It is very efficient when the number of objects is small, but it fails when the number of objects increases, since identifier (ID) switches become more frequent and are difficult to correct due to their recursive nature [36, 37]. Similarly, particle filtering estimates the states of a non-linear dynamical model, but it is not appropriate for high-dimensional state spaces due to its computational complexity [38, 39].

The common algorithm based on data association is the Hungarian algorithm, which relies on a local optimization, where only a few frames are considered for solving the association problem [40, 41].

2.2.4 Supervised learning

The recognition task is carried out by a learning algorithm that has two stages, one in which it learns from input data, commonly called training stage, and another one in which it uses this learning to classify a new observation. For the training stage, a supervised algorithm needs example input data and the desired output to learn a general rule that maps inputs to outputs. Supervised learning is called linear regression if the prediction is a real-valued output, and classification if the output is discrete. Regression models the relationship between input and output, which is iteratively refined by minimizing a measure of the error in the predictions. Regarding classifiers, they are categorized into two classes: generative and discriminative. The fundamental difference between them is that the former model the distribution of individual classes, and the latter learn the boundary between classes. Generative models learn the joint probability distribution $p(x, y)$ given the input x and the label y . They can be used to generate data or transformed into $p(y|x)$ using Bayes Theorem to make predictions. On the other hand, discriminative classifiers learn the conditional probability distribution $p(y|x)$ directly, or learn a mapping function from inputs x to the class labels y .

One of the most popular generative approaches are Hidden Markov Models (HMM). They infer the behaviour of their hidden states and transitions from the observed sequences, which typically are a series of associated features. On the other hand, Random forests (RF) [42], SVM [43], and AdaBoost [44], are the most relevant algorithms among discriminative classifiers. RF is an ensemble bagging algorithm that aims to reduce the variance of a large number of independent and parallel classifiers, in this case decision trees, over different sub-samples of the data. Then, a voting or average strategy is taken across all the trees' predictions. AdaBoost also combines the outputs of other "weak" classifiers to make a decision. However, these classifiers are trained sequentially, so that the results of previous individual classifiers are boosted and reweighted over many iterations to converge to a stronger classifier. SVM is a linear binary classifier that finds a hyperplane to separate the input samples into two classes, by minimizing the margin between the closest samples belonging to different classes. The so called "kernel trick" [45] is widely used along SVM to be applied to non-linearly separable data.

Because generative models are good at classifying sequential data, they are frequently used to recognize spatio-temporal patterns [46]. In the case of discriminative models, the spatio-temporal nature of input data can be modeled by a descriptor [47]. Discriminative models have proven to outperform generative ones when the amount of data is large [48]. Therefore, as is the case with feature extraction methods, the relative performance of learning algorithms depends on the application, and varies with respect to the characteristics of the observed data.

2.3 End-to-end deep learning systems

This section will first present an overview of the fundamentals of deep convolutional neural networks, and briefly describe recurrent networks to deal with sequential data. Secondly, it will present the latest DL advances to solve image classification, object detection, image segmentation, and tracking, from an end-to-end viewpoint, which are outperforming techniques described in the previous section.

2.3.1 Convolutional Neural Networks overview

The motivation of building multilayer networks to solve more complex tasks in an end-to-end manner came with Multi-Layer Perceptrons (MLP). These kinds of DL networks vectorize the input, and are composed by stacked linear and activation layers, whose neurons are densely and mutually interconnected. Therefore, MLP do not scale well to computer vision problems. Unlike them, CNNs are specifically designed to deal with images, and take advantage of its spatial structure to efficiently implement the forward function with convolution operations, and reduce the number of involved parameters.

Despite deep learning algorithms being available since the 1980s, they were too computationally expensive to allow a great deal of research on the hardware available. In particular, the first multi-layer CNN trained in a fully supervised manner using backpropagation was LeNet [49], which recognized individual characters. However, the turning point happened when AlexNet [50] won the ILSVR challenge on the ImageNet dataset [51] in 2012. This was mainly caused by two facts: the large available amount of data, and the development of enhanced hardware to facilitate the computation. As a result of the obtained accuracy performance, computer vision-related tasks have swept hand-crafted models aside. One of the main reasons is that CNNs have the ability to automatically learn complex and hierarchical representations from raw images, and carry out the given task.

2.3.1.1 CNN layers

A CNN can be divided into two parts which are designed for a different purpose, as shown in Figure 2.4. The first one comprises convolutional and pooling layers to perform feature extraction. Lower convolutional layers aim to capture spatial-invariant low-level features, such as edges, corners, and color blobs, and higher layers extract more abstract features. The second one is made up of fully-connected layers to perform the given task, which is usually the classification as in Figure 2.4. Both parts combine linear and non-linear layers to address the non-linearity present in data, and be able to learn any function that can map the input image into the desired output. The usual way of building a deep CNN architecture is by stacking several blocks of convolutional, non-linear, and subsampling layers. After the last subsampling layer, a specific number of fully-connected layers alternated with non-linear units are added until getting the desired output. The last layer, if necessary, generally is a non-linear unit that performs the given task.

Figure 2.4: Example of a CNN that classifies a given input image into three categories: *bird*, *car*, and *dog*. In the feature extraction module, convolutional and pooling layers are stacked to compute the shown feature maps, which encode spatial-invariant features of the image. Next, the classification stage comprises a series of fully-connected layers that map a vectorized representation of the very last feature map into another vector with the desired output dimension. The final layer is an activation function that carries out the classification task, and provides the predicted probabilities for each category.

Convolutional layers (conv layers)

Convolutional or conv layers consist of D' linear filters $w \in \mathbb{R}^{F \times F \times D}$, with F being the width and height, and D the number of channels of each filter. Given an input image $I \in \mathbb{R}^{W \times H \times D}$, where W , H and D are width, height and number of channels of the image, respectively, a conv layer computes dot products between each of its filters and the image I to produce a filtered image $I' \in \mathbb{R}^{W' \times H' \times D'}$ as follows:

$$I'(x, y)_d \doteq \sum_{i=0}^{F-1} \sum_{j=0}^{F-1} \sum_{k=1}^D w_d(i, j, k) I(x+i, y+j, k), \quad d = 1, \dots, D', \quad (2.1)$$

where $I'(x, y)_d$ is the d -th channel of the filtered image I' , whose spatial dimensions width and height are W' and H' respectively, and w_d is the d -th filter of the conv layer. The resulting filtered images along the network are also called feature maps, and the weights of those filters are the parameters to be learned during training.

Conv layers take advantage of the spatial structure of images to connect each neuron to only a local region of the input image. Neurons of a conv layer share parameters over the whole image, which provides a way to reduce the number of parameters with respect to MLPs. Notice the asymmetry in how spatial dimensions W and H are treated with respect to the third dimension D . Whereas neurons locally connect to the input image along spatial dimensions, they fully connect along all the channels of the image as shown in Figure 2.5.

In addition to the size of the filters or receptive field F , and the number of filters D' , conv layers are defined by the stride or step size of the convolution S , and other properties such as how they treat the image boundaries (padding), which are called the hyperparameters of these layers.

Figure 2.5: In this example, the convolution between an input image $I \in \mathbb{R}^{9 \times 8 \times 4}$ and the first convolutional filter $w_1 \in \mathbb{R}^{3 \times 3 \times 4}$ produces the first channel of the filtered image $I'_1 \in \mathbb{R}^{7 \times 6 \times 1}$. The convolution between the image and the remaining $D' - 1$ filters of the layer will produce the rest of the channels of the filtered image $I' \in \mathbb{R}^{7 \times 6 \times D'}$.

Therefore, the resulting dimensions W' and H' can be computed as:

$$W' \doteq \frac{W - F + 2P}{S} + 1, \quad (2.2)$$

and

$$H' \doteq \frac{H - F + 2P}{S} + 1, \quad (2.3)$$

respectively, where P is the amount of zero-padding used.

The sliding-window nature of convolutions provides the translational invariance property, which means the network can recognize the same feature anywhere in the image. Convolution is not naturally invariant to other transformations such as scale or rotation changes, however it can learn to be. The large number of filters in conv layers makes it possible to learn the same feature at different scales and with different rotations. The large response of the filters to any of these features is forward propagated through the network by means of pooling layers, which will be described in next section.

Pooling layers

A pooling layer can be seen as a non-linear filter which consists of an aggregation function that summarizes the region under its receptive field. It is independently applied to each channel

of the feature maps and does not introduce learnable parameters. The goal of pooling layers is to provide invariance against minor local changes, and to progressively reduce the dimension of the feature maps. That is why they are also called subsampling layers, and are used in-between successive conv layers. They are defined by the nature of the aggregation function, their receptive fields, strides, and other parameters such as padding. The most common aggregation function is *max pooling*, which takes the maximum value (see Figure 2.6). It has been proven to work better in practice [52] than other pooling units such as *average pooling*, and *L2-norm pooling*.

Figure 2.6: This example shows the resulting spatial regions after applying two 2×2 pooling filters with stride 2: a max pooling filter in the top, and an average pooling filter in the bottom.

Fully-connected layers (FC)

FC layers generate a compact representation of the feature maps resulting from the convolutional layers to compute the desired output, for instance, class scores. Unlike conv layers, each neuron of a FC layer is connected to all the neurons in the previous layer without sharing parameters. These layers apply a linear transformation on an input vector of dimension N , and output a vector of dimension N' , where usually $N' < N$. If the previous feature map is not 1-dimensional, the common practice is to flatten it before applying a FC layer.

Non-linear layers

Non-linear layers, also called activation functions, are point-wise operations that introduce non-linearity in the network to be able to learn any mapping function. Thus, they are applied to feature maps resulting of linear units such as convolutional and fully-connected layers. The most popular activation function is the Rectified Linear Unit (ReLU) [50], which computes $f(x) = \max(0, x)$. Regarding other non-linear units such as sigmoid $\sigma(x) = 1/(1 + \exp^{-x})$ or the hyperbolic tangent (tanh), ReLU proved to converge faster using stochastic gradient descent [50], and involves simpler operations.

2.3.1.2 Learning process

The aim of the learning process of any neural network is to find the optimal weights (model parameters) of the network to solve a specific task given a training dataset. As most of learning algorithms, this process consists of minimizing a cost function in the weight space that measures

the error between the predicted output and the actual one. To that end, the backpropagation algorithm [53] computes the partial derivatives of the error with respect to all weights, and uses the Stochastic Gradient Descent (SGD) method [54, 55] to update all the weights in an iterative procedure to progressively minimize the output error. The gradient descent updates weights in proportion to their contribution to the total error according to the rule:

$$w_{i,j} \leftarrow w_{i,j} - \eta \frac{\partial E}{\partial w_{i,j}}, \quad (2.4)$$

where i is a neuron in layer l , j is a neuron in layer $l + 1$, $w_{i,j}$ is the weight of their connection to be adjusted to minimize the error function E , and η is the learning rate. Most relevant optimization techniques are based on the SGD algorithm, however, other optimization techniques have been proposed to adjust the weights during training such as Adam [56], RMSProp [57], and Adadelta [58].

In practice, the learning process is performed in multiple iterations called epochs, in which all the training data is processed, by computing the gradient over a mini-batch of training examples. In general, the weights are randomly initialized with small values and centered around zero. In addition, the cost or loss function must be appropriately chosen to quantify the ability of the network to approximate the actual outputs for all training examples given the task.

2.3.1.3 Regularization techniques

Regularization approaches are used to deal with the phenomenon of overfitting. For a neural network, this means the weights perfectly fit training data, but fail to generalize to new examples. It can be caused by multiple things, but the most common are lack of training data, noisy data, or a network more complex than necessary. To overcome overfitting, several techniques have been proposed. The classical way of dealing with overfitting in machine learning has been by adding a term to the cost function to penalize the high values of the involved parameters. This term is usually based on the L2-norm and is controlled by the regularization hyperparameter. Another common technique is data augmentation, which consists of increasing the size of the training set by applying different transformations to the original images, such as illumination changes, rotations, translations, etc. More specifically for CNN are the dropout layers and the early stopping. Dropout [59, 60] prevents overfitting during training by randomly setting the output of neurons to zero with probability p , which is typically 0.5. This allows the neurons to not co-adapt to each other. On the other hand, early stopping consists of stopping the training before the network starts to overfit the training data.

2.3.2 Recurrent neural networks overview

Recurrent neural networks (RNN) are networks to handle sequential information. Unlike CNNs, a RNN computes the same task for each element of a sequence, so that it shares the same parameters across all the time steps. In addition, its output is dependent on the computation of previous elements. Therefore, they are able to make decisions by experience because of their “memory” ability. Their most popular application is Natural Language Processing (NLP), where for instance,

when predicting the next word in a sentence, it is better to know which words came before it. In computer vision, they are used along with CNN to combine feature extraction on input data and support sequence prediction. Since RNNs require an embedded layer as input, they take advantage of the great semantic representational power of CNNs to generate an embedded vector for the recurrent network. Applications such as activity recognition, image captioning, and video description make use of a CNN-RNN architecture.

A RNN looks like in Figure 2.7. In general, RNNs are good at managing past information from close connections. However, if it is necessary to learn from very past elements in the sequence, their performance considerably decrease due to vanishing and exploding gradients [61, 62]. To overcome the problem with long-term dependencies, a special type of recurrent networks called LSTM network [63] was designed, which is the most used and popular RNN nowadays. Its success lies in the way of computing the hidden state, which allows it to pass information coming from previous elements unchanged. For a better understanding of how LSTM works, the reader is referred to [2]. The algorithm to compute the gradients in a recurrent network is called back-propagation through time (BPTT), which basically consists of applying the same strategy as for CNNs on the unrolled network [64].

Figure 2.7: Compact and unrolled representations of a Recurrent Neural Network with its main parameters: x_t is the input at time step t , A is the hidden state, s_t is the “memory” of the network that is computed from previous hidden states, and h_t is the output (extracted from [2]).

2.3.3 Image classification networks

The success of AlexNet [50] in the ILSVRC 2012 challenge on the ImageNet dataset [51] has inspired the development of more complex architectures, in particular for image classification-related applications. AlexNet revolutionized the computer vision community when it outperformed the second candidate from a top-5 error of 26% to 16%. Conceptually, AlexNet has a similar architecture to LeNet, however it is deeper and bigger, with about 60 million parameters. Unlike LeNet, it used several conv layers stacked on top of each other instead of having a single conv layer immediately followed by a pooling layer. In addition, it introduced ReLU units, which showed much faster training times, and, dropout layers to overcome overfitting. By overlapping the max pooling layers, it also further reduced the classification error. At the time of its development, it was run on two Graphical Processing Units (GPU) due to the computational restrictions.

The winner of both the classification and detection ILSVRC challenges in 2014 was the GoogleNet architecture [65], which introduced the Inception module to reduce the number of parameters of a very deep network with 22 weight layers. It has around 4 million parameters compared to the 60 million of AlexNet. The Inception module consists of 1×1 , 3×3 , and 5×5 filters that are computed in parallel, and then concatenated. 1×1 filters are used before the previous ones as reduction blocks to increase the depth and width of the whole network by only marginally increasing the computational cost. A large number of parameters was also eliminated by adding an average pooling layer on top of the convolutional layers instead of a FC layer directly. As a result of several improvements, the Inception v2 and v3 [66] were obtained, and most recently the Inception v4 [67].

The runner-up of the ILSVRC 2014 classification challenge was VGG [68]. This work proposed a series of simple architectures by only using 3×3 filters, 2×2 pooling functions, ReLU activations, and steadily increasing the depth. It was proven [68] that depth is a critical component for a high performance, and that multiple 3×3 conv layers in sequence have the same effect as larger receptive fields. However, VGG networks are slow to train, and expensive to evaluate due to their memory requirements. Indeed, smaller VGG architectures were trained first and then used as initialization for the deeper ones. The two most popular VGG architectures are VGG16 with 16 layers and 138 M parameters, and VGG19 with 19 layers and 144 M parameters.

Since Residual Networks (ResNet) [69] won the ILSVRC challenge in 2015, they are the state of the art of CNNs, and generally are the default choice to use in practice. This work also proposed several architectures, even deeper than VGG networks (up to 152 layers). However, its contribution was precisely a new learning framework based on special blocks, called *residual blocks*, to address the degradation that very deep models suffer due to the optimization difficulties [69]. A residual block forces its layers to learn a residual function with respect to its preceding input layer. To that end, it uses skip connections defined as 1×1 convolutional shortcuts [70] to allow the input and output of the residual block to be added. This makes it possible to propagate the errors directly to the preceding units, and easier train very deep networks. The whole architecture also lacks FC layers at the end of the network.

As stated before, deep CNNs have dominated image classification tasks, being successful in every ImageNet challenge since 2012. In addition to the ILSVRC challenge, there exist other commonly used for evaluation and benchmarking, which are summarized in Table 2.1. Due to the lack of training data in practice, the previously defined architectures already pre-trained on such huge datasets are used to transfer the learning to smaller datasets and evaluate different visual tasks. There are two common ways to proceed: use the pre-trained model as feature extractor by taking only the convolutional layers, or retrain the classification layers to fine-tune the model on the new dataset.

2.3.4 Object recognition networks

In addition to image classification, CNNs have been also successfully applied to more complex tasks such as object detection. In Fast Region-based Convolutional Neural Network (Fast R-CNN) [71], feature extraction is computed over the whole input image by a CNN, and then the Selective Search algorithm [1] generates region proposals based on the last feature map. Finally, a single

Dataset	Description	N° classes	N° images
MNIST	Handwritten digits	10	70000
CLATECH-101	Color images	101	9146
CALTECH-256	Color images	256	30607
CIFAR-10	Natural color images	10	60000
CIFAR-100	Natural color images	100	60000
ILSVRC	High-resolution color images	1000	> 1.2 million

Table 2.1: Summary of several benchmarks for image classification.

softmax layer is trained to output the class probabilities. However, the adoption of object proposal methods as external modules independent of feature extraction implies a high computational cost that makes it unfeasible for real-time implementations.

Later works offer more promising results by sharing a single CNN computation for region proposal and object classification. For instance, the Faster R-CNN [72] replaced the slow Selective Search algorithm with a fast Region Proposal Network (RPN) that proposes regions for object detection, and introduced a second module that uses the proposed regions for object recognition, which is basically the Fast R-CNN network. The RPN network consists of a 3×3 sliding window that scans the last conv layer of a CNN, and generates multiple possible regions according to k fixed-ratio *anchor boxes* (default bounding boxes). An objectness score and 4 coordinates defining a bounding box are provided for each proposed region. Although the training scheme alternates between fine-tuning for the RPN and for object detection, proposal computation at test time is almost cost-free regarding the computation of the detection network. In addition, Faster R-CNN not only improved in detection speed, but also in recognition performance regarding its predecessor. A similar approach is Region-based Fully Convolutional Net (R-FCN) [73]. It first runs a CNN, for instance ResNet, and then adds a fully-convolutional layer to generate a number of position-sensitive score maps. A score map is computed per relative position in a $k \times k$ grid, and per object class. Considering C object classes plus background, $k^2(C+1)$ position-sensitive score maps are generated. A RPN is applied to the position-sensitive score maps, providing region proposals that are divided into the same $k \times k$ relative positions. Finally, a voting strategy among all the positions is used for each class, resulting in a $C+1$ -dimensional vector representing the region proposal that is classified with a softmax function. R-FCN is faster than R-CNN, and achieves a comparable accuracy.

Unlike all the previous approaches, another wave of object-detection networks proposes simultaneously inferring the bounding box and the class of the object. One of them is Single-Shot Detector (SSD) [74], which also uses a standard architecture (in this case VGG16) as base model, and adds more convolutional layers to progressively decrease the feature maps and allow predictions at multiple scales. Every feature map is discretized into a set of default boxes over different aspect ratios, which are equivalent to the anchor boxes of Faster R-CNN. Then, for each box, bounding box offsets and class probabilities are predicted. Since this approach analyzes every single position in the feature maps, it uses a non-maxima suppression technique. It has proven to be faster than the previous approach since it does everything in a “single shot”. The You Only Look Once (YOLO) approach [3] also follows the “one shot” strategy, but it does not include multi-scale

feature maps, as shown in Figure 2.8. When YOLO was introduced, although it was fast, it did not achieve a comparable accuracy with other detectors. However it has been later improved [75], and has proven to be as accurate as SSD but three times faster, becoming the state-of-the-art detector. YOLO consists of a single CNN that addresses the prediction of bounding box coordinates and class probabilities as a single regression problem. Unlike previous approaches, YOLO makes a decision looking at the whole image, so that it encodes global context about other classes. YOLO applies a CNN to the full image, which is divided into a $S \times S$ grid, and predicts B bounding boxes in the form (x, y, h, w) , an objectness score c , and C class probabilities for each grid cell. Small objects are well localized by the bounding box that provides only one cell, however larger objects are localized by multiple cells, for which non-maximal suppression is used.

Figure 2.8: YOLO network (extracted from [3]). In this case, the final feature map is divided into a 7×7 grid, and for each cell grid 30 values are predicted: 2 bounding boxes including x , y , h , w , and c , and the probabilities corresponding to 20 classes.

The same as for image classification, several challenges have been released for object detection. All the previous approaches have been evaluated in the three most popular benchmarks, which are PASCAL VOC [76], COCO [77] and ILSVRC [51].

2.3.5 Image segmentation networks

Another way of providing the localization of objects is by means of semantic segmentation. To that end, dense pixel-wise annotations are needed, which are provided in datasets such as VOC2012 [78], and COCO [77]. Unlike the approaches defined in the previous section, the idea of segmentation networks is to learn a mapping from pixels to pixels, without extracting any region proposals.

Fully Convolutional Networks (FCN) [79] was the first popular approach for semantic segmentation based on CNNs without any FC layers. FCN consists of first computing compact feature maps resulting of the conv layers, and then applying learnable up-sampling layers, which results in

the classification of every pixel in the image. However, it achieves a coarse segmentation because pooling layers discard spatial information when aggregating the context that is not recovered in the up-sampling layers. To overcome this issue, more sophisticated networks replace the up-sampling layers by a deconvolutional network (DeconvNet) [4], which consists of transposed versions of the convolutional and pooling layers used to infer the semantic representation of the scene, as shown in Figure 2.9. The whole architecture is better known as encoder-decoder network. In SegNet [80] shortcut connections are introduced to transfer pooling indices from encoder to decoder to improve the segmentation resolution. On the other hand, since FC layers are prohibited in a segmentation architecture, Global Convolutional Network (GCN) [81] proposes conv layers with larger kernels as a way to improve the classification task maintaining the spatial information. Encoder-decoder architectures offer a better performance than fully convolutional networks, but their computational cost is much higher.

Figure 2.9: Deconvolutional neural network (extracted from [4]).

A different approach consists of using dilated convolutions [82], which allows the network to aggregate multi-scale context without loss of resolution since no pooling layers are used. The dilated convolution operator is the convolution operator with different dilation factors, so that the receptive field grows exponentially while the number of parameters grows linearly. Other works that use dilated convolutions provide multiple rescaled versions of the input image to the network [83], and combine them by using multiple parallel dilated convolutional layers with different sampling rates. It is also worth mentioning Mask R-CNN [84], which extends Faster R-CNN [72] to predict an object mask in parallel with the existing network for bounding box regression and class categorization. It adds a third branch to outputs the object's mask, which consists of an FCN and a *RoIAlign* layer to preserve the per-pixel spatial correlation.

2.3.6 Object tracking networks

Traditional tracking-by-detection approaches (Section 2.2.3) that rely on hand-crafted features are restricted to work well for particular scenarios, and fail in challenging conditions because they do not capture semantic information. Thus, the first attempt has been using CNNs as classifiers to build a discriminative appearance model, and acquire richer visual representations. To that end, the first frame is used to learn the model, which is then fine-tuned using samples obtained in the subsequent frames. Due to the lack of training data in online tracking, most of the works adopt a transfer learning strategy that uses large-scale datasets [85, 86, 87]. However, online fine-tuning a

pre-trained CNN could result in overfitting [88], and is generally slow.

A second strategy consists of template matching, which is faster than tracking-by-detection because it is not necessary to update the network's parameters [89, 90, 91]. These methods build the matching model from the target in the first or previous frame, and generate the template in the feed-forward pass. More recently, template-based approaches have adopted recurrent neural networks (LSTM) [92] to update the model, since memory states can be used to manage the object template over a long time.

The application of deep neural networks to object tracking is under exploration. Template matching is outperforming tracking-by-detection networks in terms of accuracy and speed, but still an effective technique of adapting the appearance variations of the object online is needed.

2.4 Conclusions

The trends in machine learning for computer vision inevitably go towards DL end-to-end systems due to the considerable gap between traditional machine learning algorithms, and deep neural networks in terms of performance, generalization ability and time saved in designing intermediate steps of the pipeline.

Although machine learning has experienced a huge progress in the last decades, it still presents some challenges in the field of computer vision. Due to the high variability in images, research efforts have focused on solving a computer vision problem by applying the "divide and conquer" concept. This means a task is partitioned into several sub-tasks to progressively separate relevant information from irrelevant one while dealing with other important vision issues, such as different illumination conditions and viewpoints, occlusions, transformations, clutter, blurring, and so on. Although it appears to be a good strategy, it also have some drawbacks: multiple stages can accumulate large errors and latency across the processing pipeline. In addition, algorithms involved in the learning process are not able to fully generalize under different conditions, and they need to be designed accordingly to the task.

From six years ago to now, computer vision community has experienced a change of paradigm in machine learning methodology. A single convolutional neural network is able to solve a classification task in an end-to-end manner by automatically learning features from raw images faster, more efficiently, with a higher generalization ability and with better results than with traditional approaches. This has been possible thanks to advances in GPUs that speed up the learning process, efficient implementations specifically designed to deal with images, implementation sharing, and the large amount of available data. While still pushing for solutions to common tasks, network architectures keep growing more and more complex, which makes it possible to address more challenging computer vision problems.

