Recognizing Semantics in Human Actions with Object Detection

OSCAR FRIBERG
Recognizing Semantics in Human Actions with Object Detection

OSCAR FRIBERG
Abstract

Two-stream convolutional neural networks are currently one of the most successful approaches for human action recognition. The two-stream convolutional networks separates spatial and temporal information into a spatial stream and a temporal stream. The spatial stream accepts a single RGB frame, while the temporal stream accepts a sequence of optical flow. There have been attempts to further extend the work of the two-stream convolutional network framework. For instance there have been attempts to extend with a third network for auxiliary information, which this thesis mainly focuses on.

We seek to extend the two-stream convolutional neural network by introducing a semantic stream by using object detection systems. Two contributions are made in thesis: First we show that this semantic stream can provide slight improvements over two-stream convolutional neural networks for human action recognition on standard benchmarks.

Secondly, we attempt to seek divergence enhancements techniques to force our new semantic stream to complement the spatial and the temporal streams by modifying the loss function during training. Slight gains are seen using these divergence enhancement techniques.
Sammanfattning

Faltningsnätverk i två strömmar är just nu den mest lyckade tillvägagångsmetoden för mänsklig aktivitetsigenkänning, vilket delar upp rumslig och timlig information i en rumslig ström och en timlig ström. Den rumsliga strömmen tar emot individella RGB bildrutor för igenkänning, medan den timliga strömmen tar emot en sekvens av optisk flöde. Försök i att utöka ramverket för faltningsnätverk i två strömmar har gjorts i tidigare arbete. Till exempel har försök gjorts i att komplementera dessa två nätverk med ett tredje nätverk som tar emot extra information.

I detta examensarbete söker vi metoder för att utöka faltningsnätverk i två strömmar genom att introducera en semantisk ström med objektdetektion. Vi gör i huvudsak två bidrag i detta examensarbete: Först visar vi att den semantiska strömmen tillsammans med den rumsliga strömmen och den timliga strömmen kan bidra till små förbättringar för mänsklig aktivitetsigenkänning i video på riktmärkesstandarder.

För det andra söker vi efter divergensutökningstekniker som tvingar den semantiska strömmen att komplementera de andra två strömmarna genom att modifera förlustfunktionen under träning. Vi ser små förbättringar med att använda dessa tekniker för att öka divergens.
# Contents

1 Introduction ............................................. 1  
  1.1 Problem Statement .................................. 2  
  1.2 Ethical, Societal and Sustainability Aspects .... 2  
  1.3 Outline of Thesis .................................. 3  

2 Background ............................................. 5  
  2.1 Rise of Convolutional Neural Networks .......... 5  
  2.2 Image Recognition .................................. 6  
  2.3 Human Action Recognition in Videos ............ 7  
    2.3.1 Datasets for Human Action Recognition .... 8  
  2.4 Object Detection Systems ......................... 9  
  2.5 Related Work .................................... 10  

3 Deep Learning Introduction ............................... 12  
  3.1 Machine Learning .................................. 12  
    3.1.1 Supervised Learning ......................... 13  
    3.1.2 Unsupervised Learning ...................... 13  
    3.1.3 Regression .................................. 13  
    3.1.4 Classification ............................... 15  
  3.2 Deep Convolutional Networks ....................... 17  
    3.2.1 Non-linear activations ...................... 18  
    3.2.2 Convolutional Layers ....................... 19  
    3.2.3 Batch Normalization ......................... 22  
    3.2.4 Dropout ................................... 22  
  3.3 Parameter Optimization .............................. 23  
    3.3.1 Categorical Crossentropy .................... 23  
    3.3.2 Gradient Descent ............................ 23  
    3.3.3 Stochastic Gradient Descent ............... 24  
    3.3.4 Weight Decay ................................ 25
4 Human Action Recognition

4.1 Two-Stream Convolutional Networks
4.1.1 VGG16 - A Very Deep Convolutional Network
4.1.2 Optical Flow
4.1.3 Spatial Stream
4.1.4 Temporal Stream

4.2 Object Detection
4.2.1 YOLO — You Only Look Once
4.2.2 SSD — Single Shot MultiBox Detector

4.3 Semantic Stream
4.3.1 Where to connect YOLO
4.3.2 Where to connect SSD
4.3.3 Architecture of Semantic Stream
4.3.4 Fusion of Streams

4.4 Divergence Enhancement
4.4.1 Average Layer
4.4.2 Kullback-Leibler Divergence
4.4.3 Dot Product Divergence
4.4.4 Combining Average with Divergence

4.5 Implementation Details
4.5.1 Training
4.5.2 Datasets
4.5.3 Evaluation
4.5.4 Equipment

5 Results
5.1 Setting up the Spatial and the Temporal Streams
5.2 Objects Detected in Clips
5.3 Experiments
5.3.1 Where to Connect the Semantic Stream
5.3.2 Dropout Rates
5.3.3 Adding Additional Hidden Layers
5.3.4 Divergence Enhancement
5.3.5 Experiments on HMDB-51
5.3.6 Mean Accuracy over all Three Splits
5.3.7 Per Class Accuracy
6 Discussion

6.1 Comparison of Object Detection Systems ............ 59
6.2 Comparison with State-of-the-Art .................... 59
6.3 Directions of Improvement .......................... 60
  6.3.1 Temporal Setting ............................... 60
  6.3.2 Very Deep Semantic Streams .................. 61
  6.3.3 Further Investigations on Divergence Enhancement ............. 61
  6.3.4 More Object Categories ........................ 62

7 Conclusion ........................................ 63

Bibliography ....................................... 64

A Dataset Classes .................................. 71
  A.1 HMDB-51 ........................................ 71
  A.2 UCF-101 ......................................... 72
  A.3 COCO ........................................... 73
Chapter 1

Introduction

What kind of features are essential for recognizing different human actions? This is a natural question to ask when designing human action recognition systems. Human action recognition is the problem of deciding what kind of human action a video clip represents. One can see human action recognition as an extension to image recognition, which concerns about what a single image represents instead of a video.

One could argue that a temporal awareness is essential for human action recognition. For example, a still image of a person about to sit down might look indistinguishable from a still image of a person standing up from a sitting position. If we on the other hand are able to track the motion of the person over multiple consecutive frames, we will probably be able to deduce if the person is about to either sit down or stand up.

Of course, while temporal awareness is important for action recognition, a spatial awareness can be important too. Some actions are often made in certain environments. For example, an action is probably related to cooking or eating if we know the action is performed in a kitchen.

The two-stream convolutional neural network is a current state-of-the-art model for human action recognition which combines the spatial information with the temporal information of the video by using two separate convolutional neural networks [14]. One convolutional neural network for spatial information, and another for temporal information. These two networks are respectively called the spatial stream and the temporal stream.

The spatial stream accepts one individual RGB frame, while the
temporal stream accepts the optical flow of a sequence of frames. A human action is then predicted by a fusion of the two streams. The combination of using RGB and optical flow in this way has proven to be very effective on human action recognition, and thus inspired further advancements in human action recognition.

But there are certainly other representations useful for human action recognition other than RGB frames and optical flow. There have been attempts to extend the two streams with a third stream. For example, the pose of the human actors of an action has been used for human action recognition together with the two streams with some success [63].

In this thesis we investigate if the awareness of the location of different objects is important for human action recognition. Many actions are related to human-object interactions.

For example, if a hammer is detected close to the hands of a person, the action is probably related to hammering. Another example is an action is likely to be related to riding a horse if a person is detected above a horse. These two examples motivates why object detection could potentially be important for human action recognition.

Object detection systems are nowadays very efficient. Some of the fastest systems are not much slower than image recognition. This allows us to use object detection systems without a major concern about performance issues.

1.1 Problem Statement

The question this thesis investigates is if object detection systems can complement the two-stream model in [14] for human action recognition. To answer this question we implement a semantic stream which maps object detection features from state-of-the-art object detection systems to human action recognition classes.

1.2 Ethical, Societal and Sustainability Aspects

Human action recognition have some potential real world applications. One potential real world application is visual surveillance [41].
For example, a good human action recognition system could possibly be used to recognize if there is a robber inside a store. The police could automatically get alerted about the ongoing robbery without any action from the shop keeper.

However, surveillance comes with some ethical issues, especially if the surveillance is made in public spaces by a government. Surveillance in public spaces could of course help solve crimes, but it could also help a government to suppress oppositional forces. Human action recognition could be used to detect when oppositional citizens protest against the regime. This technology has some serious ethical issues if used by wrong actors, especially dictatorships with limited free speech.

From a sustainability standpoint the progress of human action recognition could be used for animal surveillance of endangered species or domestic animals in farms, assuming that animal actions have similar properties to human actions. Techniques of human action recognition could possibly be used to recognize when an animal in danger is in the need of help. Endangered species could therefore be protected and domestic farm animals can get the required health assistance in time. We do not know if there have been studies regarding these kinds of animal surveillance, so we do not know if this application is possible.

1.3 Outline of Thesis

This thesis will start with a introduction with the current progress of convolutional neural networks, human action recognition and object detection in chapter 2. The chapter will mostly focus on architectures with convolutional neural networks, but the reader should have in mind that human action recognition has a rich history predating the current progress with convolutional neural networks.

Chapter 3 introduces some of the deep learning techniques used by the explored architectures in this thesis. Readers already common with machine learning and artificial neural networks can skip most of the sections in this chapter.

Chapter 4 outlines the human action recognition techniques used in this thesis. This chapter should be read in detail in order to get a full understanding of the experiments presented in chapter 5.

