Augmentation of Two-stream CNN architectures with context and attention for action detection and recognition

Pedro Diogo Fernandes de Abreu

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Alexandre José Malheiro Bernardino

Examination Committee

November 2018

Acknowledgments

I would like to thank my mom and my dad for their continued support through all these years. To my grandparents, uncle, aunt and cousins for countless good memories and for fostering in me an interest in science.

I would like to give a special dedication to my friend and colleague João Antunes for the ideas concerning the approach in this dissertation and for his invaluable contribution to the work carried out. I would also like to thank Rui Figueiredo for providing one of the methods used in the dissertation and for his willingness to help us use it.

I would like to thank my dissertation supervisor Prof. Alexandre Bernardino for his guidance, valuable suggestions and a large amount of patience to answer all my questions.

I would like to thank all the people I have ever met, friends or not for all the lessons they have ever taught me.

Lastly, I would like to thank the SPARSIS project, under which I did a research on Sparse Modeling and Estimation of Motion Fields, with the reference PTDC/EEIPRO/0426/2014 with the support of FCT (Fundação para a Ciência e Tecnologia).

Abstract

Tasks such as action detection and recognition are a promising step in several areas such as retail, security, robotics and recommendation systems. Recently, challenging datasets have been introduced, which are representative of the task of multi-person spatiotemporal action detection and recognition task with multi-labels. We propose to augment the statof-the-art two-stream CNN architectures for this task. These architectures are limited in that they try to detect the actions independent of the background and other humans in the same video. To this end, three novel contributions are presented: attention filtering, context streams and a combination of both. For attention filtering, with the goal of not only extracting information from a target but from the image background, we train two-stream CNN architectures with different kinds of filters applied on RGB and Optical Flow inputs. For context streams, with the goal of predicting the labels of a target using the labels of the surrounding neighbours, we use dataset labels to explicitly encode the relationship of classes performed by multiple humans as context features and train LSTM networks on these features. Finally, we combine these methods by fusing the context streams with the two-stream approaches trained with attention filtering. Results show the combination of the first two methods outperforms each of them and that all augmentations improve on a two-stream CNN baseline.

Keywords

Action Detection, Action Recognition, Multi label datasets, Attention filters, Spatiotemporal context label relationships, Two-stream Convolutional Neural Networks

Resumo

Tarefas como reconhecimento de acções são um passo promissor em várias áreas como vendas, robótica, classificação de videos e sistemas de recomendação. Recentemente, foram apresentados datasets dificeis que são representativos da tarefa de detecção e reconhecimento de acções multi-pessoa e multi-label. Propomos melhorar as arquiteturas two-stream CNN estado-da-arte para esta tarefa. Estas arquiteturas estão limitadas no facto em que tentam detectar acções independentemente do background e de outras pessoas no mesmo video. Com este fim, três novas contribuições são apresentadas: filtros de atenção, streams de contexto e uma combinação de ambos. Para os filtros de atenção, com o objectivo de não só extrair informação de um target mas também do background, treinamos arquiteturas two-stream CNN com diferentes tipos de filtros aplicados nos inputs RGB e Optical Flow. Para as streams de contexto, com o objectivo de prever as labels de um target usando as labels dos seus vizinhos, usamos as labels do dataset para codificar explicitamente a relação entre classes executadas por multiplas pessoas como features de contexto e treinamos redes LSTM nestas features. Finalmente, combinamos estes métodos através da fusão das streams de contexto com as arquiteturas two-stream treinadas com filtros de atenção. Os resultados mostram que a combinação dos primeiros dois métodos supera a performance de cada um e todos os melhoramentos superam a baseline.

Palavras Chave

Detecção de Acções, Reconhecimento de Acções, Datasets Multi-label, Filtros de atenção, Relações espatiotemporais, Redes Neuronais Convolucionais Two-stream

Contents

List of Figures

List of Tables

Acronyms

BB	Bounding Box
CNN	Convolutional Neural Network
IoU	Intersection over Union
mAP	mean Average Precision
RPN	Region Proposal Network
OF	Optical Flow
FC	Fully-Connected
NHU	Number of Hidden Units
MLP	Multi-Layer Perceptron
RNN	RNNRecurrent Neural Network
LSTM	LSTMLong-short term memory Temporal Segment Network
R-CNN	R-CNNRegion-Convolutional Neural Network
MLD	Multi-label dataset
SLD	Single-label dataset
BPTT	Backpropagation Through Time
SVM	Support Vector Machines
RF	Random Forests
BB	Bounding Box
AP	Average Precision
iDT	improved Dense Trajectories
FPS	Frames Per Second
GBB	Gaussian Background Blur

1 Introduction

Contents

1.1 Motivation

Recently video-based human action detection and recognition has been getting more attention since it allows for the detection of a larger range of actions and incorporation of contextual information than inertial sensors [72]. As a consequence video-based human action detection and recognition has many possible applications in retail, security [91], robotics and as a representative task for video understanding (i.e to understand human video dynamics): given a video, understand what is being shown, what are the elements being seen (visually and structurally), and how they relate to each other.

Although object recognition currently gets most of the research spotlight, partly because it is easier to define and understand, a large commercial payoff would come from accurate human action recognition and detection: with the amount of videos online increasing everyday (as demonstrated by the Youtube-8M dataset [2]), a major drawback is the need to analyze video either for improving suggestions or to check for undesired content by hand.

1.2 Concepts

In this section, we introduce fundamental concepts for the problems we will further discuss. We separate this section in two parts: in the first one we discuss basic concepts involving atomic, action and activities and for the second one, we discuss the concepts necessary to understand the problem we want to solve, namely action recognition opposed to action detection and multi-class opposed to multi-label tasks.

1.2.1 Atomic Actions, Actions and Activities

While most literature uses a somewhat fluid notion of many of these concepts, it is important to separate the notions of action, activity and atomic action for humans. These concepts are related hierarchically and it has been noted by [76] that human behaviour in general can be decomposed in a hierarchy. Furthermore, [9] [94] show how actions can be decomposed into and grouped according to an action structure that can be learned only from data analysis which supports this claim. However, since that is not our goal we stick to the three concept hierarchy of atomic action, action and activity.

Firstly, [51] define action as motion created by the human body, which may or may not be cyclic. According to [62] an **atomic action** is a movement or performance that can be described more at a limb level (e.g *leg forward*) normally with a verb, an **action** is a sequence of atomic actions, and an **activity** is a whole body movement containing a number of subsequent actions. These definitions are the ones we will stick to. The idea of using verbs for actions is discussed by [97] which introduce verb-only labels as a means to reduce the ambiguity in many class labels when creating datasets in this field. Normally verb-only labels tend to be associated with atomic actions. Recent datasets, such as Charades/Holywood In Homes [78], Moments in Time[63], AVA [35] (the one used for our experiments) have as one of its goals working towards atomic actions with that same intention of reducing ambiguity, while older datasets such as UCF101 [81], HMDB [57], Sports-1M [52], tend to focus on broader activities. Note that while we are mostly concerned with atomic actions (i.e very elementary or "fine-grain" actions) in this dissertation we also want to understand how they interact in more complex patterns.

This idea of actions as sequences and an activity as a sequence of actions is also introduced by [16] which separates human activities into four levels: gestures, actions, interactions and group activities, where an action is a sequence of human body gestures or movements and may involve several body parts, and interactions/group activities are just activities with a meaning.

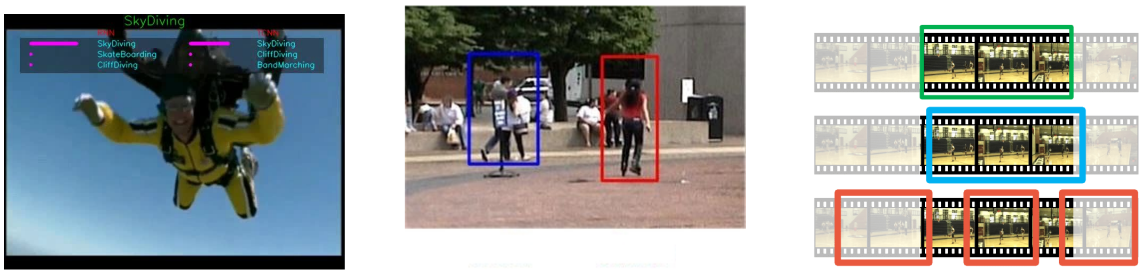
Similarly to the notion of actions as sequences, which we will discuss further in section 3.4.1 of this dissertation, the idea of using meaning to categorize actions is also popular, indeed [40] define action as the most elementary human movement that surrounds interaction with meaning. The meaning associated with this interaction is called the *category* of the action. This allows for the separation of actions into conceptual groups, thus reducing ambiguity when labeling datasets.

1.2.2 Action Recognition/Action Detection, Multi-Class/Multi-Label Tasks

Our task is a supervised learning task. From a computer vision standpoint this means that the objective is to match an observation (video) with one or more labels of actions, by training a model with a fixed amount of samples with known labels. It is important to distinguish between recognition and detection to avoid misunderstandings. Action recognition (classification is a synonym) [51] can be defined as the task of categorizing an action in a video clip to one or more of the pre-defined set of actions. On the other hand, action detection [51] (localization/segmentation is a synonym) is the task of correctly specifying via a boundary (e.g a bounding box) where an action is located throughout the video. This means that action detection can be both interpreted as spatial and temporal.

Note that this also means there may be simultaneous actions happening for different people as shown in Fig. 1.1. An easy way to understand this categorization is to ask a key question. For Action Recognition (What?), for Action Spatial Detection/Localization (Where?) and for Action Temporal Detection/Localization (When?). The main difference is that detection implies that the aforementioned boundary is a target: whether spatial, temporal or both, while action recognition only requires a class label to be attributed.

The problem of recognition is normally non-binary (in that case it is often called actionness [106]), that is, it is not simply that of determining if there is an action occurring. Additionally, the provided videos can be untrimmed without a fixed duration or already trimmed videos of a fixed duration where it is known that at least one action occurs in that duration [24].



(a) Example action recognition

(b) Example action spatial detection

(c) Example action temporal detection

Figure 1.1: Real examples illustrating each type of task. Adapted from `https://ghassanalregib.com/demos/`, `http://www.bu.edu/ids/research-projects/action-recognition/` and `https://www.researchgate.net/figure/An-example-of-temporal-action-detection-The-green-box-denotes-an-instance-of-the_fig1_314355215` respectively.

Another pertinent question is whether or not a single-person can have multiple labels, that is if a single person can perform one or more actions at the same time. This distinguishes between Single-label dataset (SLD)/Multi-class and Multi-label dataset (MLD) [82], and the tasks must adapt accordingly. In Fig. 1.2 we can see a conceptual example of this distinction.

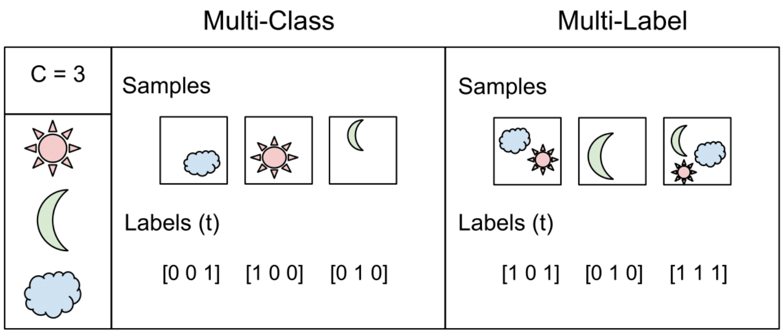


Figure 1.2: A conceptual example with only 3 classes of the distinction between multi-class and multi-label problems. The AVA [35] dataset is multi-label. Adapted from `https://gombru.github.io/2018/05/23/cross_entropy_loss/`

This is relevant as we can now categorize the task we want to tackle as an multi-label spatio-temporal action detection and recognition task. We expect most of the datasets for this task to be class imbalanced, which means that some classes have many more training and testing samples than other.

1.3 Objectives

Since we believe many recent datasets do not reflect the real complexity of this task [35] [13], our main goal in this work is to go beyond the common challenges of action recognition and tackle a multi-label multi-person action detection and recognition task in an imbalanced dataset where we can exploit a rich action structure, as previously described.

Since this type of task is very recent there were few implementations against which we could compare results. The task is similar enough to other tasks, such as action recognition that we could base our approaches on pre-existing ones. However, most of these action recognition models are large and the trend is for larger datasets, it becomes unfeasible for single individuals or small teams to approach these tasks. This was made more difficult due to the amount of computational resources we had available. As such we opted to try alternative approaches that could extend already existing state of the art approaches but always with computational feasibility in mind.

The first of these alternative approaches was that, given an actor bounding box, we want to use pre-processing to filter the image inputs with several possible attention filters, so classifiers can learn relevant features. This was inspired by the fact that, since we approach an action detection task, we had to first extract bounding boxes annotations. This means that we could use that explicit information to aid classifiers by filtering their input.

The second of these approaches is to encode relationships between the actions being performed by humans across time as features. We believe using these explicit features could aid a classifier. This approach is inspired by the fact that, as we previously discussed, actions can be divided into categories. Our main focus was the category of actions that implied some sort of contextual relationship between humans (e.g *give* and *take*, or *fight* and *martial art*). The fact that the task is a multi-label multi-person task should make these contextual relationships much more rich.

That being said, we focused on how to exploit the interactions between the actions being performed by humans across a time window. Namely we would like to predict the labels of a given bounding box (i.e a human) given the labels of their neighbours across time. We named this process as trying to encode *context* in some sort of features that could be used to train a model. This type of encoding and exploiting class relationships has been used in the literature [49], but to our knowledge not for this type of task.

mAP@0.5IoU	UCF101-24 (2013)	JHMDB (2013)	AVA (2018)
Action Recognition	76.3%	76.7%	15.6%
Actor Detection	84.8%	92.8%	75.3%
Gap	8.5%	16.1%	59.7%

Table 1.1: Ablation study from [35] showing the difficulty of action recognition vs actor detection in several datasets.

We want to mention how comparatively difficult the task is when compared to image detection tasks like pedestrian/human detection (named *actor* for action tasks). Note how challenging it is as shown in Table 1.1 and how the trend has been that in more recent datasets the gap between these tasks has been increasing, meaning that action recognition is much harder than actor detection (i.e extracting correct bounding boxes, without classifying them). The results obtained on these datasets show how the problems action recognition and by consequence action detection are still far from being solved and all previous results on smaller and balanced datasets were too optimistic [53]. This means that by approaching this task on realistic datasets the research community can make further advances in the field in a fashion similar to how early Deep Learning [34] research used the ImageNet [75] challenge dataset to advance image recognition [36]. It is also important to note that, due to what was previously discussed, our main focus lies in action recognition rather than the detection component of extracting BBs.

1.4 Challenges

Despite the success of Deep Learning [34] architectures in image classification [75], action recognition has been a tougher problem. The task addressed in this dissertation is even tougher due to the required spatiotemporal localization and multi-label, imbalanced data. The main challenge of multi-label problems are that the number of possible combinations of valid labels is much higher which increases ambiguity. We discuss the challenges of imbalanced data further in section 2.2.5. Regarding action recognition there are several challenges that explain why the problem is so hard:

- *Computational Cost*. Many of the state-of-the-art approaches are computationally heavy [13]. As such, most models tend to have a large number of parameters and take a long time to train, making overfitting likely [88] and architecture optimization difficult. Several ways to reduce this cost exist. For example, [96] use neural networks that receive only single frames or small volumes of frames as input and use a voting scheme to agglomerate predictions across frames of video segments.

- *Long temporal context*. Since the notion of action itself implies a dynamic process occurring over time [56] it is natural for models to require the capacity to process long frame sequences. As such, the large required temporal receptive field has two main implications. First, it enhances the ambiguity between classes as there can be large variations in mode of action performance due to anthropometric difference in activities like gait or physical shape contributing to larger intra class variation and lower inter-class variation[70]. Second, designing architectures that can capture spatiotemporal information that may be cyclic or not involve multiple design options which are non-trivial to evaluate, for example where to temporally segment long actions.

- *Lower quality of data*. While it is easier nowadays to obtain high quality image data for classification, large amounts of high quality video data for action recognition are still hard to obtain. As such most datasets, like HMDB51 [57], AVA [35] and more, resort to using data from movies. These videos tend to have low resolution, occlusions, motion blur and camera-related issues such as sudden camera angle changes or the movement of the camera being interpreted as motion of the action [96]. Note that this is also related to the first challenge: video datasets tend to be much larger than image datasets, and often use compression and re-scaling methods to compensate for this.

1.5 Overview of approaches

We can separate approaches in two categories: before CNNs and after CNNs. Before CNNs, most pipelines were heavily focused on feature representation and feature encoding. These are often called *handcrafted* since the features were often computed via an interpretable, analytical method. These approaches, of which improved Dense Trajectories (iDT)[93] is a good example, can be broken down into the following 3 steps:

1. High-dimensional visual features that describe a region of the video are extracted and linked across time.

2. The extracted features get encoded into a fixed-sized video description via BoVW [18], LLC [54], FV [69][74] or VLAD [50].

3. A classifier, often Support Vector Machines (SVM) or Random Forests (RF) is trained on the encodings.

After the results of CNNs in the domain of image recognition [75], it was believed that the advances in performance would also transition into action recognition [36]. However, many handcrafted methods still outperformed CNN-based methods regardless of the growing interest in the use of convolutions for action recognition over temporal stacks [47]. Many network-based approaches [87] [13] still use an averaging with iDT predictions in order to improve performance. Early network-based approaches [52] tried using single 2D CNNs and their results were worse than traditional methods for two reasons: the learned features did not capture either motion and long temporal context and the UCF101 [81] dataset was not large or varied enough for the networks to learn such features.

After data-driven approaches became more powerful the use of CNN architectures in action recognition quickly became the state of the art [87]. More modern action recognition approaches normally differ in how they employ temporal information [13] and can be separated in categories according to this as shown in Fig. 1.3.

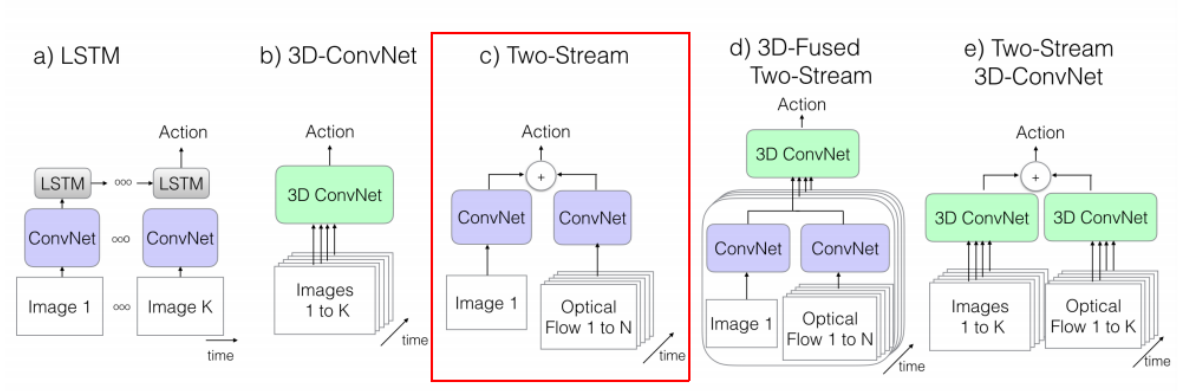


Figure 1.3: Examples of typical action recognition approaches with CNNs as shown in [13]. The rightmost two approaches tend to be computationally heavy while the leftmost two tend to lack explicit motion features like Optical Flow.