Chapter 3

Multi-stage traditional learning system for hand-gesture recognition

3.1 Introduction

As stated in Chapter 2, recognition systems generally rely on multiple stages to map an image into the desired output. Before the DL era, this way of proceeding made huge progress in less than a decade thanks to the successful combination of specific descriptors (such as HOG, SIFT, and LBP), and new detection approaches based on machine learning [93, 94]. The pipeline of these systems usually includes detection, feature extraction, and inference stages. The first one proposes region candidates that can be obtained exhaustively via sliding window, or more efficiently by means of a region proposal method. Hand-crafted features are designed to describe specific objects, so they are computed from candidate regions that potentially contain the object of interest to maximize their performance. Then, an inference algorithm such as a classifier, or a regressor, evaluates those image regions to decide if the object of interest is present within them.

The increasing popularity of HCI interfaces based on gestures in the last decades has motivated the development of a hand-gesture recognition system. These interfaces are considered more natural, intuitive, friendly, and less intrusive for the user, so that they present some advantages in comparison with traditional HCI devices in many cases. Their applications go from video games, where user experience is much more rewarding if he/she is using his/her own body, to medical systems, where sterile human-machine interfaces are important to avoid the contamination associated with sealed touch screens. In particular, the proposed system is based on the recognition of a series of hand gestures to simulate a mouse-like pointing device to interact with a computer. The system comprises detection, tracking and recognition stages to robustly locate the gesture in both spatial and time domains, discriminate among different gesture categories, and validate the performed gesture to trigger a mouse functionality.

To carry out the detection and recognition tasks, several image and video descriptors have been designed. When building feature descriptors, several considerations must be taken into account. They should be able to represent the image or video region in a reliable way independent of

the scene conditions. In general, characteristics such as invariance to translations, rotations, scale changes, and dramatic illumination changes are very desirable. Since region-proposal stages usually gather most of the processing time, the descriptors also need to be fast to compute. Another requirement imposed by the model training is that features have a reduced dimensionality. The proposed descriptors have been designed to recognize hand gestures by extending the capabilities of the LBP descriptor. They inherit all its powerful properties, and also include global spatial information and temporal information to recognize dynamic gestures in an efficient way.

The rest of the chapter is organized as follows. Section 3.2 reviews related work on hand-gesture recognition through multi-stage systems and machine learning techniques. Section 3.3 describes the pipeline of the proposed system, which is based on three stages: detection, tracking and recognition. Sections 3.4 and 3.5 present the proposed image and video descriptors, respectively, which are integrated in the previous recognition system to describe and characterize hand gestures. The evaluation of such descriptors and the performance of the whole system are discussed in Section 3.6, and conclusions are drawn in Section 3.7.

3.2 Related work

In spite of the great body of work in hand-gesture recognition, there are still some challenges affecting its performance. Recognizing a hand, and characterizing its shape and motion in images or videos is a complex task. The hand dynamics is highly complex because of the hand's articulated nature with more than 25 degrees of freedom [95]. This fact makes modeling the different poses and motions very difficult. The appearance of a hand can change dramatically because of illumination changes, scaling, blurring, orientations, and occlusions. Moreover, intraclass and interclass variance of the gestures are very high. The same action performed by the same individual several times is slightly different, and this problem gets worse if the same action is performed by two different individuals. In addition, since gestures typically appear within a continuous stream of motion, a temporal segmentation to determine when they start and end is necessary.

State-of-the-art approaches in hand-gesture recognition can be divided into several categories according to the gesture classification into the *form* and *flow* dimensions, as proposed in [46]. The *form* dimension distinguishes two categories depending on the nature of the gesture: *path* and *pose*. The former includes those hand gestures characterized by different paths without any particular hand poses (e.g. a circle). The latter includes those gestures characterized by different hand poses without any specific paths (e.g. a pointer-like hand pose that moves around to point something on a display). The *flow* dimension can be in turn divided into two categories according to the system design: *discrete* and *continuous*. A discrete gesture is delimited in time so that the system responds after the gesture is recognized (e.g. swipe left or swipe right), whereas a continuous gesture needs ongoing recognition so that the system can respond frame by frame (e.g. continuous scrolling). Figure 3.1 summarizes this classification.

In general, the most popular approaches for *pose*-gesture recognition, either discrete or continuous, are based on feature extraction to describe the hand appearance, along with some template matching or machine learning technique to perform the recognition task. For instance, the work in [96] describes the hand pose by its contour shape, and performs the gesture classification by

Figure 3.1: Gesture taxonomy along *form* and *flow* dimensions.

using template matching through a shape distance metric called Finger-Earth Mover's Distance (FEMD). The work in [97] presents a framework based on a 2D volumetric shape descriptor that is delivered to a SVM for hand posture classification using depth imagery. Regarding *path*-gesture recognition, motion features play an important role. For instance, spatio-temporal interest points (STIP) [98] are detected and described by means of HOG and HOF features (Histograms of Optical Flow) to recognized temporal-segmented *path*-gestures in [99]. On the other hand, authors in [46] recognize both *pose* and *path* gestures using the Kinect sensor. In particular, they track the hand by using the hand joint position reported by the Kinect SDK, and extract motion and HOG features from the hand region. Then, a hierarchical HMM is trained to recognize both types of gestures.

3.3 Hand-gesture recognition system

3.3.1 System overview

A more natural and intuitive HCI interface based on a robust hand-gesture recognition system has been developed to replace the traditional interface provided by the mouse device. The system is composed of three stages: hand-pose detection frame by frame, hand-pose tracking, and recognition of the hand gesture. The detection phase employs a spatial descriptor and a binary classifier to detect potential hand poses in every frame. An exhaustive search through the image following a multi-scale and sliding-window strategy is used as region-proposal method for detection. In this stage of development, where computational cost is not the primary criteria, an exhaustive search

has been chosen because of its simplicity. For future real-time operation, this method would have to be revised. The tracking phase uses those detections as input for a multiple object tracker that estimates potential trajectories of the hand along time. These trajectories contain a volume of regions of interest (ROI) that represents different dynamics of the hand gesture. Finally, the recognition phase analyzes trajectories to temporally localize and recognize the type of hand gesture by using a highly-efficient video descriptor along with a set of classifiers. Machine learning techniques have been used along with feature extraction to carry out the detection and recognition tasks. In particular, Support Vector Machine (SVM) have been used as classifiers, since they work very well with high-dimensional data, and are capable of delivering high performance in terms of classification accuracy. The aforementioned stages are explained in detail in the following sections.

The proposed system has been used as a Human-Computer Interaction (HCI) interface that simulates a mouse-like pointing device to interact with a computer by means of dynamic hand gestures. For this purpose, a new dataset has been created, containing hand gestures based on mouse functionalities (see Section 4.5.3.1).

3.3.2 Detection

The aim of the detection stage is to localize those hand poses that can be part of the considered dynamic hand gestures in every frame. These specific hand poses (positive class) must be recognized among other irrelevant hand poses and background (negative class). An overview of this stage can be seen in Figure 3.2.

Figure 3.2: Block diagram of the detection stage.

The first stage consists in providing region proposals to detect hand poses in different spatial locations and scales. To that end, every frame is scanned by a spatial sliding window at multiple scales, as shown Figure 3.3. The multi-scale analysis is carried out by generating a multi-resolution pyramid, which contains different scales of the frame that is being processed. Then, an overlapped and fixed rectangular window is slid along each scale. The system allows to set the number of scales, and the minimum and maximum scales, which are design parameters for this step. This strategy has been chosen instead of sliding several windows with different sizes applied to only one scale, because the computational cost is lower and the detector is easier to train. The

overlapping magnitude between consecutive windows is determined by the spatial step of the sliding window. The choice of this parameter is a trade-off between the computational cost and the accuracy of the detection. For instance, if the windows tightly surround the object of interest, the classifier might be trained more accurately. However, the spatial step should be smaller, having to process more windows. Instead, using windows that are rather larger than the object along with a bigger spatial step, the number of windows to process would be lower, but the ability of the classifier to precisely detect and localize objects will decrease.

The following step applies an image descriptor to every spatial window to compute a feature vector that represents the image region for detection. Then, every feature vector feeds the classifier, which determines if that image region is a positive or a negative sample. Since this stage has to deal with two classes, a binary SVM classifier is used for detection. Image regions containing hand poses belonging to the dynamic hand gestures are used as positive samples. On the other hand, image regions containing background and other irrelevant hand poses are used as negative samples. Spatial windows that tightly surround the object are used to train the classifier, this way a small spatial step for sliding must be considered. To increase the performance of the SVM facing non-separable data, the kernel trick has been adopted, which allows SVM to remain linear achieving fast learning. For this purpose, a Hellinger kernel, also known as Bhattacharyya coefficient [100], is used to project features into a higher-dimensional space. To alleviate the high computational cost of such kernel, and efficiently compute the new features, it is implemented using the square root of the feature vectors [100] as follows:

$$k(f, f') \approx \sum_i \sqrt{f(i)f'(i)}, \quad (3.1)$$

where f and f' are two normalized feature vectors.

The last stage consist of non-maxima suppression. Windows overlapping makes close image regions be classified as positive because they contain a significant fraction of a hand. This causes the same hand pose to be counted multiple times. Therefore, the potential regions that have been classified as positive need to be filtered. In order to select the one that better represents the hand pose, a non-maxima suppression (NMS) technique is applied in every scale. It consists in selecting those those samples that present a local maximum score, according to two design parameters: a threshold and a radius [101]. However, the problem persists among different scales of the multiresolution pyramid. For this reason, another non-maxima suppression technique based on an overlapping criterion is applied to normalized windows from different scales: if the overlapping of two normalized windows (i.e. normalized to a common scale) is bigger than a specific threshold, the window that presents the highest classification score is labeled as a positive sample, and the other one as a negative sample.

As a result, a set of filtered detections is obtained for every frame and used as input of the tracking stage. In particular, each detection is expressed as a (x, y, w, h, s) vector, where the first four components define its bounding box, and the last one s is the classification score. (x, y) are the coordinates of the upper-left corner of the bounding box, and w and h its width and height respectively. The spatial features (i.e., the computed spatial image descriptors) extracted from each filtered detection are stored to be used later in the recognition stage.

Figure 3.3: Multiresolution pyramid and sliding window strategy.

3.3.3 Tracking

The goal of the tracking phase is to estimate temporal hand trajectories from previous detections. When the tracker receives the first set of detections, it creates as many trajectories as the number of detections in the frame. Detected hand poses in the next frames are associated to their corresponding trajectories according to their locations in previous frames.

Since there can be missing detections due to occlusions or strong changes in hand appearance, and also false detections generated by background structures, the estimation of the trajectories can be inaccurate. Erroneous associations between detections and trajectories can also happen. On the other hand, the computational cost of the tracking could inevitably grow exponentially with the number of objects. To deal with these problems, a multiple object tracker which is robust to erroneous, distorted, and missing detections [102] has been used. This tracker is based on a constant velocity model for predicting the object locations, and a soft probabilistic data association that recursively estimates the best correspondence between measurements/detections and existing objects/trajectories. In addition to improving the accuracy, the computational cost is reduced since the complexity with the number of objects becomes linear, instead of exponential.

Since the global system has been designed to recognize gestures performed with one hand, only one trajectory should be created. However, more than one trajectory can appear due to false detections. Figure 3.4 shows how these false detections should not persist along time because they produce very short trajectories. The tracker allows removing them by imposing a threshold for the minimum trajectory length. However, if this problem persists, it will be also handled by the recognition phase.

The resulting trajectories can be seen as volumes of image regions containing hand poses. They are represented by their corresponding bounding boxes, which are defined by (x, y, w, h) vectors.

3.3.4 Recognition

The recognition stage extracts spatio-temporal volumes from previous trajectories to localize a gesture in time, and then analyzes every volume to recognize the performed gesture. Overlapping volumes are compared to validate the gesture and get rid of useless transitions between gestures.

Figure 3.4: Tracking of detected hand poses. Blue bounding boxes correspond to detections that propagates along frames, so that the tracker is able to create a trajectory of such image regions. The tracker is robust against missing detections (yellow bounding boxes), and also against false detections (red bounding boxes). The latter do not generate a trajectory because of their short length.

Once a gesture is placed in a specific instant and classified as a particular action, the application can trigger the associated mouse functionality. The overview of the recognition stage can be seen in Figure 3.5.

Figure 3.5: Block diagram of recognition stage.

The first step temporally segments a trajectory to extract spatio-temporal volumes. To that end, a temporal window is defined to determine where a dynamic gesture starts and ends. It can be seen as an L -frame FIFO buffer with 1-frame enqueue and dequeue, as shown in Figure 3.6. The size of the temporal window L is fixed according to the average duration of the different dynamic hand gestures. The number of frames to enqueue and dequeue is a trade-off between computational cost and recognition. The smaller the enqueue and dequeue size, the larger the number of temporal windows to process. Although the recognition accuracy and the temporal resolution to place a gesture improve. Once the trajectory's length reaches the size of the buffer, the temporal window can be applied.

In the next stage, a video descriptor extracts a spatio-temporal feature vector from the segmented volume to carry out the recognition task. In particular, video descriptors that take advantage of the spatial features that have been already computed and stored in the detection stage are proposed (see Section 3.5). The resulting spatio-temporal feature vector feeds a set of C SVM classifiers to carry out the recognition task, where C is the number of classes. The way of training the set of C classifiers is following a *one-vs-all* strategy, where each classifier is trained by considering as positive samples all the sequences from the class that is being trained, and as negative samples sequences belonging to the rest of classes. All the classifiers use the same kernel as in

the detection phase (see Equation 3.1). There are two main advantages in using a discriminative classifier together with video feature extraction techniques in the recognition phase, in comparison to generative models. First, its computational complexity is lower since it is not so dependent on the number of gestures to be recognized. In addition, if a new gesture needs to be recognized, it is only necessary to update the training dataset and model, instead of having to modify the whole gesture model.

The final step validates the predicted gesture along consecutive temporal windows to make a decision about the performed gesture. Because of the small size of the enqueue and dequeue, consecutive windows differ only in a few frames, and therefore they should predict the same output class. By imposing this condition, a validation window is defined as a K -window FIFO buffer with K -window enqueue and dequeue. Therefore, the validation window analyzes the output classes of K consecutive windows. If they agree about the predicted gesture, the system triggers the associated mouse functionality at the end of the $K - th$ temporal window. Otherwise, the system does not perform any action. In any case, a new validation window is initialized at the beginning of the $(K + 1) - th$ temporal window to validate the next three ones. This strategy makes the system react only after an agreement upon a gesture is reached, solving potential errors due to gesture transitions and erroneous estimated trajectories.

Figure 3.6: Temporal window approach to segment the trajectories of cropped images in time. It works as a FIFO buffer with 1-frame enqueue and dequeue to increase the temporal resolution when localizing a gesture. A feature vector is extracted from each temporal window to feed a set of SVM classifiers and recognize the gesture within it.

3.4 Design of spatial descriptors

One of the main contributions of this thesis is the design of advanced image and video descriptors to achieve reliable and compact representations from images and video sequences. They are inspired by Local Binary Patterns (LBP) [6], since they present very important and powerful characteristics.

The LBP descriptor has been successful in several applications, such as texture recognition [103], face detection/recognition [104], and facial expression recognition [105]. However, the fact that it discards the spatial order of local patterns, severely limits its descriptive power of the image representation. This lack of global spatial information could be a disadvantage in some cases.

Textures can be seen as a set of patterns that recur several times. As for faces, they are formed by a uniform surface (forehead skin, cheeks, and chin), and by four patterns that do not greatly change their relative positions (two eyes, a nose, and a mouth). In all these cases, the global spatial information is not very determining since the appearance can only change in a limited and controlled way. However, a hand, for instance, is a deformable object with more than 25 degrees of freedom [95], and its appearance can change largely. Neither the hand patterns spread out uniformly as in textures, nor are they located in specific areas of the image, as in faces. For hand poses, knowing from what part of the image the patterns come is as important as knowing the type of patterns, and the number of times they appear.

Two new image descriptors have been implemented to deal with hand patterns in the spatial domain, spatiograms and Local Binary sub-Patterns (LBsP), which are spatial extensions of the LBP descriptor. The former includes global spatial information in a controlled way to be more discriminative. The latter is a further improvement that produces a more compact feature representation.

3.4.1 Spatiogram of Local Binary Patterns (S-LBP)

A spatiogram is a representation of the spatial distribution of one specific LBP pattern in the image. The idea behind this concept is to know what part of the image every LBP pattern comes from. The specific procedure is as follows.

First, the multi-scale LBP descriptor ($LBP_{P,R}$) [6] is computed as defined in Appendix A. It thresholds a circular neighborhood of each pixel, which is defined by its radius R and the number of neighbors P , by the intensity value of the center pixel in order to extract local spatial structures from an image region. The thresholded values are concatenated in a P -bit binary number, and converted to decimal for a more compact representation. Finally, they are used to generate a histogram of 2^P labels that represents the image region. Figure 3.7 shows the LBP image and the histogram of LBPs (H-LBP) as a result of the $LBP_{P,R}$ computation. The former is used to build the spatiograms, and thus extract global spatial information, while the latter contains local spatial information. The S-LBP inherits the property of invariance against monotonic illumination changes due to the use of LBP structures in its computation.

Figure 3.7: LBP computation: (a) original image, (b) LBP image, and (c) histogram of LBPs (H-LBP).

Next, an orthogonal sub-lattice of M rows and N columns (red dots in Figure 3.8(a)) is considered to encode the global spatial information of every LBP. These $M \times N$ lattice points are the labels of the spatiograms, identified as 1, 2, ..., $M \times N$ from left to right and top to bottom (Figure 3.8(b)). They serve as gathering knots to represent the spatial distribution of the LBP patterns in a compact way. The choice of M and N is a trade-off between memory requirements and the discrimination ability. To know how an LBP pattern type is spread out along the image, the distances between all the LBP patterns of that type and the lattice points are computed. However, to keep a reasonable compromise between rich description and efficiency, only the four closest lattice points to every LBP are considered when generating the final histogram. For this purpose, a bilinear interpolation is computed as follows. Let (x, y) be the coordinates of an LBP pattern in a coordinate system where the coordinates of the four lattice points are $(0, 0)$, $(0, 1)$, $(1, 0)$, and $(1, 1)$ for the upper-left, upper-right, bottom-left, and bottom-right points, respectively. Then, the weights of the bilinear interpolation are $(1 - x)(1 - y)$, $x(1 - y)$, $(1 - x)y$, and xy for the previous points, respectively. The bilinear interpolation approach increases the robustness against slight image translations, and the grid effect. Figure 3.8 shows a geometric visualization of the bilinear interpolation (b), and how a specific LBP pattern contributes to its corresponding spatiogram (c). As a result, a histogram of 2^P labels (H-LBP) and 2^P spatiograms (one per LBP type) of $M \times N$ labels each one are obtained, where P is the number of bits in a local binary pattern.

Figure 3.8: (a) Orthogonal sublattice of $M=8$ rows and $N=8$ columns (red points) overlaid on the LBP image to select the key image coordinates. (b) Geometric visualisation of the weights of the bilinear interpolation. The spatial distribution of the LBP pattern at the red spot is measured with respect to its four closest lattice points. (c) Resulting spatial distribution for the LBP pattern type at the red spot. LBP patterns of the same type will contribute to the same spatiogram.

Finally, the H-LBP and all the spatiograms are concatenated as shown in Figure 3.9, obtaining a more discriminative descriptor since the local spatial information from H-LBP is complemented with global spatial information from the spatiograms. The resulting descriptor is called Spatiograms of Local Binary Patterns (S-LBP), whose notation is $S-LBP_{P,R,M,N}$, and its dimensionality is $2^P(1 + M \times N)$.

Figure 3.9: Spatiograms of Local Binary Patterns (S-LBP)

3.4.2 Local Binary sub-Patterns (LBsP)

The S-LBP descriptor defined in the previous section, could have a excessive high dimensionality depending on the choice of its parameters P , M , and N . In particular, small values of M and N are desirable to keep short the length of the spatiograms and summarize the global spatial information in a compact way, although it implies a lower spatial resolution information that can impact in the discriminative power of the descriptor. Regarding P , not only the dimension of the H-LBP grows exponentially with the number of neighbors, but also the number of spatiograms. Therefore, the dimensionality of the S-LBP descriptor could be impractical for some configurations.

Figure 3.10: LBsP computation. The LBP type 00111111 is divided into the LBsP types 0011 and 1111, each one contributing to a different histogram. The two resulting histograms are concatenated to form the H-LBsP histogram.

To solve this issue, a new extension of the LBP descriptor [6] is proposed. It is called Local Binary sub-Patterns (LBsP), and its main objective is to reduce the number of pattern types while preserving the same local spatial information from each image region. The LBsP descriptor follows the same strategy as LBP. It thresholds the neighborhood of each pixel and concatenates them to form a P -bit binary number. However in this case, the P -binary number is divided into n local binary sub-patterns of P/n bits, which are defined as:

$$LBsP_{P,R,n}^i \doteq \sum_{p=\frac{P}{n}(i-1)}^{\binom{P}{n}i-1} s(g_p - g_c) 2^{(p \bmod P/n)}, \quad (3.2)$$

where $i = 1, \dots, n$, and $a \bmod b$ is the modulus operator or remainder after division between a and b .

LBP extracts only one pattern per pixel, and generates only one histogram. In the case of LBSP, it extracts n sub-patterns per pixel, and therefore it generates n histograms for the given image region. Then, every sub-pattern in the pixel contributes to a different histogram with $2^{P/n}$ bins each. To produce an only descriptor for the given image region, the n histograms are concatenated to form the Histogram of LBSP (H-LBSP). Figure 3.10 shows the computation of the H-LBSP descriptor for a 3×3 neighborhood, where $P = 8$ neighbors, and $n = 2$ local binary sub-patterns.

While the LBP descriptor extracts 2^P different patterns per image region, LBSP extracts $2^{P/n}$. Therefore, this approach reduces the dimensionality of the final histogram from 2^P bins down to $n2^{P/n}$ bins. This is particularly interesting when using neighborhoods with a large number of sampling points, since the number of pattern types grows exponentially with P , as shown in Figure 3.11. Table 3.1 summarizes the main differences between LBP and LBSP.

	LBP _{P,R}	LBSP _{P,R,n}
Patterns per pixel	1	n
Pattern types per image	2^P	$2^{P/n}$
Dimension	2^P	$n2^{P/n}$

Table 3.1: Comparison between LBP and LBSP in terms of number of pattern types per pixel and per image, and descriptor dimension.

Figure 3.11: Comparison of the exponential growth of the number of patterns per image with P for the LBP and LBSP descriptors. For the LBSP descriptor $n = 2$ sub-patterns are considered. The left part of the figure is zoomed in to better show the difference between both descriptors. For instance, whereas LBP produces 256 pattern types for $P = 8$ neighbors, LBSP just produces 16.

The spatiograms of every LBP pattern are computed in the same way as for LBP patterns. In this case, a histogram of $n2^{P/n}$ labels (H-LBSP) and $2^{P/n}$ spatiograms of $M \times N$ labels are obtained. By concatenating them as in Figure 3.9, the Spatiograms of Local Binary sub-Patterns (S-LBSP) descriptor is obtained, whose notation is $S\text{-LBSP}_{P,R,n,M,N}$. The dimensionality of this descriptor is $2^{P/n}(n + M \times N)$, also lower than in the original S-LBP descriptor, which is $2^P(1 + M \times N)$.

3.5 Design of temporal descriptors

Regarding LBP extensions to include temporal information, there are two main ones, VLBP and LBP-TOP [29], which are further described in Appendix A. They essentially present the same problem as LBP: the lack of localized patterns, which turns into a more complex problem when working in the time domain. Additionally, the VLBP leads to too high dimensional features, and the feature extraction process for both VLBP and LBP-TOP is very slow. Therefore, adding global spatial information to these descriptors to obtain more discriminative features could be prohibitive in terms of computational cost and memory requirements.

To recognize spatio-temporal hand patterns, two new approaches are proposed: volumetric spatiograms and Temporal Pyramid Matching (TPM). They take advantage of previously computed spatial features to reliably include spatial information, and temporal information in a fast and compact way. The former adds temporal information from a series of key frames to remove redundant information. The latter uses a more sophisticated approach to densely pack all the temporal instants. As a result, these descriptors make it possible to highly discriminate among different spatio-temporal pattern types with different lengths. Both of them will be explained below.

3.5.1 Volumetric Spatiograms of Local Binary Patterns and sub-Patterns

Previous spatial or image descriptors can be extended to spatio-temporal applications (i.e. from image to video-based) by concatenating the spatial descriptors extracted from a set of key frames to form a video descriptor. The reasoning is as follows. In general, a video sequence is composed by a number of ordered frames which describe a continuous movement in time. Frames close in time hardly change their appearance, containing redundant information. This fact could make unnecessary analyzing all the frames to identify the action that is being performed in such sequence. Hence, taking a suitable number of sampled frames, extracting discriminative spatial features from them, and concatenating all together, can represent a spatio-temporal pattern appropriately.