Finally, chapter 6 discusses the results presented in chapter 5. Some
suggestions of potential future improvements are also discussed. The thesis is concluded with chapter 7.
Chapter 2

Background

2.1 Rise of Convolutional Neural Networks

Computer vision has changed in a rapid pace the past few years with the rise of convolutional neural networks. The popularity of convolutional neural networks started when it was proven to be effective on large image recognition benchmarks such as ImageNet [10]. Ever since then, convolutional neural networks have been used to win multiple competitions [19].

Even if convolutional neural networks made its breakthrough with image recognition, researchers have found it to be useful for other computer vision applications as well. For example, convolutional neural networks have been used to generate captions to images and videos [56] [13] [23], generate sound for silent videos [37] and even generate colorized images from grayscale images [6]. Convolutional neural networks have even found use in applications beyond computer vision, such as evaluating board positions and moves for a champion level Go AI [49]. The possibilities are seemingly endless.

Convolutional neural networks are not a new invention. It has been used even early as the 1990s for image recognition problems [32]. The reason why convolutional neural networks became popular just this decade is due to the introduction of large-scale public repositories such as ImageNet, which made it possible to train deep convolutional neural networks [10].

Also, another part of the increased popularity is the increased computing power capacity and accessibility to advanced computing equipment such as GPUs. Training convolutional neural networks on GPUs
can be faster than CPU training by a factor 10 [48]. The advancement of computing equipment has allowed to stack multiple convolutional layers in a deep network, allowing for learning complex representations.

### 2.2 Image Recognition

Image recognition, or image classification, is the problem to label an image by its contents. A typical example of a simple image recognition problem is digit recognition in the MNIST database of handwritten digits [31]; decide which digit is represented in an image. More complex image recognition problems, such as ILSVRC-2012 classification task from ImageNet, an image recognition system has to correctly label 50,000 images into 1,000 categories [46].

The raw pixel values of images are often very high dimensional. For example, the image space of a square RGB image with size $256 \times 256$ has $256 \times 256 \times 3 = 196,608$ dimensions. However, this many degrees of freedom for the image features is certainly unnecessary for an image recognition problem. A random sample of this image space is very likely to be just random noise with no valuable information.

It is therefore a common practice to project the features of the images to a lower dimensional space to make image recognition more viable. This lower dimensional space should ideally contain as much valuable information as possible from the original image. For instance, traditional face recognition methods like Eigenfaces projects the input to a linear subspace with Principal Component Analysis (PCA) [2].

Another practice to make classification more viable is to change the descriptor of the image. The typical RGB descriptor is good for representing the colors of the image, but does not necessarily represent the shapes or the structure of the image. Histogram of Oriented Gradients (HOG) is a hand-crafted descriptor which is better at representing the structural properties of an image. The HOG descriptor maps the image into a HOG-space where local gradients of the image is represented, and this descriptor has been used for traditional image recognition systems [12].

But convolutional neural networks have made handcrafted representations and descriptors such as HOG obsolete for image recognition. Instead of using hand-crafted representations, convolutional
neural networks are used to learn the representation directly from the RGB features of the image. Convolutional neural networks are seemingly flexible enough to learn representations by itself. It has even been shown that handcrafted features such as HOG can be interpreted as corresponding to a part of convolutional neural networks [28].

### 2.3 Human Action Recognition in Videos

Human action recognition in videos is the problem of recognizing what kind of human action is performed in a video. Examples of human actions to recognize can be simple actions such as handwaving, walking or jumping, or more advanced actions such as playing basketball, salsa dancing or tai chi.

Even if human action recognition is seemingly similar to image recognition, convolutional neural networks have so far not benefited human action recognition over hand-crafted video representation substantially.

Dense trajectories is a hand-crafted video representation which has been proven successful for action recognition [58]. Dense trajectories was introduced for instance in [57], and uses optical flow fields to track densely sampled points. Another hand-crafted approach uses dense trajectories together with bag of visual words with great success [39].

3D convolutional neural networks have been used to combine the spatial and the temporal features of the input in the same model for human action recognition. However, the initial success of 3D convolutional neural networks in human action recognition was small [22]. A possible reason for the small success is that the available datasets for human action recognition are currently too small to properly learn the 3D convolutional neural networks.

Combining the predictions of 2D convolutional neural networks from a sequence of frames has also been tried. [24] propose different strategies to combine the predictions of convolutional neural networks from multiple frames, and conclude that a slow-fusion approach is the most effective.

One of the most successful approaches of using convolutional neural networks for human action recognition divides the spatial and the temporal features into two separate networks, commonly known as two-stream networks [50]. Two-stream networks separately learns the
features of the RGB frames and the optical flow of the videos. This is apparently inspired by observations in how the brain recognizes actions. Evidently from its success, the RGB frames and the optical flow of a video provides complementary predictions required for human action recognition.

Recurrent neural networks, commonly used for sequential data, have also been considered for action recognition. Long short-term memory (LSTM) is a specific type of recurrent neural network proven to be capable of large-scale learning of speech recognition and other natural language processing problems [20] [55].

The original two-stream approach [50] predicts on a single-frame basis. In its standard form it is unable to model temporal structures, and there are some attempts to cope with this issue. LSTM has been used to extend the two-stream approach to accept sequential data, with an observed improvement over the single-frame baseline [13].

Another way to cope with modeling temporal structures with two-stream networks by deriving the consensus between sampled snippets of the clip [59]. For each of the snippets an action is predicted using the two streams. The predictions are then combined by deriving the consensus among the prediction. This is to our knowledge the best approach achieved on UCF-101 with 94.2%.

2.3.1 Datasets for Human Action Recognition

As human action recognition in videos has progressed, the demand for more complex datasets has increased. Early datasets, like the KTH [29] and the Weizmann dataset [3], were small and constrained. The KTH dataset contains six different types of actions performed by 25 subjects in four different environments. The camera is static and filmed so the full bodies of the subjects are visible. The background of the videos are also free from any kind of eventual distractions.

The Weizmann dataset is very similar to the KTH dataset. The actions are mostly the same, but is less constrained by letting the subjects to be in more complex environments. The camera is still static and the full bodies of the subjects are mostly visible.

UCF-Sports is a dataset with sport actions collected from various TV broadcasts [45]. This dataset is more challenging than the previous datasets by being in a much less constrained setting with various camera angles, lighting, backgrounds. Similarly, the Hollywood dataset
Figure 2.4.1: Example of a traditional sliding windows approach for object detection. First a classifier is used on windows of different sizes at different locations of the image. Regions yielding probable detections are proposed. Possible duplicates are ruled out so only the best detection is used. Nowadays a single convolutional network can be used to get the detections directly.

collects different human actions from Hollywood movies [30]. This dataset also includes shots within clips.

The two most common datasets for human action recognition as of today are UCF-101 [52] and HMDB-51 [26]. Both datasets are collections of human actions from many different kinds of sources, like television programs, internet videos and feature films. This makes the datasets difficult as the videos are unconstrained.

2.4 Object Detection Systems

While human action recognition has not enjoyed the same leap in progress as image recognition, object detection systems have surprisingly benefited very well by convolutional neural networks.

Traditional object detection systems, like ensemble of exemplar-svms or deformable parts models (DPM), used a sliding window approach for object detection [35] [15]. These approaches find the objects in an image by running a classifier at evenly spaced locations over the entire image (depicted in figure 2.4.1). Only one classifier need to be learned for the sliding window approach. However, using sliding windows is very costly since a classifier has to be used many times for each image, which makes real-time detections very difficult with this approach.

More recent approaches, like Faster R-CNN, avoids the sliding window approach by using a convolutional neural network for proposing the regions of the objects [44]. After proposing the regions and ruling
out possible duplicates, a classifier is run over each region proposal to get the class scores. The downside of approaches like Faster R-CNN is that two networks need to be learned. One network for proposing regions and another for classifying the proposed regions. Faster R-CNN is also slower than real-time even on powerful GPUs like GeForce GTX Titan X [43].

The object detection system You Only Look Once (YOLO) on the other hand, proved it is possible to unify the region proposal and class score prediction in one single convolutional neural network. This unified network is much simpler to optimize than Faster R-CNN since the object detection can be framed as a single regression problem. Faster R-CNN requires that multiple components in a complex pipeline are trained separately [42]. The single convolutional neural network architecture also allows YOLO to achieve a real-time performance.

YOLO has been improved upon its first original introduction in a second version, YOLOv2 [43]. YOLOv2 is similar to the original version, but some adjustments in the model make it both more accurate and faster. Single Shot Multibox Detector (SSD) is another network which follows the success of YOLO by performing object detections in one single convolutional neural network [34].

### 2.5 Related Work

There have been other attempts to extend the two-stream fusion with other complementary inputs or using object detections for human action recognition.

M. Zolfaghari et al. use 3D convolutional neural networks instead of 2D convolutional neural networks for action recognition [63]. As in [50], they train a spatial and a temporal stream for action recognition. But they also attempt to complement the two streams with a third stream based on the pose of the persons in the clip. This approach made most success on HMDB-51 dataset with 69.7%, which to our knowledge is the best result achieved on HMDB-51.

The work of Y. Wang et al. is currently the most similar to our work. In their paper they investigated how object detection regions could be used for action recognition [60]. They also looked into how to incorporate object detection with optical flow.

The above approach is mainly inspired by another action recogni-
tion approach based on object detections by G. Gkioxari, R. Girshick and J. Malik, which also made use of object detections for action recognition [17]. However this is done on still images on PASCAL VOC Action dataset. At the time this work was a huge success on still image action recognition.