It should be noted that the rise of CNN approaches brought yet another design choice: there is also the option of end-to-end training vs feature extraction and classification. End-to-end training often involves more complex models that need fine-tuning and are prone to issues such as overfitting. On the other hand, feature extraction and classification involves a 2-step procedure that may be inefficient to train and test. We explain roughly the idea of the approaches in Fig. 1.3:

- LSTM (e.g Beyond Short Snippets [65], LRCN [20]). The main idea of all these approaches is to use LSTMs to model sequences [5] of trained features maps extracted with CNNs to capture long temporal information. Recently, SSN [66] have shown good results on long untrimmed videos and these approaches still have the largest temporal receptive fields of all approaches. Due to this reason we also used them in our implementation of the methods in this thesis but without CNN features.

- 3D-ConvNet (e.g C3D [87] [100]) These types of approaches built on earlier single-stream 2D approaches [52], but instead used 3D convolutions on video volumes. Two interesting parts of this work is how they intended to use their large models as feature extractors on other models and how they interpreted the features extracted, reaching the conclusion that their networks were focusing on spatial appearance on the first few frames and then constructing motion features in subsequent frames.

- Two-Stream CNN/ConvNet (e.g Two-Stream [80], Two-StreamFusion [29], TSN [96]). The most novel addition introduced by this type of network model is the use of an additional 2D CNN trained on explicit motion features such as optical flow to try and capture short temporal context. Many aspects of this original approach have been further augmented (e.g ActionVLAD [32], HiddenTwoStream [107]) and they are particularly attractive due to the possibility of achieving competitive results while using small amounts of computational resources.

- 3D Fused Two-Stream, Two-Stream 3D-ConvNet (e.g I3D [13], S3D [99], R(2+1)D[89]). While these approaches are currently the state-of-the-art, they extend two-stream approaches by performing 3D convolutions on RGB and Optical Flow stacks. As such, these approaches tend to be more computationally heavy than all others.

Two main lessons learned when looking at these approaches is that discarding optical flow or any alternative motion features seem to be harmful to performance and so is not allowing the networks to learn temporal features. Throughout the rest of this dissertation we will focus mostly feature extraction followed by two-stream CNN classification.

1.6 Contributions

We believe our work is a valid contribution to the specific problem of multi-label imbalanced action detection and recognition. We would like to separate our contributions into conceptual contributions and technical contributions:

- Regarding conceptual contributions, we try to bridge the aforementioned gap by introducing a novel architecture that builds on previous two-stream approaches [80][28][96] to deal with this type of task. Additionally, our approach also extends two-stream approaches by using additional mechanisms such as:
  - Several attention pre-processing filters for the inputs of the two-stream CNNs, both RGB and Optical Flow. We provide individual CNN models trained with each of these attention filters.
  - Encoding human action interactions as context features and training LSTM neural networks on these features, with the goal of predicting the actions of a target based on the actions of their neighbours. Several parameters for this feature generation and several architectures were tested.
  - Network fusion of the previous two improvements, together with a novel testing scheme to use context generation at test time. Using both mechanisms together outperformed each of them.

  As mentioned, due to the high computational demands of most solutions and large dataset size, we partitioned the AVA [35] dataset to obtain a small representative set with a similar distribution, named miniAVA. This also allowed us to iterate faster between architectures. With the resources available we managed to demonstrate that not only is our approach viable but all our augmentations improved the original baseline.

- As for technical contributions [1] [2], we would like to highlight how there is a lack of uniform standards in the research community and how models are often available in non-compatible frameworks. Not only that, but to our knowledge we provide the first implementation based on the open source software Keras [17] (their implementation [28] is based on Matconvnet) of two-stream architectures for action recognition. Additionally we improved the efficiency of an existing library by the authors of [27] for the computation of optical flow for the TV-L1 [103] algorithm [3] and we created a library for conversion of Matconvnet [92] (.mat) and Caffe [48] (.caffe) weights to Keras/Tensorflow [1] (.hdf5) weights for generic neural networks. We also highlight that most of the available frameworks rarely have any utilities for multi-label or imbalanced problems and this fact often forced us to extend already existing frameworks to suit our needs.

1.7 Dissertation Outline

In the remainder of this thesis we propose and explain a complete action detection system. It is organized in the following manner:

- Chapter 2 is separated in two parts: we address some necessary background and then we present detailed descriptions of the state-of-the-art methods from [80] [29] and [35] that are relevant to our implementation.
- Chapter 3 begins with the description of our two-stream multi-label architecture to fuse RGB frames and optical flow data. Then we explain inter-human class relationship context and attention mechanisms and how these can be used to augment the previous architecture.
- Chapter 4 provides a detailed description of the original AVA [35] dataset and our partition of it and the metrics used in further experiments.
- Chapter 5 presents experimental results that validate our implementation and conclusions are drawn from the performed experiments that provide insights into how the augmentations improve the baseline architecture.
- Finally, in chapter 6 a summary of the work carried out in this dissertation is provided, making also references to future improvements that can be done about the presented implementation.

[1] https://github.com/pedro-abreu/twostream-attention
[2] https://github.com/pedro-abreu/action-semantic-lstms
[3] https://github.com/feichtenhofer/gpu_flow/pull/12

2

Related Work

Contents

2.1 Overview

In the following section we present a general background of some needed theoretical concepts followed by the description of two state-of-the-art architectures relevant to our implementation. We assume the reader has a basic understanding of linear algebra, machine learning concepts and of the general concept of shallow 1D neural networks/MLP [64].

2.2 Background

In this section we firstly give a brief theoretical and architectural overview of the state-of-the-art 2D CNNs and their key aspects for our implementation and a concrete example of the ResNet [39] architecture. We also briefly mention RNNs and LSTMs and why they are useful for action recognition tasks. Next, we give an in depth analysis of output layers for neural networks in general and their loss functions and why they are relevant for multi-label tasks such as ours. Next we analyze an existing attention filter for pre-processing images for 2D CNNs and also general guidelines for dealing with imbalanced data.

2.2.1 2D CNNs

Traditional methods for recognition of objects or people in 2D images were often based on edge detection using various 2D filters or kernels. The main idea introduced by the use of convolutional neural networks was that instead of using 1D vectors as weights for the connections between layers as in an MLP, the kernels for these filters are the network weights, thus allowing the network to learn optimal 2D filters [34] and by extension spatial relations encoded in the input features.

Since the input image can have multiple channels (i.e RGB images would have 3) n_C and we may want to use more than a single $f \times f$ filter in one layer since they can learn different features. All n_F filters must have the same number of channels n_C as the input volume and each filter channel is applied to its corresponding input channel (the convolution is in truth a volume convolution). These filters are often small kernels that are applied all over the image. This is often the reason why input images and filters are depicted as cubes, with the tendency being that of the number of channels increasing as their width and height decreases as in Fig. 2.1. This encodes features to a suitable vector representation which is a reason why these networks are so frequently used as feature extractors. These small kernels can therefore be though of as a sliding cube, for which all channels are added after a per-channel convolution. The number of filters per layer can be thought of as the next volume's number of channels since the outputs of the filters are concatenated in a similar fashion. Given an image of size $n \times n$ and

n_F filters of size $f \times f$ all with n_C channels, their output volume is obtained by:

$$(n \times n \times n_C) \circledast (f \times f \times n_C) \times n_F \implies (n - f + 1) \times (n - f + 1) \times n_F \tag{2.1}$$

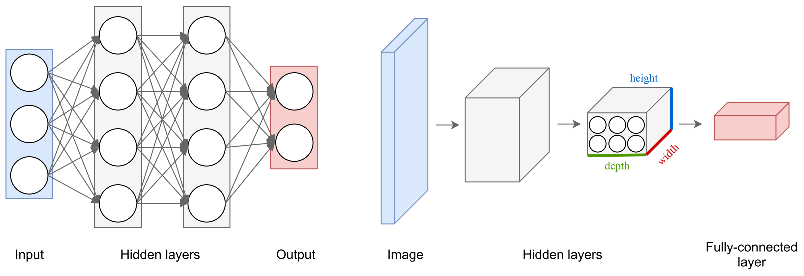


Figure 2.1: MLP vs CNN. Image adapted from http://blog.christianperone.com/2015/08/convolutional-neural-networks-and-feature-extraction-with-python/.

Another way of looking at CNNs (or ConvNets) is that they can be seen, in a similar fashion to MLPs, as function approximators [60] which are arbitrarily complex. Due to the convolution mechanism, CNNs are capable of learning spatial representations from lower-level features at their earlier layers (such as edges) to higher-level features (such as ears or eyes in pedestrian detection) at their final layers [102].

Other improvements to CNN architectures like padding and pooling layers have allowed the accuracy to improve while using less weights. Padding is simply to add extra boundary pixels to an input volume as one problem visible in equation 2.1 with convolution operations is that the outcome filtered image is smaller than the original image. A special case of padding is when, knowing the size of the filters f, one can make the output image have the same size as the original input image by having $p = \frac{f-1}{2}$ empty (with value 0) one pixel layers all around the image. This is also often called *same convolution*.

Pooling layers [34] perform a similar operation to convolutional layers, however where in convolution an element-wise multiplication and then addition is performed while sliding the kernel, with pooling the maximum element is picked in the kernel window in the case of *max pooling* or all elements in the kernel window are averaged in the case of *mean pooling* (mean pooling is used less often). All the weights from the kernels are then added with a bias and an activation function is applied. Many activation functions exist but recently the ReLU [34] function has been proved to provide faster gradient descent for CNNs.

Besides pooling layers and convolutional layers often at the end of the network the weights are concatenated in a single large vector and 2 to 3 FC layers like those used in 1D neural networks/MLPs are used before the output layer. This output layer is intimately linked with the cost function and we explore it in more detail in section 2.2.3.

Given training labels it is possible to use backpropagation [19] to learn gradients for adjusting weights using an optimizer. For this optimizer it is necessary to set to a learning rate as it minimizing a cost which is based on a loss function aggregated over a number of samples as shown in equation 2.2. There are several optimizers based on gradient descent but the state-of-the-art is the Adam optimizer [55].

As both action recognition and imbalanced data present a larger danger of overfitting than other tasks, such as balanced image recognition, it is relevant to mention measures that prevent this. While several measures can be taken to reduce variance, we focus mostly on two: regularization [34] and dropout [83]. Regularization adds an additional term depending on the pre-existing weights w to the cost function that is being optimized by backpropagation process. In more precise terms the cost function can be written as:

$$J(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(y_i^{'}, y_i) + \frac{\lambda}{2N}\sum_{l=1}^{L}\|w^l\|^2 \tag{2.2}$$

where λ is a regularization hyperparameter, N are the number of training samples, w are the weights and $\mathcal{L}(y_i^{'}, y_i)$ is the loss function for a single sample. In equation 2.2, since the term is $\|w^l\|^2$ this represents L2-Regularization, whereas L1-Regularization would have a $\|w^l\|$ term. A dropout layer is a FC layer in which for each node in the layer, for each training batch of samples that node is not considered for back propagation with a uniform probability of p which is the dropout hyperparameter. The number of nodes that are thus turned off should not be too high. As this is done for every training batch not only is the network faster to train but in general nodes are trained on a more varied configuration. Batch normalization [45] is also a possible regularization method however, it has been shown to have a negative impact [59] on dropout (and vice-versa) and as such we do not use it explicitly on our fusion process, opting instead for using dropout.

2.2.1.A ResNet

As an example of a 2D CNN architecture we chose the ResNet [39] architecture. This architecture has been proven to work well for action recognition [28] [36] [30] and is also the one we used in our implementation. After VGG [79] was deemed not only too deep and slow to converge but was also was shown to be degrading results due to vanishing or exploding gradients, ResNet was a proposed alternative. The problem of vanishing or exploding gradients means that while Deep Learning seemed to suggest adding more and more layers would lead to very small classification errors, very deep networks not only take much longer to train but also actually see an increase in the classification error after a large number of layers [34]. ResNet [39] presents a better alternative. The idea is that it is easier to optimize residual intermediate mappings in deep networks rather than an input to output mapping in a very deep network. Residual nets introduce a main new innovation, the residual block. The residual block allows for the output of a layer to be passed further into the network by summation of the gradients after activation, hence the name residual, which means that when performing backpropagation a given layer not only depends on the output of the previous layer but on additive residuals of the previous layers.

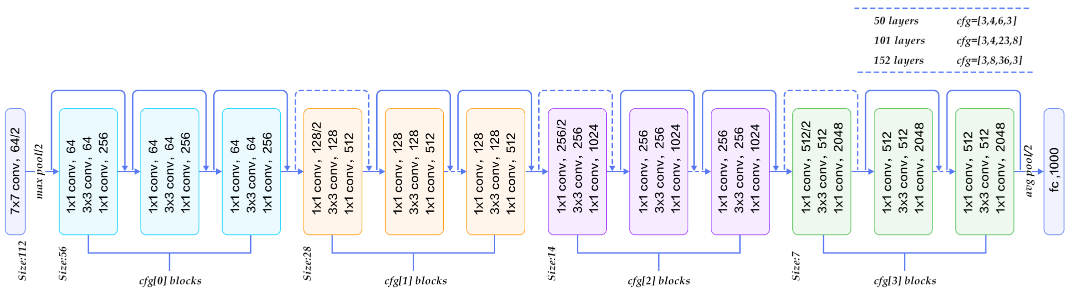


Figure 2.2: ResNet[39] architectures. Image adapted from `http://jwarndt.com/tensorflowblog.html`.

These networks have several advantages. Lower-level features which might be important for the classification are more easily propagated throughout the network (i.e easier to keep identity mappings). The use of residual block helps with the problem of vanishing or exploding gradients thus improving their classification error, training time and in general networks are able to be deeper [34]. Several iterations of the architecture were tested by the authors varying only in the number and sequence of blocks used. Blocks can have different configuration in terms of layers, size of filters and number of filters increasing as shown in Fig. 2.2.

2.2.2 RNNs/LSTMs for Sequence-to-Sequence Learning

Since human actions can be interpreted as sequences over time and many architectures have done so as we mentioned in chapter 1 an architecture that was designed to exploit temporal sequences are RNNs.

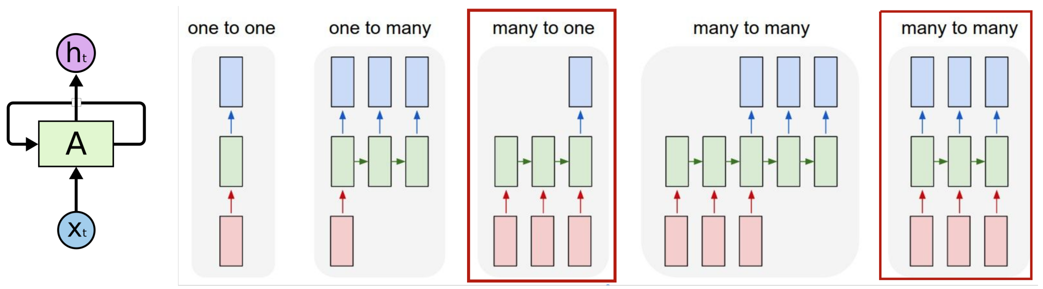


Figure 2.3: Left - Conceptual RNN as a recursive model. Right - After unrolling the model, RNNs can be used for several types of Sequence-to-Sequence Learning we use many-to-one and many-to-many approaches further in this dissertation. Image adapted from `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`.

The input to these networks are time sequences of a fixed duration or single timesteps and their output can also be a sequence or a single timestep as shown in Fig. 2.3. As they are not organized in sequential layers and neurons are connected to themselves this gives the network a recurrent nature. However, through Backpropagation-Through-Time(Backpropagation Through Time (BPTT)) [34] it is possible to train them in a similar manner to other networks. This process converts the RNN into a feed-forward like structure by unrolling it over time as shown in Fig. 2.3, where A are cell memories which have a more complex structure than the neurons of FC layers in a MLP to account for the temporal propagation and that their weights are shared.

Due to these reasons training RNNs is more complex as one would have to express the derivative in backpropagation as a function of itself and the unrolling leads to very long neural networks, which tend to have the problem of vanishing gradients if sequences are very long [6]. To solve the problem of vanishing gradients a more efficient architecture was proposed by [41] named LSTM, which was able to handle longer sequences. The main idea is to alter the main repeating cell (represented as A in Fig. 2.3) to have memory elements.

This memory unit has 3 logistic gates which have learnable associated weights: write, keep and read. Information gets into the cell whenever its "write" gate is on, information stays in the cell so long as its "keep" gate is on, information can be read from the cell by turning on its read gate. This means that the neurons no longer do a multiplication and bias adding operation, but the key is that the LSTM cell chain (once unrolled) are differentiable and can be backpropagated upon and these tend to have gradients which are less prone to go to zero.

2.2.3 Output layers, loss and cost functions

For all previously presented architectures an output layer is needed. While most problems are deemed as multi-class tasks and thus easier to define that is not necessarily the case in the real world. Therefore it is necessary to understand how to adapt the architectures to deal with different types of tasks and that is normally done via the output layers and cost functions. The activation functions of these layers are particularly important as their output is what is used to compute the cost function that is then backpropagated through the network. A large of the explanation here is based on [34], [46] but mostly on the implementation we use from the Keras [17] framework.

We stick to the definition that the term loss function is used when talking about a sum of costs over all training samples as in 2.2 and a cost function is used when referring to the cost applied to a single sample [58].

2.2.3.A Output Layers

The output layer itself is always a FC layer, but the choice of activation function depends on the task. For multi-class classification each sample can belong only to one of C classes. Therefore the architecture will have C output neurons that can be gathered in a vector s (s stands for prediction scores). The target or ground truth label t will be a one-hot encoded vector with a single positive class and $C-1$ negative classes, since classes cannot occur simultaneously. For these tasks, the function that is applied to s is the softmax function. It squashes the vector in the range $[0.0, 1.0]$ and all the resulting elements add up to 1. This means that as elements represent a class, these can be interpreted as class probabilities, thus implying that each score depends on the others. Therefore, for a given class i and all other classes denoted by j the softmax function is:

$$f(s)_i = softmax(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \tag{2.3}$$

On the other hand, for multi-label tasks each sample can belong to more than one of C classes and the architecture will still have C output neurons. However, the target t will be a binary vector with P positive classes and $C - P$ negative classes. The task is normally treated as C binary and independent classification problem where each output neuron decides if a sample belongs to a class or not. Each output neuron applies the sigmoid or logistic function to each element of s, where the sigmoid or logistical function is:

$$f(s_i) = \sigma(s_i) = \frac{1}{1 + e^{s_i}} \tag{2.4}$$

It is important to note that regardless of the activation function, the output layer is a FC layer as any other and as such each $\mathbf{s} = \mathbf{w}^T\mathbf{h} + \mathbf{b}$. where w are learned weights, h is the input vector and b are bias terms. Also important is that, in the testing phase, when the loss is no longer required, these activation functions are also used to get the outputs of the architecture.

2.2.3.B Cross-Entropy Loss

Cross-entropy or (multinomial) logistic loss can be defined more accurately as the negative log-likelihood. We will focus on variations of cross-entropy. The name comes from the calculation of the cross-entropy between the training data and the model distribution, or:

$$\mathcal{L} = -\sum_i^C t_i \log(f(s_i)) \tag{2.5}$$

where t_i and s_i are the target and the prediction score for each class i in C and f is the chosen activation function.

2.2.3.C Categorical Cross-Entropy Loss

This loss is an application of the loss in equation 2.5 with a softmax activation function and is also called Softmax Loss. In this case the labels are one-hot and as such only the positive class C_p keeps its term in the loss since $t_p = 1$. Therefore we can discard all other elements of the summation which are the negative classes and write:

$$\mathcal{L}_c = -\log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right) \tag{2.6}$$

where s_p is the score for the only positive class.

2.2.3.D Binary Cross-Entropy Loss

This loss is an application of the loss in 2.5 with a sigmoid activation function and is also called sigmoid loss. This loss is computed independently for each vector component (i.e for each class) and then summed up. This means the the loss computed for every class is not affected by the others. This sets up a binary classification between 2 conceptual values for every sigmoid activation, where the negative value is also called the "background class" as it notes the absence of a class. In more precise terms, for every j classes in C we setup a binary classification problem such that the global loss is:

$$\mathcal{L}_b = \sum_{j}^{C} \left(- \sum_{i=0}^{1} t_{j,i} \log(\sigma(s_{j,i})) \right) = \sum_{j}^{C} (-t_j \log(\sigma(s_j)) + (1 - t_j) \log(1 - \sigma(s_j))) \tag{2.7}$$

Since the activation function does not depend on the scores of the other classes, the gradient in respect to the score s_i will only depend on the loss given by its binary problem when its backpropagated.

