UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

VIDEO UNDERSTANDING THROUGH THE DISENTANGLEMENT OF APPEARANCE AND MOTION

A Master's Thesis
Submitted to the Faculty of the

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

By

Carlos Arenas Gallego

In partial fulfilment
of the requirements for the degree of

MASTER IN TELECOMMUNICATIONS ENGINEERING

Advisors:

Xavier Giró-i-Nieto, Sebastian Palacio, Víctor Campos

Barcelona, October 2018

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

telecom BCN

Video Understanding through the Disentanglement of Appearance and Motion

Author: Carlos Arenas

Advisors: Xavier Giro-i-Nieto, Sebastian Palacio, V ctor Campos

Abstract

Understanding the inner workings of deep learning algorithms is key to e ciently exploit the large number of videos that are generated every day. For the self-supervised learning of the spatio-temporal information contained within these videos, there are several types of algorithms based on convolutional neural networks (CNNs) following an auto-encoder style architecture. However, we have checked that this type of models, trained for the frame prediction task, learn jointly these spatio-temporal information, so the model is not able to recognize appearance-motion combinations not seen during training. Our proposed model, called DisNet, can learn separately the appearance and motion through disentanglement, so that it solves the generalization and scalability problems. To demonstrate this, we conducted numerous experiments under highly controlled conditions, generating speci c datasets that make the \conventional" model fails for the appearance and motion classi cation tasks, and analyzing how well our proposal behaves under the same conditions.

Keywords: deep learning, convolutional neural networks, auto-encoders, disentanglement, motion, appearance.

Resumen

Entender el funcionamiento de los algoritmos de aprendizaje profundo es clave para poder explotar de manera e ciente la gran cantidad de v deos que se generan cada d a. Para el aprendizaje auto-supervisado de la informacion espacio-temporal contenida en los v deos se emplean diversos tipos de algoritmos basados en redes neuronales convolucionales (CNNs) siguiendo una arquitectura de tipo auto-encoder. Sin embargo, hemos comprobado que este tipo de modelos, entrenados para la tarea de prediccion de frames, aprenden de forma combinada esta informacion espacio-temporal, de modo que el modelo no es capaz de reconocer combinaciones apariencia-movimiento no vistas durante el entrenamiento. Nuestro modelo propuesto, denominado DisNet, es capaz de aprender de forma separada la apariencia y el movimiento mediante disentanglement, de modo que resuelve el problema de generalizacion y escalabilidad. Para demostrarlo, realizamos numerosos experimentos bajo condiciones muy controladas, generando bases de datos espec cas que hagan fallar al modelo \convencional" para la tarea de clasi cacion de apariencia y movimiento, y analizando como de bien se comporta nuestra propuesta bajo las mismas condiciones.

Keywords: aprendizaje profundo, redes neuronales convolucionales, auto-encoders, disentanglement, movimiento, apariencia.

Acknowledgements

This project supposes the culmination of my master's degree, in which it allowed me to implement the knowledge learned in some of the lectures given, as well as those acquired during its development.

Thanks to Xavier Giro-i-Nieto, for the e ective work in the guidance of the project that is now presented, allowing me to progress constantly in the right direction. It is also important to mention its teaching and organization of lectures, seminars, conferences and research lines, which woke up my interest to know more about this exciting eld.

To V ctor Campos, for the support and advice received, allowing me to understand in a more rigorous way the actual reasons of some of the problems that arose or results obtained, making possible the implementation, development and debugging of the proposed model.

To Sebastian Palacio, for the constant dedication and time invested in solving all the doubts that came to me, as well as allowing me the integration and familiarization in the host research center.

Finally, thanks to my parents for all the help and understanding received which, even in the distance, have been present every day, supporting and encouraging me to continue forward.

Revision history and approval record

Revision	Date	Purpose
0	14/09/2018	Document creation
1	17/10/2018	Document revision
2	17/10/2018	Document approval

DOCUMENT DISTRIBUTION LIST:

Name	e-mail
Carlos Arenas	carlosargal@gmail.com
Xavier Giro-i-Nieto	xavier.giro@upc.edu
Sebastian Palacio	sebastian.palacio@dfki.de
V ctor Campos	victor.campos@bsc.es

Written by:		Reviewed and approved by:	
Date	14/09/2018	**Date**	17/10/2018
Name	Carlos Arenas	**Name**	Xavier Giro-i-Nieto
Position	Project Author	**Position**	Project Supervisor

Contents

List of Figures

List of Tables

List of Abbreviations

AE	**A**uto-**E**ncoder
AI	**A**rti cial **I**ntelligence
API	**A**pplication **P**rogram **I**nterface
BCE	**B**inary **C**ross **E**ntropy
BoW	**B**ag **o**f **W**ords
CNN	**C**onvolutional **N**eural **N**etwork
CPU	**C**entral **P**rocessing **U**nit
CV	**C**omputer **V**ision
DFD	**D**isplacement **F**rame **D**i erence
DL	**D**eep **L**earning
DNN	**D**eep **N**eural **N**etwork
ETL	**E**xtraction/**T**ransformation/**L**oading
FC	**F**ully **C**onnected
GPU	**G**raphics **P**rocessing **U**nit
GT	**G**round **T**ruth
HOG	**H**istogram **o**f **G**radients
HSL	**H**ue, **S**aturation, **L**ightness
HSV	**H**ue, **S**aturation, **V**alue
HVS	**H**uman **V**isual **S**ystem
LBP	**L**ocal **B**inary **P**atterns
ML	**M**achine **L**earning

NLP	**N**atural **L**anguage **P**rocessing
NN	**N**eural **N**etwork
OF	**O**ptical **F**low
RNN	**R**ecurrent **N**eural **N**etwork
SIFT	**S**cale **I**nvariant **F**eature **T**ransform
VAE	**V**ariational **A**uto-**E**ncoder

Chapter 1

Introduction

1.1 Motivation and Objectives

We need large-scale datasets for training any *deep learning* (DL) algorithm and even more when we talk about videos. For this case, there are plenty of input data x in any website (e.g. YouTube). However, almost all these inputs lack of a target output y. That is why arises the need to use *self-supervised* learning algorithms, such as *auto-encoders* (AE), since the main advantage this entails is that input data do not need prede ned target outputs, due to they are generated automatically during training process an adequate use of their own inputs. Thus, videos become an almost unlimited data source that we can exploit to train *deep neural networks* (DNNs) in order to not only analyzing videos themselves but also to *pre-train* a wide range of models, so that we only need to *ne-tune* on a few data points for their target tasks. Moreover, videos have a remarkable advantage with respect to images; they present a temporal coherence, that is, they follow a logical order. This temporary information learned by the network can be used later on in target tasks such as detecting whether a sequence is ordered or not [24]. However, the main problem these type of models presents is they learn jointly these spatio-temporal information.

Although some of the nomenclatures and concepts used in *machine learning* (ML) are inspired by the brain, does not mean that these models \learn" or \think" in the same way as the human being does. Nevertheless, being aware of their real behavior (based on statistics and algebra) and their limitations or possibilities that this entails, it is possible to infer certain human skills, such as breaking down the information captured and learning it independently. This ability is known as *learning disentangled representations* and is the subject of study for many researchers during the last few years [4, 18, 35, 26, 16, 13].

Focusing visual scenes understanding, it is interesting to know how people are able, at a glance and automatically, to recognize the objects we are visualizing, the actions they perform and under what circumstances they are. This is due to the ability to disentangle what we observe and individually analyze that information (i.e. foreground, motion and background respectively). Based on this assumption, we highlight the work done by Lin et al.[21], since through the extension of such work I took my rst steps as a researcher in this eld, as well as it inspired me to tackle the problem around which this new project is developed.

In this context, the present *Master's Thesis* addresses the study of learning disentangled representations (speci cally *appearance and motion*) through the development of a model based on AE called *DisNet*, which is able to alleviate and even solve some of the limitations that *vanilla AE* models entail when they try to acquire such information jointly, as is the case of *generalization* and *scalability*. This kind of *self-supervised* learning algorithms, are able to nd patterns of similarity (*features*) in a reduced dimensionality space (*latent space*) from the input data. In case of videos, that information is usually extracted by *predicting future frames* based on the observation of the previous sequence. Therefore, to verify the bene ts of our proposal and demonstrate our hypothesis, both models (DisNet and vanilla AE) undergo through some experiments under controlled conditions. This aims to achieve the following objectives:

Be able to disentangle and learn individually the information of appearance and motion contained in videos with our proposal.

Understand the internal behavior of the models presented, by simplifying the problem as much as possible so that its study is easier and more evident.

Show that our proposal is able to generalize to appearance-motion combinations not seen during training.

Address the scalability problem with respect to the number of appearances and motions learned.

1.2 Hardware and Software Resources

For the training of any DNN is highly recommended the use of powerful hardware devices that allow us to run several experiments without the need to wait for a long time between each of them, as it would considerably slow down the work of the researcher. These components are the aforementioned GPUs, responsible for e ciently computing large blocks of input data within the models, as well as updating their *weights*. In addition to GPUs, we need another kind of components, the so-called *central processing units* (CPUs) which are necessary to access memory, load and manage the input data to feed the network, although they are not as powerful as GPUs. In this way, the combination of both in a proper way speeds up the process, since it releases the GPU for those tasks, focusing only on the heavy computations.

All hardware resources are provided by the *German Research Center for Arti cial Intelligence* (DFKI), where I mainly developed my thesis. These resources are grouped into *5 compute nodes*, whose specs are summarized in table 1.1.

Name (alias)	GPU	RAM (GiB)	CPU	RAM (GiB)
kasan	8x GTX 1080 TI	8x 11.17	Xeon E5-2683W v4 (32+32HT x 2.1GHz)	512
kansas	2x GTX 1080 TI	2x 11.17	Xeon E5-2687W v3 (20+20HT x 3.1GHz)	256
kiew	2x GTX 1080	2x 8.11	Opteron 6348 (24x 2.8Ghz)	128
koeln	2x GTX 1080	2x 8.11	Opteron 6180 (24x 2.5GHz)	128
kassel	GTX 1080 + Titan Black	8.11+11.17	Opteron 6172 (24 2.1GHz)	64

Table 1.1: Hardware resources specs

Although our architectures are e cient and fast enough to run on a single GPU, access to all these resources allowed us to run in parallel the numerous experiments needed to design and adjust the con guration of them, as well as to generate the datasets used in their training.