Our contribution differentiates from the above approaches by only looking at the location and the target class of the bounding boxes. We do not make a decision based on the contents of the regions.
Chapter 3

Deep Learning Introduction

This chapter serves as a quick introduction to the deep learning techniques used in this paper. No prior knowledge of machine learning techniques is required for reading this chapter.

Section 3.1 discusses the machine learning techniques which are important for this paper. Section 3.2 continues by describing the basis for deep convolutional neural networks. Finally section 3.3 discuss how the parameters of the deep convolutional network can be optimized in a classification setting.

3.1 Machine Learning

Machine learning is a common name of techniques that learn from data. Many problems are difficult to solve directly programmatically by hand. For example, imagine to handcraft a system that recognizes a cat in an image. A cat can look in many different ways, with many different fur colors and shapes. Also, we need to take into consideration of different light conditions and different orientations of the cat. The cat can also be in different poses, which makes the problem even more difficult. There are too many cases we need to consider, which makes fully handcrafted systems infeasible to implement.

The machine learning way of solving this particular problem is to let the program learn how to recognize cats from data. For instance, we prepare some examples of images with cats and without cats, and label the desired output of the images accordingly. The program will then attempt to find the best possible mapping between the input and the output. The goal is that the program will be able to recognize if
there are cats in an image not included in the examples.

In a more general sense, machine learning covers problems where we want to learn the "best" mapping $f$ between the input $X$ and output $Y$ by observing a subset $X_{\text{train}} \subset X$. The meaning of "best" depends on the problem we want to solve and if the desired output $Y$ is known or not.

### 3.1.1 Supervised Learning

Supervised learning is the machine learning setting when we know both the input $X$ and output $Y$ during training. In this case we normally want to minimize the error between the predicted target $\tilde{y}^{(i)}$ and desired target $y^{(i)} \in Y$ where $\tilde{y}^{(i)} = f(x^{(i)})$ and $x^{(i)} \in X$.

The problem with supervised learning is that we need labeled data. Often the data must be labeled by hand, which makes it expensive to gather large amounts of data. The advantage of labeling the data by hand is that we can decide the desired outcome of system.

### 3.1.2 Unsupervised Learning

Unsupervised learning is the machine learning setting when we know the input $X$, but do not know the output $Y$. In these cases the program has to learn both the mapping $f$ and the "best" representation of $Y$. Again, the meaning of "best" depends on the problem. Commonly we want $Y$ to maintain as much valuable information of $X$ as possible.

It is normal to use unsupervised learning to enhance supervised learning. The input $X$ might in its raw form be too complicated for the supervised learner. An unsupervised learner can in these cases be used to find an easier representation of $X$. More formally explained, we want to learn two mappings $f$ and $g$ such that the error between $\tilde{y}^{(i)} = f(g(x^{(i)}))$ and the desired target $y^{(i)}$ is minimized, where $f$ is a supervised learner, and $g$ is an unsupervised learner.

The combination of unsupervised learning and supervised learning is crucial for deep convolutional networks, which we will return to in section 3.2.

### 3.1.3 Regression

Regression is the supervised learning problem of finding the best real-valued mapping $f : \mathbb{R}^n \to \mathbb{R}^m$, where $n$ and $m$ are the dimensions of
the input and output, respectively. While human action recognition is a classification problem (detailed in section 3.1.4), regression has an important role in classification, so it is necessary to cover the details.

In this subsection we will only detail regression with linear functions. Later sections will explain how linear regression can be extended for non-linear regression.

**Linear Regression**

Linear regression is the problem of finding a linear function that best maps a given real-valued pair of inputs and outputs. In the single-variate case, the linear regression is formalized as

$$wx + b = y$$ (3.1.1)

where $w$ and $b$ are the parameters we want to learn. We call $w$ the weight and $b$ the bias. However, normally we need to do regression with multiple input and output variables. Linear regression with multi-
variate input is formalized as the sum of multiple single-variate linear regressions.

\[ D \sum_{i=0}^{D} w_i x_i + b_i = w^T x + \sum_{i=0}^{D} b_i = w^T x + b = y \]  \hspace{1cm} (3.1.2)

Where \( b = \sum_{i=0}^{D} b_i \). Linear regression is easily extended to multi-variate output by modeling a single-output linear regression for each output value, which is formalized in matrix form as.

\[ W^T x + b = y \]  \hspace{1cm} (3.1.3)

### 3.1.4 Classification

Classification is the problem to associate the given input to the most appropriate class. The classes are represented in a binary vector \( c \) where \( c_i = 1 \) represents that the input belongs of the \( i \)-th class. We only consider the case where the input can belong to only one class, so the other components of \( c \) have to be 0.

In computer vision this type of problem is commonly called the image recognition problem. Image recognition concerns if a certain target is present in an image or not. Note that we are not interested in the location of the target in the image. The location of the targets are considered in the detection problem, which we will return to in section 4.2.

When modeling the classification problem, it is easier to consider the output as a probability distribution \( \bar{c} \). The component \( \bar{c}_i \) represents the probability that the input belongs to the \( i \)-th class. Then \( c_i = 1 \) for \( i = \arg\max_i \bar{c}_i \). Representing the output as a probability distribution allows us to approach the classification problem as a regression problem. Also, a probability distribution allows us to model the certainty of the prediction. A prediction is seen as more certain if the probability for one class is close to 1.

### Logistic Regression

But how do we model the classification as a probability distribution? We start with the case with only one target class \( c \), where \( c \) is the probability that the input belongs to the given class. A value closer to 1 represent high certainty that the input belongs to the given class, and 0 represent a low certainty.
Logistic regression is quite similar to linear regression. The difference is that the output is bounded between 0 and 1 (otherwise it would not be a probability). Let $z$ be the linearly dependent on the input $x$ by

$$z = \mathbf{w}^T \mathbf{x} + b \tag{3.1.4}$$

To bound the output between 0 and 1, we use the sigmoid function $\sigma$ defined as

$$\sigma(z) = \frac{e^z}{e^z + 1} \tag{3.1.5}$$

As can be seen, $\sigma(z)$ has the properties

$$\sigma(z) \to 1 \text{ when } z \to \infty, \tag{3.1.6}$$

$$\sigma(z) \to 0 \text{ when } z \to -\infty \text{ and } \tag{3.1.7}$$

$$\sigma(0) = 0.5 \tag{3.1.8}$$

which makes the sigmoid function a viable choice for modeling a probability.

**Softmax**

Logistic regression is only applicable for classification problems with only one target class. For the case with multiple target classes, we use the softmax-function to model a probability distribution. Softmax works similarly to logistic regression, and is defined by

$$\text{softmax}(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=0}^{K} e^{z_k}} \tag{3.1.9}$$
Where $z$ is linearly dependent of $x$ by

$$z = W^T x + b$$  \hspace{1cm} (3.1.10)

Important for deep convolutional networks is that softmax and logistic regression both have closed form derivatives. Closed form derivatives are necessary for computing gradients for back-propagation when training deep convolutional networks (see section 3.3).

**Beyond Linear Classification**

Softmax is an example of a linear classification function, which means that the decision boundaries[^1] of each class is a linear function. Some classification problems are linearly classifiable, but far from all are not.

However, if we can map $x_j$ to a linearly separable feature space, it is possible to use a linear classifier for the classification problem. Deep convolutional networks attempts to learn a linearly separable feature space of $x_j$ and the linear classification simultaneously, which is seen in the following section.

### 3.2 Deep Convolutional Networks

Deep learning, also known as deep structured learning or hierarchical learning, is a class of machine learning algorithms for learning representation in multiple levels [11]. Mathematically, deep learning can be seen as the composition of learnable functions $f_0, f_1, \ldots, f_n$ such that

$$f_0 \circ f_1 \circ \ldots \circ f_n (x) = \tilde{y}.$$  \hspace{1cm} (3.2.1)

As can be seen in equation 3.2.1 a deep convolutional network is feedforward; the input of the function $f_i$ is only dependent on the output of the functions $f_j$ for $j > i$.

The choice of the functions $f_i$ depends on the application. Often the functions are simple. For classification problems $f_0$ is typically chosen to be the softmax function defined in section 3.1.4. An important condition the functions must follow is that they have to be differentiable. Otherwise it is not possible to compute the gradients for back-propagation (see section 3.3). From now on, we refer to these

[^1]: The decision boundary of the softmax function for the $j$-th class is all $x$ such that $\text{softmax}(x)_j = 0.5$
3.2.1 Non-linear activations

The most basic layer used in deep convolutional networks are the fully connected layers. A layer $f_{fc}$ is fully connected if each component $y_i \in \tilde{y} = f_{fc}(x)$ is dependent on all components of $x$.

The most basic form of a fully connected layer is when $\tilde{y}$ and $x$ are linearly dependent — $f$ is a linear function as defined in equation 3.1.3. However, the composition of linear functions cannot model non-linear behaviors. If $f_1$ and $f_2$ are both linear, then the composition $f_1 \circ f_2$ is also linear.

A common goal of adding hidden layers to a deep convolutional network is to model non-linear behaviors. To make a fully connected layer non-linear, we activate each component of $\tilde{y}$ with a non-linear activation function. Below we describe the activation functions used in this thesis. Note that activation functions are not limited to fully connected layers. Other layers, such as convolutional layers (see section 3.2.2), make use of activation functions as well.