To build this video descriptor, a combination of an uniform sampling plus a random offset is applied to a video sequence as shown in Figure 3.12. The purpose of this random offset is to obtain a different set of sampled frames every time the descriptor is applied. Firstly, two parameters need to be introduced: Δ_e and δ_{max} . Δ_e is the sampling interval, which is computed as follows:

$$\Delta_e = \frac{L}{S}, \quad (3.3)$$

where L is the length of the sequence, and S is the number of sampled frames. The additive

random shifting Δ_{shift} follows a discrete uniform distribution in the interval $[-\delta_{max}, \delta_{max}]$. Taking very close consecutive samples must be avoided, so that δ_{max} is taken as 25% of the sampling interval Δ_e :

$$\delta_{max} = \lfloor \frac{\Delta_e}{4} \rfloor. \quad (3.4)$$

Once these parameters are computed, the next step is to calculate the index of the first sampled frame s_0 . This variable follows a discrete uniform distribution in the interval $[0, s_{0,max}]$, where $s_{0,max}$ is defined as the maximum index of the first sampled frame to ensure taking S samples from a sequence of length L :

$$s_{0,max} = L - 1 - \delta_{max} - \lfloor \Delta_e \rfloor (S - 1). \quad (3.5)$$

where $\lfloor x \rfloor$ is the largest integer not greater than x . Finally, the indices of the remaining $S - 1$ sampled frames are computed as an initial offset determined by the index of the first sample s_0 , plus i times the temporal interval Δ_e , plus a random shifting defined by Δ_{shift} :

$$s_i = s_0 + \lfloor i\Delta_e \rfloor + \Delta_{shift}, \quad i = 0, \dots, S - 1, \quad (3.6)$$

where s_i is the index of the i -th sample.

Once the key frames are obtained, the following step consists of extracting spatial features from them using any of the S-LBP or S-LBSP descriptors. Finally, the resulting video descriptor is built by concatenating all the spatial descriptors in sequential order. It is fast to compute since key frames are known *a priori*. The random sampling algorithm is run before processing video sequences, so that spatial features are only extracted from key frames, discarding remaining ones.

The random nature of this temporal sampling allows to obtain a different set of key frames every time it is applied. This simulates different spatio-temporal patterns with slight biases in the execution. Moreover, it can be very useful in order to generate several training feature vectors from the same video sequence, since every time it is applied, the obtained video descriptor is different. This is also a versatile approach since it can be used along with any image descriptor, keeping in mind the overall length of the final video descriptor. Its dimensionality depends on the length of the spatial features, and the number of sampled frames, so that S must be suitable for selecting enough key frames that correctly represent the spatio-temporal pattern, while avoiding the creation of a very long video descriptor. In addition, it can be applied to video sequences with different lengths.

When temporal sampling is used along with the spatial descriptors defined in previous sections, the obtained video descriptors are called Volumetric Spatiograms of Local Binary Patterns (VS-LBP_{P,R,M,N,S}) and Volumetric Spatiograms of Local Binary sub-Patterns (VS-LBSP_{P,R,n,M,N,S}), respectively. In comparison with other spatio-temporal variants of the LBP descriptor, such as VLBP and LBP-TOP, the temporal sampling only cuts down dimensionality with respect to VLBP, however its computation is much faster in any case. In addition, VS-LBP and VS-LBSP descriptors include global spatial information, so their discriminative power is higher.

Figure 3.12: Temporal sampling procedure with $S = 4$ samples, applied to a video sequence of length $L = 16$. (a) Δ_e , δ_{max} and s_0 are computed. Then, a uniform sampling is applied to select the green dotted frames. Green frames are the ones that belong to the random shifting interval. (b) Random shifting is applied to the green dotted frames to obtain the remaining s_i samples (red frames), from which spatial features will be extracted.

3.5.2 Temporal Pyramid Matching (TPM)

A more advanced approach to deal with spatio-temporal signals consists of computing some statistics of the spatial descriptors extracted from a series of partitions of the video sequence. This idea has been taken from Spatial Pyramid Matching (SPM) [106], which has been adapted to time domain and called Temporal Pyramid Matching (TPM).

SPM is an extension of the BoW strategy to model the spatial layout of local features extracted from an image. SPM first partitions an image into $2^l \times 2^l$ regions in different scales $l = 0, 1, 2$. Then, it computes a BoF histogram within each region, and concatenates all the histograms to build a feature vector that represents the image.

In the case of TPM, a video sequence is represented as a time-ordered collection of spatial descriptors extracted from each frame. The TPM method builds a multi-resolution temporal pyramid, where the video sequence is divided into 2^l partitions in different levels $l = 0, 1, 2, \dots, L$, as shown Figure 3.13. A feature vector is computed within each partition by applying the average pooling function to its corresponding set of spatial descriptors. Finally, all feature vectors z_{lj} from all partitions and pyramid levels are concatenated to form a multi-temporal representation of the video sequence. The average pooling function is defined as:

$$z_{lj} = \frac{1}{K_{lj}} \sum_{k \in X_{lj}} x_k, \quad (3.7)$$

where K_{lj} is the number of frames within the j -th partition in pyramid level l , X_{lj} is the set of indices of the frames belonging to the j -th partition in pyramid level l , and x_k is the spatial descriptor extracted from the k -th frame.

Figure 3.13: Temporal Pyramid Matching. For each level l , the video sequence is partitioned into 2^l sub-sequences. S-LBP descriptors extracted from each sub-sequence are averaged by means of the average pooling function. The resulting z_{lj} histograms are concatenated to form the final feature vector representing the video sequence.

Notice that TPM does not discard any frame when extracting temporal features, unlike the previous approach. Even more, whereas a suitable number of frames S must be considered in VS-LBP and VS-LBsP to compute the video descriptor and keep its dimensionality low, TPM offers a scalable solution by using all the frames. Due to the spatial descriptors being averaged, histograms z_{lj} keep the same dimensionality, so that the length of the video descriptor is $\sum_{l=0}^L 2^l$ times the length of the spatial descriptors. This results in a higher discriminative power since not only the most representative states are considered, but also the whole evolution of the spatio-temporal patterns. However, it is slower to compute since spatial features need to be extracted from all the frames. Nonetheless, TPM encodes the temporal information in a very compact fashion, and also makes it possible to describe spatio-temporal patterns of different temporal length.

3.6 Experimental results

The proposed image and video descriptors have been evaluated for hand-gesture recognition, as well as the performance of the complete system. For this purpose, a new dataset has been created, which contains specific hand gestures to control the different mouse functionalities.

3.6.1 Dataset

A new dataset has been created to validate the hand-gesture recognition system and imitate a mouse-like pointing device, called Hand-gesture Database (Set 2) and is publicly available on http://www.gti.ssr.upm.es/data/HandGesture_database.html. It contains a set of hand gestures that represents the main functions of a mouse device: cursor, left click, right click; and two additional functions from the viewpoint of the application, such as mouse activation, and mouse deactivation. As a result, five different hand gestures are considered to simulate a mouse device and interact with a computer, which can be seen in Figure 3.14. They are classified as *pose* gestures since they are characterized by a specific pose without any particular path. Notice that all of them are *discrete*, except for the cursor.

The dataset contains 36 video sequences divided into 6 different individuals. For each individual, 5 short video sequences have been recorded containing several repetitions of each gesture, and 1 long video sequence, where the individual is recorded using all the hand gestures as if he/she was using the application. In total, the dataset includes 30 short video sequences, which are used for training, and 6 long video sequences for testing.

Training sequences contain more than 20500 frames and are used to train both the detection and recognition phases. The former takes image regions containing hand poses from the considered gestures as positive samples, and other irrelevant hand poses and background regions as negative samples. The latter takes segmented spatio-temporal volumes containing the realization of one gesture. The spatial and temporal annotations of hand poses and gestures have been done manually. Testing video sequences are used to evaluate the whole hand-gesture recognition system.

All the video sequences are recorded in a realistic scene, which means a standard environment with a non-uniform background, and other moving objects. The typical scene structure is composed of an individual interacting with the computer at a distance of approximately 0.5 meters. The sensor is located on top of the screen.

(a) Mouse activation: O.K. symbol.

(b) Cursor: pointing-like hand pose moving around.

(c) Left click: from the cursor pose, extend and close all fingers.

(d) Right click: from the cursor pose, extend and close the forefinger and the middle finger, at the same time.

(e) Mouse deactivation: fist.

Figure 3.14: Proposed hand gestures (visual examples).

3.6.2 Image descriptor evaluation for hand-pose detection

In this section, the image descriptors presented in Section 3.4 are evaluated on several datasets and compared to other state-of-the-art methods for static hand-gesture recognition using precision, recall, F1-score, and accuracy metrics. Precision and recall are the most popular metrics to quantify the results of a binary classification problem, which can be summarized into four values:

- True positives (TP): number of correct detections.
- False negatives (FN): number of missing detections.
- False positives (FP): number of false detections (false alarms).

- True negatives (TN): number of correct no-detections.

Then, precision (3.8) is defined as the proportion of positive detections that are actually correct, and recall (3.9) is defined as the proportion of actual positives that are correct detections:

$$\text{Precision} \doteq \frac{TP}{TP + FP}, \quad (3.8)$$

$$\text{Recall} \doteq \frac{TP}{TP + FN}. \quad (3.9)$$

The F1-score quantifies the quality of a detector by providing a single value that summarizes the obtained precision and recall:

$$\text{F1-score} \doteq 2 \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3.10)$$

Another related metric used in classification is the accuracy, which is defined as the proportion of true results (both TP and TN) with respect to the total number of cases evaluated.

$$\text{Accuracy} \doteq \frac{TP + TN}{TP + TN + FP + FN}, \quad (3.11)$$

First, LBP, S-LBP and S-LBsP are evaluated on the proposed dataset in terms of precision, recall, F1-score, and dimensionality. Data used to train the detection stage is extracted from the 30 training video sequences. It consists of image regions containing background and hand poses that are divided into two subsets: a training set with 80% of the images, and a validation set with the remaining 20%. Tables 3.2, 3.3, and 3.4 show the parameter configurations for which the best results have been obtained.

The three descriptors have two parameters in common, which are the number of neighbors P , and the radius of the neighborhood R . Since previous works [6] have proven that the best results for the LBP_{PR} descriptor are achieved with $R = 1$ and $P = 8$, and the multi-scale scheme will be carried out by a multi-resolution Gaussian pyramid, these values are fixed to 1 and 8, respectively. For this neighborhood, an appropriate value for the number of local binary sub-patterns is $n = 2$ in the case of S-LBsP. Regarding the parameters of the spatiograms, values from 2 to 10 have been tried for the number of samples per rows and columns, which are M and N respectively, to test the influence of the addition of global spatial information. In particular for the LBP descriptor, the number of blocks in which to divide the image region is also evaluated, which is a common technique to provide some global spatial information: an LBP descriptor is extracted from each block, and then all of them are concatenated.

The regularization parameter for the SVM classifier, defined as λ , has been also evaluated. Large values of λ can cause overfitting, while smaller values can produce a SVM model that is not able to separate the classes. Therefore, a very wide range of values has been tested, which goes from 0.01 to 100 taking steps of 0.3, in order to find the value that achieves the best performance.

In Table 3.2 it can be observed that as the number of blocks in which the image is divided increases, so does the F1-score. This also happens for S-LBP when increasing M and N , as shown

Descriptor parameters		Metrics			Dimension
P,R	Num. blocks	Precision	Recall	F1-score	Length
8,1	1	0.975	0.982	0.9785	256
8,1	4 (2×2)	0.990	0.995	0.9925	1024
8,1	9 (3×3)	0.998	0.994	0.9960	2304
8,1	16 (4×4)	0.995	0.998	0.9965	4096
8,1	25 (5×5)	0.998	0.995	0.9965	6400
8,1	36 (6×6)	0.998	0.998	0.9980	9216
8,1	49 (7×7)	0.998	1	0.9990	12544

Table 3.2: Precision, Recall, and F1-score metrics for the best parameter configurations of the LBP descriptor.

Descriptor parameters		Metrics			Dimension
P,R	M,N	Precision	Recall	F1-score	Length
8,1	2,2	0.987	0.995	0.991	1280
8,1	4,4	0.998	0.998	0.998	4352
8,1	4,10	1	1	1	10496
8,1	5,9	0.998	1	0.999	11776
8,1	8,8	0.999	1	0.9995	16640

Table 3.3: Precision, Recall, and F1-score metrics for different parameter configurations of the S-LBP descriptor.

Descriptor parameters		Metrics			Dimension
P,R,n	M,N	Precision	Recall	F1-score	Length
8,1,2	4,10	0.994	0.997	0.9955	672
8,1,2	5,9	0.994	0.998	0.996	752
8,1,2	8,8	0.995	0.998	0.9965	1056
8,1,2	10,8	1	1	1	1312

Table 3.4: Precision, Recall, and F1-score metrics for different parameter configurations of the S-LBSP descriptor.

in Table 3.3. Thanks to the addition of global spatial information, either because of block division or spatiograms, the discriminative power of both descriptors increases. Although F1-score results for LBP and S-LBP are very similar, the ones obtained for S-LBP are always slightly higher. S-LBP encodes better the global spatial information than LBP, allowing to maintain the “map” from which the LBP patterns come. Whereas the block-division strategy applied to LBP only counts how many local binary patterns there are in each image block, spatiograms additionally compute the spatial distribution of every LBP type.

Therefore, the more global spatial information, the better the detection rate. However, this also means a higher dimensionality. In particular, considering the parameter configurations for which LBP and S-LBP descriptors achieve the best results, their dimensions are around 10000, which could be prohibitive in terms of memory requirements. This would be even worse if they are used along any of the proposed temporal approaches described in Section 3.5.

As shown in Table 3.4, local binary sub-patterns are able to overcome this dimensionality problem. S-LBsP seems to be slightly less discriminative than S-LBP since it needs higher values of M and N to achieve the same detection rates as S-LBP. Although the same information is extracted from the local neighborhood of a pixel, it is encoded using 4 bits instead of 8. As a result, the dimension of S-LBsP for every parameter configuration is much lower than with S-LBP, despite the raise of M and N . In particular, S-LBsP’s dimension decreases around 87% with respect to S-LBP when considering the best case for both descriptors.

The three descriptors are also compared with other methods in the literature, which have been introduced in Section 3.2, on the NTU dataset [96]. This dataset, which provides both intensity and depth images, is collected from 10 subjects, and contains 10 poses. Each subject performs each pose 10 times, so that it has in total 1000 cases. For each pose, the subject changes the orientation, scale, and articulation. Table 3.5 shows the results on this dataset in terms of accuracy. This metric has been computed using the following training strategies for comparison with the rest of methods: splitting the dataset into 50%-50% for training and testing respectively (h-h), and following a leave-one-out (l-o-o) strategy. The former uses the half of the data for training, and the other half for testing, including all the users and all the gestures in both sets. The latter consists of training the algorithm with samples from five individuals, and evaluating it with samples from the remaining individual. Since data contains six different individuals, this process is repeated six times, leaving out a different individual in each round. Then, the average performance is computed. The selected parameter configurations for LBP, S-LBP, and S-LBsP on this dataset are those for which the best F1-scores have been obtained in the previous tables, which are $LBP_{8,1}$ with 49 blocks, $S-LBP_{8,1,4,10}$, and $S-LBsP_{8,1,2,10,8}$, respectively.

Regarding accuracy results, it can be observed that methods based on classifiers obtain, in general, a higher accuracy than those based on template-matching techniques. In addition, the proposed image descriptors, S-LBP and S-LBsP, also outperform the other machine learning approach in both training strategies, h-h and l-o-o. This work, based on a polar map of the hand region [97], extracts a rotation-invariant shape descriptor from it using depth information only. From a recognition viewpoint, depth imagery can offer more advantages, since it provides more structural information, and it allows to remove background features easier. However, a shape descriptor could not be discriminative enough in more difficult scenarios, such as the l-o-o setting. As it can be observed, it is very close to our work in terms of accuracy, but only for the h-h

setting. For the l-o-o setting, the accuracy decreases even below some of the template-matching approaches.

Methods based on template matching	Accuracy	
Shape context with bending cost [96]	0.791	
Shape context without bending cost [96]	0.832	
Skeleton matching [96]	0.786	
Thresholding Decomposition+FEMD [96]	0.932	
Near-convex Decomposition+FEMD [96]	0.939	
Methods based on classifiers	Accuracy	
	h-h	l-o-o
DL + SVM [97]	0.971	0.911
LBP + SVM	0.958	0.935
S-LBP + SVM	0.973	0.959
S-LBsP + SVM	0.979	0.967

Table 3.5: Comparison of the performance of different state-of-the-art image descriptors on the NTU database in terms of average accuracy.

3.6.3 Video descriptor evaluation for gesture recognition

In this section, the temporal approaches described in Section 3.5 are evaluated along with S-LBsP spatial features, and compared with the two LBP-based temporal descriptors, Volume Local Binary Patterns (VLBP) and Local Binary Patterns from Three Orthogonal Planes (LBP-TOP) [29]. Tables 3.6, 3.7, 3.8, and 3.9 summarize the evaluations of VLBP, LBP-TOP, VS-LBsP, and TPM-SLBsP, respectively. They show the best accuracy results for several parameter configurations of the mentioned descriptors, and the resulting descriptors' dimensionalities on the proposed dataset. In particular, the 30 training video sequences have been spatially and temporally segmented, generating new sub-sequences to train the video descriptors. These sub-sequences are divided into two subsets: a training set with 80% of the sub-sequences, and a validation set with the remaining 20%.

To obtain the aforementioned tables, different values for the main parameters have been evaluated, as well as the regularization parameter for the SVM classifiers in the same way as in the previous section. The $VLBP_{L,P,R}$ descriptor depends on three parameters, which are the temporal interval L , the number of neighbors P , and the radius of the neighborhood R . The original implementation of the VLBP descriptor is restricted to a value of $P = 4$ because of the computational cost. In addition, other works [29] have proven that the best results are achieved for the $VLBP_{4,1,1}$ configuration, therefore R is set to 1. Regarding the temporal interval, a range of values that goes from 1 to 4 has been tested to limit the computational complexity.

In the case of $LBP-TOP_{P_{XY},P_{XT},P_{YT},R_X,R_Y,R_T}$ the number of neighbors in each plane can be increased up to $P_{XY} = P_{XT} = P_{YT} = 8$, since it is less computationally demanding. Other works [29] have proven that the best results are obtained for the $LBP-TOP_{8,8,8,1,1,1}$ configuration, therefore 8 neighbors are fixed for every plane, and R_X and R_Y are fixed to 1. Regarding the radius in the

temporal axis, a range of values that goes from 1 to 4 has been tested.

The parameters for VS-LBsP_{P,R,n,M,N,S} are those corresponding to the S-LBsP descriptor (intraframe encoding), and the number of samples S for temporal sampling (interframe encoding). Some of the best parameter configurations showed in Table 3.4 have been used for S-LBsP. Notice how the performance evolves with respect to S , for which a maximum of 10 sampled frames has been tested.

Regarding TPM-LBsP_{P,R,n,M,N,S,l}, the intraframe parameters also correspond to the S-LBsP descriptor, and the interframe ones to the levels l of the temporal pyramid. l encodes both the number of levels and the partitions inside each level. In this case, the best configuration of S-LBsP has been tested for five different TPM settings, as shown in Table 3.9.

Descriptor parameters			SVM parameter	Metrics	Dimension
P	R	L	λ	Accuracy	Length
4	1	1	0.31	0.773	16384
4	1	2	30.61	0.336	16384
4	1	3	0.01	0.555	16384
4	1	4	0.01	0.382	16384

Table 3.6: Average accuracy for different parameter configurations of the VLBP descriptor.

Descriptor parameters						SVM parameter	Metrics	Dimension
P _{XY}	P _{XT}	P _{YT}	R _X	R _Y	R _T	λ	Accuracy	Length
8	8	8	1	1	1	75.01	0.555	768
8	8	8	1	1	2	41.11	0.555	768
8	8	8	1	1	3	80.41	0.382	768
8	8	8	1	1	4	0.01	0.382	768

Table 3.7: Average accuracy for different parameter configurations of the LBP-TOP descriptor.

Descriptor parameters		SVM parameter	Metrics	Dimension
Intra-frame features	Inter-frame param. S	λ	Accuracy	Length
S-LBsP _{8,1,2,4,4}	10	66.31	0.978	2880
S-LBsP _{8,1,2,10,4}	10	23.71	0.987	6720
S-LBsP _{8,1,2,10,8}	5	0.01	0.983	6560
	9	3.31	1	11808

Table 3.8: Average accuracy for different parameter configurations of the VS-LBsP descriptor.

In comparison with the VLBP and LBP-TOP descriptors (Tables 3.6 and 3.7), VS-LBsP and TPM-SLBsP (Tables 3.8 and 3.9) are superior in terms of average accuracy because of the specific combination of spatial and temporal information, and more precisely due to the addition of global

Descriptor parameters		SVM parameter	Metrics	Dimension
Intra-frame features	Inter-frame param. l	λ	Accuracy	Length
S-LBsP_{8,1,2,10,8}	0	3.11	0.973	1312
	0,1	31.01	0.985	3936
	0,1,2	23.11	0.991	9184
	1,2,3	40.81	0.986	18368
	0,1,2,3	66.31	0.985	19680

Table 3.9: Average accuracy for different parameter configurations of the TPM-SLBsP descriptor.

spatial information. In particular, they reach a recognition accuracy of approximately 20% higher than the VLBP descriptor, and 40% higher than the LBP-TOP descriptor. As S-LBsP features are more discriminative than the ones used by VLBP and LBP-TOP, they need to use more temporal information to be as discriminative as possible. For instance, whereas VS-LBsP achieves a perfect accuracy considering only 9 frames, VLBP and LBP-TOP need to analyze every frame ($L = 1$ and $R_T = 1$, respectively) to achieve a high recognition rate. Results for the LBP-TOP descriptor are worse than for the VLBP descriptor, as can be observed in Tables 3.6 and 3.7. The LBP-TOP descriptor was designed to solve the dimensionality problem of the VLBP descriptor. However, this design is based on decreasing the length of the feature vector by reducing the information. As a result, the dimensionality problem is solved, but the extracted features are much less discriminative, obtaining a poor performance. In terms of dimensionality, the shorter descriptor is LBP-TOP, however it could not be considered for this application due to its poor performance. The remaining descriptors keep the same order of magnitude, although TPM-SLBsP is slightly shorter.

When comparing temporal sampling and TPM, it would be expected for TPM to be a bit more discriminative because it extracts information from all the frames. However, they achieve very similar recognition rates. For this particular application, the fact that frames close in time hardly change their appearance makes TPM consider redundant information. This can be also seen in Table 3.9, where the accuracy does not increase when considering more levels in the temporal pyramid. Therefore, an approach that only analyzes a set of key frames, such as temporal sampling, can be enough to obtain a high performance.

3.6.4 Evaluation on the global system

The results for the global system, integrating the detection, tracking and recognition phases are presented in this section. The system has been evaluated on the proposed dataset, by using the aforementioned descriptors.

To train the classifier in the detection stage, spatial windows that tightly surround the hand poses have been used. Therefore, to increase the classifier's ability to detect such hand poses, the spatial step of the sliding window has to be small. In particular, it has been set to 4 pixels. The parameters for the multi-resolution pyramid and the sliding window are the number of scales, the minimum scale and the maximum one, and the dimension of the sliding window. They have been chosen taking into account the sizes of all the bounding boxes annotated in the training images

used for detection. First, the aspect ratio of the sliding window is fixed to the most recurrent value, and the width dimension is fixed to the median value. Then, the size of the sliding window is 87×65 . According to the minimum and maximum size of bounding boxes in training data, the pyramid is then comprised of 8 scales to achieve a trade-off between computational cost and detection rate, and the scale factors for each go from 0.4 to 1.18, equally spaced. For the non-maxima suppression in the same scale, a radius of 8 and a threshold of 10 have been used. For the non-maxima suppression between different scales, the overlapping threshold has been set to 0.75.

The size of the temporal window in the recognition stage is fixed. Its value is estimated considering that the temporal window must fit those *discrete* gestures that last longer, which are left click and right click. In particular, the median value of the lengths of all the training video sequences for recognition is taken, which is in this case $L = 24$ frames. Regarding the validation window, its size has been set to $K = 3$ temporal windows.

The evaluation of the whole system is carried out by using the six testing long sequences in the dataset in terms of average accuracy, and is shown in Table 3.10. The four descriptors evaluated in the previous section and the combinations VS-LBP and TPM-SLBP have been used as feature extraction techniques considering their optimum parameter configurations.

As expected, the recognition scores are in general worse than those obtained with segmented video sequences in the previous section. The continuous execution of several gestures in the testing sequences makes some temporal windows contain the end of one gesture and the beginning of the following one. As a rejection class has not been trained for the recognition phase, these windows containing transition between gestures are incorrectly labeled. However, the validation window tries to solve this by analyzing the output of consecutive temporal windows. In addition, the fact that the size of the temporal window is fixed can degrade the recognition accuracy for those individuals whose execution speed is not appropriate for the size of the estimated temporal window. Despite these facts, the recognition accuracy exceeds a rate of 0.9 for all the sequences when using the proposed descriptors, outperforming other approaches such as VLBP and LBP-TOP.

The VS-LBsP and TPM-SLBP show independently the efficiency of the LBsP and TPM concepts. The first one employs the spatial descriptor S-LBsP and the temporal sampling scheme. As the LBsP approach reduces the number of pattern types, it allows to consider more global spatial information (10×8 spatiograms with respect to the 10×4 of VS-LBP). This not only achieves a shorter length for the final video descriptor, but also a slight rise in the accuracy. This happens for all the testing sequences, as can be seen in the third row of Table 3.10. The second one combines the spatial descriptor S-LBP with the TPM approach. In this case, the recognition rate also increases with respect to VS-LBP for every testing sequence, however the resulting dimension for the final video descriptor is unmanageable. Regarding VS-LBP, it suffers from a dimensionality problem that could limit its practical application due to memory requirements. If less spatial and temporal information are considered, the dimensionality of the VS-LBP descriptor decreases, but at the expense of a lower discriminatory ability. Therefore, a combination of the two approaches S-LBsP and TPM, which is TPM-SLBsP, seems to be the best candidate. Whereas TPM achieves the highest recognition rate, LBsP makes the descriptor's length manageable. Only LBP-TOP is shorter, although it has a very poor recognition rate.

