UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Representation learning of spatio-temporal features from video

Gabriel de Barros Paranhos da Costa

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP
SÃO CARLOS

Gabriel de Barros Paranhos da Costa

Representation learning of spatio-temporal features from video

Thesis submitted to the Institute of Mathematics and Computer Sciences – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Doctor in Science. *EXAMINATION BOARD PRESENTATION COPY*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Moacir Antonelli Ponti
Co-advisor: Prof. Dr. Rodrigo Fernandes de Mello

USP – São Carlos
August 2019

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

P223r

Paranhos da Costa, Gabriel de Barros
Representation learning of spatio-temporal features from video / Gabriel de Barros Paranhos da Costa; orientador Moacir Antonelli Ponti; coorientador Rodrigo Fernandes de Mello. -- São Carlos, 2019.
169 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2019.

1. Aprendizado de características. 2. Extração de características. 3. Aprendizado profundo. 4. Aprendizado de máquina. 5. Características espaço-temporais. I. Ponti, Moacir Antonelli, orient. II. de Mello, Rodrigo Fernandes, coorient. III. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Gabriel de Barros Paranhos da Costa

Aprendizado de características espaço-temporais em vídeos

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *EXEMPLAR DE DEFESA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Moacir Antonelli Ponti
Orientador: Prof. Dr. Rodrigo Fernandes de Mello

USP – São Carlos
Agosto de 2019

ACKNOWLEDGEMENTS

I would like to thank the *Fundação de Amparo a Pesquisa do Estado de São Paulo* (FAPESP) and the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES) for the opportunity to have the required funding to execute the research during my PhD. I am also grateful for all the assistance offered by the *Instituto de Ciências Matemáticas e Computação* (ICMC-USP). A special thanks to my supervisor Moacir A. Ponti for all the support and guidance during all these years working together. Finally, I am very grateful for all my friends and family, that encouraged me during my entire PhD.

“The way I see it, every life is a pile of good things and bad things.
The good things don’t always soften the bad things, but vice versa,
the bad things don’t necessarily spoil the good things
or make them unimportant ”
(The Doctor)

RESUMO

PARANHOS DA COSTA, G. B. **Aprendizado de características espaço-temporais em vídeos**. 2019. 169 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2019.

Um dos principais desafios em visão computacional é codificar as informações presentes em imagens e vídeos em um vetor de características que depois pode ser utilizado, por exemplo, para treinar um modelo (aprendizado de máquina). Vídeos incluem um desafio a mais, uma vez que tanto informações espaciais quanto temporais precisam ser consideradas. Para reduzir a necessidade da criação de novos métodos de extração de características, métodos de aprendizado de características buscam criar representação diretamente a partir dos dados; esses métodos obtiveram resultados no estado da arte em diversas tarefas de visão computacionais baseadas em imagens. Por esses motivos, o aprendizado de características espaço-temporais a partir de vídeos é considerado como um próximo passo natural. Apesar de diversas arquiteturas terem sido propostas com esse objetivo, os resultados obtidos por esses métodos, quando aplicados a vídeos, são semelhantes aos obtidos pelos métodos tradicionais e apresentaram vantagens consideravelmente inferiores do que em aplicações focadas em imagens. Nós acreditamos que para encontrar melhorias na área de aprendizado de características espaço-temporais é necessário obter um maior conhecimento sobre como as informações são codificadas por esses métodos, permitindo a tomada de decisão mais bem informada sobre quando cada arquitetura deve ser usada. Com esse fim, nós propomos um novo protocolo de avaliação que utiliza um problema sintético em três diferentes configurações onde a informação relevante para a tarefa aparece somente nas dimensões espaciais, na dimensão temporal ou em ambas. Nós também investigamos as vantagens de se utilizar um método de aprendizado de características ao invés de características projetadas manualmente, em especial com relação ao seu uso em diferentes tarefas. Então, nós propomos um método de regularização baseado em redes generativas e transferência de conhecimento como forma de melhorar o espaço de características obtido por métodos de aprendizado de características. Os resultados mostram que quando realizando aprendizado de características espaço-temporais é importante incluir a informações temporal durante todos os estágios. Também notamos que apesar das arquiteturas que utilizam convolução na dimensão temporal obterem os melhores resultados dentre as arquiteturas testadas, essas têm dificuldade para se adaptar a mudanças na informação temporal. Quando comparando o desempenho de características manualmente projetadas e de características aprendidas a partir dos dados, as primeiras obtiveram resultados superiores na tarefa para o qual foram projetadas, mas seu desempenho cai significativamente em outra tarefa, obtendo desempenho inferior nesse caso. Finalmente, nós mostramos que redes generativas possuem em transferência de conhecimento uma promissora aplicação, apesar de ser necessário expandir a análise para incluir características

espaço-temporais.

Palavras-chave: Aprendizado de características, Extração de características, Aprendizado profundo, Aprendizado de máquina, Visão computacional, Processamento de vídeos.

ABSTRACT

PARANHOS DA COSTA, G. B. **Representation learning of spatio-temporal features from video**. 2019. 169 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2019.

One of the main challenges in computer vision is to encode the information present in images and videos into a feature vector that can later used, for example, to train a machine learning model. Videos include an extra challenge since both spatial and temporal information need to be considered. To address the challenges of creating new feature extraction methods, representation learning focuses on creating data-driven representations directly from raw data; these methods achieved state-of-the-art performance on many image-focused computer vision tasks. For these reasons, spatio-temporal representation learning from videos is considered a natural next step. Even though multiple architectures have been proposed for video processing, the results obtained by these methods when applied to videos are still akin to the ones obtained by hand-crafted feature extraction methods and reasonably bellow the advantages obtained by representation learning on images. We believe that to advance the area of spatio-temporal representation learning, a better understanding of how the information is encoded by these methods is required, allowing for more knowledgeable decisions regarding when each architecture should be used. For this purpose, we propose a novel evaluation protocol that looks at a synthetic problem in three different settings where the relevant information for the task appears only on spatial dimensions, temporal dimension or both. We also investigate the advantages of using a representation learning method over hand-crafted feature extraction, especially regarding their use on different (previously unknown) tasks. Lastly, we propose a data-driven regularisation method based on generative networks and knowledge transfer to improve the feature space learnt by representation learning methods. Our results show that when learning spatio-temporal representations it is important to include temporal information in every stage. We also notice that while architectures that used convolutions on the temporal dimension achieved the best results among the tested architectures, they had difficulties adapting to changes in the temporal information. When comparing the performance of hand-crafted and learnt representations on multiples tasks, hand-crafted features obtained better results on the task they were designed for, but considerably worst performance on a second unrelated task. Finally, we show that generative networks have a promising application on knowledge transfer, even though further investigation is required in a spatio-temporal setting.

Keywords: Representation learning, Feature extraction, Deep learning, Machine learning, Computer vision, Video processing.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

Adaline	Adaptive Linear Neuron
BEGAN	Boundary Equilibrium GAN
BOF	Bag-of-Features
BOV	Bag-of-Visual-Words
BOW	Bag-of-Words
C3D	3D Convolutional Network
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DCGAN	Deep Convolutional GAN
DNN	Deep Neural Network
DT	Dense Trajectories
EBGAN	Energy Based GAN
FV	Fisher Vector
GAN	Generative Adversarial Network
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
HOF	Histogram of Optical Flow
HOG	Histogram of Oriented Gradients
IDT	Improved Dense Trajectories
LDA	Linear Discriminant Analysis
LMS	Lest Mean Squares
LSTM	Long Short-Term Memory
MBH	Motion Boundary Histogram
MC	Mixed Convolutional Network
MLE	Maximum Likelihood Estimate
MLP	Multilayer Perceptron
MMD	Maximum Mean Discrepancy
P3D	Pseudo-3D
PCA	Principal Component Analysis
RANSAC	Random Sample Consensus
ReLU	Rectified Linear Unit

rMC	“Reversed” Mixed Convolutional Network
RNN	Recurrent Neural Network
SFV	Spatial Fisher Vectors
SIFT	Scale-Invariant Feature Transform
SPM	Spatial Pyramid Matching
STIP	Spatio-Temporal Interest Point
STPM	Spatial-Temporal Pyramid Matching
SURF	Speeded-Up Robust Features
SVM	Support Vector Machine
TSN	Temporal Segment Network
tSNE	t-Distributed Stochastic Neighbour Embedding
VC-dimension	Vapnik–Chervonenkis Dimension
WGAN	Wasserstein GAN

CONTENTS

CHAPTER

1

INTRODUCTION

Videos are spatio-intensity patterns that vary with time, which are also commonly seen as time sequences of static images. In recent years, with the development of on-demand video platforms and the cheapening of image and video capture cameras, the availability of visual data has increased at an accelerated pace. For example, only in the Youtube service, 400 hours of video are stored per minute [1]. Due to the amount of data being created, computers need to make sense of it all so that it can be presented to the users in an orderly manner. To achieve this, machine learning algorithms try to identify patterns in the available data to power broadly used aspects of modern society such as web searching, content filtering on social networks and recommendations on e-commerce websites (LECUN; BENGIO; HINTON, 2015).

Video processing and analysis has been a topic of interest in machine learning and pattern recognition for years focusing on many different problems and tasks, such as action recognition (TRAN *et al.*, 2018), action localisation (SHOU; WANG; CHANG, 2016), anomaly detection (XU *et al.*, 2017), scene recognition (QIU; YAO; MEI, 2017), among others. One of the main difficulties when processing videos is to deal with their high dimensional and spatio-temporal nature. Each frame, in principle, can be seen as a static image containing visual (spatial) information. This simple fact makes the task of video processing computationally expensive even when processing short video clips, given that it may contain a high number o images. Moreover, since a dynamic between spatial content of consecutive frames exists, this creates a temporal dimension. In most cases, the spatial content changes slowly with time (temporal coherence), but there can also be abrupt transitions. How to describe both spatial and temporal information to understand a video's content is still a matter of investigation and one of the main limitations encountered when applying machine learning techniques to video-related tasks.

Originally, when working with images and videos, the most commonly used descriptors were based on the associated textual information, such as keywords (CHENG; DALE; LIU,

[1] According to Business Insider, 07/11/2017. Source: <https://www.businessinsider.com/viewers-find-objectionable-content-on-youtube-kids-2017-11>

2008). However, there is no guarantee that this meta-data accurately describes the visual data and often produced unreliable results. To solve this problem, feature extraction methods were proposed to encode visual information into feature vectors. Initially, these methods focused on describing static images by encoding visual information such as colour, shape and texture. These methods were also applied to videos but did not consider any type of temporal information. Currently, the standard approach used to extract features from raw videos considers each frame as a static image, where it detects interest points and then applies classical static image feature extraction methods, such as SIFT (LOWE, 1999) or HOG (DALAL; TRIGGS, 2005), in a region surrounding each of these points (KARPATHY *et al.*, 2014).

A similar idea was used to incorporate stronger temporal relation between frames into the descriptor (WANG; SCHMID, 2013), in which optical flow was used to densely estimate the trajectories of pixels in a sequence of images. These trajectories define where classical feature extraction methods are used. Some of these classical feature extraction methods were also adapted to incorporate temporal information. For example, the SIFT and HOG methods were both adapted so they could be applied to action recognition tasks. These adaptations are called SIFT-3D (SCOVANNER; ALI; SHAH, 2007) and HOG3D (KLASER; MARSZAŁEK; SCHMID, 2008).

Conventional machine learning methods rely on good feature extraction techniques to learn relevant information about the data. Designing a good feature extraction method is complicated and requires careful engineering and vast knowledge of the domain, which makes it expensive and time-consuming. Also, most feature extraction methods are task-specific, which causes new tasks, or even new datasets, to require the development of new techniques. This fact lead researches towards *representation learning* methods (BENGIO; COURVILLE; VINCENT, 2013). Representation learning focuses on allowing computers to automatically discover relevant representations needed for detection or classification directly from raw data (LECUN; BENGIO; HINTON, 2015). This is currently even more relevant since it allows machine learning methods to take advantage of the high quantities of available data.

These methods have recently gained prominence in both the academy and the industry due to the results achieved by *deep learning* techniques. These techniques use a hierarchy with multiple levels of representations and create more abstract representations at higher levels of the hierarchy by combining simpler representations from lower layers (GOODFELLOW; BENGIO; COURVILLE, 2016). This approach has successfully achieved state-of-the-art results in many different areas, specially in computer vision and artificial intelligence (MNIH *et al.*, 2015), in tasks such as image classification (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), object (SZEGEDY *et al.*, 2014) and speech (HINTON *et al.*, 2012) recognition. The use of deep learning concepts for video processing is considered the natural next step for the area, due to the success achieved in static image applications (LÄNGKVIST; KARLSSON; LOUTFI, 2014).

In this context, there are many ways to create end-to-end deep learning models that

leverage spatial and temporal information from videos. Some of the most common approaches use either convolutions to process both spatial and temporal information, or a combination of convolutional and recurrent layers to process spatial (visual) and temporal information, respectively. Another way of approaching this problem is by using a separate network for each information type, usually both convolutional, which process traditional images and temporal information encoded into images by using optical flow. However, most of the proposed methods model short temporal information, using fixed-size windows containing a small number of consecutive frames. The main disadvantage of modelling a short video clip is that each time window only contains a small amount of the information present in the full video. This may result in a feature vector that describes little more than when analysing individual frames (NG *et al.*, 2015). The use of fixed-size windows also makes it difficult to work with real-world datasets, where video length is highly variable and forces the use of quantisation techniques to obtain a fixed-size length feature vector.

Even though these architectures are being used for many different applications, little is known about the impact of the choice of each architecture, which are their strong and weak points and whether one architecture is more suitable for a specific domain. Also, to the best of our knowledge, there are no studies that investigate how these architectures encode spatial and temporal information and if they can generalise when there are small changes to one of these. Furthermore, it is important that these methods learn to extract generic video representations since it would help solve many different computer vision tasks in a homogeneous way (TRAN *et al.*, 2015). As far as we know, only one spatio-temporal representation learning technique was proposed for a generic setting. This method (TRAN *et al.*, 2015) uses a three dimensional Convolutional Neural Network (CNN) to capture both spatial and temporal information simultaneously. However, in addition to using a small fixed-size window, this method was not able to overcome traditional CNNs when applied to datasets with a small number of videos. Also, the resolution of the inputs had to be reduced due to the computational cost of the algorithm.

The experiments and results presented in this document provide new insight and evidence on how to approach the problem of extracting spatio-temporal descriptors from videos. In order to better evaluate representation learning methods, we present a novel evaluation framework that leverages a set of datasets based on the BouncingMNIST dataset (SRIVASTAVA; MANSIMOV; SALAKHUDINOV, 2015) designed to address different settings of a spatio-temporal problem. As far as we know, there is no previous work that explicitly investigates the behaviour of deep learning architectures encoding spatio-temporal information from videos. We also investigate how deep learning architectures compare to state-of-the-art hand-crafted feature extraction methods when applied to a setting where there is not enough data on the target dataset to train the network. To tackled this problem a larger non-related dataset is used to train the network, which is then used to extract features from the target dataset. Lastly, as a parallel work, we investigate the use of generative neural networks as a knowledge transfer method that takes advantage of models created to address similar tasks and creates a task-specific regulariser.

Our results show that, even though a deep network is currently responsible for the best results on action recognition (I3D (CARREIRA; ZISSERMAN, 2017)), there are lots of intrinsic information in videos that are not modelled by current spatio-temporal representation learning architectures. Little is known about how each of these architectures deals with video data and which are the architectures that should be used in each setting or task. We believe our results take a step in the direction of increasing our understanding. Additionally, we show that generative networks can be used for knowledge transfer, guiding training procedures and helping representation learning methods overcome the lack of labelled data, while also decreasing the amount of time required to train a new model. Moreover, since many representation learning methods depend directly on their initial condition to achieve good results, the proposed generative network model for knowledge transfer can also provide a better initial condition.

1.1 Hypothesis

Let $x \in \mathbb{R}^{c \times w \times h \times t}$ be a video with duration t (temporal dimension – number of frames), resolution $w \times h$ and c channels (e.g. $c = 3$ for RGB videos, $c = 1$ for greyscale) (spatial dimensions). Representation learning algorithms try to find a function that maps each input into a n dimensional space optimised based on a loss function $\mathscr{L}$, i.e. $f(x) : x \rightarrow \hat{x}$, where $\hat{x} \in \mathbb{R}^n$. These representation learning methods are typically used in a stacked organisation to create deep learning architectures, on which backpropagation is used so that a single loss $\mathscr{L}$ is used to optimise multiple representation learning stages.

Since representation learning via deep learning methods lead to state-of-the-art results in many computer vision tasks that focus on analysing static images, it is only natural to extend them to analyse videos. When dealing with video inputs, we believe that $f(x)$ should consider both temporal and spatial dimensions to find the feature space where $\mathscr{L}(f(\hat{\mathbf{x}}))$ is minimal. In this context, we hypothesise that by having an adequate protocol that allows for better evaluation of how spatial and temporal information is encoded by a representation learning algorithm, we will be able to identify which architectures are more appropriate for a given task. We also believe that representation learning algorithms provide representations that more general than the ones extracted by hand-crafted feature extraction methods, and that the learnt representations can be used for different applications without the need to retrain or redesign while maintaining a reasonable performance level. Lastly, we conjecture that representation learning algorithms can benefit from data-driven regularisation and knowledge transfer.

1.2 Publications

1.2.1 Accepted or published papers (descending chronological order)

- Ponti, M. A.; Paranhos da Costa, G. B.; Santos, F. P.; Silveira, K. U. *Supervised and Unsupervised Relevance Sampling in Handcrafted and Deep Learning Features obtained from Image Collections*. In: Applied Soft Computing, 2019.

- Nazare, T. S.; Paranhos da Costa, G. B.; de Mello, R. F.; Ponti, M. A. *Color quantization in transfer learning and noisy scenarios: an empirical analysis using convolutional networks*. In: SIBGRAPI 2018 - Conference on Graphics, Patterns and Images, 2018, Foz do Iguaçu - PR - Brazil.

- Ponti, M.; Paranhos da Costa, G. *Como funciona o Deep Learning*. In: Tópicos em Gerenciamento de Dados e Informações, 2017.

- Nazare, T.; Paranhos da Costa, G.; Contato, W.; Ponti, M. *Deep convolutional neural networks and noisy images*. In: Iberoamerican Conference on Pattern Recognition (CIARP), 2017.

- Ponti, M.; Chaves, A.; Jorge, F. R.; Paranhos da Costa, G. B.; Coulturato, A.; Branco, K. R. J. C. *Precision Agriculture: Using Low-Cost Systems to Acquire Low-Altitude Images*. In: IEEE Computer Graphics and Applications, 2016.

- Contato, W. A.; Nazaré, T. S.; Paranhos da Costa, G. B.; Ponti, M.; Batista Neto, J. E. S. *Improving Non-Local Video Denoising with Local Binary Patterns and Image Quantization*. In: SIBGRAPI 2016 - Conference on Graphics, Patterns and Images, 2016, São José dos Campos - SP - Brazil.

- Paranhos da Costa, G. B.; Contato, W. A.; Nazaré, T. S.; Batista Neto, J. E. S.; Ponti, M. *An empirical study on the effects of different types of noise in image classification tasks*. In: WVC 2016 - XII Workshop de Visão Computacional, 2016, Campo Grande - MS - Brazil.

1.2.2 Papers in writing or review stage

- Paranhos da Costa, G. B.; Ponti, M. A. *Learning spatio-temporal representations from video data*. Submitted to Pattern Recognition

- Ribeiro, L. S. F.; Paranhos da Costa, G. B.; Ponti, M. A. *TOPGAN: triplet optimised generative adversarial networks*.

1.3 Document organisation

The remainder of this document is organised as follows:

Chapter 2 introduces basic concepts and techniques relevant to the methods and experiments explained in other chapters;

Chapter 3 presents state-of-the-art representation learning and feature extraction methods used to acquire spatio-temporal descriptors from video;

Chapter 4 establishes a novel spatio-temporal representation learning evaluation framework that allows for better understanding on how each method encodes information present in videos and highlights the pros and cons of several deep learning architectures selected for these experiments;

Chapter 5 compares the ability of state-of-the-art hand-crafted feature extraction and a deep learning architecture to generalise the information learnt to other datasets and domains;

Chapter 6 describes a novel regularisation technique based on generative neural networks that try to model the distribution of known classifiers to leverage their knowledge to similar tasks;

Chapter 7 summarises the results presented in previous chapters and discusses their meaning and impact on spatio-temporal representation learning from video.

CHAPTER

2

FUNDAMENTAL CONCEPTS

2.1 Opening remarks

Most machine learning methods are limited by their ability to process real-world data in their raw form. Careful engineering and domain knowledge are necessary so that the raw data can be transformed into a relevant set of features, i.e. mapped into a feature space, in order to serve as input to machine learning techniques.

One of the greatest problems in this context is the definition of which features are relevant for the computer to be able to understand real-world abstractions (GOODFELLOW; BENGIO; COURVILLE, 2016). Although domain-specific knowledge can be used and would allow the development of features that are relevant to an application, using such an application-specific effort is expensive and time costly.

This chapter presents key machine learning and feature extraction concepts used to address the problem of generating descriptors based on images and videos. These concepts were used to design the methods and experiments presented in the following chapters of this document.

2.2 Representation learning

One of the possible solutions to the problem of extracting relevant information from data is to use models that are able to not only map representations to concepts but also to learn the representations themselves. This is the goal of a research area known as *feature learning* or *representation learning* (BENGIO; COURVILLE; VINCENT, 2013). Features learned, in general, achieve better results than the ones obtained from features that were designed manually, which are called hand-crafted representations (GOODFELLOW; BENGIO; COURVILLE, 2016). Another advantage of using representation learning is that it allows artificial intelligence systems

to adapt rapidly to a new task, reducing the need of human intervention, since most of the time features extracted by using representation learning can be generalised to many different applications.

Bengio, Courville and Vincent (2013) proposed some general-purpose priors that should be taken into consideration when proposing feature extraction or representation learning methods. These priors were created in an attempt to describe certain rules followed by the real world and they are not task-specific. Some of these priors are:

- **Smoothness** – if $x \approx y$ then $f(x) \approx f(y)$, f being the function to be learned;
- **Multiple explanatory factors** – the distribution of the data is composed by different underling factors. Most of what is learned about one of these factors can be generalised to other factors;
- **Hierarchical organisation of explanatory factors** – concepts that describe the world can be combined to create other, more abstract, concepts;
- **Semi-supervised learning** – a subset of the factors that explain the distribution of the data also explains a lot about their classes, given the data;
- **Shared factors across tasks** – multiple tasks can be explained by similar factors;
- **Manifolds** – probability mass concentrates on regions that have smaller dimensionality then the data's original space;
- **Natural clustering** – different values of categorical variables can be associated to separate manifolds;
- **Temporal and spatial coherence** – consecutive or spatially nearby observations tend to be associated with the same value of categorical variables or a small move on the surface of a high-density manifold;
- **Sparsity** – only a small fraction of the possible factors are relevant when representing a given observation;
- **Simplicity of factor dependencies** – good high-level representations use linear dependencies to connect factors.

2.3 Artificial neural networks

The name "artificial neural network" originates from attempts to represent information processing in biological systems using mathematical formulations (BISHOP, 2006). Ever since, this term has been broadly used to address many different models, even for some whose biological

plausibility has been widely contested. In pattern recognition and machine learning context, biological plausibility would impose unnecessary constraints on the models, when the focus is directed to creating models that efficiently performs statistical pattern recognition. Thus, the "neural network" term in machine learning does not guarantee that the model is biologically realistic.

Neural networks focus on solving problems that depend on subtle factors that can not be incorporated into an algorithm (KRIESEL, 2007). To achieve this, these methods try to mimic the capability of biological neural networks to adapt and learn the best ways to deal with such complex problems. An artificial neural network is composed of a pool of simple processing units that communicate through signals passed over a large number of weighted connections (KRÖSE *et al.*, 1996). The most commonly used processing units are known as the *Perceptron* model (ROSENBLATT, 1958) and the Adaptive Linear Neuron (Adaline) model (WIDROW; HOFF *et al.*, 1960).

2.3.1 Processing unit

The Perceptron model, proposed by Rosenblatt (1958), was the first processing unit model for artificial neural networks known. This model uses linear thresholds to achieve a binary output. The second processing unit model, Adaline (WIDROW; HOFF *et al.*, 1960), works similarly to the Perceptron model, though it produces a linear output without applying a threshold. Both of these processing units use a variant of a delta rule for their training procedure (HENSELER, 1995).

Given a vector $\mathbf{x} \in \mathbb{R}^n$ as input, the Perceptron model generates a linear binary classifier that searches the feature space for a hyperplane that divides that feature space into two possible classes, with the objective of minimising training error. This is done by gradually adjusting the weights $\mathbf{w} \in \mathbb{R}^n$ and bias $b \in \mathbb{R}$ of the model $f(\cdot)$ (Equation (2.1)) by following a descending gradient. With each update to the weights and bias, the model tries to reduce the number of errors committed in a set of samples called training set. Figure 1 shows an example of classifier found by the Perceptron model.

$$f(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{w}\mathbf{x} + b > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

This learning procedure takes the form of a *delta rule* when written in a mathematical form (HENSELER, 1995). Defining the updates to the weights $\mathbf{w}$ and the bias b as $\Delta\mathbf{w}$ and Δb, their respective delta rules can be written as Equations (2.2) and (2.3), where $f(\mathbf{x})$ is the output of the current classifier given $\mathbf{x}$ as input and Y is the ground truth label of $\mathbf{x}$.

$$\Delta w_i = -(f(\mathbf{x}) - Y)x_i \tag{2.2}$$

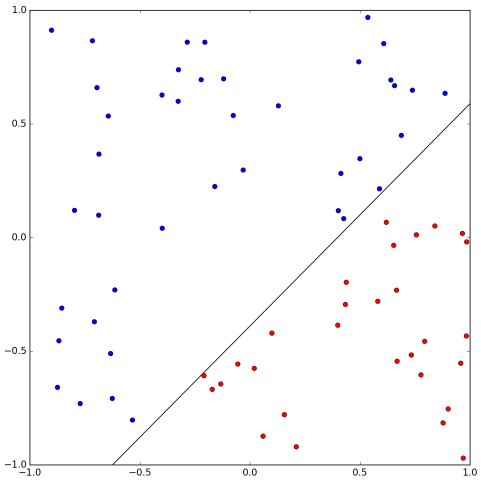


Figure 1 – Example use the Perceptron model in a random set of points.

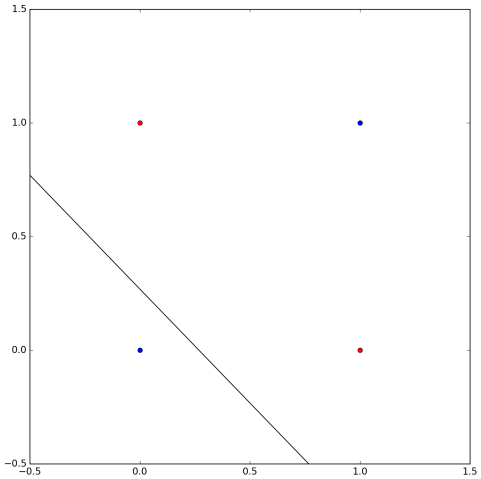


Figure 2 – Example result of the Perceptron model applied to the XOR problem. By looking at the feature space, it is possible to see that there is no possible way of solving the XOR problem using a single linear hyperplane. Different colours indicate different target outputs.

$$\Delta b = f(\mathbf{x}) - Y \tag{2.3}$$

In 1969, after a publication by Minsky and Seymour (1969), the research area that covered the Perceptron model was stuck since it was proven that it could not deal with problems similar to the Exclusive-OR (XOR), shown in Figure 2. These problems were shown to be unsolvable by any of the classifiers created by the Perceptron model, which can only solve linearly separable problems.

By adding a third feature to the XOR problem that consists of the *logical-and* function (AND), it would be possible to solve the XOR problem using the Perceptron model. This could be done using an intermediate neuron that computes this input feature. However, by doing so, it invalidates the learning procedure used by the Perceptron model, since there is no target output for this intermediate unit. This intermediate neuron was called *hidden* due to not having a direct

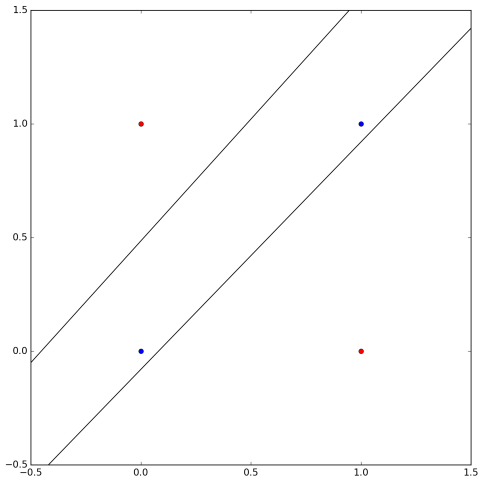


Figure 3 – Example solution to the XOR problem obtained by a feedforward neural network. Different colours indicates different target outputs.

connection to the network's output (HENSELER, 1995).

Gains in computers processing capability and the development of the *generalised delta rule* (WERBOS, 1974; RUMELHART; HINTON; WILLIAMS, 1988) brought new light into this research area and allowed the introduction of an intermediate layer of adaptive connections enabling the method to address previously unsolvable problems.

2.3.2 Feedforward networks

The generalised delta rule (WERBOS, 1974; RUMELHART; HINTON; WILLIAMS, 1988) eliminates the problem faced by the Perceptron when dealing with XOR-like problems by replacing the target error of hidden neurons with the error gradient in the Lest Mean Squares (LMS) procedure. This allowed the formulation of the so-called *feedforward networks*, or Multilayer Perceptron (MLP), quintessential artificial neural network models (GOODFELLOW; BENGIO; COURVILLE, 2016). Figure 3 shows a possible solution to the XOR problem obtained by using a feedforward neural network.

The goal of a feedforward network is to approximate a function $f^*(\cdot)$ which maps an input $\mathbf{x} \in \mathbb{R}^n$ to a class or category $y \in \mathscr{C}$, where $\mathscr{C}$ is the set of all possible classes of that application. This mapping $y = f(\mathbf{x}, \mathbf{w})$ is defined by the network, who learns the values of the parameters $\mathbf{w} \in \mathbb{R}^n$ by minimising an error function based on the LMS procedure. This error function varies depending on if the processing unit is part of a visible or hidden layer.

The "feed-forward" term comes from the fact that information flows through the network connections from the input $\mathbf{x}$, passing through calculations that define f and reaching the output y. This is done without any connection being used to give feedback to previous layers. By including feedback connections the model used changes to a RNN model, which is explained in Section 2.6.

The *back-propagation* algorithm (RUMELHART; HINTON; WILLIAMS, 1988) is used to train the network. It computes the gradients of the error function, allowing the parameters w to be adjusted in the search for the minimum error. This algorithm uses partial derivatives of the cost function, with respect to each weight, to know how quickly the cost will change according to changes made to each weight. It also gives an intuitive interpretation of how the changes made to the weights will affect the overall behaviour of the network (NIELSEN, 2015).

2.4 Deep learning

In 2006, deep learning or hierarchical learning emerged as one of the main research areas involving machine learning (HINTON; OSINDERO; TEH, 2006; BENGIO, 2009). With the rise of the processing capabilities of computers and the growth in the availability of data for training, techniques that were developed using deep learning concepts had a big impact on many research areas, like signal and information processing (DENG; YU, 2014). These techniques aim to make computers learn through experience and to understand the world through a hierarchy of concepts, in which each concept is defined by the combination of multiple simpler ones.

Even though there is no formal definition of deep learning, all approaches in the area have as a central idea: the use of nested representations of data. Each approach may define depth differently, sometimes referring to the number of steps in a graph that describe the computations necessary to produce the final representation or the minimum number of connections between representations before achieving the final results.

The main goal of deep learning is to avoid the difficulties of extracting high-level abstract features from raw data by using layers of features that are not designed by human engineers but learned from data using general-purpose learning procedures (LECUN; BENGIO; HINTON, 2015). This is done by taking advantage of rises in the amount of available data and computational capabilities. Deep learning methods are representation learning methods with multiple levels of representation, which are obtained by composing simple non-linear units that transform representations starting from the raw input into more abstract representations at higher levels. Many different deep learning architectures can be used in a supervised or an unsupervised setting. The currently most used architectures are Convolutional Neural Network (Section 2.5) and Recurrent Neural Network (Section 2.6).

2.5 Convolutional networks

CNNs are specialised in processing data whose topology is known and that has a grid-like organisation (GOODFELLOW; BENGIO; COURVILLE, 2016). Examples of data that present the required characteristics include short time series, that can be represented by a one-dimensional grid where each sample is obtained after a fixed time window, and images, two-dimensional

grids of pixels.

When applied to a coloured image, the input of the convolutional neural network is composed of a set of matrices that contain the colour channels of the image. Each of these colour channels is called *feature maps*. Feature maps are also the name used to refer to the input and output of each layer of a Convolutional Neural Network. Each feature map represents a specific feature extracted from each position of the input.

CNNs aim to simulate a set of cells similar to the ones present in the visual cortex, based on the model proposed by Hubel and Wiesel (HUBEL; WIESEL, 1968), where each cell is responsible for a region of the visual field, called receptive fields. Receptive fields are organised to cover all the visual field. Each cell works like a local filter over the input, exploring local correlations present in natural images. In Hubel and Wiesel (1968), two types of cells were identified: simple cells, that have a maximum response when the stimulus is similar to a border, and complex cells, whose receptive field is bigger and activation is locally invariant to the position of the detected pattern. Both these behaviours are simulated by CNNs.

The name "Convolutional Neural Network" comes from the fact that these networks are based on artificial neural networks and use the mathematical operation *convolution*. Due to this, convolutional neural networks can be defined as artificial neural networks that use convolutions instead of matrix multiplications.

Convolution is applied to two functions, e.g. $x(t)$ and $k(t)$ for the 1-d case, consisting of the application of the function $k(t)$, called *kernel* or *filter*, on each moment t of $x(t)$, function known as the input. Convolution is often employed for filtering, for example, denoising by smoothing the input function via a weighted mean of a neighbourhood for each moment t. The symbol used to denote a convolution is an asterisk, like exemplified in Equation (2.4).

$$s(t) = (x * k)(t) \tag{2.4}$$

It is possible to interpret the convolutions of discrete functions as local matrices multiplication. Any artificial neural network algorithm that uses matrices multiplication and does not depend on the structural properties of the matrix will work when using convolutions, without the necessity of any other modifications (GOODFELLOW; BENGIO; COURVILLE, 2016).

In traditional artificial neural networks, the matrices multiplication is used to connect the network's input with each output, making every network input influence all of the network's output. In convolutional neural networks, each interaction between the network's input and output is done locally. This is known as sparse connectivity or sparse weights and is illustrated by Figure 4, obtained by using a kernel function smaller than the input. This makes it possible to store a smaller number of parameters, increase the statistical efficiency and reduce the number of computations. It also allows the network to find important local features.

To enable the use of a kernel function that is smaller than the input, parameter sharing, also called connected weights, is performed. This concept is illustrated by Figure 5. In a traditional artificial neural network, each element of the weight matrix is used only once in the computation of the output, during the multiplication by an element of the input. In a CNN, the same element of the kernel function is used for each element of the input. This way, instead of learning a set of parameters for each localisation, the network searches for a single set of parameters that is applied to every location of the image through convolution. The use of convolution, combined with parameter sharing, results in an equivariance to translation property, which means that if the input is modified by a translation operation, the output will be distorted by the same operation.

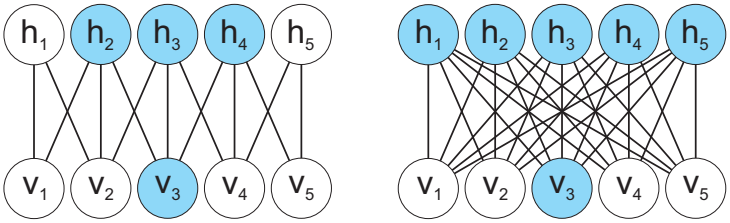


Figure 4 – Sparse connectivity used by convolution neural networks (right) compared to the fully connected layers of a traditional artificial neural network (left). Highlighted in blue are the impact of a visible unit v_i on the the hidden units h_j, when using a kernel function k of size 3. Image based on: (GOODFELLOW; BENGIO; COURVILLE, 2016).

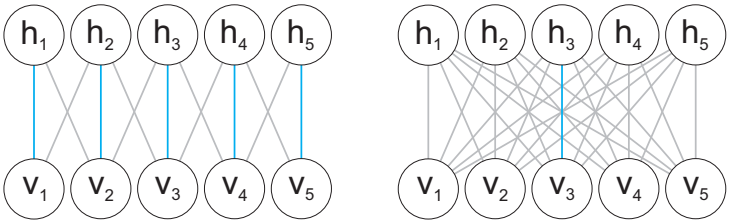


Figure 5 – Parameter sharing used by convolutional neural networks compared to a weight matrix used by traditional artificial neural networks. Highlighted in blue is an example of connections that share the same parameter. In this example, a kernel function of size 3 was used. Image based on: (GOODFELLOW; BENGIO; COURVILLE, 2016).

When used to process a time series, equivariance to translation makes convolution create a timeline that shows in what instant each feature is shown in the input. When applied to images, convolution creates a kind of two-dimensional map (called feature map) that shows where each feature occurs in the input.

A typical CNN used for pattern recognition in static images is composed by a series of convolutional layers, activation functions and pooling operators, ending in a classification module. The three most common layers of data processing are (LECUN; KAVUKCUOGLU; FARABET, 2010):

(1) Convolutional layer or Filter Bank Layer: receives as input a three-dimensional tensor containing a set of feature maps. Each feature map is called x_i and each of its elements

x_{ijk}. The output of this layer is also a three-dimensional tensor, y, consisting of another set of feature maps y_l. A kernel function k_{il} connects the input x_i to the output y_l. This kernel function is used as a filter and is learnt during the training stage. It is applied using convolution on the inputs to obtain specific features in all possible positions of the input. Additionally, to the kernel function, a bias parameter b_j is also learnt during the training stage. Both optimal kernel function and bias parameter are found using Equation (2.5).

$$y_l = b_l + \sum_i (k_{il} * x_i) \tag{2.5}$$

(2) Activation Function or Non-Linearity Layer: applies a non-linear activation function to every position (ijk), for example, a $\tanh()$ function or a $R_{\text{abs}} : \text{abs}(g_i \cdot \tanh()))$, where g_i is a trainable gain parameter. In this layer, normalisation functions are also applied to the data, like subtractive normalisation, divisive normalisation or local contrast normalisation.

(3) Feature Pooling Layer: this layer considers each feature map separately, applying a pooling function that replaces the output of the layer with summaries of neighbourhoods. The most popular pooling functions are *max* P_M and *average* P_A, which are usually applied with a stride (gaps) bigger than one and smaller or equal to the size of the considered neighbourhood. By doing this, the size of the output is reduced, allowing the network to receive inputs of different sizes. Also, the application of a pooling function makes the resulting representations invariant to small translations in the input. When applied to outputs of separately parametrised convolutions, it allows the representations to learn which transformations it should be invariant to (GOODFELLOW; BENGIO; COURVILLE, 2016).

Training CNNs as supervised methods is usually done using a stochastic gradient descent that tries to minimise the difference between the expected output and the one achieved by network. These gradients are computed by using the back-propagation algorithm (WERBOS, 1974).

2.6 Recurrent neural networks

While feedforward networks and convolutional networks have many restrictions regarding input and output, both accepting only fixed-sized vectors as input and producing a fixed-sized vector as output and computing the mapping from inputs to outputs using a fixed amount of steps (KARPATHY, 2015), Recurrent Neural Network (RNN) allow more flexibility by operating over sequences of vectors. That is, RNNs allow inputs, outputs or both to be composed by *sequential data*, i.e. data where observations consist of a sequence and each observation is allowed to have a different sequence length (GOODFELLOW; BENGIO; COURVILLE, 2016).

The use of RNN models on sequences with variable length are only possible due to parameter sharing across different parts of the model. If each value of the time index was designated its own separate parameter, it would not be possible to generalise across different sequence lengths and different positions in time. This is particularly important in applications where the important part of a sequence can occur at any arbitrary point of that sequence. The idea is similar to the parameter sharing used by CNNs (Section 2.5), where the same convolutional kernel is used multiple times. In RNN models, parameter sharing is achieved by defining each member of the output as a function of the previous members of the output, all produced using the same update rule. This approach results in the sharing of parameter through a deep computational graph (GOODFELLOW; BENGIO; COURVILLE, 2016).

Recurrent neural networks can be created using many different design patterns, two of the most important are shown in Figure 6. Some of these design patterns, like the one shown in Figure 6a, are proven to be powerful computational tools since they can process any computable function by simulating a Turing machine using a finite size network (GOODFELLOW; BENGIO; COURVILLE, 2016). Forward and back-propagation equations may vary with the chosen design pattern.

Considering the simplest RNN architecture (Figure 6a) (KARPATHY; JOHNSON; LI, 2015), hidden state vectors h_t^ℓ are arranged in a two dimensional grid, where $t = 1, \ldots, T$ indicates time and $\ell = 1, \ldots, L$ is the layer containing that unit or its depth. $h_t^0 = x_t$ are the units that hold the input vectors x_t and h_t^L are the units used to predict the output vector y_t. All intermediate vectors h_t^ℓ are computed with a recurrence formula based on h_{t-1}^ℓ and $h_t^{\ell-1}$ (Equation (2.6)), where all $h \in \mathbb{R}^n$.

$$h_t^\ell = \tanh W^\ell \begin{pmatrix} h_t^{\ell-1} \\ h_{t-1}^\ell \end{pmatrix} + b_t^\ell \tag{2.6}$$

W^ℓ is the ℓ-th layer's parameter matrix with $n \times 2n$ dimensions and $b_t^\ell \in \mathbb{R}$ is the bias variable of unit h_t^ℓ. Parameters are shared through time but may vary between layers.

RNN models use a weak form of coupling (SUTSKEVER; MARTENS; HINTON, 2011) that links the inputs from the layer below in depth ($h_t^{\ell-1}$) and the layer before in time (h_{t-1}^ℓ) to the current unit (h_t^ℓ). These inputs are transformed and interact additively before being squashed by a activation function, tanh in this example, applied element-wise. Other models, like the Long Short-Term Memory (LSTM) (HOCHREITER; SCHMIDHUBER, 1997), include other multiplicative interactions (KARPATHY; JOHNSON; LI, 2015).

2.6.1 Long Short-Term Memory (LSTM)

Proposed by Hochreiter and Schmidhuber (1997), Long Short-Term Memory (LSTM) is a variant of the RNN model that can store and retrieve information over long periods of

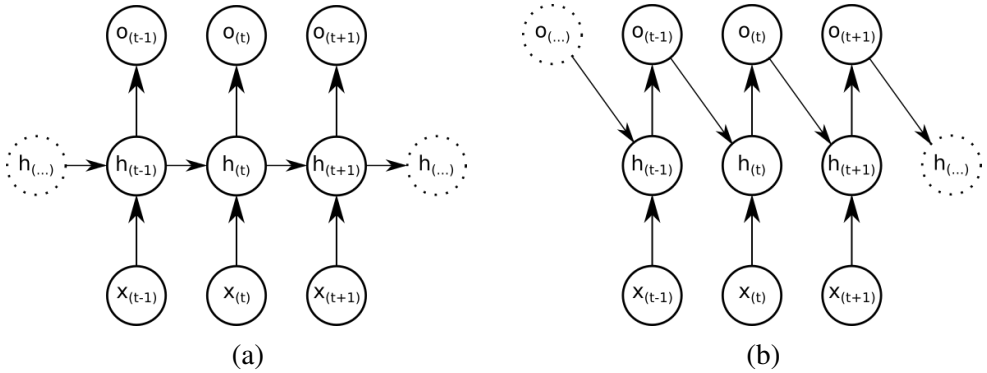


Figure 6 – Two of the most important recurrent neural network design patterns. 6a shows a pattern were the recurrence occurs in hidden-to-hidden setting while in 6b recurrences occurs from the output to the hidden layer. Images based on: (GOODFELLOW; BENGIO; COURVILLE, 2016).

time using an explicit gating mechanism and a built-in constant error carousel (KARPATHY; JOHNSON; LI, 2015). This model also addresses some of the difficulties of training the RNN model (BENGIO; SIMARD; FRASCONI, 1994). By using the back-propagation algorithm, the gradients in a RNN either vanish or explode. To alleviate the *exploding gradient problem*, clippings at some maximum value of the gradients were used (PASCANU; MIKOLOV; BENGIO, 2012), while LSTMs were designed to address the *vanishing gradient problem*. This is done maintaining, not only a hidden state vector h_t^ℓ, but also a memory vector c_t^ℓ.

A LSTM decides, at each time step, if it should read from, write to or reset the cell using explicit gating mechanisms. These are defined by Equations (2.7), (2.8) and (2.9), where the sigm and tanh functions are applied element-wise. W^ℓ is a $4n \times 2n$ weight matrix (n being the number of cells in the ℓ-th layer) that varies between layers but is shared though time.