Rectified Linear Unit

One of the most commonly used activation layers is the Rectified Linear Unit (ReLU). The definition of ReLU is simple; it is the identity function as layers, and $f_1, \ldots, f_n$ in 3.2.1 are referred as hidden layers. This section introduces the layers that are used in this thesis.
for positive input, and 0 for negative input. Mathematically, ReLU is expressed as

\[
\text{ReLU}(x) = \begin{cases} 
  x & \text{if } x \geq 0 \\
  0 & \text{otherwise}
\end{cases}
\]  

Observations in neuroscience suggests that activations of neurons in the brain can be approximated by a rectifier [18]. This has inspired the use of ReLU in deep convolutional networks.

**Leaky Rectifier Linear Unit**

A variation of Rectified Linear Unit is the Leaky Rectified Linear Unit. Here the negative input is scaled with a fraction. In this paper we scale the ReLU with 0.1 if negative.

\[
\text{LeakyReLU}(x) = \begin{cases} 
  x & \text{if } x \geq 0 \\
  0.1x & \text{otherwise}
\end{cases}
\]  

This definition allows some information to pass through the activation function even if the input is negative. In this paper LeakyReLU is only used for the YOLO object detection system, detailed in section 4.2.1.

### 3.2.2 Convolutional Layers

The problem with fully connected layers is that many parameters are required when the input is large. For example, if the input and output of a fully connected layer both have dimension 4096, then 4096 ×
4096 = 16,777,216 weight parameters will be required for the fully connected layer. Images are often very high dimensional, so fully connected layers are often infeasible.

Another limitation of fully connected layers is that spatial information in images is not considered. A fully connected layer do not know if a component of the input represents the features from pixels at the upper right corner or the lower left corner.

Convolutional Layers solves both of these limitations by using convolutional filters. Let us start with the 1-dimensional case of convolutional filters.

1 Dimensional Convolution

For the 1 dimensional case, the input $x$ is a vector of length $W$, and $f_{\text{conv}}$ is a 1 dimensional convolutional layer operating on $x$. Then $\tilde{y} = f_{\text{conv}}$ is defined for each $\tilde{y}_i$ as

$$\tilde{y}_i = w_0 x_{i-[K/2]} + \ldots + w_{[K/2]} x_i + \ldots + w_{K-1} x_{i+[K/2]-1} + b \quad (3.2.4)$$

Here, $K$ is the length of the convolutional filter of $f_{\text{conv}}$. As can be seen in 3.2.4, the component $\tilde{y}_i$ only depend on $K$ components in a proximity of $x_i$. This means some spatial information of $x$ is maintained in $\tilde{y}$. Also, the components of $\tilde{y}$ shares the same weights $w$. Only $K$ trainable parameters is therefore required by the layer.

Also, note that a bias term $b$ is added to each output component. The 1 dimensional convolution is illustrated in figure 3.2.2a.

2 Dimensional Convolution

For images we need to consider both the vertical and horizontal proximity of the input components. This requires the use of 2 dimensional convolutional filters.

Let the input $x$ have size $W \times H$, and the convolutional filter have size $K \times L$. For simplicity, we only consider the case when $K = L = 3$, but it is trivial to extend the definition for other values of $K$ and $L$. The component $y_{i,j}$ is defined as

$$\tilde{y}_{i,j} = w_{0,0} x_{i-1,j-1} + w_{1,0} x_{i,j-1} + w_{2,0} x_{i+1,j-1} +$$
$$w_{0,1} x_{i-1,j} + w_{1,1} x_{i,j} + w_{2,1} x_{i+1,j} +$$
$$w_{0,2} x_{i-1,j+1} + w_{1,2} x_{i,j+1} + w_{2,2} x_{i+1,j+1} + b \quad (3.2.5)$$
Similarly to the 1 dimensional case, the parameters \( w \) are shared between all components of \( \tilde{y} \), so only \( K \times L \) trainable weights with one trainable bias are required by the layer.

### 2 Dimensional Convolution with Channels

So far we have not considered that images may use multiple channels. For example, RGB images have 3 channels — one for red, green and blue each. Let us consider the general case with images of \( C \) channels so the input \( x \) has size \( W \times H \times C \). Since the order of the channels bear little meaning — RGB representation is essentially the same as BGR representation — all channels are considered in the convolution. Let the \( x_{i,j,\ast} \) be the vector of all channels in the location \((i, j)\). Then the component \( y_{i,j} \) is defined as

\[
\tilde{y}_{i,j} = w^T_{0,0,\ast} x_{i-1,j-1,\ast} + w^T_{1,0,\ast} x_{i,j-1,\ast} + w^T_{2,0,\ast} x_{i+1,j-1,\ast} +
\]
\[
w^T_{0,1,\ast} x_{i-1,j,\ast} + w^T_{1,1,\ast} x_{i,j,\ast} + w^T_{2,1,\ast} x_{i+1,j,\ast} + (3.2.6)
\]

\[
w^T_{0,2,\ast} x_{i-1,j+1,\ast} + w^T_{1,2,\ast} x_{i,j+1,\ast} + w^T_{2,2,\ast} x_{i+1,j+1,\ast} + b
\]

Similar to the previous cases, the parameters \( w \) are shared. So the number of trainable weights is \( K \times L \times C \) with one trainable bias.

The 2 dimensional convolution with channels is illustrated in figure 3.2.2c.

### Multiple Convolutional Filters

Finally, we reach the final definition of the convolutional layer. The definition in equation 3.2.6 outputs a representation with 1 channel. If we want to output \( D \) channels, we define \( D \) different convolutional filters and stack the resulting representation on the extra channel space.

So a convolutional layer with \( D \) filters with width \( K \), height \( L \) applied on a representation with \( C \) channels have \( K \times L \times C \times D \) trainable weights with \( D \) trainable biases.

To achieve non-linearly, the components of the output representation \( y \) are activated with a non-linear activation function, such as ReLU.

### Padding

The above definition of convolutional filters is only well defined for \( \tilde{y}_{i,j} \) where \( \lfloor K/2 \rfloor < i < W - \lceil K/2 \rceil \) and \( \lfloor L/2 \rfloor < j < W - \lceil L/2 \rceil \). For
other values of $i$ and $j$ the convolution will cover components outside the representation. To deal with this, we pad the representation with 0 around its borders so the convolution is well defined for all valid components of the representation.

**Max Pooling**

Max pooling is a way to reduce the spatial size of a representation [8]. It works by dividing the representation in cells with size $N \times M$. For each cell, the component with the highest value is returned. Normally $N = M = 2$.

Reducing the spatial size of a representation reduces the number of parameters in a network, which may reduce the risk of overfitting [8].

**Strides**

An alternative to max pooling is striding. Striding also reduces the spatial size of a representation. If the strides are $N \times M$ the convolution will skip to the next $N$-th input in the $x$-axis and $M$-th input in the $y$-axis.

### 3.2.3 Batch Normalization

A batch normalization layer learns the mean and variance of its input [21]. The layer then normalizes the input so the output have 0 mean and 1 variance according to the learned mean and variance.

An alleged benefit of batch normalization is faster training [21]. For this thesis batch normalization is only used for YOLO object detection system described in section 4.2.1.

### 3.2.4 Dropout

A danger with training models is that the trained model might rely too much on patterns specific to the training data and misses general patterns also present in unobserved data. If a model performs well on observed data, but performs bad on unobserved data, we say that the model is overfitting.

A simple way to prevent overfitting is to use the dropout layer [53]. The dropout layer has a dropout rate hyperparameter $p$ which determines the probability that a component of the input is chosen to be
dropped out during training. If a component is dropped out, it is set to 0. During validation, the dropout layer has no effect.

To avoid inconsistencies between training and validation, the output of the dropout layer is scaled by \(1/(1-p)\).

### 3.3 Parameter Optimization

So far we have only discussed the building blocks of deep convolutional networks, but we have not discussed on how the parameters of the network are learned. This section discusses how to optimize the parameters given the training data.

#### 3.3.1 Categorical Crossentropy

When training a model we want to reduce the error between the predicted targets \(\hat{y}\) and the true targets \(y\) as much as possible. Let \(L(\hat{y}, y)\) be a loss function, representing the error between \(\hat{y}\) and \(y\). The parameters \(\theta_{optimal}\) of a model \(f\) are optimal when

\[
\theta_{optimal} = \arg \min_{\theta} \sum_{i=0}^{N} L(f(x^{(i)}; \theta), y^{(i)})
\]  

(3.3.1)

For human action recognition in videos we want to minimize the classification error. A common objective used as classification error is categorical crossentropy, defined as

\[
L_{cross}(\hat{y}, y) = -\sum_{k=0}^{K} y_k \log \hat{y}_k
\]

(3.3.2)

Here, a classification is perfect when \(L_{cross}(\hat{y}, y) = 0\).

#### 3.3.2 Gradient Descent

Gradient descent is a straightforward technique for optimizing non-linear objectives\(^\text{2}\). Starting from an initial assumption \(\theta_0\), iteratively compute \(\theta_{t+1}\) by following the gradient of the objective \(g\) with the parameters \(\theta_t\).

\(^\text{2}\)For those unfamiliar with multi-variate calculus, a gradient \(\nabla f(x)\) is the vector of the partial derivatives of \(f(x)\). The gradient typically points towards the direction of the steepest slope of \(f(x)\) for a given \(x\).
The update of $\theta_{t+1}$ is formally defined as

$$
\theta_{t+1} = \theta_t - \gamma \nabla \theta_t g(x_i; \theta_t)
$$

(3.3.3)

where $g$ is the objective we want to minimize, and $\gamma$ is a learning rate factor. The learning rate factor is used to control how large steps the gradient will take. Lower learning rates are usually more accurate with the expense of converging at a slower rate. A common practice is to start with a high learning rate and lower the learning rate according to a given schedule or when the objective saturates.