2.2.4 Attention Filters

The main idea for attention filters is that, given a region in an image (e.g a bounding box), we filter the image in such way that a classifier learns features related to that region rather than with the surrounding background. Other approaches to using attention mechanisms exist such as [86] which use optical flow. However, these approaches often operate at a network-level while here we are mostly concerned with attention as a pre-processing filtering step before the input is fed to a neural network.

Two naive approaches are possible: the first would be to crop the area outside that region. and the second would be to simply apply a Gaussian blur with a Gaussian kernel as in 2.8 with an appropriate σ to everything outside that region. Throughout the rest of this dissertation we will refer to the first one as Crop filter and the second one as Gaussian Background Blur (GBB). However two arguments against these approaches can be made. Firstly, for the crop filter, there is no background context which might help a classifier, for example, an abundance of blue might help a classifier guess "swim" action due to the presence of water. Secondly, for both the GBB and the Crop filter there are regions where large contrasts exist, in the crop filter there is a sharp transition from the relevant region to black and in the GBB there is a sharp transition from the relevant region to a blurred region. Since CNNs tend to learn edges early in the network as more low-level features [102], we hypothesize this would compromise their learning. Therefore the ideal would be to have some sort of filter where no artificial edges are introduced. As such, inspired by human foveal vision [90] and the work of [12] and [4], we use the following fast and efficient system to replicate this type of filtering which provides a smooth blur transition between the bounding box and the background:

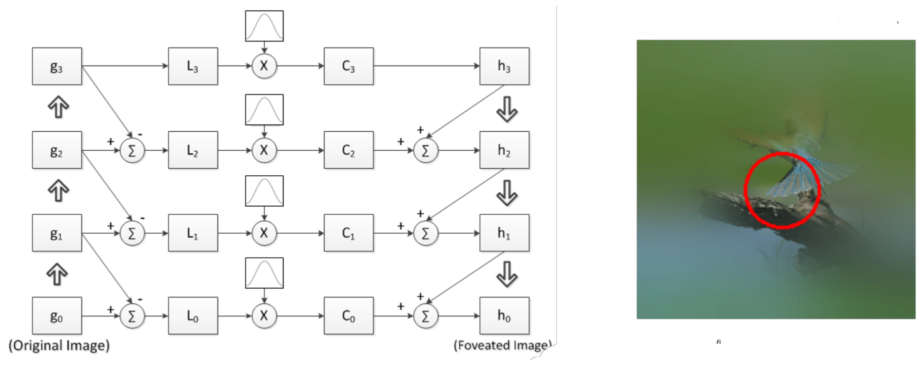


Figure 2.4: Artificial Foveal Vision as described in [4] with Laplacian Pyramid [12] upsampling and downsampling for $K = 4$ (4 levels) and an example of a fovea filtered image from the original paper with $f_0 = 30$ (see 2.9)

The procedure can be described as follows. Firstly, a Gaussian pyramid is built with increasing levels of blur, but same resolution. The image g_{k+1} can be obtained from g_k via convolution with 2D isotopic Gaussian filter kernels with progressively higher $\sigma_k = 2^{k-1}\sigma_1$ standard deviations for each kth level of K levels:

$$G(u, v, \sigma_k) = \frac{1}{2\pi\sigma_k} e^{-\frac{u^2+v^2}{2\sigma_k^2}}, \qquad 0 \leq k \leq K \tag{2.8}$$

where u and v represent the image coordinates. Next, a Laplacian pyramid [12] is computed from the difference between adjacent Gaussian levels. Finally, exponential weighted kernels are multiplied by each level of the Laplacian pyramid to emulate a smooth fovea:

$$k(u, v, f_k) = e^{-\frac{(u-u_0)^2+(v-v_0)^2}{2f_k^2}}, \qquad 0 \leq k \leq K \tag{2.9}$$

where $f_k = 2^k f_0$ denotes the exponential kernel standard deviation at the k-th level and (u_0, v_0) are the the kernel center which defines the fovea focus. The authors [4] use this together with 2D CNNs and validate its use as a filter.

2.2.5 Balancing Strategies for Imbalanced data

In the literature it is referenced that most MLDs suffer from a high level of imbalance, although it is a challenge to establish a single metric to quantify this imbalance [15]. Indeed, [105] mention that the number of positive training instances with respect to each class label for many labels is far less than its negative counterparts, which may lead to performance degradation for most multi-label learning techniques or to deceivingly give optimistic results if using the wrong metrics (better stated as the accuracy paradox [82] when learning the imbalanced distribution). Regardless, most strategies for addressing class imbalanced datasets can be divided into two categories [37]. Data-level methods alter the training set such that the data is artificially balanced, while classifier-level methods keep the training set unaltered but adjust the training process of the algorithms to compensate for the imbalance. An important caveat is that assuming the imbalance is the main cause for this degradation may be misleading as some inner problems of the imbalance problem such as the lack of data itself [31] from certain classes can be as impactful as the relative imbalance between the classes, but using data-level balancing methods tends to help even in this case. Below we present one data-level method and one classifier-level method which are commonly used[11]:

- Oversampling [37]: This process at its most basic form involves simply replicating randomly selected samples from minority classes until the dataset is balanced (random minority oversampling). Data augmentation methods can also be used on the replicated samples, however there is always the eventual chance that even with repetition the data space is still too small for the network to learn any relevant features.

- Transfer Learning [67] [101]: Transfer Learning is the method of initializing a network or algorithm with weights learned from another data distribution. This can be seen as a classifier-level method, however more classical ones are penalized cost or weighting distributions[22]. Transfer learning has been shown to help with the problem of imbalanced data, since the classifier is not overfitted on the imbalanced distribution [3]. Fine-tuning, which is a similar idea, is to use these weights as initialization and then train the network on the new distribution and has the same benefits. Recent approaches [96] have demonstrated the effectiveness of fine tuning in the problem of action recognition.

2.3 State of the art

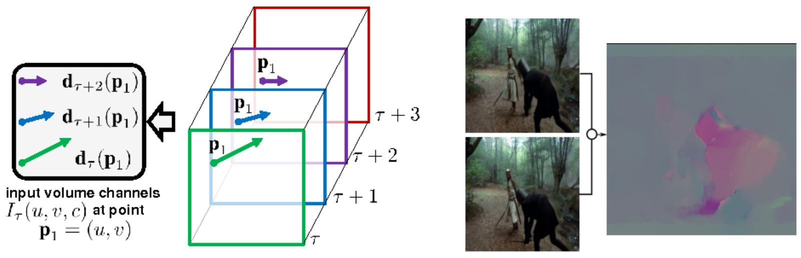
In the following section we discuss two state of the art approaches using two-stream CNNs. The first [29] is a groundbreaking work on two-stream fusion in action recognition using optical flow, which improves on earlier two-stream architectures [80]. We explore how these approaches can be competitive against more complex models and how different types of fusion between streams can be employed. The second approach we describe is the model provided in the original AVA dataset paper [35] as a baseline. We discuss how they extend the Faster R-CNN (R-CNN) architecture for the temporal domain. It is relevant to show this implementation not only to have a reliable benchmark but also to demonstrate how hard these tasks can be even with large amounts of computational resources available. This approach is itself a two-stream CNN approach but with several adaptations for the specific task.

2.3.1 Two-Stream networks and Two-Stream Fusion

Initially, action recognition architectures used a single 2D CNN with RGB inputs. Inspired by human neurology [33], a novel approach was introduced in [80] and then extended by [29] [28]. The main innovation was having two separate CNN streams: one for spatial appearance which receives RGB inputs and another for motion, which receives Optical Flow (OF) inputs. Then, one of several types of fusion is performed in order to obtain a final prediction.

2.3.1.A Optical Flow

Given the difficulty of previous deep architectures [52] to learn motion features, [80] explicitly modeled motion features in the form of stacked (over pair-wise video frames) optical flow vectors as shown in Fig. 2.5. Optical flow computes the displacement vectors between two frames to obtain a representation of motion. While other features have been suggested to represent motion like Dynamic Images [7] or Enhanced Motion Vectors [104], optical flow is the one that is still used by most state-of-the-art approaches.



$\mathbf{d}_{\tau+2}(\mathbf{p}_1)$
$\mathbf{d}_{\tau+1}(\mathbf{p}_1)$
$\mathbf{d}_{\tau}(\mathbf{p}_1)$
input volume channels $I_\tau(u,v,c)$ at point $\mathbf{p}_1=(u,v)$
$\mathbf{p}_1$
$\tau+3$
$\tau+2$
$\tau+1$
τ

Figure 2.5: Optical Flow showing the displacement vectors across time at a given point (as explained in [80]) and an example of TV-L1 [103] optical flow encoded as an RGB image.

Optical flow algorithms can be divided into two types: sparse and dense. Sparse algorithms [77] use certain key points to calculate displacement vectors, while dense algorithms calculate pixel-wise displacement as shown in Fig. 2.5. Note that in Fig. 2.5 Red encodes the x direction of the displacement vectors, Green the y direction and Blue encodes the norm of the vectors, but we only use the first two channels. Even though OF could be encoded as two grayscale images, this method makes image retrieval faster. Furthermore, for dense optical flow, we can distinguish algorithms between network-based and traditional algorithms. Network-based implementations such as FlowNet [21] use neural networks as function approximators for image to OF mappings. Traditional algorithms such as Farneback [26], Brock [10] and TV-L1 [103] formulate the optical flow problem as an optimization problem under a set of constraints and often arrive at an iterative algorithmic solution. While harder to integrate, only recently have network-based approaches surpassed traditional algorithms [25] [44] and they require more computational resources to implement. Another advantage of traditional algorithms is how they can compensate camera motion, such as warped flow [96].

As such, we focus mostly on the TV-L1 [103] algorithm since we use it in our implementation and [25] [13] showed it outperformed other methods (including network-based methods) offering a good balance of efficiency and accuracy. As discussed in [85], the name of this method comes from the fact that it is based on the minimization of an optimization function containing a term using the L1 norm and a regularization term using the total variation (TV) of the flow. This optimization is done for every pair of sequential input frames. Additionally, the authors introduce an auxiliary variable $\mathbf{v}$ to enable a convex relaxation of the problem and the formulation can be written as:

$$\min_{\mathbf{u},\mathbf{v}} \sum_{\mathbf{x}\in\Omega} (|\nabla \mathbf{u}_1| + |\nabla \mathbf{u}_2|) + \frac{1}{2\theta}|\mathbf{u}-\mathbf{v}|^2 + \lambda|\rho(\mathbf{v})| \tag{2.10}$$

Where λ, θ are parameters of the algorithm, $\mathbf{u}(\mathbf{x}) = (u_1(\mathbf{x}), u_2(\mathbf{x}))$ is the displacement of the point $\mathbf{x}$ from frame t to the next frame $t + t$, as the point $\mathbf{x}$ can move from frame to frame across video frames and the optical flow must track such displacement. Ω are all the coordinates of all pixels in a frame, $\rho(\mathbf{v})$ gives the difference in pixel intensity as $\mathbf{x}$ moves according to $\mathbf{v}$.

The main feature of this formulation is that it allows discontinuities in the flow field, while being more robust to noise than classical approaches. The algorithm to solve 2.10 is an efficient numerical scheme, which is solved by alternate minimization of $\mathbf{u}$ and $\mathbf{v}$ (a more in depth explanation can be found in [25]).

2.3.1.B Architectures

The first architecture to employ two CNN streams for action recognition [80] is shown in Fig. 2.6. The input to the spatial stream is a single RGB frame of the video, and the input to the temporal stream are 10 successive optical flow frames centered around the input RGB frame. This means video level prediction was done by averaging over sampled frames. This leads to the false label assignment problem, that is the ground truth of each of these clips is assumed to be the same ground truth of the video which may not be the case if the action just happens for a small duration within the entire video. Nonetheless, most two-stream architectures have employed a similar assumption in order to use 2D CNNs. Each stream was pre-trained separately and the output scores of both streams are averaged as seen in Fig. 2.6 which is called *class score fusion*. Despite seeming a rudimentary approach, other state-of-the-art methods have achieved good results with this type of fusion [96] for an even larger number of streams.

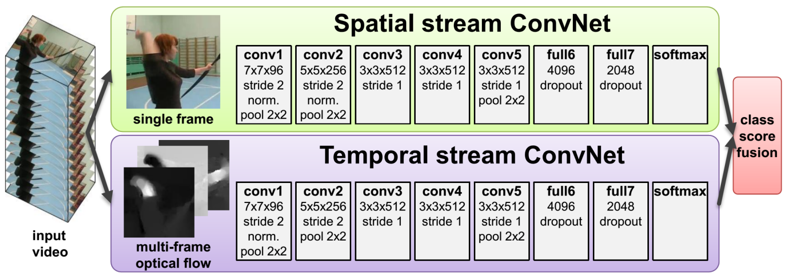


Figure 2.6: Early 2-stream CNN [80] as an example of Class Score Fusion and two-stream inputs.

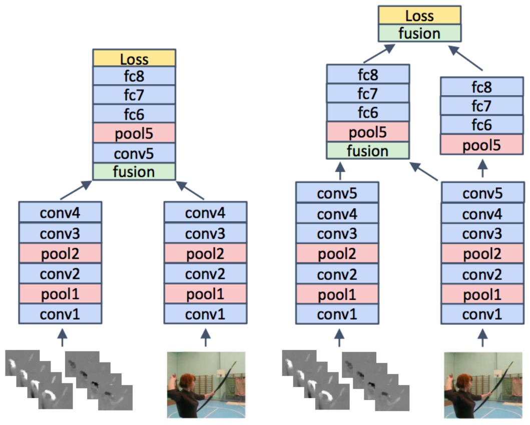


Figure 2.7: Fusion in 2-stream CNNs as explained in [29], exemplifying several strategies for convolutional and FC fusion.

More recently, [29] go beyond the architecture of [80] and explore alternative fusion strategies that allow for the learning of combined spatial and temporal features. Their work focuses mostly on the fusion of convolutional layers however the methods they propose can easily be extended for the fusion of FC layers as shown in 2.7. In other words, the definitions used by [29] for the fusion of convolutional layers generalizes to FC layers without 2 dimensions and it is possible to see in 2.7 that they used it.

An example of such method is *concatenation fusion*. While the definition used by [29] is for the fusion of convolutional layers the same ideas were used for the fusion of fully connected layers was used. Concatenation fusion does not define a layer with filters itself but leaves this to subsequent fully connected layers to learn suitable weights. While these results are tested on relatively small networks, the same authors [28][27] tested their concepts also on deeper architectures like ResNet [39] variations with similar results.

An example of one of these fusion methods is concatenation fusion, which is also implemented in the Keras[17] framework. We believe that the definition provided by the Keras [17] framework is easier to interpret. This layer concatenates a list of inputs, and its action is self-explanatory: it takes as input a list of layers, all of the same shape except for the concatenation axis, and returns a single layer, the concatenation of all inputs. For the concatenation of FC layers, there is only one axis, the length of the FC layers.

Note that concatenation fusion does not define a correspondence but leaves this to subsequent (FC) layers to define by learning suitable (spatiotemporal) filters in those the layers. While these results are tested on relatively shallow networks, the same authors [28][27] tested their concepts also on deeper architectures such as ResNet [39] variations with similar results.

2.3.2 AVA Action Localization Model

After the release of the AVA dataset, the authors also released a model using state-of-the-art concepts. They extend the work of [68] by using the I3D architecture. I3D [13] extends the two-stream models by replacing the 2D convolutions in InceptionNet [84] by 3D Convolutions. The AVA Action Localization Model uses I3D initialized with Kinetics [53] in conjunction with a modified version of Faster R-CNN [73], a well known network architecture tailored for object detection which has been also extended for action detection before in [43] [14].

With the same idea, instead of using Faster R-CNN as a pre-processing step they integrate it in their framework and use a 2D Resnet50 [39] initialized with ImageNet [75] for the first component of the network, region proposal. This smaller network, called Region Proposal Network (RPN) has as its only purpose to compute several hypothetical proposals for regions where the target might be. This means that even at an intermediary layer these regions can already exist at a more abstract feature level and the authors use this hypothesis to their advantage. This network has an output feature map from the conv4 layer which has the same size as the Mixed4e layer of the I3D network, thus allowing them to perform pooling over these merged volumes and over all timesteps. Then they fuse the two layers after average pooling as shown in Fig. 2.8.

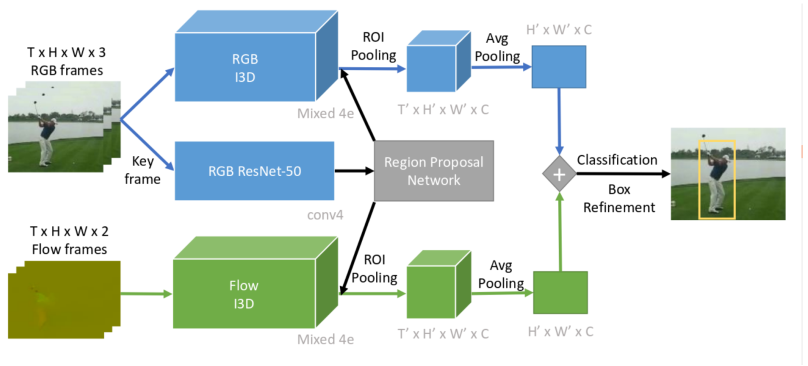


Figure 2.8: AVA Localization Model. Note that the I3D [13] streams can easily be replaced with 2D CNNs using a voting scheme for each frame with results as shown in Table 2.1.

A key difference that exists between this and the previous related work we mentioned [29] is that the action labels of AVA are not mutually exclusive and to address this they replace the standard softmax loss function as shown in equation 2.6 used for UCF101 [81] by a sum of binary sigmoid losses, one for each class as shown in equation 2.7.

Model	Temp + Mode	UCF101-24	AVA
2D	1 RGB + 5 Flow	60.1%	13.7%
3D	20 RGB	77.0%	14.5%
3D	20 Flow	71.3%	9.9%
3D	40 RGB + 40 Flow	76.0%	15.6%

Table 2.1: mAP@0.5IoU for action detection on UCF101-24 and AVA. Note how the difference between 2D and 3D methods is smaller in AVA, revealing that the larger receptive field and consequent computational cost may not necessarily be the key for better results. These results also suggest AVA has a richer temporal context.

3 Implementation

Contents

3.1 Overview

In the following chapter, we explain our base two-stream CNN architecture followed by all the augmentations we add to it. Original videos are split in 3s second segments. We use the TV-L1 [103] algorithm for computing optical flow, which we store in disk, and we use groundtruth bounding boxes at training time. We discuss key aspects of the pipeline such as the attention filtering to filter out the background of frames outside bounding boxes or using labels to generate context features which are then used to train LSTM architectures. We finalize by talking about how to use the previously introduced neural network fusion concepts in section 2.3.1.B for context fusion. We also explain a novel testing scheme necessary for the model to be used in real scenarios where groundtruth test labels are not available. This novel testing scheme involves generating context features from predicted labels either from the two-stream architecture or from a previous context fusion. All these conceptual steps can be summarized in Fig. 3.1 where we illustrate the pipeline of our approach, highlighting the components of the pipeline we will most focus on.