Variables $\mathbf{i}$, $\mathbf{f}$ and $\mathbf{o} \in \mathbb{R}^n$ are binary vectors that control whether the memory cells will be updated, reset to zero or if its local state is revealed in the hidden vector, respectively. These vectors are smoothed by a sigmoid function ranging between zero and one, keeping the model differentiable. $\mathbf{g} \in \mathbb{R}^n$ ranges from -1 to 1 and is used to modify the contents of memory cells $\mathbf{c}$. This modification is done additively and allows gradients on the memory cells $\mathbf{c}$ to flow backwards through time until the flow is interrupted by a multiplicative interaction of an active forget gate.

$$\begin{pmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^\ell \begin{pmatrix} h_t^{\ell-1} \\ f_{t-1}^\ell \end{pmatrix} \tag{2.7}$$

$$c_t^\ell = f \odot c_{t-1}^\ell + i \odot g \tag{2.8}$$

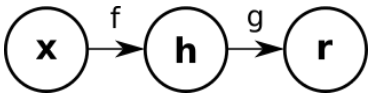


Figure 7 – General structure of an autoencoder, where **x** is the input of the network which is mapped to the output (reconstruction) **r** going through the internal representation **h**. The connection between the input and the representation is done through function f, called encoder. Another function is used to map the representation to the reconstruction, this function g is called decoder. Image based on: (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$h_t^{\ell} = o \odot \tanh(c_t^{\ell}) \tag{2.9}$$

2.7 Autoencoders

An autoencoder is an unsupervised artificial neural network architecture whose main goal is to copy its input to its output (GOODFELLOW; BENGIO; COURVILLE, 2016). It can be seen as an identity function estimator, that is, given a dataset, the autoencoder tries to approximate a non-trivial identity function by comparing its input with its output and minimising the differences. Figure 7 shows an example of the general structure used on autoencoders, the input **x** is mapped to the output **r**, called reconstruction, by passing through an internal representation **h**.

These networks can be divided into two parts: an encoder function $\mathbf{h} = f(\mathbf{x})$ and a decoder function $\mathbf{r} = g(\mathbf{h})$. The encoder function f is the result of training hidden layers that describe a code used to represent the input. To avoid overfitting the training samples and finding the trivial identity function, autoencoders are designed to be unable to learn to copy perfectly the input to the output. This is done by forcing the model to prioritise which aspects of the input should be copied. Traditionally, autoencoders were used for dimensionality reduction and feature learning and, more recently, because they were linked to latent variable models, generative modelling (GOODFELLOW; BENGIO; COURVILLE, 2016).

There are many different ways used to force autoencoders away from the trivial solution to the task of copying the input on to the output of the network. One of these ways is to constrain **h** to have dimension smaller than **x**. This makes the generated code capture the most salient features from the training data. Autoencoders that use this rule are called *undercomplete*. During the learning process, a loss function $L(\mathbf{x}, g(f(\mathbf{x})))$ that penalises $g(f(\mathbf{x}))$ for being different from **x** is minimised.

If the decoder function is linear and L is the mean squared error, a subspace very similar to the one obtained by the Principal Component Analysis (PCA) method is learned by an undercomplete autoencoder. When using nonlinear encoder and decoder functions, the autoencoder can learn to perform the copying task without learning anything useful about the data. This occurs when the capacity of the autoencoder is allowed to become too great (GOODFELLOW;

BENGIO; COURVILLE, 2016).

Regularised autoencoder models were proposed based on the idea that, by choosing the code dimension and the capacity of the encoder and decoder functions according to the distribution to be modelled, any architecture of autoencoder would be successfully trained. These models use loss functions that encourage other properties, like sparsity of the representation, smallness of the derivative of the representation and robustness to noise or missing inputs, to be learned by the model, in addition to copying the input to the output. Regularised autoencoders will learn useful information about the data distribution even if the model has the ability to learn the trivial identity function. Whenever an autoencoder's code dimension is higher then the dimension of the original data, it is called *overcomplete* autoencoder. Overcomplete autoencoders require regularisation to avoid learning the trivial identity function.

2.8 Generative adversarial networks

Generative Adversarial Network (GAN) (GOODFELLOW *et al.*, 2014) explore the idea of training two deep neural networks in an adversarial manner, that is, by making them compete against each other. In general, GANs try to learn how to mimic a certain dataset by learning about its distribution.

The first important characteristic of GANs is that it is a generative approach. This differs from most of the common machine learning algorithms, which use a discriminative approach. The main objective of discriminative algorithms is to, given features that describe the input data, try to predict to which label or category that example belongs to. This means that discriminative algorithms try to model the boundary between classes, which is done by finding a mapping f from $\mathscr{X}$, the feature space for the input data, and $\mathscr{Y}$, the label space, $f : \mathscr{X} \longrightarrow \mathscr{Y}$.

On the other hand, generative algorithms try to generate new examples from the input data by modelling its probability distribution. For GANs, this is done by mapping a known distribution to the input data distribution, while also modelling this distribution's probability density function.

The second characteristic of GAN models is the adversarial approach. When training a GAN two neural network models have to be defined: a generator and a discriminator. The generator is responsible for learning the mapping g from $\mathscr{N}$, a known probability distribution (usually a standard Gaussian distribution or a uniform distribution), to $\mathscr{X}$, the input data probability distribution. The discriminator will function as an estimator of the input's probability density function. This way, the adversarial training occurs by challenging the generator to confuse the discriminator, while the discriminator tries to differentiate between real input samples and generated samples. This is done by using loss functions for the generator and the discriminator that oppose themselves, as seen in Equations 2.10 and 2.11, where $\mathscr{L}(G, \mathbf{z})$ is the generator loss, $\mathscr{L}(D, \mathbf{x}, G, \mathbf{z})$ is the discriminator loss, D and G are respectively the discriminator and generator

functions, $\mathbf{z}$ are samples from a known probability distribution $p_{\mathbf{z}}$) and $\mathbf{x}$ are samples of real data.

$$\mathscr{L}(G,\mathbf{z}) = \max \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log D(G(\mathbf{z}))] \tag{2.10}$$

$$\mathscr{L}(D,\mathbf{x},G,\mathbf{z}) = \max \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1-D(G(\mathbf{z})))] \tag{2.11}$$

The main challenge faced when training GANs is called mode collapse. Most real-world data distributions are complex and multimodal, which means that the probability distribution that describes the data has multiple "peaks" (modes) where different sub-groups of samples are concentrated. A mode collapse happens when the GAN converges to a model that is able to reproduce only a small number of modes. For example, if the generator learns that generating samples from a single mode is effectively confusing the discriminator to think they are real samples, the discriminator will then try to counter this effect by considering a different mode as containing only real samples. This will lead to generator and discriminator alternating between modes indefinitely, never rewarding the generator to cover more than a few modes. Mode collapse is usually diagnosed by analysing the variety of the output samples of the generator since it causes the generator to output a very low variety of samples.

2.8.1 Deep Convolutional GAN (DCGAN)

Deep Convolutional GAN (DCGAN) (RADFORD; METZ; CHINTALA, 2016) established a set of guidelines for training GANs using convolutional network architectures. These guidelines aim to stabilise the adversarial training and improve the diversity and realism of the generated samples.

The first suggestion is to replace deterministic spatial pooling functions (e.g. max-pooling) with strided convolutions, similar to what is done by all convolutional net (SPRINGENBERG *et al.*, 2015). This allows the network to learn its own spatial down-sampling. A more stable training was achieved by replacing pooling layers with strided convolutions for the discriminator and fractional-strided convolutions for the generator.

Another way to improve the DCGAN training is to eliminate fully connected layers used on top of convolutional features. Also, the authors state that using batch normalisation (IOFFE; SZEGEDY, 2015) stabilises the training procedure by normalising the input of each unit to have zero mean and unit variance. This helps solve some problems due to poor initialisation and allow the gradients to flow deeper into the model, which proved necessary when using deep generators by preventing it from collapsing all generated samples to a single point. It is important to notice that batch normalisation should not be applied to the generator output layer or the discriminator input layer since it causes model instability when applied to all layers.

Lastly, the selection of appropriate activation functions has a big impact on GAN's training stability and the achieved results. Layers belonging to the generator should usually use ReLU activation (NAIR; HINTON, 2010), except for the output layer that has achieved better results using the *tanh* function. On the other hand, leaky rectified activation (MAAS; HANNUN; NG, 2013; XU *et al.*, 2015) are better for discriminator layers and help improve the overall results.

2.8.2 Wasserstein GAN (WGAN)

Wasserstein GAN (WGAN) (ARJOVSKY; CHINTALA; BOTTOU, 2017) propose a change in the loss function of traditional GANs by using an approximation of the Wasserstein distance, also known as Earth Mover's distance, which measures the distance between two probability distributions. This distance is proven to be differentiable almost everywhere, allowing the discriminator, in this case, called critic (because it does not explicitly classify the samples into real or fake), to reach convergence before updating the generator. This guarantees more accurate gradients used for back-propagation. It is important to notice that training the discriminator to convergence using traditional GANs is not possible since it would lead to vanishing gradients.

Such change to the loss function improves the stability of the GAN training and has shown to achieve a certain level of correlation between the discriminator loss and the perceptual quality of the generated samples. This is important since, for traditional GANs, it is very difficult to judge whether training is progressing as expected without a thorough analysis of the generated samples. However, to ensure certain required conditions for the WGANs to work it is necessary to perform weight clipping to a small fixed range. To avoid weight clipping, improved WGANs training (GULRAJANI *et al.*, 2017) has proposed the use of gradient penalty.

2.8.3 Boundary equilibrium GAN (BEGAN)

Boundary Equilibrium GAN (BEGAN) (BERTHELOT; SCHUMM; METZ, 2017) is a variation of traditional GANs inspired by Energy Based GAN (EBGAN) (ZHAO; MATHIEU; LECUN, 2016) that aims at improving the stability of the adversarial training by using an autoencoder as the discriminator. This modification is accompanied by a change to the loss function, which now measures the quality of reconstruction achieved by the discriminator on real and generated images. The reconstruction loss is the error associated with reconstructing examples using an autoencoder, in this case, the discriminator.

BEGANs assume that matching the distributions of the reconstruction losses can be a suitable alternative to matching the data distributions. The loss function used from training is based on this idea and derived from the Wasserstein distance between the reconstruction losses of real and generated data. An equilibrium term to balances the training of the discriminator and the generator is also included in the loss function to avoid needing to find this equilibrium

manually.

The adversarial part of training consists of the discriminator learning how to reconstruct real images continuously better while also increasing the reconstruction loss for generated images. The generator is trained to minimise the reconstruction loss outputted by the discriminator for generated samples.

2.9 Concluding remarks

This chapter explained several fundamental concepts and methods required to understand methods typically used in machine learning, feature extraction and representation learning techniques. These concepts are important in the context of the methods presented in Chapter 3 and to better understand the experiments shown in Chapters 4, 5 and 6.

CHAPTER

3

SPATIO-TEMPORAL REPRESENTATION LEARNING

3.1 Opening remarks

Video processing has been a topic of interest in computer science for years and has addressed many different problems, such as action recognition (HERATH; HARANDI; PORIKLI, 2017; WANG, 2018), action localisation (WEINZAEPFEL; HARCHAOUI; SCHMID, 2015; SHOU; WANG; CHANG, 2016), anomaly detection (POPOOLA; WANG, 2012; SABOKROU *et al.*, 2018), scene recognition (SANDE; GEVERS; SNOEK, 2010; PROTASOV *et al.*, 2018), among others. The proposed solutions for many of these problems have one stage in common: feature extraction. Feature extraction is considered one of the most important steps in a machine learning pipeline (BENGIO; COURVILLE; VINCENT, 2013).

Many classical image feature extraction methods, which encode spatial information, were extended for videos, that is, to also include temporal information. These methods were initially data-independent and designed for a small set of problems or datasets. The features extracted by these methods are known as *hand-crafted features* (Section 3.2). Data-driven methods, whose objective is to learn which are the relevant features based on a training dataset, are discussed in Section 3.3.

3.2 Hand-crafted spatio-temporal features

Hand-crafted features are manually designed, that is, they are not directly influenced by the data. Designing hand-crafted features often require expensive human labour and rely on expert knowledge about the application or dataset for which it will be used. This incorporated knowledge usually causes hand-crafted features not to generalise well to different applications. When dealing with video processing, incorporating temporal information into the feature vector

is one of the main challenges of feature extraction.

Many of the spatio-temporal feature extraction methods derive from image descriptors which where extended to include a temporal dimension. For example, Laptev and Lindeberg (2003) proposed Spatio-Temporal Interest Points (STIPs) as an extension of the Harris corner detectors (HARRIS; STEPHENS, 1988); SIFT (LOWE, 2004) and HOG (DALAL; TRIGGS; SCHMID, 2006) inspired SIFT-3D (KLASER; MARSZAŁEK; SCHMID, 2008) and HOG3D (SCOVANNER; ALI; SHAH, 2007), both proposed for action recognition. Other approaches try to combine local appearance and motion information extracted by image descriptors, such as HOG, HOF and MBH based on spatio-temporal interest points organised in a dense grid or around dense point trajectories. These features are then encoded to produce a fixed-size feature vector that describes the entire video. The current state-of-the-art method for spatio-temporal hand-crafted feature extraction from video is the Improved Dense Trajectories (IDT) (WANG; SCHMID, 2013) method encoded by fisher vectors (ONEATA; VERBEEK; SCHMID, 2013). This method is detailed in Section 3.2.1.

3.2.1 Dense trajectories

One of the most popular ways of representing videos is by associating local features with space-time interest point detectors (KARPATHY *et al.*, 2014). Combined in a bag-of-features representation, local features have achieved high-quality results in action recognition. However, space and time domains in videos have different characteristics, which makes it more intuitive to treat each of these dimensions in different ways instead of using interest point detection in a joint three dimensional space (WANG *et al.*, 2011).

Motivated by the results achieved by the use of motion information in video for action recognition (MESSING; PAL; KAUTZ, 2009; SUN *et al.*, 2009), the method proposed by Wang *et al.* (2011) uses an efficient approach to extract trajectories from videos. This method also takes advantage of the fact that the results are further improved by using dense over sparse sampling (WANG *et al.*, 2009). The extracted trajectories are computed by tracking densely sampled points using optical flow fields while imposing global smoothness constrains to improve their robustness to large displacements.

To compute dense trajectories, feature points are sampled on a grid spaced by W pixels and tracked in multiple spatial scales, separately. The value of $W = 5$ was empirically defined (WANG *et al.*, 2011) and is used with eight different spatial scales, each one defined by using a factor of $\frac{1}{\sqrt{2}}$. The tracking of each point $P_t = (x_t, y_t)$ at frame t is done to the next frame $t+1$ by using median filtering in a dense optical flow field $\omega = (u_t, v_t)$. Each point P_{t+1} is defined by using the following equation:

$$P_{t+1} = (x_{t+1}, y_{t+1}) = (x_t, y_t) + (M * \omega)|_{(\bar{x}_t, \bar{y}_t)}, \tag{3.1}$$

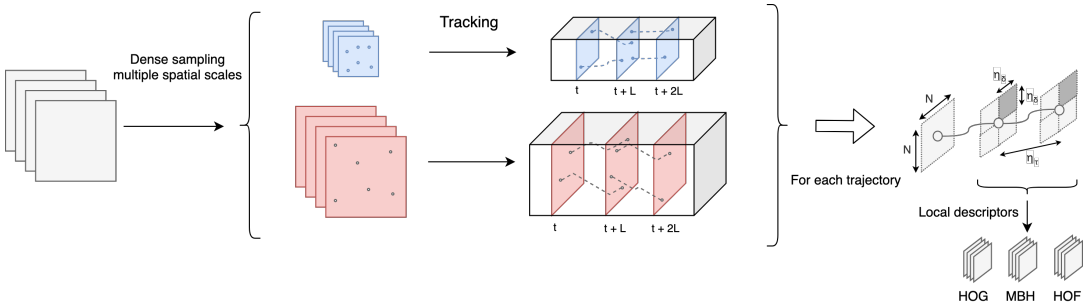


Figure 8 – Pipeline used to extract feature vectors using the dense trajectory method. Adapted from Wang *et al.* (2011).

where M is the median filtering kernel and $(\bar{x}_t, \bar{y}_t)$ is the rounded position of (x_t, y_t). The rounded position is used to avoid problems with the optical flow tracking a point into a subpixel region. The trajectories are created by concatenating points of subsequent frames. After the dense optical flow field is calculated, points can be tracked densely without any additional cost. The algorithm proposed by Farnebäck (2003) is used to extract dense optical flow.

To prevent trajectories from drifting from their initial location during tracking, the length of each trajectory is limited to L frames. That is, after L frames, the trajectory is removed from the tracking process. The existence of a trajectory in each cell of the grid is verified for every frame, assuring dense coverage of the video. If there are no tracked points in a $W \times W$ neighbourhood, a feature point is sampled and included in the tracking process. L was empirically defined by Wang *et al.* (2011) as 15 frames. Figure 8 illustrates the pipeline used to compute dense trajectories representations.

Image areas without any structure make it hard to track points. To avoid such problems, when a feature point is sampled, the smallest eigenvalue of its autocorrelation matrix is compared to a threshold. Whenever this eigenvalue is smaller than the threshold, this point is not included in the tracking process. Trajectories with sudden large displacements are assumed to be wrong and are also removed.

Dense trajectories also contain information about local motion patterns. This information can be described by a sequence $S = (\Delta P_t, \ldots, \Delta P_{t+L-1})$ of displacement vectors $\Delta P_t = (P_{t+1} - P_t) = (x_{t+1} - x_t, y_{t+1} - y_t)$. A normalisation by the sum of the magnitudes of the displacement vectors is then applied (Equation (3.2)). The resulting vector is called trajectory descriptor.

$$S' = \frac{(\Delta P_t, \ldots, \Delta P_{t+L-1})}{\sum_{j=t}^{t+L-1} \left\| \Delta P_j \right\|} \tag{3.2}$$

To take advantage of the motion information in dense trajectories, descriptors are computed within a space-time volume around the trajectory of size $N \times N$ pixels and L frames. This volume is divided into a spatio-temporal grid of size $n_\sigma \times n_\sigma \times n_\tau$, embedding structure information into the representation. By default, these parameters are set to $N = 32, n_\sigma = 2, n_\tau = 3$ (WANG *et al.*, 2011). These values were found based on cross validation performed on the training set of

the Hollywood2 Actions dataset (MARSZAŁEK; LAPTEV; SCHMID, 2009).

The local descriptors HOG (Section 3.2.1.1), HOF (Section 3.2.1.2) and MBH (Section 3.2.1.3) were used with the dense trajectory method in its original paper (WANG *et al.*, 2011). Both HOF and MBH benefit from the optical flow computed to extract dense trajectories, making the computation process more efficient. Due to the its robustness to camera motion, Motion Boundary Histogram (MBH) (DALAL; TRIGGS; SCHMID, 2006) achieved the best results when used to extract features with dense trajectories.

To improve these results, Wang and Schmid (2013) estimates the global background motion by assuming that two consecutive frames are related by a homography (SZELISKI, 2006). This is true since, in most cases, global motion between two consecutive frames is small. The homography is estimated by finding correspondences between two frames and then using RANSAC (FISCHLER; BOLLES, 1981), which allows rectification of the image to remove camera motion. When compared to the original optical flow, the rectified version is capable of suppressing background camera motion, enhancing foreground moving objects.

To find the correspondence between two frames Speeded-Up Robust Features (SURF) features (BAY; TUYTELAARS; GOOL, 2006) are extracted and then matched based on the nearest neighbour rule. This is combined with an optical flow estimation for salient feature points by thresholding the smallest eigenvalue of the autocorrelation matrix (SHI; TOMASI, 1994). These two approaches are complementary, given that one focuses on blob-type structures and the other on corners and edges.

The results obtained by using dense trajectories improves consistently when cancelling out camera motion due to the removal of trajectories created by camera motion (WANG; SCHMID, 2013). These trajectories are identified by thresholding the displacement vectors of each trajectory in the warped flow field. Also, motion descriptors benefit from this, as shown by the degradation of the HOF descriptors in the presence of camera motion (WANG; SCHMID, 2013).

Furthermore, the use of a human detection method to remove matches from human regions is proposed as a way to improve camera motion estimation (WANG; SCHMID, 2013). The human detector is used as a mask to remove feature matches inside the bounding boxes when estimating homography. However, homography fails to fit the background, despite human detection, when the background is represented by two planes, one of which is closer to the camera.

Since human detection methods do not work perfectly, missing humans due to pose or viewpoint changes, all bounding boxes obtained by the human detector are tracked. This is done by propagating the detection made to the next frame using the average flow vector. Bounding boxes are tracked for at most 15 frames and tracking is stopped if a 50% overlap with another bounding box occurs.

Due to the number of features extracted by the dense trajectories method and that videos in a single dataset may have different lengths, a feature encoding technique must be used to reduce the dimensionality of the feature vectors. The most used methods for feature encoding are Bag-of-Features (BOF) (NOWAK; JURIE; TRIGGS, 2006) and Fisher Vector (FV) (SÁNCHEZ *et al.*, 2013). In order to capture the spatio-temporal layout of the features, a concatenation of several histograms can be used, each one computed over several space-time cells overlaid on the video.

BOF constructs a codebook based on feature vectors with K quantisation cells. This is done by clustering a subset of the feature vectors of the training set using the k-means algorithm (MACQUEEN, 1967). Then, each descriptor is assigned to their closest vocabulary word based on Euclidean distance, resulting in histograms of visual word occurrences which are used as representations for each video. Multiple initialisations of the k-means algorithm, keeping the result with the lowest error, can be used to increase precision.

Fisher Vectors extends the Bag-of-Visual-Words (BOV) representation (SIVIC; ZISSERMAN, 2003), which is based on the quantisation of the local descriptor space using off-line k-means clustering on a large collection of local descriptors. FVs encode for each quantisation the number of assigned descriptors and their mean and variance along each dimension, resulting in $K(2D+1)$ dimensions, where K is the number of quantisation cells and D the number of dimensions in the descriptor.

In FV representations, each local descriptor is assigned in a weighted manner to multiple clusters by using the posterior component probability given the descriptor. Principal Component Analysis (PCA) (JOLLIFFE, 2002) can be used to reduce the dimensionality before FV encoding. FV has been proven to be among the most effective encoding methods for object recognition (CHATFIELD *et al.*, 2011).

Oneata, Verbeek and Schmid (2013) proposes the use of FV with Spatial Pyramid Matching (SPM) (LAZEBNIK; SCHMID; PONCE, 2006) as an alternative to BOF histograms to encode the features extracted by the dense trajectories method, obtaining state-of-the-art results in action recognition and complex event recognition, while using representations with fewer dimensions.

3.2.1.1 Histogram of Oriented Gradients (HOG)

Based on evaluating well-normalised local histograms of image gradient orientations in a dense grid, Histogram of Oriented Gradients (HOG) (DALAL; TRIGGS, 2005) takes advantage of the distribution of local intensity gradients or edges directions to characterise the local object appearance and shape. This is done by dividing the image into small connected regions, called *cells*, in which a local histogram of gradient directions or edge directions is computed over all pixels. The final representation is obtained by combining the histograms computed in all cells. HOG descriptors are particularly suited for human detection (DALAL; TRIGGS, 2005).

Contrast-normalisation can be applied to achieve better results due to invariance to illumination. This can be done by dividing the image into larger spatial regions, called *blocks* and accumulating a measure of local histogram "energy" that is then used to normalise all cells included in the block. These normalized descriptors are the so-called HOG descriptors. These descriptors are capable of capturing gradient structures characteristic of local shapes by using local representations with an easily controllable degree of invariance to local geometric and photometric transformations. This means that the resulting descriptors are robust to local translations or rotations that are smaller than the local spatial or orientation bin size.

To extract HOG descriptors from an image, firstly, gradient values must be computed. This is most commonly done by filtering the colour or intensity data of the image using the one-dimensional centred point discrete derivative mask in the horizontal ($[-1,0,1]$) and vertical directions ($[-1,0,1]^T$). Then, the image is divided into small cells of rectangular (R-HOG) or circular shape (C-HOG). Each pixel contained by a cell is used in a weighted manner to create an orientation-based histogram. This histogram is created for each cell and its bins are evenly spread over the orientation of the gradients. The range of the orientation can be defined over 0 to 180 degrees or over 0 to 360 degrees, depending on if the gradient is "signed" or "unsigned". The contribution of a pixel to each bin of the histogram is weighted based on the magnitude of the gradient or some function of this magnitude.

To increase the robustness of the descriptors to illumination and contrast changes, gradient strengths are locally normalized by grouping cells together into blocks. Some of the most common methods used for normalisation are: ℓ_2-norm (Equation 3.3), hysteresis-based ℓ_2 normalisation (LOWE, 2004) or ℓ_1-sqrt (Equation 3.4), where v is the non-normalized vector containing all histograms of a given block, $\|\Delta v\|_k$ is its k norm for $k = 1,2$ and e is a small constant.

$$f = \frac{v}{\sqrt{\|\Delta v\|_2^2 + e^2}} \tag{3.3}$$

$$f = \sqrt{\frac{v}{(\|\Delta v\|_1 + e)}} \tag{3.4}$$

Blocks are typically allowed to overlap, which means that a cell can contribute to more than a block, and, therefore, to the final descriptor. The size and shape of the cells and blocks and the number of bins in each histogram are hyperparameters set by the user.

3.2.1.2 Histogram of Optical Flow (HOF)

While the Histogram of Oriented Gradients (HOG) descriptor focuses on describing the local appearance, Histogram of Optical Flow (HOF) (DALAL; TRIGGS; SCHMID, 2006) main goal is to extract information based on the local motion at each frame. The steps used to compute HOF descriptors are very similar to the ones used to extract HOG descriptors.

Firstly, optical flow displacements for both, horizontal and vertical directions are computed. Then, for each small spatial divisions (cells), displacements are combined into an orientation histogram using their magnitudes as weights. Afterwards, cells are aggregated into larger overlapping spatio-temporal regions (blocks), for which a normalisation technique is applied. Normalised histograms are concatenated resulting in HOF descriptors.

3.2.1.3 Motion Boundary Histogram (MBH)

Motion Boundary Histogram (MBH) descriptors were proposed by Dalal, Triggs and Schmid (2006) for human detection in video. Its main goal is to encode the relative motion between pixels. These descriptors were later used for other purposes like action recognition (WANG *et al.*, 2013). MBH descriptors are obtained by computing the derivatives ($\mathbf{I}_{xx}, \mathbf{I}_{xy}, \mathbf{I}_{yx}, \mathbf{I}_{yy}$) for the horizontal ($\mathbf{I}_x$) and vertical ($\mathbf{I}_y$) components of the optical flow separately, where $\mathbf{I}_{xy} = \frac{d}{dy}\mathbf{I}_x$ is the y-derivative of the horizontal (x) component of optical flow. $\mathbf{I}_\mathbf{w} = (\mathbf{I}_x, \mathbf{I}_y)$ denoting the two-dimensional flow image ($\mathbf{w} = (x, y)$). Spatial derivatives are computed by using one-dimensional centred point discrete derivative mask in the horizontal ($[-1, 0, 1]$) and vertical directions ($[-1, 0, 1]^T$).

To extract MBH descriptors, both flow components $\mathbf{I}_x$ and $\mathbf{I}_y$ are treated as independent "images", whose gradients are computed separately. Then, after diving the images into small regions (cells), their corresponding gradient magnitudes and orientations are used as weighted votes into a local orientation histogram, similar to what is done to compute the HOG descriptor. Normalisation is done by using larger overlapping regions of the image (blocks). The most common method of normalisation for MBH descriptors is the hysteresis-based ℓ_2 normalisation (LOWE, 2004). MBH descriptors are the result of the concatenation of the histograms obtained from each cell of each optical flow component.

3.3 Spatio-temporal representation learning

Representation learning methods aim to discover automatically, i.e. directly from raw data, which are the relevant representations (features) for given task (BENGIO; COURVILLE; VINCENT, 2013; LECUN; BENGIO; HINTON, 2015). The recent increase in processing capabilities of CPUs and GPUs, combined with the growth in data availability, allowed representation learning algorithms, such as deep learning, to excel in different areas of machine learning, especially computer vision. In general, features learnt from data were shown to achieve better results than the ones that are manually designed (GOODFELLOW; BENGIO; COURVILLE, 2016). Representation learning also allows machine learning systems to adapt rapidly to new tasks, reducing the need for human intervention, since features extracted by representation learning methods tend to generalise well to different applications.

Deep learning methods have achieved state-of-the-art performance in many different

areas such as natural language processing (YOUNG *et al.*, 2018), speech recognition (AMODEI *et al.*, 2016) and image processing (DRUZHKOV; KUSTIKOVA, 2016; LITJENS *et al.*, 2017). Representation learning from videos is considered one of the natural continuations to the development of deep learning algorithms (LÄNGKVIST; KARLSSON; LOUTFI, 2014). However, when applied to video processing the benefits of using deep learning are not as clear (TRAN *et al.*, 2018). The main challenges faced when using deep learning methods for video tasks are: (1) the computational costs involved in processing videos since each video contains hundreds or even thousands of images; (2) video duration tend to vary a lot even when dealing with specific tasks; (3) the event of interest can occur in a very short time window, that is, the number of frames that portray the event of interest can be significantly smaller then the total number of frames in a video; and (4) camera motion.

Many different approaches were used to incorporate temporal information into the representations learnt by deep learning methods while dealing with the aforementioned challenges. We discuss the most commonly used approaches on Sections 3.3.1 and 3.3.2.

3.3.1 Temporal information fusion

Temporal information fusion consists of combining the spatial information learnt by the network over the temporal dimension. The most used methods of temporal information fusion are: concatenation of the output of multiple single-frame streams (KARPATHY *et al.*, 2014); temporal pooling (KARPATHY *et al.*, 2014; NG *et al.*, 2015); temporal convolutions (KARPATHY *et al.*, 2014); and recurrent neural networks (NG *et al.*, 2015; MONTES *et al.*, 2016).

The first approach (Section 3.3.1.1) feeds a second network with the concatenation of the output of multiple single-frame networks (usually with shared parameters). This second network needs, then, to assess temporal information by comparing the concatenated features. The second approach (Section 3.3.1.2) uses pooling operations on the temporal dimension to combine the spatial features learnt by the previous layers of the network. When applied to neural networks, pooling can be used as a layer, which allows for temporal information to be processed at different depths in the network. Depending on where it is applied, the resulting feature vector will have different characteristics.

The third approach (Section 3.3.1.3) extends convolution to include the temporal dimension by using 3-dimensional kernels that capture short temporal information which is then slowly fused as the information is passed to the following layers. This way, higher layers in the network have access to more global information. The forth approach (Section 3.3.1.4) leverages recurrent networks to combine the temporal information learnt by CNNs from single or multiple frames. This is similar to the first approach, however, instead of concatenating the outputs of multiple single-frame networks, these feature vectors are passed to a recurrent network as a sequence.

3.3.1.1 Fusion by concatenation

Fusion by concatenation consists of using a network to process each frame (or time-step, for non-video problems) and then passing the concatenation of the resulting feature vectors to another network, which tries to infer temporal information by comparing the representations extracted from each time-step. This approach was used in Karpathy *et al.* (2014) where a single fully convolutional network was used to extract spatial information from each frame. The resulting feature vectors were concatenated and passed through a set of fully connected layers, whose output indicates the predicted class for the video. The main drawbacks from using this method are that it requires the fusion to occur over a fixed-size temporal window and it can be computationally expensive depending on the size of the feature vector and the length of the temporal window used.

3.3.1.2 Temporal pooling

Temporal pooling was commonly applied to bag-of-words representations. In that context, image-based and/or motion features are computed for every frame of a video, the produced features are then quantised and pooled across time to produce a single feature vector to represent a clip (a part of a longer video) or an entire video (NG *et al.*, 2015). The same can be done when working with neural networks by using a pooling layer. This allows for pooling to be applied at different depths in the network. When applied after convolutional layers, pooling is able to maintain spatial information. If applied at higher (fully connected) layers, pooling combines high-level information learnt for each time-step (NG *et al.*, 2015).

Another option is to apply pooling to smaller temporal slices at lower layers in the network and then later use another pooling operation that combines the information learnt from these temporal slices to generate a single feature vector. This approach makes the network group temporally local features before combining high-level information from a high number of frames (NG *et al.*, 2015). However, pooling does not consider the order in which the feature vectors are organised, which means that shuffling the frames before passing them to the network would result in the same final descriptor.

Ng *et al.* (2015) investigated the previously mentioned architectures and different pooling operations. They state that max-pooling achieves better results when compared to average-pooling and pooling performed by a fully connected layer. This occurs due to a large number of gradients generated by both average-pooling and a fully connected layer, while max-pooling creates sparser updates which help the network learn faster.

3.3.1.3 Temporal convolution

Two-dimensional convolutional layers receive two-dimensional feature maps as inputs and compute features by convolving two-dimensional kernels on both dimensions. These layers are commonly used on images where both dimensions are spatial. When dealing with videos, a

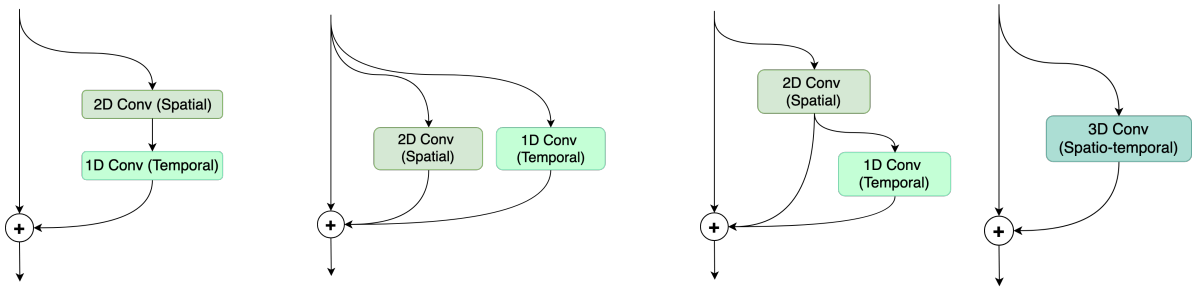
third (temporal) dimension exists and it is desirable to include such dimension in the computations in order to capture temporal information encoded in consecutive frames (BACCOUCHE *et al.*, 2011; JI *et al.*, 2013). In this case, 3-dimensional kernels are convolved over a cube formed by stacking multiple contiguous frames. That way, the feature maps produced by 3D convolutional layers are connected to multiple frames in the previous layer, allowing the network to capture temporal information. By using 3D convolutional layers, the network is capable of slowly fusing temporal information throughout the network, such that the higher the layer is positioned in the network, the more global is the information it receives (both spatially and temporally) (KARPATHY *et al.*, 2014).

Karpathy *et al.* (2014) used 3D convolution on the first three layers of the network to slowly fuse temporal information over 10 frames. Tran *et al.* (2015) experimented with different kernel sizes for 3D convolution and found empirically that $3 \times 3 \times 3$ kernels achieve the best results among the tested configurations. They also show that 3D CNNs (also known as C3D) are good feature learning methods that capture both appearance (spatial) and motion (temporal) information.

These networks are generally composed by 3D convolutional and 3D pooling layers followed by few fully connected layers. By using deconvolutions, Tran *et al.* (2015) show that C3Ds start by focusing on spatial information for the first few frames and then tracks salient motion in subsequent frames. Their experiments use as input blocks of 16 frames per video clip. Varol, Laptev and Schmid (2018) extends these architectures by increasing the number of input frames up to 100 and show that increasing the temporal coverage of the convolutions helps the network achieve better results.

Tran *et al.* (2018) introduces Mixed Convolutional Network (MC) which use both 3D and 2D convolutions as part of residual blocks. MCs start by using 3D convolutional blocks which are followed by 2D blocks. The reason why 3D convolutions are used at the start of the network is that motion modelling is considered to be a low or mid-level operation, while spatial reasoning is performed over the motion features. The opposite was also analysed and was called "Reversed" Mixed Convolutional Network (rMC).

The biggest drawback of using C3Ds is their computational cost since the model size suffers a quadratic growth when compared to traditional CNNs (QIU; YAO; MEI, 2017), and the fact that the input needs to have a fixed size, including for the temporal dimension. Fixed length videos are not common in real-world datasets. This fact makes it necessary to classify smaller slices of the video and combine the predictions made for each slice to achieve a full video prediction. It also hampers the ability of the network to learn long-term temporal information, since only a limited number of frames can be passed to the network. Thus, an approximation of 3D convolutional layers was proposed by using two separate layers: a spatial 2D convolution and a temporal 1D convolution (QIU; YAO; MEI, 2017; TRAN *et al.*, 2018). This approximation reduces the model size significantly making it possible to create much deeper architectures,



(a) P3D-A / R21D (b) P3D-B (c) P3D-C (d) R3D

Figure 9 – Pseudo-3D, R21D and R3D block architectures. Images based on Qiu, Yao and Mei (2017).

which have been shown to improve results (BALLAS *et al.*, 2015).

Qiu, Yao and Mei (2017) proposes a family of bottleneck residual blocks that takes advantage of this approximation. These blocks are called Pseudo-3D (P3D) blocks. Three different P3D block architectures are used in their proposed network. The first block (P3D-A) stacks a spatial and a temporal layer sequentially. The second possibility (P3D-B) is to connect both layers in parallel, granting them both a direct connection to the output of the block. The last proposed option (P3D-C) is a compromise between the previous blocks, that is, while the input of the temporal layer is the output of the spatial layer, the spatial layer also has a direct connection to the output of the block.

A similar architecture is proposed by Tran *et al.* (2018). However, their architecture is homogeneous (with a single block architecture). The so-called R21D convolutions follow an organisation similar to P3D-A blocks, but without using bottleneck layers. Also, R21D blocks can be designed to match the number of parameters on C3D layers. The architectures of these blocks are shown in Figure 9. Traditional three-dimensional convolutions organised into residual blocks (R3D) are also tested by Tran *et al.* (2018).

3.3.1.4 Recurrent network

Recurrent layers, specially LSTM layers, have been used in different types of networks designed for video processing. In this section, we will discuss their use to fuse information learnt by other types of layers (usually, convolutional layers) that appear earlier in the network. The main motivation behind using recurrent networks is that the same operation should be applied at time-step in order to propagate information to the next time-step since the physics of the world remains the same regardless of the input (SRIVASTAVA; MANSIMOV; SALAKHUDINOV, 2015).

Ng *et al.* (2015) consider a recurrent network (5-layer LSTM network) as a replacement to the max-pooling layer. This change causes the network to consider multiple CNN activation as a sequence, which allows it to learn from variations between frames instead of treating the information as a bag-of-features without any particular order. This is interesting from the

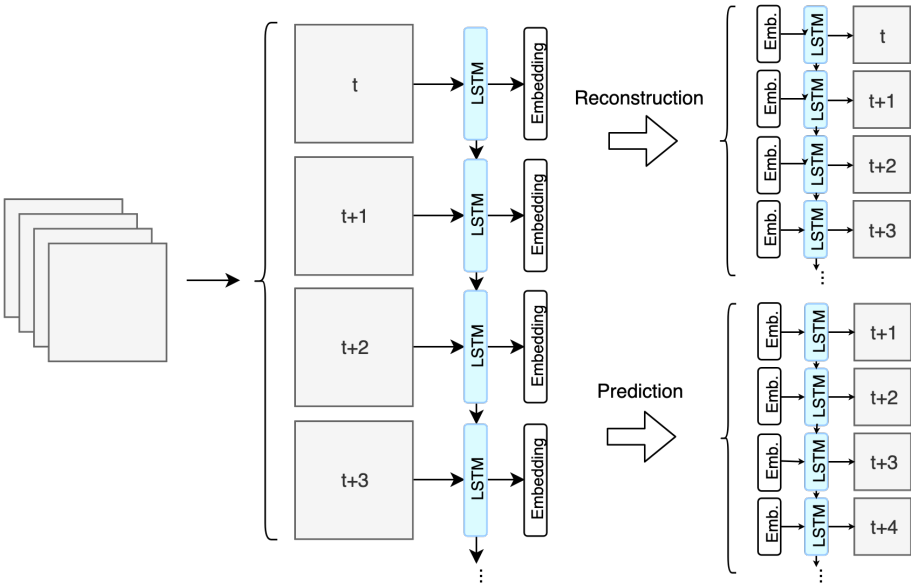


Figure 10 – LSTM autoencoder architecture proposed by Srivastava, Mansimov and Salakhudinov (2015) which performs reconstruction and prediction simultaneously.

video processing perspective since frame order is relevant. The recurrent network is followed by a softmax classifier (shared at every time-step). In a similar manner, Montes *et al.* (2016) combine a pre-trained C3D network with LSTM layers as a way to learn both short and long-term temporal information. A softmax classifier is used to obtain predictions at each time-step. These predictions are then averaged to obtain full video predictions. Both of the mentioned architectures are capable of performing recognition and localisation tasks since the network outputs predictions for each input (frames or video clips).

Srivastava, Mansimov and Salakhudinov (2015) uses LSTM networks to fuse representations extracted using a pre-trained 2D CNN in an unsupervised manner. To achieve this, an autoencoder setting is used, with a single layer LSTM network (encoder) is used to encode the fixed-size representation and another (decoder) is used either to reconstruct the input or to predict future time-steps. A combination of both, input reconstruction and future time-step prediction, is also presented by using two decoder networks. This approach is also used directly on images, making the LSTM network learn both spatial and temporal patterns from the video. Figure 10 shows the LSTM architecture used.

3.3.2 Two-stream networks

Methods discussed in the previous Sections receive stacks of video frames as input and are expected to learn features that include both spatial and temporal information. Two-stream networks (SIMONYAN; ZISSERMAN, 2014) leverage a different architecture that uses two separate networks, one that analyses spatial information from video frames and another that focused only on temporal information. The second network receives dense optical flow images as input, which are previously extracted from each video. The results obtained from both streams

are then combined.

One of the greatest advantages of using two-stream networks is that, by decoupling the spatial an temporal networks, it is possible to leverage the large amount of image data available by pretraining the spatial network in large datasets such as ImageNet (DENG *et al.*, 2009). Also, the use of stacked optical flow displacement fields extracted from several consecutive frames as the temporal network's input makes learning relevant temporal features easier, since it already explicitly provides relevant motion information. On the other hand, two-stream networks are not able to leverage both spatial and temporal information at the same time, that is, they are not capable of identifying the actors of each movement pattern (FEICHTENHOFER; PINZ; ZISSERMAN, 2016). Also, their temporal scale is limited by the number of stacked optical flow frames.

Simonyan and Zisserman (2014) used convolutional networks for both streams and combined them by using late fusion on the softmax scores. They also show that using only spatial data (independent video frames) can provide, by itself, relevant information for action recognition. Furthermore, a discussion is presented about how the temporal network can generalise spatio-temporal local features that derive from optical flow, such as HOF and MBH. Using a similar architecture, Ng *et al.* (2015) state that even temporal networks can benefit from using pre-trained parameters learnt from images, as it helps the network converge faster.

Feichtenhofer, Pinz and Zisserman (2016) tries to overcome the main drawbacks from two-stream models by experimenting with different spatial and temporal fusion at different stages of the networks. Their proposed architecture achieved state-of-the-art results on action recognition by combining information from the temporal stream into the spatial stream after the last convolutional layer (post ReLU activation). The fusion is computed using 3D convolution followed by 3D pooling. The temporal stream is not truncated after the fusion and also uses a 3D pooling after the last convolutional layer. Each stream (spatio-temporal and temporal) has its own loss and classification output. These outputs are combined by summation of the softmax outputs. The results improved when the softmax outputs from the proposed network were combined with IDT by averaging with outputs of a SVM, which shows that there is complementary information between the hand-crafted representations and the information learnt by the networks. Feichtenhofer, Pinz and Wildes (2016) uses convolutional layers organised as residual blocks to build a two-stream network.

Temporal Segment Network (TSN) (WANG *et al.*, 2016) was proposed for action recognition in videos with the main idea of modelling long-term temporal information while maintaining reasonable computational costs by using a sparse temporal sampling strategy and video-level supervision. Different multiple stream configurations were proposed and evaluated in an attempt to extract relevant information while dealing with many different complexities in the data, such as scale variations, viewpoint changes, and camera motion. The final TSN architecture uses a three-stream network to process information from RGB images, stacked optical flow images,

and stacked warped optical flow images. During processing, each video is broken into short snippets which are then sparsely sampled into segments. Each segment is classified by the network and these classifications are aggregated to achieve a video-level prediction. Since the loss values are computed over video-level predictions and passed to the entire network through backpropagation, its parameters are optimised considering information from the entire video (long-term information).

Carreira and Zisserman (2017) proposed Two-Stream Inflated 3D ConvNets (I3D), which leverage state-of-the-art image classification architectures to create 3D convolutional two-stream networks. This is done inflating pre-trained image classification architectures, that is, both their filters and pooling kernels are transformed from two dimensional into three dimensional by replicating the learnt kernels on the new dimension. This allows very deep spatio-temporal networks to be trained, overcoming the size of currently available labelled video datasets and reducing the amount of training required on video data.

3.3.3 Video clip selection

The most common way to achieve a video-level prediction is to combine all predictions obtained by analysing multiple randomly sampled video clips. This is usually done while treating every video clip equally, that is, without using any kind of weighting based on the importance of each clip. However, there is no guarantee as to how representative the random sample video clips are, especially when considering that relevant information may occur sparsely within a video. By using irrelevant video clips to achieve a video-level prediction, the performance of the end-to-end classifier can be hindered (ZHU *et al.*, 2016).