In our case the objective we want to minimize is the empirical risk $E_N(L)$ — the average loss across all inputs.

$$
E_N(L) = \frac{1}{N} \sum_{i=0}^{N} L(f(x^{(i)}; \theta), y^{(i)})
$$

(3.3.4)

The gradient descent for categorical cross entropy is therefore defined as

$$
\theta_{t+1} = \theta_t - \gamma \frac{1}{|X_t|} \sum_{i \in X_t} \nabla \theta_t L(f(x^{(i)}; \theta_t), y^{(i)})
$$

(3.3.5)

A drawback of gradient descent is that it needs to see the whole dataset before the parameters are updated one step [4]. This can be very costly and possibly infeasible. Gradient descent is therefore rarely used for large datasets.

### 3.3.3 Stochastic Gradient Descent

Stochastic gradient descent (SGD) approximates gradient descent by, instead of updating the parameters after seeing all inputs in the training data, updating the parameters after seeing a random batch of the training data. Larger batches lead to a more accurate optimization at the expense of a more costly training.

Let $X_t$ be a random sample of the dataset at step $t$. SGD is then defined as

$$
\theta_{t+1} = \theta_t - \gamma \frac{1}{|X_t|} \sum_{i \in X_t} \nabla \theta_t L(f(x^{(i)}; \theta_t), y^{(i)})
$$

(3.3.6)
Momentum

An extension of SGD is to instead accumulate a velocity $v_t$ for each iteration $t$ of the parameter update, instead of updating the parameters directly [54]. At each iteration, the parameters are updated by the accumulated velocity $v_t$. The gradients will have an accelerating effect on the parameters.

SGD with momentum is defined as

$$v_{t+1} = \mu v_t - \gamma \frac{1}{|\mathcal{X}_t|} \sum_{i \in \mathcal{X}_t} \nabla_{\theta_t} \mathcal{L}(f(x^{(i)}; \theta_t), y^{(i)})$$

$$\theta_{t+1} = \theta_t + v_{t+1}$$

(3.3.7)

where $\mu \in [0, 1]$ is the momentum coefficient. The momentum coefficient determines how slow the accumulated velocity will decrease. $\mu = 0$ means no momentum is used, and corresponds to the standard SGD.

3.3.4 Weight Decay

Weight decay is a simple way of reducing the risk of over-fitting in deep convolutional networks by adding an extra term in the loss function [25]. Over-fitting can occur when deep convolutional networks learns complex structures in the training set, while there is actually little information in the training set. Weight decay constrains the network by penalizing large weights.

Returning to the standard form of gradient descent in equation 3.3.5, the weight decay is formalized as

$$\theta_{t+1} = \theta_t - \gamma \left( \frac{1}{N} \sum_{i=0}^{N} \nabla_{\theta_t} \mathcal{L}(f(x^{(i)}; \theta_t), y^{(i)}) - \alpha \theta_t \right)$$

(3.3.8)

where $\alpha$ is the weight decay factor. Naturally, the weight decay term can also be included in SGD and SGD with momentum.

3.3.5 Transfer Learning

Traditional machine learning algorithms assumed that the training and validation set for a task are drawn from the same distribution. This means that machine learning models are re-trained from scratch for each new task. However, it has been increasingly more popular to
transfer parameters learned from a previous task to a new task. This is called transfer learning [38]. This allows knowledge gained from a previous task to be transferred to a new task.

It is easy to perform transfer learning on deep convolutional networks. A common practice is to replace the top layers (often fully connected layers) with new, un-trained, layers more suitable for the task [1].

For example, if we have a model trained on a classification problem with 1000 targets, and we want to use the same model for a classification problem with 100 targets, we replace the last softmax layer to a softmax layer with 100 targets.

For image recognition it is common to use networks pre-trained on large datasets such as ImageNet. Some of these models are freely available to download.
Chapter 4

Human Action Recognition

With the basics of deep learning introduced, we can now detail the methods of human action recognition used in this thesis. This chapter will start by discussing the two-stream convolutional network model in section 4.1, which is the base-line used for this thesis.

Section 4.2 and 4.3 discuss object detection systems and how to extend object detection systems for human action recognition as a semantic stream.

Section 4.4 explores some ideas to jointly train the semantic stream together with the spatial and the temporal stream, without changing the parameters of the latter two streams.

Implementation details are detailed in section 4.5, which discuss how the streams are trained and evaluated.

4.1 Two-Stream Convolutional Networks

The two-stream convolutional network model, introduced by [50], is one of the most successful approaches to human action recognition in videos. The idea of the two-stream convolutional network is to combine the predictions of two separate networks trained on different information.

The first network is trained on the spatial information — raw RGB images — of the videos. This network is called the spatial stream, and is very similar to image recognition systems. The spatial stream sees only one frame at a time, so it cannot capture any motion. A way of interpreting the spatial stream is that it sees the context of the video. For example, if the action is in a kitchen-like environment, then the
action is likely related to cooking.

The second network is trained on the temporal information of the videos. This network is called the temporal stream, and captures the motion of the video. The input of the temporal stream is the optical flow of multiple consecutive frames of the video (more on optical flow in section 4.1.2).

These two streams provide complementary predictions. Experiments show that the average predictions from both streams yield a more accurate prediction than using the streams separately [50].

4.1.1 VGG16 - A Very Deep Convolutional Network

VGG16 is used as a base for the spatial and the temporal streams, which is a state-of-the-art model for the ILSVRC-2012 image recognition challenge [51]. It is common to perform transfer learning on VGG16 to other image recognition problems. Pre-trained weights from ILSVRC-2012 are used as initial weights when training the spatial and the temporal streams for the human action recognition problem. The architecture of VGG16 is depicted in table 4.1.1.

Note that the last softmax layer consists of 1000 targets. The last softmax layer is replaced with a softmax layer with the number of target classes for the specific human action recognition dataset.

4.1.2 Optical Flow

In this section we follow the same notation as used in [50]. The optical flow is a sequence of displacement vector fields $d_t$. The displacement vector $d_t(u,v)$ represents the motion of the point $(u,v)$ between two consecutive frames $t$ and $t+1$.

The displacement vector consists of a horizontal vector $d_t^x(u,v)$ and a vertical vector $d_t^y(u,v)$. The optical flow between two consecutive frames is therefore $W \times H \times 2$, where $W$ and $H$ is the width and height of the image. It can be seen as a two-channel image where the two channels represent the horizontal flow and the vertical flow.

The motion across a sequence of frames is represented by stacking the flow channels of $L$ consecutive frames. The input is therefore represented as a volume of $2L$ channels.

Mathematically, the input $I_\tau$ of a frame $\tau$ for the temporal stream is
<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Filters</th>
<th>Kernel</th>
<th>Output shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>input</td>
<td>3</td>
<td>224 x 224</td>
<td></td>
</tr>
<tr>
<td>Conv</td>
<td>conv1_1</td>
<td>64</td>
<td>3 x 3</td>
<td>224 x 224</td>
</tr>
<tr>
<td>Conv</td>
<td>conv1_2</td>
<td>64</td>
<td>3 x 3</td>
<td>224 x 224</td>
</tr>
<tr>
<td>Maxpool</td>
<td>pool1</td>
<td>2 x 2</td>
<td>112 x 112</td>
<td></td>
</tr>
<tr>
<td>Conv</td>
<td>conv2_1</td>
<td>128</td>
<td>3 x 3</td>
<td>112 x 112</td>
</tr>
<tr>
<td>Conv</td>
<td>conv2_2</td>
<td>128</td>
<td>3 x 3</td>
<td>112 x 112</td>
</tr>
<tr>
<td>Maxpool</td>
<td>pool2</td>
<td>2 x 2</td>
<td>56 x 56</td>
<td></td>
</tr>
<tr>
<td>Conv</td>
<td>conv3_1</td>
<td>256</td>
<td>3 x 3</td>
<td>56 x 56</td>
</tr>
<tr>
<td>Conv</td>
<td>conv3_2</td>
<td>256</td>
<td>3 x 3</td>
<td>56 x 56</td>
</tr>
<tr>
<td>Conv</td>
<td>conv3_3</td>
<td>256</td>
<td>3 x 3</td>
<td>56 x 56</td>
</tr>
<tr>
<td>Maxpool</td>
<td>pool3</td>
<td>2 x 2</td>
<td>28 x 28</td>
<td></td>
</tr>
<tr>
<td>Conv</td>
<td>conv4_1</td>
<td>512</td>
<td>3 x 3</td>
<td>28 x 28</td>
</tr>
<tr>
<td>Conv</td>
<td>conv4_2</td>
<td>512</td>
<td>3 x 3</td>
<td>28 x 28</td>
</tr>
<tr>
<td>Conv</td>
<td>conv4_3</td>
<td>512</td>
<td>3 x 3</td>
<td>28 x 28</td>
</tr>
<tr>
<td>Maxpool</td>
<td>pool4</td>
<td>2 x 2</td>
<td>14 x 14</td>
<td></td>
</tr>
<tr>
<td>Conv</td>
<td>conv5_1</td>
<td>512</td>
<td>3 x 3</td>
<td>14 x 14</td>
</tr>
<tr>
<td>Conv</td>
<td>conv5_2</td>
<td>512</td>
<td>3 x 3</td>
<td>14 x 14</td>
</tr>
<tr>
<td>Conv</td>
<td>conv5_3</td>
<td>512</td>
<td>3 x 3</td>
<td>14 x 14</td>
</tr>
<tr>
<td>Maxpool</td>
<td>pool5</td>
<td>2 x 2</td>
<td>7 x 7</td>
<td></td>
</tr>
<tr>
<td>FC-4096</td>
<td>fc1</td>
<td></td>
<td>4096</td>
<td></td>
</tr>
<tr>
<td>FC-4096</td>
<td>fc2</td>
<td></td>
<td>4096</td>
<td></td>
</tr>
<tr>
<td>FC-1000</td>
<td>fc3</td>
<td></td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>Softmax</td>
<td>predictions</td>
<td></td>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1.1: VGG16 architecture.
constructed as follows:

\[
I_{\tau}(u, v, 2k-1) = d_{x_{\tau+k-1}}^x(u, v), \\
I_{\tau}(u, v, 2k) = d_{y_{\tau+k-1}}^y(u, v), \\
u = [1; w], v = [1; h], k = [1; L] 
\]

(4.1.1)

We use the precomputed optical flow from [50], which is estimated with the TV-L1 optical flow estimation [40].