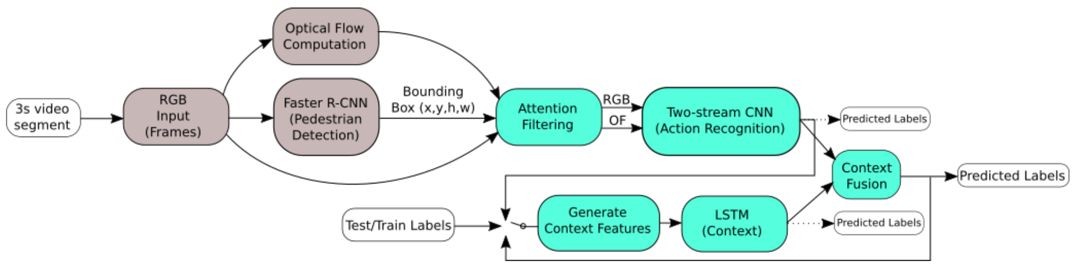


Figure 3.1: Conceptual pipeline of our approach. In green are the pipeline components we are mostly focused on, the ones we do not focus on are grayed out. In white are the inputs and outputs of the pipeline. The switch shows how context features can be generated from 3 different sources (training labels are necessary at training time). In dotted lines are possible intermediary predictions from the pipeline.

Before we explain our implementation, in Fig. 3.1 we can see that some error propagation from using Faster R-CNN [73] as a pre-processing step can occur. We do not focus on extending the Faster R-CNN [73] and we use the provided models trained on the AVA [35] dataset to extract the bounding box annotations we need for our experiments. We argue this impact is not the main priority when trying to improve performance by mentioning the ablation study shown in Table 1.1. Furthermore, we note that the two-stream architecture is tailored for this specific task and as such some aspects of it may not generalize to all action recognition datasets. These tailored aspects include our subsampling strategy and our custom loss (generalized binary loss) function.

3.2 Base Architecture

We propose a base architecture inspired by recent two-stream approaches such as [29] [28]. Each stream as shown in Fig. 3.2 is a ResNet50 [39] 2D CNN. We chose a ResNet50 [39] architecture for each stream for two reasons: we had access to pre-existing weights [29] trained on UCF101 [81] and this is a modern architecture without some issues of previous deep architectures like VGG [79] or the large computational cost of even deeper ResNet [39] variations. Each network stream, RGB and OF, was individually trained on our data.

We fuse these networks using concatenation fusion of their last FC layers and then train a FC layer so that spatiotemporal features can be learned as shown in 3.2. Since retraining the whole architecture would be too time consuming, their individually fine-tuned weights are loaded and then the original layers of both streams are frozen (i.e not updated in backpropagation) so we only train the desired spatiotemporal filters. This is a concatenation using the FC layer immediately after the last convolutional layer in the ResNet50 [39] architecture of each stream.

The first reason why we did not employ convolutional concatenation fusion is that would involve having extra convolutional layers to learn 2D filters and FC layers, despite hypothetically being less descriptive, are less computationally heavy. Furthermore, the usage of FC layers in CNNs is to detect specific global configurations of the features detected by the convolutional lower layers. They usually are used at the end of CNNs at a point when the input has been reduced by the previous convolutional layers to a compact representation of features. As such, we believe that these compact representations are enough to learn joint spatiotemporal features from both streams. An additional reason for using concatenation fusion with FC layers is that we also planned to simultaneously fuse context architectures (explained in section 3.4.2) with this approach, as shown in Fig. 3.7, and our context architectures are not convolutional.

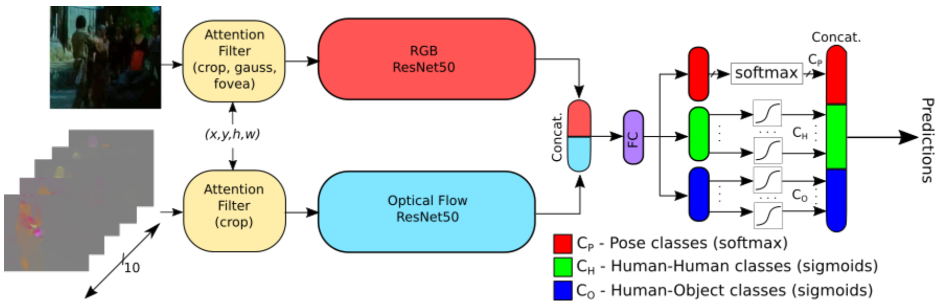

Attention Filter (crop, gauss, fovea)
RGB ResNet50
(x,y,h,w)
Attention Filter (crop)
Optical Flow ResNet50
10
Concat.
FC
softmax
Concat.
C_P
C_H
C_O
Predictions
🟥 C_P - Pose classes (softmax)
🟩 C_H - Human-Human classes (sigmoids)
🟦 C_O - Human-Object classes (sigmoids)

Figure 3.2: Our proposed two-stream architecture. The input is a single frame and an optical flow stack and the output is a set of floating point predictions.

While omitted in Fig. 3.2 we use two intermediate dropout layers (one before the spatiotemporal FC layer and one after). The two dropout layers imply that the spatiotemporal features learned can be more robust to overfitting. We opt for dropout values of 0.5. Despite action recognition architectures often having very high values of dropout [96] we also did not want to hinder the performance of our architecture while trying to compensate for overfit, and therefore chose this value as a compromise. For the architectures shown in Fig. 3.6 and 3.7 we also use this dropout approach in the same manner.

In order to understand the architecture output layers it is necessary to understand our custom loss (generalized binary loss) function as described in section 3.2.1. In order to understand the architecture inputs it is necessary to understand our subsampling and voting scheme as described in section 3.2.2.

3.2.1 Generalized Binary Loss Function

We will first address the output layers and how we compute our loss function. We follow the notation used previously in section 2.2.3.

At first glance, since our task is a multi-label one and if we followed the example of the provided AVA Localization Model as described in section 2.3.2 we would have an output layer of C independent sigmoid activation functions, where C is the number of classes in the dataset. This would mean we would simply use a binary cross-entropy loss as explained in section 2.2.3.D.

Observing the label structure of the dataset we use (see chapter 4 for further detail) we notice that the labels are separated in three categories: mutually exclusive pose classes (C_P of them), non-mutually exclusive human-human interaction classes (C_H of them) and non-mutually exclusive human-object (C_O of them) interaction classes, such that $C = C_P + C_H + C_O$. While this means that the task is still a multi-label task, it implies that the problem is not entirely mutually exclusive as the sigmoids only output layer used by the AVA Localization Model would suggest.

In our architecture we address this incongruity and we suggest having three separate output layers after the learned spatio-temporal filters, one of has a softmax activation function (corresponding to the pose classes) and size C_P and the other two each have many sigmoid activation functions for each of their element vectors and their sizes are respectively C_H and C_O as shown in Fig. 3.2.

This implies that the loss function cannot only be categorical cross-entropy as in equation 2.6 or only binary cross-entropy as in equation 2.7 (both are explained in section 2.2.3). Each category must have its own loss computed, however the whole architecture must minimize a global loss which can be expressed as:

$$\mathcal{L}_{GB} = \overbrace{-\alpha_P \log\left(\frac{e^{s_p}}{\sum_j^{C_P} e^{s_j}}\right)}^{pose\ classes\ loss} + \overbrace{\alpha_H \sum_j^{C_H}\left(-\sum_{i=0}^{1} t_{j,i}\log(\sigma(s_{j,i}))\right)}^{human-human\ classes\ loss} + \overbrace{\alpha_O \sum_j^{C_O}\left(-\sum_{i=0}^{1} t_{j,i}\log(\sigma(s_{j,i}))\right)}^{human-object\ classes\ loss} \quad (3.1)$$

where α_P, α_H and α_O are hyperparameters that offer the possibility to weight the contribution of each of the output layers for this global loss (see $loss\_weights$ in the Keras[17] framework). We set $\alpha_P = \alpha_H = \alpha_O = 1.0$, thus weighting all of them equally. Updating the weights of each output layer is possible as the encoding of the target labels is done as a binary vector as shown Fig. 3.6 which can then be partitioned for each of the output layers into smaller vectors. Given this, and since we could not find any type of similar loss in the literature, we decided to name this loss *generalized binary loss*.

We used this loss in all other experiments as we believe this assumption better reflects the labeling structure. Additionally, we believe that in the future, as architectures tend to model more complex tasks, these types of loss functions will become more relevant.

3.2.2 Subsampling and Voting Scheme

Similarly to the previous discussion, we refer to chapter 4 that discusses the dataset used for this task in detail, where we can see that what is labeled are 3 second video segments at 30 Frames Per Second (FPS) centered around a keyframe. This presents two major issues. Intuitively, a receptive field of 90 frames would be needed and a 3D CNN architecture would be necessary to obtain the labels and process a single segment at once.

To solve the issue of requiring a large receptive field, we propose a subsampling scheme as shown in Fig. 3.3. While it is inevitable to lose some information, this makes our implementation much more computationally feasible. We consider that our sampling is spaced enough that the frames are representative of the labeling in each segment. Although informed by the rest of the segment, the labeling is heavily dependent on the middle-frame or keyframe and our subsampling method reflects this.

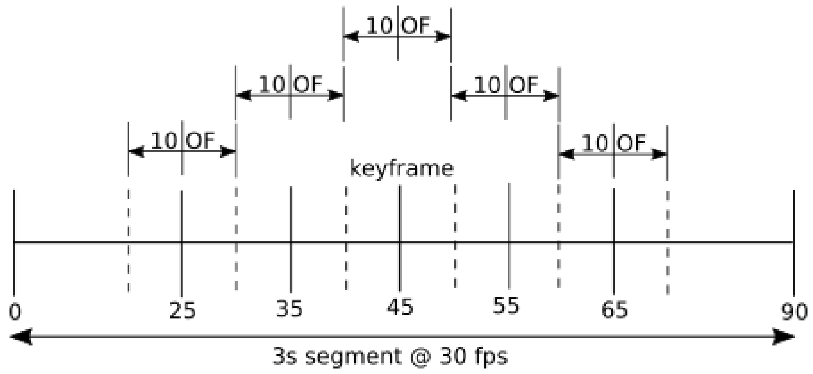


Figure 3.3: Our subsampling method. Each 3s segment is subsampled at 10 frame intervals centered around a keyframe to obtain 5 frames (including the keyframe). Around each of the 5 extracted frames, 10 consecutive optical flow stacks are extracted from pair-wise frames.

The optical flow frames are pre-computed for every segment and stored in disk, encoded as shown in Fig. 2.5.

In order to solve the issue of how to obtain predictions from a sequence of frames, we use a voting scheme. Since our architecture is based on 2D CNNs, to obtain the predictions for a single segment we pass each frame of our 5 subsampled representative frames and their corresponding optical flow volumes through one of our architectures (such as the two-stream model shown in Fig. 3.2 for example) and we store all predictions. After having the predictions on each of the subsampled frames, we compute votes from each stored prediction. As the labels of the dataset are grouped according to mutually exclusive labels (pose classes) and non-mutually exclusive labels (human-human and human-object classes) we compute votes in the following manner. For the mutually exclusive predictions we count the maximum valued prediction as a vote and for all non-mutually exclusive predictions we count any prediction values above a certain threshold v as valid vote. This v threshold or *voting hyperparameter* is a necessary hyperparameter of our architecture and most architectures that have sigmoid activation functions on their output layers. For the mutually exclusive classes we take the most voted class as the predicted class for the segment. For each non mutually exclusive class category we take the top 3 most voted classes as the predicted classes for the segment. Note that for the non-mutually exclusive classes the number of predicted classes can be 0 if no prediction is above the threshold v. Note that as the predictions for either the mutually exclusive classes and the non-mutually exclusive classes are the output of the softmax or several sigmoid activation functions their value is between 0 and 1, and so is v. As previously mentioned, the final labels obtained via this voting scheme are represented as a binary vector as illustrated in Fig. 3.6.

3.3 Attention

Following the discussion of our subsampling method shown in Fig. 3.3 and our previous discussion of attention filters in section 2.2.4, we explain what attention filters we used for this task. From Fig. 3.1 we can see that the inputs to the attention filters are a frame (RGB or optical flow) and a bounding box. Bounding box notations normally differ between (x_0, y_0, x_1, y_1) where x_0, y_0 are top-left corner coordinates and x_1, y_1 are the bottom-left corner coordinates, or (x, y, h, w) where x, y are top-left corner coordinates and h, w are the height and width of the bounding box. We stick to the latter.

Firstly, for the RGB input frames we evaluate three different filters: a crop filter, a GBB filter and a fovea filter. While the first two were already discussed in section 2.2.4 we adapt the fovea filter to allow for an elliptical fovea which we believe is helpful, as most BBs tend to be taller than wider. The main difference is that the fovea exponential kernel weights are now expressed as:

$$k(u, v, f_{kx}, f_{ky}) = e^{-\left(\frac{(u-u_0)^2}{2f_{kx}^2} + \frac{(v-v_0)^2}{2f_{ky}^2}\right)}, \qquad 0 \leq k \leq K \tag{3.2}$$

where $u_o = x + w/2$, $v_o = y + h/2$ and we define $f_{kx} = 2^k f_{0x}$, $f_{ky} = 2^k f_{0y}$ with $f_{0x} = \frac{1}{2}w$, $f_{0y} = \frac{1}{2}h$. We use $K = 4$ levels of the Laplacian Pyramid [12] explained previously and $f_0 = 10$. All filters are exemplified in Fig. 3.4. Note that while the fovea filter seems to blur everything, this blur is very tenuous near the center of the BB and gets progressively more intense the further the pixels are from the center of the BB.

Secondly, for the Optical Flow input frames we decided to only employ the crop filter. The reason is that as both GBB and fovea filtering involve blurring, the blurring of motion features would have a possibly misleading and undesirable interpretation of creating artificial motion vectors in incorrect places.

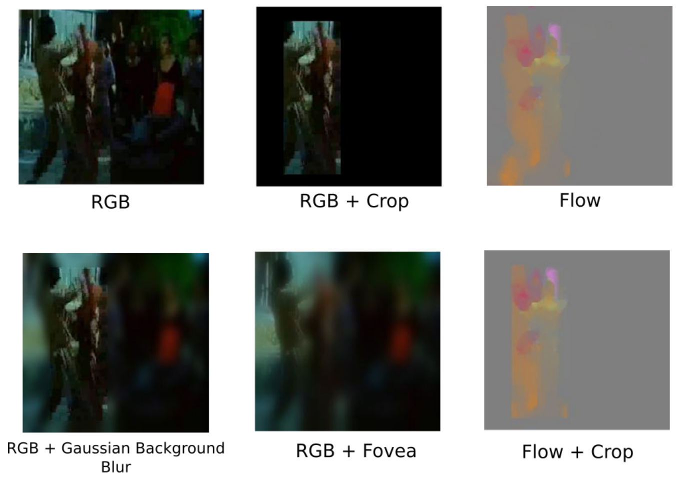


Figure 3.4: Example of all the attention filters applied to frame of a subsampled segment. For the fovea filter $K = 4$ and $f_0 = 10$ and for the GBB filter $\sigma = 10$.

3.4 Context

In this section we firstly explain how we generate context features and then how we train LSTM architectures so that they can learn to predict labels from these features.

3.4.1 Context Features

We define context as the goal of learning the actions of one actor based on the labels of their neighbours across time. Note that since our task is a multi-person action detection task the labeling is done in a person-centric manner (i.e labels for each BB) for each keyframe in a long video snippet at the rate of 1Hz. As previously mentioned, 3s segments centered around these keyframes inform the labeling. For our context approach a single segment or a keyframe are seen as the same: they consist of one timestep in terms of context.

Therefore, we need to use features that can encode the labels of an actor's neighbours over time in such a way that these can be inputs for a learning task to learn complex inter-human class-relationships.

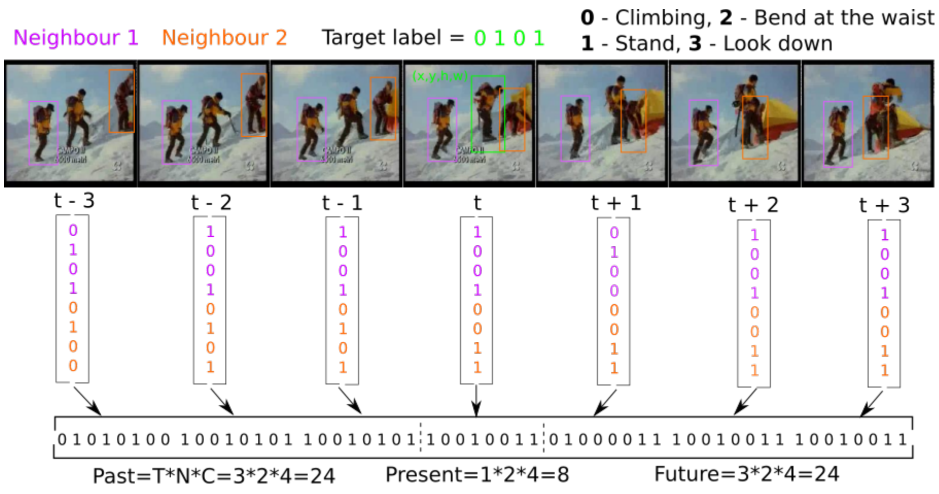


Figure 3.5: A simple example of context generation with number of backward and forward timesteps $T = 3$, neighbours $N = 2$ and number of classes $C = 4$. At the end of the procedure we obtain the context vector for target (x, y, h, w) at time t. The same procedure must be done for the other targets at other times.

A simple demonstration of our context generation method can be seen in Fig. 3.5. For all further discussions, we assume the number of timesteps T used from the past and the number of timesteps T used from the future are identical. That being said, if we chose the N closest neighbours to an actor and if our task has C classes, then each timestep has $N \times C$ binary features and the length of the context vector is $(T + 1 + T) \times N \times C$.

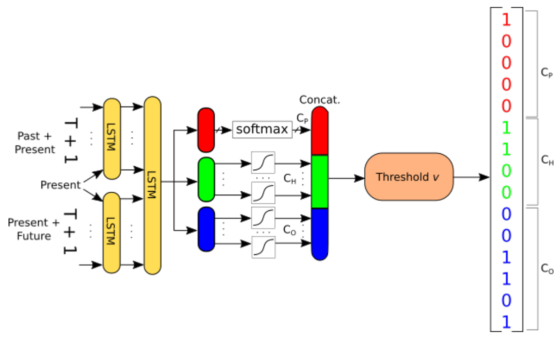
We identify the N closest neighbours using a pixel-wise Euclidean distance between their BB's center and we assume this order is kept throughout the entire context time window. This might lead to identity switches but we assume those are exceptions. Additionally there can also exist cases where one or more of the neighbours are no longer in a keyframe. In this case, we assign a sub-vector with 0's to the non-existing neighbours and put them at the end of the feature vector from that timestep. Using the target's own action labels for input features would likely overfit as the actions for a target tend to be consistent for a time window and thus the networks would learn a 1 to 1 mapping of repeating those labels, instead of learning the relationships between the actions being performed which is our main goal in this section. As such when we retrieve the labels from the past and future we discard the 1st closest neighbour so we do not the target itself, and we use the N next closest neighbours, as shown in Fig. 3.5.

Since context is generated at a segment/keyframe level and labelling is done at a 1Hz rate, the time window of the context is in actual seconds (eg. the total time window in Fig. 3.5 is 7 seconds). Note that our context generation model is flexible enough to use asymmetric context if we so desired, that is using more past keyframes than future keyframes and this allows our idea to conceptually adapt to problems where no future context could be used, such as real-time recognition.

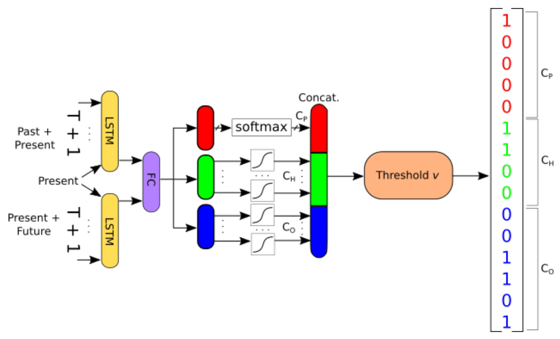
3.4.2 Context Architectures

Given our previous discussion on the usage of RNNs/LSTMs for sequence modeling in section 2.2.2 their usage for this problem seems ideal. The problem itself is presented in that way: the features for each timestep of a sequence fed to the network are the number of neighbours and their labels encoded as a binary vector of size C.