The development of the entire project (i.e. generation of datasets, design and adjustment of the di erent architectures used, as well as the execution of the numerous experiments) were carried out through the use of high level APIs from *TensorFlow*[1] framework, which o er exibility, e ciency and simplicity during work. In order to speed up the process through the GPUs computations, *CUDA* platform and *cuDNN* libraries supported by this framework were also needed. All these software resources were completely integrated within all the previous mentioned compute nodes, running on *openSUSE 42.3/x86_64* operating system . Finally, the interaction with all the aforementioned resources (both hardware and software) was done through the *PyCharm* development environment. All code generated during this project is publicly available at:

`https://github.com/carlosargal/DisNet-2018-tfm/`

1.3 Work Plan

For the completion of my Master's Thesis I invested around 9 months (from 02/18 to 11/18). Since it is a research-oriented project of a certain temporal extension it is very di cult to establish a previous complete work plan, because many of the objectives and motivations of this (and any other) research line are evolving along with the development of the project over time. That is why, the planning of the numerous tasks carried out were established in a short term. The dynamics consisted of a couple of weekly meetings, one with the department of the host research center (DFKI), where each member explained their progress, and another with the members of this project, where we proposed the tasks to be done for next week.

The rst month implied adapting both to the lifestyle of the destination country (Germany) and to the work dynamics in the DFKI, in terms of familiarization with colleagues and the hardware and software resources mentioned in section 1.2.

During the next two months I focused on settle the basis of this project, through the bibliographic consultation of many works carried out up to date in areas such as unsupervised learning, video analysis and learning disentangled representations. This allowed me to establish a well-de ned research line, based on the motivation and objectives described in section 1.1, from which start to develop the problem.

Throughout the following 3 months, both the datasets and architectures described in chapter 4 were implemented. During this period I devoted a lot of e ort and time to understand and gain experience programming with the Tensor ow APIs, which I had not used until then and with which I nally generated all the code from scratch.

After the internship in Germany, I was able to continue using all resources remotely, which allowed me to carry out the experiments that validated the established hypothesis and draw di erent conclusions. This was carried out during the rst two months after my return, so the last month was devoted to writing this document, which structure is detailed in the next and last section of this chapter.

[1] `https://www.tensorflow.org/`

1.4 Document Structure

Chapter 1 has allowed the reader to have a clear idea, on both what the implementation of this project entailed and the magnitude of it, as well as what will be discussed throughout the document.

Chapter 2 de nes the fundamentals and basic notions which holds the eld of arti cial intelligence, contextualizing and generating a well-de ned conceptual map.

Chapter 3 details everything related to the *multimedia data analysis* (images and videos) and the evolution since their beginnings until now.

Chapter 4 describes the methodology followed to generate the synthetic videos datasets, as well as the architectures used for their subsequent analysis by performing a set of experiments.

Chapter 5 analyzes some of the experimental results obtained during the code development, after making several adjustments and validations (both to improve the results and to solve some of the problems that arose), until achieving the nal version of it. These results lead us to di erent conclusions and future development, which are summarized in *Chapter 7*.

Chapter 6 details the estimated costs, associated to the time spent along with resources, both human and material).

Finally, all the *bibliography* is provided at the end of this document.

Chapter 2

Artificial Intelligence

It is extraordinary to see how the eld of *arti cial intelligence* (AI) has evolved in the last decade and how it has been fully integrated into society in practically all the areas of our daily lives. Although the concept of AI has been used for decades, today we still nding multiple de nitions of it. The reason for this ambiguity is due to the fact that this concept depends on the de nition of intelligence, which can be interpreted from di erent points of view. If we extract a common idea of all these de nitions we could say that *\arti cial intelligence is a sub-discipline of computer science eld, that looks for the creation of machines that can imitate intelligent behaviors"*. When we talk about imitating we refer to the realization of the tasks itself and not how they do them. Therefore, there are numerous alternatives to perform the same problem, being all of them valid by de nition.

There are many types of intelligent behavior that machines can simulate, where in certain situations they achieve a higher performance than humans. However, what makes us di erent from this kind of intelligence is the ability to perform multiple tasks, since they are limited to simulating only one of those behaviors or a very small group of them [1] (at least until now). In this way, AI includes a set of subcategories, which correspond to the di erent types of intelligent behaviors to be imitated. Fields such as robotics, *natural language processing* (NLP) or *computer vision* (CV) are some of them. However, if there is a behavior that really de nes an agent as intelligent, it is the ability to learn, that is, *machine learning* (ML).

2.1 Machine Learning

Some of the processes that occur in the brain continue being a great mystery to neuroscientists. Despite this, in elds such as ML, the brain has been a source of inspiration on which many of the most important concepts have been developed.

By de nition, machine learning *\is the branch of arti cial intelligence that seeks to equip machines with learning capacity"*. When talk about learning we refer to the mechanisms that allow us to generate knowledge from a set of experiences. All the algorithms and techniques within this eld can be classi ed into three large groups, depending on the learning paradigm they are applying. These three groups are:

Supervised learning: $y = f(x)$

Unsupervised learning: $f(x)$

Reinforcement learning: $y = f(x); z$

Supervised learning is based on discovering the relationship between some input and output variables. In other words, learning arises from teaching these algorithms what is the result we want to obtain for a given input value. Therefore, the key of supervised learning is to generalize the knowledge learned through observation, being the paradigm that has had the most practical application during the last decades.

On the other hand, *unsupervised learning* is able to produce knowledge only from the data that is provided as input, without the need to explain to the system what result we want to obtain. To do this, it looks for similarity patterns within the input data and groups them based on it (clustering), so that for a new entry the algorithm is able to classify it within one of these groups based on the similarity of its features. Despite the difficulty that this entails, one of the main advantages that can be deduced is obtaining large-scale datasets without much effort, since it is not necessary to carry out any kind of manual labeling. That is why scientific community around ML field admits that the future of it goes through this paradigm.

Finally, in *reinforcement learning*, there is an agent that learns to make decisions based on observations to maximize a future reward (or minimize a penalty). Although it can be applied to a wide range of tasks, this learning has always been closely related to the use in traditional video games, due to its great similarity in terms of performance dynamics.

We already know that ML is another discipline contained within the field of AI. However, unlike the others, it stands out for how it performs a certain task (i.e. the system learns to perform that task, instead of being programmed in a classical way), regardless the task itself. This difference makes ML being one step ahead, maintaining a strong connection with the remaining disciplines that respond to the different intelligent behaviors.

Within ML we find a wide range of techniques that serve to cover different types of applications (e.g. decision trees, support vector machines, regression and classification models, clustering techniques). However, the technique that has made the ML field famous during the last decade has been the *neural networks (NN)*.

What makes NN interesting is that they are able to learn in a hierarchical way, that is, information is learned by levels, where the first layers learn very basic concepts and in the later layers the previously learned information is used to learn more abstract concepts. This means that as we add more layers, the information that is learned is more abstract and interesting. The increase in the number of layers and complexity is what makes these algorithms are known as *deep learning (DL)* algorithms.

2.2 Deep Learning

The massive increase in data that is continuously generated from more sources, the social awareness of its importance and the cheapening prices on storage devices, have caused a trend to accumulate large amounts of data, which is well known as *Big Data*. This, along with the access to increasingly powerful *graphic processing units* (GPUs), which are able to perform a large number of computational operations in a short period of time, are the reasons that have led to the development and implementation of powerful and complex algorithms, called *deep neural networks (DNN)*. The joint growth of these three essential components (i.e. big data, big computation and complex algorithms) have created tools and applications that make our lives much easier on a personal and professional level. All these clear bene ts have made these algorithms widely accepted for all of us, although the lack of knowledge about their performance can generate distrust among the most skeptical.

Within the scienti c community that studies and develops these powerful algorithms, called *deep learning* (DL) algorithms, this lack of knowledge is not so evident. However, since its nature is based on hierarchical architectures of large depth and dimensionality (hence the name of *deep neural networks* or DNN), sometimes the researcher who has developed a model is not fully aware about its inner workings.

Over the last years, it has been a common practice to develop deep learning models capable of improving the results of previous works, by the "simple" fact of increasing its complexity thanks to the addition of more levels of abstraction (*layers*) and/or the combination of several methods. However, this trend is changing, since in a certain way it is increasingly di cult to overcome the *state-of-the-art* in some tasks. Therefore, the current concern is not that the model obtains very good results, but rather doing it e ciently and in a way so that we are able to understand what and how the network learns.

So far we have focused on familiarizing with many of the concepts that are heard today, which in some cases they are overlapped or interpreted in a wrong way, leading in many cases to a greater confusion. This contextualization allows us to clearly see where is located one of the most demanded elds in the last decade. It is in section 3.3, where we explain some of the DL algorithms that are directly related to this thesis. Worth noting the work of the author Goodfellow et al. [11] which includes in an extensive way the most important concepts on which DL is based.

Chapter 3

Multimedia Data Analysis

3.1 Basics of Images and Videos

All groups of animals, even the most primitive, need mechanisms to perceive and interact with their environment. This allows them to adapt to adverse changes, feed themselves, interact with other individuals or avoid risky situations, etc.

Well-known as *sensory organs*, there are several types of them from the simplest, as chemical receptors, to others that react to physical stimuli (e.g. light or sound). Eyes can be considered one of the most complex sensory organs. There is a set of *photoreceptor cells* located in the retina which are stimulated by electromagnetic radiations. These stimuli are sent to the brain as nerve impulses, where the reconstructions of the images are nally performed.

Objects perception, in terms of shape and color, known as *illumination of the object* $I(\lambda)$, depends on three essential factors:

Spectral density of a source $L(\lambda)$.

Reflectivity of the object material $r(x, y, \lambda)$.

Sensitivity of the photoreceptors to that illumination. In the *human visual system* (HVS) we nd two kinds of them; *rods*, sensitive even at low light intensity although with black and white vision (*achromatic vision*) and *cones*, which receive color information (*chromatic vision*) although they need a higher light intensity (g.3.1).

This chromatic vision is composed by the combination of three colors: red, green and blue (RGB) according to the *Young{Helmholtz Theory*, in which *\any color can be reproduced with an adequate mixture of three primary colors"* [42]. This theory postulates the existence of three di erent types of cones (shown for the rst time in 1956 by Gunnar Svaetichin [36]), where each of them is de ned by a *sensitivity curve* centered around the wavelength that characterizes each of the three mentioned colors.