4.1.3 Spatial Stream

As mentioned earlier, the spatial stream makes a prediction based on the still RGB frames. The network is based on VGG16 in which the last layer is replaced with a fully connected layer with the correct number of target classes for the dataset. The input of the spatial stream is subtracted with the average color values of ILSVRC-2012.

4.1.4 Temporal Stream

The temporal stream is also based on VGG16, but with optical flow as input instead of RGB frames. One issue with the temporal stream is that the input uses \(2L\) channels instead of \(3\) channels for RGB. To resolve this issue, the first convolutional layer of the temporal stream is replaced with a layer with \(2L\) input channels. We use \(L = 10\) for the temporal stream, which means the input of the temporal stream is 10 consecutive optical flow images.

4.2 Object Detection

Object detection is a more challenging extension of image recognition. While image recognition only consider the presence of a certain target in an image, object detection also considers the location and size of the target. Often there are multiple targets present in the same image for the object detection system to detect.

4.2.1 YOLO — You Only Look Once

YOLO Introduction

YOLO (You Only Look Once) is a fast and accurate object detection system consisting of one single convolutional network [42]. This means
YOLO proposes bounding boxes and their corresponding class scores simultaneously.

We use the YOLOv2 and YOLO9000 models. The YOLOv2 trained on the COCO dataset, which is a dataset for object detection with 80 target classes [33]. YOLO9000 on the other hand is trained on both the COCO dataset and the ImageNet object detection challenge.

**YOLOv2 Architecture**

The architecture of YOLOv2 is depicted in table 4.2.1. YOLOv2 mostly consists of conventional layers for convolutional neural networks, but there are a few important notes to point out.

Each convolutional layer is followed by a batch normalization layer. The batch normalization layers are in turn activated by a Leaky ReLU (see section 3.2.1). The bias of each convolutional layer is also applied after the corresponding batch normalization layer.

At layer 24 the channels of ’conv5-3’ output (from layer 16) is stacked on the channels of ’conv6-5’ output (from layer 22). This operation is called concatenation. But this operation requires that all channels have the same shape. This is not the case with ’conv5-3’ and ’conv6-5’. The shape of ’conv5-3’ is $26 \times 26$, while the shape of ’conv6-5’ is $13 \times 13$.

To solve this the channels of ’conv5-3’ are reorganized at layer 23 so that the new shape of ’conv5-3’ is $13 \times 13$. This means that the number of channels of ’conv5-3’ will increase from 512 to 2048. The concatenation will therefore have 3062 channels.

The concatenation is then followed by two convolutional layers at layers 25 and 26 before the regions are predicted at layer 27. The output of the network is an array with the shape $13 \times 13 \times 425$.

**YOLO9000 Architecture**

The architecture of YOLO9000, depicted in table 4.2.2, is very similar to YOLOv2. The layers are identical between YOLOv2 and YOLO9000 up until layer 25, ’conv6-6’. For YOLO9000, ’conv6-6’ is the final convolutional layer before the region layer. YOLO9000 supports 9418 different object categories with 3 predictions for each cell, which is why ’conv6-6’ has 28269 output channels ($3 \times (9418 + 5) = 28269$).
<table>
<thead>
<tr>
<th>#</th>
<th>Type</th>
<th>Name</th>
<th>Filters</th>
<th>Kernel</th>
<th>Output shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Input</td>
<td>input</td>
<td>3</td>
<td></td>
<td>416 x 416</td>
</tr>
<tr>
<td>2</td>
<td>Conv</td>
<td>conv1-1</td>
<td>32</td>
<td>3 x 3</td>
<td>416 x 416</td>
</tr>
<tr>
<td>3</td>
<td>Maxpool</td>
<td>pool1</td>
<td></td>
<td>2 x 2</td>
<td>208 x 208</td>
</tr>
<tr>
<td>4</td>
<td>Conv</td>
<td>conv2-1</td>
<td>64</td>
<td>3 x 3</td>
<td>208 x 208</td>
</tr>
<tr>
<td>5</td>
<td>Maxpool</td>
<td>pool2</td>
<td></td>
<td>2 x 2</td>
<td>104 x 104</td>
</tr>
<tr>
<td>6</td>
<td>Conv</td>
<td>conv3-1</td>
<td>128</td>
<td>3 x 3</td>
<td>104 x 104</td>
</tr>
<tr>
<td>7</td>
<td>Conv</td>
<td>conv3-2</td>
<td>64</td>
<td>1 x 1</td>
<td>104 x 104</td>
</tr>
<tr>
<td>8</td>
<td>Conv</td>
<td>conv3-3</td>
<td>128</td>
<td>3 x 3</td>
<td>104 x 104</td>
</tr>
<tr>
<td>9</td>
<td>Maxpool</td>
<td>pool3</td>
<td></td>
<td>2 x 2</td>
<td>28 x 28</td>
</tr>
<tr>
<td>10</td>
<td>Conv</td>
<td>conv4-1</td>
<td>256</td>
<td>3 x 3</td>
<td>52 x 52</td>
</tr>
<tr>
<td>11</td>
<td>Conv</td>
<td>conv4-2</td>
<td>128</td>
<td>1 x 1</td>
<td>52 x 52</td>
</tr>
<tr>
<td>12</td>
<td>Conv</td>
<td>conv4-3</td>
<td>256</td>
<td>3 x 3</td>
<td>52 x 52</td>
</tr>
<tr>
<td>13</td>
<td>Maxpool</td>
<td>pool4</td>
<td></td>
<td>2 x 2</td>
<td>26 x 26</td>
</tr>
<tr>
<td>14</td>
<td>Conv</td>
<td>conv5-1</td>
<td>512</td>
<td>3 x 3</td>
<td>26 x 26</td>
</tr>
<tr>
<td>15</td>
<td>Conv</td>
<td>conv5-2</td>
<td>256</td>
<td>1 x 1</td>
<td>26 x 26</td>
</tr>
<tr>
<td>16</td>
<td>Conv</td>
<td>conv5-3</td>
<td>512</td>
<td>3 x 3</td>
<td>26 x 26</td>
</tr>
<tr>
<td>17</td>
<td>Conv</td>
<td>conv5-4</td>
<td>256</td>
<td>1 x 1</td>
<td>26 x 26</td>
</tr>
<tr>
<td>18</td>
<td>Conv</td>
<td>conv5-5</td>
<td>512</td>
<td>3 x 3</td>
<td>26 x 26</td>
</tr>
<tr>
<td>19</td>
<td>Maxpool</td>
<td>pool5</td>
<td></td>
<td>2 x 2</td>
<td>13 x 13</td>
</tr>
<tr>
<td>20</td>
<td>Conv</td>
<td>conv6-1</td>
<td>1024</td>
<td>3 x 3</td>
<td>13 x 13</td>
</tr>
<tr>
<td>21</td>
<td>Conv</td>
<td>conv6-2</td>
<td>512</td>
<td>1 x 1</td>
<td>13 x 13</td>
</tr>
<tr>
<td>22</td>
<td>Conv</td>
<td>conv6-3</td>
<td>1024</td>
<td>3 x 3</td>
<td>13 x 13</td>
</tr>
<tr>
<td>23</td>
<td>Conv</td>
<td>conv6-4</td>
<td>512</td>
<td>1 x 1</td>
<td>13 x 13</td>
</tr>
<tr>
<td>24</td>
<td>Conv</td>
<td>conv6-5</td>
<td>1024</td>
<td>3 x 3</td>
<td>13 x 13</td>
</tr>
<tr>
<td>25</td>
<td>Conv</td>
<td>conv6-6</td>
<td>1024</td>
<td>3 x 3</td>
<td>13 x 13</td>
</tr>
<tr>
<td>26</td>
<td>Conv</td>
<td>conv6-7</td>
<td>1024</td>
<td>3 x 3</td>
<td>13 x 13</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13 x 13</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13 x 13</td>
</tr>
<tr>
<td>29</td>
<td>Conv</td>
<td>conv7-1</td>
<td>1024</td>
<td>3 x 3</td>
<td>13 x 13</td>
</tr>
<tr>
<td>30</td>
<td>Conv (Linear)</td>
<td>conv7-2</td>
<td>425</td>
<td>1 x 1</td>
<td>13 x 13</td>
</tr>
<tr>
<td>31</td>
<td>Region</td>
<td>region</td>
<td></td>
<td></td>
<td>13 x 13</td>
</tr>
</tbody>
</table>