Zhu *et al.* (2016) proposes an architecture and training procedure that considers action recognition in videos as a weakly supervised learning problem, where only video-level labels are available, without precise time windows identification for each label. The proposed framework tries to simultaneously identify key video clips and train a classifier that is not influenced by video clips that irrelevant to the given task. This is done by optimising two separate objects during different stages of the backpropagation procedure.

During the forward pass, a random selection of video clips is passed through the network that identifies the most relevant ones for each class. The backward pass then updates the network parameters considering the output obtained during the first stage. This training procedure is enhanced by an unsupervised key volume proposal algorithm and a “stochastic out” operation that help the network identify the most relevant video clips.

3.4 Concluding remarks

This chapter presented the main approaches used to perform representation learning on video data. Even though many approaches have been proposed, none of them has a clear

performance advantage over the others and there are currently no studies that help select which is the best approach given a dataset and task. Also, the evaluation protocol may not be adequate to understand the behaviour under spatio-temporal representations from videos. This is probably why the literature not yet reached the same performance level as representation learning from images, and so, is considered to be an open research area.

CHAPTER

4

SPATIO-TEMPORAL REPRESENTATION ANALYSIS

4.1 Opening remarks

The main difficulty of working with video datasets is to consider both spatio and temporal information simultaneously. A video can be processed as a set of static images, where spatial information can be extracted. However, when the images are considered to be in a sequence, the evolution of the spatial information creates a temporal dimension which contains relevant information for most video-related tasks. This evolution usually follows certain rules, such as temporal coherence, which can be leveraged when encoding spatio-temporal information, but there are no guarantees that these rules will never be broken. Finding ways to encode both spatial and temporal information from videos into representations is still considered to be an open research area. Evaluating such representations is another important matter still to be investigated.

Different approaches have been proposed to create end-to-end deep learning architectures to model spatio-temporal information from videos. The most common ones use either convolutional layers to model both temporal and spatial information or convolutional layers that deal with spatial information and recurrent layers that encode temporal information. Another way of dealing with videos is to use multiple networks that deal with each type of information separately before combining the learnt information into a single feature space (e.g. two-stream networks). Nonetheless, it is still unclear when each of these network architectures should be used, which are their main advantages and disadvantages and which domains are they suitable for. To the best of our knowledge, no studies have been made to investigate the different ways these architectures encode spatial and temporal information and how well the representations learnt generalise to slight changes in this information. In this chapter, we present a novel evaluation protocol for spatio-temporal representation learning from video data. This is the first experimental pipeline focused on investigating the behaviour of a deep learning architecture when dealing with spatial,

Model	Number of trainable parameters
C3D	1122708
R3D	1209492
R21	1212671
CNN+LSTM	18495208
CNN+LSTM (2)	14055484

Table 1 – Number of trainable parameters per architecture

temporal and spatio-temporal information in videos.

4.2 Experimental setup

The main goal is to understand how state-of-the-art deep learning architectures deal with spatio-temporal information in videos when learning representations. For this, we propose a new experimental pipeline based on a set of three datasets based on the BouncingMNIST dataset (SRIVASTAVA; MANSIMOV; SALAKHUDINOV, 2015). Each of the datasets was designed to expose the representation learning algorithm to a specific spatio-temporal setting while being consistent so that the results are comparable across dataset versions. We configured the datasets as a multilabel problem where each video contains two digits to be recognised. The multilabel setting allows the representations to be organised into groups and subgroups by approximating videos that have at least one label in common, which are then separated into smaller groups based on their second label. The length of videos in each dataset version can be set according to the length required for tested networks. In this case, each video was set to have a length of 16 frames.

The proposed pipeline was used to evaluate the following architectures (all of which are described in Chapter 3): C3D (TRAN *et al.*, 2015), R3D (TRAN *et al.*, 2018), R21D (TRAN *et al.*, 2018), and two versions of CNN+LSTM (SRIVASTAVA; MANSIMOV; SALAKHUDINOV, 2015). The first version of the CNN+LSTM is trained from scratch for both the convolutional network and the LSTM on the training dataset, while CNN+LSTM (2) uses a pre-trained CNN: a MobilenetV2 (SANDLER *et al.*, 2018) pre-trained on the Imagenet dataset (DENG *et al.*, 2009). In a similar manner as proposed in Tran *et al.* (2018), the C3D, R3D and R21D architectures were defined so that they have a similar number of trainable parameters. A batch normalisation layer is used after each convolutional layer on every architecture used for the experiments. These architectures are illustrated in Figure 11 and the number of trainable parameters in each one can be seen in Table 1.

For every architecture, three models are generated, one for each dataset version. These models were named after the dataset they were trained on: *Spatial*, *Temporal* and *Spatio-temporal*; these names indicate which information is relevant for the recognition task. Since all the models

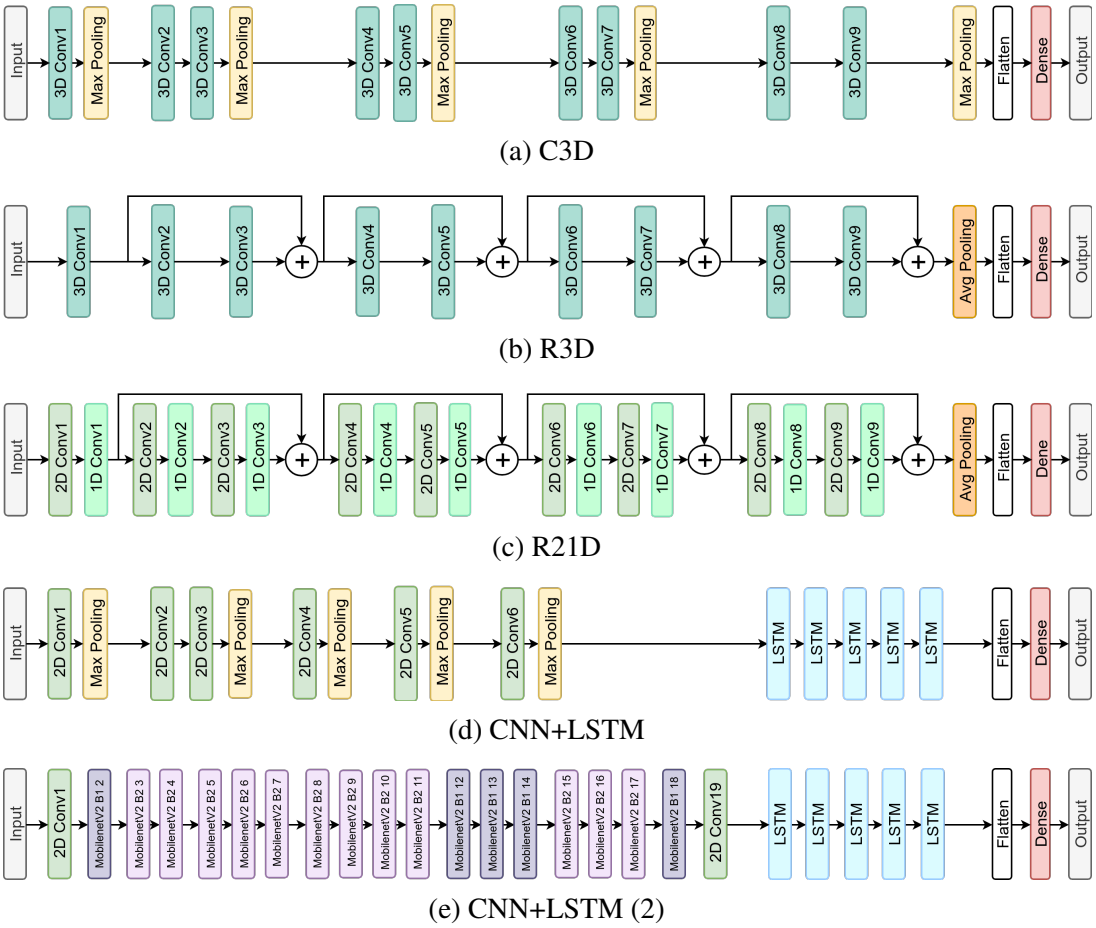


Figure 11 – Network architectures used in the following experiments. Each type of layer or layer block is represented by a different colour. Batch normalisation layers were omitted. The CNN+LSTM (2) architecture uses a pretrained MobilenetV2, in which each box represents a residual block as described in (SANDLER *et al.*, 2018).

are evaluated using every version of the dataset, nine different results are obtained for each architecture, one for every <training dataset version, test dataset version> pair. The pipeline used for these experiments is shown in Figure 12.

We use a model selection scheme where all architectures are trained for the same number of epochs (50) and the model that achieves the best training accuracy during these 50 epochs is used for further evaluation. This is done because different architecture requires a different number of epochs to reach good results and by doing so we believe that the comparison between the results from different architectures will be fairer.

Multiple approaches are used to evaluate the organisation of the learnt feature spaces. First, we classify samples using a linear SVM model trained using a One-vs-Rest context. The quality of these classifiers indicates how linearly separable each class is from the others on a given features space. Then, several different projection algorithms (t-Distributed Stochastic Neighbour Embedding (tSNE) (MAATEN; HINTON, 2008), Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA)) are used to visualise each learnt feature space.

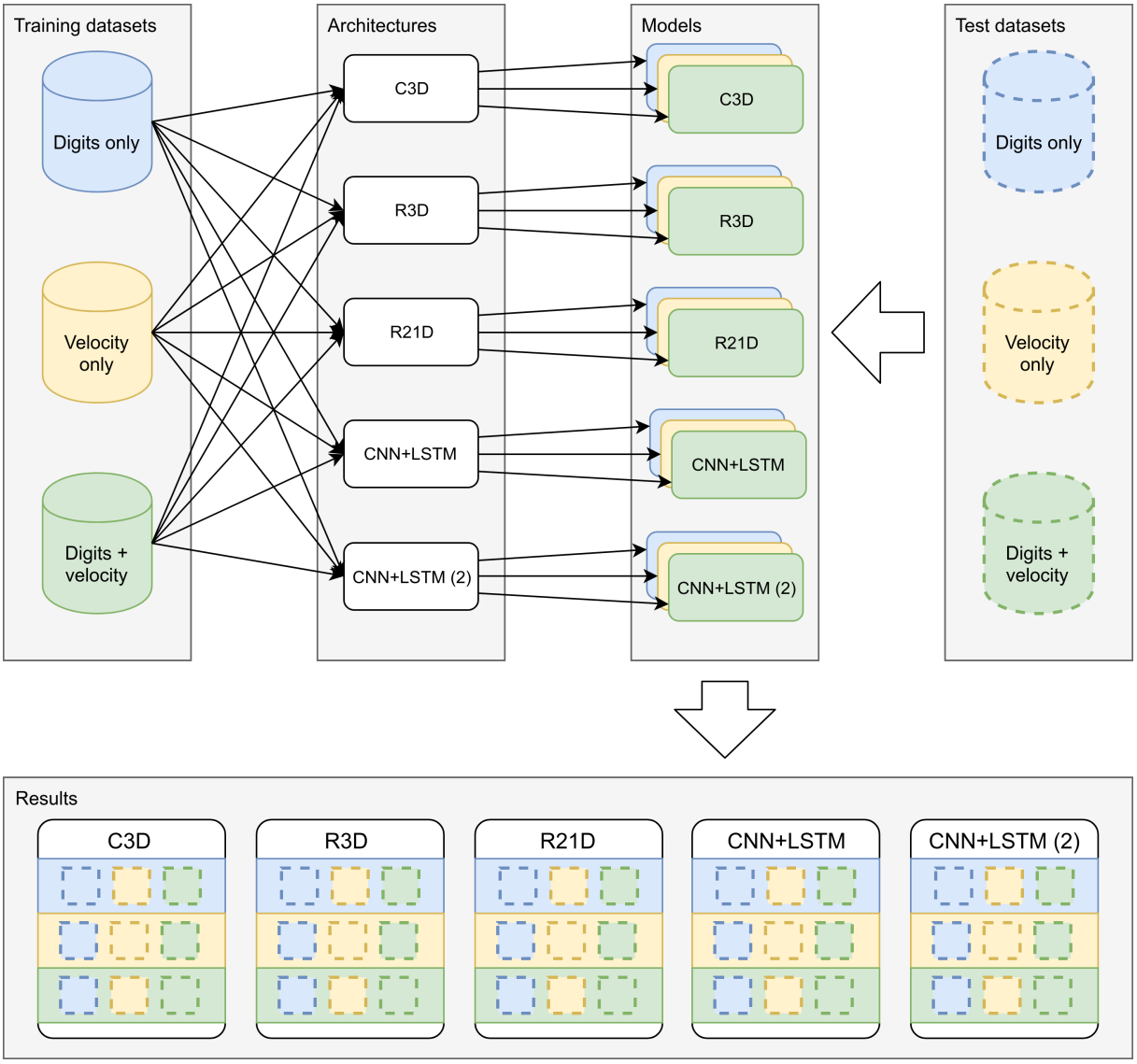


Figure 12 – Experiment pipeline used to obtain the results presented in this chapter. Each training dataset is used to train a model for each of the considered architectures, creating three models for each architecture. Every model is then tested on each one of the three test datasets, producing nine different results for each architecture.

The main goal of the visualisations is to get a better sense of the capability of each architecture to describe and organise spatio-temporal data.

Every model was trained for 50 epochs, with 32 video clips in each mini-batch and using Adam optimisation (KINGMA; BA, 2014). The learning rate was set to 0.0005 for all architectures except the LSTM architectures which used a $5e-6$ learning rate. These were made following literature standards and also based on empirical results.

Since we configured the datasets to address a multilabel character recognition problem, the last layer of all networks was set to have $2 \times n\_classes$ instead of $n\_classes$ as usual. In our setup, each class is represented by a pair of neurons in the output layer, where one neuron indicates the presence of that class while the other indicates its absence. The use of pairs of

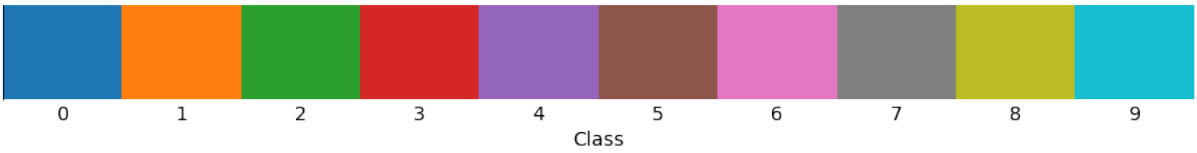


Figure 13 – Colour representation for each digit used for feature space visualisations.

neurons instead of a single neuron for each class removes the need to set a threshold that defines if the output is positive or negative since the highest activation is selected. *Sigmoid* activation is used for the last layer. Also, this setup allows for any number of classes to be assigned by the classifier to every input. For this dataset, $n\_classes = 10$.

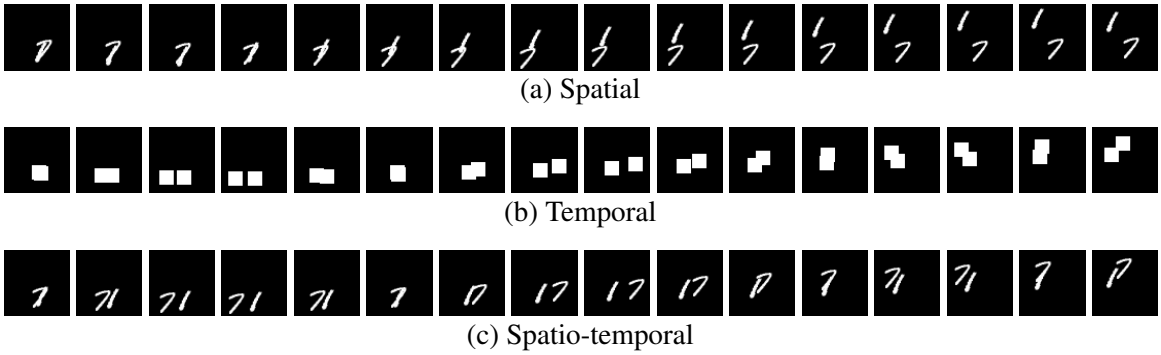
To visualise the feature space in a multilabel setting, each projection is shown twice with different colours for each observation. The labels for each input were placed on a list and sorted; the colour on the first visualisation shows the first label in the list for every input while the second visualisation shows the last label. LDA projections depend on the labels, so the projections are different for the same feature space, however, the way they are generated follows the same procedure as the other visualisations. That is, the first labels are used to create the first projection and the last labels, the second projection. Each colour represents the presence of a digit in the video (Figure 13). If the same digit appears twice, the colour is repeated on both visualisations.

These experiments focus on achieving a better understanding of how well each architecture learns about spatial and temporal information from videos and how they encode this information when it is only partially relevant. We also try to evaluate each architecture's capability of generalising the learnt representations to similar datasets when information is presented in a slightly different way or when different information is relevant to the task at hand.

4.3 Datasets

4.3.1 BouncingMNIST

The BoucingMNIST dataset is a synthetic video dataset created using MNIST digits that move around following random paths and bouncing off the edges. Since it is a synthetic dataset, most of the characteristics can be adapted to fit different cases; some of the characteristics that are adjustable are: resolution, number of digits, digits sampling configuration, digit moving speed and digit size. For these experiments, we use: 64×64 resolution with no digit resizing; 2 digits are randomly sampled for each video. Digits are allowed to overlap, which creates a more challenging digit recognition problem. A multilabel setting with one or two classes per video was selected because it allows the analysis to reach a better understanding of the feature space organisation since it allows the observations to be organised into groups and subgroups by approximating videos that have a single label in common and then splitting them into subgroups



(a) Spatial

(b) Temporal

(c) Spatio-temporal

Figure 14 – Sample video generated for each version of the BouncingMNIST dataset. The main focus of this dataset is to analyse how spatio-temporal representation learning behaves in different settings.

based on the second label.

We propose three different versions of the BouncingMNIST dataset to analyse how spatio-temporal representation learning encode the information present in videos[1]. These versions are:

1. **Spatial:** digits move with a constant velocity c. The only relevant information for digit recognition is the spatial (visual) information;

2. **Temporal:** the velocity of each digit is determined by the a linear function based on their labels ($(digit_label + 1) \cdot c$). Digits are replaced by a 14×14 squares so that temporal information is the only relevant information for digit recognition;

3. **Digits and velocity:** a combination of the previous versions where both temporal (velocity based on label) and spatial (digit appearance) information are relevant for digit recognition;

To make results from different versions more comparable, random seeds were set to a fixed value so that the n-th video of every version of the dataset is generated with the same randomly sampled digits and follow the same random paths. This reduces the influence of random factors included during the data generation. An example of a video generated on all three versions of the dataset can be seen in Figure 14.

It is important to notice that the only difference between the training and test splits are a pool of digits used for random sampling. That is, when random sampling digits to compose a video, the digits are selected from either the training or test splits of the MNIST dataset depending on the split being generated (training or test). This means that there are no differences between the training and test sets of the Temporal version of the dataset.

[1] The source code used to generate these datasets is available at <https://github.com/gbpcosta/video_representations>.

Model	Spatial	Temporal	Spatio-temporal
C3D	0.90525	0.98850	0.98900
R3D	0.83825	0.96300	0.97500
R21D	0.88325	0.98325	0.98775
CNN+LSTM	0.78150	0.07675	0.88025
CNN+LSTM (2)	0.50250	0.01875	0.63750

Table 2 – Accuracy obtained by the each architecture by evaluating each model on test set of the dataset version used for training. We select the model used for evaluation by choosing the one that achieved (at the end of an epoch) the best training accuracy.

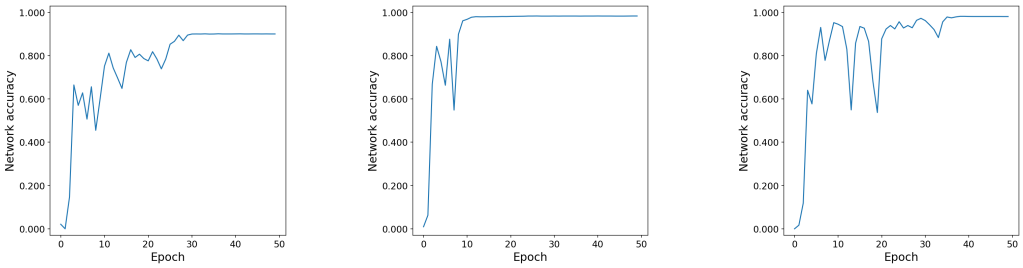
4.4 Results

4.4.1 Intra-dataset analysis

We start focusing on the results obtained by each architecture when they are trained and tested in the same dataset version. As mentioned before, every architecture is trained for 50 epochs and a checkpoint is created at the end of every epoch. The checkpoint that achieves the best training accuracy is selected as the final model. If multiple models achieve the same accuracy, the one from the earliest epoch is selected among those. By doing so, we aim to minimise possible instabilities in the training stage presented by any of the architectures. This model is then evaluated on the test set. The results of this evaluation are shown in Table 2.

The C3D, R3D and R21D architectures had similar performances for the Temporal and Spatio-temporal datasets, with an overall difference of 0.1%. However, R21D achieved better accuracy than the other two architectures when classifying the Spatial dataset – 2.2% and 6.7% higher accuracy than the C3D and R3D methods, respectively. CNN+LSTM and CNN+LSTM (2) had a notably worse performance, especially for the Temporal dataset version. Since the CNNs in these architectures are pre-trained on images, the performance of the final model was probably hindered when no relevant spatial information is available. That is, these results indicate that the LSTM is not able of capturing the temporal information from the features learnt by the CNN when the spatial information is not relevant. Also, the pre-trained network used in CNN+LSTM (2) affected the performance of the final model even further.

In general, processing spatial and temporal information simultaneously seems more adequate when dealing with videos, when compared to processing spatial information followed by a temporal combination of the spatial features. Furthermore, residual blocks had no impact on the final accuracy, however, replacing the 3D convolutions by combinations of 2D and 1D convolutions significantly improved the results.



(a) Spatial (b) Temporal (c) Spatio-temporal

Figure 15 – Performance (per epoch) of the C3D architecture on the test sets. The training and test sets used to evaluate each one of the three models belong to the same dataset version.

4.4.1.1 C3D

As expected, the dataset version containing relevant information for digit recognition only in the spatial dimensions presented a greater challenge than the other versions. This happens because the Spatial version of the dataset has greater intra-class variability. Even so, C3D models achieved high accuracy in all versions of the dataset, nearly reaching a perfect score when relevant temporal information was available. By analysing Figure 15, it is possible to notice how the quality of the learnt representations evolved during the training of each C3D model over 50 epochs. In this figure, each image shows the accuracy obtained by the model at the end of every epoch by evaluating it on the respective test sets.

When both (spatial and temporal) pieces of information were relevant, the C3D architecture created a feature space able to separate the classes. We reach this conclusion due to the architecture's better performance when relevant class information is available as temporal and spatial information in the videos. Moreover, when trained in the Spatio-temporal dataset, this architecture reaches a performance similar to the one obtained with the Temporal dataset, which indicates that temporal information was more important to discern between the classes (as discussed earlier in this chapter). Also, similar fluctuations in the performance when training on the Spatial and the Spatio-temporal versions of the dataset supports our previous conclusion.

Figure 16 shows the visualisations created using tSNE projections of the features extracted from the previous to last layer of the C3D architecture. Each dot represents a video from the test set. Similar to all the other results shown in this Section, these visualisations were created using training and test sets from the same version of the dataset.

By analysing the projections, we noticed that when temporal information is irrelevant for the task, the network has more difficulty in creating a feature space capable of separating the classes. When relevant temporal information was included, the network was able to create representations that separates the data into groups and subgroups based on the classes present in the videos. It is important to consider that the high quality of the feature space obtained when temporal information is relevant most likely occurs due to the simplicity of this information, that is linearly proportional to the class and has zero intra-class variability.

4.4.1.2 R3D

The R3D architecture is similar to C3D, however, its layers are organised as residual units. For the datasets considered in this experiment, the performance of the R3D architecture was inferior to the obtained by C3D, which also achieved more stable results during the experiments. These results can be seen in Figure 17, where the network's accuracy on the test set is shown at the end of each epoch, for each of the dataset versions.

Overall, the behaviour of the R3D architecture was similar to C3D: greater difficulty in classifying the Spatial version of the dataset and high accuracy scores on the Temporal and Spatio-temporal versions. However, in comparison to the C3D architecture, R3D was more unstable during the training on Temporal and Spatial than when training on Spatio-temporal. That is, the R3D architecture was more stable when both spatial and temporal information were relevant to the task.

Multiple visualisations were created using PCA projections of the features from the penultimate layer of each R3D model. These visualisations can be seen in Figure 18. It is possible to see that by including temporal information the learnt feature space was able to increase the separability between classes. Also, it is interesting to notice that the changes to the feature space when temporal information is relevant occur primarily in a limited number of directions (similar to vectors), starting at a central point where there is a higher density of samples. Even though the transformation made to the feature space by the PCA method is linear, we notice that, even when looking at only two dimensions, the classes show considerable separability and organisation into groups and subgroups based on the classes in each video. This fact is even more evident when using Temporal and Spatio-temporal.

4.4.1.3 R21D

Among the architectures tested in this experiment, R21D achieved the best overall performance by a small margin (Table 2), only losing on the Temporal version of the dataset. When looking at the performance of the R21D architecture overall 50 training epochs (Figure 19) and comparing the results to the ones obtained with C3D and R3D architectures, we noticed that R21D had more drastic variations on the performance over time, however, these variations happen less frequently.

Including relevant temporal information helped to stabilise the performance of the network during training, achieving the best accuracy after a smaller number of epochs. This behaviour is similar to what was observed with previous architectures. When both spatial and temporal information where relevant, the behaviour of the R21D architecture was a combination of what was seen with the C3D and R3D architectures, where small fluctuations on the performance happen more frequently. Similar to other analysed architectures, once the maximum accuracy achieved by the method was reached, it stabilised.

To help better understand the learnt feature space, tSNE projections are shown in Figure 20. As expected (and discussed in Section 4.3), relevant temporal information made the problem of learning a feature space where the classes are separable easier. When compared to tSNE projections obtained with C3D, it is possible to see that the R21D architecture was able to build a space with better organisation for Spatial. This conclusion is reassured by the increase in the accuracy obtained by R21D for this dataset version. For the other versions, R21D also showed better structuring of the feature space into groups and subgroups, increasing, in most cases, the gap that separates a group/subgroup from others.

4.4.1.4 CNN+LSTM

When dealing with architectures that combine CNNs with LSTMs, training was performed in two stages. First, the CNN network is trained for 50 epochs to identify digits in individual frames randomly selected from videos. The trainable parameters in this network are frozen and the last (softmax) layer is discarded. Then, the LSTM network is trained by using the previously trained CNN to extract features from each frame, which are passed to the LSTM as time steps in a time series. Figure 21 shows the performance of the CNN+LSTM architecture during both stages of the training procedure. These images show that the performance of the CNN+LSTM architecture was unstable during training and that it achieved results similar to the ones obtained by the CNN independently. We believe that this occurs because the CNN network was not capable of passing enough relevant information to the LSTM for it to learn about each digit's velocity. This is even more clear when looking at the performance of this architecture for Temporal. Also, this architecture showed to be the hardest to find adequate hyper-parameters. While all other architectures were reasonably stable and achieved similar results for similar hyper-parameters, CNN+LSTM's performance varied largely when making small changes to hyper-parameters such as the learning rate.

When looking at the projections obtained using the LDA method on the features extracted from the LSTM network, we noticed that when spacial information is relevant to the task at hand, the feature space learnt by the CNN achieves a higher separability between classes than the entire CNN+LSTM architecture. However, the projections also show that the feature space created by the CNN+LSTM architecture was capable of encoding temporal information when trained on the Temporal version of the dataset. We believe that it is possible to improve the results achieved in these experiments by making a broader search for hyper-parameters and architecture combinations (for both networks, CNN and LSTM). Moreover, for the configuration used in these experiments, CNN+LSTM was not able to leverage the available information when both, temporal and spatial information, were relevant.

4.4.1.5 CNN+LSTM (2)

Replacing the CNN network used in the previous model (CNN+LSTM) with a MobileNetV2 network (SANDLER *et al.*, 2018) pre-trained on the Imagenet dataset changed the model's behaviour. Figure 23 shows that the pre-trained CNN training process becomes more stable, which indicates that the representations extracted using the CNN model and passed to the LSTM network may be one of the main causes of the instability presented by the previous model. Even though CNN+LSTM (2) was more stable than CNN+LSTM, CNN+LSTM (2) did not achieve the same performance level as the CNN+LSTM. Another interesting point is that the second combination had a worse performance even for the Temporal version of the dataset, which shows that the CNN trained specifically for this dataset was capable of learning relevant information, such as the position of the squares inside each frame.

The features extracted from the previous to last layer of the CNN+LSTM (2) architecture were extracted and used to create LDA projections, which can be seen in Figure 24. The projections for the CNN features extracted from the Temporal version of the dataset were not included because the LDA method did not converge. We believe this happened because the features extracted by the network could not encode the variability present in the original data. Nevertheless, the LSTM network was able to infer some information about the class from the features of the CNN, as seen by the organisation of the samples in the LSTM feature space in Figure 24b. Also, by comparing these projections with the ones obtained by the previous model (CNN+LSTM), we noticed that the pre-trained CNN feature space provided less intra-class separability than the CNN trained specifically for this dataset. The quality of this feature space was probably the main reason why the CNN+LSTM (2) architecture did not achieve the same performance as the CNN+LSTM architecture.

4.4.2 Cross-dataset analysis

In this experiment, we evaluate the capability of each architecture to generalise the knowledge acquired from one version of the dataset to each other version. To this end, we select the model that achieved the best accuracy during the training stage. That is, for each architecture, three models are selected with respect to the evaluation performed at the of each epoch in the training stage, wherein the evaluation is done using the training set. When more than one model achieves the same accuracy, we select the model that corresponds to the earliest epoch.

The results obtained by each model trained in the Spatial version of the dataset can be seen in Table 3. In this case, all architectures considered in the experiment were capable of generalising the information learnt to the Spatio-temporal version. However, the performance of the C3D, R3D and R21D architectures fell 9.5% on average. Among these, the C3D architecture achieved the best generalisation to Spatio-temporal (reduction of 5.9% on the accuracy). On the other hand, the CNN+LSTM and CNN+LSTM (2) architectures had an increase in accuracy when comparing the results for the versions Spatial and Spatio-temporal (0.9% and 3.3%, respectively).

Model	Training set: Spatial		
	Spatial	**Temporal**	**Spatio-temporal**
C3D	0.90550	0.02675	0.37225
R3D	0.84675	0.02175	0.28750
R21D	0.87725	0.01525	0.34475
CNN+LSTM	0.83150	0.01075	0.84075
CNN+LSTM (2)	0.60125	0.01750	0.63375

Table 3 – Accuracy obtained by the "best" model trained on Spatial for each architecture and evaluated on the test set of every dataset version.

We believe that this happens because the CNN networks were capable of capturing the relevant spacial information in a way that when temporal information was relevant, the performance was not hindered.

The performance of all architectures behaved as expected when evaluated on Temporal. Since there is no intersection between the relevant class information present in the training set (Spatial) and test set (Temporal), the accuracy achieved by all models was significantly low, reaching an average of 1.4%.

Table 4 shows the results obtained by each model trained on Temporal and evaluated on every version of the dataset. Differently to what was observed when the models were trained on Spatial, in this case, none of the models were capable of generalising the learnt temporal information to the Spatio-temporal version in a satisfactory manner. The best performance was achieved by the C3D and R3D models, reaching 13.9% and 14.7%, respectively. As discussed when analysing the previous set of experiments, it was expected that the accuracy when training on Temporal and evaluating on the Spatial would be low, since there is no intersection between the relevant information present in both versions.

The CNN+LSTM and CNN+LSTM (2) models were the ones that presented the worst results among the models addressed in these experiments, which likely happened because the CNNs were not capable of passing relevant information to the LSTM, making it hard for the LSTM to learn about the temporal evolution of each video. However, based on the analysis shown in the previous section, we expected that the CNNs passed information about the position of the squares in each frame of the video, so the LSTM network would be able to infer the velocity of each object and, consequently, their classes; which did not happen. This indicates that the CNNs learnt to track squares and that this knowledge was not extended to other objects in the image.

Finally, the results obtained by the models trained on Spatio-temporal are shown in Table 5. Even though the models were trained with both (spatial and temporal) relevant information, none of the methods were able to generalise that knowledge to any of the other dataset versions, where only one of the two types of information was relevant. This shows that all the architectures

Model	Training set: Temporal		
	Spatial	Temporal	Spatio-temporal
C3D	0.02050	0.98900	0.13925
R3D	0.01100	0.96875	0.15650
R21D	0.01225	0.98025	0.05550
CNN+LSTM	0.01350	0.10975	0.07000
CNN+LSTM (2)	0.00825	0.02350	0.00975

Table 4 – Accuracy obtained by the "best" model trained on Temporal for each architecture and evaluated on the test set of every dataset version.

Model	Training set: Spatio-temporal		
	Spatial	Temporal	Spatio-temporal
C3D	0.04875	0.23025	0.98500
R3D	0.04375	0.24100	0.97250
R21D	0.05250	0.16925	0.98450
CNN+LSTM	0.76225	0.00850	0.57650
CNN+LSTM (2)	0.59600	0.01525	0.56025

Table 5 – Accuracy obtained by the "best" model trained on Spatio-temporal for each architecture and evaluated on the test set of every dataset version.

addressed in these experiments rely on a combination of spatial and temporal information to classify the objects correctly, and if one information domain presents some kind of variation, the network will not be able to generalise the knowledge learnt accordingly.

When considering the generalisation capability of the architectures considered in this experiment, the CNN+LSTM and CNN+LSTM models were the ones that reached the highest accuracy on Spatial (76.2% and 59.6%, respectively). Their performance is probably due to the CNN networks being responsible to only learning spatial information, allowing the LSTM network to "ignore" the temporal information without any changes to the architecture. Other models, that use convolutional operations to encode temporal information, were not capable of generalising the knowledge to any of the other versions of the dataset.

The results shown in this section demonstrate the difficulty of spatio-temporal representation learning methods to generalise the knowledge encoded even when the datasets are very similar. They also show that this difficulty is heightened when both temporal and spatial information are processed simultaneously and when the temporal information varies from the training set to the test set. These results led to another experiment shown in Section 4.4.3, where we aim to evaluate if spatio-temporal representation learning methods can generalise for cases where there is a consistent variation in the temporal information present in the videos.

Model	Training set: Velocity–		Training set: Velocity+	
	Velocity–	Velocity+	Velocity–	Velocity+
C3D	0.98500	0.03050	0.03450	0.96525
R3D	0.97250	0.01500	0.05100	0.94325
R21D	0.98450	0.01850	0.06175	0.96750
CNN + LSTM	0.80600	0.85925	0.73125	0.88025
CNN + LSTM (2)	0.64775	0.70175	0.60075	0.73400

Table 6 – Analysis of the generalisation capability of the spatio-temporal representation learning methods to consistent changes on temporal information. This experiment considers two different settings of the Spatio-temporal dataset where digits in the setting **Velocity+** move 3 times faster than digits in **Velocity–**.

4.4.3 Velocity variation analysis

One of the most common ways of dealing with the high amount of information in a video is to perform undersampling on the frames. On average, each video contains 25 frames per second, which makes it necessary to process a high number of images (frames) to cover a long temporal window. The number of images that needs to be analysed so that the temporal information is encoded makes the problem of video processing computationally expensive. A common solution that reduces the computational cost of video analysis is to select a single frame at each time interval (e.g. every second).

One of the main problems caused by this undersampling of frames is the risk of losing continuity (temporal coherence), especially for faster moving objects. In an attempt to simulate this effect, we modified the Spatio-temporal version of the dataset so that the movement of the digits from one frame to the next is more or less drastic (**Velocity+** and **Velocity–**, respectively). The velocities in these settings vary by a factor of 3 (i.e. $\textbf{Velocity}+ \leftarrow 3 \times \textbf{Velocity}--$). Models were trained and tested using both settings and their results are shown in Table 6.

The results show that the models that use convolution to deal with temporal information (C3D, R3D and R21D) are not capable of generalising what they learnt from one setting to the other, unlike the CNN+LSTM and CNN+LSTM (2) models. Also, the results indicate that the CNN+LSTM and CNN+LSTM (2) models are more suited for tasks that deal with fast-moving objects.

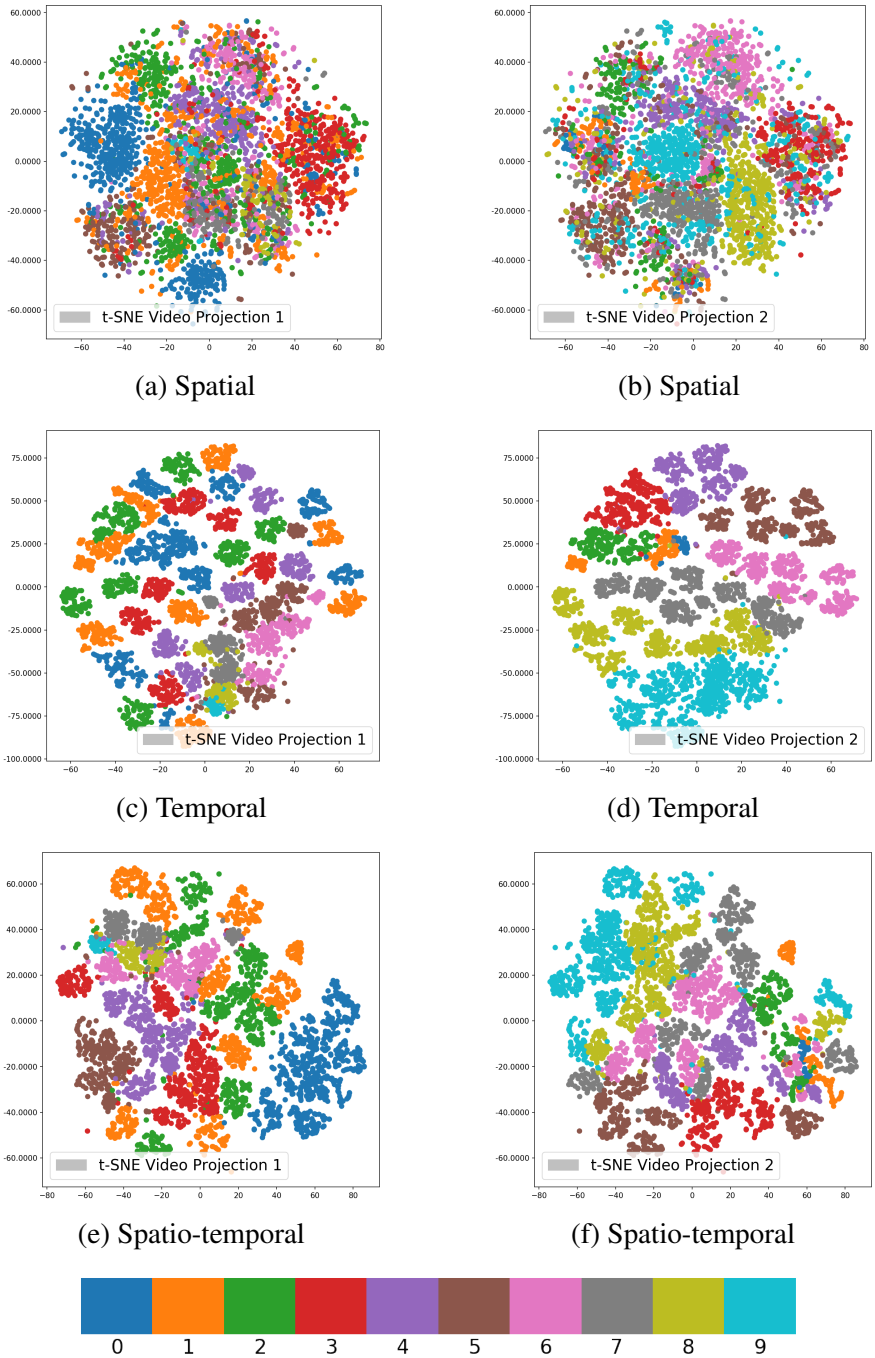
4.5 Concluding remarks

In this chapter, we present a novel evaluation protocol for models that learn spatio-temporal representations from videos. We show that, when learning representations from videos, it is important to process spatial and temporal information during all stages of learning rather than first learning spatial features and only then offer this to a model that incorporates temporal information. However, currently, the most common methods that perform spatio-temporal

learning rely on convolutions, which fails to generalise for the same spatial data with different temporal characteristics, pointing out to the need for future studies.

Important evidence is given on how different methods behave when learning from videos. Although the different studied approaches showed potential in particular scenarios, a gap still exists — especially in terms of generalisation — when comparing methods that learn from images. Different architectures and training strategies can be analysed using the presented protocol, allowing researchers to reach a better understanding of the pros and cons of each architecture/method. We believe exploring new ways of incorporating the temporal information along all stages of training is the next step to significantly improve the current performance of spatio-temporal representation learning methods.

We take a first towards our hypothesis with the proposed evaluation pipeline, which focuses on achieving a better understanding of how spatial and temporal information is encoded by different representation learning architectures. The proposed evaluation framework sheds a new light on the representations learned from video. The comparison of the results on the pipeline for different architectures shows that including spatial and temporal information encoding in all the stages of representation learning improves the learnt feature space. However, encoding spatial and temporal information in two consecutive stages improves the capability of the learnt feature space to changes in the temporal dimension.

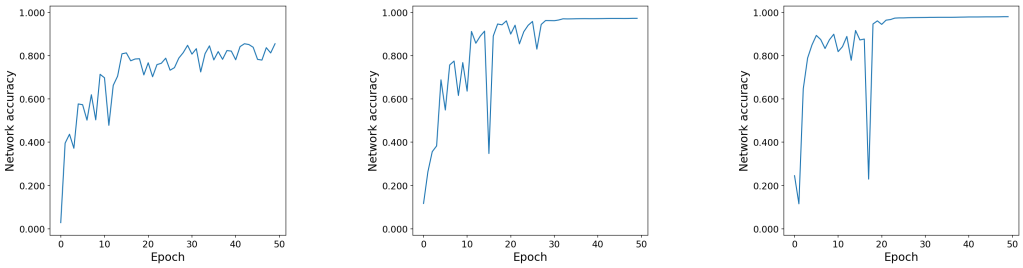


(a) Spatial (b) Spatial

(c) Temporal (d) Temporal

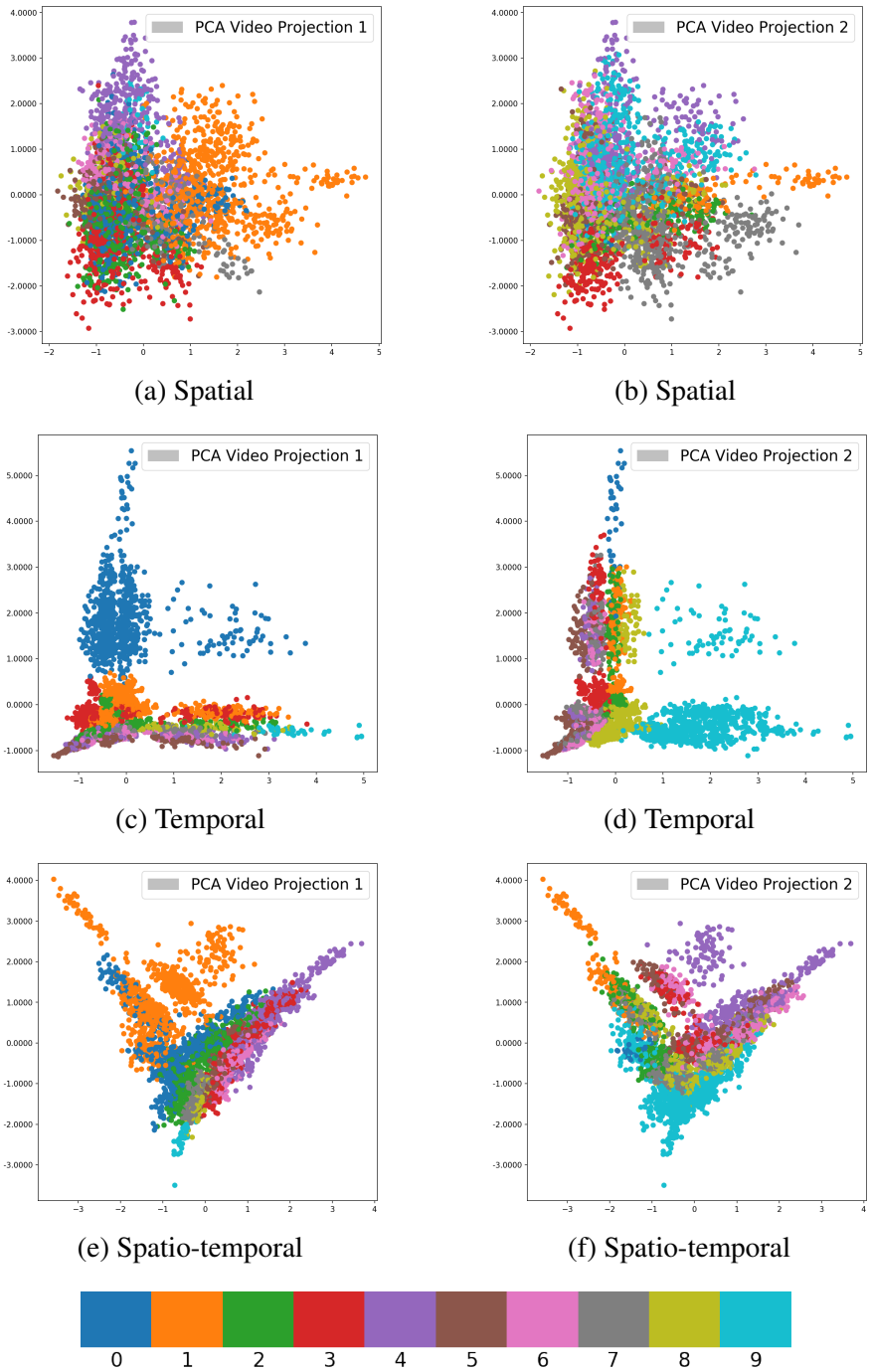
(e) Spatio-temporal (f) Spatio-temporal

Figure 16 – tSNE projections obtained from the representations extracted from the previous to last layer of the C3D architecture for each model (dataset version). Both training and test sets belong to the same dataset version. Due to the analysis being done in a multilabel setting, each projection is shown twice, where the colour of each dot indicates the class it belongs to (classes from each video are sorted so that the first projection – left column – shows the colour of the smallest class, and the right column shows the colour of the largest class present in the video).