Method	Seq_1	Seq_2	Seq_3	Seq_4	Seq_5	Seq_6	Average accuracy	Dimension
VLBP _{4,1,1} [29]	0.720	0.765	0.711	0.695	0.689	0.770	0.725	16384
LBP-TOP _{8,8,8,1,1,1} [29]	0.507	0.521	0.535	0.487	0.485	0.510	0.507	768
VS-LBP _{8,1,4,10,5}	0.949	0.961	0.935	0.923	0.897	0.959	0.937	52480
VS-LBsP _{8,1,2,10,8,9}	0.984	0.972	0.946	0.934	0.908	0.970	0.952	11808
TPM-SLBP _{8,1,10,8,3}	0.974	0.961	0.976	0.960	0.926	0.975	0.962	145152
TPM-SLBsP _{8,1,2,10,8,3}	0.980	0.957	0.981	0.972	0.924	0.974	0.965	9184

Table 3.10: Comparison of the accuracy and dimensionality among different methods using the whole system on the proposed dataset.

In addition to the discriminatory abilities of the designed descriptors, the proposed system obtains excellent results thanks to several mechanisms to address typical detection/recognition problems. First, the hand detector is properly trained considering a large number of positive and negative samples, and tight windows. Second, the tracking algorithm is able to deal with noisy, false, and missing detections coming from the detection stage. It estimates the best association between current detections and previous detections, in order to generate robust hand trajectories. Finally, the segmentation and validation windows to locate the gesture in time and make a decision over several consecutive temporal windows provide a way to reach a high recognition accuracy. Some examples of correct location, tracking and recognition can be seen in Figure 3.15.

(a) Frame-by-frame trajectory of the cursor.

(b) Example of a left click recognized at the end of the temporal window.

(c) Example of a right click recognized at the end of the temporal window.

(d) Example of O.K.

(e) Example of fist.

Figure 3.15: The above figures show the response of the system while a user executes a natural sequence of hand gestures to use the mouse application. (a) shows the trajectory of the cursor in blue. Once the gesture is recognize as *cursor*, the system responds providing the location of the hand frame by frame. (b) and (c) show the recognition of a *left click* and a *right click* at the end of four temporal windows, where the video descriptor TPM-SLBsP is evaluated, that belong to the same execution sequence. Notice how the system responds even before the gestures end due to the high discriminative power of the descriptor. (d) and (e) show the recognition of the static gestures: *ok* and *fist*.

3.7 Conclusions

In this chapter, a robust hand-gesture recognition system has been designed for HCI with the aim of imitating a mouse-like pointing device. The goal of the system is to locate hands in the spatial domain and recognize gestures along time to trigger the corresponding mouse functionality. To that end, the system comprises three stages to detect, track and recognize hand gestures while

addressing changes in illumination, multiple scales, and user intra-class and inter-class variances. In addition, a hand-gesture dataset has been collected and released to train the system on the defined task. It comprises 5 classes and more than 20500 images.

Novel spatial and temporal features based on LBP descriptor have been designed to improve the performance of the whole system. They are integrated into the detection and recognition stages to detect potential hand regions, and distinguish among different hand gestures, respectively. The proposed descriptors present richer spatio-temporal information than state of the art, and a manageable computational cost.

An extensively evaluation of the proposed image and video descriptors, as well as of the whole system, has been carried out under different configurations in terms of performance and dimensionality. Obtained results have shown the high discriminative power of the proposed descriptors, and the high performance achieved by the whole system with respect to the state of the art, on several publicly available datasets.

Chapter 4

End-to-end deep learning systems for different vision applications

4.1 Introduction

The success of Convolutional Neural Networks (CNN) in image classification [50], and the release of multiple challenges [51, 76] have encouraged their adaptation to other common computer-vision tasks, such as image segmentation [4], and object detection and recognition [107]. Systems that solve these tasks have experienced a clear evolution from multi-stage machine learning, which is composed, at least, by a feature extraction module followed by a shallow classifier/regressor, to end-to-end deep learning, where feature extraction and prediction are jointly addressed. This evolution has been possible thanks to the appearance of large, high-quality labeled data, and the increasing computation capability at lower prices.

New network architectures have arisen to address the typical problems of image classification and object detection. Image classification is straightforward solved by a one-branch deep neural network, where the input is an image, and the output the predicted category. Regarding object detection and recognition networks, they are typically based on two different stages: a region proposal approach that selects a set of regions of interest (ROI) in the image, and a CNN that predicts the object category for every ROI. In a different and novel way, new deep learning designs have been developed to address the task of object detection and recognition. Two different scenarios are considered depending if the object location is relevant for the application goal, or can be simply omitted. If the object location is relevant, an approach based on image segmentation have been developed to address object detection at multiple scales (see Section 4.2). It has been applied to traffic monitoring in highways, where one of the main tasks is vehicle detection. However, if the location is not of interest for the application as in hand-gesture recognition, where the semantic of a gesture is usually independent of its location, an image-classification approach (Section 4.3) can be used to detect objects of interest, even when they occupy a small portion of the image and are embedded in clutter.

CNNs are being also adopted for more complex scenarios such as robotic applications [8, 9].

Generating training data for these types of systems is generally expensive, and sub-optimal performance can pose a danger for users, the environment, and the robot itself, as in collision avoidance, and human-robot cooperation. Therefore, the mapping functions to be learned in a supervised manner must be chosen carefully. Nevertheless, applications such as robot navigation [108] and self-driving cars [9] are finding creative solutions to deal with the lack of training data, and their high complexity when operating in safety-critical environments. For autonomously flying a UAV through the streets of a city (see Section 4.4), the lack of data has been solved by a transfer learning approach through a CNN called DroNet. Unlike other approaches that use simulation to get training data [108], this network has been trained to imitate the behaviour of manned vehicles, and designed to predict the steering angle and the probability of collision to safely fly in a urban environment. Regarding self-driving cars, DL-based systems using standard cameras have demonstrated to achieve a good performance when limited to a set of discrete actions [8]. However, more complex approaches that address a continuous estimation problem such as directly regressing steering angles, did not obtained much improvement with respect to the former. A Dynamic Vision Sensor (DVS), also known as event camera [109], has been used to solve steering angle regression (Section 4.5) due to its ability to capture the dynamics of a scene is higher than with traditional cameras. An approach that unlocks the potential of asynchronous event sensors for synchronous DL algorithms has been developed to benefit from their natural response to motion, and accurately predict a car steering angle.

4.2 A structured CNN for vehicle detection at multiple scales

New intelligent transportation systems based on vision technologies are becoming very important for traffic monitoring and flow analysis, advanced driver assistance, and surveillance. In these scenarios, one of the main tasks is the detection of vehicles in real situations, under different daylight conditions, and with diversity of road configurations. The three main methods that have been proposed for vehicle detection in the past are background subtraction [110], feature-based [111], and motion-based [112]. However, these traditional systems still fail since vehicles can be partially visible, either because they are outside the image boundaries, or occluded by different traffic-related objects (streetlights, crash barriers, bridges, etc.), and the projective geometry of a typical road scene originates dramatic changes in the vehicles' scale.

Recently, DL approaches have shown a higher performance and generalization power than traditional systems in several computer vision-related tasks, such as image classification (see Section 2.3.3), object recognition (see Section 2.3.4), and image segmentation (see Section 2.3.5). For the vehicle-detection task in a heterogeneous network of monitoring cameras, an image-segmentation network seems more appropriate since input images can be of any size. In addition, a semantic segmentation provides a better way to further analyze vehicles regarding their shape, size, and type than with an object-recognition network, specially in those scenarios where the camera viewpoint makes bounding boxes not perfectly fit the vehicle region.

In this section, a segmentation-based approach through a CNN is proposed to detect vehicles in a traffic-monitoring environment. It consists of a fully-convolutional neural network that provides a spatially structured output (structured CNN) that generates a pixel-wise segmentation. Then, a non-maxima suppression technique is applied to obtain a binary segmentation between ve-

hicles and background. The proposed architecture provides a pixel-level classification like image-segmentation networks, but it is simpler since it lacks down-sampling and up-sampling layers. The structured CNN uses the receptive field of every pixel as context of the decision, so that it does not learn abstract features but local ones. A shallower network than the state of the art is considered, which reduces the memory requirements imposed by the fact that feature maps do not decrease across the network. In addition, image-segmentation networks require dense pixel-level labels for training, which demands a high human effort. On the other hand, the proposed design is able to obtain a coarse segmentation using only bounding-box-like annotations.

The rest of the section is structured as follows. Subsection 4.2.1 reviews related work on vision-based vehicle detection. Subsection 4.2.2 describes the proposed methodology. Subsection 4.2.3 shows quantitative and qualitative results of the proposed methodology.

4.2.1 Related work

The advantages of using cameras (low cost, wide field of view, and effectiveness), the advances in image-processing algorithms, and the increasing computation capability have allowed researchers to address new transportation applications such as traffic monitoring [113] and accident detection [114]. In particular, many approaches have been proposed for monocular vehicle detection in the last decade, which are mainly based on appearance and motion information. The former mainly consist of a region proposal method that provides candidates containing potential targets, followed by a feature extraction stage that computes a feature vector (describing the structural and appearance information of vehicles), and a classifier [111, 115] that determines the presence of a vehicle. For instance, histograms of oriented gradients (HOG) are combined with a latent SVM in [111]. A deformable object model is learned and a scanning window approach is used as (exhaustive) region proposal technique. Regarding motion-based works, the most popular techniques are based on background subtraction [110] and optical flow [112], however they have been less common than appearance-based methods for monocular vehicle detection, mainly due to their significant lower performance.

DL-based methodologies for vehicle detection can be divided in two main categories: (i) object-recognition networks (Section 2.3.4), which predict bounding boxes and object categories, and (ii) image-segmentation networks (Section 2.3.5), which perform a semantic segmentation of objects. Regarding the former, OverFeat [116] has been adopted to detect road lanes and cars in a highway using an on-board camera [117]. This work converts the FC layers of OverFeat into convolutional layers to produce a grid of feature vectors. Each feature vector represents a different context location within the pixel space, and is used to predict the presence of an object and a single bounding box. Several scales of the input image need to be passed to the network. Some of the object-recognition neural networks presented in Section 2.3.4, such as Faster R-CNN [72] and YOLO [107], have been also applied to vehicle detection. For instance, Faster R-CNN has been evaluated on the challenging KITTI benchmark [118], which present large variations in vehicle appearance and camera viewpoint, as well as severe occlusions. Parameter tuning and algorithmic modifications were necessary to achieve competitive results [119]. YOLO has been used in a highway environment by pre-training the network on a large multi-purpose dataset, and then fine-tuning the last layer on road images [120].

4.2.2 System description

This section describes the proposed network architecture and the learning process to detect vehicles from an image segmentation viewpoint. The proposed architecture consists of a simpler FCN than image-segmentation networks presented in Section 2.3.5, since it does not include sub-sampling and up-sampling layers. In addition, dense-pixel annotations are not required to train the network on the image segmentation task.

4.2.2.1 Network architecture

The proposed structured convolutional network aims at detecting all the vehicles in the scene from a segmentation viewpoint, providing a structured output of the same size as the input image. The resulting network is shown in Figure 4.1. Its architecture lacks pooling layers to preserve the spatial resolution, and lacks fully-connected layers to maintain the spatial structure. Input RGB images are then forward propagated through a FCN that only contains convolutional layers, and rectified linear units (ReLU). The last layer of the network consists of a classifier that makes a decision over each pixel according to its local context. A sigmoid layer is used to address a binary classification since only two classes are considered: *vehicle* and *background*. The output of the network is a 1-channel probability map that implicitly contains the locations of the objects as groups of pixels with a high probability.

After convolutions, every pixel in the resulting feature maps contains a compact representation of its local context extracted from the receptive fields along the network. A very deep network is not needed in this case because the goal is not learning global abstract features, but local ones. Moreover, a very deep architecture is neither desired to avoid the potential overfitting caused by the reduced available dataset, and the fact of not reducing the feature maps across the network. The number of kernels is related to the variability of the vehicles that must be learnt in terms of types, scales and points of view.

Figure 4.1: The proposed structured CNN for vehicle detection carries out a pixel-level classification from a single RGB image. Its architecture consists of a FCN that lacks pooling layers to preserve the spatial resolution. It contains 5 blocks of convolutional and ReLU layers to compute a compact representation of the local context of every pixel. Then, a sigmoid layer provides a probability of belonging to the *vehicle* class for every pixel. In the notation above, the number of kernels is indicated before the @, and then the kernel's size for every convolutional layer. The stride of the convolutions is 1.

4.2.2.2 Training

The fact that no pooling layers are used increases the memory requirements at training time, since resulting feature maps always have the same size. One solution is to reduce the size of the network input. However, down-sampled versions of the images are not considered because pixel labels from very small vehicles could disappear. Instead, small image patches cropped from the original images are passed as input to the network, which is possible thanks to the fully-convolutional nature of the network. In addition, since the proposed architecture carries out a pixel-wise classification, the information of the whole image is not needed to make decisions. Taking into account the memory requirements, the size of the image patches has been set to 150×150 pixels. In particular, 90% of the image patches are cropped in such a way that the object of interest is centered, and the 10% left are random patches which can contain whole objects, parts of them, or just background.

Labels are the corresponding binary masks of the image patches according to the bounding-box-like annotations. To avoid the network from learning background features contained in bounding boxes, they are reduced by 20% when generating the binary masks. Examples of input patches and their corresponding labels are shown in Figure 4.2.

To overcome the limited amount of data to train the network, data augmentation techniques have been used to create more positive and negative samples, and thus to improve the robustness of the proposed network and to prevent overfitting. In particular, illumination changes are simulated by applying affine transformations to the original image illumination. In addition, patches are also horizontally flipped, and/or translated ± 10 pixels in both directions.

Figure 4.2: Examples of input patches and their corresponding labels from GRAM-RTM dataset [5]: (a) original image, (b) 150×150 image patches after applying augmentation that are used as input to the network, and (c) labels corresponding to the above image patches, where bounding-box annotations have been reduced 20% regarding their original size.

The network has been trained from scratch, initializing the weights of all convolutional layers randomly by a zero-mean Gaussian distribution with a standard deviation 0.01. Binary cross-entropy is used as cost function. In addition, the stochastic gradient descent (SGD) optimizer has

been used with an adaptive learning scheme. It is decreased as the number of iterations increases, from 0.001 for the first 12 epochs, 0.0001 for the following 6 epochs, up to 0.00001 for the rest. Also the momentum method has been considered to improve the learning speed. In particular, a value of 0.9 has been set.

4.2.3 Experimental results

To demonstrate the ability of the proposed approach to detect vehicles in traffic-monitoring scenarios, qualitative and quantitative results in terms of detection are provided on the publicly available GRAM-RTM dataset [5]. The proposed framework has been implemented using the MatConvNet library [121], and the training was performed on a NVIDIA Titan X.

4.2.3.1 Dataset

The Structured CNN has been evaluated on the GRAM Road-Traffic Monitoring (GRAM-RTM) dataset [5], which is a benchmark for multi-vehicle detection and tracking in real-time. It consists of 3 challenging video sequences, recorded under different conditions and with different platforms. Since the interest of this work is the traffic monitoring in highways, only the video sequences recorded in a highway environment are used for the experiments. They are the one captured in a sunny day with a resolution of 480×800 (M-30), and the one captured during a cloudy day with a resolution of 720×1200 (M-30-HD).

The M-30 sequence has been split into two sub-sequences, M-301 and M-302, where the first one contains 90% of the frames, and the second one the rest due to the limited size of the dataset. The M-301 sequence is used for training purposes, and the M-302 for testing. The same setting has been considered for the M-30-HD sequence, from where sub-sequences M-301-HD and M-302-HD have been obtained for training and testing, respectively. Image patches of size 150×150 are extracted from training sequences as explained in Section 4.2.2.2. Only vehicles within the region of interest (ROI) (see the blue area in the third and fourth rows of Figures 4.3 and 4.4) are considered, according to the annotations provided in the dataset. Although multiple vehicle categories (*car*, *truck*, *van*, and *big-truck*) are provided, only a binary detection is carried out (vehicle/background) because of the high imbalance of specific-class samples.

4.2.3.2 Qualitative results

At testing time, the image sequence of traffic flow is passed to the network frame by frame. Figures 4.3 and 4.4 show visual detection results for several frames from the M-302 and M-302-HD video sequences, respectively. The probability values of the network output, which are shown in the first rows, are encoded as a heat map, where red means high probability, and dark blue very low one. As can be seen from these probability maps, the network is able to learn what pixels belong to vehicles, and differentiate them from other objects belonging to background. Just looking at them, it is possible to locate the vehicles at a glance without any type of post-processing. The second rows show the binary masks generated from the probability maps after applying a non-

maxima-suppression threshold of 0.75. This threshold has been computed as the minimum value that maximizes the detection results in terms of F1-score. This step provides a coarse segmentation of the vehicles present in the scene. Since the number of pixels correctly classified cannot be determined because the GRAM-RTM dataset only provides annotations in form of bounding boxes, the previous coarse segmented area is overlapped with the original frame (red area) in the third row to visually compare the predicted and current vehicle areas. However, the segmented areas can be used to compute bounding boxes, as shown in fourth rows.

Figure 4.3: Visual results in four different scenarios (columns) in the M-302 sequence. The first row shows the network output as a probability map, where red means high probability of belonging to the *vehicle* class, and dark blue very low one. The second one shows the binary masks extracted from the probability maps after applying a non-maxima-suppression threshold. The third row shows the segmented area (in red) overlapped with the original frame. The fourth row shows the bounding boxes computed from the segmented area: green bounding boxes are predicted by the network, and red ones are from ground truth.

The number of false positive pixels depends on the non-maxima-suppression threshold used to classify pixels as positive or negative. As can be observed in the first row of Figures 4.3 and 4.4, the background noise is very low. Therefore, a high threshold can be applied to remove false positive pixels. The result of applying a threshold of 0.75 is shown in the second row of the Figures 4.3 and 4.4, where only pixels belonging to vehicles are observed.

Regarding the false negative pixels, they usually appear in challenging vehicles such as motorbikes and trucks. For instance, the first column in Figure 4.3 shows a motorbike sample whose segmentation after applying the threshold has a lower quality in comparison with other types of

Figure 4.4: Visual results in four different scenarios (columns) in the M-302-HD sequence. The first row shows the network output as a probability map, where red means high probability of belonging to the *vehicle* class, and dark blue very low one. The second one shows the binary masks extracted from the probability maps after applying a non-maxima-suppression threshold. The third row shows the segmented area (in red) overlapped with the original frame. The fourth row shows the bounding boxes computed from the segmented area: green bounding boxes are predicted by the network, and red ones are from ground truth.

vehicles. The same happens for trucks, as can be seen in the third column of Figure 4.4. In this case, the truck presents some holes corresponding to pixels with a very low probability in the network output, so that its segmentation gives confusing information about its shape and size. This poor segmentation is caused by the small number of training samples for these types of vehicles, so that the set of pixels defining this type of vehicles generally present a lower probability than the ones belonging to cars. False negative pixels also appear in the contour of all vehicles, as can be observed in the third rows of Figures 4.3 and 4.4. This is due to the reduction of 20% of the bounding boxes, so that pixels outside of the reduced bounding boxes have not been learnt by the network. This makes the computed bounding boxes in green be smaller than the ones from ground truth in red, as can be seen in the corresponding fourth rows. This could be solved by training the network using masks containing the segmented vehicle. However, in this case, the GRAM-RTM dataset only provides annotations in form of bounding boxes.

Nonetheless, most of the pixels belonging to vehicles are correctly classified with a very high probability, independently of the object size. Therefore, the Structured CNN is able to manage multiresolution local features without using any previous stage. Every filter in the convolutional

layers is focused on learning a particular feature at a specific scale, so that the network learns general features from different objects that can appear at multiple scales. It is also worth mentioning that, although the network has only been trained with vehicle samples within the ROI (blue area), it is able to correctly predict pixels in those vehicles that are outside of the ROI, and present a slightly different viewpoint, as shown Figure 4.5.

Figure 4.5: Examples of network predictions outside the ROI for the M-302-HD sequence. Red-dotted line indicates the boundary of the ROI, so that vehicles outside it are in the upper-left region.

4.2.3.3 Quantitative results

As the dataset does not provide annotations in terms of segmentation, the number of pixels correctly classified cannot be determined. However, the dataset does provide ground truth by means of bounding boxes to locate the vehicles in the scene. Bounding boxes have been obtained from the segmentation of the network output after applying the non-maxima suppression, by computing the smallest rectangle containing the segmented region, as can be seen in the fourth rows of Figures 4.3 and 4.4. In general, the predicted bounding boxes are slightly smaller than the ones from the ground truth because of the false negative pixels that appear in the contours of the vehicles. Nevertheless, they contain most parts of the vehicle and can be used to locate it.

To objectively measure the detection results, precision (3.8), recall (3.9), and F1-score (3.10) metrics are used. These metrics have been computed for every frame, and Table 4.1 shows the average values for every test sequence. As can be observed, both precision and recall present very high rates. In both sequences, precision is slightly lower than recall because the number of false positive bounding boxes is larger than the false negative ones. This happens when a compact segmentation is not obtained, and several bounding boxes are generated for the same vehicle. For instance, the segmented parts of the motorbike in Figure 4.3 are not connected, hence multiple segmentations are obtained for it. Therefore, the number of false positive predictions increases.

In general, it is desirable that recall be higher than precision for object detection, because it shows if every annotated vehicle in the dataset has been predicted. Therefore, it is very important that the number of false negatives is as low as possible. In this case, the recall metric is very close to 1 for the M-302 sequence, and approximately 1 for the M-302-HD sequence. F1-score rates exceed 0.87 in both sequences, confirming that the proposed network is able to robustly and

accurately locate vehicles in a highway environment.

Table 4.1: Detection results in terms of precision, recall, and F1 score for the testing sequences in GRAM-RTM dataset [5].

Sequence	Precision	Recall	F1 score
M-302	0.800	0.972	0.877
M-302-HD	0.884	0.999	0.938

4.3 Hand-gesture recognition from an image classification viewpoint

Hand-gesture recognition is a clear example where the location of the hand is not necessary for the recognition of the gesture that will trigger some command. However, as seen in Chapter 3, vision systems based on traditional machine learning require a previous hand-localization stage that provides region proposals, with the purpose of minimizing the signal/clutter relationship and then, improving the recognition performance. This has been a common practice, specially when the hand occupies a small portion of the image and is embedded in clutter. Hand-crafted image and video descriptors are then applied to the regions of interest to maximize their discriminative power. Now in the DL era, hand-gesture localization and recognition could be addressed in an end-to-end manner (see the approaches described in Section 2.3.4), such as Faster R-CNN [119], and YOLO [107]. However, the semantic of a gesture is independent of its location in many vision-based applications [122]. Since CNNs are able to learn spatial-invariant features automatically, they can be used to perform the classification of hand gestures from an image classification viewpoint. This has two advantages. First, it reduces the complexity of the network since it does not require any previous detection stage or region proposal mechanism. Second, it abolishes the hard manual labor of annotating bounding boxes, which is required for all the aforementioned approaches.

To recognize hand gestures under the mentioned conditions (without detection stage, and therefore with a challenging signal to clutter ratio), a deep convolutional neural network has been designed from scratch to directly carry out a classification task. Unlike other object detection approaches, the output of the network is the prediction of a potential performed hand gesture. The proposed network can effectively recognize hand gestures in arbitrary positions that occupy a reduced image area in comparison with the whole image. This fact has conditioned the design of the CNN to deal with the potential overfitting problems. In addition, a new dataset has been created, containing seven static hand gestures (see Section 4.5.3.1). The data collection has been simplified since no bounding boxes are required for the training stage.

The rest of the section is organized as follows. Subsection 4.3.1 reviews related work on hand-gesture recognition. Subsection 4.3.2 describes the proposed learning approach. Subsection 4.3.3 evaluates the performance of the proposed methodology, and discuss the quantitative and qualitative results obtained.

4.3.1 Related work

Feature extraction techniques have been the most popular for hand-gesture recognition in vision-based systems, from simpler such as motion [123] and skin color [124], to more complex like hand-crafted image and video descriptors [98]. Other works have proposed articulated models [125] for pose estimation. They have been proposed along with classifiers, such as Support Vector Machines (SVM) [126], and generative approaches, such as Hidden Markov Models (HMM) [46], Conditional Random Fields (CRF) [127]. Moreover, multi-modal approaches [128, 129, 130, 97] were also introduced to improve hand-gesture recognition, since they could be more robust to varying lighting conditions, orientations, and intra-class and inter-class variances. All of them have in common the location of the hand in order to recognize the gesture. For example, the work in [97] describes a hand gesture recognition system that includes the following stages: hand segmentation using depth information, feature extraction via a 2D shape descriptor, and classification based on SVMs. In [130], hand location is carried out by a short-range radar, which guides the segmentation of the hand region from a depth camera that is in turn combined with radar information to obtain a radar image. Finally, depth, grayscale and radar images are used to train a neural network. The work in [131] recognizes dynamic hand gestures from segmented images by means of a combined DNN, which consists of two 3D convolutional neural networks that carries out a classification task. Unlike it, the proposed network demonstrates that a single CNN can learn hand-gesture features from only raw intensity images, even when hands occupy a small portion of the image and the background is complex.

4.3.2 System description

The proposed approach is based on a deep neural network that is able to directly recognize hand gestures from an image-classification viewpoint, but with the challenge of hand regions only represent a small part of the image. Unlike typical approaches, it neither relies on region proposal networks nor infers hand location. Consequently, the system can be trained more easily (no hand locations must be provided), and more accurately (no errors are inherited from hand segmentation/location modules). The counterpart is that the image classification model must face a situation of extreme clutter, i.e. most of the image contains structures not related with hand gestures. To that end, a deep CNN has been designed to learn high-level abstract features, since deeper architectures allow networks to learn more powerful semantic representations. In addition, one of the main guidelines of the design has been to deal with overfitting, which can cause misleading behaviors during learning, such as trying to infer the hand gesture from non-related image areas (for example the human body or the human face). To prevent this kind of behaviors, small convolutional filter sizes have been adopted to restrict the number of network parameters. This leads to a significant improvement in the accuracy in comparison with larger filter sizes. Additionally, a dropout learning strategy [60] has been considered in the last fully connected layers.