Therefore, the radiation absorbed by the eyes comes from the radiation re ected by objects that are exposed to a light source $I(x, y, \lambda) = r(x, y, \lambda)L(\lambda)$. For HVS, that light source must be within the wavelength range [380, 780]nm, known as visible light spectrum (g.3.2).

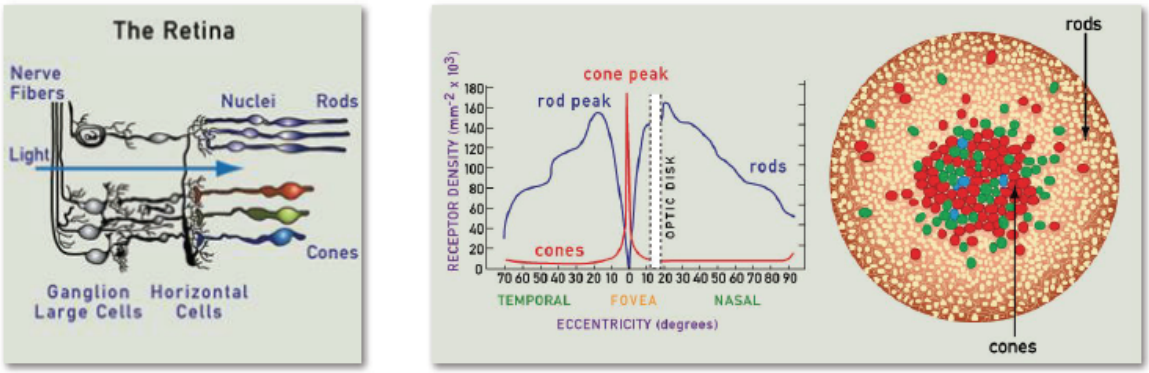


Figure 3.1: Rods and cones distribution in the HVS.

Since the HVS is quite limited in terms of wavelength ranges, we find certain limitations that make it impossible for us to observe the totality of our environment or with the accuracy we would like to. That is why science has always devoted great efforts in the development of tools that overcome these barriers, to progress in the research of fields as diverse as medicine, biology, physics, astronomy, meteorology or signal theory, among others. Although these tools are very relevant for research, there are other industry sectors such as photography, television, cinema or even motor industry in which the acquisition and treatment of images are of special interest.

Current devices, like microscopes, telescopes, photo and video cameras, as well as viewers and scanners that allow visualizing all kinds of invisible spectra to our eyes, differ in the way they capture, process and represent images. However, most of them have a common feature, the transformation of the signal from analog to digital domain.

A *digital image* is basically a multidimensional matrix in which each cell contains a discrete numerical value representing a small spatial region of the original image. These cells are called *pixels (picture elements)* and their values are obtained by a procedure of sampling and quantization of the original image, characterized by a continuous and analog function $f(x, y)$. The quality of a digital image depends on two main factors:

Resolution: Number of pixels per unit area. The higher the resolution, the smaller the size of each pixel, which leads to an image with more detailed elements in terms of shape. This property is closely related to the sampling process.

Value range: Amount of values each pixel can take. The higher the range, the more natural and textured the elements of the image. This property is closely related to the quantization process.

It is possible to choose between several types of color spaces to represent the images, depending on their nature and application. The *RGB space* is one of the most widespread in the use of devices for capturing and representing color images (e.g. digital cameras or screens), where each digital image is represented by a three-dimensional matrix in which the x, y components determine the spatial position of the cell in the image and the z component contains the intensity value for each of the three primary colors, also known as *\channels"*.

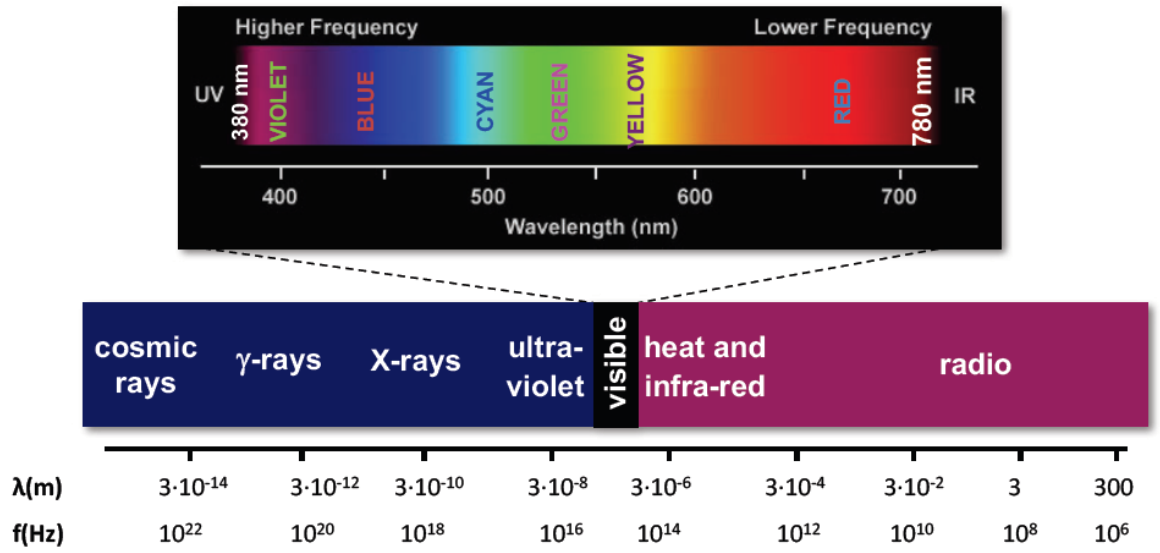


Figure 3.2: The electromagnetic spectrum.

The *perceptual space* is another color space where the values of the z component are determined by the *luminance* and *chrominance* of the image. In this space we find different types like HSV (Hue, Saturation, Value) or HSL (Hue, Saturation, Lightness).

Digital videos are a set of digital images (a.k.a. *frames*) arranged sequentially and ordered in time, so they add an extra dimension (time) to the matrix described above. In addition to the resolution and the value range of each image, the digital video quality depends on the amount of frames per second (fps) represented, so the higher the fps, the more fluent the video. This property depends on the temporary sampling of the analog video.

With current devices, it is possible to capture and process all kinds of representations of the environment for different ranges of frequency spectrum. In case of images, these representations correspond to a specific instant, so the information we can extract is limited to the spatial domain. However, in case of videos such representations evolve over time, so the information obtained is much higher (spatial and temporal information).

3.2 Computer Vision

"Vision" is a much wider concept than simply capturing and processing images or videos. It is also necessary to extract and analyze the information contained within such representations to obtain a *coherent interpretation* of our environment and thus be able to make decisions based on it. *computer vision* arose from the need to provide that interpretation from visual information to machines since, until then, it was a task performed only by humans.

Common situations for us such as recognizing objects or actions become a big challenge for computers, where any external factor that modify the perception of representations (e.g. objects partial occlusion, presence of reflections and shadows, changes in illumination and object pose or even inter-class variability) make this tasks even more difficult.

Computers are unable to recognize an entire object from a digital image without a previous processing. However, through the use of *feature extraction techniques*, they are able to detect edges, contours, corners, interesting points or parameterizable shapes.

These techniques are widely varied in terms of methodology, complexity and application. For instance, one of the simplest ways to obtain edges is computing the rst derivative (gradients) or second derivative of an image through the use of linear lters (e.g. *Sobel, Prewitt, Roberts, laplacian, di erence of Gaussian*), which detect discontinuities in the image intensity. Including other processes to the use of lters results in more sophisticated edge detectors, like *Canny* (adds noise reduction, thresholding and binarization) or *gPb* (adds color and texture information [14]). Another example is obtaining *invariant local features (descriptors)*, which capture the information around detected interesting points. There is a wide variety of algorithms in charge of detecting and extracting many kinds of descriptors, such as SIFT (*scale invairant feature transform*) [22], LBP (*local binary patterns*) [25], HOG (*histograms of gradient orientations*) [6], etc. Since these features are local and invariant to some image parameters (e.g. translations, rotations, scaling) they are robust against some phenomena, like occlusion and clutter. Combining some of these techniques/algorithms allow computers to scale and generalize the problem to recognize from basic and parameterizable shapes (*linear regression, Hough transform, RANSAC*) to increasingly complex and varied objects (*generalized Hough transform [2], sliding window [41], bag of words [19]*).

Something similar happens with videos. Pre-processing techniques are needed to model the object temporal evolution within the sequence of images. The most common technique to extract and model this motion is well known as *optical ow*.

3.2.1 Optical Flow

Once computers are able to recognize objects, arises the need to know the state of them, that is, the action they perform. This is possible thanks to the extraction of motion contained within the temporal domain. But how is possible to know the *real motion* of an object in the 3D space from a sequence of 2D images? Actually, this is not possible, since the spatial dimensionality reduction makes we lose valuable information about the action that is being carried out. Therefore, we can only make estimates from the intensity variability of the pixels along the temporal sequence of images. This estimate is known as *apparent motion* or optical ow. If we consider that changes in pixel values are only due to the motion in the scene, any intensity value $I(x^0, y^0)$ at time $t + \Delta t$ can always be found (and associated to a pixel) in the image at time t (and vice versa). This hypothesis is re ected in the *brightness constancy equation*:

$$I(\vec{r^0}, t + \Delta t) = I(\vec{r^0} \quad \vec{D}(\vec{r}), t) \tag{3.1}$$

Where $\vec{r^0} = (x^0, y^0) = (x + \Delta x, y + \Delta y)$ is the *position vector* at time $t + \Delta t$ and $\vec{D}(\vec{r}) = (\Delta x(\vec{r}), \Delta y(\vec{r}))$ is the *displacement vector* containing the optical ow values of all pixels of the image at time t.

The main drawback of this kind of motion is the inclusion of "noise" due to the camera motion as well as the lighting changes. However, having enough knowledge about that noise it is possible to compensate or even mitigate it from videos [39]. For that reason, it is necessary to relax the constraint of the previous hypothesis, assuming optical ow cannot always ful ll it:

$$I(\vec{r}, t + \Delta t) \quad I(\vec{r} \quad \vec{D}(\vec{r}), t) \tag{3.2}$$

To measure the quality of the estimate we simply subtract both terms from the previous equation, obtaining an \image" where each pixel contains the di erence between the intensity value of the estimated pixel at time $t + \Delta t$ and the intensity value of its associated pixel (ground truth) at time t. This image is known as DFD (*displaced frame di erence*) and the closer to zero are their values, the more accurate is the estimated optical ow.