(a) Spatial (b) Temporal (c) Spatio-temporal

Figure 17 – Accuracy (at the end of each epoch) of the R3D architecture on each test sets. Both the training and test sets belong to the same dataset versions.

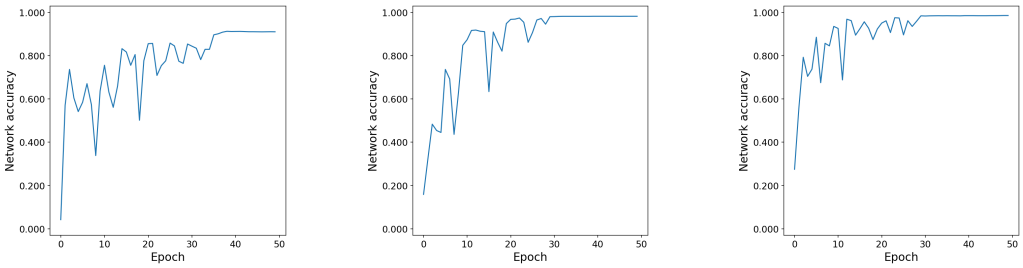


(a) Spatial (b) Spatial

(c) Temporal (d) Temporal

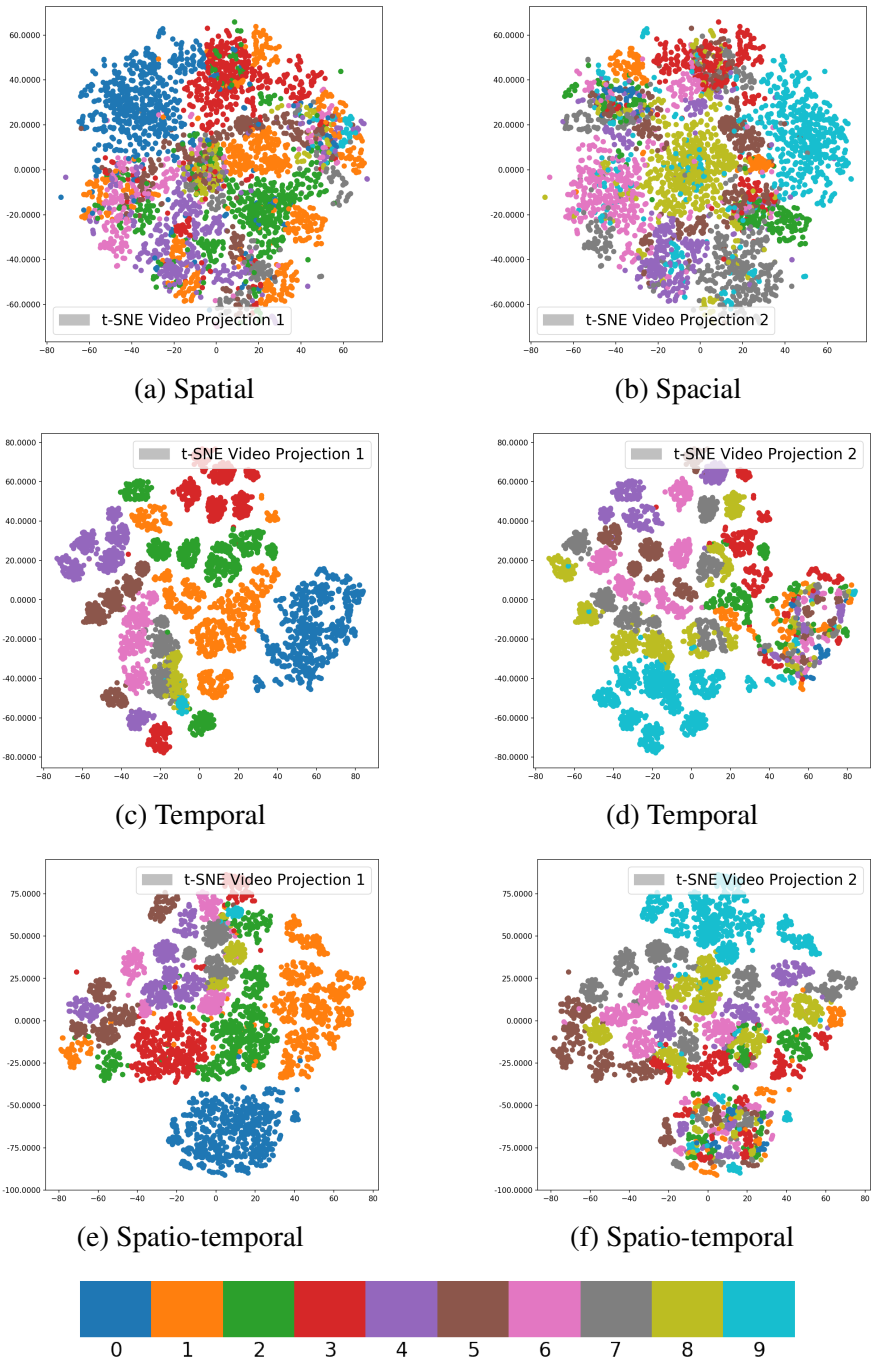
(e) Spatio-temporal (f) Spatio-temporal

Figure 18 – PCA projections obtained from the representations extract from the activation of the penultimate layer of each R3D model, using the test set of each version of the dataset. Due to this being a multilabel problem, all projections are shown twice. Images in the first column (left) show the colour of the dots that indicate the lowest class in the video, while the second column shows the colour of the class representing the highest digit in the video.



(a) Spatial (b) Temporal (c) Spatio-temporal

Figure 19 – Accuracy (at the end of each epoch) of the R21D architecture when tested on the test set of each version of the dataset. Both training and test sets belong to the same dataset version.

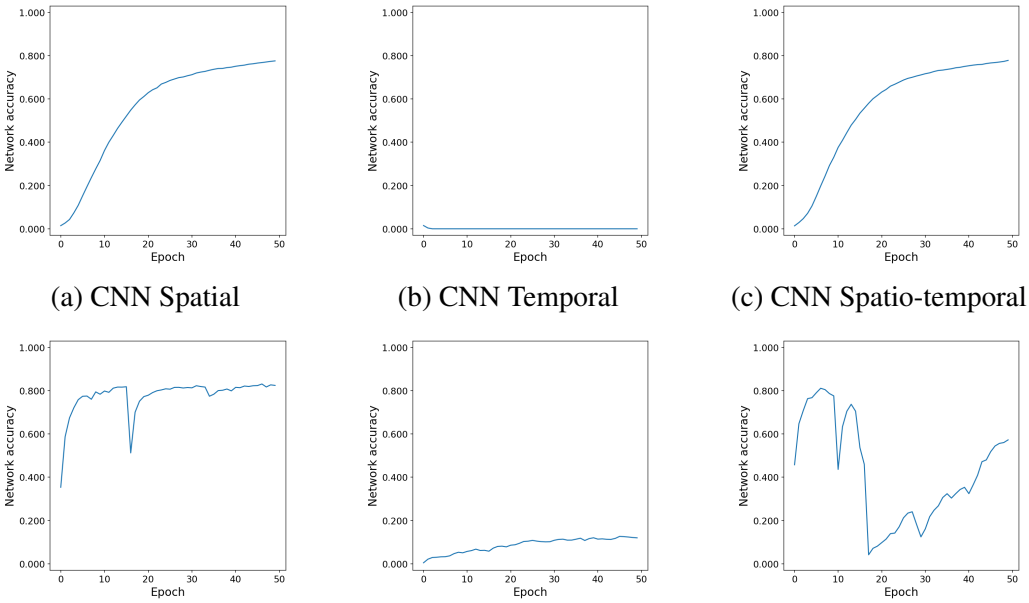


(a) Spatial (b) Spacial

(c) Temporal (d) Temporal

(e) Spatio-temporal (f) Spatio-temporal

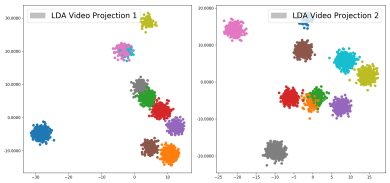
Figure 20 – tSNE projections obtained from representations extracted from the previous to last layer of the R21D architecture for each model. Both training and test sets belong to the same dataset version. Due to the analysis being done in a multilabel setting, each projection is shown twice, where the colour of each dot indicates the class it belongs to (classes from each video are sorted so that the first projection – left column – shows the colour of the smallest class, and the right column shows the colour of the largest class present in the video).



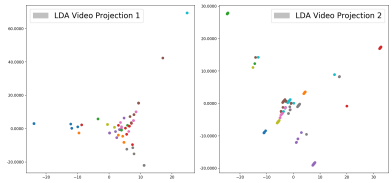
(a) CNN Spatial (b) CNN Temporal (c) CNN Spatio-temporal

(d) Spatial (e) Temporal (f) Spatio-temporal

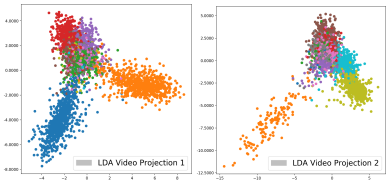
Figure 21 – Performance (per epoch) of the CNN+LSTM architecture on test sets. The training and test sets used to evaluate each one of the three models belong to the same dataset version.



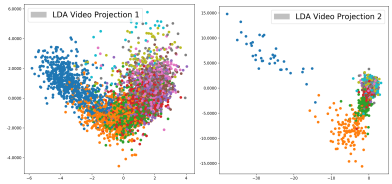
(a1) CNN



(b1) CNN



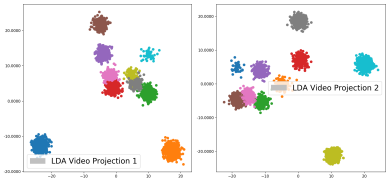
(a2) CNN+LSTM



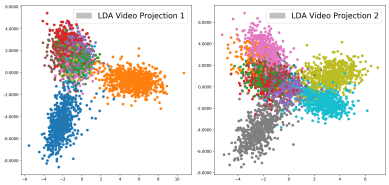
(b2) CNN+LSTM

(a) Spatial

(b) Temporal



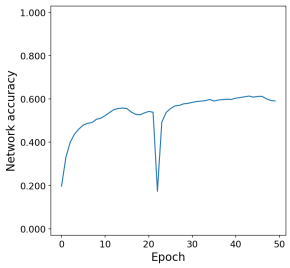
(c1) CNN



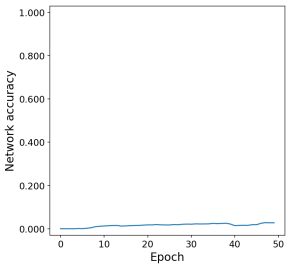
(c2) CNN+LSTM

(c) Spatio-temporal

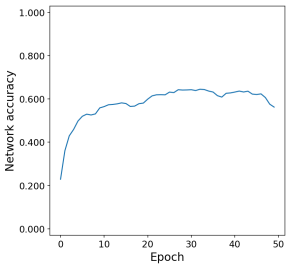
Figure 22 – LDA projections obtained from the representations extracted from the previous to last layer of the CNN+LSTM architecture for each model (dataset version). Both training and test sets belong to the same dataset version. Due to the analysis being done in a multilabel setting and LDA being a supervised projection method, two different projections are shown. The colour of each dot indicates the class it belongs to. Classes from each video are sorted so that the first projection – left column – shows the colour of the smallest class, and the right column shows the colour of the largest class present in the video.



(a) Spatial

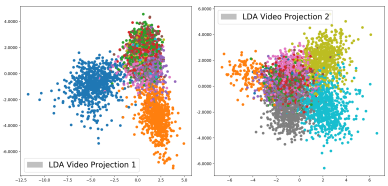


(b) Temporal

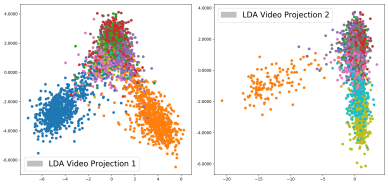


(c) Spatio-temporal

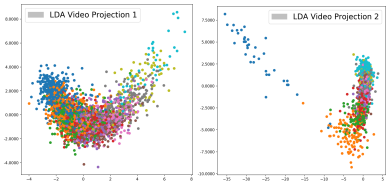
Figure 23 – Performance (per epoch) of the CNN+LSTM (2) architecture on the test sets. The training and test sets used to evaluate each one of the three models belong to the same dataset version.



(a1) CNN



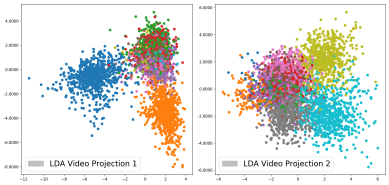
(a2) CNN+LSTM



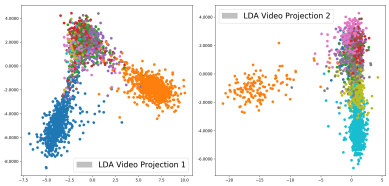
(b2) CNN+LSTM

(a) Spatial

(b) Temporal



(c1) CNN



(c2) CNN+LSTM

(c) Spatio-temporal

Figure 24 – LDA projections obtained from the representations extracted from the previous to last layer of the CNN+LSTM (2) architecture for each model (dataset version). Both training and test sets belong to the same dataset version. Due to the analysis being done in a multilabel setting and LDA being a supervised projection method, two different projections are shown. The colour of each dot indicates the class it belongs to. Classes from each video are sorted so that the first projection – left column – shows the colour of the smallest class,and the right column shows the colour of the largest class present in the video. The projections for the CNN features extracted using the Temporal version of the dataset (b1) are not shown because LDA did not converge.

CHAPTER

5

REPRESENTATION GENERALISATION ANALYSIS

5.1 Opening remarks

One of the desirable characteristics of feature extraction or representation learning algorithms is for it to extract features that can generalise to multiple situations, such as different datasets, domains or tasks. This allows the same algorithm to be applied to different settings without requiring humans to input specialised knowledge or to adapt the representations in any way.

In this chapter, we evaluate the generalisation capabilities of a representation learning architecture and compare it to the state-of-the-art hand-crafted feature extraction algorithm. This is done through four experiments, each one using a different dataset. The selected datasets cover two tasks (action recognition and dynamic scenes recognition) with different characteristics, such as dataset size, video resolution and camera motion. The main goal of these experiments is to analyse the quality of the feature space created by each feature extraction method in multiple tasks and settings. These experiments complement the analysis presented in the previous chapter by looking at real-world datasets and comparing the results of a selected representation learning architecture and the state-of-the-art hand-crafted feature extraction method on the task it was proposed for and a second task unknown for both methods.

5.2 Experimental setup

In this set of experiments, we selected two state-of-the-art methods were to extract features from four different datasets. Later, the resulting feature vectors were fed to linear SVMs for classification purposes. The goal of these experiments is to analyse the linear separability of the features extracted and their generalisation capabilities to different datasets, domains and tasks.

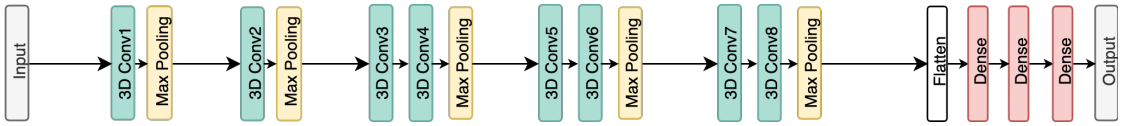


Figure 25 – C3D architecture pretrained in the Sports-1M dataset used in the experiments. This architecture contains 8 convolution, 5 max-pooling, 3 fully connected layers, wherein the last fully connected layer uses a softmax activation function. The convolution kernels were set to $3 \times 3 \times 3$ while using a stride of 1 in both spatial and temporal dimensions. The number of units in each layer is specified in each box. The 3D pooling layers use kernels of size $2 \times 2 \times 2$, with the exception of the first pooling layer, whose kernel size is $1 \times 2 \times 2$. The softmax output layer contains 487 units, one for each class in the original task, whereas the previous dense layers have 4096 units each. The pretrained model is provided by Tran *et al.* (2015).

The chosen methods were: improved trajectories encoded by Fisher vectors (ONEATA; VERBEEK; SCHMID, 2013) (IDT-FV) and three dimensional convolutional neural networks (TRAN *et al.*, 2015) (C3D).

IDT-FV is considered to be the state-of-the-art hand-crafted feature extraction method for videos and uses the standard approach to video classification: local visual features are used to describe a region of the video surrounding a set of interest points which are then combined into a fixed-sized video level description. The C3D method is, to the best of our knowledge, the only representation learning method that focused on producing generic representations. A pre-trained neural network trained on the Sports-1M dataset (Section 5.3.1.3) for a sports recognition task was used to extract the features. The architecture of this network is illustrated by Figure 25. By using a network that was trained in an entirely different dataset for a different task, it is possible to evaluate how generic the learnt representations are. This approach also highlights the possibility of leveraging large datasets to address the lack of labelled data in certain tasks

To extract features using a pre-trained C3D network, a video is split into blocks containing a predefined number of frames. The user may choose how densely these blocks are sampled by defining the stride between each block, that is, the number of frames skipped when starting a new block. If the stride is smaller than the number of frames in a block, the blocks will overlap. Features are extracted by passing a block through the network and collecting the activation of one or more fully connected layers, which are then concatenated to create the feature vector that describes that block. In this experiment, the network was designed to use 16-*frame blocks* with stride 1 and the features were extracted using all three fully connected layers, which resulted in a descriptor of size 8679.

Since IDT-FV extracts a variable number of features from each video and C3D is capable of extracting features only from fixed-size windows, it is necessary to use a combination or quantisation method to reduce the number of features to a fixed-size feature vector that describes the entire video. For IDT-FV, Oneata, Verbeek and Schmid (2013) proposed the use of Fisher Vector (FV) (SÁNCHEZ *et al.*, 2013) together with a Spatial Pyramid Matching (SPM)

representation as an alternative to bag-of-words histograms, and achieved state-of-the-art results using fewer dimensions and linear classifiers. FV is an extension of the Bag-of-Visual-Words (BOV) that records, for each quantisation cell, the number of assigned descriptors, their mean and their variance, for each dimension. This results in a feature vector with dimension $K(2D+1)$, where K is the number of quantisation cells and D is the dimension of the descriptors. Instead of using k-means clustering to create the dictionary like the BOV method, FV uses a Gaussian Mixture Model (GMM).

The setup proposed by Oneata, Verbeek and Schmid (2013) also applies a Principal Component Analysis (PCA), before computing the FV, and selects the first 64 components. This speeds up the FV computation and, since FV size scales linearly with feature dimension, reduces the size of the final descriptor. PCA is also useful because it decorrelates the data, improving the fit of the diagonal co-variance assumption from the Gaussian components. PCA and GMM are both fitted on a subset of descriptors from the training set. Finally, a ℓ_2 normalisation is applied over all descriptors extracted from a video. In our experiments, a similar setting was used, where MBH extracts local information for IDT, 64 dimensions were selected using PCA and a dictionary with $k = 1000$ words was created using GMM.

Spatio-temporal grids were used to include a weak notion of spatio-temporal location of the local features into the final descriptor. This was done by using three spatial pyramid settings: $(1,1,1)$, $(1,1,2)$, $(1,3,1)$, where the first number indicates how many splits were made in the temporal dimension, the second is related to the number of vertical splits and the last one, to horizontal splits. Spatial Fisher Vectors (SFV), which computes the mean and variance of the 3D spatio-temporal location per visual word, of the assigned features, was also used. SFV provides similar information modelling as the one achieved by extending the feature vectors to 3D locations (ONEATA; VERBEEK; SCHMID, 2013). These methods are combined by computing SFV in each SPM cell. The final descriptor is obtained by concatenating all the extracted feature vectors.

For C3D, Tran *et al.* (2015) used the average of the representations extracted from all blocks of frames followed by a ℓ_2 normalisation to generate the representation vector for an entire video. In this experiment we also explore other combination methods besides averaging for C3D features: **voting** (when applicable), where each frame-block casts a vote for the class of the entire video; **k-means quantisation**, where the k-means algorithm is used to create a dictionary based on the representations of frame-blocks and the final descriptor is a normalized histogram of the words in the video; **statistical measures**, which concatenates the average, standard deviation, skewness, kurtosis, maximum and minimum computed using all representations extracted from a video. When applicable, we also classified directly the representation of each block. This allows for an analysis of the performance of the representations without the influence of a combination method.

For the classification stage, linear Support Vector Machine (SVM)s (BOSER; GUYON;

VAPNIK, 1992) were used due to the guarantees it provides regarding the VC-dimension and the selection of the maximum margin hyperplane. Since SVMs are designed to work in a binary classification context, a "one-vs-all" setting was used where a classifier was trained to distinguish the observations belonging to a single class from all remaining classes (RIFKIN; KLAUTAU, 2004). When a new observation needs to be classified, all classifiers are used and the class indicated by the one which outputs the most positive value is chosen. The choice of not using kernels for the SVM was made so that the SVM's performance shows how linearly separable the classes are, given a feature space. That way, it is possible to use the results of the classifications to directly evaluate representation quality.

A grid search approach was used to determine the parameters of the SVM classifier. This was done by comparing the average results of 5-fold cross-validation experiments using the training set. That is, the training set was split into five-folds, four of which were used to train a classifier and the other to evaluate its performance. After cycling through all folds, the average performance was estimated by using an evaluation measure, such as the F1-score. This was done for each possible parameter combination in a set of predefined parameters and the combination with the highest average efficiency was selected for later use.

For linear SVMs, the only parameter that needs to be set is the penalty factor C, which determines how soft is the margin of the SVM. In other words, the C parameter defines how important it is to avoid misclassifying training observations. When C is set to a high value, a hyperplane that better classifies observations from the training set is preferred. Accordingly, when a small value is selected for C, the algorithm will allow the classifier to make more mistakes in the training set. For all the reported experiments, the tested values of C were: 1, 10, 100 or 1000.

Classification results were evaluated using F1-score, which performs a statistical analysis of binary classification by measuring the test's accuracy considering both its precision (the ability of the classifier not to label as positive a sample that is negative) and the recall (the ability of the classifier to find all positive samples). The relative contribution of precision and recall to the F1-score are equal. It reaches its best value at 1 and worst at 0. The formulas used to compute precision, recall and F1-score are shown in Equations (5.1), (5.2) and (5.3), respectively.

These evaluation measures are defined in terms derived from a confusion matrix, a table that allows the user to visualise the performance of a classifier. In the confusion matrix, typically, each column shows observations that were predicted to be in each class, while each row indicates their actual classes. The name "confusion matrix" comes from its main functionality, to identify which pairs of classes the classifier is unable to differentiate. The most used terms computed from the confusion matrix are:

- True positives (tp): number of observations correctly classified on a specific class;
- False positives (fp): number of observations wrongly classified on a specific class;

- True negatives (*tn*): number of observations correctly classified as not being from a specific class;
- False negatives (*fn*): number of observations wrongly classified as not being from a specific class.

$$Precision = \frac{tp}{tp + fp} \tag{5.1}$$

$$Recall = \frac{tp}{tp + fn} \tag{5.2}$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{5.3}$$

When working on tasks which include more than two classes, a weighted average of these evaluations measures can be used. This is done by computing the metric for each label and then finding their average, weighted by the number of instances belonging to each class. It is important to notice that this may cause the resulting F1-score not be between the values obtained for precision and recall. When the average μ and standard deviation σ of the F1-score for multiple classifications are shown, the value of the standard deviation was multiplied by 2. This was done so that the interval between $\mu - 2 \cdot \sigma$ and $\mu + 2 \cdot \sigma$ contains 95% of all obtained results.

By using the same setting for every feature extraction method, these classification results can be used to compare the representation space created by each method. Since the classifier used during the experiments was a linear SVM, it is possible to say that if a specific class presented a good classification result, then that class is (mostly) linearly separable from the other classes. This allows the comparison of multiple feature spaces and the analysis of which classes overlap in each one of them.

5.3 Datasets

The datasets selected for this set of experiments cover two different tasks: action recognition and dynamic scene recognition; while covering two different domains in each task. For action recognition, KTH-Actions (Section 5.3.1.2) provides a controlled setting while Hollywood2 Actions (Section 5.3.1.1) is completely unconstrained resulting in a more realistic and challenging domain. Since IDT was proposed to be used for action recognition, it is interesting to see if the representation learning method will be as efficient as a hand-crafted task-specific feature extraction method. For the KTH-Actions and Hollywood2 Actions datasets, the selection of training and test sets was done using the split proposed in their original papers.

Scene recognition is the task of classifying the place where an action or event occurs (DERPANIS *et al.*, 2012). The ability to distinguish between scenes can help many different tasks by providing priors for the presence of actions, surfaces and objects. The Maryland dataset (Section 5.3.2.1) was built in a way to confuse descriptors that use only spatial information or only temporal information. Therefore, it is a nice benchmark for the quality of both temporal and spatial information encoded into a descriptor. The YUPENN dataset (Section 5.3.2.2) provides a more constrained setting which emphasises short temporal information while restricting samples to videos shots taken with a single static camera. Maryland and YUPENN datasets were previously used in a "leave-one-out" setting, so a training and test split was proposed for this experiments. This split was done by taking advantage of the numbers originally associated with each video, selecting all odd-numbered videos for training and even-numbered videos for testing. This resulted in training and test sets of equal size.

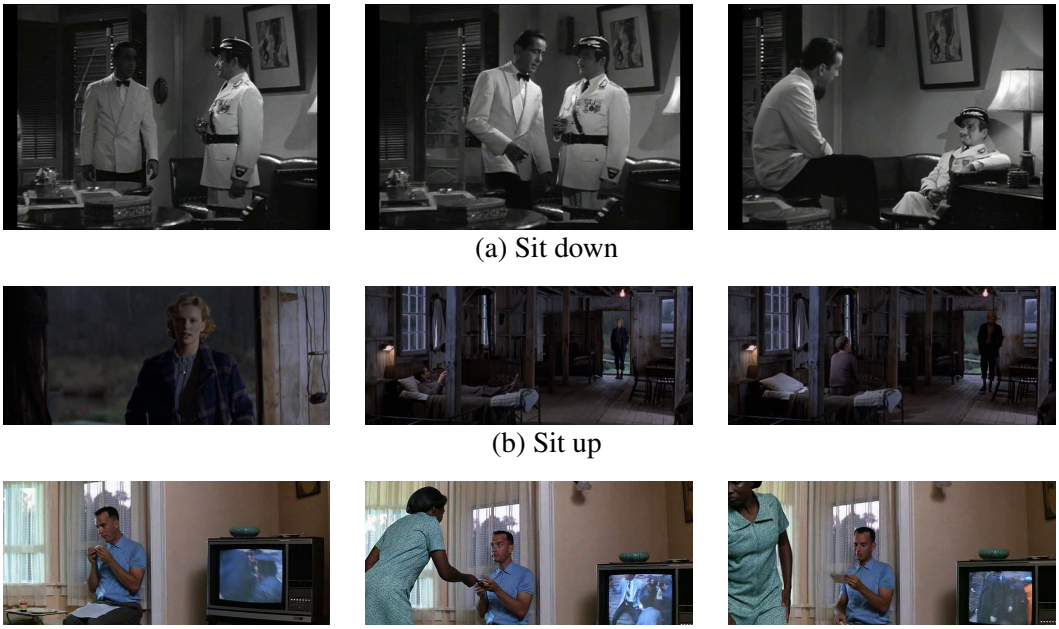
5.3.1 Action recognition

5.3.1.1 Hollywood2 Actions

The Hollywood2 Actions dataset (MARSZAŁEK; LAPTEV; SCHMID, 2009) consists of a set of videos extracted from 69 Hollywood movies divided into 12 classes of human actions by using automatic script-to-video alignment. Two possible training sets are available: an automatically separated training subset with noisy action labels and, based on the first subset, a clean training subset where action labels were manually verified to be correct. The test subset was also manually verified. With almost 20.1 hours of video, this dataset contains about 150 samples per actions. Since the samples were extracted from videos that were not recorded in a controlled environment, this dataset provides realistic and challenging circumstances for action recognition. Example frames of three videos are shown in Figure 26. It is important to notice that this dataset is unbalanced, so the list of classes and the number of instances in each class of this dataset is presented on Table 7.

5.3.1.2 KTH-Action

Considered one of the main benchmarks in the area of video processing (LÄNGKVIST; KARLSSON; LOUTFI, 2014), the KTH-Action dataset (SCHULDT; LAPTEV; CAPUTO, 2004) is one of the most used datasets to evaluate action recognition systems. It consists of videos of 25 people performing six different actions: *walking, jogging, running, boxing, waving* and *clapping hands*. Each of these actions is performed in four different scenarios: outdoors, outdoors with scale variations, outdoors with different clothes and indoors. Each footage was divided into shorter videos, generating a total of 2391 videos with resolution 160×120 pixels. All actions are performed throughout the entire video. Example frames showing three people, each one performing a different action in three of the four possible scenarios are shown in Figure 27.



(a) Sit down

(b) Sit up

(c) Eat

Figure 26 – Example frames from the Hollywood2 Action dataset.

Table 7 – Distribution of samples on each class of the Hollywood2 (Actions) dataset.

	Training set (clean)	Training set (automatic)	Test set (clean)
AnswerPhone	66	59	64
DriveCar	85	90	102
Eat	40	44	33
FightPerson	54	33	70
GetOutCar	51	40	57
HandShake	32	38	45
HugPerson	64	27	66
Kiss	114	125	103
Run	135	187	141
SitDown	104	87	108
SitUp	24	26	37
StandUp	132	133	146
All Samples	823	810	884

5.3.1.3 Sports-1M

Composed of more than one million videos whose classification was made automatically, the Sports-1M dataset (KARPATHY *et al.*, 2014) is divided into 487 different categories, where each category represents a different sport, such as *judo, yoga, skiing* and *wrestling*. Due to the fact that this classification was acquired automatically by looking at each video's textual description, the labels are not guaranteed to be correct.

(a) Boxing

(b) Hand waving

(c) Running

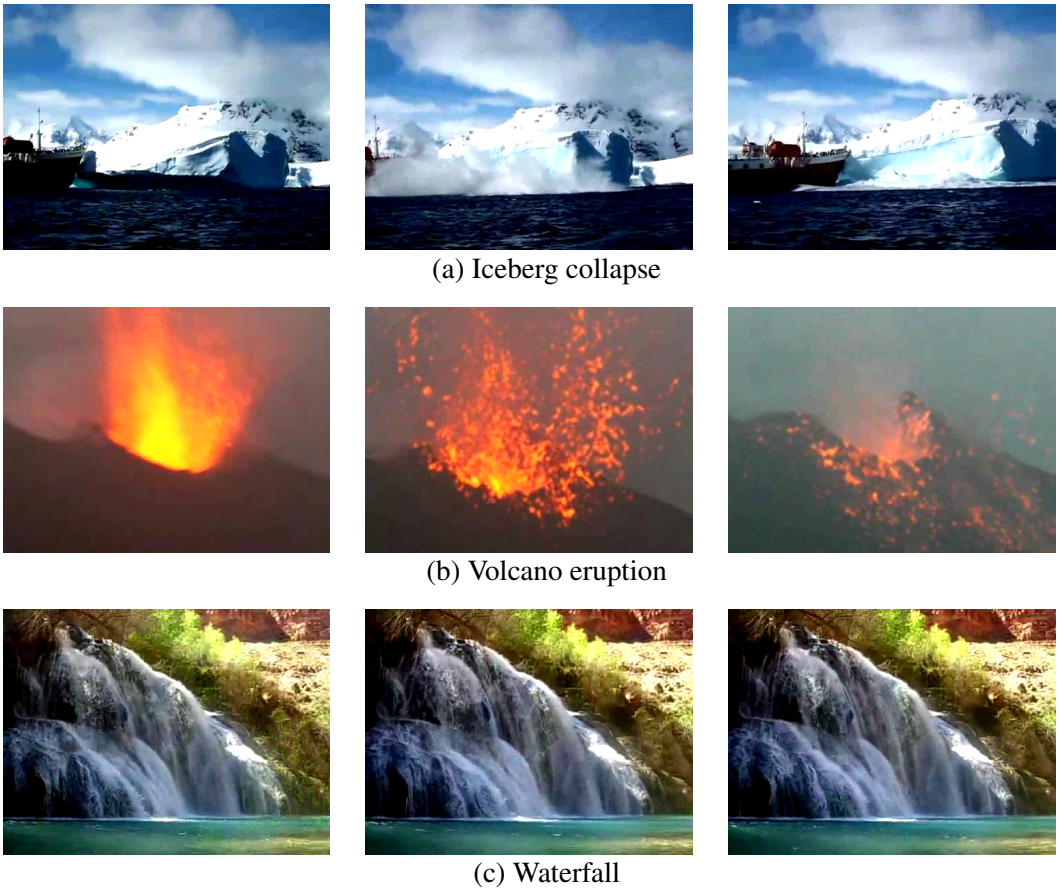
Figure 27 – Example frames from the KTH-Action dataset. Here three different people are shown performing one of the six different actions included in the dataset in three of the four possible scenarios.

5.3.2 Dynamic scene recognition

5.3.2.1 Maryland Dynamic Scenes (UMD)

The UMD dynamic scenes dataset (SHROFF; TURAGA; CHELLAPPA, 2010), also known as Maryland, is an in-the-wild dynamic scenes videos dataset consisting of 13 classes containing 10 videos per class. Each video was obtained from video hosting websites like Youtube, so there was no control over the video capturing process, producing a dataset with large variations in terms of camera resolution, camera dynamics, illumination, frame rate, point of view and scale. These variations resulted in a high intra-class variation throughout this dataset.

The possible dynamic scene categories are: *avalanche, boiling water, chaotic traffic, forest fire, fountain, iceberg collapse, landslide, smooth traffic, tornado, volcanic eruption, waterfall, waves* and *whirlpool*. It is important to emphasise that pure static appearance feature extraction methods used in randomly chosen frames from these videos would result in high confusion rate between certain classes, such as: *avalanche* and *iceberg collapse*; *waterfall* and *fountain*; *landslide* and *volcanic eruption* (SHROFF; TURAGA; CHELLAPPA, 2010). By using only the dynamics of the scenes to extract features, the confusion would occur between the



(a) Iceberg collapse

(b) Volcano eruption

(c) Waterfall

Figure 28 – Example frames from the Maryland Dynamic Scenes dataset that represent three of the thirteen possible scene classes.

following classes: *avalanche, landslide* and *volcanic eruptions; tornado* and *whirlpool; boiling water* and *forest fire* (SHROFF; TURAGA; CHELLAPPA, 2010). Figure 28 illustrates three of the classes contained in this dataset.

5.3.2.2 YUPENN Dynamic Scenes

The YUPENN dynamic scenes video dataset (DERPANIS *et al.*, 2012) was proposed as a dataset that emphasises a scene's specific temporal information over a short time duration due to objects and surfaces rather than camera-induced. This dataset contains 420 colour videos evenly divided into fourteen dynamic scene categories: *beach, city street, elevator, forest fire, fountain, highway, lightning storm, ocean, railway, rushing river, sky-clouds, snowing, waterfall* and *windmill farm*. Since the videos were obtained from various sources, they present significant differences in image resolution, frame rate, scene appearance, scale, illumination conditions and camera viewpoint. All video samples in the dataset are restricted to stationary cameras and do not contain scene cuts. Three of the classes in this dataset are illustrated in Figure 29.

(a) Beach

(b) City street

(c) Forest fire

Figure 29 – Example frames from the YUPENN Dynamic Scenes dataset illustrating three of the fourteen possible scenes: *beach, forest fire* and *city street*.

5.4 Results

5.4.1 KTH-Actions

On the KTH-Actions dataset, the hand-crafted features performed significantly better than the ones extracted by the representation learning technique. The overall results are shown in Table 8. One of the reasons this might have occurred is because IDT was designed specifically for the task of action recognition. Also, the KTH-Actions dataset is composed of videos recorded in a controlled setting, not including complex circumstances such as occlusions and camera motion. Such a constrained environment, together with task-specific knowledge used during the design of the hand-crafted feature extraction method, explain the difference in the performance of the evaluated methods.

Even though it did not achieve results as good as the ones of the IDT-FV method, the features obtained by the C3D method provided a good description of the actions on this dataset considering the network used did not contain any information about this dataset. This indicates that the representations learnt were capable of generalising the learnt knowledge to a different dataset in a similar task. Furthermore, it also shows that training the model in a large dataset is a

Table 8 – Overall results on the KTH-Actions dataset.

	IDT-FV	C3D (Blocks)	C3D (Voting)	C3D (k-Means quantisation)	C3D (Average)	C3D (Statistical moments)
Weighted F1-score	0.9186	0.8386	0.8823	0.7564	0.8750	0.7902

Table 9 – Grid search results for the IDT-FV method using the KTH-Actions dataset. The highlighted parameter was the one selected for the remainder of this experiment.

Kernel	C	F1-score (Average)	F1-score (Standard Deviation)
Linear	1	0.906	0.048
Linear	**10**	**0.932**	**0.024**
Linear	100	0.930	0.023
Linear	1000	0.930	0.023

possible way to compensate for the lack of available data.

Since the videos in KTH-Actions dataset show the same action in almost all frames of a given video, it is possible to classify each of the 16-frame-blocks described by the C3D network. This allows a direct evaluation of the extracted features, without the influence of quantisation/combination methods. Considering that the prediction of each block is relevant given that the action occurs on almost every frame of the video, it is also possible to use a voting scheme where each block casts a vote to define the class of the entire video.

The best results using C3D features for the KTH-Actions dataset were achieved by combining the descriptors through voting and using the average of all block-level representations in a video. We believe that the good results achieved by using the average of all blocks to describe a video were due to the actions in KTH-Actions dataset being repeated throughout the video since all the representations contained relevant information about the video-level label.

5.4.1.1 IDT-FV

After training the SVM on the entire training set and using the parameters selected by the grid search (whose results can be seen in Table 9), the resulting classifier was applied to the test set. IDT-FV features performed well in all classes from the KTH-Actions dataset (Table 10), wherein most of the mistakes were made by confusing observations from the *jogging* and *running* classes. Figure 30 shows the confusion matrix obtained from the final classification.

Table 10 – Per class performance of the IDT-FV features on the test set of the KTH-Actions dataset.

Class	Precision	Recall	F1-score	Number of observations
Boxing	0.9156	0.9860	0.9495	143
Hand clapping	0.9103	0.9167	0.9135	144
Hand waving	1.0000	0.9097	0.9527	144
Jogging	0.8148	0.9167	0.8627	144
Running	0.9421	0.7917	0.8604	144
Walking	0.9533	0.9931	0.9728	144
Average / Total	0.9227	0.9189	0.9186	863

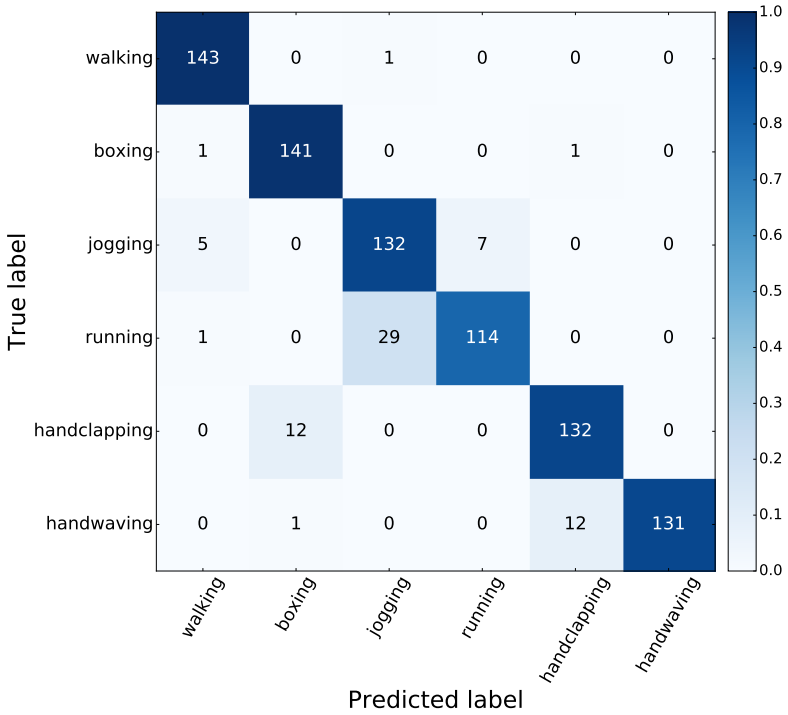


Figure 30 – Confusion matrix obtained by using a SVM classifier on the descriptor extracted by the IDT-FV method on the KTH-Actions dataset. The training and testing split was proposed by Schuldt, Laptev and Caputo (2004). The colours indicate the percentage of observations from each class contained in each cell.

5.4.1.2 C3D

5.4.1.2.1 Classification of 16-frame blocks

Table 11 shows the weighted average and standard deviation of the F1-score resulting from the 5-fold cross-validation performed on the training set during the grid search used to define the C parameter of the SVM. Since all the tested parameters achieved the same results, C was set to the smallest value, that is, $C = 1$.

From the per class results shown on Table 12 and the confusion matrix on Figure 31, it is possible to see that C3D features had difficulties to differentiate between observations from the

Table 11 – Grid search results while classifying the 16-frame blocks descriptors extracted by the C3D method from the KTH-Actions dataset. The highlighted parameter was the one selected for the remainder of this experiment.

Kernel	C	F1-score (Average)	F1-score (Standard Deviation)
Linear	**1**	**0.840**	**0.030**
Linear	10	0.840	0.030
Linear	100	0.840	0.030
Linear	1000	0.840	0.030

Table 12 – Per class performance of the C3D features on the test set of the KTH-Actions dataset when classifying the 16-frame blocks individually.

Class	Precision	Recall	F1-score	Number of observations
Boxing	0.9747	0.9678	0.9712	13123
Hand clapping	0.8660	0.9611	0.9111	12645
Hand waving	0.9709	0.8767	0.9214	15594
Jogging	0.5287	0.5755	0.5511	7454
Running	0.5995	0.7257	0.6565	4826
Walking	0.8156	0.7233	0.7667	12894
Average / Total	0.8451	0.8363	0.8386	66536

walking, *jogging* and *running* classes and, to lesser extent, between observations from the *hand clapping* and *hand waving* classes. This may indicate that the representations learnt by the C3D network trained on the Sports-1M dataset do not describe well the differences in velocity.

It is important to notice that the number of observations in this experiment indicates the total number of 16-frame-blocks extracted for each class. Since the videos in the KTH-Actions dataset have different lengths, the classification task had to deal with some unbalanced class distribution.

5.4.1.2.2 Combination by voting

To perform voting combination to achieve a video-level prediction from 16-frame-blocks predictions, each block needs to be classified individually, so, the parameter selected to train this classifier was the same as the one used in the previous experiment ($C = 1$, as shown in Table 11). After classifying each 16-frame-block, a video's prediction was defined by the class with the highest number of block predictions. This resulted in the per-class performance displayed in Table 13. As expected, the overall performance improved from the previous experiment, since mistakes made on a few blocks from a video do not influence the final chosen class. Most errors committed by the final classification were still between the classes *walking*, *jogging* and *running*, while the number of mistakes made regarding other classes was considerably reduced

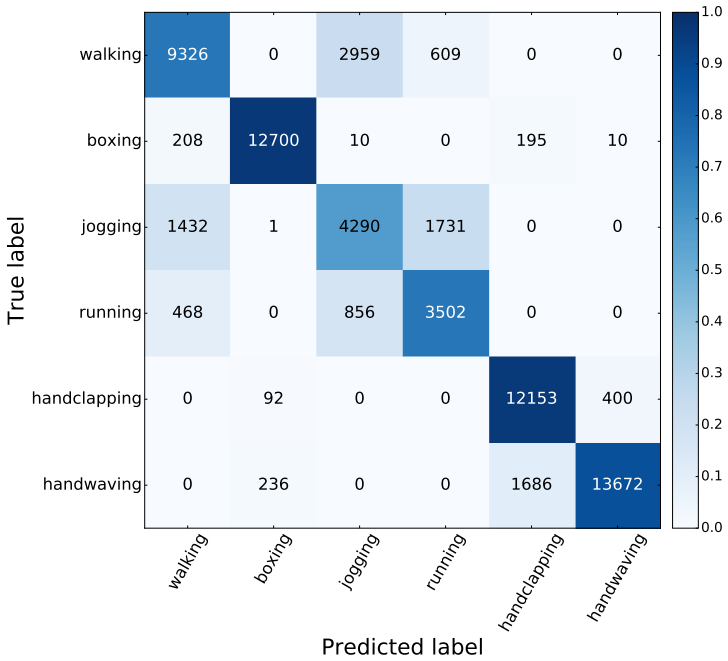


Figure 31 – Confusion matrix obtained by classifying the 16-frame blocks extracted by C3D from the KTH-Actions dataset using the training and testing split proposed by Schuldt, Laptev and Caputo (2004). The colours indicate the percentage of observations from each class contained in each table cell.