Table 4.2.1: The architecture of YOLOv2.
<table>
<thead>
<tr>
<th>#</th>
<th>Type</th>
<th>Name</th>
<th>Filters</th>
<th>Kernel</th>
<th>Output shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Input</td>
<td>input</td>
<td>3</td>
<td></td>
<td>416 × 416</td>
</tr>
<tr>
<td>2</td>
<td>Conv</td>
<td>conv1-1</td>
<td>32</td>
<td>3 × 3</td>
<td>416 × 416</td>
</tr>
<tr>
<td>3</td>
<td>Maxpool</td>
<td>pool1</td>
<td></td>
<td>2 × 2</td>
<td>208 × 208</td>
</tr>
<tr>
<td>4</td>
<td>Conv</td>
<td>conv2-1</td>
<td>64</td>
<td>3 × 3</td>
<td>208 × 208</td>
</tr>
<tr>
<td>5</td>
<td>Maxpool</td>
<td>pool2</td>
<td></td>
<td>2 × 2</td>
<td>104 × 104</td>
</tr>
<tr>
<td>6</td>
<td>Conv</td>
<td>conv3-1</td>
<td>128</td>
<td>3 × 3</td>
<td>104 × 104</td>
</tr>
<tr>
<td>7</td>
<td>Conv</td>
<td>conv3-2</td>
<td>64</td>
<td>1 × 1</td>
<td>104 × 104</td>
</tr>
<tr>
<td>8</td>
<td>Conv</td>
<td>conv3-3</td>
<td>128</td>
<td>3 × 3</td>
<td>104 × 104</td>
</tr>
<tr>
<td>9</td>
<td>Maxpool</td>
<td>pool3</td>
<td></td>
<td>2 × 2</td>
<td>28 × 28</td>
</tr>
<tr>
<td>10</td>
<td>Conv</td>
<td>conv4-1</td>
<td>256</td>
<td>3 × 3</td>
<td>52 × 52</td>
</tr>
<tr>
<td>11</td>
<td>Conv</td>
<td>conv4-2</td>
<td>128</td>
<td>1 × 1</td>
<td>52 × 52</td>
</tr>
<tr>
<td>12</td>
<td>Conv</td>
<td>conv4-3</td>
<td>256</td>
<td>3 × 3</td>
<td>52 × 52</td>
</tr>
<tr>
<td>13</td>
<td>Maxpool</td>
<td>pool4</td>
<td></td>
<td>2 × 2</td>
<td>26 × 26</td>
</tr>
<tr>
<td>14</td>
<td>Conv</td>
<td>conv5-1</td>
<td>512</td>
<td>3 × 3</td>
<td>26 × 26</td>
</tr>
<tr>
<td>15</td>
<td>Conv</td>
<td>conv5-2</td>
<td>256</td>
<td>1 × 1</td>
<td>26 × 26</td>
</tr>
<tr>
<td>16</td>
<td>Conv</td>
<td>conv5-3</td>
<td>512</td>
<td>3 × 3</td>
<td>26 × 26</td>
</tr>
<tr>
<td>17</td>
<td>Conv</td>
<td>conv5-4</td>
<td>256</td>
<td>1 × 1</td>
<td>26 × 26</td>
</tr>
<tr>
<td>18</td>
<td>Conv</td>
<td>conv5-5</td>
<td>512</td>
<td>3 × 3</td>
<td>26 × 26</td>
</tr>
<tr>
<td>19</td>
<td>Maxpool</td>
<td>pool5</td>
<td></td>
<td>2 × 2</td>
<td>13 × 13</td>
</tr>
<tr>
<td>20</td>
<td>Conv</td>
<td>conv6-1</td>
<td>1024</td>
<td>3 × 3</td>
<td>13 × 13</td>
</tr>
<tr>
<td>21</td>
<td>Conv</td>
<td>conv6-2</td>
<td>512</td>
<td>1 × 1</td>
<td>13 × 13</td>
</tr>
<tr>
<td>22</td>
<td>Conv</td>
<td>conv6-3</td>
<td>1024</td>
<td>3 × 3</td>
<td>13 × 13</td>
</tr>
<tr>
<td>23</td>
<td>Conv</td>
<td>conv6-4</td>
<td>512</td>
<td>1 × 1</td>
<td>13 × 13</td>
</tr>
<tr>
<td>24</td>
<td>Conv</td>
<td>conv6-5</td>
<td>1024</td>
<td>3 × 3</td>
<td>13 × 13</td>
</tr>
<tr>
<td>25</td>
<td>Conv</td>
<td>conv6-6</td>
<td>28269</td>
<td>1 × 1</td>
<td>13 × 13</td>
</tr>
<tr>
<td>26</td>
<td>Region</td>
<td>region</td>
<td></td>
<td></td>
<td>13 × 13</td>
</tr>
</tbody>
</table>

Table 4.2.2: The architecture of YOLO9000.
Figure 4.2.1: Illustration of the output of YOLOv2. The image is divided into a $13 \times 13$ grid. Each cell of the grid holds 5 predicted bounding boxes with their corresponding coordinates $x, y, w, h$, confidence of prediction $c$ and the class scores.

**YOLO Input Format**

Unlike VGG16, no mean subtraction is performed on the input images of the YOLO models. The input of YOLOv2 models are rescaled by the factor $1/255$ instead.

**YOLOv2 Output Format**

The output of YOLOv2 has fixed size; it is always an array with shape $13 \times 13 \times 425$ no matter how many objects are detected. The output represents a $13 \times 13$ grid of the input image, where each cell of the grid holds 5 predicted bounding boxes. This means YOLO will always predict $13 \times 13 \times 5 = 845$ bounding boxes.

Each bounding box holds 4 values for their coordinates in the grid, 1 value for the confidence of prediction and a class score vector. In our case we use YOLOv2 trained on COCO, so the class score vector holds 80 values, one for each class. This means each bounding box is represented by a vector of 85 values. The value for confidence is used to rule out uncertain predictions or background predictions.

Since each cell in the grid holds 5 predictions, each cell holds $85 \times 5 = 425$ values.
YOLO9000 Output Format

The output format of YOLO9000 is similar to YOLOv2. The main difference is how the class scores are represented. YOLO9000 uses a WordNet hierarchy for the class scores. WordNet is a language database which models connections between related words in a directed graph [36]. For example, in WordNet, “hunting dog” is a hyponym of “dog”.

The class scores of YOLO9000 uses the hyponyms of the class labels to model a hierarchy of classes. This allows YOLO9000 to be jointly trained on both COCO and ImageNet. For example, COCO has the target class "airplane", while ImageNet has the target classes "jet", "biplane", "airbus" and "stealth fighter", all of which are hyponyms of "airplane". Using the hyponym hierarchy, YOLO9000 is able to deduce the relation between "airplane" and "jet".

Other than a different representation of the class scores, YOLO9000 only predicts 3 different objects for each cell.

4.2.2 SSD — Single Shot MultiBox Detector

SSD (Single Shot Multibox Detector) is an object detection system that follows the success of YOLO [34]. In a similar fashion, SSD predicts and classifies the detections in one step. While YOLO and SSD are quite similar in function, there are some notable differences.

Similar to YOLO, SSD is also trained on the COCO dataset. SSD is therefore able to predict 80 different types of object. SSD has also one class designated for background. The background class is used to tell that no object is detected at a particular region. SSD therefore predicts class scores for 81 different classes for each detection box.

SSD Architecture

SSD has a bit more complicated architecture than YOLO. The output of SSD involves predictions in many different levels. While YOLO outputs detection boxes in a $13 \times 13$ grid, SSD outputs detection boxes in grids of various sizes.