However an issue stands in the way of using a simple out-of-the-box LSTM model: we do not want to predict labels for each timestep of a sequence nor do we want to predict a label at the start or at the end of a sequence. Instead, our target label should be an output from the middle of the sequence. Due to the way the implementation of LSTMs work in the framework we use (Keras [17]) we solved this problem by having two input LSTM layers: one receives a sequence of the past timesteps and the present timestep (i.e it must have a $T+1$ timestep receptive field) and another receives a sequence of the future timesteps and the present timestep (i.e it also must have a $T+1$ timestep receptive field). The present timestep is used in both inputs since using it only as input in one layer would introduce a bias towards either past or future timesteps. This involves splitting and reshaping the input context vector. However, this creates the issue of how to merge the outputs of these two LSTM layers. Since an ideal way is not immediately clear we propose the two models displayed in Fig. 3.6:



(a) Model A



(b) Model B

Figure 3.6: Our two proposed context models: model A encodes the input sequence into yet another sequence as input to a second LSTM layer, model B outputs two predictions which are then fused with a FC layer. The predicted labels are merely illustrative.

We call these models model A and model B. Model A represents a type of many-to-many sequence learning in the two first layers while model B represents a type of many-to-one sequence learning as previously shown in Fig. 2.3. These architectures are created for prediction so the output of each of the input layers in model B is in one case the conceptual timestep after the present and in another the conceptual timestep before the present. The LSTM layer using only future and present input timesteps can be implemented in the same way as the other layer if the sequence is reversed.

In order to train these architectures by themselves, no subsampling or voting scheme is required as shown in Fig. 3.6: target labels are obtained from a single prediction vector (i.e as if votes were only cast once). However, since the output layers still have sigmoid activation functions we use the same hyperparameter v for these networks as we used in the networks before.

3.5 Context Fusion

After presenting the previous two methods, it is intuitive to use them simultaneously. We propose to explore two ways in which the fusion of our context models and the two-stream models trained with attention filtering can be made: concatenation fusion and class score fusion. In a first approach we used groundtruth labels to generate context vectors for our context models to evaluate if context does indeed help. Additionally, since we cannot use groundtruth test labels to generate context vectors for our context models in a real scenario, we propose a two-pass testing scheme where this is no longer the case.

3.5.1 Concatenation Fusion

In Fig. 3.7 and Fig.3.8 we show how we propose to use concatenation fusion to train a FC layer that could not only learn spatiotemporal weight but spatiotemporal context weights. Note that whenever we refer to concatenation fusion it is implied that it is a FC concatenation fusion. We use the same training method as used for the model in 3.2, that is all the layers from the original models are frozen and only the FC layer is trained. As shown in Fig. 3.8, at testing time the context features are at a segment-level and as such it is assumed that they are the same for every subsampled frame that is used. At training time each frame and Optical Flow and training context vector is used as an input as shown in Fig. 3.7. The labels are assumed to be the same for in each segment during training.

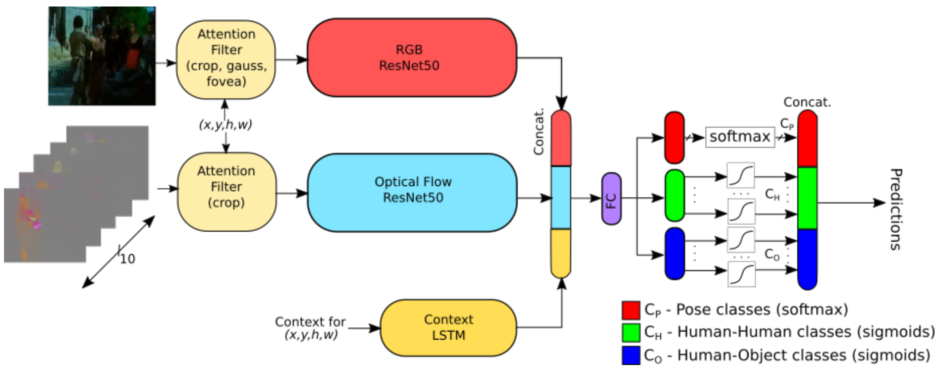
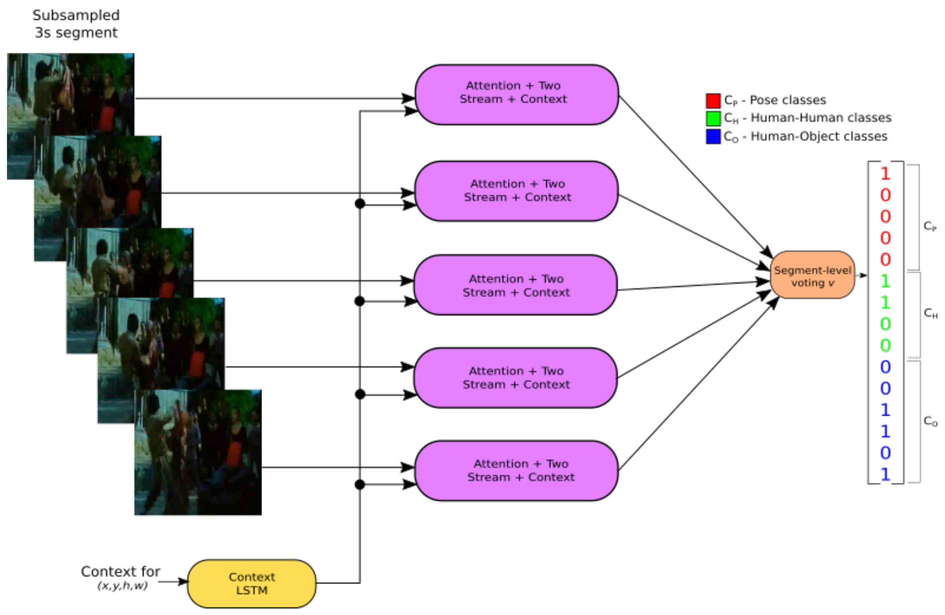

Attention Filter (crop, gauss, fovea)
RGB ResNet50
(x,y,h,w)
Attention Filter (crop)
Optical Flow ResNet50
10
Concat.
FC
softmax
Concat.
C_P
C_H
C_O
Predictions
Context for (x,y,h,w)
Context LSTM
🟥 C_P - Pose classes (softmax)
🟩 C_H - Human-Human classes (sigmoids)
🟦 C_O - Human-Object classes (sigmoids)

Figure 3.7: Our proposed concatenation fusion model for context. Note that the figure shows only the input of a single frame and that each 3s segment is subsampled to 5 RGB frames.

While conceptually this method can learn shared weights, it is necessary to train the FC layer by using the previously learned models together, which means that easily iterating and testing several types of combinations of models to fuse is time consuming.



Subsampled 3s segment
Attention + Two Stream + Context
Attention + Two Stream + Context
Attention + Two Stream + Context
Attention + Two Stream + Context
Attention + Two Stream + Context
Context for (x,y,h,w)
Context LSTM
Segment-level voting v
C_P - Pose classes
C_H - Human-Human classes
C_O - Human-Object classes
1 0 0 0 0 1 1 0 0 0 0 1 1 0 1
C_P
C_H
C_O

Figure 3.8: Our proposed concatenation fusion model for context. Note that the figure shows only the input of a single frame and that each 3s segment is subsampled to 5 RGB frames. The predicted labels are merely illustrative.

3.5.2 Class Score Fusion

In Fig. 3.9 we show our proposed class score fusion method. There are precedents of class score fusion of CNN streams and LSTMs [98] but with different models. The main idea behind this type of fusion is that at testing time the predictions from the two-stream model and the context model are averaged for each subsampled frame in a segment. Note that since context operates at a keyframe/segment level the prediction for every subsampled frame is assumed to be the same, but the predictions for each frame from the two-stream model differ as shown in 3.9.

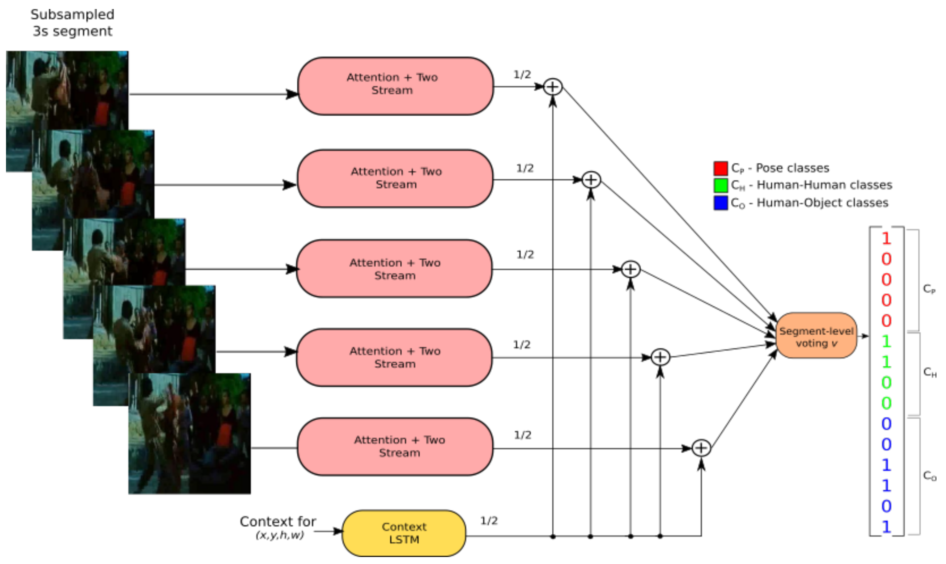


Figure 3.9: Our class score fusion model. We omit that each two-stream model (like in Fig. 3.2 also receives the optical flow volume corresponding to its frame according to Fig. 3.4 for simplicity. The predicted labels are merely illustrative.

This type of fusion has the advantage that it does not require training as it can only be applied at testing time. While this means it can more easily improved upon, it has the disadvantage that if one of the two models performs badly it might hinder the global performance. In this type of fusion there is no adaptive learning of which contributions should be weighted more nor does the option to learn more complex mixed features exist.

3.5.3 Two-Pass Testing Scheme

In the explanation of the two previous fusion methods an important issue was left unattended, which was how to generate context features at testing time. According to the example in Fig. 3.5 to generate context features for a test set we would need to have access to test labels.

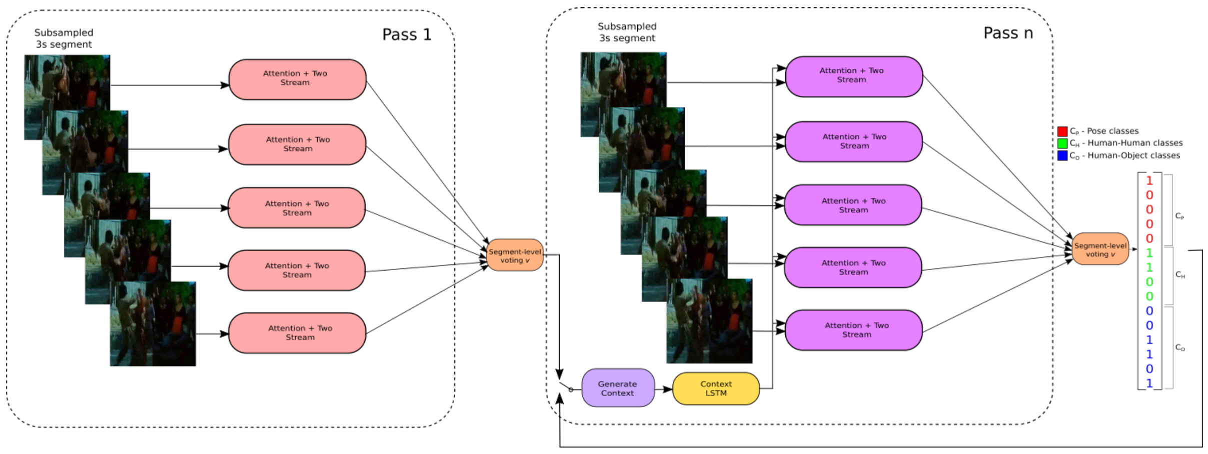


Figure 3.10: Our proposed two pass testing scheme, where the second or n further passes uses the concatenation fusion method. The switch illustrates that first the predictions of the two-stream models are used to generate labels which can then be used by our context models on a further passes. The predicted labels are merely illustrative.

While for the training and validation procedures we can generate context features for training the LSTM networks and we can use groundtruth test labels just to verify if context helps in an ideal situation, in a real scenario it is not possible to have groundtruth labels at test time. As such we propose this novel testing scheme as a way to solve this problem. As shown, this testing scheme can be used with the two previously discussed context fusion methods: concatenation fusion as shown in Fig. 3.8 and class score fusion as shown in Fig. 3.9.

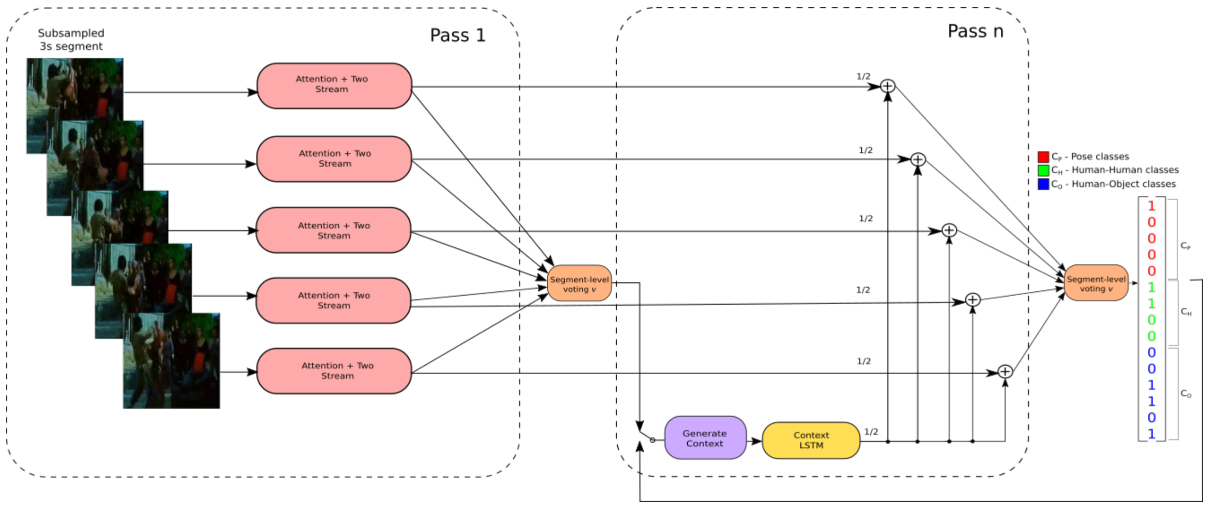


Figure 3.11: Our proposed two pass testing scheme, where the second or n further passes uses the class score fusion method. The predicted labels are merely illustrative.

This testing scheme can be divided into two conceptual stages. In the first stage or first pass we use the previously explained two-stream and attention methods to generate predictions on the test set. These predictions are then used to obtain the predicted labels which are then used to generate context features. These are the inputs to our context models for a second stage using one of the previously described fusion methods. As shown in Fig. 3.11 and Fig. 3.10 it would be possible to use the outputs of this procedure for yet another pass. While we test the results of doing this in chapter 5 we focus mostly on only performing two passes.

If the two-stream model in the first pass was able to perfectly predict the labels of the test set, we would be in the ideal situation we first described. The main idea is that of using the context models with their learned weights on the training set as a sort of correction of the predictions generated by the two-stream models.

4

Dataset and Metrics

Contents

4.1 Overview

Here we explain the AVA dataset and our partition of it (miniAVA) in more detail. We also explain the metrics used to evaluate the performance of our implementation on this dataset.

4.2 AVA

The AVA dataset was introduced as part of the ActivityNet [24] challenge as Task B - Spatio-Temporal Action Localization. This task was further divided into Task 1, which is the task discussed in this dissertation, where only computer vision approaches are accepted and Task 2, where additional provided features like audio can be used. This is a very recent dataset and is the most representative dataset we found for our task (multi-label multi-person spatiotemporal action detection and recognition).

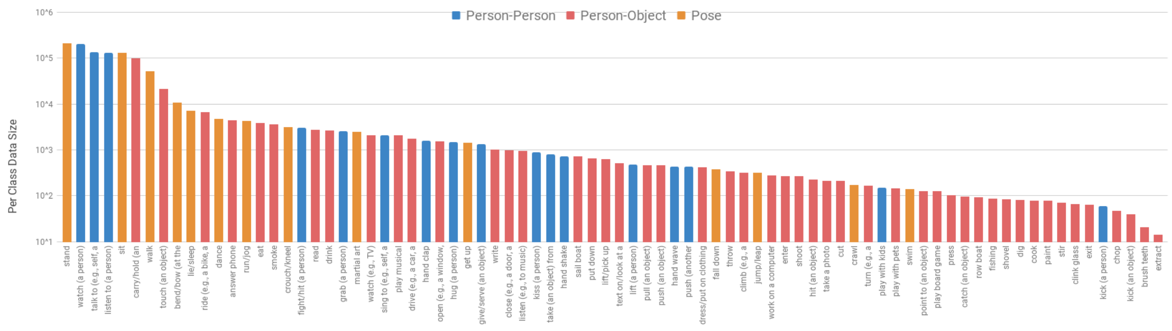


Figure 4.1: Long-tail [42] distribution of action classes in the AVA [35] dataset (all splits)

While most difficult datasets used for action recognition and detection [78] [57] [53] for the past years have allowed the field to progress, there are certain elements lacking in these datasets. AVA was created to fix this and be purposefully challenging to modern approaches, as shown in Table 2.1 and Table 1.1. AVA [35] introduces lacking elements such as having person-centric annotated actions with multiple people in videos and multiple action labels consistent over a series of frames. A key aspect is how the videos purposefully show actions that transition over time and must be recognized using temporal context. All actions are atomic actions and as such are mostly described with verb-only labels. Instead of having activity level labels such as *apply lipstick* in UCF101 [81]), AVA [35] has labels such as *stand* or *talk*. Another example of the focus in person-centric atomic actions is how no object annotations are provided and all human-object action labels are not object specific (e.g *touch (an object)*).

Another way in which the dataset is different, is how it is heavily imbalanced to emulate real data acquisition, following a Zipfs law [71] distribution as shown in Fig. 4.1. According to the nomenclature of [37], the AVA dataset has an intrinsic imbalance, that is, the action dataspace is itself naturally imbalanced.

4.2.1 AVA specifications

As for technical specifications, the annotations are done at a sampling frequency of 1 Hz, each sample corresponding to a keyframe. Every person is spatially localized using a groundtruth BB and labeled separately with attached labels corresponding to one action corresponding to pose, and there may be additional (0 to 3) actions corresponding to interactions with objects and then (0 to 3) additional actions corresponding to interactions with other people. An example of this is shown in Fig. 4.2.