Figure 4.6: Architecture of the proposed network for hand-gesture recognition. It predicts the category of a gesture from a 320×240 grayscale image. It comprises 9 convolutional layers for feature extraction, and 3 FC layers for classification. Each FC layer is followed by a dropout unit with a 0.5 dropout ration to prevent overfitting. The last layer consists of a softmax function with 7 units corresponding to 7 gesture categories. In the notation above, first the kernel's size, then the number of filters, and eventually the stride if it is different from 1, are indicated for each convolution, and the output dimension for each FC layer.

To train the network, synthetic translations have been performed over the training images. The stochastic gradient descent (SGD) algorithm has been used as optimizer with a learning rate of 0.01, a regularization parameter of 5×10^{-4} , and a momentum of 0.9. As loss function, the categorical-cross entropy (CCE) has been used. Finally, an early stopping criteria for the training has been adopted as soon as the performance on the validation data starts to decrease, which complements the set of measures oriented to avoid overfitting.

4.3.3 Experimental results

4.3.3.1 Dataset

A new hand-gesture dataset with more than 260.000 images has been created to evaluate the proposed network. The data-collecting task has been especially designed to avoid the manual bounding-box annotations, which makes it possible to create a huge a dataset with less effort in comparison with other approaches. The collection procedure consisted of recording one video per person and per gesture, and automatically labeling the frames. The dataset is divided into 40 people, and 7 different hand gestures. 25% of the images were recorded in simple (uniform) background, and the rest in complex ones. The considered environments with complex backgrounds are highly cluttered and the illumination conditions undergo large variations. In addition, people were asked to move their hands freely, so that gestures are performed in different locations of the image. Some examples can be seen in Figure 4.7, where hand gestures only occupy 10% or less of pixels of the image.

(a) Simple background

(b) Complex background

Figure 4.7: Collected dataset: examples of the seven gestures embedded in the two types of background. From left to right and top to bottom: *fist*, *L*, *OK*, *palm*, *pointer*, *thumb down*, and *thumb up*.

4.3.3.2 Qualitative and quantitative results

To evaluate the proposed network on the collected dataset, 3 rounds of cross-validation have been performed, each one with 25 people for training, 5 people for validation, and 10 for testing. In addition, testing images with simple and complex backgrounds have been evaluated separately. The classification performance of the proposed network is measured by using the confusion matrix and the average accuracy. In addition, to better understand how the network is able to recognize such small hand gestures, and better explain the reasons of misclassification, the class activation maps (CAM) [132] have been computed. They provide a visualization of the regions of the image that the network uses to make a decision. CAMs have been computed for some correctly classified and some misclassified testing images, shown in Figures 4.8 and 4.9, respectively. These figures also show the predicted class and its corresponding probability or confidence score. As can be observed in Figure 4.8, the network learns particular parts of the hand that are different among gestures. For instance, for hand gestures that could have a similar pose, such as *palm*, *ok*, and *L*, the network mostly focuses on the area of the thumb and the forefinger. For the rest of the gestures, the network mainly pays attention to the center area of the hand.

Figure 4.8: Examples of correct classification. The first row shows simple backgrounds, and the second one complex backgrounds. The caption of each sub-figure indicates the inferred class and its corresponding confidence score.

Table 4.2 shows the confusion matrix for testing images corresponding to a simple background for one round of cross-validation. In general, all the gestures achieve a high recognition rate, above 90% (main diagonal of the confusion matrix), obtaining an average accuracy of 0.991. The gesture that obtains the poorest performance is *OK*, which is mostly misclassified with *palm* and *thumb up*. According to the CAM in Figure 4.8 (d), the network focuses on the circle the user makes with his thumb and forefinger fingers to recognize an *OK*. However, if the circle is occluded due to the viewpoint, it can be misclassified as a *palm*, as shown Figure 4.9 (b). The network predicts a *palm* with a probability of 0.527, and an *OK* with a probability of 0.472. Other misclassification cases occur because the network focuses on different areas of the image to make a decision, such

Figure 4.9: Examples of misclassification. The first row shows simple backgrounds, and the second one complex backgrounds. The caption of each sub-figure indicates the inferred class and its corresponding confidence score.

as in Figures 4.9 (c) and (d). The reasons why the filters trained to recognize *fist* and *palm* did not obtain large responses in these cases could be due to blurring for the former, and illumination for the latter.

Class	fist	L	OK	palm	pointer	thumb D	thumb U
fist	0.993	0.001	0	0	0	0	0
l	0.001	0.994	0.004	0	0	0	0.002
ok	0	0	0.969	0.015	0	0	0.016
palm	0	0	0	0.995	0.005	0	0
pointer	0	0	0.001	0	0.999	0	0
thumb D	0	0	0	0	0.002	0.998	0
thumb U	0	0	0	0	0.008	0.001	0.991

Table 4.2: Confusion matrix for hand-gesture recognition on simple backgrounds.

Table 4.3 shows the confusion matrix for testing images corresponding to a complex background for the same round of cross-validation as in the previous table. As expected, the recognition performance decreases with respect to simple backgrounds to 0.842, however, it is still over 75% for all the gestures. Notice that in complex backgrounds, hands occupy a smaller area of the image, and the background is highly cluttered. In this case, *OK* is the one with the lowest recognition rate again, followed by *pointer*. An example of *pointer* misclassification can be observed in Figure 4.9 (e). The network focuses on background areas to decide an *L* because filters trained to recognize *L* features obtained a large response in those areas. In other cases, such as in Figure 4.9 (g), the network pays attention to the correct area of the *palm*, but also to another that obtains a

larger response. As a result, 4.9 (g) is classified as a *fist* with a probability of 0.543, and as a *palm* with a confidence score of 0.454.

Class	fist	L	OK	palm	pointer	thumb D	thumb U
fist	0.848	0.079	0.004	0.100	0.049	0.004	0.004
l	0.062	0.879	0.014	0.011	0.011	0.001	0.024
ok	0.025	0.075	0.759	0.066	0.017	0.001	0.056
palm	0.003	0.002	0.003	0.976	0.012	0.001	0.004
pointer	0.152	0.029	0.001	0.032	0.762	0.002	0.019
thumb D	0.013	0.001	0	0.005	0.001	0.866	0.115
thumb U	0.014	0.073	0.003	0.034	0.021	0.048	0.806

Table 4.3: Confusion matrix for hand-gesture recognition on complex backgrounds.

Table 4.4 shows the average results obtained for the 3 rounds of cross-validation, for both simple and complex backgrounds. The proposed network is also compared with AlexNet [50] and VGG19 [68] in terms of number of network parameters, average accuracy, and GPU time required to predict a gesture from an input image. They have been trained from scratch. As can be observed, the proposed model outperforms the other approaches in both average accuracy and computational time. VGG has a similar accuracy to the proposed network in simple backgrounds, however it is 7.7% worse in complex backgrounds. In addition, its processing time per image is 10 times longer, since its architecture is more complex (around 56×10^6 parameters). Although AlexNet achieves an average accuracy of more than 85% for simple backgrounds and is faster than VGG, its recognition rate for complex backgrounds is very poor.

Architecture	Num. of parameters	Average accuracy		GPU time (ms/per image)
		Simple backg.	Complex backg.	
AlexNet [50]	60×10^6	0.863	0.694	4.89
VGG19 [68]	56×10^6	0.962	0.776	25.10
Proposed net	2×10^6	0.971	0.853	2.96

Table 4.4: Comparison among the proposed network, AlexNet and VGG19 in terms of accuracy and GPU processing time per image.

The proposed hand-gesture recognition system has demonstrated to achieve competitive results addressing the problem from an image-classification viewpoint. Although image classification seems an easy task, it turns into complex when hand gestures occupy less than 10% of the image, and are embedded in a highly cluttered background. Notice that the proposed network is able to generalize to unseen users, since both confusion matrices show results on one round of cross-validation, which does not contain all the users for evaluation.

Unlike the multi-stage system presented in Chapter 3, the proposed system does not rely on previous hand-detection stages that require a lot of time, not only to design every specific step

of the pipeline (detection stage, descriptor, classifier, etc.), but also to achieve a real-time performance. According to this, the proposed DL-based approach, is more affordable and efficient, both in speed and memory.

4.4 DroNet: a CNN to learn how to fly autonomously

Safe and reliable outdoor navigation of autonomous systems, *e.g.* unmanned aerial vehicles (UAVs), is a challenging open problem in robotics. Being able to successfully navigate while avoiding obstacles is indeed crucial to unlock many robotics applications, *e.g.* surveillance, construction monitoring, delivery, and emergency response [133, 134, 135]. A robotic system facing the above tasks should simultaneously solve many challenges in perception, control, and localization. These become particularly difficult when working in urban areas. In those cases, the autonomous agent is not only expected to navigate while avoiding collisions, but also to safely interact with other agents present in the environment, such as pedestrians or cars.

The traditional approach to tackle this problem is a two step interleaved process consisting of (i) automatic localization in a given map (using GPS, visual and/or range sensors), and (ii) computation of control commands to allow the agent to avoid obstacles while achieving its goal [133, 136]. Even though advanced SLAM algorithms enable localization under a wide range of conditions [137], visual aliasing, dynamic scenes, and strong appearance changes can drive the perception system to unrecoverable errors. Moreover, keeping the perception and control blocks separated not only hinders any possibility of positive feedback between them, but also introduces the challenging problem of inferring control commands from 3D maps.

Civilian drones are soon expected to be used in a wide variety of tasks, such as aerial surveillance, delivery, or monitoring of existing architectures. Nevertheless, their deployment in urban environments has so far been limited. Indeed, in unstructured and highly dynamic scenarios, drones face numerous challenges to navigate autonomously in a feasible and safe way. In contrast to traditional “map-localize-plan” methods, a data-driven system, called DroNet, to cope with the above challenges is proposed. DroNet is a convolutional neural network that can safely drive a drone through the streets of a city. Designed as a fast 8-layer residual network, DroNet produces two outputs for each single input image: a steering angle to keep the drone navigating while avoiding obstacles, and a collision probability to let the UAV recognize dangerous situations and promptly react to them. The proposed design represents a good fit for navigation-related tasks, enabling real-time processing of the video stream recorded by a UAV’s camera. The challenge is however to collect enough data in an unstructured outdoor environment such as a city. Clearly, having an expert pilot providing training trajectories is not an option given the large amount of data required and, above all, the risk that it involves for other vehicles or pedestrians moving in the streets. Therefore, data collected by cars and bicycles are used to train a UAV. Since these vehicles are already integrated into the urban environment, they would not endanger other vehicles and pedestrians. Although trained on city streets from the viewpoint of urban vehicles, the navigation policy learned by DroNet is highly generalizable. Indeed, it allows a UAV to successfully fly at relative high altitudes and even in indoor environments, such as parking lots and corridors. The proposed learning-based system, although achieving remarkable results, do not try to replace traditional “map-localize-plan” approaches for drone navigation, but rather investigate whether a

similar task could be done with a single shallow neural network.

The rest of the section is structured as follows. Subsection 4.4.1 presents most relevant related work on autonomous drone navigation. Subsection 4.4.2 describes the proposed methodology regarding learning approach and drone control. Subsection 4.4.3 shows quantitative and qualitative results of the performance of the proposed methodology.

4.4.1 Related work

A wide variety of techniques for drone navigation and obstacle avoidance can be found in the literature. At high level, these methods differ depending on the kind of sensory input and processing employed to control the flying platform.

A UAV operating outdoor is usually provided with GPS, range, and visual sensors to estimate the system state, infer the presence of obstacles, and perform path planning [133, 136]. Nevertheless, such works are still prone to fail in urban environments where the presence of high rise buildings, and dynamic obstacles can result in significant undetected errors in the system state estimate. The prevalent approach in such scenarios is SLAM, where the robot simultaneously builds a map of the environment and self-localizes in it [137]. On the other hand, while an explicit 3D reconstruction of the environment can be good for global localization and navigation, it is not entirely clear how to infer control commands for a safe and reliable flight from it.

Recently, there has been an increasing research effort in directly learning control policies from raw sensory data using Deep Neural Networks. These new approaches have offered a way to tightly couple perception and control, achieving impressive results in a large set of tasks. These methodologies can be divided into two main categories: (i) methods based on reinforcement learning (RL) [138, 108] and (ii) methods based on supervised learning [139, 140, 141, 7, 142]. RL methods suffer from significantly high sample complexity, hindering their application to UAVs operating in safety-critical environments. In contrast, supervised-learning methods offer a more viable way to learn effective flying policies [139, 141, 7], but they still leave the issue of collecting enough expert trajectories to imitate. Additionally, as pointed out by [7], collision trajectories avoided by expert human pilots are actually necessary to let the robotic platform learn how to behave in dangerous situations.

While RL-based algorithms have been successful in learning generalizing policies [138, 143], they usually require a large amount of robot experience which is costly and dangerous to acquire in real safety-critical systems. In contrast, supervised learning offers a more viable way to train control policies, but clearly depends upon the provided expert signal to imitate. This supervision may come from a human expert [139], hard-coded trajectories [7], or model predictive control [140]. However, when working in the streets of a city, it can be both tedious and dangerous to collect a large set of expert trajectories, or evaluate partially trained policies [139]. Additionally, the domain-shift between expert and agent might hinder generalization capabilities of supervised learning methods. Indeed, previous work in [141, 142] trained a UAV from video collected by a mountain hiker but did not show the learned policy to generalize to scenarios unseen at training time. Another promising approach has used simulations to get training data for reinforcement or imitation learning tasks, while testing the learned policy in the real world [144, 145, 108]. Clearly,

this approach suffers from the domain shift between simulation and reality and might require some real-world data to be able to generalize [108]. Indeed, current simulators still fail to model the large amount of variability present in an urban scenario and are therefore not fully acceptable for this task. Additionally, even though some pioneering work has been done in [144], it is still not entirely clear how to make policies learned in simulation generalize into the real world.

To overcome the above-mentioned limitations, a neural network policy is trained by imitating expert behaviour which is generated from wheeled manned vehicles only. Even though there is a significant body of literature on the task of steering angle prediction for ground vehicles [8, 146], the goal is not to propose yet another method for steering angle prediction, but rather to prove that this expertise can be deployed also on flying platforms. The result is a single shallow network that processes all visual information concurrently, and directly produces control commands for a flying drone. The coupling between perception and control, learned end-to-end, provides several advantages, such as a simpler and lightweight system, and high generalization abilities.

4.4.2 System description

The proposed learning approach aims at reactively predicting a steering angle and a probability of collision from the drone on-board forward-looking camera. These are later converted into control flying commands which enable a UAV to safely navigate while avoiding obstacles. During the training procedure, only imagery recorded by manned vehicles is used. Steering angles are learned from images captured from a car, while probability of collision, from a bicycle.

4.4.2.1 Learning approach

Since the goal is to reduce the bare image processing time, a single convolutional neural network (CNN) with a relatively small size is advocated. The resulting network, which is called DroNet, is shown in Figure 4.10 (a). Its architecture is partially shared by the two tasks to reduce the network's complexity and processing time, but is then separated into two branches at the very end due to their different nature and output range.

The part of the network that is shared by the two tasks consists of a ResNet-8 architecture followed by a dropout of 0.5 and a ReLU non-linearity. The residual blocks of the ResNet, proposed in [69], are shown in Figure 4.10 (b). Dotted lines represent skip connections defined as 1×1 convolutional shortcuts to allow the input and output of the residual blocks to be added. The residual scheme has been primarily introduced to address the degradation problem [69], which arises from difficulties in networks' optimization. Therefore, since residual architectures are known to help generalization on both shallow and deep networks [69], this design choice has been adapted to increase model performance. After the last ReLU layer, tasks stop sharing parameters, and the architecture splits into two different fully-connected layers. The first one outputs the steering angle, and the second one a collision probability.

Figure 4.10: (a) DroNet is a forked Convolutional Neural Network that predicts, from a single 200×200 frame in gray-scale, a steering angle and a collision probability. The shared part of the architecture consists of a ResNet-8 with 3 residual blocks (b), followed by dropout and ReLU non-linearity. Afterwards, the network branches into 2 separated fully-connected layers, one to carry out steering prediction, and the other one to infer collision probability. In the notation above, first the kernel's size, then the number of filters, and eventually the stride if it is different from 1, are indicated for each convolution.

Steering prediction is a regression problem, while collision prediction is addressed as a binary classification problem. Mean-squared error (MSE) and binary cross-entropy (BCE) are used to train the steering and collision predictions, respectively. However, a naive joint optimization poses serious convergence problems due to the very different gradients' magnitudes that each loss produces. More specifically, imposing no weighting between the two losses during training results in convergence to a very poor solution. This can be explained by the difference of the gradients' magnitudes in the classification and regression task at the initial stages of training, which can be problematic for optimization [147]. Indeed, the gradients from the regression task are initially much larger, since the MSE gradients' norms are proportional to the absolute steering error. To solve this problem, a weighting strategy is used, in which more and more weight is given to the classification loss in later stages of training. Once the losses' magnitudes are comparable, the optimizer will try to find a good solution for both at the same time. This can be seen as a particular form of curriculum learning [147]. More in detail, the weight coefficient corresponding to BCE is defined in (4.1), while the one for MSE is always 1. Experiments showed that classification loss was comparable to regression loss from epoch 10 on, therefore $decay = \frac{1}{10}$, and $epoch_0 = 10$ have been set.

$$L_{tot} = L_{MSE} + \max(0, 1 - \exp^{-decay(epoch - epoch_0)})L_{BCE} \quad (4.1)$$

The Adam optimizer [56] is used with a starting learning rate of 0.001 and an exponential per-step decay equal to 10^{-5} . Hard-negative mining is also employed for the optimization to focus on those samples which are the most difficult to learn. In particular, the k samples with the highest

loss in each epoch are selected, and the total loss is computed according to Equation (4.1). The value of k is defined so that it decreases over time.

4.4.2.2 Drone control

The outputs of DroNet are used to command the UAV moving on a plane with forward velocity v_k and steering angle θ_k . More specifically, the probability of collision p_t provided by the network is used to modulate the forward velocity: the vehicle is commanded to go at maximal speed V_{max} when the probability of collision is null, and to stop whenever it is close to 1. We use a low-pass filtered version of the modulated forward velocity v_k to provide the controller with smooth, continuous inputs ($0 \leq \alpha \leq 1$):

$$v_k = (1 - \alpha)v_{k-1} + \alpha(1 - p_t)V_{max}. \quad (4.2)$$

Similarly, the predicted scaled steering s_k is mapped into a rotation around the body z -axis (yaw angle θ), corresponding to the axis orthogonal to the propellers' plane. Concretely, s_k is converted from a $[-1, 1]$ range into a desired yaw angle θ_k in the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$ and low-pass filtered:

$$\theta_k = (1 - \beta)\theta_{k-1} + \beta \frac{\pi}{2} s_k. \quad (4.3)$$

In all the experiments $\alpha = 0.7$ and $\beta = 0.5$ have been set, while V_{max} was changed according to the testing environment. The above constants have been selected empirically trading off smoothness and reactivity of the drone's flight. As a result, a reactive navigation policy that can reliably control a drone from a single forward-looking camera is obtained. An interesting aspect of this approach is that a collision probability from a single image can be produced without any information about the platform's speed. Indeed, it is conjectured that the network makes a decision on the base of the distance to observed objects in the field of view. Convolutional networks are in fact well known to be successful on the task of monocular depth estimation [145].

4.4.3 Experimental results

Quantitative and qualitative results of the proposed methodology are shown in this section. First, the datasets used to train the network are presented. Then, the accuracy of DroNet is evaluated with a set of performance metrics, and also its control capabilities, comparing them against a set of navigation baselines.

The experiments were performed on a Parrot Bebop 2.0 drone. Designed as an outdoor hobby platform, it has a basic and rather inaccurate, visual odometry system that allows the user to provide only high-level commands, such as body-frame velocities, to control the platform. Velocity commands are produced by the proposed network running on an Intel Core i7 2.6 GHz CPU that receives images at 30 Hz from the drone through Wi-Fi.

4.4.3.1 Dataset

To learn steering angles from images, one of the publicly available datasets from Udacity's project [148] is used. This dataset contains over 70,000 images of car driving distributed over 6 experiments, 5 for training and 1 for testing. Every experiment stores time-stamped images from 3 cameras (left, central, right), IMU, GPS data, gear, brake, throttle, steering angles and speed. For the ongoing experiments, only images from the forward-looking camera (Figure 4.11 (a)) and their associated steering angles are used.

Since there are no public datasets that associate images with collision probability according to the distance to the obstacles, a collision dataset is collected by mounting a GoPro camera on the handlebars of a bicycle. The bicycle is driven along different areas of a city, trying to diversify the types of obstacles (vehicles, pedestrians, vegetation, under-construction sites) and the appearance of the environment (Figure 4.11 (b)). This way, the drone is able to generalize under different scenarios. The recording starts when the bicycle is far away from an obstacle and stop when it is very close to it. In total, around 32,000 images distributed over 137 sequences for a diverse set of obstacles are collected. The sequences are manually annotated, so that frames far away from collision are labeled as 0 (no collision), and frames very close to the obstacle are labeled as 1 (collision), as can be seen in Figure 4.11 (b). Collision frames are the types of data that cannot be easily obtained by a drone but are necessary to build a safe and robust system. In addition, they do not require any state estimate or even an expert drone pilot, while it exposes pedestrians, other vehicles, and the drone itself to no danger.

(a) Udacity dataset

(b) Collected dataset

Figure 4.11: (a) Udacity images used to learn steering angles. (b) Collected images to learn probability of collision. The green box contains no-collision frames, and the red one collision frames.

4.4.3.2 Regression and classification results

The regression and classification performances of the proposed model are evaluated employing the testing sequence from the Udacity dataset [148], and 8 sequences from the collision dataset, respectively. To assess the performance on collision prediction, the already seen average classification accuracy and the F1-score are used. To quantify the performance on steering prediction, two metrics are used: root-mean-squared error (RMSE) and explained variance ratio (EVA). The RMSE (4.4) measures the average magnitude of the prediction error, indicating how close the observed values α are to those predicted by the network $\hat{\alpha}$,

$$\text{RMSE} \doteq \sqrt{\frac{1}{N} \sum_{j=1}^N (\hat{\alpha}_j - \alpha_j)^2}. \quad (4.4)$$

The EVA (4.5) measures the proportion of variation in the predicted values with respect to that of the observed values. Such variations are given by the variance of the residuals $\text{Var}(\hat{\alpha} - \alpha)$ and the variance of the observed values $\text{Var}(\alpha)$.

$$\text{EVA} \doteq 1 - \frac{\text{Var}(\hat{\alpha} - \alpha)}{\text{Var}(\alpha)}. \quad (4.5)$$

If predicted values approximate the observed values well, the residual variance will be less than the total variance, resulting in $\text{EVA} \lesssim 1$. Otherwise, the residual variance will be equal or greater than the total variance, producing $\text{EVA} = 0$ or $\text{EVA} < 0$, respectively.

Table 4.5 compares DroNet against a set of other architectures from the literature [69, 68, 141]. Additionally, some weak baselines such as a constant estimator, which always predicts 0 as steering angle and “no collision”, and a random one are also used for comparison. From these results it can be observed that the proposed design, even though 80 times smaller than the best architecture, maintains a very good performance in both tasks while achieving real-time operation (20 frames per second). Furthermore, the positive comparison against the VGG-16 architecture indicates the advantages in terms of generalization due to the residual learning scheme, as discussed in Section 4.4.2.1. The proposed design succeeds at finding a good trade-off between performance and processing time as shown in Table 4.5 and Figure 4.12. Indeed, in order to enable a drone to promptly react to unexpected events or dangerous situations, it is necessary to reduce the network’s latency as much as possible.

Model	EVA	RMSE	Average accuracy	F-1 score	Number layers	Number parameters	Processing time [fps]
Random baseline	-1.0 ± 0.022	0.3 ± 0.001	$50.0 \pm 0.1\%$	0.3 ± 0.01	-	-	-
Constant baseline	0	0.2129	75.6%	0.00	-	-	-
Giusti et al. [141]	0.672	0.125	91.2%	0.823	6	5.8×10^4	23
ResNet-50 [69]	0.795	0.097	96.6%	0.921	50	2.6×10^7	7
VGG-16 [68]	0.712	0.119	92.7%	0.847	16	7.5×10^6	12
DroNet	0.737	0.109	95.4%	0.901	8	3.2×10^5	20

Table 4.5: Quantitative results on regression and classification tasks: EVA and RMSE are computed on the steering regression task, while average accuracy and F1-score are evaluated on the collision prediction task. The on-line processing time in frames per second (fps) is additionally reported, achieved when receiving images at 30 Hz from the UAV.

Figure 4.12: Model performance: (a) Probability Density Function (PDF) of actual vs. predicted steerings on the Udacity dataset testing sequence. (b) Confusion matrix on the collision classification evaluated on testing images of the collected dataset. Numbers in this matrix indicate the percentage of samples falling in each category.

4.4.3.3 Quantitative results on DroNet’s control capabilities

DroNet system has been tested by autonomously navigating in a number of different urban trails, including straight paths and sharp curves. Moreover, to test the generalization capabilities of the learned policy, experiments in indoor environments have been also performed. An illustration of the testing environments can be found in Figures 4.13 and 4.14. The proposed approach is compared against two baselines:

(a) *Straight line policy*: trivial baseline consisting in following a straight path in open-loop. This baseline is expected to be very weak, given that we always tested in environments with curves.

(b) *Minimize probability of collision policy*: strong baseline consisting in going toward the

direction minimizing the collision probability. For this approach, the algorithm proposed in [7] was implemented, which was shown to have very good control capabilities in indoor environments. The same architecture as in DroNet is employed along with the collected dataset in order to estimate the collision probability.