Optical ow optimization focuses on minimizing DFD and is usually approached in two ways:

Direct exploration methods: Only the values of the original functions (images) are used. This is the case of *Block Matching*.

Differential methods: The values of the original functions as well as their derivatives are used. This is the case of *Lukas-kanade*.

All of them assume that images can be segmented into small regions (which di er between models), where all the pixels within that region have a small and homogeneous motion de ned by a given parametric model. The complexity of such a parametric model will not always make the estimate more reliable, as well as it is computationally more expensive (number of regions x number of parameters). The most common types of parametric motion models are translation and a ne mode.

Regarding the di erential methods (Lukas-kanade) there are two main problems that can lead to an error in the optical ow estimation:

Aperture problem: All regions are de ned by a sliding window centered on each pixel of the image. Therefore, when the window is not large enough it is possible that during the exploration of the image, the component of the ow perpendicular to the gradient (i.e. parallel to the edge) can not be measured. The solution to this problem is displacing the window in more than one direction.

Large motion problem: Small motion assumption does not hold in practice. To solve it, we use a multi-resolution algorithm, in which downsampling the image we also reduce the motion magnitude.

The state of the art regarding optical ow is quite obsolete [3], due to its limitations in terms of exibility, computational cost and scaling. The culmination of this whole process comes when computers go from interpreting this visual information to learning that understanding for themselves. This is where *deep learning algorithms* come in, as we will see below. However, optical ow is a powerful tool that is still being used to improve the results of these novel and more advanced algorithms [9, 29, 32, 43, 5, 27]. In other cases, what is tried is to obtain the same bene ts as OF, completely avoiding its use as as shown in Fan et. al [10] work.

3.3 Deep Learning for Video Analysis

DL algorithms used for videos are very similar to those used for images, since videos are essentially a set of images arranged sequentially, as we discussed in section 3.1.The most used DL algorithm to extract and learn features contained in images and videos are the *convolutional neural networks* (CNNs), which we describe below.

3.3.1 Convolutional Neural Networks

In general, the number of connections for each of the neurons that the input layer contains in a traditional neural network (*multilayer perceptron*) is equivalent to the number of parameters that each sample has. For images each pixel is considered as a parameter, so they have a high-dimensionality, which increases substantially with a higher resolution/size. This dimensionality supposes a very high computational cost for the network. That is why CNNs emerged, which manage to reduce this dimensionality through the use of *convolutional lters*, so that the network learns more e ciently and quickly. In this way, the feature extraction is obtained through the convolution of images with a set of lters, which are learned during training.

Since the nature of images presents a *grid-like topology* (as explained in section 3.1), there are three factors that motivate the use of this kind of architecture:

Local connectivity: Each neuron is only connected to a region (patch) of the input image, where the size of the patch is equivalent to the lter dimension (a.k.a. *receptive eld*). In this way, the coordinated combination of di erent neurons allow to \visualize" the input image with a smaller number of connections, which leads to a reduction in the number of parameters as well as a faster computation of the activation function in every neuron.

Parameter sharing: Once we can \visualize" the whole image through a set of neurons it is possible to share its parameters, so that these reduction is even greater. Thus, the number of parameters to convolve an image corresponds to the number of elements contained in the convolutional lter matrix. All the neurons that share their parameters (i.e. use the same lter) are grouped within the same *"feature map"*. There are as many feature maps as lters are applied to the image in each layer, so the total amount of parameters is equal to the number of lter elements times the number of lters applied.

Pooling and subsampling: Between convolutional layers a reduction of the number of neurons within the same neighborhood is made. This again reduces the dimensionality while providing invariance to small changes.

Following these 3 essential principles we nd numerous models with very varied complexity (i.e. size and number of lters as well as number and types of layers, such as convolutional, sub-sampling, fully connected, or batch normalization layers), depending both on their target tasks and their inputs. The model that gave fame to this type of architectures was designed by a group of students in 2012 [17], with which they achieved much higher classi cation results than the previous one on *ImageNet* dataset [7], following a framework similar to that proposed by Lecun et al. [20] in 1988. Since then, constant improvements were made every year, achieving better and better results with models such as VGGNet [30], GoogLeNet [33] or ResNet [12].

There are several ways to extend its use to a sequence of images (videos), which can be summarized in 4 main types:

Single frame models: It consists of performing a spatial 2D convolution for each frame of the video individually, in the same way that we would do with a single image, but nally we combine all the outputs on top of a pooling operation (e.g. max, sum, average). This combination can be done in many ways as Ng et al. [43] showed. This is a fairly simple implementation in which we can exploit the content of the whole video and what is more interesting to reuse pre-trained models for images. However, the main drawback of this method is that it does not contain temporal information about it, since pooling is not aware of the temporary order.

CNN + RNN: To solve this lack of temporal information, instead of pooling the output of the frame by frame convolutions, we replace it with another type of algorithm commonly used for the exploration of sequential information, called *recurrent neural networks* (RNN) [8]. The problem is that we can not parallelize the learning process in the same way as we do with the previous method, which implies limitations in terms of storage, so for videos with very long sequences this strategy is not valid.

3D CNN: If we think of an image as a N-dimensional matrix, a video is basically a matrix of dimensions N + 1. Therefore, it is possible to perform convolutions through the use of lters with an extra dimension. This is how the *3D convolutional networks* (C3D) arise [37], which jointly perform a spatial and temporal convolution of all the video at the same time. In this way we solve the problem of the sequentiality that the previous method presented. The main drawback is its high computational cost, so the problem of very long sequences still exists. To mitigate this it is necessary to split the video into *chunks* (also known as *clips*), where the number of frames must always be the same in order to feed the network properly. Therefore, C3D can work with videos of any length. However, these clips usually have 16 frames (in videos with a frame rate of 30 fps, their clips last approximately half a second), so the temporary information captured by the network is too short, since it does not exceed a longer length than the input clips.

Two-stream CNN: Thanks to the use of OF it is possible to obtain from the original clip another clip that represents the motion contained within the image (e.g. instead of having 3 channels for the RGB color representation, we have 2 channels representing vertical and horizontal motion). In this way, with the original clip we feed a CNN frame by frame, which learns the temporal information, and in parallel we use the generated clip that contains the OF as input of another replicated CNN, which learns the motion (temporal) information [29]. The main drawback is the bottleneck produced by the OF, which makes the network not scalable due to the computational and memory cost.

3.3.2 Auto-Encoders

There are di erent architectures that exploit the bene ts of unsupervised (or self-supervised) learning, based on the assumption that there is a relationship between input data that allows them to be grouped in a space of characteristics lower than the input space (*manifold assumption*). Among all of them we highlight the *auto-encoder* (AE) models. This type of algorithms consist essentially of two blocks, encoder and decoder, which are characterized by having a *mirrored symmetric topology*, that is, the decoder reverses the operations performed by the encoder.

The main function of the *encoder* is the extraction of features contained in a space of reduced dimensionality (*latent space*) from the direct exploration of typically unlabelled inputs data x. In the case of image analysis, this exploration is based on CNNs models, where the combination of convolutional and pooling layers allow reducing the dimensionality of the input. The *decoder*, located just after the encoder, is generally responsible for reconstructing the input x of the model from the feature vector that the encoder has generated for that input, so that the network learns to interpret that information. Certain types of AE also introduce random noise to the encoding-decoding process, which has been shown to improve the robustness of the resulting patterns, well-known as *variational auto-encoders* (VAE).

Therefore, thanks to the combined work of encoders and decoders, the AEs are able to generate a latent space between both blocks of great value. In this way, through the pre-training of AEs for a reconstruction task, it is possible to do transfer learning for a nal task (such as classi cation), so that we can obtain good results without the need of a very large labeled dataset. However, in order to learn the temporal information of the videos, it is not enough simply to do a reconstruction of the input, but rather that more semantically rich tasks are required, such as the prediction of future frames. However, we can not learn the temporal information of the videos simply by doing a reconstruction of the input, but we need more semantically rich tasks, such as the *prediction of future frames* [32, 23, 28, 38, 40, 15]. For this, AEs exploit some of the techniques already discussed in the previous section, such as the use of CNN + RNN, C3D or two CNN stream as the structure of the encoder.

Chapter 4

Methodology

As we have seen in the previous chapter, the appropriate use of convolutional (and also recurrent) layers, following an AE style architecture, allows us to exploit the temporal and spatial information from videos without the need of target outputs for their training.

The main problem with this type of vanilla AE is that the appearance and motion information learned (and contained in the latent space) is directly related. In this way, it is highly probable that the network learns to associate that a certain object always performs one motion and not another, since the use of real videos datasets can not assure that the network sees all the possible combinations during training (*generalization problem*). In addition, not generalizing for unseen appearance-motion combinations, makes the learning complexity of the model increases in a *multiplicative factor* when we scale to more types of appearance and motions, since it is necessary to previously learn all the possible combinations (*scalability problem*).

Therefore, if we want to further exploit the advantage of large amount of unlabeled videos that arise constantly, we must eliminate the relation between appearance and motion learned, generating two completely independent latent spaces, such as makes our proposed DisNet model. Thanks to disentangling we alleviate the problem of scaling the space of possibilities (regarding the combinations of appearance and motion), so that by increasing the number of possible appearances and possible motions the complexity increases in an *additive factor*, since learning is independent. This implicitly leads to solve the generalization problem, so it is not necessary to see all the appearance-motion combinations during training, but it is enough seeing such appearance and motion, even if they are not explicitly in the same video.

Given this hypothesis, we are going to validate that vanilla AEs really have these problems and how our proposal can solve them. That is why we designed the *Seen/NotSeen benchmark*, forcing the vanilla AE to fail and seeing how our proposal of disentangling behaved under the same conditions. To do this, we generated a couple of datasets of synthetic videos, under controlled and very restricted conditions (*MovingSymbols2_Seen* and *MovingSymbols2_NotSeen*). This allowed us to pre-train both models for the task of predicting the $t+1$ frame (*vanilla frame predictor* and *disentangled frame predictor*). Once this is done, we adapt the architectures for the task of classifying appearance and motion (*vanilla classi er* and *disentangled classi er*) and we ne-tune on the classi er part, freezing the rest of the architecture in order to maintain the latent space features unchanged.