Table 13 – Per class performance of the C3D features on the test set of the KTH-Actions dataset when choosing the majority class from the 16-frame blocks classification.

Class	Precision	Recall	F1-score	Number of observations
Boxing	0.9929	0.9790	0.9859	143
Hand clapping	0.9793	0.9861	0.9827	144
Hand waving	0.9859	0.9722	0.9790	144
Jogging	0.7273	0.6667	0.6957	144
Running	0.7987	0.8542	0.8255	144
Walking	0.8121	0.8403	0.8259	144
Average / Total	0.8826	0.8830	0.8823	863

when compared to the evaluation performed on individual blocks (Section 5.4.1.2.1). These conclusions supported by the analysis of the confusion matrix shown on Figure 32.

5.4.1.2.3 Combination by k-means quantisation

Combination by using k-means quantisation ($k = 1000$) to build a dictionary of words from the 16-frame blocks did not perform well. It increased the confusion between classes such as *hand clapping* and *hand waving*, while still mistaking observations from the classes *walking*, *jogging* and *running*. Grid search for hyper-parameter selection and per class results on the test set are shown in Table 14 and 15, respectively; while Figure 33 illustrates the confusion matrix

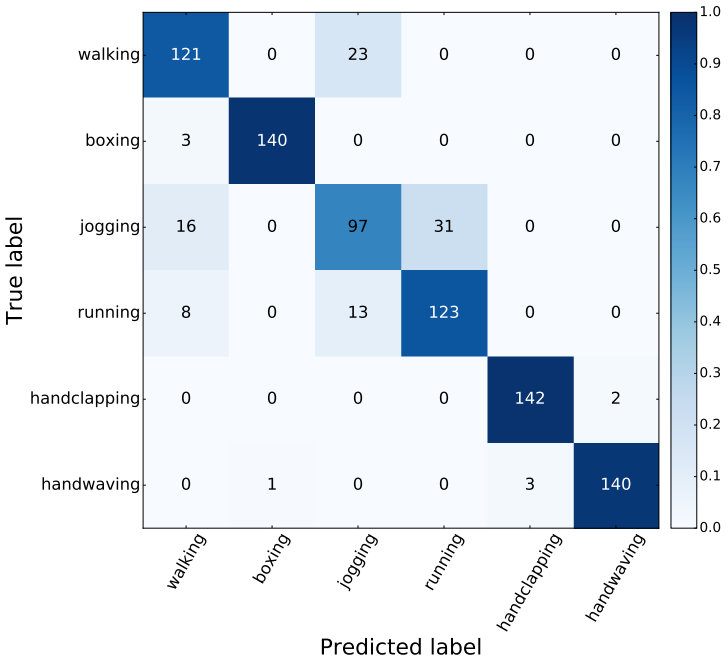


Figure 32 – Confusion matrix obtained by choosing the majority class from the 16-frame blocks classification extracted with the C3D method from the KTH-Actions dataset. Training and test sets as defined by Schuldt, Laptev and Caputo (2004). The colours indicate the percentage of observations from each class contained in each table cell.

Table 14 – Grid search results obtained after applying k-means quantisation on the descriptors extracted by the C3D method from the KTH-Actions dataset. The highlighted parameter was the one selected for the remainder of this experiment.

Kernel	C	F1-score (Average)	F1-score (Standard deviation)
Linear	1	0.214	0.040
Linear	**10**	**0.215**	**0.044**
Linear	100	0.203	0.079
Linear	1000	0.191	0.096

relative to the final classification of the test set.

5.4.1.2.4 Combination by average

For the combination by average approach, grid search returned the same result for all the hyper-parameters tested, so, the smallest parameter value was selected to be used ($C = 1$ – Table 16). Table 17 shows that the *boxing* class was almost perfectly classified. The *hand clapping* and *hand waving classes* also achieved good performances, wherein mistakes were only made by wrongly classifying *hand waving* observations as *hand clapping* (information available in the confusion matrix – Figure 34).

By combining the 16-frame-blocks' descriptors using the average of all blocks in a video, the SVM classifier achieved the second-best results on the KTH-Actions dataset, following

Table 15 – Per class performance of the C3D features on the test set of the KTH-Actions dataset when using 16-frame blocks to create a dictionary using the k-means algorithm. A histogram of this dictionary is then used as the descriptors for the classification.

Class	Precision	Recall	F1-score	Number of observations
Boxing	0.9688	0.8671	0.9151	143
Hand clapping	0.8897	0.8403	0.8643	144
Hand waving	0.8750	0.8750	0.8750	144
Jogging	0.4800	0.5000	0.4898	144
Running	0.6918	0.7639	0.7261	144
Walking	0.6644	0.6736	0.6690	144
Average / Total	0.7614	0.7532	0.7564	863

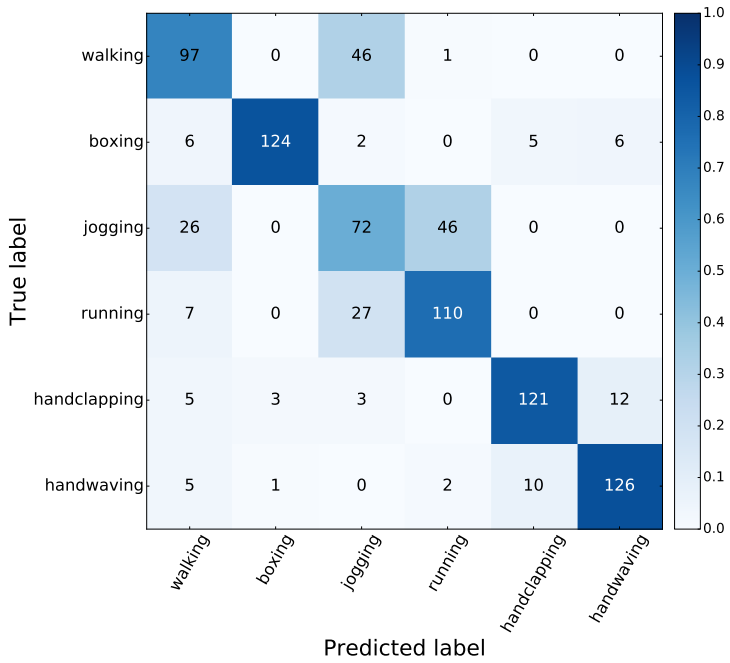


Figure 33 – Confusion matrix obtained by using the k-means quantisation on the 16-frame blocks classification extracted with the C3D method from the KTH-Actions dataset. Training and test sets as defined by Schuldt, Laptev and Caputo (2004). The colours indicate the percentage of observations from each class contained in each table cell.

closely the results achieved using combination by voting. The confusion matrix, shown in Figure 34, indicates that most of the mistakes were made on the following classes: *walking*, *jogging* and *running*; *hand clapping* and *hand waving*. It is important to highlight that in the videos from the KTH-Actions dataset, each action is performed repeatedly throughout the entire video, which might have benefited the combination by average. However, by using this approach, relevant information, like high and low values, might be disregarded.

Table 16 – Grid search results obtained after using the average of the descriptors extracted by the C3D method to describe each video from the KTH-Actions dataset. The highlighted parameter was the one selected for the remainder of this experiment.

Kernel	C	F1-score (Average)	F1-score (Standard Deviation)
Linear	**1**	**0.890**	**0.108**
Linear	10	0.890	0.108
Linear	100	0.890	0.108
Linear	1000	0.890	0.108

Table 17 – Per class performance in the KTH-Actions dataset using C3D features averaged over all 16-frame blocks to obtain a video-level descriptor.

Class	Precision	Recall	F1-score	Number of observations
Boxing	1.0000	0.9930	0.9965	143
Hand clapping	0.8623	1.0000	0.9260	144
Hand waving	1.0000	0.8472	0.9173	144
Jogging	0.7426	0.7014	0.7214	144
Running	0.7530	0.8681	0.8065	144
Walking	0.9308	0.8403	0.8832	144
Average / Total	0.8813	0.8749	0.8750	863

Table 18 – Grid search results obtained by using the concatenation of statistical measures of the representations extracted by the C3D method to describe each video from the KTH-Actions dataset. The highlighted parameter was the one selected for the remainder of this experiment.

Kernel	C	F1-score (Average)	F1-score (Standard deviation)
Linear	**1**	**0.834**	**0.043**
Linear	10	0.834	0.043
Linear	100	0.834	0.043
Linear	1000	0.834	0.043

5.4.1.2.5 Combination by statistical measures

When using the concatenation of multiple statistical measures (average, standard deviation, skewness, kurtosis, minimum and maximum), overall performance was considerably lower than when combining using only the average of representation of all 16-frame blocks in a video. Grid search and evaluation results are shown in Tables 18 and 19, respectively. The confusion matrix in Figure 35 shows that this combination method hindered the ability of the SVM classifier to differentiate between the classes: *walking*, *jogging* and *running*; *hand clapping* and *hand waving*. It also caused the SVM to confuse observations from the *hand clapping* and *hand waving* classes with the *boxing* class.

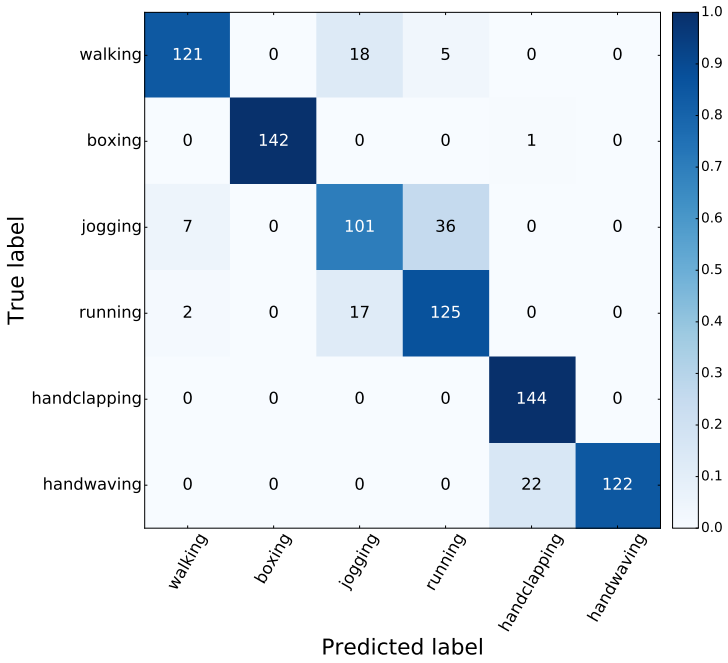


Figure 34 – Confusion matrix obtained by classifying the test set of the KTH-Actions dataset using the average representation extracted by the C3D method over all 16-frame blocks. Training and test sets as defined by Schuldt, Laptev and Caputo (2004). The colours indicate the percentage of observations from each class contained in each table cell.

Table 19 – Per class performance on the KTH-Actions dataset of C3D features when using the concatenation of statistical measures to combine the descriptors of 16-frame blocks for the videos.

Class	Precision	Recall	F1-score	Number of observations
Boxing	0.8551	0.8252	0.8399	143
Hand clapping	0.6915	0.9028	0.7831	144
Hand waving	0.9100	0.6319	0.7459	144
Jogging	0.6842	0.7222	0.7027	144
Running	0.7922	0.8472	0.8188	144
Walking	0.8931	0.8125	0.8509	144
Average / Total	0.8043	0.7903	0.7902	863

5.4.2 Hollywood2 Actions

For Hollywood2 Actions, the overall results were similar to the ones achieved on KTH-Actions, with IDT-FV features performing significantly better than C3D features. Again, we believe that this happened because IDT-FV was designed using task-specific information. The weighted average of the F1-score obtained by each method is shown in Table 20.

Since labels don't include timestamps, it is not possible to guarantee that the actions are performed in every 16-frame-block, so, classifying each 16-frame-block individually and combining block-level predictions by voting is not applicable. Considering the other evaluated combination methods for C3D representations, combination by averaging obtained the best

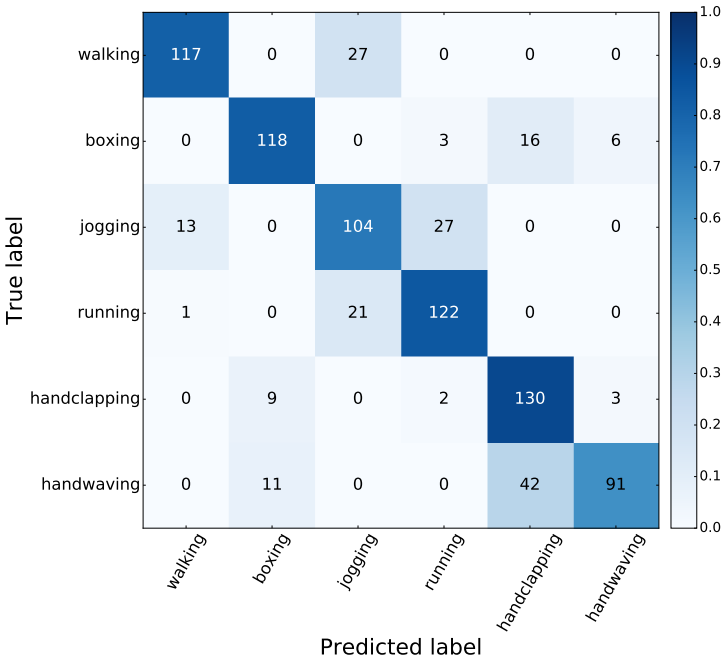


Figure 35 – Confusion matrix obtained by classifying the test set of the KTH-Actions dataset using the concatenation of statistical measures computed using the 16-frame blocks descriptors extracted by the C3D method. Training and test sets as defined by Schuldt, Laptev and Caputo (2004). The colours indicate the percentage of observations from each class contained in each table cell.

Table 20 – Overall results on the Hollywood2 Actions dataset.

	IDT-FV	C3D (Blocks)	C3D (Voting)	C3D (k-Means quantisation)	C3D (Average)	C3D (Statistical Moments)
Weighted F1-score	0.5097	-	-	0.2864	0.4016	0.3251

performance. Even so, its F1-score was around 10% lower than the result achieved by the IDT-FV method. The overall results indicate the high complexity of the problem proposed by the Hollywood2 Actions dataset.

Since the actions occur in videos in a sparse manner, it is possible that the performance of classifiers trained on C3D representations were affected by the combination methods. The best performing combination method was the average, which results in a descriptor which disregards information that appears in a small number of 16-frame-blocks. Given that most videos in the Hollywood2 Actions dataset are substantially longer than the actions in them, relevant information for the action recognition task was probably lost. Another possible reason for the low performance obtained by C3D's representations is that the actions contained in Hollywood2 Actions are too different from the ones on the dataset used to train the network (Sports-1M), which made the learnt knowledge not generalise well to the new dataset/task.

Table 21 – Grid search results for the IDT-FV method using the Hollywood2 Actions dataset. The highlighted parameter was the one selected for the remainder of this experiment.

Kernel	C	F1-score (Average)	F1-score (Standard deviation)
Linear	1	0.107	0.047
Linear	10	0.330	0.044
Linear	100	0.469	0.041
Linear	**1000**	**0.479**	**0.012**

Table 22 – Per class performance of the IDT-FV features on the test set of the Hollywood2 Actions dataset.

Class	Precision	Recall	F1-score	Number of observations
AnswerPhone	0.3095	0.2031	0.2453	64
DriveCar	0.7143	0.8824	0.7895	102
Eat	0.3800	0.5758	0.4578	33
FightPerson	0.4815	0.7429	0.5843	70
GetOutCar	0.5405	0.3509	0.4255	57
HandShake	0.3750	0.0667	0.1132	45
HugPerson	0.2812	0.1364	0.1837	66
Kiss	0.3631	0.5922	0.4502	103
Run	0.7172	0.7376	0.7273	141
SitDown	0.5917	0.6574	0.6228	108
SitUp	0.1429	0.0270	0.0455	37
StandUp	0.6124	0.5411	0.5745	146
Average / Total	0.5168	0.5370	0.5097	972

5.4.2.1 IDT-FV

When using the IDT-FV method to extract features from the Hollywood2 Actions dataset, the grid search selected $C = 1000$ as the best parameter, according to a 5-fold cross-validation in the training set (Table 21). The per class results, shown in Table 22, and confusion matrix (Figure 36) indicate that some classes, such as *kiss* and *answerPhone*, presented a high overlapping rate with other classes, which hampered the performance of the SVM classifier. The *handShake*, *sitUp* and *hugPerson* classes were heavily confused with other classes, causing them to be almost completely misclassified. On the other hand, the classes *driveCar*, *run* and *fightPerson* presented good classification results, probably due to being visually different from other classes in the dataset.

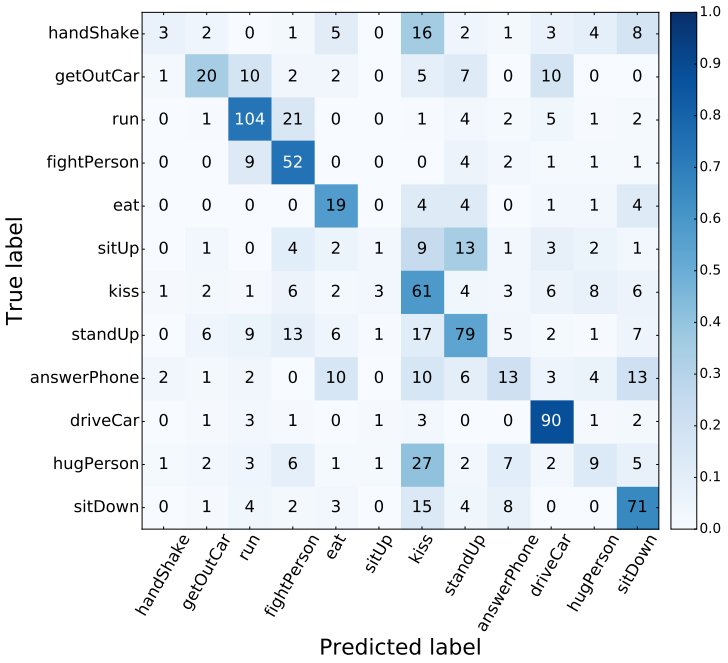


Figure 36 – Confusion matrix obtained by using the IDT-FV feature extraction on the Hollywood2 Actions dataset. The splits used for training and testing were proposed by (MARSZAŁEK; LAPTEV; SCHMID, 2009). The colours indicate the percentage of observations from each class contained in each table cell.

5.4.2.2 C3D

5.4.2.2.1 Combination by k-means quantisation

The results obtained by using k-means quantisation to combine the descriptors of the 16-frame blocks were remarkably lower than the ones obtained by the IDT-FV method and when combining C3D features by averaging. After defining the parameter C of the SVM classifier as 10, according to the results of the grid search presented in Table 23, the classifier was evaluated in the test set (Table 24).

By analysing these results and the confusion matrix shown in Figure 37, it is noticeable that the classifier was unable to correctly classify almost any class, mistaking them with the either *kiss*, *standUp* or *sitDown*. Also, *eat* was completely misclassified. This indicates that the classes in this dataset were not well described in this feature space since most of them are probably concentrated in the same region, making it impossible for a linear classifier to distinguish between them.

5.4.2.2.2 Combination by average

Since all results from the grid search achieved the same average performance when using combination by average (F1-score – Average: 0.353, Standard deviation: 0.071), the smallest tested parameter value was chosen ($C = 1$). In Table 25, the performance of the classification is shown for each class by using the evaluation measures: precision, recall and F1-score. The

Table 23 – Results from the grid search when using k-means quantisation to combine the descriptors extracted by C3D from the Hollywood2 Actions dataset. The highlighted parameter was the one selected for the remainder of this experiment.

Kernel	C	F1-score (Average)	F1-score (Standard deviation)
Linear	1	0.147	0.076
Linear	**10**	**0.185**	**0.073**
Linear	100	0.181	0.054
Linear	1000	0.170	0.051

Table 24 – Evaluation of the performance of k-means quantisation as a combination method for the features extracted by C3D from the Hollywood2 Actions dataset.

Class	Precision	Recall	F1-score	Number of observations
AnswerPhone	0.1364	0.1875	0.1579	64
DriveCar	0.6049	0.4804	0.5355	102
Eat	0.0000	0.0000	0.0000	33
FightPerson	0.2750	0.1571	0.2000	70
GetOutCar	0.0968	0.0526	0.0682	57
HandShake	0.2500	0.0222	0.0408	45
HugPerson	0.2414	0.1061	0.1474	66
Kiss	0.3387	0.4078	0.3700	103
Run	0.4346	0.7305	0.5450	141
SitDown	0.1667	0.2222	0.1905	108
SitUp	0.2500	0.1622	0.1967	37
StandUp	0.2683	0.3014	0.2839	146
Average / Total	0.2932	0.3107	0.2864	972

analysis of this results, together with the information available in the confusion matrix (Figure 38), lead to the conclusion that the classes *answerPhone* and *sitUp* were the most difficult to classify. Also, there was a high number of *fightPerson* observations being misclassified as being *run*. Most mistakes were made by wrongly assigning observations to *standUp*, *hugPerson* and *sitDown*.

5.4.2.2.3 Combination by statistical measures

As what happened with the combination by average, the grid search resulted in the same results all for hyper-parameters evaluated, so C was set to 1 (F1-score – Average: 0.300, Standard deviation: 0.055). The classification performance (seen in Table 26 and through the confusion matrix in Figure 39) was deeply compromised by the use of combination by statistical measures. *answerPhone*, *handShake*, *hugPerson* and *sitUp* were completely misclassified. Nearly all observations were classified as being from classes *standUp*, *sitDown* and *kiss*. We believe this might have happened because of non-linearities inserted into the descriptor by the statistical measures used.

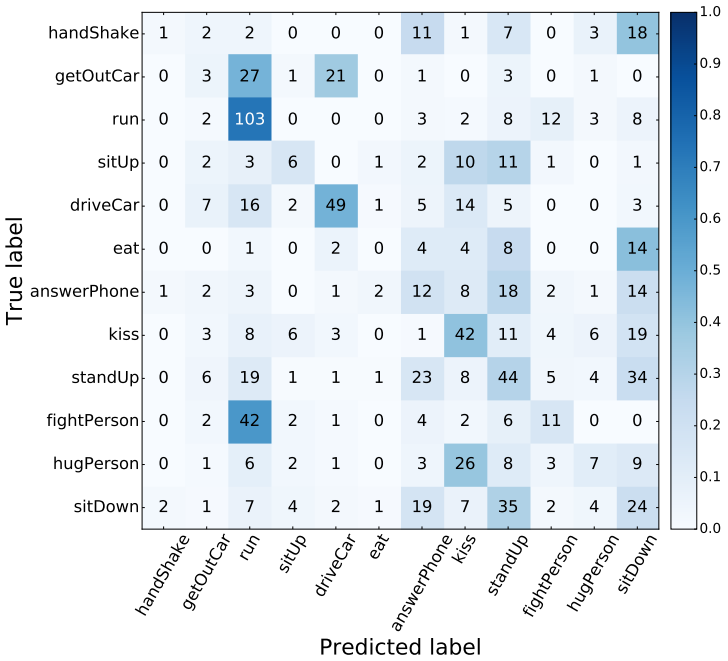


Figure 37 – Classification results on the test set of the Hollywood2 Actions dataset after applying k-means quantisation combination to features extracted by C3D, presented as a confusion matrix.

Table 25 – Per class evaluation of the C3D descriptors combined using the average for videos in the test set of the Hollywood2 Actions dataset. These results were obtained using a linear SVM classifier on a "one-vs-all" setting.

Class	Precision	Recall	F1-score	Number of observations
AnswerPhone	0.2115	0.1719	0.1897	64
DriveCar	0.7453	0.7745	0.7596	102
Eat	0.2703	0.3030	0.2857	33
FightPerson	0.4062	0.3714	0.3881	70
GetOutCar	0.4833	0.5088	0.4957	57
HandShake	0.3226	0.2222	0.2632	45
HugPerson	0.2097	0.1970	0.2031	66
Kiss	0.4334	0.5340	0.4783	103
Run	0.6242	0.6596	0.6414	141
SitDown	0.2080	0.2407	0.2232	108
SitUp	0.4444	0.1081	0.1739	37
StandUp	0.2733	0.2808	0.2770	146
Average / Total	0.4056	0.4084	0.4016	972

5.4.3 Maryland Dynamic Scenes

For the Maryland dataset, which focuses on the task of dynamic scene recognition, the features learnt by C3D achieved results more than 25% better than when using IDT-FV to extract features. The different methods used to combine the 16-frame-block descriptors from the C3D method also showed very different performances, reaching a difference of more than 60% when

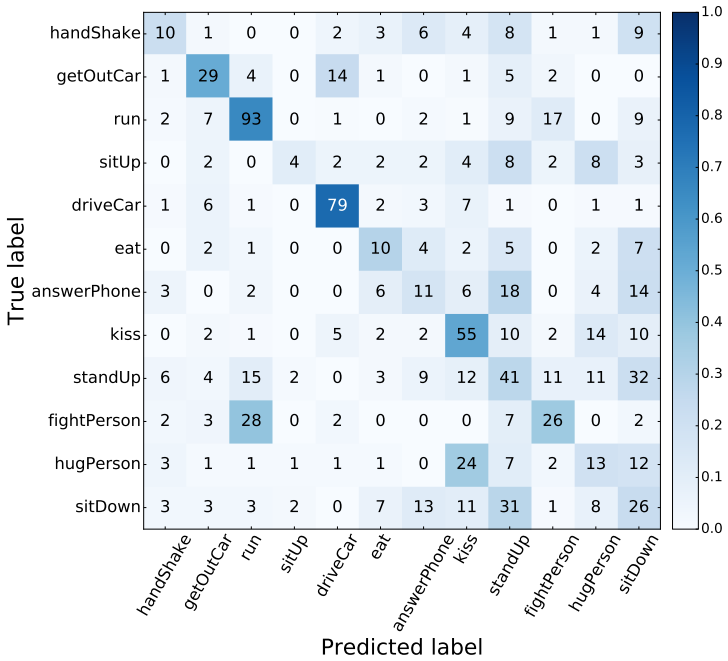


Figure 38 – Results of the classification using SVM classifiers on the descriptors extracted by C3D and combined using average for video-level prediction on the Hollywood2 Actions dataset (shown as a confusion matrix).

Table 26 – Per class performance of C3D features when using concatenation of statistical measures to combine the descriptors of 16-frame blocks for the videos in the Hollywood2 Actions dataset.

Class	Precision	Recall	F1-score	Number of observations
AnswerPhone	0.0000	0.0000	0.0000	64
DriveCar	0.8281	0.5196	0.6386	102
Eat	0.8000	0.1212	0.2105	33
FightPerson	0.4286	0.4714	0.4490	70
GetOutCar	0.2941	0.1754	0.2198	57
HandShake	0.0000	0.0000	0.0000	45
HugPerson	0.0000	0.0000	0.0000	66
Kiss	0.3717	0.4078	0.3889	103
Run	0.5772	0.6099	0.5931	141
SitDown	0.1818	0.2407	0.2072	108
SitUp	0.0000	0.0000	0.0000	37
StandUp	0.2567	0.6575	0.3692	146
Average / Total	0.3440	0.3601	0.3251	972

comparing the best and worst combination methods. The overall results are presented in Table 27.

We believe that the low performance obtained when using features extracted with IDT-FV was probably due to the high intra-class variation on the videos in this dataset. Also, since the IDT-FV method was initially proposed for action recognition, it is expected that, when it is applied to different tasks like dynamic scene recognition, the results will not be as good as

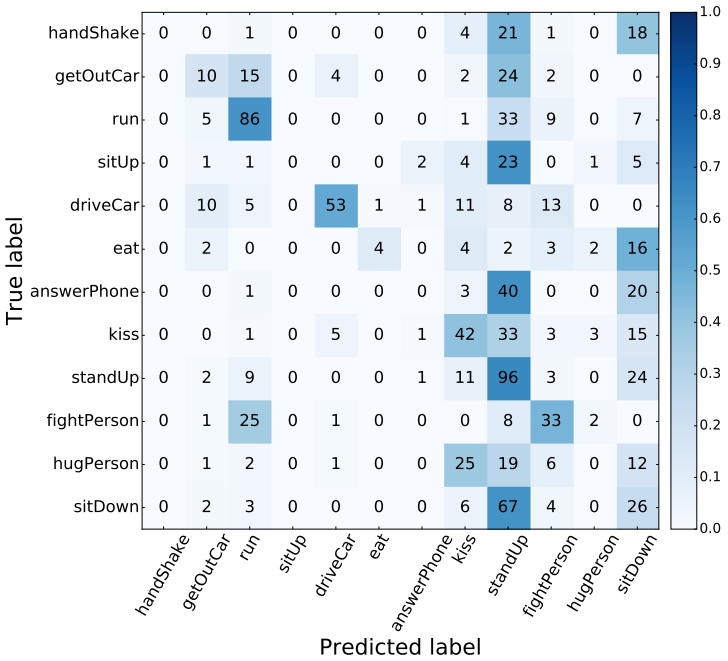


Figure 39 – Confusion matrix of the prediction made by linear SVMs classifying the videos from the Hollywood2 Actions dataset using descriptors extracted by C3D and combined using the concatenation of multiple statistical measures.

Table 27 – Overall results on the Maryland dataset.

	IDT-FV	C3D (Blocks)	C3D (Voting)	C3D (k-Means quantisation)	C3D (Average)	C3D (Statistical Moments)
Weighted F1-score	0.4750	0.7436	0.7768	0.2614	0.7448	0.1382

when used for the task it was designed for. The performance might be improved if different local feature extraction methods, like HOG, HOF or SIFT were used instead of MBH, or if multiple local feature extraction methods were used at the same time (and combined by concatenation).

Regarding C3D representations, the efficiency was satisfactory when classifying 16-frame-blocks individually and when the combination of the descriptors was done using averaging or voting. For descriptors created using k-means quantisation and combination by statistical measures, the classifier performed poorly, which indicates the importance of selecting a suitable combination method. C3D representations could probably be further improved by fine-tuning the network on the new dataset, that is, by adjusting the parameters of the pre-trained network through back-propagation on the new dataset, and by increasing the resolution of the inputs used in the network. Since dynamic scene recognition and sports recognition are very different tasks, the good performance achieved by the C3D representations show that the learnt features are capable of generalising to a completely different task, even without performing any fine-tuning.

Table 28 – Grid search results for IDT-FV representations extracted from the Maryland dataset. The highlighted parameter was the one selected to be used during the rest of this experiment.

Kernel	C	F1-score (Average)	F1-score (Standard deviation)
Linear	1	0.178	0.107
Linear	10	0.224	0.216
Linear	100	0.325	0.207
Linear	**1000**	**0.337**	**0.189**

Table 29 – Per class performance of IDT-FV representations on the test set of the Maryland dataset.

Class (Abbreviation)	Precision	Recall	F1-score	Number of observations
Avalanche (Aval)	0.0000	0.0000	0.0000	5
Boiling water (Boil)	1.0000	0.6000	0.7500	5
Chaotic traffic (cTraffic)	0.6667	0.8000	0.7273	5
Forest fire (fFire)	0.2500	0.2000	0.2222	5
Fountain	0.2500	0.2000	0.2222	5
Iceberg collapse (icebergc)	0.3333	0.4000	0.3636	5
Landslide	0.3333	0.2000	0.2500	5
Smooth traffic (smTraffic)	0.6000	0.6000	0.6000	5
Tornado	0.5000	0.6000	0.5455	5
Volcano eruption (vEruption)	0.5000	0.8000	0.6154	5
Waterfall	1.0000	0.6000	0.7500	5
Waves	0.5714	0.8000	0.6667	5
Whirlpool	0.3750	0.6000	0.4615	5
Average / Total	0.4908	0.4923	0.4750	65

5.4.3.1 IDT-FV

After using grid search to define the best hyper-parameters for the SVM model (Table 28), the evaluation results on the test set are shown in Table 29 for each class of the Maryland dataset. These results and the confusion matrix displayed in Figure 40 were obtained by using the descriptors extracted using IDT-FV. It is possible to notice that IDT-FV descriptors had difficulties describing the *Forest fire*, *Fountain*, *Landslide*, *Iceberg collapse* and *Avalanche*, wherein the *Avalanche* class was completely misclassified.

5.4.3.2 C3D

5.4.3.2.1 Classification of 16-frame blocks

After performing grid search to find the best value for the C hyper-parameter of the SVM classifier (which resulted in using $C = 1$ since all values of C achieved the same F1-score – 0.919 ± 0.153), the resulting classifier was evaluated on a previously unseen test set. Considering

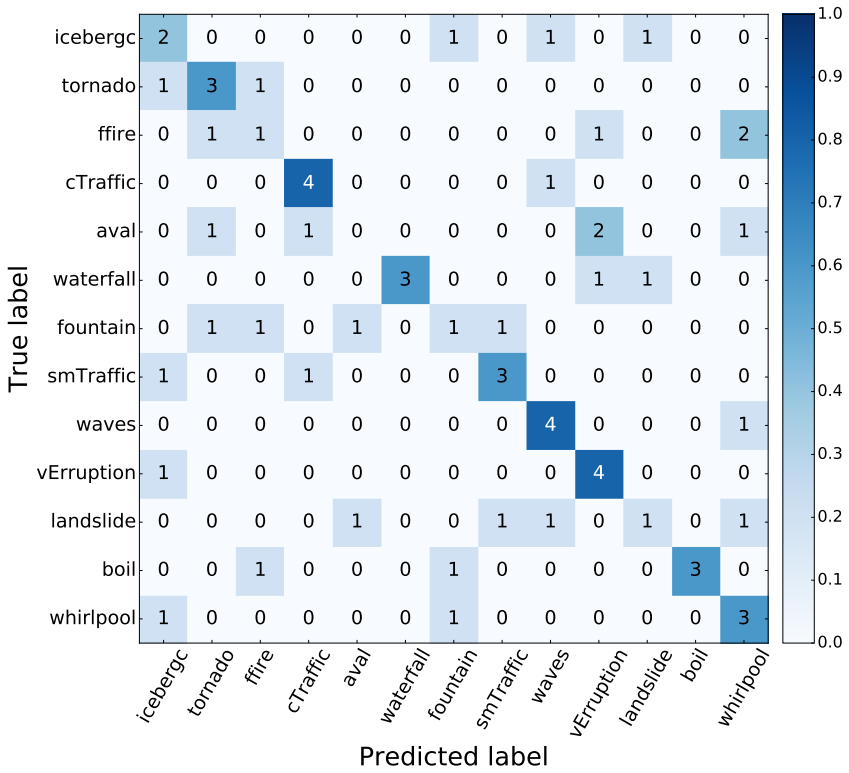


Figure 40 – Confusion matrix obtained by using IDT-FV representations from the Maryland dataset. The colours indicate the percentage of observations from each class contained in each table cell.

that the number of observations in this experiment is the number of 16-frame-blocks extracted from videos in the Maryland dataset and by analysing the results for each class presented in Table 30 and the confusion matrix in Figure 41, one can conclude that C3D was able to learn a feature space that provides good intra-class separability.

In this feature space, the biggest overlaps occur between the classes: *Iceberg collapse* and *Waves*; *Fountain* and *Whirlpool*; *Smooth traffic* and *Chaotic traffic*; *Whirlpool* and *Landslide*. Some of these classes are clearly similar, for example, *Smooth traffic* and *Chaotic traffic* observations both portray city streets containing cars and motorcycles, while most *Iceberg collapse* observations also include *Waves*. The remaining classes are almost completely linearly separable from the others.

5.4.3.2.2 Combination by voting

Since combination by voting demands every 16-frame-blocks to be classified and the combination is only done later, the grid search is unnecessary once the best parameter for the classification of 16-frame-blocks was already selected. Table 31 shows that by combining the predictions using a voting approach the mistakes made during the previous classification were reduced and some classes reached a perfect precision and/or a perfect recall. By also analysing the confusing matrix in Figure 42, we can see that the most misclassified observations belonged to

Table 30 – Per class performance of C3D representations from the Maryland dataset's test set when classifying each 16-frame blocks independently.

Class (Abbreviation)	Precision	Recall	F1-score	Number of observations
Avalanche (Aval)	0.6949	0.8801	0.7766	2101
Boiling water (Boil)	0.8567	0.8362	0.8463	1337
Chaotic traffic (cTraffic)	0.7809	0.9887	0.8726	7370
Forest fire (fFire)	0.9745	0.7957	0.8760	3553
Fountain	0.8092	0.4976	0.6162	2046
Iceberg collapse (icebergc)	0.4728	0.2617	0.3369	1960
Landslide	0.3728	0.6790	0.4813	1726
Smooth traffic (smTraffic)	0.9897	0.5942	0.7347	5277
Tornado	0.8450	0.9866	0.9103	5381
Volcano eruption (vEruption)	0.7741	0.7163	0.7441	2665
Waterfall	0.6550	0.9419	0.7727	1566
Waves	0.5487	0.9923	0.7067	1697
Whirlpool	0.7665	0.4376	0.5571	5107
Average / Total	0.7838	0.7533	0.7436	41786

Table 31 – Per class performance of C3D descriptors on the test set of the Maryland dataset when choosing the majority predicted class from the 16-frame blocks in each video.

Class (Abbreviation)	Precision	Recall	F1-score	Number of observations
Avalanche (Aval)	0.7143	1.0000	0.8333	5
Boiling water (Boil)	1.0000	0.8000	0.8889	5
Chaotic traffic (cTraffic)	0.7143	1.0000	0.8333	5
Forest fire (fFire)	1.0000	0.8000	0.8889	5
Fountain	1.0000	0.6000	0.7500	5
Iceberg collapse (icebergc)	0.5000	0.4000	0.4444	5
Landslide	0.7500	0.6000	0.6667	5
Smooth traffic (smTraffic)	1.0000	0.6000	0.7500	5
Tornado	0.8333	1.0000	0.9091	5
Volcano eruption (vEruption)	0.8000	0.8000	0.8000	5
Waterfall	0.7143	1.0000	0.8333	5
Waves	0.7143	1.0000	0.8333	5
Whirlpool	0.7500	0.6000	0.6667	5
Average / Total	0.8070	0.7846	0.7768	65

Iceberg collapse, which were confused with observations from similar classes: *Waves*, *Waterfall*, *Landslide* and Volcano eruption.

The only other classes that shared more than one mistake was the Chaotic traffic and Smooth traffic classes, which are also clearly related. This indicates that the learnt features are capable of capturing relevant information about each class and that these representations describe each observation in a similar way to what a human would, placing similar classes close together.

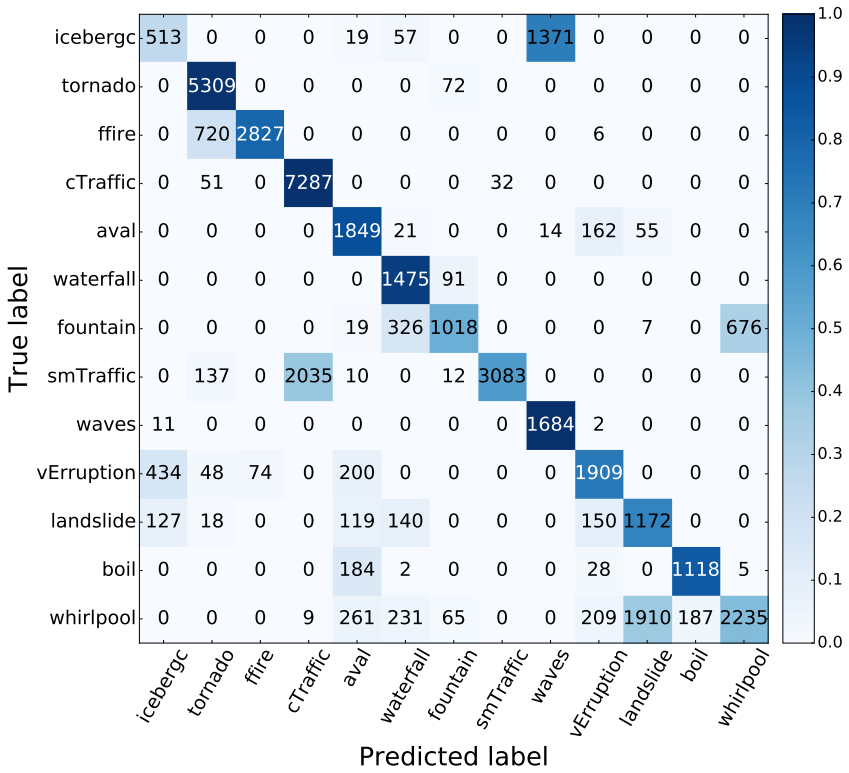


Figure 41 – Confusion matrix resulting of the classification of the 16-frame blocks extracted by C3D from the Maryland dataset. The colours indicate the percentage of observations from each class contained in each table cell.

Table 32 – Grid search for the *C* parameter of a linear SVM classifier using a k-means quantisation to combine the descriptors of multiple 16-frame blocks to create representations for videos of variable length from the Maryland dataset.

Kernel	C	F1-score (Average)	F1-score (Standard deviation)
Linear	**1**	**0.011**	**0.000**
Linear	10	0.011	0.000
Linear	100	0.011	0.000
Linear	1000	0.011	0.000

5.4.3.2.3 Combination by k-means quantisation

When the k-means quantisation was used to combine the descriptors of multiple 16-frame blocks from the Maryland dataset, the SVM classifier (with parameters defined by a grid search whose results are shown in Table 32) had difficulties to differentiate between classes and most observations were predicted as being from the *Volcano eruption* class. This fact can be seen in the results presented in Table 33 and illustrated by the confusion matrix in Figure 43. The only classes that were well represented by this feature space were *Chaotic traffic* and *Waves*, which still presented a overlap with the *Smooth traffic* and *Iceberg collapse*, respectively.

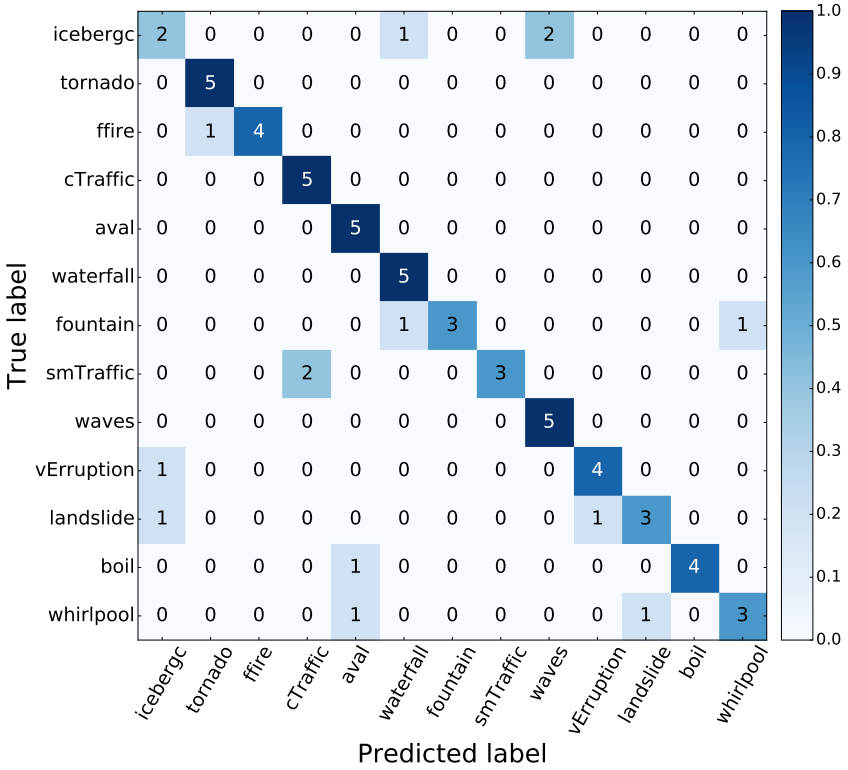


Figure 42 – Confusion matrix of the classification achieved by choosing the majority class from the predictions made for each 16-frame block extracted by C3D from the videos in the Maryland dataset.