Figure 4.13: Testing environments: (a) *Outdoor 1* is a 90° curve with a dead end. This scenario is also tested with the drone flying at high altitude (5 m), as shown in Figure 4.14. (b) *Outdoor 2* is a sharp 160° curve followed by a 30 m straight path. A closer view of this environment can be seen in (f). (c) *Outdoor 3* is a series of 2 curves, each of approximately 60° , with straight paths in between. Moreover, DroNet is tested on scenarios visually different from the training ones, such as (d) an indoor parking lot, and (e) an indoor corridor.

The average distance travelled before stopping or colliding is used as metric. Results from Table 4.6 indicate that DroNet is able to drive a UAV the longest on almost all the selected testing scenarios. The main strengths of the policy learned by DroNet are twofold: (i) the platform smoothly follows the road lane while avoiding static obstacles; (ii) the drone is never driven into a collision, even in presence of dynamic obstacles, like pedestrians or bicycles, occasionally occluding its path. Another interesting feature is that DroNet usually drives the vehicle to a random direction in open spaces and at intersections. In contrast, the baseline policy of minimizing the probability of collision was very often confused by intersections and open spaces, which resulted in a shaky uncontrolled behaviour. This explains the usually large gaps in performance between the proposed methodology and the considered baselines.

Interestingly, the policy learned by DroNet generalizes well to scenarios visually different from the training ones, as shown in Table 4.6. Regarding experiments at high altitude, only a very little drop in performance was noticed when the vehicle was flying at 5 m. Even though

the drone’s viewpoint was different from a ground vehicle’s one (usually at 1.5 m), the curve could be successfully completed as long as the path was in the field of view of the camera. More surprisingly was the generalization of DroNet to indoor environments, such as a corridor or a parking lot. In these scenarios, the drone was still able to avoid static obstacles, follow paths, and stop in case of dynamic obstacles occluding its way. Nonetheless, some domain-shift problems appeared. In particular, some drifts were noticed at intersections, which were sometimes too narrow to be smoothly performed by the proposed algorithm. In contrast, as expected, the baseline policy of [7], specifically designed to work in narrow indoor spaces, outperformed the proposed method.

Policy	Urban Environment			Generalization to other Environments		
	Outdoor 1	Outdoor 2	Outdoor 3	High Altitude Outdoor 1	Corridor	Garage
Straight	23 m	20 m	28 m	23 m	5 m	18 m
Gandhi et al. [7]	38 m	42 m	75 m	18 m	31 m	23 m
DroNet	52 m	68 m	245 m	45 m	27 m	50 m

Table 4.6: Average travelled distance before stopping: navigation results using three different policies on several environments. Recall that [7] uses only collision probabilities, while DroNet uses also predicted steering angles. *High Altitude Outdoor 1* consists of the same path as *Outdoor 1*, but flying at 5 m altitude, as shown in Figure 4.14

Figure 4.14: High altitude Outdoor 1: In order to test the ability of DroNet to generalize at high altitude, the drone is flown at 5 m altitude in the testing environment *Outdoor 1*. Table 4.6 indicates that the proposed policy is able to cope with the large difference between the viewpoint of a camera mounted on a car (1.5 m) and the one of the UAV.

4.4.3.4 Qualitative Results

The behaviour of DroNet in some of the considered testing environments can be observed in Figure 4.15. Unlike previous work [141], DroNet always produced a safe and smooth flight. In

particular, the drone always reacted promptly to dangerous situations, *e.g.* sudden occlusions by bikers or pedestrians in front of it.

Figure 4.15: DroNet predictions. The above figures show predicted steering and probability of collision evaluated over several experiments. Despite the diverse scenarios and obstacles types, DroNet predictions always follow common sense and enable safe and reliable navigation.

To better understand the proposed flying policy, the technique outlined in [149] is used to visualize the activation or attention maps. Figure 4.16 shows which part of an image is the most important for DroNet to generate a steering decision. Intuitively, the network mainly concentrates on the “line-like” patterns present in a frame, which roughly indicate the steering direction. Indeed, the strong coupling between perception and control renders perception mainly sensitive to the features important for control. This explains why DroNet generalizes so well to many different indoor and outdoor scenes that contain “line-like” features. Conversely, it is expected for DroNet to fail in environments missing those kind of features, for instance in a forest, where no evident path was visible. However, placed in a forest surrounding a clearly visible path, the drone behaved better.

Figure 4.16: Activation maps: spatial support regions for steering regression in city streets, on (a) a left curve and (b) a straight path. Activations on (c) an indoor parking lot, and (d) an indoor corridor are also shown. It can be observed that the network concentrates its attention to “line-like” patterns, which approximately indicate the steering direction.

The proposed methodology comes with the advantages and limitations inherent to both traditional and learning-based approaches. The advantages are that, using the proposed simple learning and control scheme, a drone is allowed to safely explore previously unseen scenes while requiring no previous knowledge about them. More specifically, in contrast to traditional approaches, there is no need to be given a map of the environment, or build it online, pre-define collision-free waypoints and locate them within this map. An advantage with respect to other CNN-based controllers [142, 141, 140, 139, 108] is that the proposed methodology can leverage the large body of literature present on steering angle estimation [8, 146] on both the data and the algorithmic point of view. As shown in the experiments, this gives the proposed model high generalization capabilities. Indeed, its flying policy can reliably fly in non-trivial unseen scenarios without requiring any re-training or fine-tuning, as it is generally required by CNN-based approaches [108].

The limitations are primarily that the agile dynamics of drones is not fully exploited, and that it is not directly possible to explicitly give the robot a goal to be reached, as it is common in other CNN-based controllers [142, 141, 150]. There are several ways to cope with the aforementioned limitations. To exploit drone agility, 3D collision-free trajectories could be generated, as *e.g.* in [150], when high probability of collision is predicted. To generalize to goal-driven tasks, either the network could be provided with a rough estimate of the distance to the goal [151], or, if a coarse 2D map of the environment is available, exploit recent learning-based approaches developed for ground robots [152]. Moreover, to make the proposed system more robust, a measure of uncertainty could be provided, as in [153]. In such a way, the system could switch to a safety mode whenever needed.

4.5 Steering prediction from event-based imagery for self-driving cars

Event cameras, such as the Dynamic Vision Sensor (DVS) [109], are bio-inspired sensors that naturally capture the dynamics of a scene, filtering out redundant information. In contrast to traditional cameras, they do not acquire full images at a fixed frame-rate, but rather have independent pixels that output only *intensity changes* (called “events”) asynchronously at the time they

occur. Hence, the output of an event camera is not a sequence of images but a stream of asynchronous events. Event cameras have multiple advantages over traditional cameras: very high temporal resolution (microseconds), very high dynamic range (HDR) (140 dB), and low power and bandwidth requirements. Moreover, since events are generated by moving edges in the scene, event cameras are natural *motion detectors*, and automatically filter out any temporally-redundant information. Due to their principle of operation and unconventional output, event cameras represent a paradigm shift in computer vision, and so, new algorithms are needed to exploit their capabilities. Indeed, event cameras present many advantages in all tasks related to motion estimation [154, 155, 156, 157, 158].

Recently, deep learning (DL) algorithms were shown to perform well on many applications in the field of motion estimation [159, 160, 8]. In this section, a deep neural network approach that unlocks the potential of event cameras is proposed on a challenging motion-estimation task: prediction of a vehicle's steering angle. To make the best out of this sensor–algorithm combination, state-of-the-art convolutional architectures are adapted to the output of event sensors and extensively evaluated on a publicly available large scale event-camera dataset. Qualitative and quantitative results are presented to explain why event cameras allow robust steering prediction even in cases where traditional cameras fail, *e.g.* challenging illumination conditions and fast motion. The advantages of leveraging transfer learning from traditional to event-based vision are also demonstrated, even if the networks were trained on frames collected by traditional cameras. This is the first large-scale (≈ 1 million images covering over 1000 km) application of deep learning to event-based vision on a regression task, which has demonstrated to outperform state-of-the-art algorithms based on standard cameras.

As already pointed out by previous works [8], learning-based approaches will be ultimately needed to handle complex scenarios and, more importantly, corner cases in which self-driving cars will maneuver. However, the goal of the proposed approach is not to develop a framework to actually control an autonomous car or robot, as already proposed in [161]. The aim is to understand how learning-based approaches oriented to motion-estimation tasks could benefit from the natural response of event cameras to motion, their inherent data redundancy reduction, high speed, and very high dynamic range. It is shown that the ability of event cameras to capture the dynamics of a scene at low-latency combined with specifically-designed neural networks outperforms state-of-the-art systems which are based on standard cameras.

The rest of the section is organized as follows. Subsection 4.5.1 reviews related work on the problem of autonomous driving. Subsection 4.5.2 describes the proposed methodology based on deep learning. The performance of the learning process is extensively evaluated in Subsection 4.5.4, as well as the discussion of the obtained results.

4.5.1 Related work

Developing robust policies for autonomous driving is a challenging research problem. Highly engineered, modular systems demonstrated incredible performance in both urban and off-road scenarios [162]. Another approach to the problem is to directly map visual observations to control actions, tightly coupling the perception and control parts of the problem. The first attempt to learn a visuomotor policy was done with ALVINN [163], where a shallow network was used to predict

actions directly from images. Even though it only succeeded in simple scenarios, it suggested the potential of neural networks for autonomous navigation. More recently, NVIDIA used a CNN to learn a driving policy from video frames [9]. In spite of being a very simple approach, the learned controls were able to drive a car in basic scenarios. Afterwards, several research efforts have been spent to learn more robust perception-action models [8, 160] to cope with the diversity of visual appearance and unpredictability usually encountered in urban environments. The work in [8] proposed to leverage large-scale driving video datasets and do transfer learning to generate more robust policies. The model showed good performance, but was limited to only a set of discrete actions, and was susceptible to failures in undemonstrated regions of the policy space. In [160] the authors proposed a method to directly regress steering angles from frames while providing an interpretable policy. However, regarding performance, very little improvement was achieved with respect to [9].

All previous methods operate on images acquired by traditional frame-based cameras. In contrast, event cameras (asynchronous, pixel-wise brightness changes with very low latency and high dynamic range), which naturally respond to *motion* in the scene, are proposed in this case to learn policies.

The capabilities of event cameras to provide rich data for solving pattern recognition problems has been initially shown in [164, 165, 166, 167, 161]. In all these problems, machine learning algorithms were applied on data acquired by an event camera to solve classification problems, and were generally trained and tested on datasets of limited size. For example, [164, 165, 166] use neural networks on event data to recognize cards of a deck (4 classes), faces (7 classes), or characters (36 classes). A similar case is that of [167], where a network is trained to recognize three types of gestures (rocks, papers, scissors) in dynamic scenes. So far, estimation problems in which the unknown variable is continuous were tackled by discretization, i.e., the solution space was partitioned into a finite number of classes. This is the case, for example, of the predator-prey robots in [161], where a network trained on the combined input of events and grayscale frames from a Dynamic and Active-pixel Vision Sensor (DAVIS) [168] produced one of four outputs: the prey is on the left, center, or right of the predator's field of view (FOV), or it is not visible in the FOV. Another example is that of the optical flow estimation method in [169], where the network produced motion vectors from a set with 8 different directions and 8 different speeds (i.e., 64 classes).

As opposed to the classification approximation of *all* previous methods, a continuous estimation problem (steering angle prediction) from a *regression* point of view is described in this section. This is the first work that tackles continuous estimation problems with event cameras in a principled way without resorting to partitioning the solution space; the angles produced by the proposed network can take any value in the range $[-180^\circ, 180^\circ]$, not just discrete ones. Moreover, in contrast to previous event-based vision learning works which use small datasets, results on the largest and most challenging (due to scene variability) event-based dataset to date are shown.

4.5.2 System description

The proposed system aims at predicting steering wheel commands from a forward-looking DVS sensor [109] mounted on a car. As shown in Figure 4.17, a learning approach that takes as input

the visual information acquired by an event camera and outputs the vehicle's steering angle is proposed. The events are converted into *event frames* by pixel-wise accumulation over a constant time interval. Then, a deep neural network maps the event frames to steering angles by solving a regression task. In the following, the different steps of the learning process are detailed.

Figure 4.17: Block diagram of the proposed approach. The output of the event camera is collected into frames over a specified time interval T , using a separate channel depending on the event polarity (positive and negative). The resulting synchronous event frames are processed by a ResNet-inspired network, which produces a prediction of the steering angle of the vehicle.

4.5.2.1 Event-to-frame conversion

All recent and successful deep learning algorithms are designed for traditional video input data (i.e., frame-based and synchronous) to benefit from conventional processors. In order to take advantage of such techniques, asynchronous events need to be converted into synchronous frames. To do that, events $e_k = (x_k, y_k, t_k, p_k)$ are accumulated over a given time interval T in a pixel-wise manner, obtaining 2D histograms of events. An event e_k consists of the spatiotemporal coordinates (x_k, y_k, t_k) of a relative brightness change of predefined magnitude together with its polarity $p_k \in \{-1, +1\}$ (i.e., the sign of the brightness change). Since event cameras naturally respond to moving edges, these histograms of events are maps encoding the relative motion between the event camera and the scene. Additionally, due to the sensing principle of event cameras, they are free from redundancy.

Inspired by [166], separate histograms for positive and negative events are used. Then, the histogram for positive events is

$$h^+(x, y) \doteq \sum_{t_k \in T, p_k = +1} \delta(x - x_k, y - y_k), \quad (4.6)$$

where δ is the Kronecker delta, and the histogram h^- for the negative events is defined similarly, using $p_k = -1$. The histograms h^+ and h^- are stacked to produce a two-channel event image. Events of different polarity are stored in different channels, as opposed to a single channel with the balance of polarities ($h^+ - h^-$), to avoid information loss due to cancellation in case events of opposite polarity occur in the same pixel during the integration interval T .

4.5.2.2 Preprocessing

A correct normalization of input and output data is essential for reliably training any neural network. Since roads are almost always straight, the steering angle's distribution of a driving car is mainly picked between $[-5^\circ, 5^\circ]$. This unbalanced distribution results in a biased regression. In addition, vehicles frequently stand still because they are exposed, for example, to traffic lights and pedestrians. In those situations where there is no motion, only noisy events will be produced. To handle those problems, the output variable (*i.e.* steering angles) is pre-processed to allow successful learning. To cope with the first issue, only 30 % of the data corresponding to a steering angle lower than 5° is deployed at training time. For the latter, data corresponding to a vehicle's speed smaller than 20 km h^{-1} is filtered out. To remove outliers, the filtered steering angles are then trimmed at three times their standard deviation and normalized to the range $[-1, 1]$. At testing time, all data corresponding to a steering angle lower than 5° is considered, as well as scenarios under 20 km h^{-1} . The regressed steering angles are denormalized to output values in the range $[-180^\circ, 180^\circ]$. Finally, the network input (*i.e.*, event images) is scaled to the range $[0, 1]$.

4.5.2.3 Network architecture

To unlock the power of convolutional architectures, first they have to be adapted to accommodate the output of the event camera. Initially, event frames of different polarity are stacked, creating a 2D *event image*. Afterwards, a series of ResNet architectures, *i.e.*, ResNet18 and ResNet50, are deployed since they have proven to be easier to optimize as the number of layers (depth) increases, and to better cope with overfitting [69]. As these networks have been designed for image classification purposes, they are used as feature extractors for the regression problem, considering only their convolutional layers. To encode the image features extracted from the last convolutional layer into a vectorized descriptor, a global average pooling layer [170] is used to return the features' channel-wise mean. This choice has proven to improve the performance, compared to directly adding a fully-connected layer, since it minimizes overfitting by reducing the total number of parameters, and it better propagates the gradients. After the global average pooling, a fully-connected (FC) layer (256-dimensional for ResNet18 and 1024-dimensional for ResNet50) is added, followed by a ReLU non-linearity and the final one-dimensional fully-connected layer to output the predicted steering angle.

4.5.3 Experimental setup

This section presents the employed large-scale dataset to evaluate the proposed system, as well as the different types of images that are passed to the network as input.

4.5.3.1 Dataset

To predict steering angles from event images the publicly available DAVIS Driving Dataset 2017 (DDD17) [10] is used. It contains approximately 12 hours of annotated driving recordings (for a total of 432 GB) collected by a car under different and challenging weather, road, and illumination

conditions. The dataset includes asynchronous events, as well as synchronous, grayscale frames, collected concurrently by the DAVIS sensor [168]. The DAVIS camera consists of a traditional grayscale camera and an event sensor (DVS) on the same pixel array, with 346×260 pixel resolution (DDD17 dataset). Event data is produced simultaneously from the same photodiodes which give the frame-based intensity read-out. The recordings are divided into four subsets, according to the labels provided by the dataset’s authors: *day*, *day_sun*, *evening*, and *night*. Subsets differ not only in the illumination and weather conditions, but also in the route travelled. Figure 4.18 depicts some data samples.

Figure 4.18: Different input data extracted from the DDD17 dataset for the four lighting conditions. The first row depicts grayscale images collected with a traditional camera (frames of the DAVIS). The second row shows images created by difference of two consecutive grayscale images. The third row corresponds to time-integrated event images (here displayed as single-channel images, $h^+ - h^-$, to highlight the similarity with those on the second row).

Since, while driving, subsequent frames are usually very similar at driving scenarios, and have almost identical steering angles, randomly dividing the dataset into training and test subsets would result in over-optimistic estimates. Therefore, to properly test generalization ability of the learned models, the dataset is divided as follows. The recordings are split into consecutive and non-overlapping short sequences of a few seconds each, and then alternate subsets of these sequences are used for training and testing. In particular, training sequences correspond to 40 seconds of recording, while test sequences to 20 seconds. As shown in Figure 4.19, training subsets alternate with test subsets, resulting in different samples.

Figure 4.19: Maps of dataset showing (a) the whole recorded route, covering 1000 km of different roads in Switzerland and Germany; (b) and (c) training and testing frames, in red and blue respectively, for the *day_sun* subset (3 hours of driving). The training sections are different from the testing sections, corresponding to 40 and 20 seconds of recording respectively.

4.5.3.2 Performance comparison on different types of images

Steering angles are predicted using three different types of visual inputs: grayscale images, difference of grayscale images, and images created by event accumulation. The grayscale images correspond to absolute intensity frames from a traditional camera (first row of Figure 4.18) and they correspond to the typical input of state-of-the-art steering angle prediction systems. As already pointed out above, the grayscale frames coming from a DAVIS sensor allow a fair comparison with events, since, being produced by the same photodiodes, they observe exactly the same scene. The proposed methodology is also compared against temporal difference of intensity images (second row of Figure 4.18). As shown in the figure, they are similar to the event images (third row of Figure 4.18). Intensity differences incorporate temporal information, and, as it will be shown in the experiments, they provide a stronger baseline than the absolute intensity images for comparing the results of the event-data architecture. More specifically, event cameras report pixel-wise

log-brightness changes of predefined size C :

$$L(t) - L(t - \Delta t) = \pm C, \quad (4.7)$$

with $L(t) = \log I(t)$, and $I(t)$ being the intensity on the image plane. When these changes (4.7) are aggregated over some time interval, they quantify the amount of brightness change (increase or decrease) that happened at each pixel,

$$\Delta L \approx (h^+ - h^-) C. \quad (4.8)$$

For a small time interval, the difference of two consecutive grayscale frames is a first order (Taylor) approximation to such an intensity change (4.8) since

$$L(t) - L(t - \Delta t) \doteq \Delta L \approx \frac{\partial L}{\partial t} \Delta t. \quad (4.9)$$

This is why images in the second and third rows of Figure 4.18, which basically encode the temporal brightness changes over a specified time interval, look similar.

As in [8], the ground truth steering wheel angle is selected as the one at 1/3 s in the future with respect to the current frame (either event or grayscale frame).

4.5.4 Experimental results

This section shows the experiments that have been carried out to evaluate the system performance. First, the influence of the event integration time used to produce event frames is investigated (Section 4.5.4.1). Second, the advantages of using event images over grayscale or grayscale-difference as input to the network are presented (Section 4.5.4.2). Finally, the proposed system is compared to state-of-the-art methods based on traditional cameras by learning a single network over the entire dataset (Section 4.5.4.3). This section also includes a qualitative comparison between standard cameras and event cameras.

4.5.4.1 Sensitivity analysis with respect to the event integration time

In this section, the performance of the network is analyzed as a function of the integration time used to generate the input event images from the event stream. A visual comparison between the input event images for 10, 25, 50, 100, and 200 ms, is shown in Figure 4.20. These integration times were chosen to be approximately equispaced in logarithmic scale. It can be observed that the larger the integration time, the larger the trace of events appearing at the contours of objects. This is due to the fact that they moved a longer distance on the image plane during that time. It is hypothesized that the network exploits such motion cues to provide a reliable steering prediction.

Figure 4.20: Events collected for different durations of the interval T (cf. Figure 4.17). The scene corresponds to a *day_sun* sequence, with the car turning left in an urban environment.

When the integration time is small, event images are generally created out of few events only. Therefore, images associated with relatively small motion are not very discriminative and easy to learn from (Figure 4.20 (a)). Conversely, in images created with a long integration time (Figure 4.20 (e)) large motion blur washes out the contours of objects, in particular when the car motion is relatively high. Consequently, the produced images lose the discriminability necessary for a reliable estimation. This correlation between the integration time and quality of prediction can be observed in Figure 4.21.

Figure 4.21: Distribution of the relative error in steering angle prediction as a function of the ground truth steering angle and the event integration time. The performance of the network trained on large integration time (200 ms) degrades for large steering angles. In contrast, the network trained on small integration time (10 ms) predicts well large steering angles, but its performance degrades for smaller angles. The network with an intermediate integration time (50 ms) performs best for large and moderate steering angles. For small angles ($<5^\circ$), small absolute errors in the angle produce large relative errors regardless of the integration time.

Table 4.7 and Figure 4.22 report quantitative results of the proposed ResNet50 network using

the five different integration times. As can be observed, the network performs best when it is trained on event images corresponding to 50 ms, and the performance gracefully degrades for smaller and larger integration times. Therefore, in the following experiments the integration time is set to the best value, 50 ms to further analyze the performance of the proposed method.

Integration time T	EVA	RMSE
10 ms	0.790	11.53°
25 ms	0.792	10.42°
50 ms	0.805	9.47°
100 ms	0.634	13.43°
200 ms	0.457	15.87°

Table 4.7: Comparison of the ResNet50 performance for the different integration times on *day_sun* subset.

Figure 4.22: Variation of the RMSE of ResNet50 with respect to the event accumulation time (3rd column of Table 4.7).

4.5.4.2 Results on different illumination scenarios

Now fixing the integration time to 50 ms, an extensive study is conducted on the four subsets of the employed dataset (*day*, *day_sun*, *evening*, and *night*), highlighting the advantages of event frames over grayscale-based ones. To do so, a cross evaluation between architectures and types of input frames is provided in Tables 4.8 to 4.11. For fair comparison, the same convolutional network architectures are deployed as feature extractors for all considered inputs, but each network is trained independently.

It is interesting to notice that the average RMSE is slightly different among different sets. This is to be expected, since RMSE, being dependent on the absolute value of the steering ground truth, is not a good metric for cross comparison between sequences. On the other hand, EVA gives a better way to compare the quality of the learned estimator across different sequences.

The *day* subset, whose results are shown in Table 4.8, is the most difficult one of the considered partitions. It includes five hours of driving in both urban (including parking lots) and countryside scenarios. The very large variance in the input data (a.k.a. state space) made convergence difficult for the grayscale baseline. In fact, the shallower model, ResNet18, learned on it reports a quasi-constant solution ($EVA \approx 0$), therefore converging to the data average. In contrast, the proposed method, based on event images, always converged to a solution outperforming the baselines (grayscale and grayscale difference) with both architectures. Interestingly, it is observed that a very large performance gap between the grayscale difference and the event images for the ResNet18 architecture. The main reasons behind this behavior are: (i) abrupt changes in lighting conditions occasionally produced artifacts in grayscale images (and therefore also in their differences), and (ii) at high velocities, grayscale images get blurred and their difference becomes also very noisy (see, e.g., the first column in Figure 4.18). Note, however, that the ResNet50 architecture produced a significant performance improvement for both baselines (grayscale images

and difference of grayscale images). This is to be expected, since deeper architectures have more training parameters, and can therefore cope better with larger and more complicated state spaces.

A very similar pattern can be observed in the other considered scenarios. Contrary to what was expected, there was not a very large degradation of the baselines' performance when considering more challenging illumination conditions as in the *evening* and *night* sequences. However, those latter subsets are much smaller than the other two. Therefore, given the smaller state spaces, the networks have an easier job modeling the statistics of the datasets.

As can be observed in Tables 4.8 to 4.11, the event camera solution largely outperforms the baselines on all the analyzed scenarios (best results per row are highlighted in bold). In fact, the proposed methodology consistently achieves very competitive results, even with the simpler ResNet18 architecture.

Architecture	Grayscale		Grayscale diff.		Events	
	EVA	RMSE	EVA	RMSE	EVA	RMSE
ResNet18	0.047	4.57°	0.329	3.65°	0.551	2.99°
ResNet50	0.449	3.31°	0.653	2.62°	0.728	2.33°

Table 4.8: Results for *day* subset.

Architecture	Grayscale		Grayscale diff.		Events	
	EVA	RMSE	EVA	RMSE	EVA	RMSE
ResNet18	0.125	20.07°	0.729	11.53°	0.742	10.87°
ResNet50	0.383	16.85°	0.802	9.62°	0.805	9.47°

Table 4.9: Results for *day_sun* subset.

Architecture	Grayscale		Grayscale diff.		Events	
	EVA	RMSE	EVA	RMSE	EVA	RMSE
ResNet18	0.172	7.23°	0.183	7.19°	0.518	5.45°
ResNet50	0.360	6.37°	0.418	6.07°	0.602	5.01°

Table 4.10: Results for *evening* subset.