4.1 Datasets

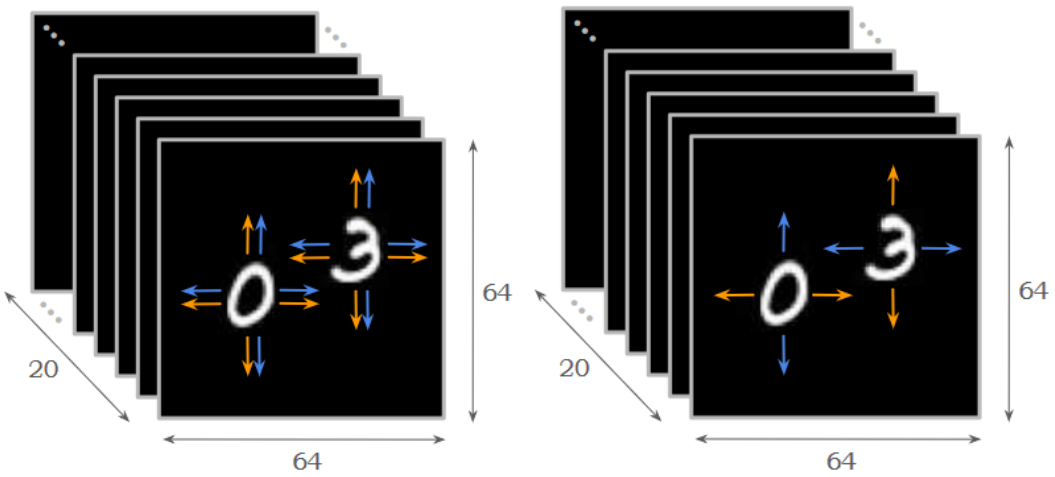
The objectives set by a research line largely determine the required dataset, so that its correct choice allows to demonstrate and justify the results obtained to validate the proposed hypothesis. For the development of DL algorithms applied to videos there is a wide range of datasets well known by the entire community of scientists around this eld (e.g. UCF101 [31], HMDB-51, KTH). All of them, besides using them to train the proposed model, are used as benchmarks to compare how good the model is for a given task with respect to other proposals. However, given the nature of their videos, these datasets are not recommended to analyze the internal behavior of the network. For this purpose, we need more simple ones, usually composed of synthetic videos, such as *Moving MNIST* dataset.

The problem of Moving MNIST is the lack of exibility when generating it. As a direct alternative we have the *Moving Symbols* dataset [34], which o ers greater control over the parameters that we want to modify, such as rotating, scaling, changing the objects appearance, changing the motion speed, adding di erent backgrounds, etc. This allows highly di erent between training and testing datasets as much as you want. However, it is not possible to de ne the type of motion made by the symbols, key aspect to carry out our experiments. The symbols start randomly (both in position and direction) with a linear motion, which bounce with a certain angle when reaching any of the limits of the image. Therefore, it was necessary to modify the *moving_symbols.py* source code , de ning the di erent types of motions performed by the symbols, as well as the creation of a new le (*generate_MovingSymbols.py*) responsible for giving the order to the source code to generate the dataset with the desired parameters.

Since we want to check how vanilla AE and DisNet models behave when they are validated on combinations of appearance-motion both seen and not seen during training, we generate two datasets which we call MovingSymbols2_Seen and MovingSymbols2_NotSeen. The number 2 corresponds to the types of symbols that can be shown (appearance) and the types of motions they can perform. All the speci cations of both datasets are collected in table 4.1. As we can see, they are simpli ed as much as possible and share most of the customization parameters.

	MovingSymbols2_Seen		**MovingSymbols2_NotSeen**	
	Training	**Validation**	**Training**	**Validation**
Number of videos	5000	500	5000	500
Appearance-motion combinations	(0 or 3)-Horizontal (0 or 3)-Vertical		0-Horizontal 3-Vertical	0-Vertical 3-Horizontal
Bounding angle	180^o			
Number of symbols	1			
Symbol scale	1:1			
Symbol speed	8 pixels/frame			
Video resolution	64x64 pixels			
Video length	20 frames			
Color output	True			
Pixel values	[0, 255]			

Table 4.1: MovingSymbols2 datasets specs.



(a) MovingSymbols2_Seen (b) MovingSymbols2_NotSeen

Figure 4.1: Graphic representation of *MovingSymbols2* datasets. Orange arrows: kind of motion seen in training; Blue arrows: kind of motion seen in validation.

What really di erentiates these datasets is the combination of appearance-motion seen in training and validation. Thus, in MovingSymbols2_Seen we have the four possible combinations both in training and validation, so that half of the dataset performs horizontal motions and the other half vertical motions, where the symbol which executes that motion in each video is chosen randomly among the digits 0 and 3 (See gure 4.1(a)). However, in case of MovingSymbols2_NotSeen, the horizontal motion in training is always performed by the digit 0 and the vertical motion by the digit 3, whereas in validation it happens just the reverse, (0 only performs vertical motions and 3 just horizontal), as shown in gure 4.1(b). This carefully selected con guration facilitates the subsequent results analysis for the following reasons:

Simplicity and similarity intra-datasets: We reduce the uncertainty of the results to the maximum, since the behavior of the videos is restricted to very speci c conditions and common to all of them. In this way, the error is only conditioned by the type of appearance and motion.

Similarity inter-datasets: The increase in di culty from MovingSymbols2_Seen to MovingSymbols2_NotSeen is only conditioned by the appearance-motion combinations that we stopped seeing during training.

The les generated for each dataset are stored following a directory structure as follows:

$$MovingSymbols2\_(Seen/NotSeen) > (train/test) > (Horizontal/Vertical) \quad (4.1)$$

For each video, a le of the form *\motion"_video_\# video".avi*, contained in the corresponding directory is generated. Additionally, 3 types of .npy les (arrays) are generated, in addition to the aforementioned video les:

List of the appearance label ids: This list is lled out at the same time that the videos are generated, so to reference each label, it is enough to look at its position in the list thanks to the number provided by the video.

List of background label ids: The creation and selection procedure is the same as for the appearance labels. However, its use is out of scope of this project, since our objective is not learning to disentangle and classify backgrounds, leaving this as future development.

Binary Bounding box: Same size and position as the symbol contained in each video, which are used as a mask to segment the foreground from the background. Since in our case the background is black, its use is not necessary, and it is especially important when a di erent background is applied. This allows us to extend this project for the segmentation task, leaving this also as future work.

The main performance bottleneck that makes it hard to use the GPU at 100% e ciency is the sequential reading of the data between training steps, because the GPU has to wait for new data to work on. Since the generated videos are not very heavy, it is specially interesting to store large blocks of them in another le format that allows faster access and loading of the input data that feed the models. We are talking about the *TFRecords* format, a simple record-oriented binary format that many TensorFlow applications use. This format allows to add in a structured way relevant information of the videos (e.g. labels, size, length, lename), as well as to parallelize the reading of the data so new training / validation data is always available whenever the GPU is ready. Therefore, the nal databases with which we feed the network are composed of this type of les, which are generated with the code *generate_MovingSymbols_tfrecord.py*.

4.2 Architectures

The models described below are the culmination of a large developing and debugging process, in which new ideas and implementations constantly emerged. Some of them were necessary to solve the problems that were arising. However, others were used to improve the results of our study, sometimes successfully and sometimes not so much.

Following the same reasoning for the development of MovingSymbols2_Seen and MovingSymbols2_NotSeen datasets, the structure and inner workings of both vanilla AE and DisNet models are similar, always seeking the e ciency and simplicity of the model. In this way, we ensure that the bene ts o ered by our proposal are only re ected by the disentangling of the appearance and motion into two di erent latent spaces. Before explaining the architecture of each model individually, we focus on explaining (whenever possible) the processes that they all share, starting with the process of *feeding the model*.

4.2.1 Input Pipeline

Beside the bene ts of feeding the models with the TFRecords les, the use of the Tensor ow *tf.data* and *tf.estimator* APIs make it possible to optimize the combined (and asynchronous) use of CPUs and GPUs for extraction/transformation/loading (known as *ETL process*) of the data, as well as the training/validation/prediction of the model, respectively. In this way, we generate an *input pipeline* where the CPU is responsible for the ETL process, which consists of the following steps:

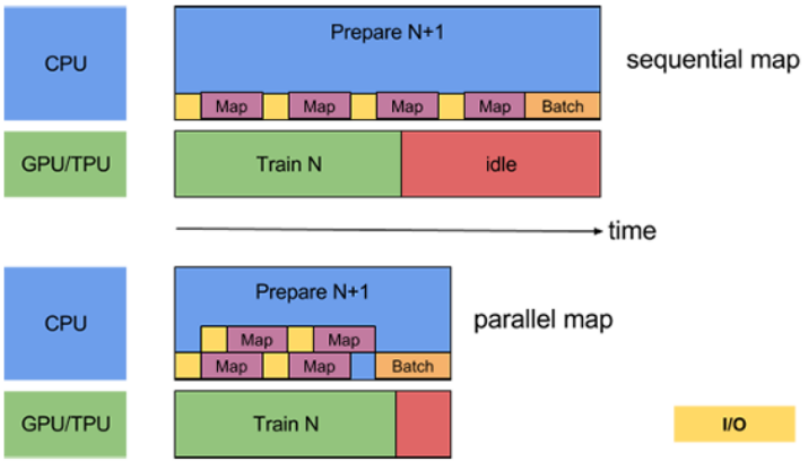


Figure 4.2: Time cost associated between sequential map and parallel map.

Extract: Access the TFRecords les stored on disk of any of the datasets previously generated.

Transform: During training, read a bu er of the given size and randomly shu e it, repeating the data according to the number of epochs (100 in frame prediction task and 50 in classi cation task). In validation, do not shu e the data and only go through the data eleven. Once this is done, the videos are decoded and stored in tensors, rescaling the value ranges from [0, 255] to [0, 1] and taking a single channel (grayscale), all in parallel (see gure 4.2) . Prefetch transformation to decouple the time data is produced by CPU from the time it is consumed by GPU (see gure 4.3).

Load: Feeds the neural network with batches of $BS = 20$ videos in each training step. In addition to the videos, appearance and motion labels are provided.