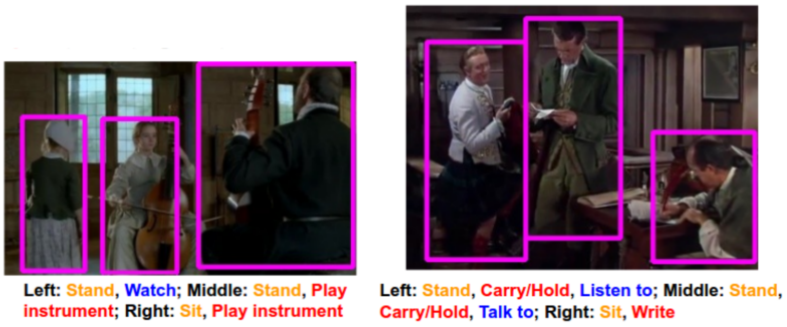

Left: Stand, Watch; Middle: Stand, Play instrument; Right: Sit, Play instrument
Left: Stand, Carry/Hold, Listen to; Middle: Stand, Carry/Hold, Talk to; Right: Sit, Write

Figure 4.2: Examples of AVA [35] labeling. In yellow: mutually exclusive pose actions (only 1 per BB), in blue: multi-label human-human actions (0-3 per BB), in red: multi-label human-object actions (0-3 per BB)

To provide temporal context, the labels are for short segments of $\pm 1.5s$ centered on the keyframes. There are 80 atomic actions labels and the dataset is sourced from the 15th to the 30th minute time intervals of 192 movies. These are called video *snippets*. At the sampling rate of 1Hz, this means $15 * 60 = 900$ keyframes for a single video snippet or 900 3s segments to classify. At an average rate of 30 FPS, this means 81,000 images per video without counting additional computed features like optical flow and our generated context features.

4.3 miniAVA

Due to the large size of the dataset we had to downsample it in order to train and test our approaches in a feasible amount of time. As we downsampled the dataset we had to keep two goals in mind. Firstly, we wanted to maintain continuity in the samples (i.e have segments from the same 15 minute snippets) as that would allow us to exploit longer time windows. Secondly, while we reduced the number of classes as shown in Table 4.1, we wanted to maintain a distribution that still roughly followed Zipfs law [71] for training, validation and test partitions as shown in Fig. 4.3. We took particular care to make sure there were at least 20 samples for all classes in the test set. As such we ended up with a partition that was roughly 10% of the original dataset size. The miniAVA classes are the ones that exist in the first 10% segments of the partitioned dataset.

Class Category	miniAVA(30)	AVA(80)	Mutually Exclusive?
Pose	10	14	Yes
Human-Object	12	49	No
Human-Human	8	17	No

Table 4.1: Distribution of class categories in the AVA [35] dataset vs miniAVA.

While normally the number of samples in the training set is larger than on the validation set, we kept all splits roughly equal in terms of samples, in order to try to keep a similar class distribution. The sampling was done from the first segments of the train, validation and test sets of the original AVA which, while not being an ideal approach, fortunately led to a similar distribution as seen in Fig. 4.3 and ensured a temporal continuity beyond the 3 second segments is maintained, which would otherwise be lost with random sampling.

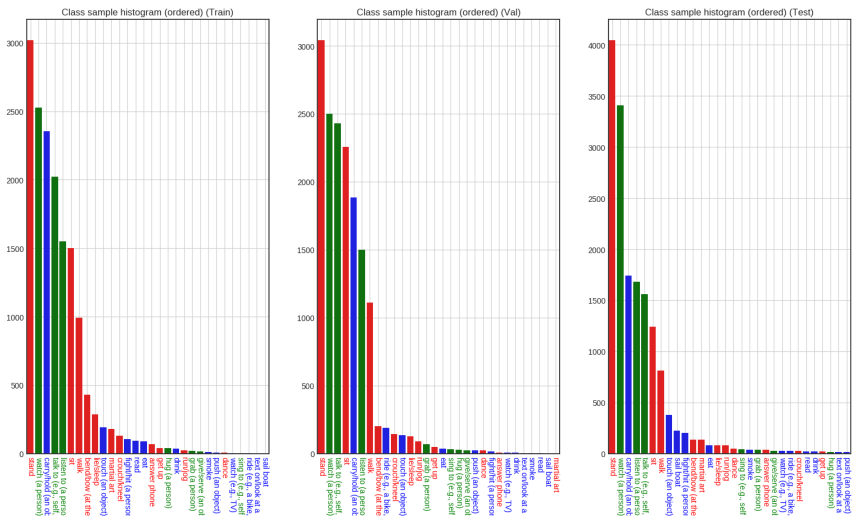


Figure 4.3: The distribution of the training, validation and test set of our partition of the AVA dataset. The colors of the labels reflect the type of action, we use alternative colors to the original dataset as in Fig. 4.2.

4.3.1 miniAVA Context

Given the discussion in section 3.4.1 it is relevant to ascertain how much information from neighbours of an actor over a time window can be learned, given that it is possible for segments to only have a single actor. It is also important to understand how much cross class information is possible to learn in our partition of the larger AVA. For the first concern, in Fig. 4.4 we can see that in the training and validation set most segments have only a single actor/BB. This means that the context network (one of the two models shown in section 3.4.2) is only learning relevant inter-person class relationship information from a small percentage of the dataset. The test dataset on the other hand seems to have a more balanced distribution in terms of BBs per frame, despite single actor segments still being heavily represented.

This means there are more than enough examples where our context approach can be applied and that we fortunately have an ideal challenging situation on which to train and then test our models.

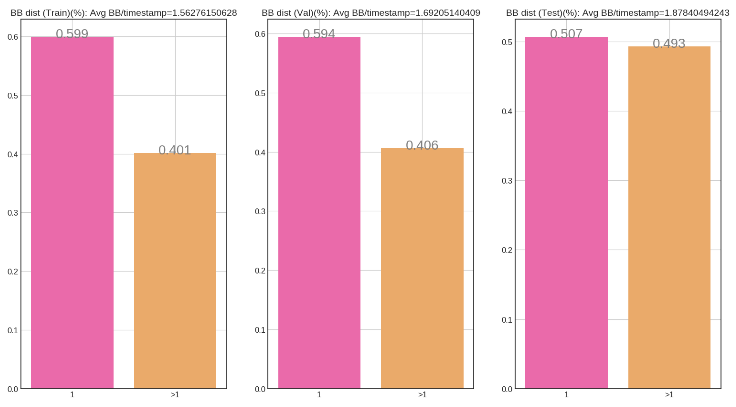


Figure 4.4: Relative number of BB's in segments. Notice that the average bounding box per video is lower on the training set than on the test set and that the test set has a larger amount of groups.

For the second concern, in Fig. 4.5 we show a matrix that depicts the normalized co-occurrence of classes in our test set. To obtain this matrix, for a given action (as a row) we count the actions of other actors in the same segment (along the columns), and then we normalize across the rows. Several interesting facts can be interpreted from this matrix that reveal the rich temporal context even in our small partition. For example, the abundance of data in the diagonal reveals that for many classes it is common for other actors to perform the same action in a segment. Additionally, there are some interesting interpretable positive co-occurrences, like *martial-arts* and *fight/hit a person* and some noticeably null co-occurrences like *stand* and *watch(e.g TV)* or *listen to (a person)* and *fight/hit a person*. However, some seem to be quirks of the particular distribution, like *sit* and *smoke* having the highest relative co-occurrence (nothing actually forces people to smoke and/or sit in a group), which we believe would be solved by using larger amounts of data. Th original AVA [35] authors also investigated this type of rich pair-wise cross-class structures using the NPMI index [8]. Note however that a context classifier which receives input from several other actors such as ours, could theoretically learn beyond pair-wise class relations which could not be depicted in this matrix or by the NPMI index [8].

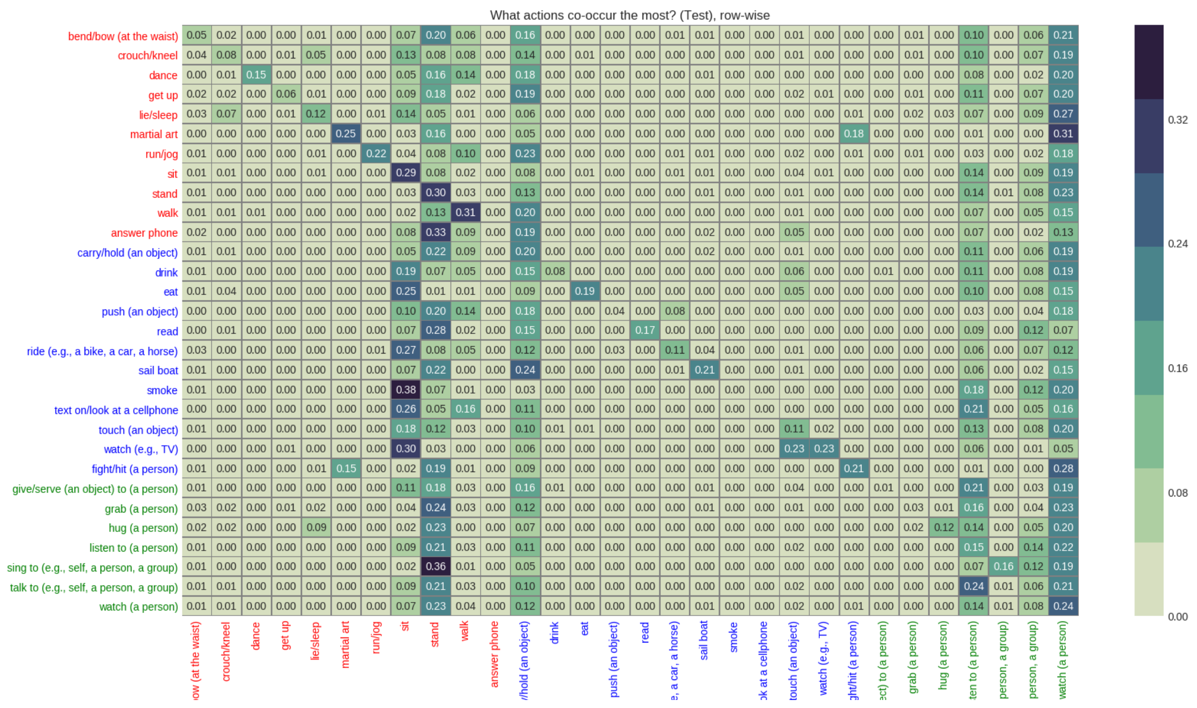


Figure 4.5: Co-occurence matrix of the miniAVA Test Set reveals a rich contextual environment even for our small dataset. For all frames of a given action (as a row) we count the actions of other actors in the same frames (across the columns), and then we normalize across the rows. In red are the pose classes, in blue are the human-object classes, in green are the human-human classes.

4.4 Metrics: mAP

Similarly to the original dataset, we only use mAP which is evaluated at a frame level and we focus on that metric. mAP is a metric to measure the accuracy for detection tasks. While several definitions of mAP exist [61] we will stick to that of the PASCAL [23] challenge. Before we introduce these concepts we must introduce the notion of precision, recall and Intersection over Union (IoU). This last concept measures how much overlap in area exists between two regions, normally a prediction and the groundtruth.

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

$$IoU = \frac{A(overlap)}{A(union)} \tag{4.3}$$

where TP are true positives, TN are true negatives, FP are false positives and FN are false negatives and a true positive only exists if its class matches the ground truth and $IoU \geq 0.5$. If we were to plot $Precision(Recall)$, the idea of AP can be conceptually viewed as finding the area under the curve of the

$Precision(Recall)$ curve. While that would imply computing the integral of the curve, [23] introduce the approximation of computing the average of precision at 11 recall levels (from 0.0 to 1.0):

$$AP = \frac{1}{11} \sum_{Recall_i} Precision(Recall_i) \tag{4.4}$$

Note that the average precision at recall i is taken to be the maximum precision measure at a recall greater or equal than recall i. This becomes a single value summarizing the shape of the precision-recall curve. For a problem with C classes mAP can be defined as:

$$mAP = \frac{1}{C} \sum_{i}^{C} AP_i \tag{4.5}$$

where AP_i is the average precision of class i as computed in 4.4.

5

Experimental Results

Contents

5.1 Overview

All the following experiments were carried out on systems with a single GTX 1080 Ti and 16-32 GB RAM, using the Keras [17] framework.

For CNNs and following the recommendations of [95] the training was done for 200 epochs, batch size is 32 with a learning rate of 0.001 decaying to 0.0001 after 80% of the epochs, while for the LSTMs the training time was for 100 epochs with a fixed learning rate of 0.001. All streams, apart from context LSTMs, were ResNet50 [39] networks which were initialized with weights from [28] for the respective RGB and OF networks trained on the UCF101 dataset. Context models were initialized using He initialization [38]. The input to the networks are 224x224 RGB images, stacks of 10 OF volumes previously discussed. For the results we use the same benchmarking tools as those provided for the AVA challenge with minor alterations only to obtain useful plots.

We report the results of several experiments on our miniAVA split of the original dataset that validate all the previously described augmentations in the implementation. This task, for all experiments, involves localizing the atomic actions in space and time, achieving the highest mAP (at 0.5 IoU) possible on 3s segments extracted around 1 FPS keyframes of 15-minute segments. For all experiments we round mAP to 2 decimal places.

While we discuss the results for all performed experiments, for the sake of brevity we only show individual graphs, such as AP for individual classes or confusion matrices, for certain experiments where they aid our discussion.

5.2 Baseline

Description: After extracting optical flow using the TV-L1 [103] algorithm, for our first experiment, we trained two separate ResNet50 [39] networks, one on RGB frames without any attention mechanisms and one on the extracted optical flow volumes, also without any attention mechanisms. Then we fused these two networks in a two-stream architecture using concatenation fusion and fine-tuned a FC layer to learn spatio-temporal features as shown in Fig. 3.2. The goal of this experiment is to establish a baseline analog to the one established in the AVA dataset (discussed in section 2.3.2), not only for the two-stream fusion but also for each of the individual streams. We do not use any context features in this experiment.

Model	mAP@0.5IoU
RGB	5.06%
Flow	5.85%
RGB + Flow	5.00%

Table 5.1: Baseline individual streams and their fusion.

Discussion: In Table 5.1 we show the results of this experiment. We note how using only Optical Flow performs better than RGB. This has been also found to be the case for some implementations of these types of networks [13] and is often due to the fact that certain actions have very clear motion patterns and as shown in Fig. 3.4 the lack of background noise may aid the classifier. Furthermore, we highlight how the baseline of the fusion is lower than both of the fused streams, which suggests that the spatiotemporal features being learned are not properly using complementary information from both streams, a result which we improve upon in further experiences. Lastly, we note how in a similar manner to the original AVA dataset the results are not very impressive (see Table 2.1). We believe that our small partition is even more challenging as we use less data and the chance of overfitting a small imbalanced dataset is higher.

5.3 Attention on individual streams

Description: This experiment can be divided into two parts: testing attention filtering applied on RGB frames and testing attention filtering applied on Optical Flow. For the first part we trained individual streams on the outputs of crop filtering, GBB filtering and fovea filtering, while for the second part we trained an individual stream on cropped optical flow. The goal of this experiment is to ascertain whether attention filtering applied as a pre-processing step for the individual streams can improve their respective baselines or not.

5.3.1 Attention on RGB streams

Model	mAP@0.5IoU
RGB	5.06%
RGB + GBB	5.63%
RGB + Crop	5.19%
RGB + Fovea	5.12%

Table 5.2: Attention filtering results on individual RGB streams vs baseline (RGB).

Discussion: In Table 5.2 we show the results of this experiment. Two main conclusions can be drawn from these results. One is that the use of all pre-filtering attention mechanisms improve results. The second is that the filtering techniques we hypothesize would lead to the networks learning artificial edges (i.e all except fovea filter) perform best. We think this is due to the fact that these artificial edges seem to be contributing to more accurate prediction of certain over represented classes, particularly *stand*, which is the most common class.

5.3.2 Attention on Optical Flow streams

Model	mAP@0.5IoU
Flow	5.85%
Flow + Crop	5.90%

Table 5.3: Attention filtering results on individual flow streams vs baseline (flow).

Discussion: In Table 5.3 we show the results of this experiment. We note that while we can see a very small improvement, that may also be due to certain artificial edges. However, the improvement of this method compared to the baseline is much smaller than in the RGB case, which leads us to believe that pre-filtering attention mechanisms for Optical Flow not only are not worthy to use but that their score is truly being extracted from motion features rather than introduced artificial edges.

5.4 Two-Stream Fusion

Description: For this experiment we fuse paired combinations of the streams trained on pre-filtered inputs from the previous experiment. We fuse them in the same fashion as shown before in Fig. 3.2. The Keras [17] framework defines a Merge Layer where it is possible to specify a concatenation option [1]. We did not fuse RGB and cropped flow as we believe the improvement on the baseline would be minimal, and we prioritized RGB attention filters over flow attention filters due to the results of the previous experiments. The goal of this experiment is to find out if the features learned from the RGB stream trained with attention filters and the features learned from the OF stream trained with attention filters when fused into spatiotemporal filters, help to improve the original two-stream baseline shown before.

mAP@0.5IoU	Flow	Flow + Crop
RGB + GBB	3.59%	4.16%
RGB + Crop	5.01%	5.06%
RGB + Fovea	**5.94**%	4.95%

Table 5.4: Testing of several combinations of streams and their respective attention filters

Discussion: In Table 5.4 we show the results of this experiment. The first result is that cropped flow seems to worsen results when fused with all other streams except the cropped RGB stream, which is an interesting result that suggests some synergy in the learned features. We also note how the fovea filter performs better than all others when fused with unfiltered flow and that it is the only two-stream approach that improves on all previous experiments. We hypothesize that this might be due to the fact that the artificial edges introduced by the other filters would harm their performance when later merged with flow features.

[1] https://keras.io/layers/merge/

However, upon analysis we conclude that all two-stream approaches seem to be fitting the distribution and learning only a few relevant classes. The two-stream approaches using fovea filtering seems to guess more samples than the others in a few of these small represented classes and in a certain way, we can confirm our hypothesis: while the fusion of motion features did not provide the large improvement we were expecting, it seemed that the features in the approaches with artificial edges are not complementary with the motion features of the Optical Flow stream.

AP@0.5IoU	Stand	Listen	Touch (an object)
Flow + (RGB + GBB)	53.3%	0.0%	0.0%
Flow + (RGB + Crop)	63.3%	0.0%	0.0%
Flow + (RGB + Fovea)	63.3%	28.4%	6.1%

Table 5.5: AP on some noteworthy classes that illustrate why the fovea filter seems to perform best.

Nonetheless, while this analysis provides valid hints the true validity of our conclusions should be further investigated by themselves on a more balanced dataset in more detail where their results are not as impacted by the heavy class imbalance.

5.5 Testing Context by itself

Description: In this experience we analyze how well the context features and the context architectures perform by themselves without fusion with the previous two-stream models. This has the advantage that the context models we tested were much less computationally heavy and as such we could test the architectures more extensively. The experiment is separated in two parts. First, as a sanity check, we try a simple MLP [64] model with the same receptive field as the LSTM models we proposed to confirm that the LSTM [41] models are superior at encoding sequences. Then we try to find the best context parameters used to generate the context features, more specifically the number of neighbours and the time window as previously explained in section 3.4.1. Finally, we also test which is the best architecture between model A and model B by testing over several values of NHU. We tested larger versions of the best model A and the best model B (with more NHU) to see if the models required a larger number of units to learn the complexity of the data. For both our context models (model A and model B) there are two input LSTM layers, and whenever we refer to NHU we are referencing to the size of each of these LSTM layers. The second is that the size of the next layer (LSTM layer for model A or FC layer for model B) is always $\frac{NHU}{2}$. Our goal with this experiment is twofold. We want to find out which is the best LSTM architecture and which are the best context generation parameters (from a limited set) for context learning.

Model	NHU	T	Neighbours	mAP@0.5IoU
MLP	128	3	3	4.49%
LSTM A	128	3	3	5.04%
LSTM B	128	3	3	5.00%

Table 5.6: Comparison of an MLP model and our proposed LSTM models.

T	Model	32	64	128	256	512	1024	2048
3	A	5.07%	4.99%	5.04%	4.96%	4.80%	–	–
5	A	4.84%	5.01%	4.97%	4.98%	**5.08%**	5.09%	**5.11%**
10	A	4.97%	4.85%	5.00%	4.68%	5.04%	–	–
T	Model	32	64	128	256	512	1024	2048
3	B	4.89%	5.01%	5.00%	4.92%	**5.12%**	5.08%	5.09%
5	B	4.90%	4.93%	4.94%	4.95%	5.04%	–	–
10	B	4.76%	4.81%	4.90%	4.92%	4.97%	–	–

Table 5.7: Evaluation of the best context generation and model A vs model B architecture. All these results use $N = 3$ (i.e three closest neighbours). The columns are the results across several values of NHU.