Architecture	Grayscale		Grayscale diff.		Events	
	EVA	RMSE	EVA	RMSE	EVA	RMSE
ResNet18	0.181	6.96°	0.449	5.73°	0.654	4.51°
ResNet50	0.418	5.88°	0.621	4.73°	0.753	3.82°

Table 4.11: Results for *night* subset.

4.5.4.3 Results on the entire dataset

To evaluate the ability of the proposed methodology to cope with large variations in illumination, driving and weather conditions, a single regressor was trained over the entire dataset. The proposed approach was compared to state-of-the-art architectures that use traditional frames as input: [9] and the CNN-LSTM architecture advocated in [8], but without the additional segmentation loss that is not available in the dataset. In this evaluation, [160] is not considered since, in spite of offering an interpretable solution, it gives almost no improvements over the simpler architecture in [9].

Table 4.12 summarizes the findings of these experiments. In terms of EVA and RMSE, the first work in the table obtains a poor performance on the regression task. Indeed, the EVA is very small (0.161) and the RMSE is very close to the dominant steering deviation in the dataset ($\pm 10^\circ$). To provide a stronger baseline, temporal information was incorporated to the grayscale frames by using a CNN-LSTM architecture (resulting EVA ≈ 0.3). This architecture was chosen because it has been reported to provide very competitive results in the evaluation of [8]. This is a more fair comparison because event images inherently contain temporal information.

Architecture	EVA	RMSE	Input
Bojarski <i>et al.</i> [9]	0.161	9.02°	Grayscale
CNN-LSTM [8]	0.300	8.19°	Grayscale
ResNet18	0.783	4.58°	Events
ResNet50 (ImageNet init)	0.826	4.10°	Events
ResNet50 (Random init)	0.800	4.40°	Events

Table 4.12: Comparison between two state-of-the-art learning approaches using grayscale frames [8, 9] and the proposed networks that process event frames, for the whole dataset [10]. For ResNet50, both random and ImageNet initializations have been evaluated.

All the proposed architectures based on event images largely outperform the considered baselines based on traditional frames. As it could be reasonably expected, the best results are obtained with the deepest architectures (ResNet50). More interestingly, some benefits were noticed when initializing the ResNet50 weights with those learned on the ImageNet challenge [51]. To reuse the weights, the filters of the first convolutional layer along the channel dimension were averaged and duplicated to convert them from 3-channel to 2-channel inputs. Even though it is well known that feature learning is generally transferable for different tasks [171], it is still remarkable that parameters learned on traditional RGB images have a positive transfer on time-integrated event images. As pointed out in [171], the main reason behind this is that the first convolutional weights of a network trained on ImageNet are sensitive to low-level features present in the image (e.g., edges), which are present on both traditional and event frames. Leveraging transfer learning from the immense ImageNet classification dataset not only makes training easier and faster (Figure 4.23), but also produces a better estimator (Table 4.12).

Figure 4.23: Comparison of training losses for ResNet50 with and without ImageNet initialization.

4.5.4.4 Qualitative comparison between event and standard cameras

A great deal of why a network produces better results on event images than on grayscale frames (or their difference) is their ability to capture scene dynamics. At high velocities, grayscale frames suffer from motion blur (e.g., side-road trees on the first column of Figure 4.18), whereas event images preserve edge details due to the very high temporal resolution (microsecond) of event cameras. In addition, positive and negative events are acquired in separate channels that are fed to the network, which avoids loss of information. The temporal aggregation needed to feed the network does, however, affect latency. Additionally, event cameras have a very high dynamic range (HDR) (140 dB compared to the 55 dB range of the grayscale frames in the dataset [10]). Hence, event data represent HDR content of the scene, which is not possible in traditional cameras since that would require long exposure times. This is beneficial in order to be robust to different illumination conditions (bright day, dark night, abrupt transitions in tunnels, etc.). Additionally, since event cameras respond to moving edges and therefore filter out temporally-redundant data, they are more informative about vehicle motion than individual grayscale frames. As shown qualitatively in Figure 4.24, and more quantitatively in Figure 4.21, focusing on moving edges facilitates solving the learning problem. Selecting a good integration time for creating event images additionally would also improve performance.

Figure 4.24: Steering angle regression performance on grayscale frames (first column) and on event camera data (second column). The first row shows a sunny day where visual features can be extracted from grayscale frames. However the camera saturation and the lack of temporal information makes the network predict a wrong steering angle. The second row shows a night scene, from which the network hardly predicts the correct steering angle when using grayscale frames. The proposed method accurately predicts a steering angle by combining event data and deep learning in both scenarios.

4.6 Conclusions

In this chapter, several end-to-end DL approaches have been designed for solving challenging computer vision problems, such as vehicle detection, hand-gesture recognition, and steering prediction for autonomous robot navigation.

A DL approach has been designed for vehicle detection from an image-segmentation viewpoint. The network architecture consists of a FCN, simpler than state-of-the-art image-segmentation networks, to provide a pixel-wise segmentation of the input image in two categories: vehicles and background. The proposed approach is able to coarsely segment vehicles, and thus locate them, even without dense-pixel label annotations for training. Experiment results on a traffic monitoring environment report F1-score results above 87% in terms of detection.

A CNN has been designed for recognizing static hand gestures without relying on previous detection or segmentation stages. The proposed architecture is an image-classification network that infers the semantic meaning of a gesture, even when it is embedded in highly cluttered backgrounds, and occupies a small region of the image. In such environments, the proposed CNN

outperforms other networks in the state of the art, achieving a recognition rate of 85% on a hand-gesture dataset with more than 260000 images, which has been collected for this task.

A DL approach has been developed for safely and autonomously flying a drone in urban environments. A relatively shallow CNN, which is called DroNet, has been designed for a real-time performance. It predicts the steering angle and collision probability from images captured by the on-board camera, which are used to produce control commands. Due to the difficult task of collecting data in such environments without endangering other vehicles and pedestrians, the proposed algorithm learns how to drive by imitating the behavior of manned vehicles such as cars, and bicycles. Experiments have shown DroNet is able to smoothly fly a drone while avoiding obstacles (even dynamic obstacles) in the streets of a city, and generalize to indoor environments, such as corridors and parking lots.

Finally, an event camera has been combined along with a DL approach for steering regression in a self-driving car application. Event cameras have several advantages over standard cameras, and naturally respond to motion in the scene, so they are a better option for the steering regression task. State-of-the-art CNNs have been adapted to the output of an event camera, since it makes the learning process easier. This sensor-algorithm combination has proven to work in a wide range of scenarios, and outperform other approaches based on standard cameras on a large-scale dataset. In addition, experiments have shown that it is possible to leverage transfer learning from pre-trained CNNs on RGB frames for classification tasks.

Chapter 5

Conclusions and future work

5.1 Conclusions

The success of computer vision techniques in real-life problems has consisted, for decades, of designing expert-knowledge systems based on multiple stages and feature engineering. The design of these systems mainly lies in how to solve the problem of separating the relevant information from the irrelevant one, while addressing several other vision challenges, such as changes in illumination and viewpoint, occlusions, dynamic scenes, blurring, and user-dependent behaviors. In this sense, hand-crafted features have played an important role since they have been able to provide reliable representations of the patterns to be located/recognized by exploiting their most relevant characteristics, achieving high performances with just shallow classifiers. However, since hand-crafted descriptors are designed to recognize specific patterns, they have to be applied to the regions of interest to maximize their discriminative power, and increase the learning accuracy. This forces traditional systems to use additional pre-processing steps. For instance, in the case of hand-gesture recognition, multiple stages are needed to isolate hands (signal) from background (clutter), which not only introduce latency, but also errors that propagate through the processing pipeline. Moreover, hand-crafted features are limited in generalization ability since they are not able to face all the possible conditions under which a machine must learn.

Conversely, DL brings a different paradigm where features are automatically learned, allowing researchers to build end-to-end systems that directly map pixels into the desired output. In particular, CNNs have revolutionized the computer vision community due to their efficient ability to deal with images. Just as hand-crafted features, they exploit the structure of an image region by localizing local features, and building a semantic representation from them. The large complexity of these neural networks makes it possible to build more abstract data representations that achieve higher performance and generalization power than traditional approaches. In addition, the searching mechanism is implicit in the nature of convolutional layers, which provides a way to detect and recognize any pattern embedded in clutter without requiring previous detection and segmentation stages. Although the required time to train these models is longer than for traditional systems, it is still shorter than the time required to design hand-crafted features. Moreover, the specialized hardware to run such deep architectures in a reasonable time, and the large amount of data needed

to train them, are not drawbacks anymore.

This thesis has contributed to both learning paradigms, multi-stage traditional learning and end-to-end deep learning, for different computer vision applications.

In the side of traditional learning, two novel LBP-based spatio-temporal descriptors have been designed for improving the performance of a hand-gesture recognition system, while providing a manageable computational cost. These descriptors are invariant against monotonic illumination changes, slight image translations, and variant against rotations. They present richer spatio-temporal information and higher discriminative ability than other methods designed to characterize hand gestures.

An extensive evaluation of the descriptors' performance has been carried out in both spatial and temporal domains on publicly available datasets. The proposed spatial descriptors outperform template-matching methods in the state of the art, and slightly improve others based on classifiers. Regarding temporal descriptors, they achieve recognition rates of 99% against the 77% of the state of the art. Therefore, the proposed features are suitable and reliable for describing hand gestures under controlled environments. However, later experiments have shown that the performance of such features decreases in very different unseen scenarios, since they are not able to generalize as well as in the environments where they have been trained. The lack of high quality and publicly available hand-gesture datasets, and the huge efforts to collect and annotate a new dataset from scratch, make difficult to address this problem. In addition, considering a larger amount of data for training would require memory requirements that traditional machine learning algorithms are not able to deal with efficiently.

An expert-knowledge hand-gesture recognition system has also been proposed for imitating a mouse-like pointing device using a standard camera. To that end, a hand-gesture dataset has been created and released, which contains a set of gestures associated with different functionalities of a mouse device, such as the moving cursor, the left click and the right click. The system comprises detection, tracking and recognition stages to spatially and temporally segment the performed gestures, as well as classify them. The detection and recognition stages carry out classification tasks relying on the previous descriptors along with shallow classifiers. The system achieves a high performance on the proposed dataset, separating hands from background in the spatial domain, and locating gestures in the temporal domain. However, it is far from running in real time due to the heavy computational cost of the detection stage.

Inspired by the issues found in the previous hand-gesture recognition approach, DL end-to-end systems have been proposed to solve previous problems and limitations. First, the object detection and recognition task has been addressed in two different ways for vehicle detection and hand-gesture recognition. Then, the established knowledge about DL has been applied to solve more complex problems such as autonomous robot navigation.

A structured CNN has been designed for vehicle detection without relying on multi-scale sliding windows or region proposal methods. The proposed CNN provides the location of vehicles through image segmentation, whose architecture is simpler than other state-of-the-art image-segmentation networks since it does not include sub-sampling nor up-sampling layers. It learns local features instead of global abstract ones to carry out a pixel-wise classification, even without dense pixel-label annotations. It has been evaluated on a publicly available dataset for traffic

monitoring in highways, achieving a performance around 90% in terms of F1-score.

A deep learning approach has been designed for recognizing static hand gestures from a classification viewpoint, not requiring previous segmentation or detection stages. To that end, an image-classification CNN has been proposed, and a new hand-gesture dataset has been created and released. Data collection in this case has been effortless since only the category/class of the gestures is needed, making possible to collect around 260.000 images. The dataset comprises two kind of scenarios, a simple one with uniform background and users close to the camera, and a more complex one where the hand occupies a small region of the image, and it is embedded in clutter. The proposed CNN has been evaluated on the collected dataset, achieving above 85% recognition rate in complex scenarios, and outperforming other image-classification networks of the state of the art.

A deep learning approach has been designed for safely and efficiently flying a drone in urban environments while avoiding obstacles. Facing the problem of collecting data from an urban scenario directly with a drone, without endangering other vehicles and pedestrians, a transfer-learning strategy has been adopted. A CNN called DroNet has been designed to learn how to fly a drone by imitating the behavior of manned vehicles, such as cars and bicycles. DroNet predicts the steering angle and the collision probability directly from images captured with the on-board camera, which is coupled with the control module to command the drone. Although trained on urban environments, DroNet has proven a high generalization power, being capable of successfully flying at high altitudes, and through indoor environments, such as corridors and parking lots.

Finally, a deep learning approach has been designed for unlocking the potential of event cameras for self-driving cars. In addition to the advantages over standard cameras (low latency, high temporal resolution, and HDR), the natural response to motion of event cameras perfectly fits the steering regression task, and make the learning process easier. Then, a CNN has been adapted to the asynchronous output of the event camera to predict a vehicle's steering angle under a wide range of conditions. In addition, it has been proven that it is possible to leverage transfer learning from RGB images to event frames. This sensor-algorithm combination has been extensively evaluated on a publicly available large-scale dataset, outperforming state of the art.

5.2 Future work

Several lines of future work have been identified after analyzing the obtained results. Some of them are focused on improving the overall performance of the proposed systems, and others on new approaches to solve different tasks:

- All the proposed systems present a temporal aspect that has been mostly solved frame-by-frame using a CNN. Therefore, the next step would be to exploit the temporal domain of such tasks by employing recurrent neural networks (RNN). By concatenating a CNN and a RNN such as LSTM [63], two approaches can be carried out: (i) both architectures can be simultaneously trained from scratch to directly map a set of temporally ordered individual frames into the desired output, and (ii) it is possible to take advantage of the already computed convolutional weights from the CNN-based solution to act like a fixed

feature extractor that feeds an LSTM to make a decision along time.

- Complex tasks such as autonomous navigation for flying and ground robots have been addressed as supervised learning approaches because it results in a more viable way to learn effective policies. Since Reinforcement Learning (RL) techniques [172] involve a trial-and-error component, they hinder their application to robots that operate in safety-critical environments. However, for the self-driving application based on event imagery, it would be interesting to use RL to produce an adaptive integration time policy that depends on the car's speed and the observed scene.
- Object detection networks have proven to localize and categorize multiple objects anywhere in the image. However, they do not seem like the most appropriate approach for a CNN that must detect patterns in pre-defined regions of the image. Object detection networks can cause false positives and negatives, so that the resulting set of predicted regions would not be the same as the one under study. One solution proposed in the literature is to run a CNN over every pre-defined region [173], which implies a high computational cost. The future work that is proposed here is a multi-output CNN that processes the whole image at once, where each output is focused on a single region to predict the presence of a pattern within it.

Appendix A

Local Binary Patterns (LBP)

Local Binary Patterns (LBP) has become a popular descriptor for several tasks, such as texture classification [103], face recognition [104], video background subtraction [174], and motion analysis [29]. The LBP operator was originally designed for texture description [24]. As can be observed in Figure A.1, this operator thresholds the 3×3 neighborhood of each pixel in the image region by the intensity value of the center pixel, and takes the sign of the differences between the values of the neighbors and the value of the center pixel. The thresholded values are concatenated in an 8-bit binary number, starting at the upper left corner, from left to right and from top to bottom, which represents the texture pattern of the considered neighborhood. This binary number could be also ordered in a different way. Next, all the extracted binary numbers are converted to decimal numbers, which represent texture labels. Finally, they are used to generate a histogram of $2^8 = 256$ labels, which is used as a descriptor of the image region.

Figure A.1: Local binary pattern from a pixel neighborhood: (a) 3×3 gray scale neighborhood. (b) Differences between the center pixel and its neighbors. (c) Thresholded neighborhood. The binary pattern is converted to decimal and represents one of the histogram's labels. (d) Histogram of LBP from the whole image.

The LBP operator is invariant to monotonic gray-scale changes because of the fact that it only considers the signs of the differences. In the presence of illumination variations, the pixel values can dramatically change, but their intensity relationship rarely changes. Therefore, the sign is more reliable than the magnitude (absolute value) of the difference. This is the most important attribute that has popularized the LBP operator. An additional advantage is the relatively short

size of the resultant feature vector, which makes it compatible with real-time classification. Hence, LBP achieves an excellent trade-off between efficiency and computational cost. In addition, it has proven to be highly discriminative for the aforementioned tasks.

On the other hand, the LBP operator does not provide global spatial information because it does not identify from what part of the image the binary patterns come. The generated histogram takes only into account the pattern occurrences discarding the spatial information. This can be a disadvantage for some applications that could use the spatial information to be more discriminative.

A.1 Multi-scale Local Binary Patterns

In order to deal with textures at different scales, the LBP operator was extended to use neighborhoods of different sizes [6]. The new neighborhood pattern is defined as a set of sampling points evenly spaced on a circle centered at the pixel to be labeled. This new version of LBP makes it possible to consider any radius and number of sampling points, which provides a way for a multiresolution processing that expands its applicability. In addition, it keeps all properties previously described for the LBP operator.

Figure A.2: Circularly symmetric neighbor sets for different P and R (extracted from [6]).

The notation for defining this new operator is $LBP_{P,R}$, where P means number of sampling points on a circle of radius R . The mathematical expression to obtain a label from $LBP_{P,R}$ is:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c) 2^p, \quad (A.1)$$

where g_c corresponds to the gray value of the center pixel of the local neighborhood, g_p ($p = 0, \dots, P-1$) corresponds to the gray values of the P equally spaced sampling points on the circular neighborhood, and $s(x)$ is the sign function defined as:

$$s(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0. \end{cases} \quad (A.2)$$

The coordinates of g_p can be expressed in polar coordinates as follows:

$$(x_p, y_p) = (R \cos(2\pi p/P), -R \sin(2\pi p/P)). \quad (A.3)$$

Figure A.2 shows circularly symmetric neighbor sets for different number of sampling points and radius. Bilinear interpolation is used when a sampling point does not fall in the center of a pixel.

A.2 Volume Local Binary Patterns (VLBP)

In order to extend texture analysis from spatial domain to spatio-temporal domain, Volume Local Binary Patterns (VLBP) and Local Binary Patterns from Three Orthogonal Planes (LBP-TOP) were proposed [29]. These extensions of the LBP operator are used to describe dynamic textures, so that they are able to combine appearance features with motion features.

The idea behind VLBP is the same as the $LBP_{P,R}$ operator, except that it is extended to the previous and posterior neighboring frames, as well. Given a pixel belonging to a specific frame, its local volume neighborhood is formed by its P spatial sampling points in the same frame (as in $LBP_{P,R}$), the center pixel in some previous frame, and its P sampling points, and the center pixel in the posterior frame and its P sampling points, obtaining a total of $3P + 2$ neighbors. Thus, VLBP descriptor uses three parallel planes, from which only the middle one contains the center pixel. Once the gray value of the center pixel is subtracted from the gray values of the circularly symmetric neighborhood, and the sign of these differences is considered, a dynamic texture is obtained as:

$$\begin{aligned} v = (v_0, v_1, \dots, v_{3P+1}) = & (s(g_{t_c-L,c} - g_{t_c,c}), s(g_{t_c-L,0} - g_{t_c,c}), \dots, \\ & s(g_{t_c-L,P-1} - g_{t_c,c}), s(g_{t_c,0} - g_{t_c,c}), \dots, \\ & s(g_{t_c,P-1} - g_{t_c,c}), s(g_{t_c+L,0} - g_{t_c,c}), \dots, \\ & s(g_{t_c+L,P-1} - g_{t_c,c}), s(g_{t_c+L,c} - g_{t_c,c})), \end{aligned} \quad (A.4)$$

where $s(x)$ is the same function as the one used in (A.2), $g_{t_c,c}$ corresponds to the gray value of the center pixel of the local volume neighborhood, $g_{t_c-L,c}$ and $g_{t_c+L,c}$ corresponds to the gray value of the center pixel in the previous and posterior neighboring frames with time interval L , and $g_{t,p}(t = t_c - L, t_c, t_c + L; p = 0, \dots, P - 1)$ correspond to the gray values of P neighbors in frame t . Finally, v is transformed into a unique $VLBP_{L,P,R}$ number that represents the spatial structure of the local volume dynamic texture:

$$VLBP_{L,P,R} = \sum_{q=0}^{3P+1} v_q 2^q. \quad (A.5)$$

Figure A.3 (a) shows how to extract a volume local binary pattern. The final histogram has 2^{3P+2} labels, where P is the parameter that determines the number of features. If P is large, the histogram will be very long, and this will restrict its applicability. However, if P is small, the feature vector loses more information, but instead it is more compact.

A.3 Local Binary Patterns from Three Orthogonal Planes (LBP-TOP)

To solve the dimensionality problems of the VLBP, the LBP-TOP descriptor proposes to concatenate LBP histograms from three orthogonal planes: XY, XT, and YT, as shown in Figure A.3 (b). The XY plane represents appearance information, while the XT plane gives a visual impression of rows changing in time, and the YT plane describes the motion of columns in temporal space. Thus, LBP-TOP uses three orthogonal planes that intersect in the center pixel.

To extract a LBP histogram from each plane, a set of parameters has to be fixed. These parameters are the number of sampling points in the circular neighborhood in each plane P_{XY}, P_{XT} , and P_{YT} , and the radius of the circular neighborhood in each plane R_X, R_Y , and R_T . Once the three histograms are computed, they are concatenated to form the feature vector. Therefore, the resultant descriptor, denoted by $\text{LBP-TOP}_{P_{XY}, P_{XT}, P_{YT}, R_X, R_Y, R_T}$, will contain $2^{P_X} + 2^{P_Y} + 2^{P_T}$ labels. If $P_X = P_Y = P_T = P$, we can see that the LBP-TOP descriptor will have $3 \cdot 2^P$ labels, instead of $2^{3 \cdot P + 2}$ ones that had the VLBP descriptor. Notice that the dimension of the feature vector is considerably reduced. Regarding the radius in the time axis, it depends on the frame rate of the analysed sequence, as in VLBP descriptor.

Figure A.3: Temporal extensions of LBP: (a) procedure to compute $\text{VLBP}_{1,4,1}$, and (b) procedure to compute $\text{LBP-TOP}_{4,4,4,1,1,1}$.

Bibliography

- [1] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, “Selective search for object recognition,” *Int. J. Comput. Vis.*, vol. 104, pp. 154–171, Sep. 2013.
- [2] C. Olah, “Understanding LSTM networks.” <http://colah.github.io/posts/2015-08-Understanding-LSTMs>, 2015. [Online; accessed: 2018-06-13].
- [3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 779–788, Jun. 2016.
- [4] H. Noh, S. Hong, and B. Han, “Learning deconvolution network for semantic segmentation,” in *IEEE International Conference on Computer Vision*, pp. 1520–1528, 2015.
- [5] R. Guerrero-Gomez-Olmedo, R. J. Lopez-Sastre, S. Maldonado-Bascon, and A. Fernandez-Caballero, “Vehicle tracking by simultaneous detection and viewpoint estimation,” in *Natural and Artificial Computation in Engineering and Medical Applications*, pp. 306–316, Jun. 2013.
- [6] T. Ojala, M. Pietikainen, and T. Maenpaa, “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 24, pp. 971–987, Jul. 2002.
- [7] D. Gandhi, L. Pinto, and A. Gupta, “Learning to fly by crashing,” in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, pp. 3948–3955, Sep. 2017.
- [8] H. Xu, Y. Gao, F. Yu, and T. Darrell, “End-to-end learning of driving models from large-scale video datasets,” in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 3530–3538, Jul. 2017.
- [9] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, “End to end learning for self-driving cars,” *arxiv:1604.07316*, Apr. 2016.
- [10] J. Binas, D. Neil, S.-C. Liu, and T. Delbruck, “DDD17: End-to-end DAVIS driving dataset,” in *ICML Workshop on Machine Learning for Autonomous Vehicles*, Sep. 2017.
- [11] A. I. Maqueda, C. R. del Blanco, F. Jaureguizar, and N. García, “Human–computer interaction based on visual hand-gesture recognition using volumetric spatiograms of local binary patterns,” *Comput. Vis. Image. Und.*, vol. 141, pp. 126 – 137, Jan. 2015.

-
- [12] A. I. Maqueda, C. R. del Blanco, F. Jaureguizar, and N. García, “Temporal pyramid matching of local binary subpatterns for hand-gesture recognition,” *IEEE Sig. Proc. Letters*, vol. 23, pp. 1037–1041, Aug. 2016.
 - [13] A. I. Maqueda, C. R. del Blanco, F. Jaureguizar, and N. García, “Structured learning via convolutional neural networks for vehicle detection,” *SPIE Int. Conf. Real-Time Image and Video Processing*, vol. 10223, pp. 10223 – 10223 – 9, Apr. 2017.
 - [14] P. Bao, A. I. Maqueda, C. R. del Blanco, and N. García, “Tiny hand gesture recognition without localization via a deep convolutional network,” *IEEE Trans. Cons. Elect.*, vol. 63, pp. 251–257, Aug. 2017.
 - [15] A. Loquercio, A. I. Maqueda, C. R. del Blanco, and D. Scaramuzza, “Dronet: Learning to fly by driving,” *IEEE Robot. Autom. Lett.*, vol. 3, pp. 1088–1095, Apr. 2018.
 - [16] A. I. Maqueda, A. Loquercio, G. Gallego, N. García, and D. Scaramuzza, “Event-based vision meets deep learning on steering prediction for self-driving cars,” in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, Jun. 2018.
 - [17] M. M. Cheng, Z. Zhang, W. Y. Lin, and P. Torr, “BING: Binarized normed gradients for objectness estimation at 300fps,” in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 3286–3293, Jun. 2014.
 - [18] C. L. Zitnick and P. Dollár, “Edge boxes: Locating object proposals from edges,” in *Eur. Conf. Comput. Vis. (ECCV)*, pp. 391–405, Sep. 2014.
 - [19] B. Alexe, T. Deselaers, and V. Ferrari, “Measuring the objectness of image windows,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 34, pp. 2189–2202, Nov. 2012.
 - [20] J. Carreira and C. Sminchisescu, “CPMC: Automatic object segmentation using constrained parametric min-cuts,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 34, pp. 1312–1328, Jul. 2012.
 - [21] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *Int. J. Comput. Vis.*, vol. 60, pp. 91–110, Nov. 2004.
 - [22] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, “Speeded-up robust features (surf),” *Comput. Vis. Image. Und.*, vol. 110, pp. 346 – 359, Jun. 2008.
 - [23] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, vol. 1, pp. 886–893, Jun. 2005.
 - [24] T. Ojala, M. Pietikäinen, and D. Harwood, “A comparative study of texture measures with classification based on featured distributions,” *Pattern recognition*, vol. 29, pp. 51–59, Jan. 1996.
 - [25] G. Zhao, T. Ahonen, J. Matas, and M. Pietikainen, “Rotation-invariant image and video description with local binary pattern features,” *IEEE Trans. Image Processing*, vol. 21, pp. 1465–1477, Apr. 2012.