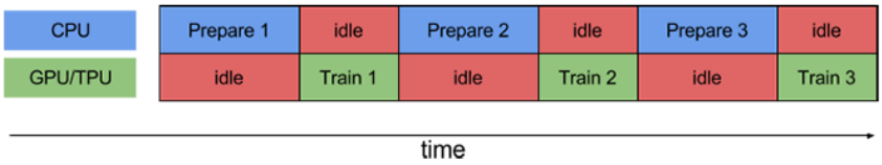
Once the models have been fed, the rst step taken by all of them is to binarize and standardize inputs to take values -1 or 1. The reason for this transformation is because the *loss function* associated to the frame prediction task is obtained through *binary cross entropy* (BCE) between the predicted frame and the ground truth input frame. So each pixel of the predicted image is \classi ed" for the possible values -1 or 1 (in other words, black or white pixels). Given the nature of these datasets, the binarization of the videos does not lead to a drastic change in their representation.

4.2.2 Frame Prediction Models

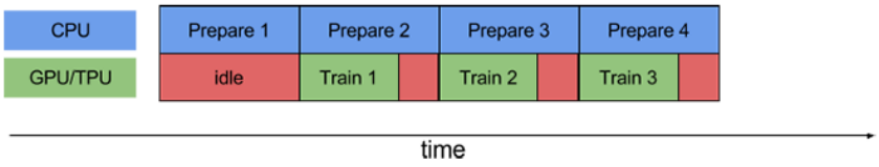
Formally, we are trying to predict future frames given a window of previous ones:

$$Prob\left(x_{t+K} | x_t, x_{t\ \ 1}, \ldots, x_{t\ \ N}\right) \tag{4.2}$$

where this probability is approximated with one of the two aforementioned proposals. Note that we have two hyperparameters: K controls how long into the future we are trying to foresee, whereas N determines the *memory* of the system (i.e. how much previous information we leverage to emit the predictions). This is easily extended to predict individually more than one frame to exploit the video content as much as possible, just sliding the window from $t = N$ to



(a) Without using prefetch



(b) Using prefetch mapping

Figure 4.3: Time cost associated to the use of prefetch transformation.

($t = T \quad K$), where T is the video length. This extension is very useful to speed up and optimize the training process even more.

In our case, we have $N = 10$ and $K = 1$, so that from the previous 10 frames we are able to predict the next one. Given the video length is $T = 20$ frames, the model is able to predict individually 10 frames (from 11^{th} to 20^{th}). Thus, in each step the *loss function* required to optimize the model by gradient descent (*adamOptimizer*), is the averaged loss of the batch where, in turn, the loss of each video is the averaged loss of the 10 predicted frames:

$$loss = \sum_{i=1}^{BS} \sum_{t=N+K}^{T} BCE(\hat{x}_{i,t}, x_{i,t}) \tag{4.3}$$

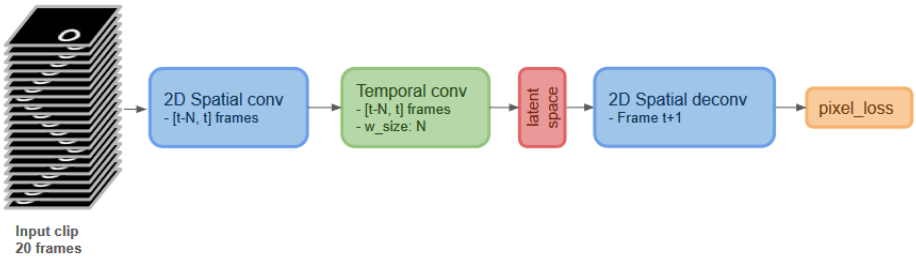
Since vanilla AE and DisNet models were designed to predict the x_{t+K} frame, from now on we call them *vanilla frame predictor* (or *vanilaFP*) and *disentangled frame predictor* (or *DisFP*) respectively, in order to di erentiate them later on from their respective adaptations to the classi cation task.

Both models (vanillaFP and DisFP) follow an auto-encoder style architecture where, in general, the encoder part is responsible for extracting spatial and temporal information from a window of input frames, $(x_t, x_{t\ 1}, \ldots, x_{t\ N})$, while the decoder is responsible for emitting the future frame x_{t+K} from the feature vector extracted by the encoder. Despite their similarities, the architecture and performance of both encoder and decoder are di erent in each model, as shown in gure 4.4

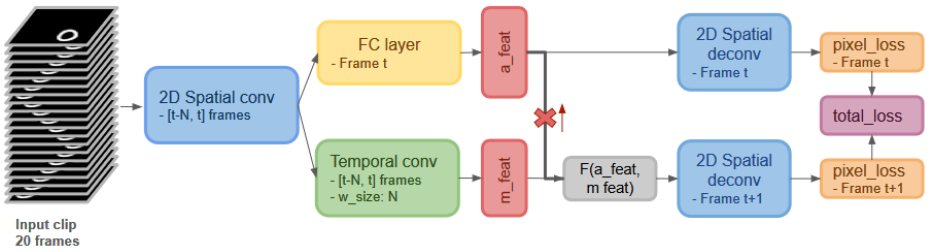
4.2.2.1 Vanilla Frame Predictor

Encoder: It consists of a 2D CNN whose weights are shared across time steps, followed by a 1D causal CNN squashing the N features coming from each frame into a single latent feature vector:

$$e_t = CNN_{1D}\left(CNN_{2D}(x_t), CNN_{2D}(x_{t\ 1}), \ldots, CNN_{2D}(x_{t\ N})\right) \tag{4.4}$$



(a) Vanilla Frame Predictor



(b) Disentangled Frame Predictor

Figure 4.4: Frame Prediction Models.

The 2D CNN acts on the *spatial dimensions*, so at the output we obtain the spatial information of the frames within the window $[t \quad N, t]$. This sub-NN consists of 6 customized layers, where each of them contains a couple of 2D conv layers, the rst with stride = 1 to explore the content of the image and the second with stride = 2 to reduce the dimensionality. Between layers *batch normalization* (BN) is applied and the *residual connections* of the previous layer are added to avoid the *vanishing gradient problem*.

Decoder: This is a 2D CNN that recovers the frame from the latent feature vector, $\hat{x}_{t+K} = CNN_{2D}(e_t)$. This sub-NN reverses the operations performed by the 2D CNN (i.e. 2D spatial deconvolution), so that both the number of layers and Iters are identical. However, to upsampling the tensor dimensions corresponding to the spatial dimensions of the video, the combined use of *NN-Resize + conv2D* layers (stride=1) is used. Both in 1D causal conv and in 2D deconv, BN layers are applied but not residual connections.

4.2.2.2 Disentangled Frame Predictor

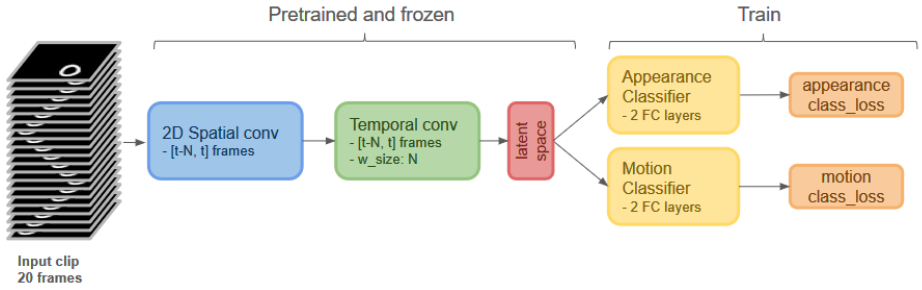
Encoder: For every window of input frames, the encoder emits two feature vectors: one which depends only on the current frame (appearance features, $e_{a,t}$), and another one which depends on the current and previous frames (motion features, $e_{m,t}$). The NNs used to get these two vectors share the spatial 2D CNN as a common backbone:

$$e_t = CNN_{2D}(x_t) \tag{4.5}$$

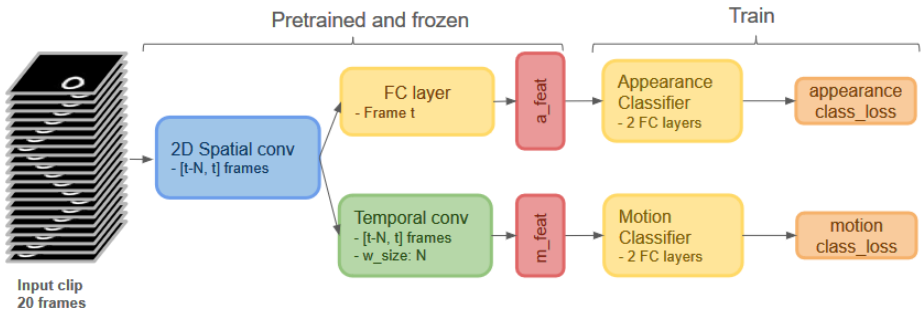
$$e_{a,t} = FC(e_t) \tag{4.6}$$

$$e_{m,t} = CNN_{1D}\left(e_t, e_{t \quad 1}, \ldots, e_{t \quad N}\right) \tag{4.7}$$

CNN_{2D} y CNN_{1D} are identical to those used in vanillaFP encoder. Regarding FC, we have two custom layers composed of *fully connected + dropout layers*.



(a) Vanilla Classi er



(b) Disentangled Classi er

Figure 4.5: Appearance and Motion Classi cation Models.

Decoder: We have a couple of 2D CNN with identical specs to the one contained inside the vanillaFP decoder. The one after the FC is in charge of recovering the frame $\hat{x}_t$ through the current appearance features $e_{a,t}$. The other one recovers the frame $\hat{x}_{t+K}$, through the appearance of future frames estimated from current appearance and motion features: $\hat{e}_{a,t+K} = f(e_{a,t}, e_{m,t})$. Function f could have di erent implementations (e.g. MLP, spatial transformer networks, cross-convolutions, addition or product). In this project the *fusion method* used is multiplicative. In order that the appearance latent space is not conditioned by the future frame prediction, we block the respective gradients ow in the *back propagation* phase (see gure 4.4(b)).

4.2.3 Appearance and Motion Classification Models

Once models have been pre-trained for the frame prediction task, we adapt them so that, taking advantage of the information contained in their latent spaces, they are able to classify both the appearance and motion of the datasets previously used.

To obtain these models (called *vanilla classi er* and *disentangled classi er*) we simply remove the decoder part and, instead, insert two fully connected layers, the rst one with 512 neurons and the second one equal to the number of classes. Since the number of symbols (0, 3) and motions (horizontal, vertical) are two, both the appearance and motion classi ers are binary. Therefore, we require a BCE loss function. This kind of loss function is the one we already use for the frame prediction task, but in this case the computed error is between the logits obtained by the appearance and motion classi ers and their respective ground truth labels.