5.4.3.2.4 Combination by average

As defined earlier, when multiple different parameters achieve the same average result during the grid search stage, the smallest one of these is the one selected for the experiment, that is, $C = 1$ (F1-score – Average: 0.627, Standard deviation: 0.232). By using the average descriptors to obtain a video-level representation, the second best overall result was achieved (Table 34). When analysing the results for each class and the confusion matrix illustrated by Figure 44, it is possible to see that *Avalanche* was perfectly classified and some other classes (*Boiling water*, *Chaotic traffic*, *Tornado*, *Forest fire* and *Smooth traffic*) achieved either perfect precision or perfect recall.

Once again, the most common mistake was between *Iceberg collapse* and *Waves*. Another noticeable misclassification that happened when using combination by averaging was between the *Boiling water* and *Whirlpool* classes. Since this confusion did not happen when using combination by voting, the use of the average representation probably disregarded important information that made the classifier capable of distinguishing between these classes on the previous experiment.

Table 33 – Efficiency of SVM classifiers for each class in the test set of the Maryland dataset while using k-means quantisation to create the video-level representations based on descriptors extracted from 16-frame blocks by C3D.

Class (Abbreviation)	Precision	Recall	F1-score	Number of observations
Avalanche (Aval)	1.0000	0.4000	0.5714	5
Boiling water (Boil)	0.0000	0.0000	0.0000	5
Chaotic traffic (cTraffic)	0.7143	1.0000	0.8333	5
Forest fire (fFire)	0.0000	0.0000	0.0000	5
Fountain	0.5000	0.2000	0.2857	5
Iceberg collapse (icebergc)	1.0000	0.2000	0.3333	5
Landslide	0.0000	0.0000	0.0000	5
Smooth traffic (smTraffic)	1.0000	0.2000	0.3333	5
Tornado	0.0000	0.0000	0.0000	5
Volcano eruption (vEruption)	0.1163	1.0000	0.2083	5
Waterfall	0.0000	0.0000	0.0000	5
Waves	0.7143	1.0000	0.8333	5
Whirlpool	0.0000	0.0000	0.0000	5
Average / Total	0.3881	0.3077	0.2614	65

Table 34 – Per class performance of C3D features when the average descriptor was used as video-level representation for videos in the Maryland dataset.

Class (Abbreviation)	Precision	Recall	F1-score	Number of observations
Avalanche (Aval)	1.0000	1.0000	1.0000	5
Boiling water (Boil)	0.7143	1.0000	0.8333	5
Chaotic traffic (cTraffic)	0.8333	1.0000	0.9091	5
Forest fire (fFire)	1.0000	0.8000	0.8889	5
Fountain	0.7500	0.6000	0.6667	5
Iceberg collapse (icebergc)	0.5000	0.4000	0.40000	5
Landslide	0.7500	0.6000	0.6667	5
Smooth traffic (smTraffic)	1.0000	0.6000	0.7500	5
Tornado	0.6250	1.0000	0.7692	5
Volcano eruption (vEruption)	0.8000	0.8000	0.8000	5
Waterfall	0.6667	0.8000	0.7273	5
Waves	0.6667	0.8000	0.7273	5
Whirlpool	0.6667	0.4000	0.5000	5
Average / Total	0.7671	0.7538	0.7448	65

5.4.3.2.5 Combination by statistical measures

The classification results for the Maryland dataset were greatly compromised when the combination of the 16-frame-block descriptors extracted with C3D was done using the concatenation of statistical measures. Firstly, the best parameter C for the SVM was selected using a grid search whose results are presented in Table 35. Then, the resulting classifier was

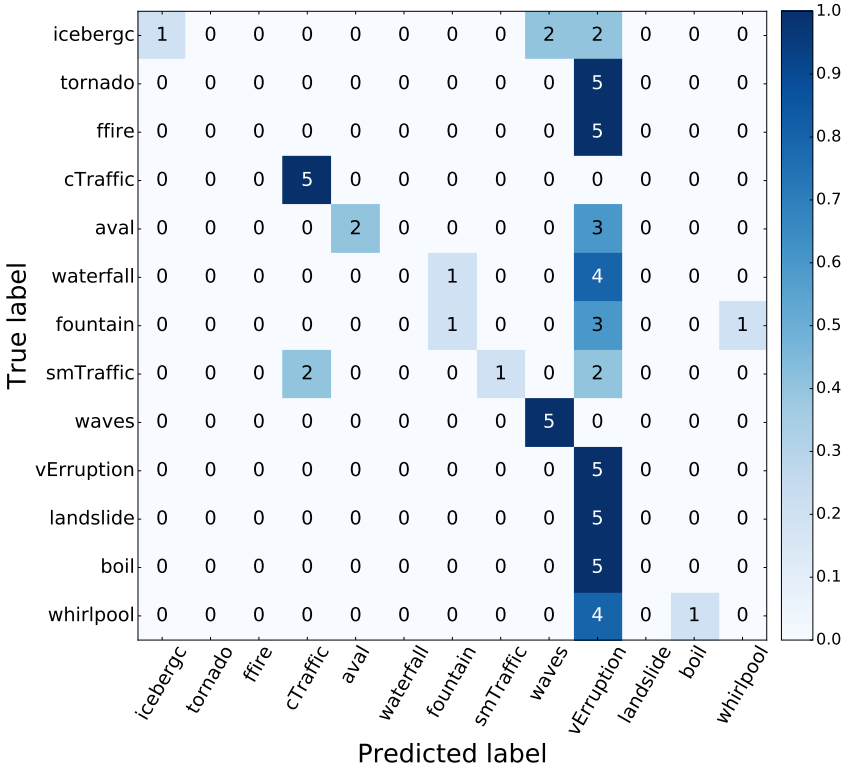


Figure 43 – Confusion matrix representing the classification predicted by a linear SVM classifier using descriptors created with k-means quantisation using representations for 16-frame blocks in the Maryland dataset. The colours indicate the percentage of observations from each class contained in each table cell.

Table 35 – Grid search results that defined the *C* parameter of the classifier. Computed by performing a 5-fold cross-validation in the training set of the Maryland dataset described by C3D and then combined the concatenation of multiple statistical measures. The chosen parameter is highlighted.

Kernel	C	F1-score (Average)	F1-score (Standard deviation)
Linear	**1**	**0.150**	**0.180**
Linear	10	0.150	0.180
Linear	100	0.150	0.180
Linear	1000	0.150	0.180

applied to the test set and the performance for each class is reported in Table 36 and in Figure 45 in a confusion matrix. The use of combination by statistical measures caused the classifier to mistake almost every class as being *Smooth traffic* and, even so, some *Smooth traffic* observations were wrongly classified as being from other classes.

5.4.4 YUPENN Dynamic Scenes

The YUPENN dataset, as the Maryland dataset, is a benchmark for the classification of dynamic scenes. The greatest difference between these datasets is that YUPENN provides a more

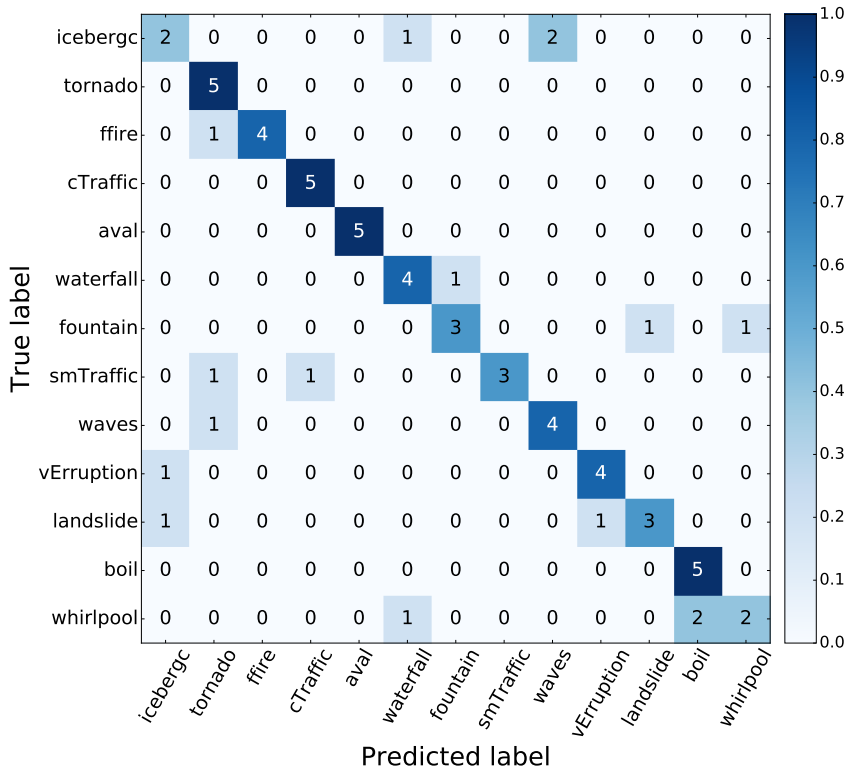


Figure 44 – Confusion matrix obtained by classifying the test set of the Maryland dataset using when using averaging to combine the 16-frame blocks descriptors extracted by C3D. The colours indicate the percentage of observations from each class contained in each table cell.

controlled setting, without camera motion. Under these circumstances, most of the representations analysed in this experiment achieved good performances, whereas the best performing descriptor was created by combining the 16-frame-blocks extracted with C3D using voting. This method was closely followed by the classification of each 16-frame-block individually and by combining these descriptors using the average. Overall results from this experiment are shown in Table 37.

Once again, IDT-FV feature vectors performed worst than the ones extracted by C3D. We believe this is due to IDT-FV method being designed for a different task, though its performance might be improved by using a different local descriptor (in this, we use MBH) or a concatenation of multiple local descriptors.

When comparing the efficiency of C3D based descriptors, it is noticeable that this method was capable of creating a good feature space which provided high intra-class separability. This quality was drastically reduced when the combination of the 16-frame-block descriptors was done using statistical measures, which might have happened because the used measures created a less linearly separable feature space. Combination by k-means quantisation also hindered the classifier's ability to differentiate between classes, especially when compared to 16-frame-blocks individual classification. Combination by averaging and by voting achieved performance similar to when no combination was used. This performance might be further improved by fine-tuning the pre-trained network on the new dataset or by creating a new network and training it directly

Table 36 – Per class performance of C3D representations in the Maryland dataset, combined by concatenating multiple statistical measures (average, standard deviation, kurtosis, skewness, maximum and minimum) and classified with a linear SVM.

Class (Abbreviation)	Precision	Recall	F1-score	Number of observations
Avalanche (Aval)	0.4000	0.4000	0.4000	5
Boiling water (Boil)	0.5000	0.2000	0.2857	5
Chaotic traffic (cTraffic)	1.0000	0.2000	0.3333	5
Forest fire (fFire)	0.0000	0.0000	0.0000	5
Fountain	0.0000	0.0000	0.0000	5
Iceberg collapse (icebergc)	0.1667	0.2000	0.1818	5
Landslide	0.0000	0.0000	0.0000	5
Smooth traffic (smTraffic)	0.0455	0.2000	0.0741	5
Tornado	0.2500	0.2000	0.2222	5
Volcano eruption (vEruption)	0.1429	0.2000	0.1667	5
Waterfall	0.0000	0.0000	0.0000	5
Waves	0.0000	0.0000	0.0000	5
Whirlpool	0.1000	0.2000	0.1333	5
Average / Total	0.2004	0.1385	0.1382	65

Table 37 – Overall results on the YUPENN dataset.

	IDT-FV	C3D (Blocks)	C3D (Voting)	C3D (k-Means quantisation)	C3D (Average)	C3D (Statistical Moments)
Weighted F1-score	0.8377	0.9596	0.9615	0.8891	0.9571	0.6174

with this dataset (if enough data is available).

5.4.4.1 IDT-FV

The results of the grid search performed to define the value of the penalty factor C using the features extracted by IDT-FV from videos in the YUPENN dataset are shown in Table 38. Results on the test set are presented in Table 39 and in the confusion matrix (Figure 46) and indicate that the SVM classifier had a tendency of misclassifying observations as being from the *Sky clouds* class. It is also notable that the classifier had difficulty to correctly classify observations from the *Waterfall* class, which were broadly confused with being from the *Fountain* class. This indicates that these classes were not well described by the feature space or that the descriptor was not able to make these classes linearly separable from each other.

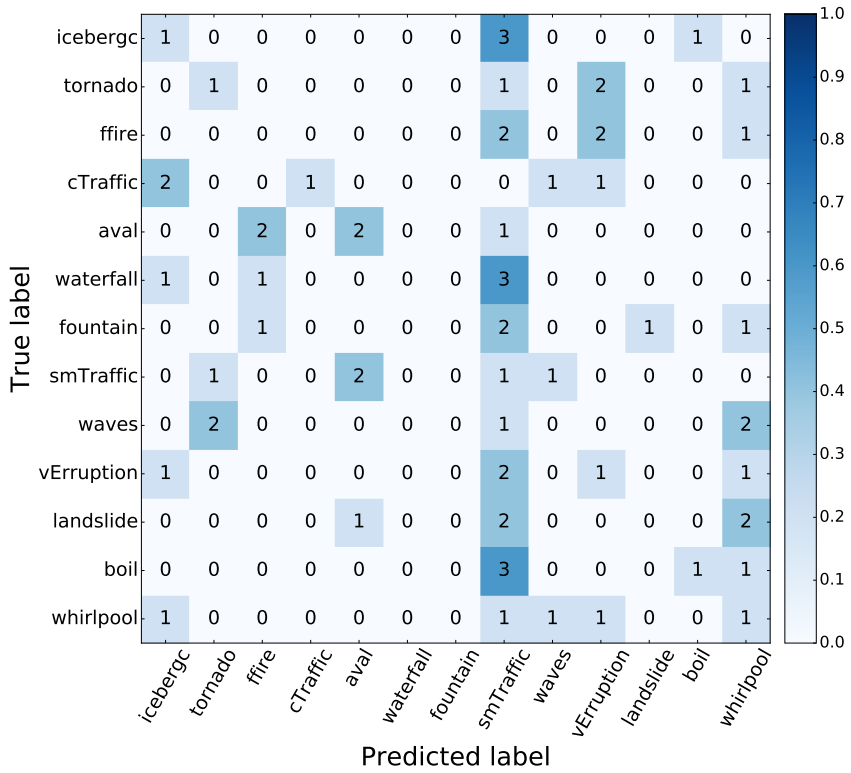


Figure 45 – Confusion matrix that resulted from the classification made using a linear SVM on the videos from the Maryland dataset. This videos were described by representations extracted using C3D and combined by the concatenation of multiple statistical measures.

Table 38 – Grid search results for the IDT-FV representations on the YUPENN dataset. The highlighted parameter was the one selected to be used during the rest of this experiment.

Kernel	C	F1-score (Average)	F1-score (Standard deviation)
Linear	1	0.486	0.105
Linear	10	0.772	0.080
Linear	**100**	**0.757**	**0.050**
Linear	1000	0.757	0.050

5.4.4.2 C3D

5.4.4.2.1 Classification of 16-frame blocks

Parameters used to train the SVM classifier were defined using a grid search, whose results are shown in Table 40. This classifier's results achieved on the test set are presented in Table 41 and in the confusion matrix (Figure 47). They indicate that the feature space provided a good separability between all classes. *Fountain*, *Waterfall* and *Lightning storm* were the most misclassified classes in this experiment. These results show that slight overlaps between these and other classes occur in the feature space, which may be reduced by introducing information from the dataset into the network by fine-tuning.

Table 39 – Per class performance of SVM classifiers on the IDT-FV features on the test set of the YUPENN dataset.

Class (Abbreviation)	Precision	Recall	F1-score	Number of observations
Beach	1.0000	0.8667	0.9286	15
Elevator	1.0000	1.0000	1.0000	15
Forest fire (ffire)	0.7692	0.6667	0.7143	15
Fountain	0.7000	0.9333	0.8000	15
Highway	0.7500	1.0000	0.8571	15
Lightning storm (lstorm)	0.9286	0.8667	0.8966	15
Ocean	1.0000	0.9333	0.9655	15
Railway	1.0000	0.8667	0.9286	15
Rushing river (rriver)	0.7500	1.0000	0.8571	15
Sky clouds (sclouds)	0.5500	0.7333	0.6286	15
Snowing	0.7059	0.8000	0.7500	15
Street	1.0000	0.6667	0.8000	15
Waterfall	1.0000	0.4667	0.6364	15
Windmill farm (wmfarm)	1.0000	0.9333	0.9655	15
Average / Total	0.8681	0.8381	0.8377	210

Table 40 – Average and standard deviation of the F1-score obtained during the grid search computed using each 16-frame blocks representation extracted by C3D for videos in the YUPENN dataset.

Kernel	C	F1-score (Average)	F1-score (Standard deviation)
Linear	**1**	**0.954**	**0.023**
Linear	10	0.954	0.023
Linear	100	0.954	0.023
Linear	1000	0.954	0.023

5.4.4.2.2 Combination by voting

When using a voting scheme to combine the descriptors of multiple 16-frame-blocks as extracted by the C3D method, the SVM classifier presented the same performance (F1-score – Average: 0.954, Standard deviation: 0.023) for all tested parameters, which lead to selecting the smallest parameter ($C = 1$) in the grid search. The resulting classifier performed well when applied to a never before seen test set, achieving perfect prediction of multiple classes (7 out of 15 classes were perfectly classified). Most of the mistakes made by the classifier are understandable, since they were made between clearly related classes, such as *Fountain* and *Waterfall* or *Sky clouds* and *Lightning storm*. These results can be seen in Table 42 and in Figure 48.

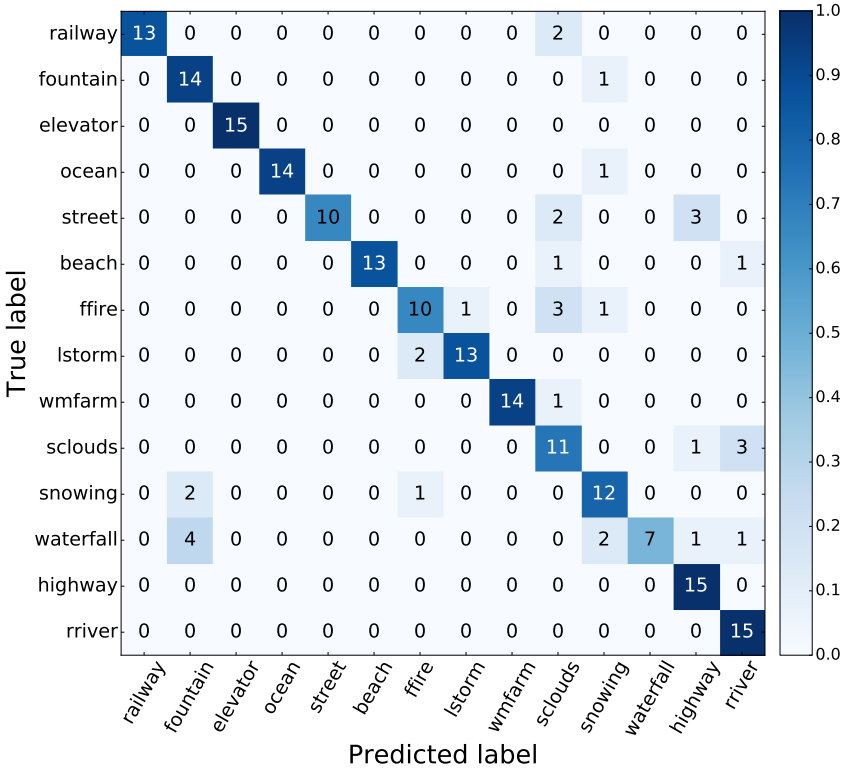


Figure 46 – Confusion matrix obtained by classifying IDT-FV representations extracted from the test set of the YUPENN dataset using a linear SVM. Colours show how many observations from each class are contained in each matrix cell.

5.4.4.2.3 Combination by k-means quantisation

Table 43 shows the results obtained during the grid search to define the parameter C of the linear SVM. Since multiple hyper-parameter resulted in the same average F1-score, the smallest value between them is the one selected to be used during the experiment. Results on test set can be seen in Table 44, as well as in Figure 49. These results show that combining the descriptors (extracted by C3D) for all 16-frame-blocks in a video from the YUPENN dataset using k-means quantisation caused the linear classifier to misclassify observations from the *Fountain class* with multiple other classes. This is probably due to an increased overlap between these classes when compared to the classification of each 16-frame-block individually. The performance for other classes was also hindered (in a less impactful way) when this combination method was used.

5.4.4.2.4 Combination by average

Grid search results are shown in Table 45, the highlighted hyper-parameter was used to train a classifier that was used to predict the class of observations in the test set. The per-class evaluation measures on this test set can be seen in Table 46 and on the confusion matrix in Figure 50. This setting achieved the best results (that did not depend on classifying each 16-frame-block) on the YUPENN dataset. Most mistakes were made by wrongly classifying

Table 41 – Performance evaluation of SVM classifiers for each class in the YUPENN dataset when classifying each 16-frame block individually (C3D representations).

Class (Abbreviation)	Precision	Recall	F1-score	Number of observations
Beach	0.9950	0.9741	0.9866	2084
Elevator	1.0000	1.0000	1.0000	2063
Forest fire (ffire)	1.0000	0.8396	0.9128	1340
Fountain	0.8909	0.8667	0.8786	1988
Highway	1.0000	0.9641	0.9817	1975
Lightning storm (lstorm)	0.9034	0.8866	0.8949	1729
Ocean	0.9897	1.0000	0.9948	2025
Railway	0.9886	1.0000	0.9943	1906
Rushing river (rriver)	0.9361	1.0000	0.967	1977
Sky clouds (sclouds)	0.9252	0.9956	0.9591	2025
Snowing	0.9284	0.9977	0.9618	2182
Street	1.0000	0.9926	0.9963	2040
Waterfall	0.8924	0.8681	0.8801	2025
Windmill farm (wmfarm)	1.0000	1.0000	1.0000	2075
Average / Total	0.9606	0.96	0.9596	27434

samples as *Fountain* and *Waterfall*, which indicate that these classes have a less defined boundary in this feature space. The feature space might be improved by including information about the new dataset in the network by fine-tuning it to this dataset.

5.4.4.2.5 Combination by statistical measures

In this experiment, representations were obtained for each 16-frame-blocks in a video using C3D and combined by concatenating multiple statistical measures computed using all blocks in each video from the YUPENN dataset. Table 47 shows the results from the grid search performed to define the penalty parameter C of the SVM. This SVM was then trained on the entire training set and used to classify a set of unseen observations. The results from this classification are shown as a per-class evaluation in Table 48 and as a confusion matrix displayed in Figure 51. By analysing these results, it is clear that combination using statistical measures hindered the ability of the SVM classifier to predict the class of each of observation. It caused most observations to be predicted as being from the classes *Forest fire* and *Fountain*. Also, *Snowing* observations were completely misclassified.

5.5 Concluding remarks

In this chapter, we presented experiments that give a better understanding of the capabilities of current state-of-the-art hand-crafted feature extraction and representation learning methods. In this experiments two techniques were used to extract features from four video dataset,

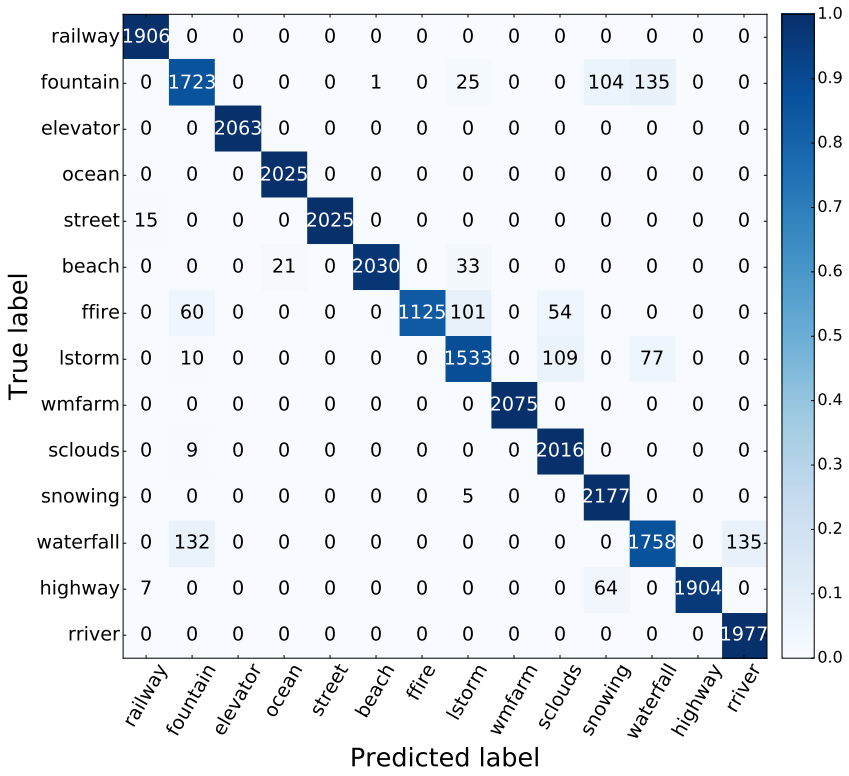


Figure 47 – Confusion matrix of the classification of 16-frame blocks from videos in the YUPENN dataset. The number of observations indicates the number of blocks extracted from each class. Colours show the percentage of observations from each class (rows – true label) that were predicted to be from each class (columns – predicted label).

two focused on action recognition and two on dynamic scenes recognition. These features were then used for classification using a linear SVM. Based on the results achieved in the proposed classification tasks it was possible to analyse the complexity of the feature space created by each of the methods, by looking at the separability of each class in different tasks and settings, and to assess the generalisation capacity of each feature extraction method tested.

For experiments with IDT-FV, the local descriptor selected was the MBH. This choice was made due to MBH's robustness to camera motion and the fact that it produces good descriptors based on each trajectory (WANG *et al.*, 2013). Even so, it is possible that other local feature extraction methods would provide better descriptors. It is important to highlight that the definition of what method will be used to describe the region surrounding each trajectory is crucial to the performance of the IDT-FV technique. This is not a simple choice and thorough knowledge of the dataset/task is required to select a suitable method.

When using IDT, it is mandatory to use a quantisation technique due to the large size of the extracted representations and because their size varies depending on the length of the video and number of trajectories detected. Fisher Vector (FV) was selected for these experiments because they were shown to achieve better results when used with a linear classifier (ONEATA; VERBEEK; SCHMID, 2013). To compute FVs, two parameters need to be defined: the number of components selected with PCA and the number of words created using GMM. Both these

Table 42 – Per class results of the classification of test videos from the YUPENN dataset described by C3D and combined using voting.

Class (Abbreviation)	Precision	Recall	F1-score	Number of observations
Beach	1.0000	1.0000	1.0000	15
Elevator	1.0000	1.0000	1.0000	15
Forest fire (ffire)	1.0000	0.8667	0.9286	15
Fountain	0.8667	0.8667	0.8667	15
Highway	1.0000	1.0000	1.0000	15
Lightning storm (lstorm)	0.9286	0.8667	0.8966	15
Ocean	1.0000	1.0000	1.0000	15
Railway	1.0000	1.0000	1.0000	15
Rushing river (rriver)	0.9375	1.0000	0.9677	15
Sky clouds (sclouds)	0.9375	1.0000	0.9677	15
Snowing	0.9375	1.0000	0.9677	15
Street	1.0000	1.0000	1.0000	15
Waterfall	0.8667	0.8667	0.8667	15
Windmill farm (wmfarm)	1.0000	1.0000	1.0000	15
Average / Total	0.9625	0.9619	0.9615	210

Table 43 – Results for the grid search on the YUPENN dataset. The representation for each video was obtained using k-means quantisation combination on 16-frame blocks descriptors extracted by C3D.

Kernel	C	F1-score (Average)	F1 score (Standard deviation)
Linear	1	0.017	0.029
Linear	**10**	**0.031**	**0.035**
Linear	100	0.031	0.035
Linear	1000	0.031	0.035

parameters were chosen according to the values used in the original IDT-FV paper (ONEATA; VERBEEK; SCHMID, 2013): 64 components and 1000 words.

Regarding C3D, the definition of the network architecture and its parameters, such as learning rate and pooling region, was avoided by using a pre-trained network. This decision was made because finding a network architecture (and its hyper-parameters) that is suitable for each dataset, and training this network is very time expensive and requires special care to avoid overfitting on the training set, especially on small datasets such as KTH-Actions, Maryland and YUPENN. Also, by using a pre-trained network, it was possible to evaluate the generalisation ability of the learnt features to different datasets and tasks. When defining the network architecture, the size of the input also needs to be defined, which will directly influence the resolution and length of the blocks of each video that will be analysed by the network. The pre-trained network selected for the experiments works with groups of 16 consecutive frames at

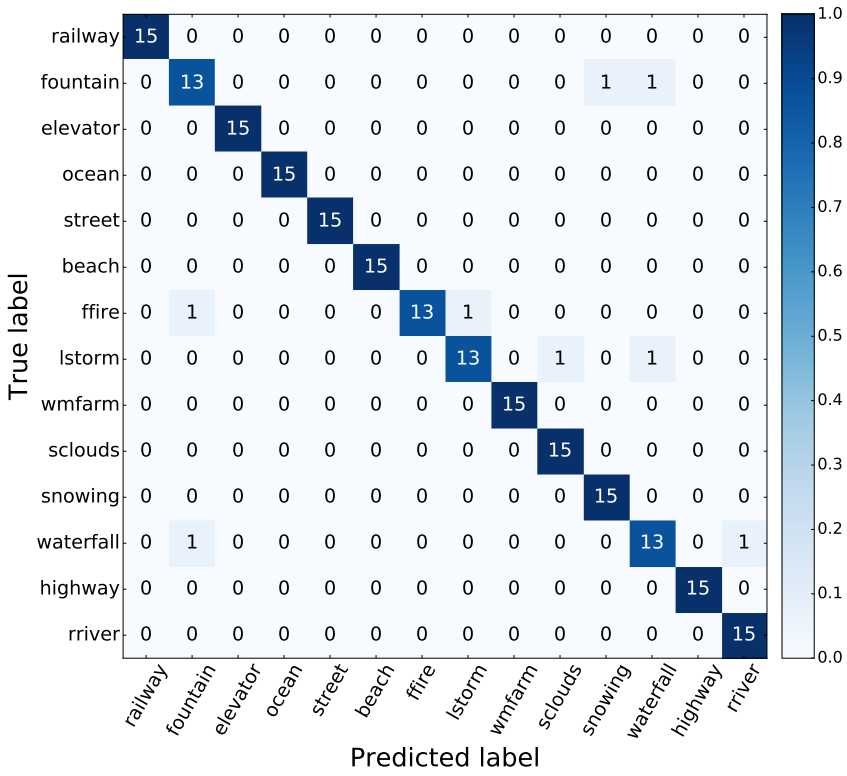


Figure 48 – Resulting confusion matrix of dynamic scene classification on the YUPENN dataset with C3D representations combined by a voting scheme to achieve a video-level prediction.

a resolution of 128×171 pixels.

With a pre-trained model, it is then necessary to define which layer activation will be used as feature vectors. This is most commonly done by choosing fully connected layers. In this experiment, the activation of all fully connected layers in the network was concatenated to achieve a block-level representation. To describe videos with variable length it is necessary to use a method that combines the representations of all blocks in a video. We explored a few simple combination methods during the experiments presented in this chapter.

When analysing the results for both action recognition datasets (KTH-Actions and Hollywood2 Actions), hand-crafted descriptors performed significantly better than the ones extracted with a representation learning method (C3D). This is probably due to task-specific information used during the design of the hand-crafted features. For KTH-Actions, different mistakes were committed depending on the method used to extract the descriptors, which indicates that each method was capable of capturing different information from the videos and that the performance may be improved if the methods are combined in an ensemble or adapted to include more information. This can be done by using different local descriptors with IDT-FV or by fine-tuning the C3D network with this specific dataset. With Hollywood2 Actions, the results were quite similar, though the classification of each 16-frame block and the combination by voting were not used because actions are not performed during the entire duration of each video. This also caused the descriptors extracted by C3D to contain a lot of irrelevant information

Table 44 – Performance of linear SVMs on the YUPENN dataset described using k-means quantisation of the 16-frame block descriptors extracted by C3D.

Class (Abbreviation)	Precision	Recall	F1-score	Number of observations
Beach	1.0000	0.9333	0.9655	15
Elevator	1.0000	1.0000	1.0000	15
Forest fire (ffire)	1.0000	0.8000	0.8889	15
Fountain	1.0000	0.4000	0.5714	15
Highway	1.0000	0.9333	0.9655	15
Lightning storm (lstorm)	0.5417	0.8667	0.6667	15
Ocean	0.9333	0.9333	0.9333	15
Railway	1.0000	0.8667	0.8286	15
Rushing river (rriver)	0.9375	1.0000	0.9677	15
Sky clouds (sclouds)	0.8333	1.0000	0.9091	15
Snowing	0.9286	0.8667	0.8966	15
Street	0.7500	1.0000	0.8571	15
Waterfall	0.9286	0.8667	0.8966	15
Windmill farm (wmfarm)	1.0000	1.0000	1.0000	15
Average / Total	0.9181	0.8905	0.8891	210

Table 45 – Grid search for the *C* parameter of the SVM when describing the videos from the YUPENN dataset using C3D representations combined by averaging.

Kernel	C	F1-score (Average)	F1-score (Standard deviation)
Linear	**1**	**0.935**	**0.091**
Linear	10	0.935	0.091
Linear	100	0.935	0.091
Linear	1000	0.935	0.091

which hindered the performance of the classifier.

On the other hand, the representation learning method performed considerably better than hand-crafted features when used for dynamic scene recognition. It is important to notice that, in this case, neither methods were trained or designed for this task. The experiments in the Maryland dataset showed that using different combination methods to create video-level representations based on representations extracted from 16-frame-blocks with the C3D method may heavily influence the final classification results. Again, the mistakes made varied with the feature extraction method, indicating that different information was encoded by each method. The same can be seen when using the YUPENN dataset to evaluate the efficiency of the extracted feature vectors for dynamic scene recognition.

By analysing the results of all the combinations used with the C3D method over all datasets, the following conclusions were made:

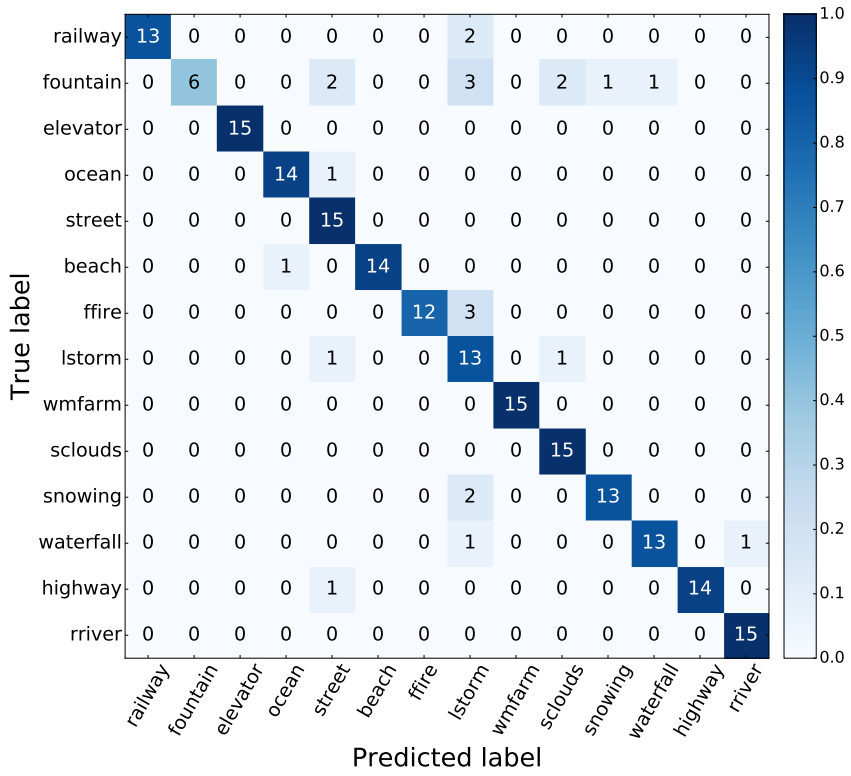


Figure 49 – Confusion matrix created based on the results of the classification of descriptors extracted by C3D and combined by k-means quantisation to obtain video-level descriptors for the YUPENN dataset.

- **Classification of 16-frame-blocks:** used to evaluate the descriptors extracted by the C3D method directly, without the influence of any combination methods. Although it would be the best way to use the representations (since no modifications are made in the feature space), it requires precise time stamps of when the event of interest will occur in the video or the assumption that it occurs during the entire video (assumed during the experiments). This method consistently achieved one of the best classification results, together with combination by voting and averaging.

- **Combination by voting:** similar to the classification of each 16-frame-blocks, using a voting scheme allows a better evaluation of the performance of the descriptors in each dataset since it enables the comparison of the results with other feature extraction methods. Again, this method depends on precise time stamps or the assumption that the event occurs during the entire video.

- ***k*-Means quantisation:** each 16-frame-block descriptor is associated with a cluster using the k-means algorithm. Then, a histogram is created to accumulate the number of 16-frame-blocks in a video that belongs to each cluster creating a histogram. This histogram is used as a video-level representation. Since videos have different numbers of 16-frame-blocks, the histogram is normalized so that it contains the percentage of 16-frame-blocks from the video in each cluster. The efficiency of the descriptors produced by combining the

Table 46 – Per class results on the test set of the YUPENN datasets using descriptors extracted using C3D and combined by averaging the representations of all 16-frame blocks contained in each video.

Class (Abbreviation)	Precision	Recall	F1-score	Number of observations
Beach	1.0000	1.0000	1.0000	15
Elevator	1.0000	1.0000	1.0000	15
Forest fire (ffire)	1.0000	0.8667	0.9286	15
Fountain	0.8125	0.8667	0.8387	15
Highway	1.0000	1.0000	1.0000	15
Lightning storm (lstorm)	0.9286	0.8667	0.8966	15
Ocean	1.0000	1.0000	1.0000	15
Railway	1.0000	1.0000	1.0000	15
Rushing river (rriver)	0.9375	1.0000	0.9677	15
Sky clouds (sclouds)	0.9375	1.0000	0.9677	15
Snowing	0.9375	1.0000	0.9677	15
Street	1.0000	0.9333	0.9655	15
Waterfall	0.8667	0.8667	0.8667	15
Windmill farm (wmfarm)	1.0000	1.0000	1.0000	15
Average / Total	0.9586	0.9571	0.9571	210

Table 47 – Grid search performed in feature vectors obtained by computing and concatenating multiple statistical measures from representations extracted using C3D from all 16-frame blocks in each video from the YUPENN dataset.

Kernel	C	F1-score (Average)	F1-score (Standard deviation)
Linear	**1**	**0.529**	**0.077**
Linear	10	0.529	0.077
Linear	100	0.529	0.077
Linear	1000	0.529	0.077

16-frame-blocks with the k-means quantisation was below the average efficiency of all tested combination methods. This was probably due to creating a feature space that is not suitable for a linear classifier.

- **Combination by average:** by combining the 16-frame-blocks' descriptors using the average of all blocks in a video, the SVM classifier achieved the second-best results, close behind the results achieved using combination by voting. Since voting is not always applicable and averaging is a simple way to create a fixed-size video-level descriptor, this is a good baseline combination method.

- **Statistical measures:** encouraged by the good performance achieved by combination by averaging. The combination with statistical measures was an attempt to include more information by concatenating multiple statistical moments: average, standard deviation, skewness, kurtosis, minimum and maximum. The inclusion of more information hindered

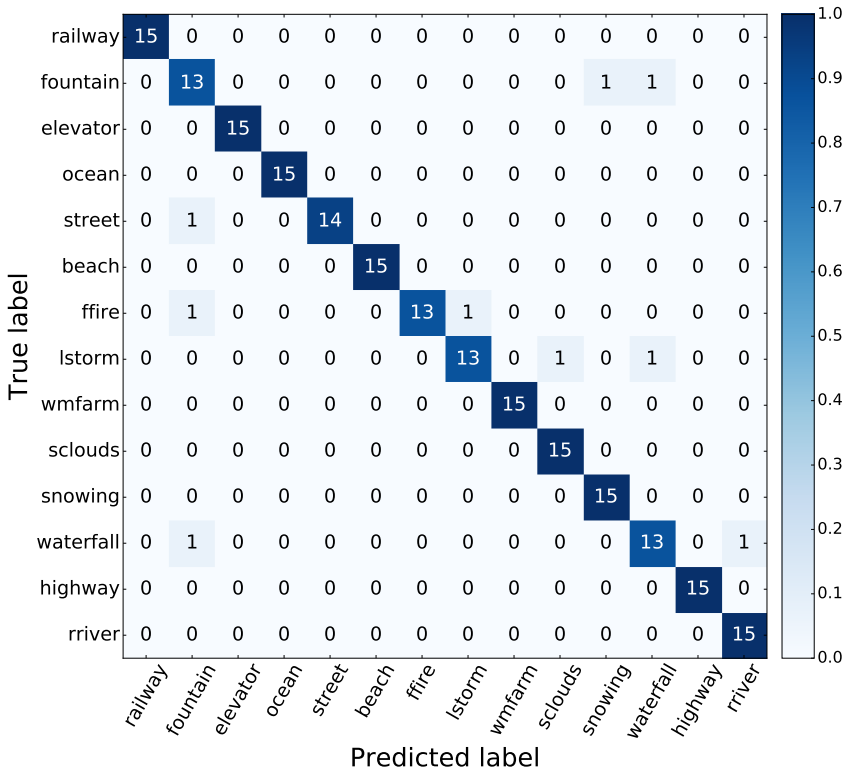


Figure 50 – Confusion matrix obtained from the classification of the test set of the YUPENN dataset. C3D descriptors were used to describe 16-frame blocks from each video and then combined by averaging all blocks to describe each video.

the performance of the linear SVM, which probably occurred due to non-linearities embedded into the new feature space.

The main focus of this chapter was to analyse one of our assumptions that representation learning methods have better generalisation capabilities than the state-of-the-art hand-crafted feature extraction method. The results show that our assumption is correct, however hand-crafted features still have better performance for the task it was designed for. Also, the results presented in this chapter complement the ones presented in the previous chapter regarding the need to have a better understanding of how the information (both spatial and temporal) is encoded by a representation learning architecture. The chosen datasets offer real-word baselines with different settings that allow us to evaluate how generic and which information the representation learning method was capable of encoding. It is important to notice that the analysis showed that the information encoded by the representation learning and the hand-crafted feature extraction methods were complementary, which shows there is room for improvement on both approaches.

Table 48 – Performance of the SVM classifier for each class in the YUPENN dataset. Each video was described using the concatenation of multiple statistical measures computed on all representations (C3D) of 16-frame blocks in a video.

Class (Abbreviation)	Precision	Recall	F1-score	Number of observations
Beach	0.8571	0.8000	0.8276	15
Elevator	1.0000	0.7333	0.8462	15
Forest fire (ffire)	0.2364	0.8667	0.3714	15
Fountain	0.2564	0.6667	0.3704	15
Highway	1.0000	0.5333	0.6957	15
Lightning storm (lstorm)	1.0000	0.6000	0.7500	15
Ocean	1.0000	0.4667	0.6364	15
Railway	0.5556	0.6667	0.6061	15
Rushing river (rriver)	0.8824	1.0000	0.9375	15
Sky clouds (sclouds)	1.0000	0.6000	0.7500	15
Snowing	0.0000	0.0000	0.0000	15
Street	1.0000	0.6000	0.7500	15
Waterfall	0.9091	0.6667	0.7692	15
Windmill farm (wmfarm)	1.0000	0.2000	0.3333	15
Average / Total	0.7641	0.6000	0.6174	210

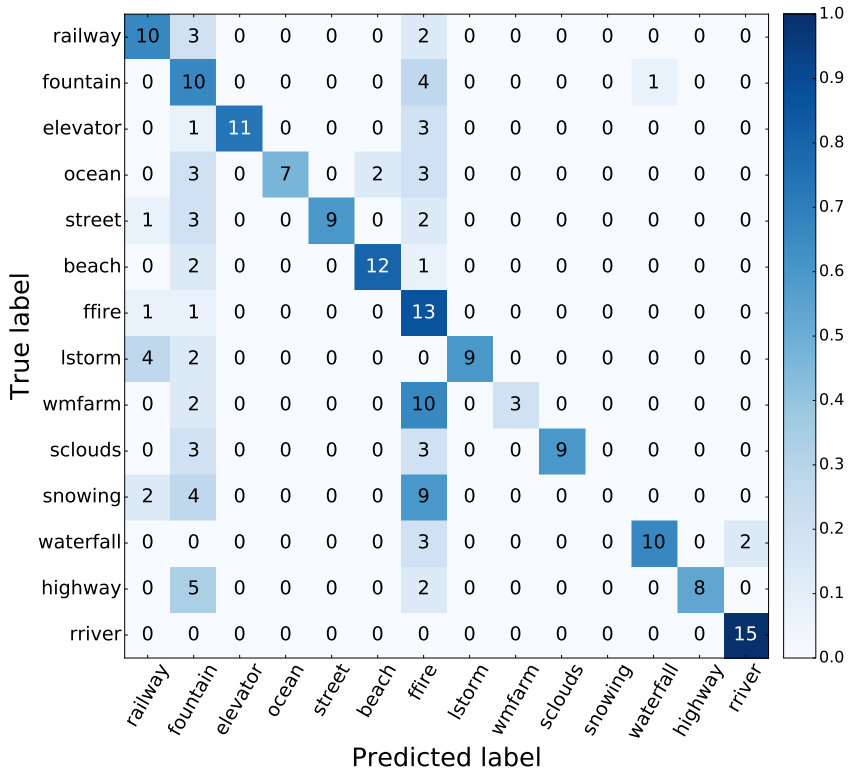


Figure 51 – Results from the classification performed by SVMs, on representations extracted by C3D and combined using multiple statistical measures, presented as a confusion matrix.