- [26] I. Laptev and T. Lindeberg, "Space-time interest points," in *Int. Conf. Comput. Vis. (ICCV)*, vol. 1, pp. 432–439, Oct. 2003.
- [27] P. Scovanner, S. Ali, and M. Shah, "A 3-dimensional SIFT descriptor and its application to action recognition," in *ACM Inter. Conf. on Multimedia*, pp. 357–360, Sep. 2007.
- [28] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, and C. Schmid, "Evaluation of local spatio-temporal features for action recognition," in *British Machine Vis. Conf. (BMVC)*, pp. 1–11, Sep. 2009.
- [29] G. Zhao and M. Pietikainen, "Dynamic texture recognition using local binary patterns with an application to facial expressions," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 29, pp. 915–928, Jun. 2007.
- [30] A. Andriyenko and K. Schindler, "Multi-target tracking by continuous energy minimization," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 1265–1272, Jun. 2011.
- [31] C. Jia, Z. Wang, X. Wu, B. Cai, Z. Huang, G. Wang, T. Zhang, and D. Tong, "A tracking-learning-detection (TLD) method with local binary pattern improved," in *IEEE Inter. Conf. on Robotics and Biomimetics (ROBIO)*, pp. 1625–1630, Dec. 2015.
- [32] C. H. Kuo, C. Huang, and R. Nevatia, "Multi-target tracking by on-line learned discriminative appearance models," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 685–692, Jun. 2010.
- [33] W. Hu, W. Li, X. Zhang, and S. Maybank, "Single and multiple object tracking using a multi-feature joint sparse representation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 37, pp. 816–833, Apr. 2015.
- [34] J. H. Yoon, M. H. Yang, J. Lim, and K. J. Yoon, "Bayesian multi-object tracking using motion context from multiple objects," in *Conf. on Applications of Computer Vision*, pp. 33–40, Jan. 2015.
- [35] K. C. A. Kumar and C. D. Vleeschouwer, "Discriminative label propagation for multi-object tracking with sporadic appearance features," in *Int. Conf. Comput. Vis. (ICCV)*, pp. 2000–2007, Dec. 2013.
- [36] X. Li, K. Wang, W. Wang, and Y. Li, "A multiple object tracking method using kalman filter," in *IEEE Inter. Conf. on Information and Automation*, pp. 1862–1866, Jun. 2010.
- [37] M. Azari, A. Seyfi, and A. H. Rezaie, "Real time multiple object tracking and occlusion reasoning using adaptive kalman filters," in *Conf. on Machine Vision and Image Processing(MVIP)*, pp. 1–5, Nov. 2011.
- [38] K. Okuma, A. Taleghani, N. de Freitas, J. J. Little, and D. G. Lowe, "A boosted particle filter: Multitarget detection and tracking," in *Eur. Conf. Comput. Vis. (ECCV)*, pp. 28–39, May 2004.
- [39] C. Yang, R. Duraiswami, and L. Davis, "Fast multiple object tracking via a hierarchical particle filter," in *Int. Conf. Comput. Vis. (ICCV)*, vol. 1, pp. 212–219, Oct. 2005.

-
- [40] B. Wu and R. Nevatia, "Detection and tracking of multiple, partially occluded humans by bayesian combination of edgelet based part detectors," *Int. J. Comput. Vis.*, vol. 75, pp. 247–266, Nov. 2007.
- [41] D. Riahi and G. A. Bilodeau, "Multiple object tracking based on sparse generative appearance modeling," in *IEEE Int. Conf. Image Processing*, pp. 4017–4021, Sep. 2015.
- [42] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, Oct. 2001.
- [43] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, pp. 121–167, Jun. 1998.
- [44] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Int. Conf. on Machine Learning*, pp. 148–156, Jul. 1996.
- [45] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.
- [46] Y. Yin and R. Davis, "Real-time continuous gesture recognition for natural human-computer interaction," in *IEEE Symp. on Visual Languages and Human-Centric Computing*, pp. 113–120, Jul. 2014.
- [47] Y. Tian, R. Sukthankar, and M. Shah, "Spatiotemporal deformable part models for action detection," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 2642–2649, Jun. 2013.
- [48] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes," in *Int. Conf. on Neural Information Processing Systems: Natural and Synthetic*, pp. 841–848, Dec. 2001.
- [49] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov. 1998.
- [50] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," *Int. Conf. on Neural Information Processing Systems*, vol. 1, pp. 1097–1105, Dec. 2012.
- [51] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and F.-F. Li, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, pp. 211–252, Apr. 2015.
- [52] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Int. Conf. on Artificial Neural Networks (ICANN)*, pp. 92–101, Sep. 2010.
- [53] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.
- [54] L. Bottou, "On-line learning and stochastic approximations," vol. 17, pp. 9–42, May 1998.
- [55] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Int. Conf. on Computational Statistics*, pp. 177–186, Sep. 2010.

- [56] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *International Conference on Learning Representations*, May 2015.
- [57] S. N. Hinton, G. and K. Swersky, “Lecture 6d - a separate, adaptive learning rate for each connection,” *Slides of Lecture Neural Networks for Machine Learning*, 2012.
- [58] M. D. Zeiler, “ADADELTA: an adaptive learning rate method,” *CoRR*, vol. abs/1212.5701, 2012.
- [59] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *arXiv:1207.0580*, Jul. 2012.
- [60] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, Jun. 2014.
- [61] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, pp. 157–166, Mar. 1994.
- [62] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *Int. Conf. on Machine Learning*, pp. 1310–1318, Jun. 2013.
- [63] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, pp. 1735–1780, Feb. 1997.
- [64] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [65] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 1–9, Jun. 2015.
- [66] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” *arXiv:1512.00567*, Dec. 2015.
- [67] C. Szegedy, S. Ioffe, and V. Vanhoucke, “Inception-v4, inception-resnet and the impact of residual connections on learning,” *arXiv:1602.07261*, Feb. 2016.
- [68] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv:1409.1556*, Sep. 2014.
- [69] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 770–778, Jun. 2016.
- [70] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” *CoRR*, vol. abs/1603.05027, 2016.
- [71] R. Girshick, “Fast R-CNN,” in *Int. Conf. Comput. Vis. (ICCV)*, pp. 1440–1448, Dec. 2015.

-
- [72] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in *Advances in neural information processing systems*, pp. 91–99, 2015.
- [73] J. Dai, Y. Li, K. He, and J. Sun, “R-FCN: Object detection via region-based fully convolutional networks,” in *Advances in neural information processing systems*, pp. 379–387, Dec. 2016.
- [74] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “SSD: single shot multibox detector,” in *Eur. Conf. Comput. Vis. (ECCV)*, pp. 21–37, Oct. 2016.
- [75] J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 6517–6525, Jul. 2017.
- [76] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (VOC) challenge,” *Int. J. Comput. Vis.*, vol. 88, pp. 303–338, Jun. 2010.
- [77] 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,” in *Eur. Conf. Comput. Vis. (ECCV)*, pp. 740–755, Sep. 2014.
- [78] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results.” <http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html>, 2012. [Online; accessed: 2018-06-13].
- [79] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 3431–3440, Jun. 2015.
- [80] V. Badrinarayanan, A. Kendall, and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 39, pp. 2481–2495, Dec. 2017.
- [81] C. Peng, X. Zhang, G. Yu, G. Luo, and J. Sun, “Large kernel matters – improve semantic segmentation by global convolutional network,” in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 1743–1751, Jul. 2017.
- [82] F. Yu and V. Koltun, “Multi-scale context aggregation by dilated convolutions,” in *International Conference on Learning Representations*, May 2016.
- [83] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 40, pp. 834–848, Apr. 2018.
- [84] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” in *Int. Conf. Comput. Vis. (ICCV)*, pp. 2980–2988, Oct. 2017.
- [85] C. Ma, J. B. Huang, X. Yang, and M. H. Yang, “Hierarchical convolutional features for visual tracking,” in *Int. Conf. Comput. Vis. (ICCV)*, pp. 3074–3082, Dec. 2015.

- [86] S. Hong, T. You, S. Kwak, and B. Han, "Online tracking by learning discriminative saliency map with convolutional neural network," in *Int. Conf. on Machine Learning*, pp. 597–606, Jul. 2015.
- [87] H. Li, Y. Li, and F. Porikli, "Robust online visual tracking with a single convolutional neural network," in *Asian Conf. on Computer Vision*, pp. 194–209, Apr. 2015.
- [88] L. Wang, W. Ouyang, X. Wang, and H. Lu, "STCT: Sequentially training convolutional networks for visual tracking," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 1373–1381, Jun. 2016.
- [89] L. Wang, W. Ouyang, X. Wang, and H. Lu, "Visual tracking with fully convolutional networks," in *Int. Conf. Comput. Vis. (ICCV)*, pp. 3119–3127, Dec. 2015.
- [90] R. Tao, E. Gavves, and A. W. M. Smeulders, "Siamese instance search for tracking," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 1420–1429, Jun. 2016.
- [91] Q. Guo, W. Feng, C. Zhou, R. Huang, L. Wan, and S. Wang, "Learning dynamic siamese network for visual object tracking," in *Int. Conf. Comput. Vis. (ICCV)*, pp. 1781–1789, Oct. 2017.
- [92] T. Yang and A. B. Chan, "Recurrent filter learning for visual tracking," in *Int. Conf. Comput. Vis. Workshops (ICCVW)*, pp. 2010–2019, Oct. 2017.
- [93] H. Harzallah, F. Jurie, and C. Schmid, "Combining efficient object localization and image classification," in *Int. Conf. Comput. Vis. (ICCV)*, pp. 237–244, Sep. 2009.
- [94] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 32, pp. 1627–1645, Sep. 2010.
- [95] J. Lin, Y. Wu, and T. S. Huang, "Modeling the constraints of human hand motion," in *Workshop on Human Motion*, pp. 121–126, Dec. 2000.
- [96] Z. Ren, J. Yuan, J. Meng, and Z. Zhang, "Robust part-based hand gesture recognition using Kinect sensor," *IEEE Transactions on Multimedia*, vol. 15, pp. 1110–1120, Aug. 2013.
- [97] Y. Wang and R. Yang, "Real-time hand posture recognition based on hand dominant line using kinect," in *IEEE International Conference on Multimedia and Expo Workshops*, pp. 1–4, Jul. 2013.
- [98] I. Laptev and T. Lindeberg, "Space-time interest points," in *Int. Conf. Comput. Vis. (ICCV)*, vol. 1, pp. 432–439, Oct. 2003.
- [99] X. Shen, G. Hua, L. Williams, and Y. Wu, "Dynamic hand gesture recognition: An exemplar-based approach from motion divergence fields," *Image and Vision Computing*, vol. 30, pp. 227 – 235, Mar. 2012.
- [100] A. Vedaldi and A. Zisserman, "Efficient additive kernels via explicit feature maps," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 34, pp. 480–492, Mar. 2012.

-
- [101] A. Mahmood and S. Khan, "Early terminating algorithms for adaboost based detectors," in *IEEE Int. Conf. Image Processing*, pp. 1209–1212, Nov. 2009.
 - [102] C. del Blanco, F. Jaureguizar, and N. García, "An efficient multiple object detection and tracking framework for automatic counting and video surveillance applications," *IEEE Trans. Cons. Elect.*, vol. 58, pp. 857–862, Aug. 2012.
 - [103] J. Huang, J. Zhao, W. Gao, C. Long, L. Xiong, Z. Yuan, and S. Han, "Local binary pattern based texture analysis for visual fire recognition," in *International Congress on Image and Signal Processing*, vol. 4, pp. 1887–1891, Oct. 2010.
 - [104] K. Meena and A. Suruliandi, "Local binary patterns and its variants for face recognition," in *International Conference on Recent Trends in Information Technology*, pp. 782–786, Jun. 2011.
 - [105] W. Liu, S. Li, and Y. Wang, "Automatic facial expression recognition based on local binary patterns of local areas," in *International Conference on Information Engineering*, vol. 1, pp. 197–200, Jul. 2009.
 - [106] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, vol. 2, pp. 2169–2178, Jun. 2006.
 - [107] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 779–788, Jun. 2016.
 - [108] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 3357–3364, May 2017.
 - [109] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128×128 120 dB 15 μ s latency asynchronous temporal contrast vision sensor," *IEEE J. Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008.
 - [110] Y. Zhu, D. Comaniciu, M. Pellkofer, and T. Koehler, "Reliable detection of overtaking vehicles using robust information fusion," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, pp. 401–414, Dec. 2006.
 - [111] H. T. Niknejad, A. Takeuchi, S. Mita, and D. McAllester, "On-road multivehicle tracking using deformable object model and particle filter with improved likelihood estimation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, pp. 748–758, Jun. 2012.
 - [112] A. Jazayeri, H. Cai, J. Y. Zheng, and M. Tuceryan, "Vehicle detection and tracking in car video based on motion model," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, pp. 583–595, Jun. 2011.
 - [113] J. Zhou, D. Gao, and D. Zhang, "Moving vehicle detection for automatic traffic monitoring," *IEEE Transactions on Vehicular Technology*, vol. 56, pp. 51–59, Jan. 2007.

- [114] Y. Chen, Y. Yu, and T. Li, "A vision based traffic accident detection method using extreme learning machine," in *Int. Conf. on Advanced Robotics and Mechatronics (ICARM)*, pp. 567–572, Aug. 2016.
- [115] B.-F. Lin, Y.-M. Chan, L.-C. Fu, P.-Y. Hsiao, L.-A. Chuang, S.-S. Huang, and M.-F. Lo, "Integrating appearance and edge features for sedan vehicle detection in the blind-spot area," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, pp. 737–747, Jun. 2012.
- [116] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: integrated recognition, localization and detection using convolutional networks," *International Conference on Learning Representations*, Apr. 2014.
- [117] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, *et al.*, "An empirical evaluation of deep learning on highway driving," *arXiv:1504.01716*, Apr. 2015.
- [118] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Conference on Computer Vision and Patter Recognition*, 2012.
- [119] Q. Fan, L. Brown, and J. Smith, "A closer look at Faster R-CNN for vehicle detection," in *IEEE Intelligent Vehicles Symposium*, pp. 124–129, Jun. 2016.
- [120] Y. Zhou, H. Nejati, T. T. Do, N. M. Cheung, and L. Cheah, "Image-based vehicle analysis using deep neural network: A systematic study," in *IEEE International Conference on Digital Signal Processing*, pp. 276–280, Oct. 2016.
- [121] A. Vedaldi and K. Lenc, "Matconvnet – convolutional neural networks for matlab," in *Proceeding of the ACM Int. Conf. on Multimedia*, 2015.
- [122] S. H. Lee, M. K. Sohn, D. J. Kim, B. Kim, and H. Kim, "Smart TV interaction system using face and hand gesture recognition," in *IEEE Int. Conf. on Consumer Electronics (ICCE)*, pp. 173–174, Jan. 2013.
- [123] S. Jeong, J. Jin, T. Song, K. Kwon, and J. W. Jeon, "Single-camera dedicated television control system using gesture drawing," *IEEE Trans. Cons. Elect.*, vol. 58, pp. 1129–1137, Nov. 2012.
- [124] D. Lee and Y. Park, "Vision-based remote control system by motion detection and open finger counting," *IEEE Trans. Cons. Elect.*, vol. 55, pp. 2308–2313, Nov. 2009.
- [125] J. J. LaViola, Jr., "An introduction to 3D gestural interfaces," in *Int. Conf. on Computer Graphics and Interactive Techniques*, pp. 1–42, Aug. 2014.
- [126] N. H. Dardas and N. D. Georganas, "Real-time hand gesture detection and recognition using bag-of-features and support vector machine techniques," *IEEE Transactions on Instrumentation and Measurement*, vol. 60, pp. 3592–3607, Nov. 2011.
- [127] S. B. Wang, A. Quattoni, L. P. Morency, D. Demirdjian, and T. Darrell, "Hidden conditional random fields for gesture recognition," in *IEEE Computer Society Conference on Comput. Vis. Pattern Recog.*, vol. 2, pp. 1521–1527, Jun. 2006.

-
- [128] F. Erden and A. E. Çetin, “Hand gesture based remote control system using infrared sensors and a camera,” *IEEE Trans. Cons. Elect.*, vol. 60, pp. 675–680, Nov. 2014.
- [129] E. Ohn-Bar and M. M. Trivedi, “Hand gesture recognition in real time for automotive interfaces: A multimodal vision-based approach and evaluations,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, pp. 2368–2377, Dec. 2014.
- [130] P. Molchanov, S. Gupta, K. Kim, and K. Pulli, “Multi-sensor system for driver’s hand-gesture recognition,” in *IEEE Int. Conf. and Workshops on Automatic Face and Gesture Recognition*, vol. 1, pp. 1–8, May 2015.
- [131] P. Molchanov, S. Gupta, K. Kim, and J. Kautz, “Hand gesture recognition with 3D convolutional neural networks,” in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 1–7, Jun. 2015.
- [132] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, “Learning deep features for discriminative localization,” in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 2921–2929, Jun. 2016.
- [133] S. Scherer, J. Rehder, S. Achar, H. Cover, A. Chambers, S. Nuske, and S. Singh, “River mapping from a flying robot: state estimation, river detection, and obstacle mapping,” *Autonomous Robots*, vol. 33, pp. 189–214, Aug. 2012.
- [134] N. Michael, S. Shen, K. Mohta, Y. Mulgaonkar, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, *et al.*, “Collaborative mapping of an earthquake-damaged building via ground and aerial robots,” *J. Field Robot.*, vol. 29, pp. 832–841, Dec. 2012.
- [135] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, “Autonomous, vision-based flight and live dense 3D mapping with a quadrotor MAV,” *J. Field Robot.*, vol. 33, no. 4, pp. 431–450, 2016.
- [136] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, “Multi-sensor fusion for robust autonomous flight in indoor and outdoor environments with a rotorcraft MAV,” in *IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 4974–4981, May 2014.
- [137] S. Lynen, T. Sattler, M. Bosse, J. Hesch, M. Pollefeys, and R. Siegwart, “Get out of my lab: Large-scale, real-time visual-inertial localization,” in *Robotics: Science and Systems XI*, Jul. 2015.
- [138] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv:1509.02971*, Sep. 2015.
- [139] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, “Learning monocular reactive UAV control in cluttered natural environments,” in *IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 1765–1772, May 2013.
- [140] G. Kahn, T. Zhang, S. Levine, and P. Abbeel, “PLATO: Policy learning using adaptive trajectory optimization,” in *IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 3342–3349, May 2017.

- [141] A. Giusti, J. Guzzi, D. C. Cireşan, F. L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. M. Gambardella, “A machine learning approach to visual perception of forest trails for mobile robots,” *IEEE Robot. Autom. Lett.*, Jul. 2016.
- [142] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield, “Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness,” *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, pp. 4241–4247, Sep. 2017.
- [143] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *Int. Conf. on Machine Learning*, pp. 1889–1897, Jun. 2015.
- [144] F. Sadeghi and S. Levine, “CAD2RL: Real single-image flight without a single real image,” in *Robotics: Science and Systems XIII*, Jul. 2017.
- [145] M. Mancini, G. Costante, P. Valigi, T. A. Ciarfuglia, J. Delmerico, and D. Scaramuzza, “Towards domain independence for learning-based monocular depth estimation,” *IEEE Robot. Autom. Lett.*, pp. 1778–1785, Jul. 2017.
- [146] J. Kim and J. Canny, “Interpretable learning for self-driving cars by visualizing causal attention,” in *Int. Conf. Comput. Vis. (ICCV)*, pp. 2942–2950, Oct. 2017.
- [147] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *Int. Conf. on Machine Learning*, pp. 41–48, Jun. 2009.
- [148] Udacity, “We’re building an Open Source Self-Driving Car.” <https://medium.com/udacity/were-building-an-open-source-self-driving-car-ac3e973cd163>, 2016. [Online; accessed: 2018-06-13].
- [149] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual explanations from deep networks via gradient-based localization,” in *Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017.
- [150] S. Yang, S. Konam, C. Ma, S. Rosenthal, M. Veloso, and S. Scherer, “Obstacle avoidance through deep networks based intermediate perception,” *arXiv:1704.08759*, Apr. 2017.
- [151] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, “From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots,” in *IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 1527–1533, May. 2017.
- [152] W. Gao, D. Hsu, W. S. Lee, S. Shen, and K. Subramanian, “Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation,” in *Conference on Robot Learning (CoRL)*, Nov. 2017.
- [153] C. Richter and N. Roy, “Safe visual navigation via deep learning and novelty detection,” in *Robotics: Science and Systems XIII*, Jul. 2017.
- [154] H. Kim, S. Leutenegger, and A. J. Davison, “Real-time 3D reconstruction and 6-DoF tracking with an event camera,” in *Eur. Conf. Comput. Vis. (ECCV)*, pp. 349–364, Oct. 2016.

-
- [155] H. Rebecq, T. Horstschäfer, G. Gallego, and D. Scaramuzza, “EVO: A geometric approach to event-based 6-DOF parallel tracking and mapping in real-time,” *IEEE Robot. Autom. Lett.*, vol. 2, pp. 593–600, Apr. 2017.
- [156] G. Gallego, J. E. A. Lund, E. Mueggler, H. Rebecq, T. Delbruck, and D. Scaramuzza, “Event-based, 6-DOF camera tracking from photometric depth maps,” *IEEE Trans. Pattern Anal. Machine Intell.*, Nov. 2017.
- [157] H. Rebecq, G. Gallego, E. Mueggler, and D. Scaramuzza, “EMVS: Event-based multi-view stereo—3D reconstruction with an event camera in real-time,” *Int. J. Comput. Vis.*, pp. 1–21, Nov. 2017.
- [158] G. Gallego, H. Rebecq, and D. Scaramuzza, “A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation,” in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 3867–3876, Jun. 2018.
- [159] F. Walch, C. Hazirbas, L. Leal-Taixe, T. Sattler, S. Hilsenbeck, and D. Cremers, “Image-based localization using LSTMs for structured feature correlation,” in *Int. Conf. Comput. Vis. (ICCV)*, pp. 627–637, Oct. 2017.
- [160] J. Kim and J. Canny, “Interpretable learning for self-driving cars by visualizing causal attention,” in *Int. Conf. Comput. Vis. (ICCV)*, pp. 2961–2969, Oct. 2017.
- [161] D. P. Moeys, F. Corradi, E. Kerr, P. Vance, G. Das, D. Neil, D. Kerr, and T. Delbruck, “Steering a predator robot using a mixed frame/event-driven convolutional neural network,” in *Int. Conf. Event-Based Control, Comm. Signal Proc. (EBCCSP)*, pp. 1–8, Jun. 2016.
- [162] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA urban challenge: Autonomous Vehicles in City Traffic*, vol. 56. Springer, 2009.
- [163] D. A. Pomerleau, “ALVINN: An autonomous land vehicle in a neural network,” in *Advances in Neural Information Processing Systems I*, pp. 305–313, Jan. 1989.
- [164] J. A. Perez-Carrasco, B. Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, S. Chen, and B. Linares-Barranco, “Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing—application to feedforward ConvNets,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 35, pp. 2706–2719, Nov. 2013.
- [165] G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor, and R. Benosman, “HFirst: A temporal approach to object recognition,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 37, pp. 2028–2040, Jan. 2015.
- [166] X. Lagorce, G. Orchard, F. Gallupi, B. E. Shi, and R. Benosman, “HOTS: A hierarchy of event-based time-surfaces for pattern recognition,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 39, pp. 1346–1359, Jul. 2017.
- [167] I.-A. Lungu, F. Corradi, and T. Delbruck, “Live demonstration: Convolutional neural network driven by dynamic vision sensor playing RoShamBo,” in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 1–1, May 2017.

- [168] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, “A 240x180 130dB 3us latency global shutter spatiotemporal vision sensor,” *IEEE J. Solid-State Circuits*, vol. 49, pp. 2333–2341, Oct. 2014.
- [169] G. Orchard, R. Benosman, R. Etienne-Cummings, and N. V. Thakor, “A spiking neural network architecture for visual motion estimation,” in *IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 298–301, Oct. 2013.
- [170] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, “Learning deep features for discriminative localization,” in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, pp. 2921–2929, Jun. 2016.
- [171] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, “CNN features off-the-shelf: An astounding baseline for recognition,” in *IEEE Int. Conf. Comput. Vis. Pattern Recog. Workshop*, pp. 806–813, Jun. 2014.
- [172] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine Learning*, vol. 8, pp. 229–256, May 1992.
- [173] S. Valipour, M. Siam, E. Stroulia, and M. Jagersand, “Parking-stall vacancy indicator system, based on deep convolutional neural networks,” in *IEEE World Forum on Internet of Things (WF-IoT)*, pp. 655–660, Dec. 2016.
- [174] S. Zhang, H. Yao, and S. Liu, “Dynamic background modeling and subtraction using spatiotemporal local binary patterns,” in *icip*, pp. 1556–1559, Oct. 2008.