Neighbours	Model	NHU	T	mAP@0.5IoU
1	A	512	5	4.94%
2	A	512	5	5.05%
3	A	512	5	**5.08%**
1	B	512	3	4.93%
2	B	512	3	2.48%
3	B	512	3	**5.12%**

Table 5.8: Evaluation of the best architectures in Table 5.7 (apart from the larger networks) with context generated as explained in section 3.4.1, with a different number of neighbours.

Discussion: In Table 5.6, 5.7 and 5.8, we show the results of this experiment, however, even before analyzing the results we can observe overall that these results are competitive with the two-stream results, despite this network being much less computationally heavy. We can explain this from the fact that, unlike what we will see in our context fusion experiment, we use test labels to generate context features for a test set, which is an idealized, unrealistic situation. Regardless of this since our study is a comparative one and to confirm a sequence learning model such as an RNN is the correct choice, we show the results of an MLP architecture and both LSTM models. Even without a large number of hidden units we can still show that both LSTM models outperform an MLP model. In Table 5.7 we show the results of varying how many timesteps are used (for looking back in the past and looking forward in the future) and varying NHU for model A and model B. Additionally, we show how the architectures perform for a larger NHU for their best time window, which is also advantageous for future experiments where we want to fuse these networks with the two-stream models that have FC layers with 1024 or 2048 hidden units.

We can draw several conclusions. The first one is that there is not a significant difference between the best model A architecture or the best model B architecture. However while model B is marginally better, model A seems to operate better with a slightly higher time window. Finally, from Table 5.8 we can see both architectures seem to have their best results for 3 nearest neighbours. Evaluation on a longer time window or more neighbours was not done since we believe actions longer than the time windows used are rare and so are complex interactions between more than 3 people.

5.6 Context Fusion

Description: In this experience we use the best results from the two previous experiments to test the architectures using concatenation fusion and class score fusion of context and the two-stream architectures as depicted abstractedly in Fig. 3.1 and each of these under a groundtruth scenario testing scheme or a realistic testing scheme. Given the results and discussion in our previous experiments we use LSTM model A for the context architecture and we use the two-stream with fovea filtering and optical flow with no filtering as the basis for this experiment in both class score fusion and concatenation fusion. The first of these testing schemes is an overly optimistic groundtruth scenario, which uses test labels (i.e groundtruth) to generate context and the other, which we call real-case scenario, uses the predictions generated by a first pass through the best of the two-stream architectures to generate context features for a final pass through the architecture as shown in Fig. 3.11 and 3.10. The goal of this experiment is to validate two parts of our method. Firstly we want to find out which context fusion method performs best between, and additionally, we want to confirm that our proposed two pass testing scheme improves the previous best two-stream CNN approach in a realistic scenario thus making our implementation possible to be used in the wild, where test labels are not provided.

5.6.1 Groundtruth Scenario - Using test labels to generate context

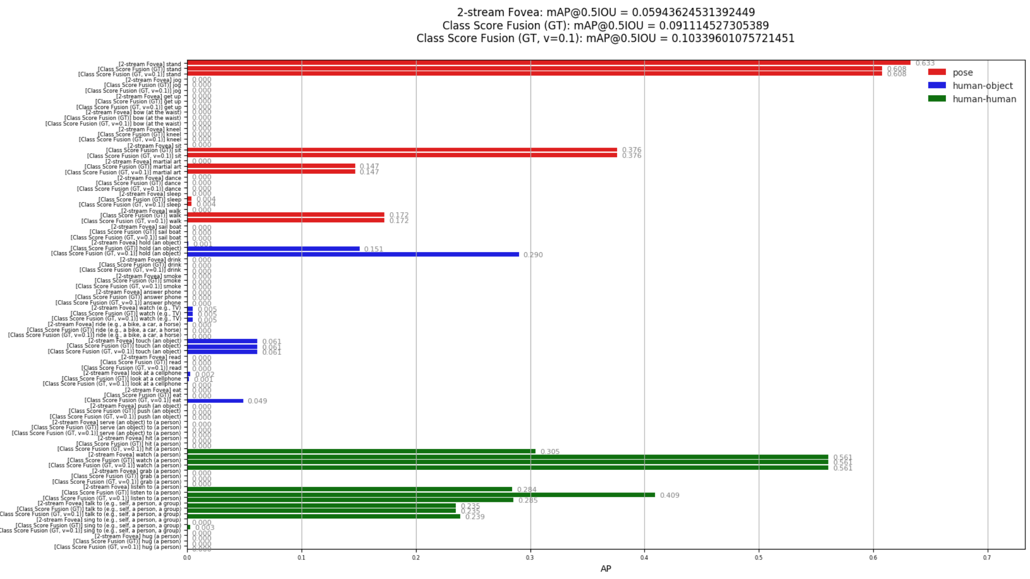


Figure 5.1: AP per class on our miniAVA split for the best two-stream approach (without context) and the class score fusion of that architecture with the best LSTM model as explained in 3.9. We show AP per class after changing the voting hyperparameter (which we discuss in the next experiment).

Model	mAP@0.5IoU
Best 2-stream (Fovea + Flow)	5.94%
(Groundtruth Context) Concatenation Fusion	5.92%
(Groundtruth Context) Class Score Fusion	**9.11%**

Table 5.9: Testing context fusion architectures under a groundtruth scenario. i.e using test labels to generate context. We can see that the results improve considerable when using class score fusion but not when using concatenation fusion.

Discussion: In Table 5.9 we show the results of this experiment. We also show the results of the best two-stream model with attention (fovea and flow) as what we want to improve on. We can see that the class score fusion largely outperforms concatenation fusion. We believe this may be because when concatenated, the internal feature representations of the LSTM models are too distinct from the two-stream spatiotemporal features for a FC layer to learn merged features. However, at a class score level the context predictions are able to correct the two-stream predictions. We can see that the class score fusion does indeed have a balancing impact, preventing the two-stream architecture from overfitting. We can see in Fig. 5.1 that it has a significantly higher score on several classes with a smaller representation in the original dataset than the two-stream approach. Note that while this situation is an idealized situation the context only use information of labels from neighbours in frames not the actors in the frames themselves and the context models were only meaningfully trained on a percentage of the training set where more than one person/BB per frame existed as depicted in 4.4.

5.6.2 Real Case Scenario - Two Pass Testing Scheme

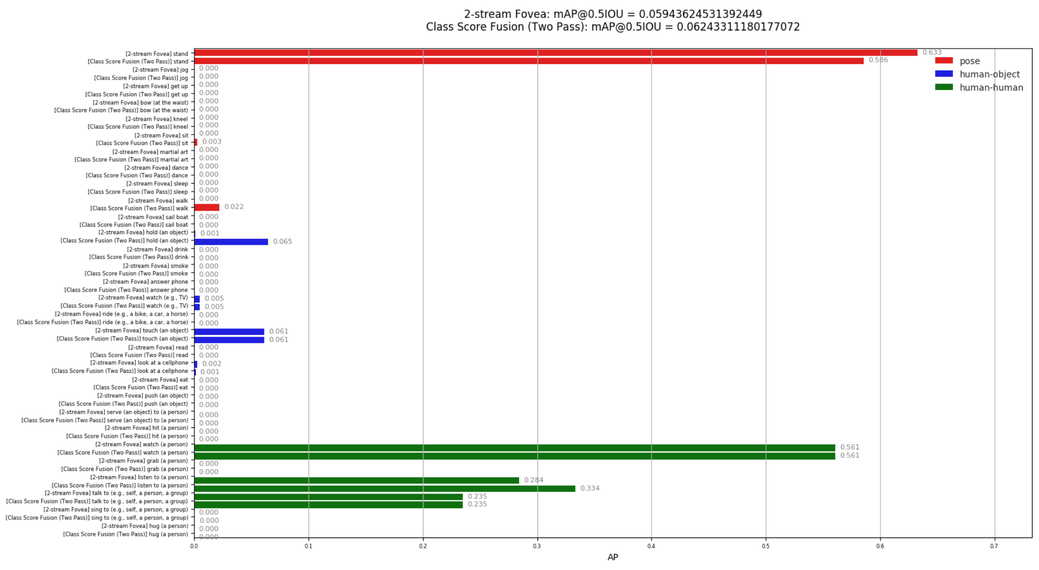


Figure 5.2: AP per class on our miniAVA split for the best two-stream approach (without context) and class score fusion with context. The difference from Fig. 5.1 is that the class score fusion uses our two pass testing scheme.

Model	mAP@0.5IoU
Best 2-stream (Fovea + Flow)	5.94%
(Two Pass) Concatenation Fusion	5.91%
(Two Pass) Class Score Fusion	**6.24%**
(Three Pass) Class Score Fusion	**6.28%**

Table 5.10: Testing context fusion architectures under a real case scenario, where we do not assume to have the test labels at testing time. As in Fig. 5.1 we show the results of the best two-stream approach and then the results of class score fusion with the two pass scheme demonstrated in Fig. 3.11. We also show the results of an additional pass.

Discussion: In Table 5.10 we show the results of this experiment. Similarly to the last experiment we can see that using concatenation fusion does not seem to improve the model and we believe the same reason applies. More interesting are the two results with class score fusion. Even without any test labels being used, only using the outputs of the previous best two-stream to generate context features, we can see that there is a small improvement of 0.30% mAP (1.24% mAP over the baseline). While the improvement may be considered small when comparing to the previous improvement of 3.16% mAP, consider that the context features in this case are being generated from a classifier with only 5.94% mAP while in the other case we could theorize an ideal classifier with 100% (which had a 4.11% mAP improvement over the baseline). We can also see in Fig. 5.2 that this improvement is due to the influence of context to force the model to make valid guesses on under represented small classes (like sit, walk, hold an object) at the expense of losing some Average Precision (AP) in the over-represented *stand* class, thus confirming the effect of negating overfitting to a certain extent we also witnessed in the last experiment. Additionally, using the results of this approach to yet generate more context features and testing again (i.e Three Pass in Table 5.10) seemed to yield only marginal improvements.

5.7 Voting Hyperparameter

Description: As described in section 3.2.2 our voting system needs a hyperparameter v to reach a segment-level consensus on the predictions. We test the optimization of this hyperparameter for two of the previous experiments on the validation set and then we use their best values on the test set. We test the best performing single stream attention filtering on an RGB stream (GBB) and the best performing context fusion method (for the groundtruth scenario) in this fashion. In Fig. 5.3 we show an example of finding the best voting hyperparameter for the RGB stream with GBB filtered inputs on the validation set, as this type of tuning on the test set is not a good practice. In all experiments besides this one this parameter was set to 0.4 as that is also the value adopted by the baseline implementation described in the original AVA[35]. Our goal is this experiment is to evaluate what impact this hyperparameter has on the results of the classifiers so that we can use the best hyperparameter for a respective approach.

Model	Set	$v = 0.4$	$v = 0.2$	$v = 0.1$
RGB + GBB	Validation	5.66%	**6.42%**	6.40%
(Groundtruth Context) Class Score Fusion	Validation	4.05%	4.54%	**4.60%**
(Groundtruth Context) Class Score Fusion	Test	9.11%	9.96%	**10.3%**
(Two Pass) Class Score Fusion	Test	6.24%	6.41%	**6.66%**

Table 5.11: Results of varying the voting hyperparameter for several approaches. Note that smaller values of this threshold tend to improve mAP as they allow for more under represented classes with lower score to be considered in the voting scheme.

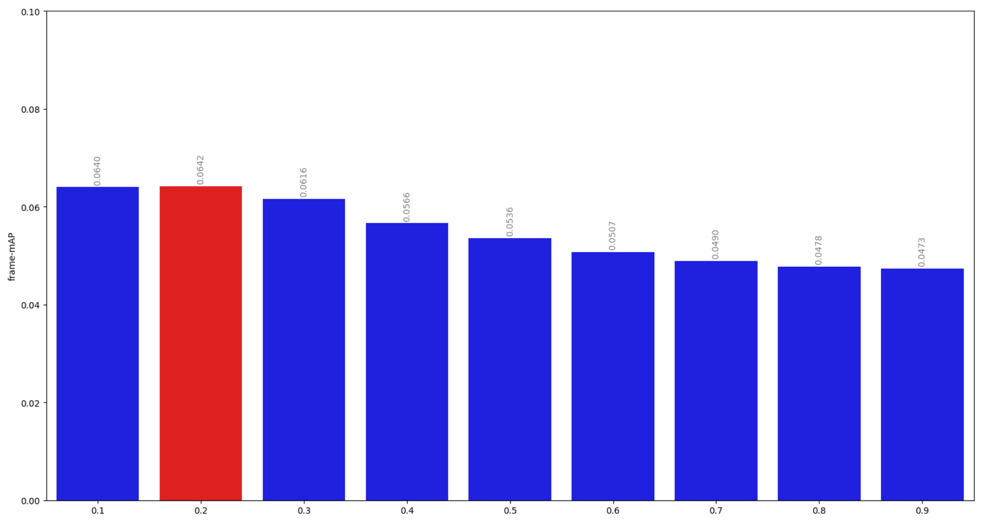


Figure 5.3: Results of an RGB stream trained with GBB attention filter with several values of the voting hyperparameter on the validation set. This is merely an illustrative example. As seen in table 5.11 other methods were tested in a similar way.

Discussion: In Table 5.11 we show the results of this experiment. From Fig. 5.1 we can see that lowering the voting threshold normally tends to make under represented classes achieve higher AP as this allows smaller values from the sigmoids to count as votes in the voting procedure without impacting the already predicted. This naturally leads to a higher mAP for smaller values of the voting hyperparameter. This is true for the two approaches we tested, one of which is our best fusion approach. This experiment serves only as a demonstration even though optimally we would fine-tune this parameter for all experiments.

5.8 Balancing

Description: For the final experiment we propose to evaluate the effects of training one of the RGB streams from a previous experiment on an artificially balanced dataset using oversampling. We chose this option as [11] mention it tends to be the most effective. A caveat is that simply applying naive oversampling on a multi-label dataset would lead to disrupting the distribution as mentioned by [82]. Therefore our oversampling method repeats all labels in the 3s segments that contain under represented classes. This means that over represented classes will probably also be repeated but the relative imbalance will be reduced. Our goal is to find out whether the oversampling helps the training process to improve their score on under represented classes or not. While we expect the overall score might not improve as the test split is itself imbalanced we expect the AP of certain under represented classes will.

Model	mAP@0.5IoU
RGB + GBB original dataset	5.63%
RGB + GBB oversampling	4.71%

Table 5.12: mAP Results on balancing using oversampling.

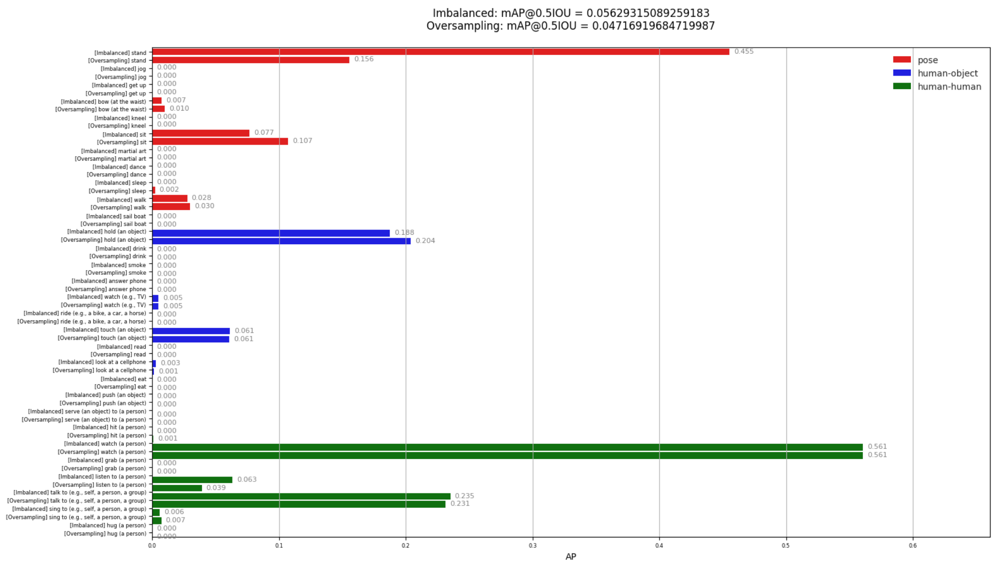
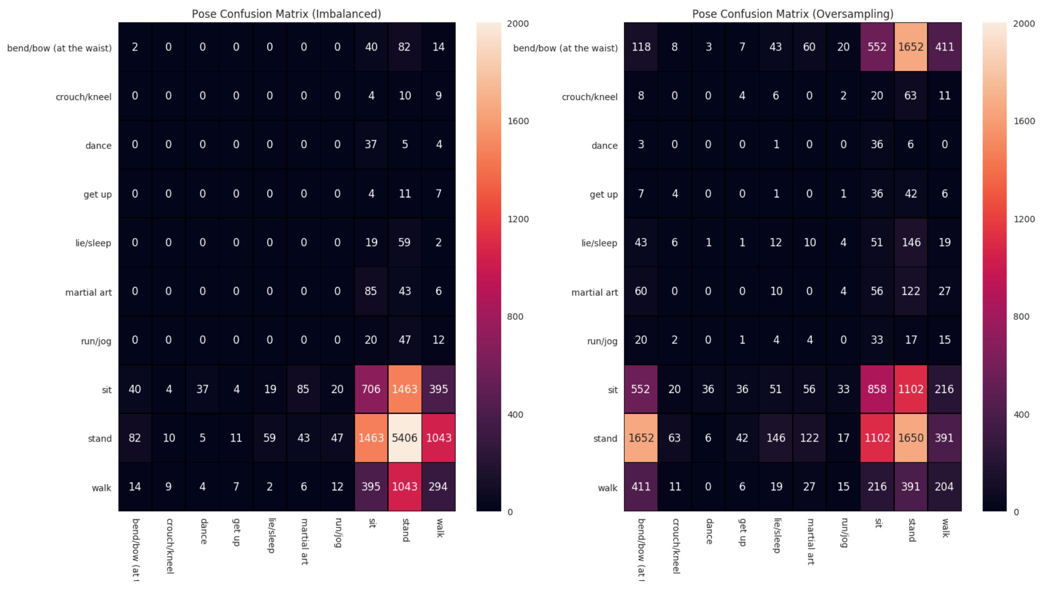


Figure 5.4: AP per class on the miniAVA split for the balancing and oversampling experiment. The model used for the oversampling experiment is the same and the repeated frames are all GBB filtered so that the only difference between the two is the oversampling strategy.



Pose Confusion Matrix (Imbalanced)

	bend/bow (at t	crouch/kneel	dance	get up	lie/sleep	martial art	run/jog	sit	stand	walk
bend/bow (at the waist)	2	0	0	0	0	0	0	40	82	14
crouch/kneel	0	0	0	0	0	0	0	4	10	9
dance	0	0	0	0	0	0	0	37	5	4
get up	0	0	0	0	0	0	0	4	11	7
lie/sleep	0	0	0	0	0	0	0	19	59	2
martial art	0	0	0	0	0	0	0	85	43	6
run/jog	0	0	0	0	0	0	0	20	47	12
sit	40	4	37	4	19	85	20	706	1463	395
stand	82	10	5	11	59	43	47	1463	5406	1043
walk	14	9	4	7	2	6	12	395	1043	294

Pose Confusion Matrix (Oversampling)

	bend/bow (at t	crouch/kneel	dance	get up	lie/sleep	martial art	run/jog	sit	stand	walk
bend/bow (at the waist)	118	8	3	7	43	60	20	552	1652	411
crouch/kneel	8	0	0	4	6	0	2	20	63	11
dance	3	0	0	0	1	0	0	36	6	0
get up	7	4	0	0	1	0	1	36	42	6
lie/sleep	43	6	1	1	12	10	4	51	146	19
martial art	60	0	0	0	10	0	4	56	122	27
run/jog	20	2	0	1	4	4	0	33	17	15
sit	552	20	36	36	51	56	33	858	1102	216
stand	1652	63	6	42	146	122	17	1102	1650	391
walk	411	11	0	6	19	27	15	216	391	204

Figure 5.5: Confusion matrices for the pose classes in the balancing experiment.