To ensure the latent spaces are not a ected by the classi cation task, we freeze the pre-trained weights of the encoder, so that the training is only performed on the classi er part (i.e. only the weights of the new fully connected layers are updated). This ne-tuning process on the classi cation task allows to speed up the training process, requiring a smaller number of epochs and data, although in this case the volume of data used is the same as for the pre-training.

The only di erence between both models are the feature vectors that feed each classi er. With vanilla classi er, the same latent feature vector e_t is used to classify appearance and motion. However, with the disentangled classi er, each one is fed with its corresponding feature vector, that is, the appearance classi er with the appearance features $e_{a,t}$ and the motion classi er with the motion features $e_{m,t}$, as shown in gure 4.5.

Chapter 5

Experimental Results

As in many other works, one of the main goals for any researcher is to obtain satisfactory results that justify the time and e ort employed. However, in a eld as complex as deep learning, studying the models behavior and understanding the meaning of their results is as relevant as obtaining good results itself. That is why, during the development of this project numerous experiments were carried out, both to calibrate the correct functioning of the implemented models as well as to verify the proposed hypothesis. These experiments allow us to study and understand how a neural network behaves when is forced to learn disentangled representations, as is the appearance and motion contained in videos.

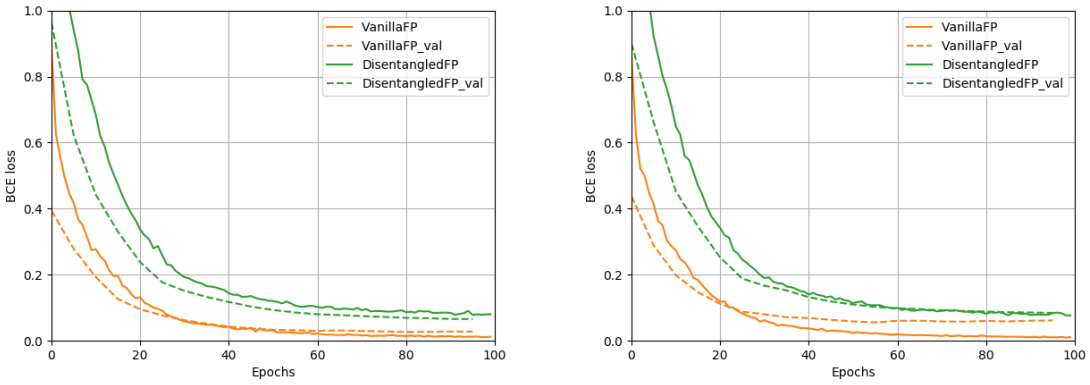
5.1 Frame Prediction Task

In this section we analyze the *quantitative* and *qualitative* results obtained when we trained the nal version of vanillaFP and DisFP models described in section 4.2.2. The experiment consists of predicting the next frame from the observation of the previous 10 frames $[t \quad 10, t]]$, using MovingSymbols2_Seen and MovingSymbols2_NotSeen datasets.

5.1.1 Quantitative Analysis: BCE Loss Function

The quantitative results correspond to the BCE loss function evolution during the training process. This training was carried out for each model in the two previously designed datasets, so that gure 5.1(a) shows the loss curves for the MovingSymbols2_Seen dataset, while gure 5.1(b) shows the corresponding curves for the MovingSymbols2_NotSeen dataset.

In both cases we see that the BCE loss in vanillaFP is lower than in DisFP, both in training and validation. This is because DisFP, apart from learning to predict the frame $t + 1$ (as in VanillaFP), it also learns to reconstruct the current frame, so the total loss function is the sum of two losses, one for each task. Something interesting that happens in VanillaFP is that the validation curve remains at from epoch 50 in MovingSymbols2_NotSeen dataset, while the training curve continues decreasing, generating a gap between both quite evident. This behavior happens when NNs start over tting, which gives us a cue to start thinking that the model is not learning to generalize for appearance-motion combinations not seen during training.



(a) MovingSymbols2_Seen (b) MovingSymbols2_NotSeen

Figure 5.1: Binary cross entropy loss function on both models and datasets.

5.1.2 Baseline: Copy Frame Predictor

In order to delimit the problem and know the maximum error that the models are allowed to make (*reference error*), we generate the simplest possible *baseline*, which does not require prior learning for the task of predicting frames.

The experiment consists of predicting the frame $t+1$ knowing only the frame t. Given that the temporal evolution of the symbol is unknown, we assume that the minimum error we can make is \copying" the values of frame t to the instant $t+1$, understanding that the symbol is static, so that the error corresponds to the BCE between the frame t and the frame $t+1$.

Given the nature of videos from our datasets, in which the motion is constant and linear, the BCE between consecutive frames is the same. Therefore, this experiment gives us an idea of the similarity degree between consecutive frames, so that the error made by the baseline is directly related to the motion speed. It is easy to understand that for a displacement of 1 pixel/frame the di erence between consecutive frames is less than if the displacement of the symbol is 8 pixels/frame (as in our case). Therefore, the motivation to apply such relatively high speed is to evidence the motion contained between frames. In this way we avoid that the network becomes \lazy" and learn to associate that the next state is very simmilar to the current state, so that it does not learn the motion information.

Although the BCE between consecutive frames within one of these videos is the same, we do not get the same error between di erent videos. This is because the overlap of the symbol between consecutive frames depends on the appearance of the symbol itself, (in addition to the motion speed, as we mentioned above). That is why we must average the error of all the videos within the dataset in order to know a more accurate value of this reference error. Within the same subset of videos this error is always the same (no matter how many times it is computed), but given the randomness of the MNIST symbols appearance, this error di er between subsets, although the di erence is not very high, as shown in table 5.1. As we can see the BCE losses obtained by both models are one order of magnitude below the baseline results, so our assumption is con rmed.

	MovingSymbols2_Seen		MovingSymbols2_NotSeen	
	Training	Validation	Training	Validation
Baseline: Copy Frame Predictor	0.35889	0.35841	0.35727	0.36041
Vanilla Frame Predictor	0.01214	0.02780	0.01179	0.06176
Disentangled Frame Predictor	0.08405	0.06602	0.07720	0.08461

Table 5.1: Binary Cross entropy at pixel level. The CopyFP values are the averaged BCE loss for all the dataset videos. In VanillaFP and Disentan edFP the training values are the averaged BCE loss of the video batch in the nal step (after 100 epochs). Validation values of VanillaFP and Disentan edFP are the averaged BCE loss of all the dataset videos after 100 epochs.

5.1.3 Qualitative Analysis: VisualRepresentations

For the qualitative study of the frame prediction task, we analyze how good the reconstruction of the frames has been in terms of the symbols' shape and position along the sequence of frames from 11^{th} to 19^{th}. As explained in the previous chapter, it is very easy to extend the prediction of the next frame for the entire video. This, in addition to optimizing the learning process, allows us to visualize how the reconstruction of the predicted frames evolves, in order to verify that the computed loss was not a ected by the bad reconstruction of a particular frame.

This is how we discovered an error in the last frame of all the videos (20^{th}), which showed how the overlap between the prediction and the ground truth was not as good as in the remaining frames. After analyzing the copyFP representations, we conclude that frame 20 was identical to 19, since in these representations the overlap was perfect. Therefore, the error was caused by a failure in the generation of the videos, which was impossible to detect due to the complexity of the code moving_symbols.py (the only one that was not designed by us). Therefore, the fastest solution was to train the models in the same way we were doing, using all the frames except the last one. That is why the frames represented range from 11^{th} to 19^{th}.

In order to improve the accuracy of its analysis, the RGB channels were used to superimpose in the same frame the ground truth and the prediction of the model. Thus, the prediction of the frame is inserted in the red channel, while the other two resulting channels are occupied by the ground truth (duplicating its values), so that it is represented in cyan color (combination of green and blue). Figure 5.2 shows 4 sequences of frames predicted in validation set corresponding to each model trained for both databases.

We can see how the predictions are fairly accurate to their respective GTs, in terms of position and appearance, being in the case of validating on unseen appearance-motion combinations (rows 3 and 4) where there is not much coincidence. These visual results support those already shown in the table, where the BCE loss of validation is slightly higher in both models.

5.2 Appearance and Motion Classification Task

It is in the classi cation task where we really know if all our e ort dedicated to the design and implementation of both the datasets and the models have served to validate our hypothesis.

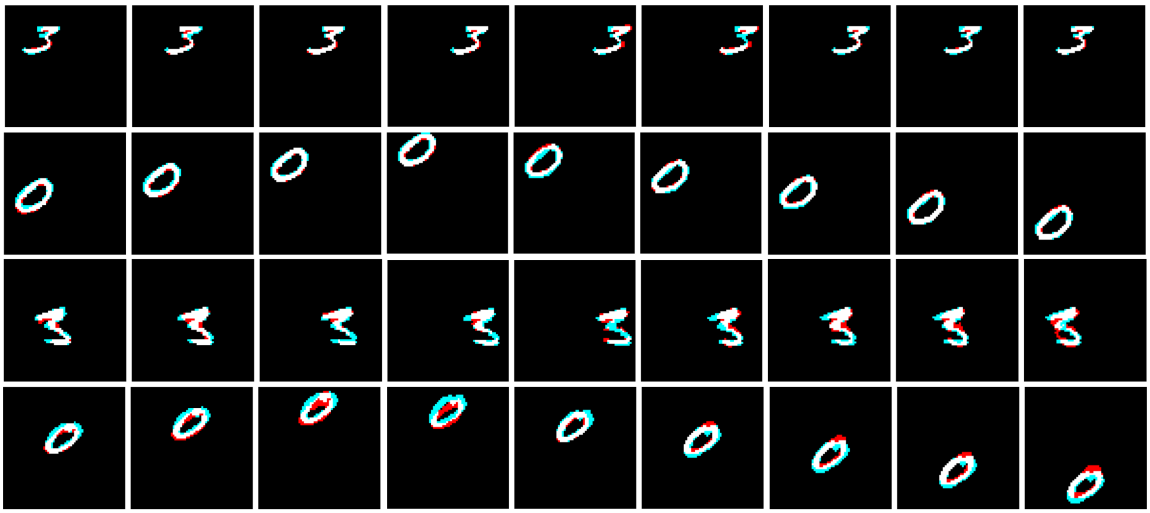


Figure 5.2: Ground truth vs Predicted frames. Each row represents the video sequence from the 11^{th} to the 19^{th} frame, where in each one the prediction is represented in red color and the GT in cyan. The white color indicates the coincidence between both representations (*true positives*). The first and third rows correspond to vanillaFP. The second and fourth rows correspond to disentangledFP. The models in the first two rows are trained with MovingSymbols2_Seen and in the following two with MovingSymbols2_NotSeen.