CHAPTER

6

GENERATIVE ADVERSARIAL NETWORKS FOR KNOWLEDGE TRANSFER

6.1 Opening remarks

As shown by the results in Chapter 5, state-of-the-art deep learning methods have shown promising generalisation capabilities when applying the representations learnt to completely different datasets and tasks. However, whenever a new dataset or task needs to be handled, it is usually required to train a new classifier using a labelled sample that contains examples from every possible class in that task. One way of improving the performance of the final classifier and reduce the need of large amounts of labelled that is through *regularisation*, that is, introducing additional information to prevent overfitting or even to help solve an ill-posed problem. For example, when solving a classification problem, learning a classifier from a finite training dataset is an underdetermined problem, since it is necessary to infer a function of any x given only a small subset of examples $x_1, x_2, \ldots, x_n$.

The simplest form of regularisation adds a regularisation term, also called a *regulariser*, $R(f)$ to the loss function. This procedure is shown in Equation 6.1, in which $\mathscr{L}$ is an underlying loss function that describes the cost of predicting $f(x_i)$ when the label is y_i, and λ is the *regularisation hyper-parameter*, which controls the importance of the regularisation term. $R(f)$ is typically chosen to impose a penalty on the complexity of f.

$$\min_f \sum_{i=1}^{n} \mathscr{L}(f(x_i), y_i) + \lambda R(f) \tag{6.1}$$

Many machine learning problems can be seen or structured as a set of smaller problems with similar characteristics. For example, multiple similar problems can be created by subsampling the classes in a problem and creating multiple slightly different classification problems, each one with a different subset of classes. The same can be done by subsampling the labelled

examples used for training a model, different sets of labelled examples will produce different models that try to solve the same problem. By creating a set of models that share a similar objective task, we believe it is possible to find patterns in those models that would help when training a new model. This could be done as a type of regularisation and which helps when training new models that consider previously unseen classes or classes with too few labelled samples, among other applications.

One of the ways to find the patterns on the set of known models is to assume there is a probability distribution from which these models were drawn. If this probability distribution is known, the search space that needs to be covered during the training procedure is reduced, making it easier to find a suitable model and to avoid bad local minima that might exist in the model space. In this chapter, we used Generative Adversarial Network (GAN) as an estimator for this probability distribution.

Traditional GANs (Section 2.8) train two neural networks in an adversarial manner, where one is called generator and tries to map a known probability distribution to the objective distribution, while the other is called discriminator and tries to differentiate between examples created by the generator and real examples. As training progresses, the generator should be able to draw examples from distributions that are increasingly more similar to the real samples and the discriminator should become a better estimator of the objective probability density function.

By using GANs to model the probability distribution of the set of known models, many different possibilities arise. The first and most explored during our experiments is the use of the discriminator as a regulariser to guide the training procedure of new models for similar problems. This was done in multiple different ways, varying according to the GAN model used. The use of the discriminator as a regulariser is further explained in Section 6.2. Another possibility is the use of the generator to sample examples that are used as an initial condition for the new classifier, aiming to reduce the training time and avoiding certain local minima. This idea is also explored in Section 6.2.

We believe that using the GAN regularisation method to guide the training procedure helps machine learning methods overcome the lack of labelled samples and increase the training speed by performing knowledge transfer from the known models used to train the GAN to the new model. Also, many machine learning methods are susceptible to their initial condition. That is, if a better initial condition is found by sampling from the probability distribution learnt by the GAN, it is likely that the network will be able to reach a good result in a smaller number of training epochs or even achieve a better final condition. Since the neural network's hidden layers are often used as a way to extract representations from raw data, these approaches are directly applicable to improve representation learning.

6.2 Experimental setup

The main idea explored during these experiments was the use of GANs to transfer knowledge from trained models to new models with similar objectives. We believe this approach will help new models converge faster, increase their generalisation capability and allow for better weight initialisation.

Two sets of experiments were conducted. The first set used linear SVM models in a one-vs-all setting. Multiple models were obtained by using random subsets of observations to train the classifier and by varying the cost parameter C. The linear SVM model consists in learning $\mathbf{w}$ and b, considering $\mathbf{wx}+b$ where the sign of the result indicates the class of $\mathbf{x}$ and $\mathbf{x}$ is a feature vector of an input observation. Learnt values of $\mathbf{w}$ and b were then used to train a GAN. This GAN's discriminator should be able to differentiate between real classifiers and generated classifiers. We believe the knowledge encoded by the discriminator can be used as a regulariser when training a new classifier with a similar objective. The generator can also be exploited to create suitable initialisation weights that would help improve training time and possibly avoid certain local minima. The pipeline used for this procedure can be seen in Figure 52.

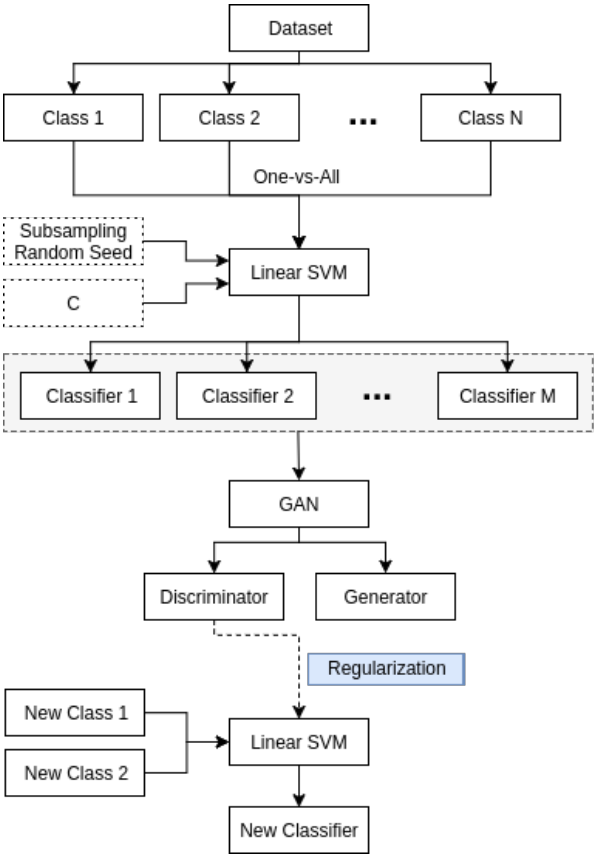


Figure 52 – Methodology pipeline.

The second set of experiments focused on using residual networks models, more specifically Resnet v1 (HE *et al.*, 2016), and on the CIFAR datasets. The CIFAR-10 dataset was used to train a Resnet v1 network formed by 20 residual blocks (this network will be called *Resnet20v1*)

from scratch. All weights from this network were then frozen and the last layer (softmax) was replaced with a new layer. This new layer was then trained by itself using subsets of classes from the CIFAR-100 dataset. The classes from the CIFAR-100 dataset were split into training and test sets and, in this stage, only training classes are used.

The weight matrices learnt by the model for the softmax layer for different subsets of classes are then used to compose the dataset. Next, a GAN is used to model the probability distribution of the weight matrices dataset, resulting in the discriminator being able to differentiate between real and generated samples and a generator capable of creating new weight vectors. Both these models have the potential to improve the results when training a new classifier. The discriminator can be used to regularise the learning of the new model by guiding it to the same probability distribution as the models in the dataset, and the generator by creating weight vectors used to improve the weight initialisation of the model. A pipeline explaining this procedure is shown in Figure 53.

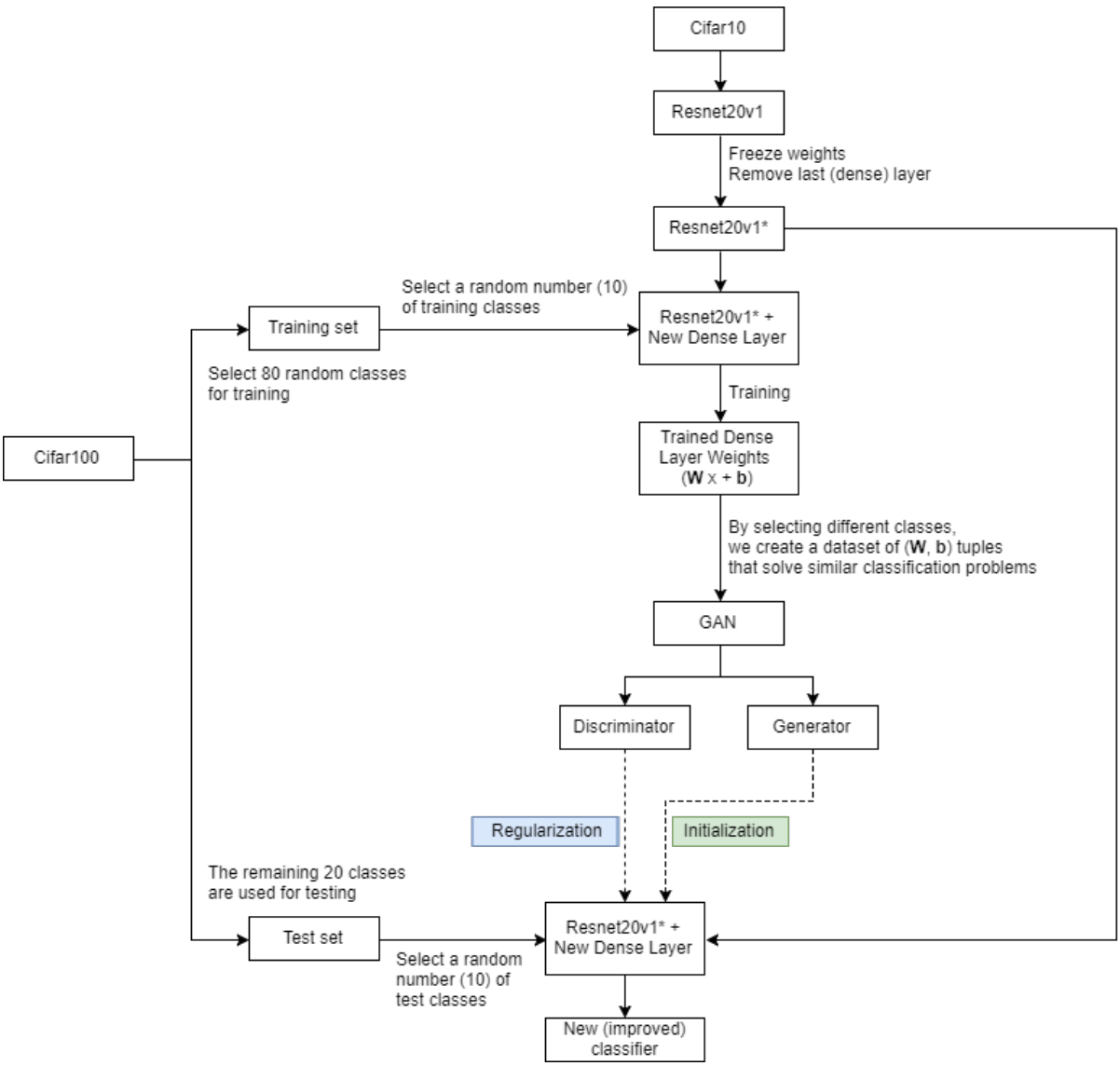


Figure 53 – Pipeline used for the second set experiments using the CIFAR datasets.

Another version of this dataset was made by replacing not only the softmax layer but also the entire last residual block, which consists of two convolutional layers and two batch normalisation layers. In this case, filters from the convolutional layers and their respective biases, in addition to the weight matrix and biases learnt by the softmax layer, were stored. More details about the created datasets are given in Section 6.3.

6.3 Datasets

To apply the idea explained in the previous section, we created datasets containing sets of weight vectors acquired by training multiple models on similar problems. This was done for different tasks and classification algorithms. Also, a two-dimensional synthetic dataset for visualisation purposes, allowing the analysis of the behaviour of both the generator and the discriminator during different stages of training.

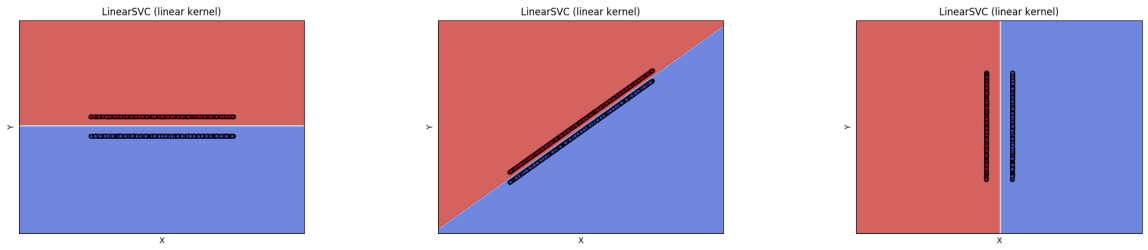
Due to the complexity of the problem being investigated and the novelty of the proposed method, we chose to explore simpler and well-known applications and datasets. To this end, we decided that a two-dimensional synthetic dataset would provide a well-controlled setting that allows for easier visualisation and understanding of the method performance. Also, we forgo the analysis of the effect of this method on spatio-temporal features and focus on spatial features for the time being. Image datasets provide lower-dimensional problems that have been more thorough explored, especially in few-shot learning settings. We believe that, once the method is established for image (spatial) problems, its extension to spatio-temporal problems is straight forward.

6.3.1 Synthetic dataset

This synthetic dataset was created by training linear SVM classifiers on samples purposely created so that every classifier behaved in a neutral or correlated manner where the positive class is always positioned in the top-left side of the classifier. Also, every classifier is centred as to avoid the need for a bias term. The main goal of using this dataset is that it allows us to visualise and compare the generated and real samples, and analyse how the discriminator classifies relevant regions in the two-dimensional model space.

Examples of this dataset can be seen in Figure 54, where the background colour indicates the output of the classifier (red being the region classified as positive and blue classified as negative). The dots (samples used to train the classifier) on the image are discarded after training has finished and only the weight vector are saved. In this case, since linear SVM was used, the weight vector is given by $\mathbf{w}\mathbf{x} + b$, where $\mathbf{x} \in \mathbb{R}^2$ is the feature vector for the sample being classified, $\mathbf{w} \in \mathbb{R}^2$ is the weight vector learnt by the classification algorithm and b is the bias vector. Since the classifiers are centred, b is always 0 and is disregarded, and only the values of $\mathbf{w}$ are saved. The resulting two-dimensional dataset can be seen represented by the red dots shown

in Figure 55. The blue dots represent a second version of this dataset where random Gaussian noise was added to the samples of the first version.



(a) Example 1 (b) Example 2 (c) Example 3

Figure 54 – Examples of classifiers used as the training set for the GAN. The set of classifiers used for training contain only centred linear classifiers which present a positive correlation between the axis and where the positive class (red) is on the top-left side of the classifier. The samples used to train the classifiers are discarded and only the linear SVM weight vector is used to create the dataset.

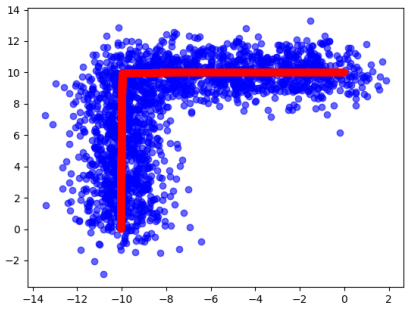


Figure 55 – Scatter plot showing the behaviour of the synthetic dataset created. The original data is shown in red, while a second version, disturbed by random Gaussian noise, is shown in blue.

6.3.2 Omniglot

The Omniglot dataset (LAKE; SALAKHUTDINOV; TENENBAUM, 2015) is composed of images of 1623 different handwritten characters from 50 different alphabets. Each image was obtained via Amazon's Mechanical Turk by asking 20 different people to draw each character. The Omniglot dataset is generally split into a training (background) set containing 30 alphabets and an evaluation set with the remainder 20 alphabets.

Initially designed for one-shot learning, we adapted the dataset in a way to fit our requirements so that we were able to train multiple models for similar tasks, in this case, character recognition. To create the trained models dataset, two different approaches were used: classifying the images based on pixel values and describing the images using HOG descriptors (DALAL; TRIGGS, 2005) before classification.

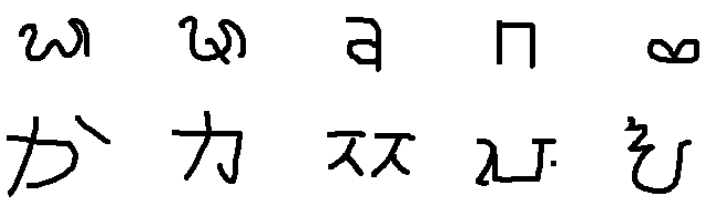


Figure 56 – Example images from the Omniglot dataset.

To create the dataset of trained models, we first extracted subsets of ten classes from the training set and then train models in a one-vs-all setting. Different subsets allow the training of multiple different models that have the same objective, character recognition. Further variance was achieved by varying the parameters used to train each model. In this case, linear SVM models were used. Examples of the classifiers obtained during this process can be visualised in Figure 57, where linear SVM models were trained on raw pixel images.

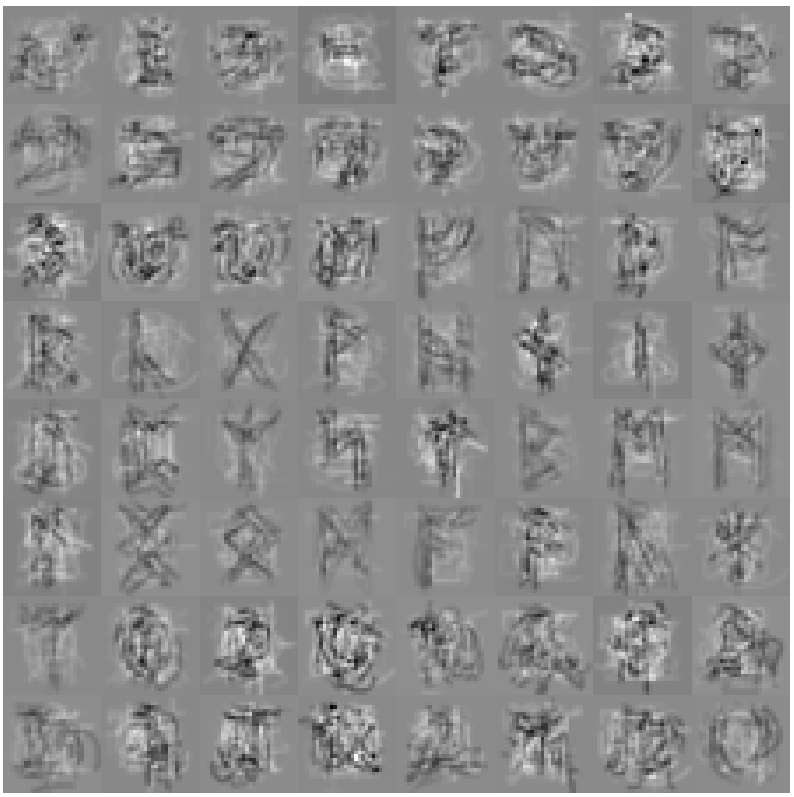


Figure 57 – Visualisation of the models obtained when training linear SVM classifiers on the raw pixels of subsets of classes of the Omniglot dataset.

6.3.3 CIFAR

There are two versions of the CIFAR dataset (KRIZHEVSKY; HINTON, 2009) known as CIFAR-10 and CIFAR-100. Both datasets are composed of 32×32 colour images which are split into disjoint sets of classes, 10 classes in CIFAR-10 and 100 classes in CIFAR-100. CIFAR-10 contains 60000 images (6000 per class) which are split into training (50000) and test

sets (10000). CIFAR-100 is similar to CIFAR-10, with the exception that the images are split into 100 classes, each containing 600 images. Again, these images are split into training images (500 per class) and testing images (100 per class).

To create the dataset of trained models, we start by training a Resnet v1 (HE *et al.*, 2016) architecture containing 20 residual blocks (*Resnet20v1*) on the CIFAR-10 dataset. All the weights learnt by the model are then fixed and the last layer (softmax) is discarded and replaced by a new layer, which is trained using a subset of classes from the CIFAR-100 dataset. The weight matrices $\mathbf{W} \in \mathbb{R}^{d \times c}$ and bias vectors $\mathbf{b} \in \mathbb{R}^{c}$ learnt by the model (where $d = 64$ is the number of features passed by the previous network layer to the softmax layer and $c = 10$ is the number of classes included in each subset) for each subset of classes are stored, creating the dataset of classifiers.

A second version of the model dataset was created where instead of training only the softmax layer on the subset of CIFAR-100, the entire last residual block plus the softmax layer are used. This leads to collecting the weights for two convolutional layers and a fully connected layer. In this case, the convolutional filters stored for both convolutional layers are $3 \times 3 \times 64$ while their respective bias vectors have 64 dimensions. The softmax weight matrices and bias vectors have the same sizes as in the previously described dataset.

6.4 Experiments

6.4.1 Understanding GANs

The first experiments aimed at achieving a better understanding of how GANs behaved during training. This was done by using the two-dimensional synthetic dataset presented in Section 6.3.1 and visualising the results at the end of each epoch. Examples of these results for DCGAN are shown in Figure 59. In them, the generated samples from the last 10 epochs can be seen in green and yellow, green being the oldest samples and yellow the newest, while real samples are shown in black. The background colours indicate the output of the discriminator, where red indicates regions classified as real and blue classified as fake. In the bottom graph, the red line shows the discriminator loss, the blue line shows the generator loss and the green line shows the log of the Maximum Mean Discrepancy (MMD) (GRETTON *et al.*, 2012), measure that compares the distribution of the newest generated samples to the distribution of the real samples, where the distributions are considered to be more similar the lower the value of the MMD is.

These results were also used to compose videos that can be found online. These videos show the behaviour of DCGANs using batch normalisation and with the discriminator being fed batches of mixed samples (real and generated)[1] or separate batches for each type[2]. The same

[1] Synthetic dataset - DCGAN - Single-batch: <https://youtu.be/PKUDwgsSgio>
[2] Synthetic dataset - DCGAN - Separate batches: <https://youtu.be/aJaKLpgNQbY>

experiment was conducted using the noisy version of the synthetic dataset[3,4]. This analysis was motivated by conflicting results when using batch normalisation with separate batches for real and generated samples, as is the most common way to use DCGANs.

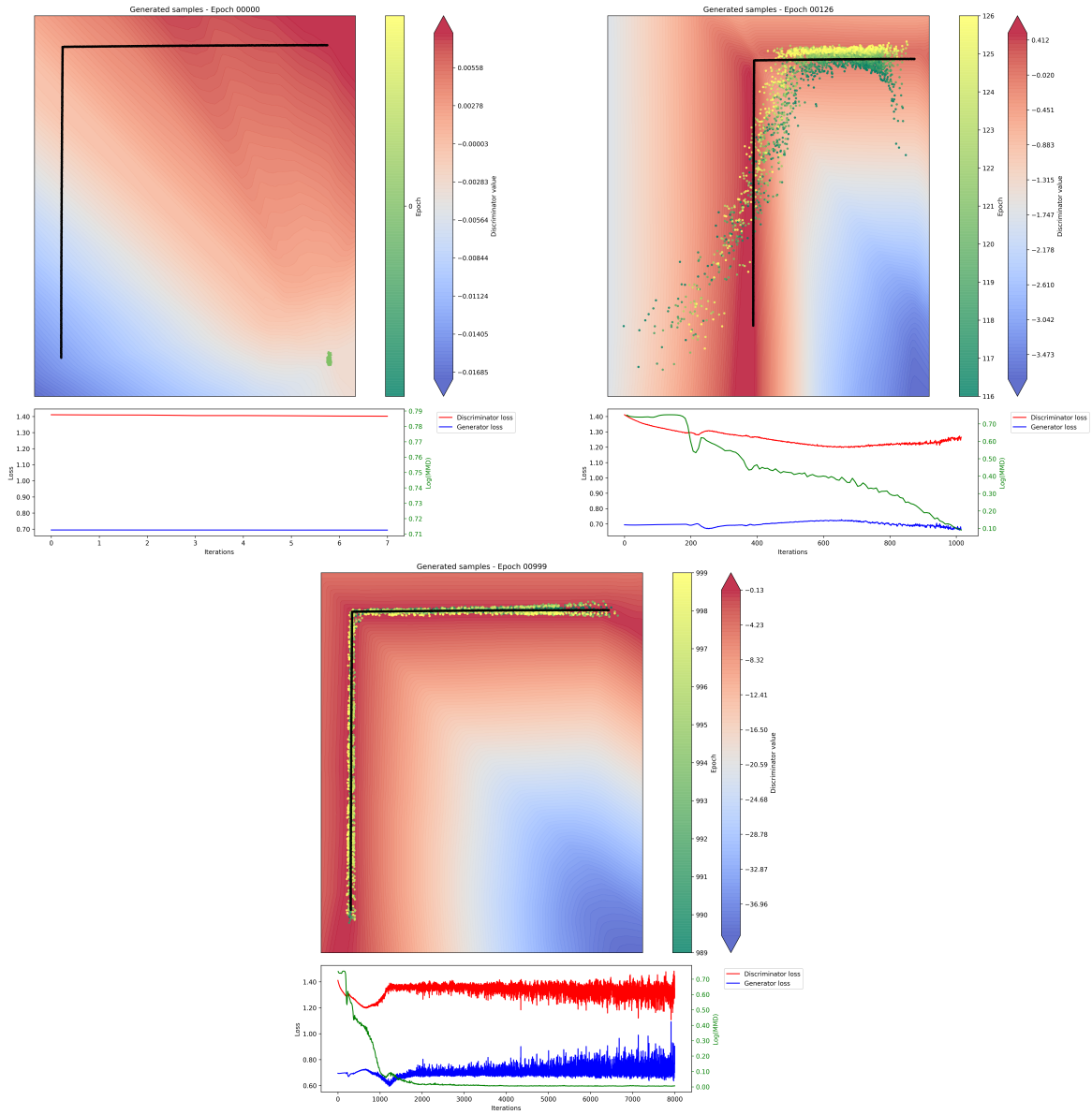
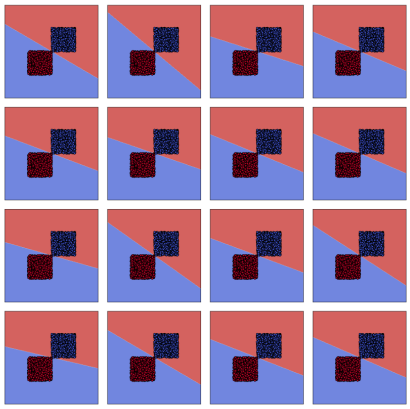


Figure 58 – Visualisation created to analyse the behaviour of the GAN during training. These examples show different stages of training (epochs) when using DCGANs on the synthetic dataset. Real samples are represented by the red dots. The green and yellow dots show the generated samples from the last 10 epochs, oldest to newest, respectively. The background colour indicates the output of the discriminator for each region, where red indicates regions classified as real and blue as fake. The bottom graph shows the discriminator loss (red), the generator loss (blue) and the log of the MMD value (green).

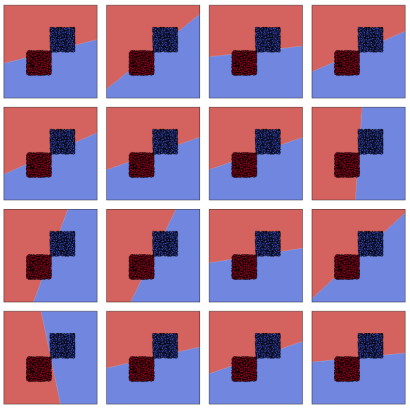
Another possible way of visualising the GAN training is by looking at the generated samples during different stages of training. Since the application is focused on modelling

[3] Noisy Synthetic dataset - DCGAN - Single-batch: <https://youtu.be/IH7vSn3f1l0>
[4] Noisy Synthetic dataset - DCGAN - Separate batches: <https://youtu.be/oBrmzb62y7I>

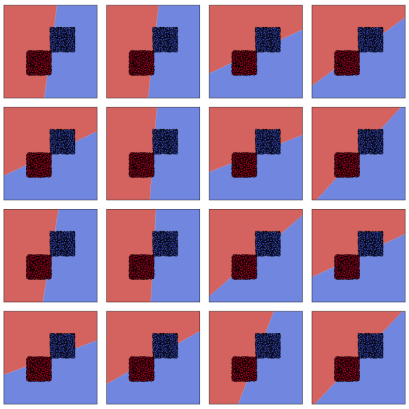
classifier distributions, the generated samples will be classifiers, specifically linear classifiers $\mathbf{wx} + b$ with $b = 0$. Randomly generated samples during different iterations of training can be seen in Figure 59. It is possible to see that the classifiers become more like the training data as training progresses.



(a) Iteration 0



(b) Iteration 450



(c) Iteration 3000

Figure 59 – Randomly generated classifiers in different stages of training of a DCGAN on the synthetic dataset. It is possible to see that the generated classifiers (background colour) become more similar to the training data as training progresses. The dots are only used to highlight the changes in the generated samples.

6.4.2 Using GANs for regularisation

After analysing the results obtained by the GAN, we noticed that the output of the discriminator can be unstable, especially if the GAN training continues after the generator has converged. Since it is hard to know when to stop the training when dealing with high dimensional datasets, an alternative to the discriminator output was proposed based on (EGHBAL-ZADEH;

WIDMER, 2017), where we compute the Maximum Likelihood Estimate (MLE) of the new classifier being from the same probability distribution as the known models dataset using the second-to-last layer output. This is done by fitting a Gaussian distribution to every feature of that layer's output. Then, we compute the sum of the chance that every feature extracted from the new classifier belongs to respective fitted Gaussian.

To test the idea of using GANs to transfer knowledge through regularisation, we first explore it by creating a new problem that contains a possible solution which can be approximated by the information contained in the synthetic dataset. This problem is shown on Figure 60, together with the optimal solution achieved by a linear SVM classifier trained without any kind of regularisation. Notice that the solution achieved by the linear SVM without regularisation and using the entire dataset is completely opposite to the models included in the synthetic dataset, even though a perfect solution is also included in that dataset.

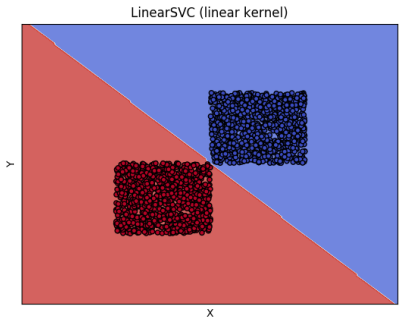


Figure 60 – Training data used to test the GAN regularisation. The dots' colours indicates their classes, while the background colour shows the classifier obtained by training a linear SVM without regularisation.

This problem is addressed in a few-shot setting using 5 samples from each class for training. Ten classifiers were trained using different subsamples (all of them using the same DCGAN discriminator for regularisation) and the average and standard deviation of the results were computed. To analyse the speed of training, we use a model selection process, where we evaluate the performance of the classifier after every epoch and store the epoch where the best results were achieved. The results obtained by using the discriminator output for regularisation can be seen Table 49, and the results for the MLE approach are shown in Table 50. The first two rows of each table show baselines for this experiment: no regularisation and *l2* regularisation.

The results show that using the discriminator output directly as a regulariser did not influence the final model. We believe this happened due to the instability of the discriminator output seen during previous experiments. On the other hand, when the MLE approach was used, the regularisation offered a significant gain for high values of d (λ parameter associated with the discriminator regulariser). The MLE approach also showed great benefits regarding the number of epochs required to achieve the best results. The most probable reason for this is that the GAN regulariser offer more useful information for the model to train on, avoiding overfitting the

Table 49 – Regularisation based on the discriminator output of a DCGAN trained on the synthetic dataset. The regularisation is performed during training a classifier on small (5) subsample of the dataset shown in Figure 60. The columns "d" and "*l2*" show the values of λ used for the discriminator regulariser and *l2* regulariser, respectively. Average and standard deviation were computed over the results of 10 repetitions using different subsets.

d	*l2*	Test Accuracy	First Best Epoch
0	0	49.5000±26.7001	62.2079±38.2427
0	0.01	49.4926±26.6850	62.7723±37.6787
1	0	49.4852±26.7176	61.6634±37.9685
10	0	49.4802±26.7098	60.2079±38.7248
100	0	49.4926±26.7308	63.8218±37.4169
1000	0	49.4777±26.7058	60.3960±38.6108
10000	0	49.4654±26.7233	59.9901±39.0378

Table 50 – Regularisation based on MLE approach using the discriminator of a DCGAN trained on the synthetic dataset. The regularisation is performed during training a classifier on small (5) subsample of the dataset shown in Figure 60. The columns "d" and "*l2*" show the values of λ used for the discriminator regulariser and *l2* regulariser, respectively. Average and standard deviation were computed over the results of 10 repetitions using different subsets.

d	*l2*	Test Accuracy	First Best Epoch
0	0	49.4876±26.7137	61.4158±38.1986
0	0.01	49.4926±26.6842	61.8713±38.321636
1	0	49.6015±26.7838	53.0495±40.001826
10	0	50.8193±26.3080	44.9604±41.284073
100	0	52.4703±26.7162	45.5347±41.5095
1000	0	56.5495±25.4229	33.0099±37.3550
10000	0	59.8267±25.4368	24.3564±31.5110

training data due to the small number of samples available for training.

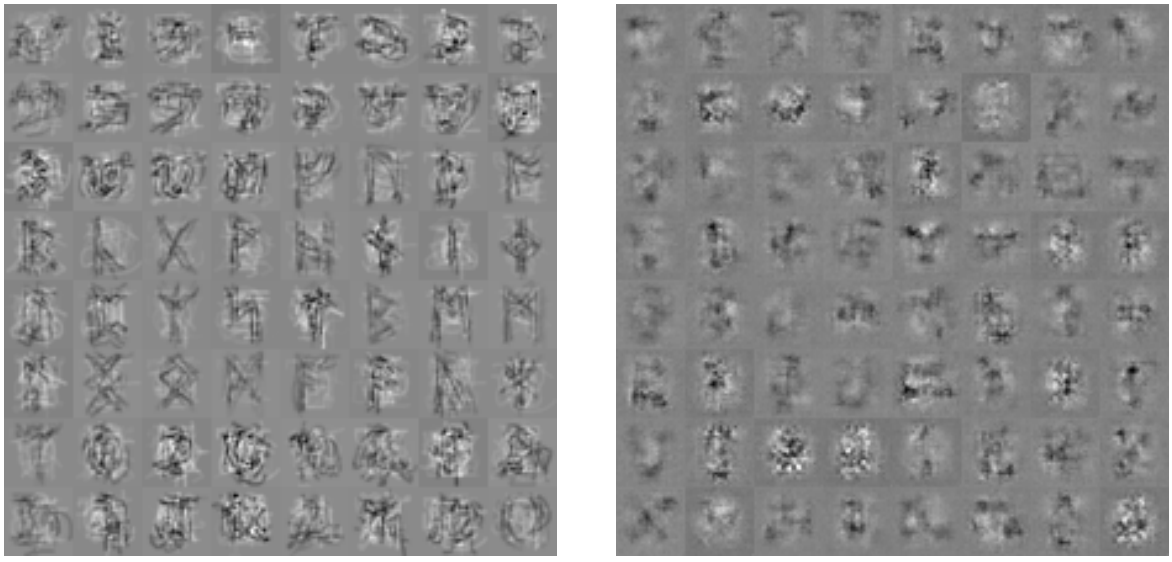
6.4.3 Regularising Linear SVM for character classification

After experiments with a synthetic dataset, extending them to a real-world problem was the natural next step. For that, we selected the Omniglot dataset using HOG descriptors and classifying directly from the raw pixels. Extending the proposed method to this task proved to be a challenge since the settings that worked for the synthetic dataset did not work for the new task. HOG descriptors are structured as a concatenation of multiple histograms, which proved to be a challenge to model using convolutions. It also made it hard to analyse and diagnose the training procedure for the GAN. It is important to notice that the GAN loss is known for not being a good indicator of the GAN's performance.

When using DCGANs, every architecture and hyper-parameter combination tested suffered from mode collapse, which was diagnosed by looking at the generated samples. We then

shifted to WGANs, whose generator seemed to produce reasonable results but when discriminator was used to regularise the learning of a new classifier, both using the discriminator output and the MLE approach, achieved very poor results, hindering the performance when compared to unregularised and *l2* regularised learning. Many different GAN modifications were studied as a possible solution to the faced problems, among them were VAEGANs (LARSEN *et al.*, 2015), VEEGANs (SRIVASTAVA *et al.*, 2017), InfoGANs (CHEN *et al.*, 2016) and BEGANs (BERTHELOT; SCHUMM; METZ, 2017). BEGANs were selected due to being proposed as a way to avoid mode collapse, their relation to the Wasserstein distance, and the fact that having an autoencoder as the discriminator creates an intuitive layer (latent embedding layer) to use for the MLE approach.

The Omniglot dataset using raw pixels and linear SVMs allowed us to visualise the classifiers generated by the BEGAN. This visualisation can be seen in Figure 61, where the left image shows the visual appearance of real classifiers while the right image shows classifiers generated by a BEGAN. It is possible to see similarities between both images, such as visible white and black patterns with different shades of grey as background. However, the generated classifiers do show greater blurriness than the real ones.



(a) Real classifiers (b) Generated classifiers

Figure 61 – Comparison between real classifiers and randomly generated classifiers for the Omniglot dataset using raw pixels and modelled by a BEGAN.

When fitting the Gaussian distributions for features extracted from the central layer of the discriminator, the distributions' standard deviation had an order of magnitude of -5. This makes the Gaussian distributions behave in a similar manner as delta functions, with 0 almost everywhere. Such behaviour makes the gradients derived from the MLE regularisation equal to 0 most of the time, not helping the training procedure. To fix this problem, different architectures with different sizes of embedding space and different types of normalisation for the input were tested but had no impact on the standard deviations.

Since the discriminator of BEGAN is an autoencoder, the output of the discriminator can not be directly used to regularise learning. Nevertheless, the BEGAN's discriminator is trained to minimise the reconstruction error of real examples while maximising the reconstruction error of fake/generated examples. For that reason, we experimented using the reconstruction error as the regularisation measure. Table 51 shows results for the Omniglot dataset in a one-vs-all few-shot learning setting, that is, using 3 examples per class for training, 2 for validation and 15 for testing. This experiment was repeated 50 times using randomly selected classes and training, validation and test sets, for each combination of λ parameters.

Table 51 – BEGAN regularisation using reconstruction error in a few-shot setting (3 training samples, 2 validation samples and 15 test samples per class) on the Omniglot dataset. Average and standard deviation over 50 repetitions using different random subsets.

d	*l2*	Test Accuracy	First Best Epoch
0.0	0.0	51.0000 ±2.1344	1.6000 ±1.2000
0.0	0.01	51.0000 ±2.1344	1.6000 ±1.2000
0.01	0.0	51.0000 ±2.1344	1.6000 ±1.2000
0.1	0.0	53.0000 ±5.2599	4.8000 ±9.4742
1.0	0.0	51.6667 ±2.6874	2.0000 ±1.3416
10.0	**0.0**	**58.6667 ±15.1438**	**22.4000 ±28.9938**
100.0	0.0	55.0000 ±13.9244	10.4000 ±20.2791

The results show considerable accuracy improvement when using $d = 10.0$, however, considerably increasing the standard deviation. It also showed an increase in the average number of epochs required to achieve the best validation accuracy ("First Best Epoch"). We believe this happens because the number of samples used for training is very small which causes the other models to quickly overfit the training data, while discriminator regularisation causes the model to learn during a longer number of epochs.

6.4.4 Regularising residual networks for the CIFAR dataset

Another real-world application used for experiments was the CIFAR dataset, in this case, modelled by a residual network composed of 20 residual blocks (*Resnet20v1*). As explained in Session 6.2, the network is first trained from scratch on the CIFAR-10 dataset, and then only the last layer is retrained for a subset of classes from the CIFAR-100 dataset. Since we define this as a 10-way (multi-class) problem, the weight vector is a matrix $\mathbf{W} \in \mathbb{R}^{d \times c}$, where d is the size of the feature vector input to the softmax layer and c is the number of classes.

Modelling the entire $\mathbf{W}$ matrix proved to be a challenge due to its structure since the columns can be shuffled and the only impact would be a change in the order of the classes of the output. The same happens when shuffling rows; if the input order changes the same way as the order of the matrix rows, there will be no impact on the output. Fully connected architectures caused the models to mode collapse. This motivated the idea of modelling each

column of the matrix separately. By breaking the matrix into columns, it is also possible to use the discriminator to regularise problems that consider any number of classes, since each column will be passed to the discriminator separately and the sum of the computed measures will be used for the regularisation. However, by doing so, the connection between the classifiers learnt is not considered by the GAN and, thereafter, during regularisation.

To evaluate the performance of the BEGAN and see how training was progressing, we generated samples (classifiers) and applied them to every image that belonged to the training classes. From the scores outputted by each classifier, a ranking was created and the top and bottom 10 ranked images were analysed. This visualisation is shown in Figure 62 where each row shows the top 10 (left) and bottom 10 (right) ranked images for different randomly generated classifiers. It is important to notice that these classifiers are generated without any relation to the others.

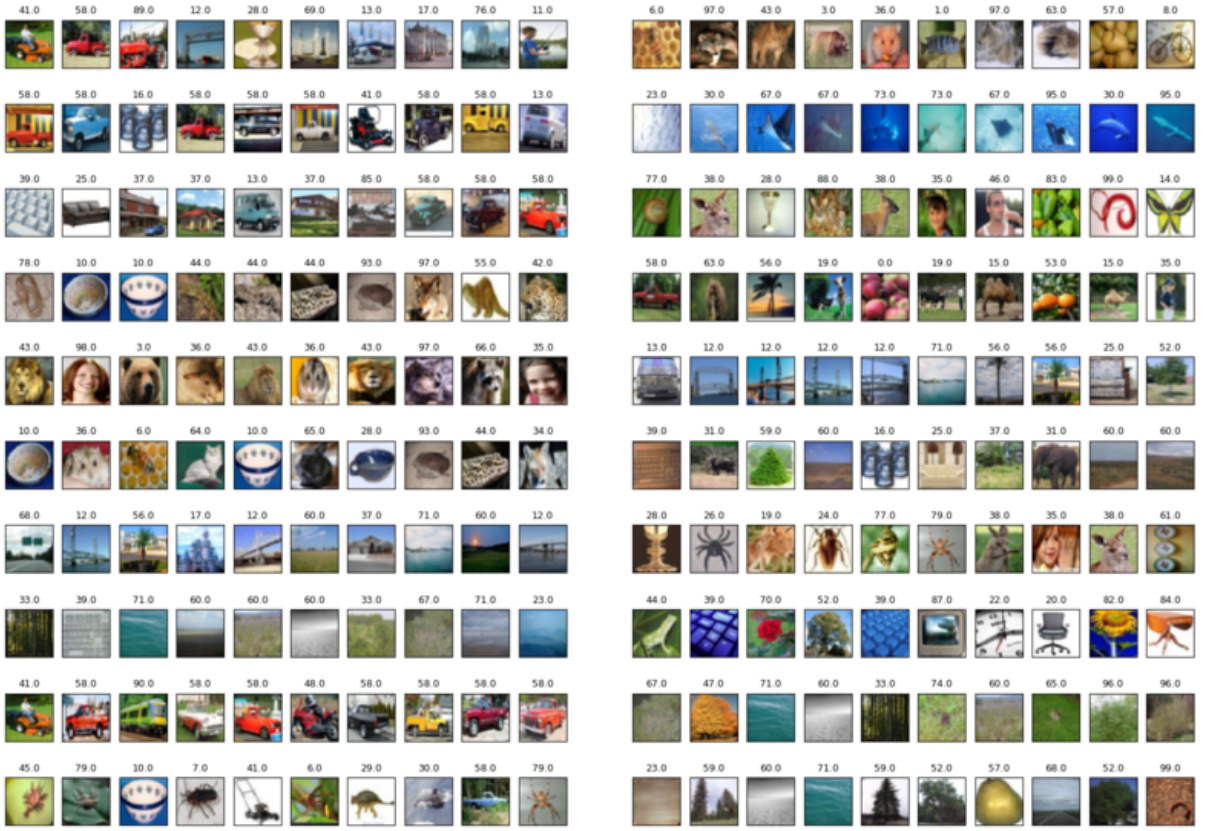


Figure 62 – Visualisation of the results of the models generated by a BEGAN trained on the CIFAR models dataset. Each row shows the results for a different generated classifier. The top 10 images who obtained the highest scores are shown on the left, while the bottom 10 scored images are shown on the right.