Discussion: In Table 5.12 and Fig. 5.4 and 5.5 we show the results of this experiment. In this final experiment we want to evaluate what impact of balancing via oversampling can have in the classifiers. Since with our method of oversampling the training set can get considerably larger and thus training time becomes considerably longer, we chose to use only a demonstrative study on a single RGB stream, the same as used for demonstration purposes in the last experiment (GBB filter). While the overall mAP results shown in Table 5.12 seem to indicate that the approach was not successful there are two important observations to make that support that this result is misleading. Firstly, if we analyze Fig. 5.4 we can see that while oversampling loses AP in more common classes like stand (refer to 4.3 to see which classes are more frequent) or talk to, we can see that for many others it outperforms the classifier without oversampling (for example, sit, hold and object, walk and bow at the waist). Interestingly enough, these improvements happen on all three types of classes which validates our approach even further. If we focus only on the pose type, since they are mutually exclusive we can easily plot a confusion matrix for each approach as shown in Fig. 5.5. We can see that while the classifier trained with oversampling seems to make more mistakes for the most common class which is stand and this may justify the loss in mAP, the distribution of the predictions is less focused on that single predominant class, and the classifier can generalize better to other classes.

Therefore we can see that the balancing is indeed working as intended, however, since the testing set itself is imbalanced the reported global results are that the classifier performs more poorly.

6 Conclusion

Contents

6.1 Conclusions and Future Work

Our objective was to implement an action detection architecture which was not computationally heavy, provide alternative solutions to the spatiotemporal action detection task and to show valid augmentations of the base two-stream approaches [29] using attention filters and context features (i.e class and actor relationships). We described several architectures, from the initial two stream architecture using only attention filtering as shown in Fig. 3.2 to our final approach shown in Fig. 3.9 where we fuse our custom context features with the two-stream approach.

We believe we have achieved our objectives as we demonstrated improvements on the baseline for each of the proposed augmentations on the aforementioned architectures. Although these results may be deemed provisional, as due to computation resource constraints we had to make our own split of the dataset, we believe it is necessary to conduct further investigation and further testing to truly validate many of them. Nonetheless, we believe our work stands on its own as a valid proof of concept for further research on the topic of using alternative features such as context and pre-processing attention filtering in the field of action detection.

An intuitive future improvement would be to either test our approaches on the larger AVA dataset or to use our approaches to augment other existing more powerful state of the art architectures, but for both of these we would need more computational resources. Future work that would not require this would be to test our attention filtering results on the UCF101-24 [81] dataset thus allowing us to test them on a balanced, single-label dataset (as shown in 6.1) which would make it easier to analyze certain aspects of the architectures.

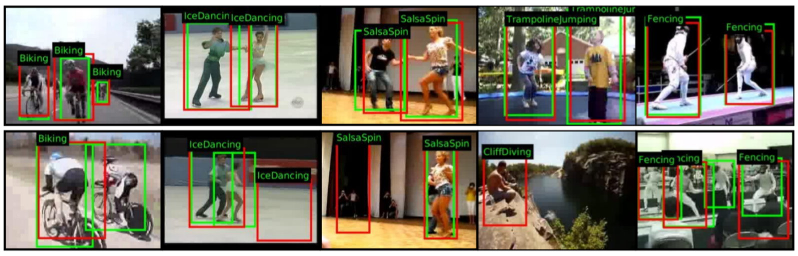


Figure 6.1: Example of action detection labels on UCF101-24. Image adapted from `https://www.researchgate.net/figure/Action-detection-localisation-results-on-UCF101-Ground-truth-boxes-are-in-green_fig4_305857635`.

This last one is particularly attractive as the UCF101-24 partition of the UCF101 dataset is much smaller than the AVA dataset and even than our miniAVA partition and besides being balanced, our transfer learning approaches would be much more effective as we have weights for the original UCF101. Additionally, the UCF101 dataset is older than the AVA dataset and many more results exist for it. Another goal would be to make the whole architecture and pipeline more end to end without requiring so many sequential computational steps, which make its implementation more difficult.

Bibliography

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[2] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Apostol (Paul) Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. In *arXiv:1609.08675*, 2016.

[3] Samir Al-Stouhi and Chandan K. Reddy. Transfer learning for class imbalance problems with inadequate data. *Knowl. Inf. Syst.*, 48(1):201–228, July 2016. ISSN 0219-1377. doi: 10.1007/s10115-015-0870-3.

[4] Ana Filipa Almeida, Rui Figueiredo, Alexandre Bernardino, and José Santos-Victor. Deep networks for human visual attention: A hybrid model using foveal vision. In Anibal Ollero, Alberto Sanfeliu, Luis Montano, Nuno Lau, and Carlos Cardeira, editors, *ROBOT 2017: Third Iberian Robotics Conference*, pages 117–128, Cham, 2018. Springer International Publishing. ISBN 978-3-319-70836-2.

[5] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, abs/1803.01271, 2018.

[6] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, March 1994. ISSN 1045-9227. doi: 10.1109/72.279181.

[7] H. Bilen, B. Fernando, E. Gavves, and A. Vedaldi. Action recognition with dynamic image networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2018. ISSN 0162-8828. doi: 10.1109/TPAMI.2017.2769085.

[8] G. Bouma. Normalized (pointwise) mutual information in collocation extraction. In *From Form to Meaning: Processing Texts Automatically, Proceedings of the Biennial GSCL Conference 2009*, volume Normalized, pages 31–40, Tübingen, 2009.

[9] C. Bregler. Learning and recognizing human dynamics in video sequences. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 568–574, June 1997. doi: 10.1109/CVPR.1997.609382.

[10] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In Tomás Pajdla and Jiří Matas, editors, *Computer Vision - ECCV 2004*, pages 25–36, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-24673-2.

[11] Mateusz Buda, Atsuto Maki, and Maciej A. Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *CoRR*, abs/1710.05381, 2017.

[12] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, April 1983. ISSN 0090-6778. doi: 10.1109/TCOM.1983.1095851.

[13] João Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4733, 2017.

[14] Yu-Wei Chao, Sudheendra Vijayanarasimhan, Bryan Seybold, David A. Ross, Jia Deng, and Rahul Sukthankar. Rethinking the faster r-cnn architecture for temporal action localization. *CoRR*, abs/1804.07667, 2018.

[15] Francisco Charte, Antonio Rivera, María José del Jesus, and Francisco Herrera. A first approach to deal with imbalance in multi-label datasets. In Jeng-Shyang Pan, Marios M. Polycarpou, Michał Woźniak, André C. P. L. F. de Carvalho, Héctor Quintián, and Emilio Corchado, editors, *Hybrid Artificial Intelligent Systems*, pages 150–160, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-40846-5.

[16] Guangchun Cheng, Yiwen Wan, Abdullah N. Saudagar, Kamesh Namuduri, and Bill P. Buckles. Advances in human action recognition: A survey. *CoRR*, abs/1501.05964, 2015.

[17] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[18] Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22, 2004.

[19] Yann Le Cun. A theoretical framework for back-propagation, 1988.

[20] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):677–691, April 2017. ISSN 0162-8828. doi: 10.1109/TPAMI.2016.2599174.

[21] Alexey Dosovitskiy, Philipp Fischery, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 2758–2766, Washington, DC, USA, 2015. IEEE Computer Society. ISBN 978-1-4673-8391-2. doi: 10.1109/ICCV.2015.316.

[22] Charles Elkan. The foundations of cost-sensitive learning. In *In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 973–978, 2001.

[23] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2): 303–338, Jun 2010. ISSN 1573-1405. doi: 10.1007/s11263-009-0275-4.

[24] Bernard Ghanem Fabian Caba Heilbron, Victor Escorcia and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–970, 2015.

[25] Lijie Fan, Wen-bing Huang, Chuang Gan, Stefano Ermon, Boqing Gong, and Junzhou Huang. End-to-end learning of motion representation for video understanding. *CoRR*, abs/1804.00413, 2018.

[26] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In Josef Bigun and Tomas Gustavsson, editors, *Image Analysis*, pages 363–370, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-45103-7.

[27] C. Feichtenhofer, A. Pinz, and R. P. Wildes. Spatiotemporal multiplier networks for video action recognition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7445–7454, July 2017. doi: 10.1109/CVPR.2017.787.

[28] Christoph Feichtenhofer, Axel Pinz, and Richard P. Wildes. Spatiotemporal residual networks for video action recognition. In *NIPS*, 2016.

[29] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. *CoRR*, abs/1604.06573, 2016.

[30] Christoph Feichtenhofer, Axel Pinz, Richard P. Wildes, and Andrew Zisserman. What have we learned from deep representations for action recognition? *CoRR*, abs/1801.01415, 2018.

[31] Alberto Fernández, Sara del Río, Nitesh V. Chawla, and Francisco Herrera. An insight into imbalanced big data classification: outcomes and challenges. *Complex & Intelligent Systems*, 3(2): 105–120, Jun 2017. ISSN 2198-6053. doi: 10.1007/s40747-017-0037-9.

[32] Rohit Girdhar, Deva Ramanan, Abhinav Gupta, Josef Sivic, and Bryan Russell. ActionVLAD: Learning spatio-temporal aggregation for action classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, Honolulu, United States, 2017. Project page: https://rohitgirdhar.github.io/ActionVLAD/.

[33] Melvyn A. Goodale and A. David. Milner. Separate visual pathways for perception and action. *Trends in Neurosciences*, 15(1):20–25, 1992.

[34] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

[35] Chunhui Gu, Chen Sun, Sudheendra Vijayanarasimhan, Caroline Pantofaru, David A. Ross, George Toderici, Yeqing Li, Susanna Ricco, Rahul Sukthankar, Cordelia Schmid, and Jitendra Malik. AVA: A video dataset of spatio-temporally localized atomic visual actions. *CoRR*, abs/1705.08421, 2017.

[36] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? *CoRR*, abs/1711.09577, 2017.

[37] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, Sept 2009. ISSN 1041-4347. doi: 10.1109/TKDE.2008.239.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1026–1034, Washington, DC, USA, 2015. IEEE Computer Society. ISBN 978-1-4673-8391-2. doi: 10.1109/ICCV.2015.123.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[40] Samitha Herath, Mehrtash Tafazzoli Harandi, and Fatih Porikli. Going deeper into action recognition: A survey. *CoRR*, abs/1605.04988, 2016.

[41] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

[42] Grant Van Horn and Pietro Perona. The devil is in the tails: Fine-grained classification in the wild. *CoRR*, abs/1709.01450, 2017.

[43] Rui Hou, Chen Chen, and Mubarak Shah. Tube convolutional neural network (T-CNN) for action detection in videos. *CoRR*, abs/1703.10664, 2017.

[44] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *CoRR*, abs/1612.01925, 2016.

[45] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[46] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *CoRR*, abs/1702.05659, 2017.

[47] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, Jan 2013. ISSN 0162-8828. doi: 10.1109/TPAMI.2012.59.

[48] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[49] Yu-Gang Jiang, Zuxuan Wu, Jun Wang, Xiangyang Xue, and Shih-Fu Chang. Exploiting feature and class relationships in video categorization with regularized deep neural networks. *CoRR*, abs/1502.07209, 2015.

[50] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3304–3311, June 2010. doi: 10.1109/CVPR.2010.5540039.

[51] Soo-Min Kang and Richard P. Wildes. Review of action recognition and detection methods. *CoRR*, abs/1610.06906, 2016.

[52] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[53] Will Kay, João Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset. *CoRR*, abs/1705.06950, 2017.

[54] Andrea Vedaldi Ken Chatfield, Victor Lempitsky and Andrew Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In *Proceedings of the British Machine Vision Conference*, pages 76.1–76.12. BMVA Press, 2011. ISBN 1-901725-43-X. http://dx.doi.org/10.5244/C.25.76.

[55] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[56] Yu Kong and Yun Fu. Human action recognition and prediction: A survey. *CoRR*, abs/1806.11230, 2018.

[57] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: a large video database for human motion recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011.

[58] Quoc V. Le, Jiquan Ngiam, Adam Coates, Abhik Lahiri, Bobby Prochnow, and Andrew Y. Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 265–272, USA, 2011. Omnipress. ISBN 978-1-4503-0619-5.

[59] Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. Understanding the disharmony between dropout and batch normalization by variance shift. *CoRR*, abs/1801.05134, 2018.

[60] Shiyu Liang and R. Srikant. Why deep neural networks? *CoRR*, abs/1610.04161, 2016.

[61] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[62] Thomas B. Moeslund, Adrian Hilton, and Volker Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104:90–126, 2006.

[63] Mathew Monfort, Bolei Zhou, Sarah Adel Bargal, Alex Andonian, Tom Yan, Kandan Ramakrishnan, Lisa M. Brown, Quanfu Fan, Dan Gutfreund, Carl Vondrick, and Aude Oliva. Moments in time dataset: one million videos for event understanding. *CoRR*, abs/1801.03150, 2018.

[64] Fionn Murtagh. Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5): 183 – 197, 1991. ISSN 0925-2312. doi: https://doi.org/10.1016/0925-2312(91)90023-5.

[65] Joe Yue-Hei Ng, Matthew J. Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. *CoRR*, abs/1503.08909, 2015.

[66] Phuc Nguyen, Ting Liu, Gautam Prasad, and Bohyung Han. Weakly supervised action localization by sparse temporal pooling network. *CoRR*, abs/1712.05080, 2017.

[67] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 1717–1724, Washington, DC, USA, 2014. IEEE Computer Society. ISBN 978-1-4799-5118-5. doi: 10.1109/CVPR.2014.222.

[68] Xiaojiang Peng and Cordelia Schmid. Multi-region two-stream r-cnn for action detection. In *European Conference on Computer Vision*, pages 744–759. Springer, 2016.

[69] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the fisher kernel for large-scale image classification. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 143–156, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15561-1.

[70] Ronald Walter Poppe. A survey on vision-based human action recognition. *Image and vision computing*, 28(6):976–990, 6 2010. ISSN 0262-8856. doi: 10.1016/j.imavis.2009.11.014. 10.1016/j.imavis.2009.11.014.

[71] David M. W. Powers. Applications and explanations of zipf's law. In *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning*, NeMLaP3/CoNLL '98, pages 151–160, Stroudsburg, PA, USA, 1998. Association for Computational Linguistics. ISBN 0-7258-0634-6.

[72] Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, and Michael L. Littman. Activity recognition from accelerometer data. In *Proceedings of the 17th Conference on Innovative Applications of Artificial Intelligence - Volume 3*, IAAI'05, pages 1541–1546. AAAI Press, 2005. ISBN 1-57735-236-x.

[73] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama,

and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[74] Douglas A. Reynolds. Gaussian mixture models. In *Encyclopedia of Biometrics*, 2009.

[75] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115 (3):211–252, Dec 2015. ISSN 1573-1405. doi: 10.1007/s11263-015-0816-y.

[76] Richard A. Schermerhorn. Midwest and its children. by roger g. barker and herbert f. wright. evanston, illinois: Row, peterson and company, 1955. 532 pp. 7.50. illustrated. *Social Forces*, 34 (4):390–391, 1956. doi: 10.2307/2573693.

[77] Tobias Senst, Volker Eiselein, and Thomas Sikora. Ii-lk – a real-time implementation for sparse optical flow. In Aurélio Campilho and Mohamed Kamel, editors, *Image Analysis and Recognition*, pages 240–249, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-13772-3.

[78] Gunnar A. Sigurdsson, Gül Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. *CoRR*, abs/1604.01753, 2016.

[79] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[80] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pages 568–576, Cambridge, MA, USA, 2014. MIT Press.

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

[82] Konstantin Sozykin, Stanislav Protasov, Adil Khan, Rasheed Hussain, and Jooyoung Lee. Multilabel class-imbalanced action recognition in hockey videos via 3d convolutional neural networks. *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 146–151, 2018.

[83] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[84] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[85] Javier Sánchez Pérez, Enric Meinhardt-Llopis, and Gabriele Facciolo. TV-L1 Optical Flow Estimation. *Image Processing On Line*, 3:137–150, 2013. doi: 10.5201/ipol.2013.26.

[86] An Tran and Loong Fah Cheong. Two-stream flow-guided convolutional attention networks for action recognition. *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 3110–3119, 2017.

[87] Du Tran, Lubomir D. Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. C3D: generic features for video analysis. *CoRR*, abs/1412.0767, 2014.

[88] Du Tran, Jamie Ray, Zheng Shou, Shih-Fu Chang, and Manohar Paluri. Convnet architecture search for spatiotemporal feature learning. *CoRR*, abs/1708.05038, 2017.

[89] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. *CoRR*, abs/1711.11248, 2017.

[90] V. Javier Traver and Alexandre Bernardino. A review of log-polar imaging for visual perception in robotics. *Robotics and Autonomous Systems*, 58:378–398, 2010.

[91] Rajesh Kumar Tripathi, Anand Singh Jalal, and Subhash Chand Agrawal. Suspicious human activity recognition: a review. *Artificial Intelligence Review*, 50(2):283–339, Aug 2018. ISSN 1573-7462. doi: 10.1007/s10462-017-9545-7.

[92] Andrea Vedaldi and Karel Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM International Conference on Multimedia*, MM '15, pages 689–692, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3459-4. doi: 10.1145/2733373.2807412.

[93] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *Proceedings of the 2013 IEEE International Conference on Computer Vision*, ICCV '13, pages 3551–3558, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-1-4799-2840-8. doi: 10.1109/ICCV.2013.441.

[94] L. Wang, Y. Qiao, and X. Tang. Mining motion atoms and phrases for complex action recognition. In *2013 IEEE International Conference on Computer Vision*, pages 2680–2687, Dec 2013. doi: 10.1109/ICCV.2013.333.

[95] Limin Wang, Yuanjun Xiong, Zhe Wang, and Yu Qiao. Towards good practices for very deep two-stream convnets. *CoRR*, abs/1507.02159, 2015.

[96] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks for action recognition in videos. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[97] Michael Wray, Davide Moltisanti, and Dima Damen. Towards an unequivocal representation of actions. *CoRR*, abs/1805.04026, 2018.

[98] Zuxuan Wu, Yu-Gang Jiang, Xi Wang, Hao Ye, Xiangyang Xue, and Jun Wang. Fusing multi-stream deep networks for video classification. *CoRR*, abs/1509.06086, 2015.

[99] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning for video understanding. *CoRR*, abs/1712.04851, 2017.

[100] Huijuan Xu, Abir Das, and Kate Saenko. R-C3D: region convolutional 3d network for temporal activity detection. *CoRR*, abs/1703.07814, 2017.

[101] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.

[102] Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas J. Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579, 2015.

[103] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime tv-l1 optical flow. In *Proceedings of the 29th DAGM Conference on Pattern Recognition*, pages 214–223, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74933-2.

[104] Bowen Zhang, Limin Wang, Zhe Wang, Yu Qiao, and Hanli Wang. Real-time action recognition with enhanced motion vector cnns. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2718–2726, 2016.

[105] Min-Ling Zhang, Yu-Kun Li, and Xu-Ying Liu. Towards class-imbalance aware multi-label learning. In *IJCAI*, 2015.

[106] Yue Zhao, Yuanjun Xiong, Limin Wang, Zhirong Wu, Xiaoou Tang, and Dahua Lin. Temporal action detection with structured segment networks. In *ICCV*, 2017.

[107] Yi Zhu, Zhen-Zhong Lan, Shawn D. Newsam, and Alexander G. Hauptmann. Hidden two-stream convolutional networks for action recognition. *CoRR*, abs/1704.00389, 2017.