Before analyzing the classification results, we had to pre-train and adapt the networks, obtaining the *vanilla classifier* and *disentangled classifier models*. The first model is asked to classify with the help of the latent vector e_t both, the appearance (i.e. knowing whether the symbol is 0 or 3) and motion (i.e. knowing whether the symbol moves horizontally or vertically) of the input video. In the case of disentangled classifier, it is firstly asked to classify appearance and motion with its specified latent vectors (i.e. $e_{a,t}$ and $e_{m,t}$ respectively), and then perform the same classification again, but inverting the vectors (i.e. $e_{a,t}$ classifies motion and $e_{m,t}$ classifies appearance). Once the encoder part is frozen and fine-tune is done on the classifier part, we obtain the accuracy values on validation set that are shown in table 5.2.

We see that, while all the possible combinations of appearance and motion are shown during training (MovingSymbols2_Seen), both models classify without any problem (or at least under these controlled conditions). However, despite the simplicity of the dataset, when some of the combinations are not shown during training (MovingSymbols2_NotSeen), the vanilla classifier only gets a 0.02% appearance accuracy, whereas when it classifies motion the accuracy rises up to 99.98%. In fact, we can see how these results are complementary, so that the error rate for motion matches exactly the success rate for appearance.

Since the appearance and motion information is within the same latent space, the model has learned to generalize only for motion, so that the appearance is basically bounded to the learned motion. That is why when the network is asked to classify the appearance of a new input during validation, What it really does is to observe the motion that the symbol performs and, based on that, it assigns to the input the appearance that the network saw performing such motion during the training process. Due to these appearance-motion combinations are never seen during validation, each time the model success the input motion, it fails its appearance and vice versa. Then when there is no disentangling the problem of generalization inevitably appears.

	MovingSymbols2_Seen		MovingSymbols2_NotSeen	
	Appearance	**Motion**	**Appearance**	**Motion**
Vanilla Classi er	99.26	100.00	0.02	99.98
Disentangled Classi er	98.20	100.00	85.97	97.46
Inv. Disentangled Classi er	99.99	65.22	2.60	14.12

Table 5.2: Appearance and motion classi cation accuracy on validation set after training (50 epochs). It shows a comparison between vanilla and disentangled models for the appearance and motion classi cation tasks when there are combinations of appearance-motion seen and not seen during training process.

In the case of disentangled classi er it also happens that the latent space associates the appearance based on the learned motion when we refer to $e_{m,t}$, in the same way that $e_{a,t}$ associates the motion with the learned appearance. This cross-complementarity of the accuracies is also shown in the table, so that when the motion vector fails to classify motion, the appearance is correct and when the appearance vector fails to classify appearance, the motion is systematically correct.

Leaving aside these internal residual associations, we see that our proposal has learned to generalize both appearance and motion in each of its respective latent spaces, obtaining very good results for the case of motion (97.46%), while the appearance, although clearly it is evident that it has learned to classify (85.97%), it does not do it really well. This may be due to a weak adjustment of the part of the network that learns appearance, leaving the improvement of it as future work.

Chapter 6

Budget

Many times, public bodies or private companies are reluctant to support certain research lines for the excessive time it would take to obtain results with practical applications. In many other cases, it is due to the uncertainty that comes with any investigation, where the expected results are not always obtained. Therefore, the required time for the execution of a project is very important when we are evaluating its cost and, often, it is crucial for the viability of the project itself. Then, for the quanti cation of the nancial costs of any project we must take into account three main factors;

Working time

Human resources

Material resources

Based on that, table 6.1 shows an estimate of the nancial cost that the completion of this project entailed. Although the time that I have dedicated for the development of this project has been greater than the carried out as traineeship, in order to estimate the time spent and its associated cost, we will base on it. Thus, during the 6 months I spent as a researcher assistant (HIWI) in the destination research center (DFKI) I registered an amount of 720 hours (6 months x 4 weeks/month x 20h/week), being paid €10/h. Throughout the 9-month duration of this project I had a one-hour weekly meeting with the 3 members of my team (all considered senior engineers), so that the time spent by each of them was 36 hours. The material resources estimation is based on the facilities, materials and services provided by the research center, as well as the computation time of the available hardware resources.

	Amount	**Cost/hour**	**Time**	**Total**
Junior engineer	1	$10.00	480h	$4,800
Senior engineer	3	$30.00	36h	$1,080
Other equipment	-	-	-	$7,000
			Total	$12,880

Table 6.1: Estimated nancial cost of the project.

Chapter 7

Conclusions and Future Development

Thanks to the results obtained and the adequate justi cation based on their analysis, we can conclude that our proposed DisNet model, based on the disentangling of appearance and motion, is able to generalize for appearance-motion combinations not seen during training. In this way, our hypothesis is validated, since we have been able to demonstrate the problem of generalization that conventional models present (as well as to solve it with our proporsal), at the same time we alleviate the problem of scalability, since the learning of appearance and motion has been totally independent.

The e ort and special attention devoted to the creation of a source code of easy to use and understand, allows the possibility for future researchers to continue developing this open research line, which is still in its initial phase. Given the bene ts of a code that uses resources e ciently, while being versatile and scalable at the same time, it is possible to perform more complex tasks without fear of nding inconsistency problems. Below, some of the possible tasks are summarized as a proposal for future development:

Explore di erent architectures that improve the appearance classi cation performance.

Generate datasets of greater complexity and analyze the behavior of the model trained on them.

Use bounding boxes and background labels to extend the learning disentangled representations to foreground, background and motion, as well as to deal with the segmentation task.

Export the experiments to some of the well-known datasets such as KTH or UCF 101 and compare its results with the state-of-the-art.

Pre-train the part of the network that learns appearance through some of the most used image datasets (e.g. ImageNet). In this way, when we train in a video dataset (e.g. UCF101), the network should only learn how that previously learned appearance evolves throughout the video, that is, its motion.

Bibliography

[1] Relja Arandjelovic and Andrew Zisserman. Look, listen and learn. In *ICCV*. IEEE, 2017.

[2] Dana H Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 1981.

[3] Thomas Brox, Christoph Bregler, and Jitendra Malik. Large displacement optical ow. In *CVPR*. IEEE, 2009.

[4] Christopher P Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in *n*beta vae. *arXiv preprint arXiv:1804.03599*, 2018.

[5] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *CVPR*. IEEE, 2017.

[6] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *CVPR*. IEEE, 2005.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*. Ieee, 2009.

[8] Je rey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2015.

[9] Alon Faktor and Michal Irani. Video segmentation by non-local consensus voting. In *BMVC*, 2014.

[10] Lijie Fan, Wenbing Huang, Stefano Ermon Chuang Gan, Boqing Gong, and Junzhou Huang. End-to-end learning of motion representation for video understanding. In *CVPR*, 2018.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[13] Jun-Ting Hsieh, Bingbin Liu, De-An Huang, Li Fei-Fei, and Juan Carlos Niebles. Learning to decompose and disentangle representations for video prediction. *arXiv preprint arXiv:1806.04166*, 2018.

[14] Bela Julesz. Textons, the elements of texture perception, and their interactions. *Nature*, 1981.

[15] Nal Kalchbrenner, Aaron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. *arXiv preprint arXiv:1610.00527*, 2016.

[16] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. *arXiv preprint arXiv:1802.05983*, 2018.

[17] Alex Krizhevsky, Ilya Sutskever, and Geo rey E Hinton. Imagenet classi cation with deep convolutional neural networks. In *NIPS*, 2012.

[18] Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. Variational inference of disentangled latent concepts from unlabeled observations. *arXiv preprint arXiv:1711.00848*, 2017.

[19] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *null*. IEEE, 2006.

[20] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Ha ner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[21] Xunyu Lin, Victor Campos, Xavier Giro-i Nieto, Jordi Torres, and Cristian Canton Ferrer. Disentangling motion, foreground and background features in videos. *arXiv preprint arXiv:1707.04092*, 2017.

[22] David G Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

[23] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

[24] Ishan Misra, C Lawrence Zitnick, and Martial Hebert. Shu e and learn: unsupervised learning using temporal order veri cation. In *ECCV*. Springer, 2016.

[25] Timo Ojala, Matti Pietikainen, and David Harwood. A comparative study of texture measures with classi cation based on featured distributions. *Pattern recognition*, 1996.

[26] Giambattista Parascandolo, Mateo Rojas-Carulla, Niki Kilbertus, and Bernhard Scholkopf. Learning independent causal mechanisms. *arXiv preprint arXiv:1712.00961*, 2017.

[27] Deepak Pathak, Ross B Girshick, Piotr Dollar, Trevor Darrell, and Bharath Hariharan. Learning features by watching objects move. In *CVPR*, 2017.

[28] MarcAurelio Ranzato, Arthur Szlam, Joan Bruna, Michael Mathieu, Ronan Collobert, and Sumit Chopra. Video (language) modeling: a baseline for generative models of natural videos. *arXiv preprint arXiv:1412.6604*, 2014.

[29] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, 2014.

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

[31] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[32] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *ICML*, 2015.

[33] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[34] Ryan Szeto, Simon Stent, German Ros, and Jason J Corso. A dataset to evaluate the representations learned by video prediction models. *arXiv preprint arXiv:1802.08936*, 2018.

[35] Valentin Thomas, Emmanuel Bengio, William Fedus, Jules Pondard, Philippe Beaudoin, Hugo Larochelle, Joelle Pineau, Doina Precup, and Yoshua Bengio. Disentangling the independently controllable factors of variation by interacting with the world. *arXiv preprint arXiv:1802.09484*, 2018.

[36] T Tomita, A Kaneko, M Murakami, and EL Pautler. Spectral response curves of single cones in the carp. *Vision research*, 1967.

[37] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, 2015.

[38] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. In *NIPS*, 2016.

[39] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *ICCV*, 2013.

[40] Tianfan Xue, Jiajun Wu, Katherine Bouman, and Bill Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *NIPS*, 2016.

[41] Ming-Hsuan Yang, David J Kriegman, and Narendra Ahuja. Detecting faces in images: A survey. *TPAMI*, 2002.

[42] Thomas Young. Ii. the bakerian lecture. on the theory of light and colours. *Philosophical transactions of the Royal Society of London*, 1802.

[43] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classi cation. In *CVPR*, 2015.