When applied to regularisation, both BEGAN regularisation approaches (reconstruction error and MLE) showed better results than unregularised and *l2* regularised experiments. The results shown in Table 52 were obtained by using 7 training examples, 3 validation examples (used to define the first best epoch) and 100 test examples per class and repeated 10 times using different subsets of classes and splits. The reported results are the average and standard deviation

of: the test accuracy after the last training epoch (100), the epoch where the best validation accuracy was achieved and the test accuracy for that epoch. These experiments used a fixed learning rate schedule, with the learning rate starting at 0.001 and being multiplied by 0.1 every 50 epochs three times, then reducing the learning rate again by half after 30 epochs. The training procedure used a total of 200 epochs.

Table 52 – Results for the experiments using BEGAN regularisation on the last layer (softmax) of a residual network. Columns "d" and "*l2*" show the values of the λ parameters for the BEGAN regularisation and *l2* regularisation, respectively. Column "MLE" indicates if the BEGAN regularisation was done using reconstruction error (False) or the MLE approach (True). Average and standard deviation was computed over the results of 10 repetitions using different subsets of classes and splits. Training was conducted using 7 examples, the validation split (used to define the best epoch) contains 3 examples and test used 100 examples.

d	*l2*	MLE	Test Accuracy (Epoch 200)	First Best Epoch	Test Accuracy (Best Epoch)
0	0	–	0.2395 ±0.0203	125.0 ±41.6269	0.2430 ±0.0198
0	0.01	–	0.2236 ±0.0423	127.6 ±40.1054	0.2272 ±0.0431
0.001	0	False	0.2203 ±0.0397	158.1 ±38.1404	0.2234 ±0.0397
0.001	**0**	**True**	**0.2464 ±0.0350**	**109.1 ±29.1700**	**0.2504 ±0.0345**
0.01	0	False	0.2181 ±0.0490	132.2 ±30.248306	0.2203 ±0.0492
0.01	0	True	0.2302 ±0.0404	121.4 ±29.2684	0.2341 ±0.0402
0.1	0	False	0.2147 ±0.05269	137.4 ±31.4013	0.2184 ±0.0534
0.1	0	True	0.2345 ±0.0244	133.4 ±40.2795	0.2378 ±0.0228
1	0	False	0.2099 ±0.0350	141.5 ±42.1456	0.2127 ±0.0356
1	0	True	0.2257 ±0.0388	120.3 ±33.3588	0.2287 ±0.0389
10	0	False	0.2274 ±0.0257	126.4 ±37.3770	0.2309 ±0.0255
10	0	True	0.2307 ±0.0446	141.8 ±31.5525	0.2345 ±0.0437
100	**0**	**False**	**0.3066 ±0.03675**	**138.8 ±31.0670**	**0.3094 ±0.0370**
100	0	True	0.1918 ±0.0322	163.7 ±34.9773	0.1941 ±0.0336
1000	0	False	0.1051 ±0.0180	32.6 ±20.8959	0.2071 ±0.0240
1000	0	True	0.1049 ±0.0242	33.1 ±46.1356	0.1282 ±0.0278

After analysing these results, we believed that the learning rate schedule and λ parameters directly influence the final results. To evaluate this, we performed a random search considering: initial learning rate, decay rate (multiplier used to reduce the initial learning rate) and number of decay steps (number of equally spaced steps the decay rate is multiplied by the initial learning rate). The λ parameters, *d* and *l2*, were also randomly defined. Each regularisation approach is tested independently for all the randomly generated learning schedules. For this experiment, 20 different learning rate schedules were used and the top 5 were selected based on the validation accuracy for each regularisation method.

Figure 63 shows the results for 9 different subsets of classes, where each colour indicates a different regularisation method, the stronger coloured lines indicate the average accuracy while the lighter colours show the standard deviation. It is possible to see that most of the time GAN regularisation achieved a better result, however it was very sensible to the learning

rate schedule, which leads to lower average performance. It is also possible to notice that the GAN regularisation also achieved a higher test accuracy faster than the other methods for most problems shown.

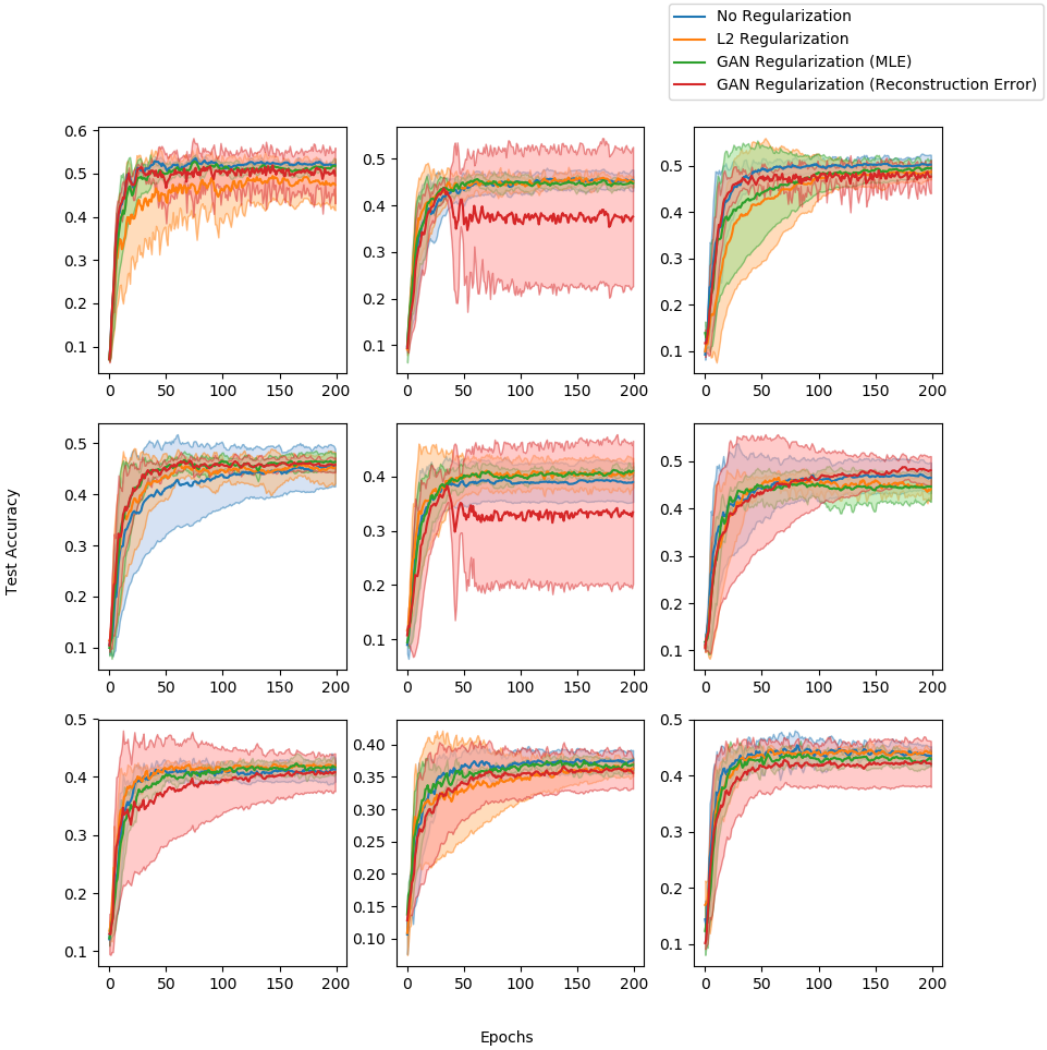


Figure 63 – Plots showing the average and standard deviation test accuracy per epoch for the CIFAR Resnet20v1 combination using different regularisation methods and random search to define learning rate schedules. The top 5 learning rate schedules are selected based on the validation accuracy and used to compute the plotted metrics for 9 different randomly selected subsets of test classes (10-way classification).

6.4.5 Initialisation using generated samples

To explore the generator learnt by the GAN, we propose the use of the generated samples as a way to find a better initialisation for the network parameters. In this experiment we use the BEGAN training on the CIFAR-Resnet20v1 combination to generate weight matrix columns. Since there is no way to know which classes the BEGAN is taking into consideration when generating samples, we combine ten generated columns to create a 10-way classifier and use

the Munkres algorithm (MUNKRES, 1957) to find the order of the columns that maximise the training accuracy.

We first evaluate the performance of the generated matrices comparing with random initialisation and generating random matrices that are then reordered by the Munkres algorithm. The results for GAN generated matrices and random generated matrices are shown in Tables 53 and 54, respectively. These results do not consider any kind of training and are a way to analyse if the probability distribution learnt by the GAN is useful for the learning procedure.

Table 53 – *Initialisation using generated samples (matrix columns)* reordered using the Munkres algorithm to maximise the initial accuracy. When multiple matrices are generated, the one with the best initial accuracy in the training set is selected. Average and standard deviation computed over 10 repetitions using different subsets of classes.

# of generated matrices	Initial Training Accuracy	Initial Validation Accuracy	Initial Test Accuracy
1	0.1324 ± 0.0489	0.1353 ± 0.0498	0.1352 ± 0.0541
10	0.1984 ± 0.0301	0.1990 ± 0.0308	0.1965 ± 0.0311
100	0.2309 ± 0.0270	0.2297 ± 0.0296	0.2319 ± 0.0306
1000	0.2545 ± 0.0244	0.2545 ± 0.0275	0.2553 ± 0.0237

Table 54 – *Initialisation using randomly generated matrices* reordered using the Munkres algorithm to maximise the initial accuracy. When multiple matrices are generated, the one with the best initial accuracy in the training set is selected. Average and standard deviation computed over 10 repetitions using different subsets of classes.

# of generated matrices	Initial Training Accuracy	Initial Validation Accuracy	Initial Test Accuracy
1	0.1121 ± 0.0278	0.1096 ± 0.0246	0.1115 ± 0.0316
10	0.1552 ± 0.0196	0.1558 ± 0.0238	0.1509 ± 0.0203
100	0.1958 ± 0.0145	0.1928 ± 0.0173	0.1920 ± 0.0176
1000	0.2295 ± 0.0169	0.2257 ± 0.0175	0.2292 ± 0.0193

To analyse the impact of the GAN initialisation on the training results we used a random search to define the learning rate schedule and plotted the test accuracy for each training epoch considering the top 5 best learning schedules based on the validation accuracy. Average and standard deviation for different numbers of generated matrices and a baseline using random initialisation can be seen in Figure 64. Results show that even though the GAN generated matrices did provide a better initial condition for the classifier, it did not have a considerable impact on the performance during later epochs.

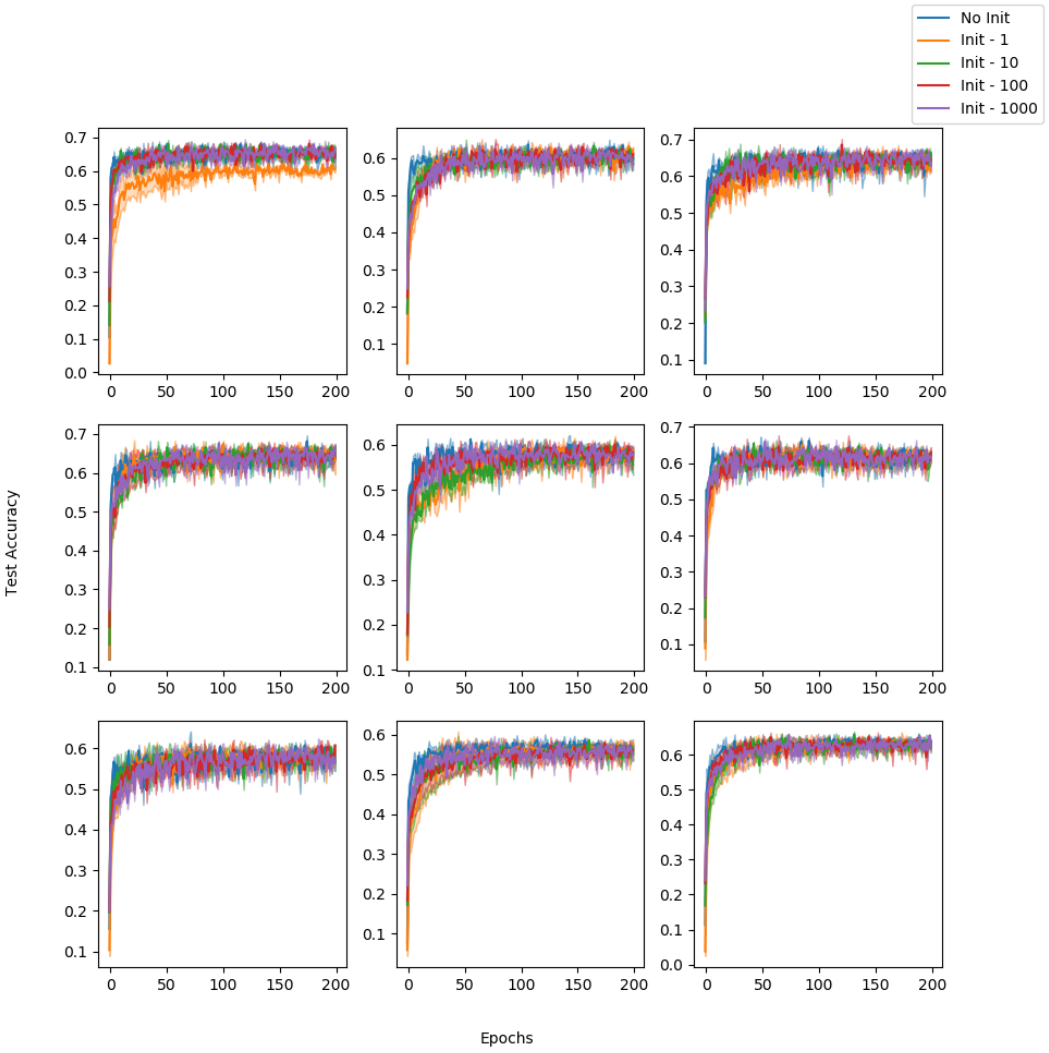


Figure 64 – Average and standard deviation of test accuracy over epochs for nine different subsets of test classes. Learning rate schedules are subjected to random search and the best 5 are selected based on the validation accuracy. Different colours indicate different numbers of initialisation matrices generated by the GAN. The Munkres algorithm is used to find the column order that maximises the training accuracy. When more than one matrix is generated, the one with the best training accuracy is used.

6.5 Concluding remarks

The results achieved in the experiments presented in this chapter show that GANs have a promising application when it comes to knowledge transfer. However, their training instability is still a major challenge that needs to be addressed. We analysed two ways of exploiting GAN models for knowledge transfer: regularisation and weight initialisation. The major challenges encountered during the experiments include: assessing model quality, mode collapse and over-training.

During our experiments, we used different procedures to analyse the results obtained by trained GAN models so that mode collapse and over-training could be detected and the

overall quality of the trained model could be inferred. Even though mode collapse is one of the most commonly tackled problems by new GAN training procedures, these are only explored for images, where it is easier to identify when a mode collapse happens. Over-training a GAN is not a widely explored problem since its impact is mostly seen in the discriminator and most GAN applications focus only on utilising the samples created by the generator. We noticed that over-training the GAN causes the discriminator to be trained on irrelevant information and hindering the quality of the discriminator.

Even though the results are promising, further analysis is required to make the approach more stable and easily applicable to real-world problems. Moreover, we believe that expanding the regularisation to previous layers, as well as the softmax layer, will further improve the impact of the GAN regularisation in the final model and that it will improve, not only the final classification results, but also the representations learnt by the network.

The method and experiments presented in this chapter go towards improving the quality of the representations learnt by representation learning algorithms by using a data-driven regularisation method. This method can also be used to overcome when the amount of data available is not enough to train a representation learning method. The proposed regularisation method could also be used on top of a pre-trained network for fine-tuning. It is important to notice that we chose to explore the proposed method on a synthetic and image datasets due to the complexity of evaluating the impact of the model in the final classifier, which was overcome by addressing simpler and more well-known problems. As future work, we intend to explore different GAN architectures for better training stability and easier model quality inference; and to analyse the impact of the proposed method on spatio-temporal representation learning.

CHAPTER

7

CONCLUSION

Machine learning methods heavily rely on good data descriptors to create high-quality models. However, the development of good representation extraction techniques is considered to be a complex problem that requires careful engineering and domain-specific knowledge. Moreover, most existing feature extraction techniques are task-specific, with out-of-the-box integrated domain-specific knowledge. This fact hinders the performance of most of these techniques or even makes them not suitable for different tasks. *Representation learning* tries to solve this problem by proposing data-driven methods that automatically generate representations that are relevant to a given task. This allows end-to-end machine learning methods to be developed, where the model's input is raw data and its output is the desired output for the given task (e.g. classification score). Furthermore, these techniques take advantage of high amounts of data available to generate better models.

In this document, we explored and analysed multiple representation learning and feature extraction methods developed for videos. The main challenges faced when learning representations from video are: (1) to learn representations that include both spatial and temporal information; (2) to extract fixed-size representations from variable-length (time) and variable-dimension (space) videos; (3) to learn long-term temporal information with reasonable computational cost; (4) to extract representations capable of generalising to different settings, such as changes in velocity and focus changes from local to global information and vice-versa. These difficulties are highlighted in the results presented throughout this document.

In Chapter 4, we proposed a novel spatio-temporal representation learning evaluation pipeline for video processing methods. This pipeline involves analysing the representations created for three versions of the BouncingMNIST dataset and evaluating how the models trained with each version extrapolates their knowledge to the remaining versions. Using this pipeline, we evaluated several state-of-the-art deep learning architectures, for which we showed that representations learnt by using spatial and temporal information on every step of the network are better than the ones learnt by processing spatial and temporal information separately. However,

the methods currently used to do so have difficulty generalising the encoded knowledge when there are changes to the available information, especially temporal.

In Chapter 5, we compared the generalisation capabilities of one of the most commonly used deep learning architectures (C3D) to the state-of-the-art hand-crafted spatio-temporal feature extraction method (IDT-FV) for two different tasks (action recognition and dynamic scene recognition). The results show that, as expected, hand-crafted feature extraction methods perform better at the task they were designed for, probably due to domain-specific knowledge included in the method. However, the representation learning method used was able to generalise significantly better to a task was unknown for both methods. We also explored different ways of combining features extracted from short video clips into a fixed-size video-level representation.

Finally, in Chapter 6, we present a new data-driven regularisation and model initialisation approach based on GANs. This approach takes advantage of trained models for similar problems to improve the representations and overall quality of a new model. A GAN is trained to model the distribution of trained models; the discriminator is then used as a regulariser that guides the new model into the probability distribution, while the generator is used to create better initialisation parameters for the network. The presented results are promising and show that GANs can be used for knowledge transfer both for a synthetic task and for image classification. However, some challenges need to be overcome when training the GAN model, such as training instability, mode collapse and model quality assessment. Also, further investigation is required to understand the impact of this method on spatio-temporal representation learning; this is investigation is made particularly difficult by the challenges mentioned above.

Overall, we noticed that spatio-temporal representation learning from videos is still an open research area, regardless of end-to-end deep networks being responsible for some of the state-of-the-art results on certain video processing tasks, e.g. action recognition. Also, current spatio-temporal representation learning architectures are not able to encode all the information present videos, and there are very few approaches that allow researchers to analyse which are the best architectures for each circumstance. Our results help better comprehend how each of these architectures learns their representations and where research needs to focus to improve on the current methods and offer new tools for this purpose. Furthermore, we present a novel application for generative networks that allow for their use on knowledge transfer, improving representation learning for new models by guiding the learning procedure and providing a better initial condition.

7.1 Future Work

To make further progress in spatio-temporal representation learning, we believe that different ways of including the analysis of temporal information throughout the network need to be proposed since convolutions limit the analysis to a fixed-size window constrained by the

computational cost and do not adapt well to changes in temporal information. Also, methods that provide a better understanding of how spatio-temporal representation learning architectures encode information and their advantages and drawbacks should be explored as they would help guide advances in the area. Lastly, further investigation on the use of generative networks for knowledge transfer should be conducted since the results presented here show promise. These results would largely benefit from more stable GAN training procedures and architectures, and better evaluation assessment techniques for generative networks that do not rely on visual information.

BIBLIOGRAPHY

AMODEI, D.; ANANTHANARAYANAN, S.; ANUBHAI, R.; BAI, J.; BATTENBERG, E.; CASE, C.; CASPER, J.; CATANZARO, B.; CHENG, Q.; CHEN, G. *et al.* Deep speech 2: End-to-end speech recognition in english and mandarin. In: **International Conference on Machine Learning**. [S.l.: s.n.], 2016. p. 173–182. Citation on page 58.

ARJOVSKY, M.; CHINTALA, S.; BOTTOU, L. Wasserstein gan. **arXiv preprint arXiv:1701.07875**, 2017. Citation on page 49.

BACCOUCHE, M.; MAMALET, F.; WOLF, C.; GARCIA, C.; BASKURT, A. Sequential deep learning for human action recognition. In: SPRINGER. **International Workshop on Human Behavior Understanding**. [S.l.], 2011. p. 29–39. Citation on page 60.

BALLAS, N.; YAO, L.; PAL, C.; COURVILLE, A. Delving deeper into convolutional networks for learning video representations. In: **Proceedings of the International Conference on Learning Representations**. [S.l.: s.n.], 2015. Citation on page 61.

BAY, H.; TUYTELAARS, T.; GOOL, L. V. Surf: speeded up robust features. In: **Computer vision – ECCV 2006**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 404–417. Citation on page 54.

BENGIO, Y. Learning deep architectures for AI. **Foundations and Trends in Machine Learning**, v. 2, n. 1, p. 1–127, 2009. Also published as a book. Now Publishers, 2009. Citation on page 40.

BENGIO, Y.; COURVILLE, A.; VINCENT, P. Representation learning: A review and new perspectives. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, IEEE, v. 35, n. 8, p. 1798–1828, 2013. Citations on pages 30, 35, 36, 51, and 57.

BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. **Neural Networks, IEEE Transactions on**, IEEE, v. 5, n. 2, p. 157–166, 1994. Citation on page 45.

BERTHELOT, D.; SCHUMM, T.; METZ, L. Began: Boundary equilibrium generative adversarial networks. **arXiv preprint arXiv:1703.10717**, 2017. Citations on pages 49 and 149.

BISHOP, C. M. **Pattern recognition and machine learning**. [S.l.]: Springer, 2006. Citation on page 36.

BOSER, B. E.; GUYON, I. M.; VAPNIK, V. N. A training algorithm for optimal margin classifiers. In: ACM. **Proceedings of the fifth annual workshop on Computational learning theory**. [S.l.], 1992. p. 144–152. Citation on page 94.

CARREIRA, J.; ZISSERMAN, A. Quo vadis, action recognition? a new model and the kinetics dataset. In: **2017 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.]: IEEE, 2017. p. 4724–4733. Citations on pages 32 and 64.

CHATFIELD, K.; LEMPITSKY, V. S.; VEDALDI, A.; ZISSERMAN, A. The devil is in the details: an evaluation of recent feature encoding methods. In: **Proceedings of the British Machine Vision Conference**. [S.l.]: BMVA Press, 2011. p. 76.1–76.12. Citation on page 55.

CHEN, X.; DUAN, Y.; HOUTHOOFT, R.; SCHULMAN, J.; SUTSKEVER, I.; ABBEEL, P. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2016. p. 2172–2180. Citation on page 149.

CHENG, X.; DALE, C.; LIU, J. Statistics and social network of youtube videos. In: IEEE. **Quality of Service, 2008. IWQoS 2008. 16th International Workshop on**. [S.l.], 2008. p. 229–238. Citation on page 30.

DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: IEEE. **Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on**. [S.l.], 2005. v. 1, p. 886–893. Citations on pages 30, 55, and 142.

DALAL, N.; TRIGGS, B.; SCHMID, C. Human detection using oriented histograms of flow and appearance. In: **Computer Vision – ECCV 2006**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 428–441. Citations on pages 52, 54, 56, and 57.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In: **2009 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2009. Citations on pages 63 and 68.

DENG, L.; YU, D. **Deep Learning: Methods and Applications**. [S.l.], 2014. Citation on page 40.

DERPANIS, K. G.; LECCE, M.; DANIILIDIS, K.; WILDES, R. P. Dynamic scene understanding: The role of orientation features in space and time in scene classification. In: IEEE. **Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on**. [S.l.], 2012. p. 1306–1313. Citations on pages 96 and 99.

DRUZHKOV, P.; KUSTIKOVA, V. A survey of deep learning methods and software tools for image classification and object detection. **Pattern Recognition and Image Analysis**, Springer, v. 26, n. 1, p. 9–15, 2016. Citation on page 58.

EGHBAL-ZADEH, H.; WIDMER, G. Likelihood estimation for generative adversarial networks. **arXiv preprint arXiv:1707.07530**, 2017. Citation on page 147.

FARNEBÄCK, G. Two-frame motion estimation based on polynomial expansion. In: **Image Analysis**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 363–370. Citation on page 53.

FEICHTENHOFER, C.; PINZ, A.; WILDES, R. Spatiotemporal residual networks for video action recognition. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2016. p. 3468–3476. Citation on page 63.

FEICHTENHOFER, C.; PINZ, A.; ZISSERMAN, A. Convolutional two-stream network fusion for video action recognition. In: **2016 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.]: IEEE, 2016. p. 1933–1941. Citation on page 63.

FISCHLER, M. A.; BOLLES, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. **Communications of the ACM**, ACM, New York, NY, USA, v. 24, n. 6, p. 381–395, 1981. Citation on page 54.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citations on pages 13, 30, 35, 39, 40, 41, 42, 43, 44, 45, 46, 47, and 57.

GOODFELLOW, I.; POUGET-ABADIE, J.; MIRZA, M.; XU, B.; WARDE-FARLEY, D.; OZAIR, S.; COURVILLE, A.; BENGIO, Y. Generative adversarial nets. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2014. p. 2672–2680. Citation on page 47.

GRETTON, A.; BORGWARDT, K. M.; RASCH, M. J.; SCHÖLKOPF, B.; SMOLA, A. A kernel two-sample test. **Journal of Machine Learning Research**, v. 13, n. Mar, p. 723–773, 2012. Citation on page 144.

GULRAJANI, I.; AHMED, F.; ARJOVSKY, M.; DUMOULIN, V.; COURVILLE, A. C. Improved training of wasserstein gans. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2017. p. 5769–5779. Citation on page 49.

HARRIS, C.; STEPHENS, M. A combined corner and edge detector. In: **Alvey vision conference**. [S.l.: s.n.], 1988. v. 15, n. 50, p. 10–5244. Citation on page 52.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 770–778. Citations on pages 139 and 144.

HENSELER, J. Back propagation. In: **Artificial neural networks**. [S.l.]: Springer, 1995. p. 37–66. Citations on pages 37 and 39.

HERATH, S.; HARANDI, M.; PORIKLI, F. Going deeper into action recognition: A survey. **Image and vision computing**, Elsevier, v. 60, p. 4–21, 2017. Citation on page 51.

HINTON, G.; DENG, L.; YU, D.; DAHL, G. E.; MOHAMED, A.-r.; JAITLY, N.; SENIOR, A.; VANHOUCKE, V.; NGUYEN, P.; SAINATH, T. N. *et al.* Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. **Signal Processing Magazine, IEEE**, IEEE, v. 29, n. 6, p. 82–97, 2012. Citation on page 30.

HINTON, G.; OSINDERO, S.; TEH, Y.-W. A fast learning algorithm for deep belief nets. **Neural computation**, MIT Press, v. 18, n. 7, p. 1527–1554, 2006. Citation on page 40.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural computation**, MIT Press, v. 9, n. 8, p. 1735–1780, 1997. Citation on page 44.

HUBEL, D. H.; WIESEL, T. N. Receptive fields and functional architecture of monkey striate cortex. **The Journal of physiology**, Wiley Online Library, v. 195, n. 1, p. 215–243, 1968. Citation on page 41.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: **Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37**. [S.l.: s.n.], 2015. (ICML'15), p. 448–456. Citation on page 48.

JI, S.; XU, W.; YANG, M.; YU, K. 3d convolutional neural networks for human action recognition. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 35, n. 1, p. 221–231, 2013. Citation on page 60.

JOLLIFFE, I. **Principal component analysis**. New York, NY, USA: Springer-Verlag New York, 2002. Citation on page 55.

KARPATHY, A. Blog, **The Unreasonable Effectiveness of Recurrent Neural Networks**. 2015. <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>. Citation on page 43.

KARPATHY, A.; JOHNSON, J.; LI, F.-F. Visualizing and understanding recurrent networks. **arXiv preprint arXiv:1506.02078**, 2015. Citations on pages 44 and 45.

KARPATHY, A.; TODERICI, G.; SHETTY, S.; LEUNG, T.; SUKTHANKAR, R.; FEI-FEI, L. Large-scale video classification with convolutional neural networks. In: **2014 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.]: IEEE, 2014. p. 1725–1732. Citations on pages 30, 52, 58, 59, 60, and 97.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **Proceedings of the 3rd International Conference on Learning Representations (ICLR)**, 2014. Citation on page 70.

KLASER, A.; MARSZAŁEK, M.; SCHMID, C. A spatio-temporal descriptor based on 3d-gradients. In: BRITISH MACHINE VISION ASSOCIATION. **BMVC 2008 – 19th British Machine Vision Conference**. [S.l.], 2008. p. 275–1. Citations on pages 30 and 52.

KRIESEL, D. A brief introduction to neural networks. **Retrieved August**, v. 15, p. 2011, 2007. Citation on page 37.

KRIZHEVSKY, A.; HINTON, G. Learning multiple layers of features from tiny images. **Master's thesis, Department of Computer Science, University of Toronto**, Citeseer, 2009. Citation on page 143.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F.; BURGES, C. J. C.; BOTTOU, L.; WEINBERGER, K. Q. (Ed.). **Advances in Neural Information Processing Systems 25**. [S.l.]: Curran Associates, Inc., 2012. p. 1097–1105. Citation on page 30.

KRÖSE, B.; KROSE, B.; SMAGT, P. van der; SMAGT, P. An introduction to neural networks. 1996. Citation on page 37.

LAKE, B. M.; SALAKHUTDINOV, R.; TENENBAUM, J. B. Human-level concept learning through probabilistic program induction. **Science**, American Association for the Advancement of Science, v. 350, n. 6266, p. 1332–1338, 2015. Citation on page 142.

LÄNGKVIST, M.; KARLSSON, L.; LOUTFI, A. A review of unsupervised feature learning and deep learning for time-series modeling. **Pattern Recognition Letters**, Elsevier, v. 42, p. 11–24, 2014. Citations on pages 30, 58, and 96.

LAPTEV; LINDEBERG. Space-time interest points. In: **Proceedings Ninth IEEE International Conference on Computer Vision**. [S.l.: s.n.], 2003. p. 432–439 vol.1. Citation on page 52.

LARSEN, A. B. L.; SØNDERBY, S. K.; LAROCHELLE, H.; WINTHER, O. Autoencoding beyond pixels using a learned similarity metric. **arXiv preprint arXiv:1512.09300**, 2015. Citation on page 149.

LAZEBNIK, S.; SCHMID, C.; PONCE, J. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In: **2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition**. [S.l.]: IEEE, 2006. p. 2169–2178. Citation on page 55.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015. Citations on pages 29, 30, 40, and 57.

LECUN, Y.; KAVUKCUOGLU, K.; FARABET, C. Convolutional networks and applications in vision. In: IEEE. **Proceedings of 2010 IEEE International Symposium on Circuits and Systems**. [S.l.], 2010. p. 253–256. Citation on page 42.

LITJENS, G.; KOOI, T.; BEJNORDI, B. E.; SETIO, A. A. A.; CIOMPI, F.; GHAFOORIAN, M.; LAAK, J. A. V. D.; GINNEKEN, B. V.; SÁNCHEZ, C. I. A survey on deep learning in medical image analysis. **Medical image analysis**, Elsevier, v. 42, p. 60–88, 2017. Citation on page 58.

LOWE, D. G. Object recognition from local scale-invariant features. In: IEEE. **Computer vision, 1999. The proceedings of the seventh IEEE international conference on**. [S.l.], 1999. v. 2, p. 1150–1157. Citation on page 30.

____. Distinctive image features from scale-invariant keypoints. **International journal of computer vision**, Springer, v. 60, n. 2, p. 91–110, 2004. Citations on pages 52, 56, and 57.

MAAS, A. L.; HANNUN, A. Y.; NG, A. Y. Rectifier nonlinearities improve neural network acoustic models. In: **Proc. icml**. [S.l.: s.n.], 2013. v. 30, n. 1, p. 3. Citation on page 49.

MAATEN, L. Van der; HINTON, G. Visualizing high-dimensional data using t-sne. **Journal of Machine Learning Research**, v. 9, n. 2579-2605, p. 85, 2008. Citation on page 69.

MACQUEEN, J. Some methods for classification and analysis of multivariate observations. In: **Proceedings of the 5th Berkeley symposium on mathematical statistics and probability**. Berkeley, CA, USA: University of California Press, 1967. v. 1, n. 14, p. 281–297. Citation on page 55.

MARSZAŁEK, M.; LAPTEV, I.; SCHMID, C. Actions in context. In: **2009 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2009. Citations on pages 54, 96, and 111.

MESSING, R.; PAL, C.; KAUTZ, H. Activity recognition using the velocity histories of tracked keypoints. In: **2009 IEEE 12th International Conference on Computer Vision**. [S.l.]: IEEE, 2009. p. 104–111. Citation on page 52.

MINSKY, M.; SEYMOUR, P. Perceptrons. MIT press, 1969. Citation on page 38.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. A.; VENESS, J.; BELLEMARE, M. G.; GRAVES, A.; RIEDMILLER, M.; FIDJELAND, A. K.; OSTROVSKI, G.; PETERSEN, S.; BEATTIE, C.; SADIK, A.; ANTONOGLOU, I.; KING, H.; KUMARAN, D.; WIERSTRA, D.; LEGG, S.; HASSABIS, D. Human-level control through deep reinforcement learning.

Nature, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved., v. 518, n. 7540, p. 529–533, 2015. ISSN 0028-0836. Citation on page 30.

MONTES, A.; SALVADOR, A.; PASCUAL, S.; NIETO, X. Giro-i. Temporal activity detection in untrimmed videos with recurrent neural networks. In: **1st NIPS Workshop on Large Scale Computer Vision Systems**. [S.l.: s.n.], 2016. Citations on pages 58 and 62.

MUNKRES, J. Algorithms for the assignment and transportation problems. **Journal of the society for industrial and applied mathematics**, SIAM, v. 5, n. 1, p. 32–38, 1957. Citation on page 154.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: **Proceedings of the 27th International Conference on International Conference on Machine Learning**. [S.l.: s.n.], 2010. (ICML'10), p. 807–814. ISBN 978-1-60558-907-7. Citation on page 49.

NG, J. Y.-H.; HAUSKNECHT, M.; VIJAYANARASIMHAN, S.; VINYALS, O.; MONGA, R.; TODERICI, G. Beyond short snippets: Deep networks for video classification. In: **2015 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.]: IEEE, 2015. p. 4694–4702. Citations on pages 31, 58, 59, 61, and 63.

NIELSEN, M. A. Neural networks and deep learning. Determination Press, 2015. Accessed: 2015-03-13. Available: <http://neuralnetworksanddeeplearning.com/>. Citation on page 40.

NOWAK, E.; JURIE, F.; TRIGGS, B. Sampling strategies for bag-of-features image classification. In: **Computer Vision – ECCV 2006**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 490–503. Citation on page 55.

ONEATA, D.; VERBEEK, J.; SCHMID, C. Action and event recognition with fisher vectors on a compact feature set. In: **2013 IEEE International Conference on Computer Vision**. [S.l.]: IEEE, 2013. p. 1817–1824. Citations on pages 52, 55, 92, 93, 129, and 130.

PASCANU, R.; MIKOLOV, T.; BENGIO, Y. On the difficulty of training recurrent neural networks. **arXiv preprint arXiv:1211.5063**, 2012. Citation on page 45.

POPOOLA, O. P.; WANG, K. Video-based abnormal human behavior recognition—a review. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, IEEE, v. 42, n. 6, p. 865–878, 2012. Citation on page 51.

PROTASOV, S.; KHAN, A. M.; SOZYKIN, K.; AHMAD, M. Using deep features for video scene detection and annotation. **Signal, Image and Video Processing**, Springer, p. 1–9, 2018. Citation on page 51.

QIU, Z.; YAO, T.; MEI, T. Learning spatio-temporal representation with pseudo-3d residual networks. In: **2017 IEEE International Conference on Computer Vision**. [S.l.]: IEEE, 2017. p. 5534–5542. Citations on pages 13, 29, 60, and 61.

RADFORD, A.; METZ, L.; CHINTALA, S. Unsupervised representation learning with deep convolutional generative adversarial networks. In: **Proceedings of the International Conference on Learning Representations (ICLR)**. [S.l.: s.n.], 2016. Citation on page 48.

RIFKIN, R.; KLAUTAU, A. In defense of one-vs-all classification. **The Journal of Machine Learning Research**, JMLR. org, v. 5, p. 101–141, 2004. Citation on page 94.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological review**, American Psychological Association, v. 65, n. 6, p. 386, 1958. Citation on page 37.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Cognitive modeling**, v. 5, n. 3, p. 1, 1988. Citations on pages 39 and 40.

SABOKROU, M.; FAYYAZ, M.; FATHY, M.; MOAYED, Z.; KLETTE, R. Deep-anomaly: Fully convolutional neural network for fast anomaly detection in crowded scenes. **Computer Vision and Image Understanding**, Elsevier, 2018. Citation on page 51.

SÁNCHEZ, J.; PERRONNIN, F.; MENSINK, T.; VERBEEK, J. Image classification with the fisher vector: theory and practice. **International journal of computer vision**, Springer, v. 105, n. 3, p. 222–245, 2013. Citations on pages 55 and 92.

SANDE, K. V. D.; GEVERS, T.; SNOEK, C. Evaluating color descriptors for object and scene recognition. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 32, n. 9, p. 1582–1596, 2010. Citation on page 51.

SANDLER, M.; HOWARD, A.; ZHU, M.; ZHMOGINOV, A.; CHEN, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2018. p. 4510–4520. Citations on pages 13, 68, 69, and 77.

SCHULDT, C.; LAPTEV, I.; CAPUTO, B. Recognizing human actions: a local svm approach. In: IEEE. **Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on**. [S.l.], 2004. v. 3, p. 32–36. Citations on pages 96, 102, 104, 105, 106, 108, and 109.

SCOVANNER, P.; ALI, S.; SHAH, M. A 3-dimensional sift descriptor and its application to action recognition. In: ACM. **Proceedings of the 15th ACM international conference on Multimedia**. [S.l.], 2007. p. 357–360. Citations on pages 30 and 52.

SHI, J.; TOMASI, C. Good features to track. In: **1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.]: IEEE, 1994. p. 593–600. Citation on page 54.

SHOU, Z.; WANG, D.; CHANG, S.-F. Temporal action localization in untrimmed videos via multi-stage cnns. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2016. p. 1049–1058. Citations on pages 29 and 51.

SHROFF, N.; TURAGA, P.; CHELLAPPA, R. Moving vistas: Exploiting motion for describing scenes. In: IEEE. **Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on**. [S.l.], 2010. p. 1911–1918. Citations on pages 98 and 99.

SIMONYAN, K.; ZISSERMAN, A. Two-stream convolutional networks for action recognition in videos. In: **Advances in Neural Information Processing Systems 27**. Cambridge, MA, USA: MIT Press, 2014. p. 568–576. Citations on pages 62 and 63.

SIVIC, J.; ZISSERMAN, A. Video google: A text retrieval approach to object matching in videos. In: **IEEE 9th International Conference on Computer Vision**. [S.l.]: IEEE, 2003. p. 1470–1477. Citation on page 55.

SPRINGENBERG, J. T.; DOSOVITSKIY, A.; BROX, T.; RIEDMILLER, M. Striving for simplicity: The all convolutional net. In: **arXiv:1412.6806, also appeared at ICLR 2015 Workshop Track**. [S.l.: s.n.], 2015. Citation on page 48.

SRIVASTAVA, A.; VALKOZ, L.; RUSSELL, C.; GUTMANN, M. U.; SUTTON, C. Veegan: Reducing mode collapse in gans using implicit variational learning. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2017. p. 3310–3320. Citation on page 149.

SRIVASTAVA, N.; MANSIMOV, E.; SALAKHUDINOV, R. Unsupervised learning of video representations using lstms. In: **Proceedings of the 32nd International Conference on Machine Learning**. [S.l.]: PMLR, 2015. p. 843–852. Citations on pages 13, 31, 61, 62, and 68.

SUN, J.; WU, X.; YAN, S.; CHEONG, L.-F.; CHUA, T.-S.; LI, J. Hierarchical spatio-temporal context modeling for action recognition. In: **2009 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.]: IEEE, 2009. p. 2004–2011. Citation on page 52.

SUTSKEVER, I.; MARTENS, J.; HINTON, G. E. Generating text with recurrent neural networks. In: **Proceedings of the 28th International Conference on Machine Learning (ICML-11)**. [S.l.: s.n.], 2011. p. 1017–1024. Citation on page 44.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. **arXiv preprint arXiv:1409.4842**, 2014. Citation on page 30.

SZELISKI, R. Image alignment and stitching: a tutorial. **Foundations and Trends® in Computer Graphics and Vision**, Now Publishers Inc., v. 2, n. 1, p. 1–104, 2006. Citation on page 54.

TRAN, D.; BOURDEV, L.; FERGUS, R.; TORRESANI, L.; PALURI, M. Learning spatiotemporal features with 3d convolutional networks. In: **2015 IEEE International Conference on Computer Vision**. [S.l.]: IEEE, 2015. p. 4489–4497. Citations on pages 31, 60, 68, 92, and 93.

TRAN, D.; WANG, H.; TORRESANI, L.; RAY, J.; LECUN, Y.; PALURI, M. A closer look at spatiotemporal convolutions for action recognition. In: **2018 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2018. p. 6450–6459. Citations on pages 29, 58, 60, 61, and 68.

VAROL, G.; LAPTEV, I.; SCHMID, C. Long-term temporal convolutions for action recognition. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 40, n. 6, p. 1510–1517, 2018. Citation on page 60.

WANG, H.; KLÄSER, A.; SCHMID, C.; LIU, C.-L. Action recognition by dense trajectories. In: **2011 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.]: IEEE, 2011. p. 3169–3176. Citations on pages 13, 52, 53, and 54.

____. Dense trajectories and motion boundary descriptors for action recognition. **International journal of computer vision**, Springer, v. 103, n. 1, p. 60–79, 2013. Citations on pages 57 and 129.

WANG, H.; SCHMID, C. Action recognition with improved trajectories. In: **2013 IEEE International Conference on Computer Vision**. [S.l.]: IEEE, 2013. p. 3551–3558. Citations on pages 30, 52, and 54.

WANG, H.; ULLAH, M. M.; KLASER, A.; LAPTEV, I.; SCHMID, C. Evaluation of local spatio-temporal features for action recognition. In: **Proceedings of the British Machine Vision Conference**. [S.l.]: BMVA Press, 2009. p. 124.1–124.11. Citation on page 52.

WANG, L.; XIONG, Y.; WANG, Z.; QIAO, Y.; LIN, D.; TANG, X.; GOOL, L. V. Temporal segment networks: Towards good practices for deep action recognition. In: **Computer Vision – ECCV 2016**. Cham: Springer International Publishing, 2016. p. 20–36. Citation on page 63.

WANG, Q. X. A survey on bayesian learning model for human action recognition. In: IOP PUBLISHING. **Journal of Physics: Conference Series**. [S.l.], 2018. v. 1087, n. 6, p. 062011. Citation on page 51.

WEINZAEPFEL, P.; HARCHAOUI, Z.; SCHMID, C. Learning to track for spatio-temporal action localization. In: **Proceedings of the IEEE international conference on computer vision**. [S.l.: s.n.], 2015. p. 3164–3172. Citation on page 51.

WERBOS, P. Beyond regression: New tools for prediction and analysis in the behavioral sciences. 1974. Citations on pages 39 and 43.

WIDROW, B.; HOFF, M. E. *et al.* Adaptive switching circuits. Defense Technical Information Center, 1960. Citation on page 37.

XU, B.; WANG, N.; CHEN, T.; LI, M. Empirical evaluation of rectified activations in convolutional network. **arXiv preprint arXiv:1505.00853**, 2015. Citation on page 49.

XU, D.; YAN, Y.; RICCI, E.; SEBE, N. Detecting anomalous events in videos by learning deep representations of appearance and motion. **Computer Vision and Image Understanding**, Elsevier, v. 156, p. 117–127, 2017. Citation on page 29.

YOUNG, T.; HAZARIKA, D.; PORIA, S.; CAMBRIA, E. Recent trends in deep learning based natural language processing. **IEEE Computational Intelligence Magazine**, IEEE, v. 13, n. 3, p. 55–75, 2018. Citation on page 58.

ZHAO, J.; MATHIEU, M.; LECUN, Y. Energy-based generative adversarial network. In: **Proceedings of the International Conference on Learning Representations (ICLR)**. [S.l.: s.n.], 2016. Citation on page 49.

ZHU, W.; HU, J.; SUN, G.; CAO, X.; QIAO, Y. A key volume mining deep framework for action recognition. In: **2016 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.]: IEEE, 2016. p. 1991–1999. Citation on page 64.


ICMC
USP
SÃO CARLOS