The main architecture of SSD is depicted in table 4.2.3. Detection boxes are predicted from the features of the ‘conv4-3’, ‘fc7’, ‘conv6-2’, ‘conv7-2’, ‘conv8-2’, ‘conv9-2’ layers, each representing grids at different levels for the detection boxes (the grid sizes are $38 \times 38$, $19 \times 19$, $10 \times 10$, $5 \times 5$ and $3 \times 3$, respectively).
<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Filters</th>
<th>Kernel</th>
<th>Strides</th>
<th>Output shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>input</td>
<td>3</td>
<td>3 x 3</td>
<td></td>
<td>300 x 300</td>
</tr>
<tr>
<td>Conv</td>
<td>conv1-1</td>
<td>64</td>
<td>3 x 3</td>
<td></td>
<td>300 x 300</td>
</tr>
<tr>
<td>Conv</td>
<td>conv1-2</td>
<td>64</td>
<td>3 x 3</td>
<td></td>
<td>300 x 300</td>
</tr>
<tr>
<td>Maxpool</td>
<td>pool1</td>
<td>2 x 2</td>
<td>2 x 2</td>
<td>2 x 2</td>
<td>150 x 150</td>
</tr>
<tr>
<td>Conv</td>
<td>conv2-1</td>
<td>128</td>
<td>3 x 3</td>
<td></td>
<td>150 x 150</td>
</tr>
<tr>
<td>Conv</td>
<td>conv2-2</td>
<td>128</td>
<td>3 x 3</td>
<td></td>
<td>150 x 150</td>
</tr>
<tr>
<td>Maxpool</td>
<td>pool2</td>
<td>2 x 2</td>
<td>2 x 2</td>
<td>2 x 2</td>
<td>75 x 75</td>
</tr>
<tr>
<td>Conv</td>
<td>conv3-1</td>
<td>256</td>
<td>3 x 3</td>
<td></td>
<td>75 x 75</td>
</tr>
<tr>
<td>Conv</td>
<td>conv3-2</td>
<td>256</td>
<td>3 x 3</td>
<td></td>
<td>75 x 75</td>
</tr>
<tr>
<td>Conv</td>
<td>conv3-3</td>
<td>256</td>
<td>3 x 3</td>
<td></td>
<td>75 x 75</td>
</tr>
<tr>
<td>Maxpool</td>
<td>pool3</td>
<td>2 x 2</td>
<td>2 x 2</td>
<td>2 x 2</td>
<td>38 x 38</td>
</tr>
<tr>
<td>Conv</td>
<td>conv4-1</td>
<td>512</td>
<td>3 x 3</td>
<td></td>
<td>38 x 38</td>
</tr>
<tr>
<td>Conv</td>
<td>conv4-2</td>
<td>512</td>
<td>3 x 3</td>
<td></td>
<td>38 x 38</td>
</tr>
<tr>
<td>Conv</td>
<td>conv4-3</td>
<td>512</td>
<td>3 x 3</td>
<td></td>
<td>38 x 38</td>
</tr>
<tr>
<td>Maxpool</td>
<td>pool4</td>
<td>2 x 2</td>
<td>2 x 2</td>
<td>2 x 2</td>
<td>19 x 19</td>
</tr>
<tr>
<td>Conv</td>
<td>conv5-1</td>
<td>512</td>
<td>3 x 3</td>
<td></td>
<td>19 x 19</td>
</tr>
<tr>
<td>Conv</td>
<td>conv5-2</td>
<td>512</td>
<td>3 x 3</td>
<td></td>
<td>19 x 19</td>
</tr>
<tr>
<td>Conv</td>
<td>conv5-3</td>
<td>512</td>
<td>3 x 3</td>
<td></td>
<td>19 x 19</td>
</tr>
<tr>
<td>Maxpool</td>
<td>pool5</td>
<td>3 x 3</td>
<td>1 x 1</td>
<td></td>
<td>19 x 19</td>
</tr>
<tr>
<td>Conv</td>
<td>fc6*</td>
<td>1024</td>
<td>3 x 3</td>
<td></td>
<td>19 x 19</td>
</tr>
<tr>
<td>Conv</td>
<td>fc7</td>
<td>1024</td>
<td>1 x 1</td>
<td></td>
<td>19 x 19</td>
</tr>
<tr>
<td>Conv</td>
<td>conv6-1</td>
<td>256</td>
<td>1 x 1</td>
<td></td>
<td>19 x 19</td>
</tr>
<tr>
<td>Conv</td>
<td>conv6-2</td>
<td>512</td>
<td>3 x 3</td>
<td>2 x 2</td>
<td>10 x 10</td>
</tr>
<tr>
<td>Conv</td>
<td>conv7-1</td>
<td>128</td>
<td>1 x 1</td>
<td></td>
<td>10 x 10</td>
</tr>
<tr>
<td>Conv</td>
<td>conv7-2</td>
<td>256</td>
<td>3 x 3</td>
<td>2 x 2</td>
<td>5 x 5</td>
</tr>
<tr>
<td>Conv</td>
<td>conv8-1</td>
<td>128</td>
<td>1 x 1</td>
<td></td>
<td>5 x 5</td>
</tr>
<tr>
<td>Conv</td>
<td>conv8-2</td>
<td>256</td>
<td>3 x 3</td>
<td>2 x 2</td>
<td>3 x 3</td>
</tr>
<tr>
<td>Conv</td>
<td>conv9-1</td>
<td>128</td>
<td>1 x 1</td>
<td></td>
<td>3 x 3</td>
</tr>
<tr>
<td>Conv</td>
<td>conv9-2</td>
<td>128</td>
<td>3 x 3</td>
<td></td>
<td>3 x 3</td>
</tr>
</tbody>
</table>

Table 4.2.3: SSD main architecture.
Each cell of the grids holds the bounding boxes and their class scores. The bounding boxes and the class scores are predicted by convolutional layers connected to each of the mentioned layers. Each convolutional layer is of kernel size $3 \times 3$ with $4 \times (4 + 81) = 340$ filters (4 bounding boxes with 4 values for bounding boxes and 81 classes).

Unlike YOLO, SSD does not perform batch normalization after each layer.

**SSD Input Format**

The input format for SSD is mostly identical to VGG16. Just like VGG16, the input images are subtracted by ImageNet mean. The only difference is that SSD accepts an input image of size $300 \times 300$.

### 4.3 Semantic Stream

We present the semantic stream, a stream that predicts the human actions in RGB images depending on the predicted object detections from YOLO or SSD. Similar to the spatial stream, the input of the semantic stream is one RGB image.

A convenient property with both YOLO and SSD is that the size of the output is fixed, no matter how many objects are detected. This makes it trivial to extend the models for human action recognition just by adding new layers.
4.3.1 Where to connect YOLO

There are multiple ways of connecting the additional layers of the YOLO network for the semantic stream. The most straightforward connection point is the final region layer. Then the input of the semantic stream will be the predicted object detections.

An alternative is to connect the semantic stream at earlier layers. The output of earlier layers might hold more information, at the expense of a more complex representation.

For YOLOv2, we consider three different connection points for the semantic stream, the 'region' layer, 'conv7-1' layer and the concatenation of the 'conv5-5' and 'conv6-7' layers. The reason why we consider the concatenation of 'conv5-5' and 'conv6-7' is that the input of the 'conv7-1' layer uses the output from both of these layers.

For YOLO9000, we consider only one connection point for the semantic stream, the 'conv6-5' layer. Unfortunately, we cannot use the 'region' layer since its output is apparently too large for our equipment (the GPUs became unresponsive after training a few hours).

4.3.2 Where to connect SSD

Similarly to YOLO we will investigate different connection points for the semantic stream. However, one major difference with SSD is that the object detections of SSD is based on the output of multiple layers at different levels.

The input of SSDs ‘region’ layer are based on the ‘conv4-3’, 'fc7', 'conv6-2', 'conv7-2', 'conv8-2' and 'conv9-2' layers. So besides from the ‘region’ layer, we will connect to all of the mentioned layers to the semantic stream.

4.3.3 Architecture of Semantic Stream

We consider a few architectures for the semantic stream. The easiest form of the semantic stream is just a softmax layer. A good result on the softmax layer indicates that the object detection features are linearly separable on human action recognition.

To avoid overfitting, we see how different dropout rates affect the performance of the semantic stream. We also consider how the semantic stream benefits from adding additional hidden layers.
The parameters of YOLO/SSD networks are frozen during training. This is to enforce that the semantic stream is in a different feature space from the spatial stream and the temporal stream.

### 4.3.4 Fusion of Streams

In [14] a few functions for fusing the spatial and the temporal stream were investigated at different fusion layers. It was concluded that the sum of the softmax layers is the most efficient approach. We follow this conclusion and fuse the three streams by summing their softmax layers.

### 4.4 Divergence Enhancement

The ultimate goal with the semantic stream is that it will provide complementary information to the spatial and the temporal streams. The semantic stream does not necessarily need to perform well on its own. It is more important that it makes different kinds of errors to the spatial and the temporal stream.

We will therefore investigate a few ways to enforce the predictions diverge from each other.

Generating new models to be complementary to other models is not new. The most well-known example is AdaBoost [16], a boosting technique for weak classifiers. AdaBoost iteratively generates new models focused to complement the previous iterations of the model.
For speech recognition [5] investigated Minimum Bayes Risk Lever-
aging for training complementary models. The technique successfully
created complementary models and reduced the error rates compared
to individually trained models.

We will investigate some alternative techniques to enhance the di-
vergence between the streams. These techniques only add a new term
to the loss function.

We only train the semantic stream when using the divergence en-
hancement techniques. All parameters of the spatial and the temporal
stream are frozen. It would certainly be interesting to jointly train the
spatial and the temporal stream together with the semantic stream as
well, but this would require immense computational resources.

4.4.1 Average Layer

Possibly the most trivial approach is to train the semantic stream jointly
together with the spatial and the temporal stream.

\[
\mathcal{L}_{\text{average}}(\tilde{y}_{\text{sem}}, y) = \mathcal{L}_{\text{cross}} \left( \frac{w_0\tilde{y}_{\text{sem}} + w_1\tilde{y}_{\text{spa}} + w_2\tilde{y}_{\text{tem}}}{w_0 + w_1 + w_2}, y \right)
\]  

(4.4.1)

The weights \(w_0\), \(w_1\) and \(w_2\) are used to balance the average of the
streams. The spatial and the temporal stream both perform too well
on the training set and thus overshadow the predictions of the seman-
tic stream. We therefore set \(w_0 = 1\) and \(w_1 = w_2 = 0.25\). This gives the
semantic stream more influence, even if the predictions of the spatial
and the temporal stream are very confident on the true value.

The idea with this technique is to simulate the validation of the
model. If both the spatial and the temporal stream are both confident
on the correct answer on one sample, the semantic stream do not need
to be as confident with its answer in order to minimize the loss of the
sample.

4.4.2 Kullback-Leibler Divergence

Another approach is to use a divergence measurement to enforce di-
vergence between the predicted streams. One popular divergence mea-
urement for probability distributions is the Kullback-Leibler diver-
gence \( KL \) [27], defined for discrete probability distributions as

\[
KL(p||q) = \sum_i p_i \log \frac{p_i}{q_i}
\]  

(4.4.2)
where \( p \) and \( q \) are two discrete probability distributions (of the same size). \( KL(p||q) \) grows as \( p \) and \( q \) diverges, and \( KL(p||q) = 0 \) when \( p = q \). Note that \( KL(p||q) \neq KL(q||p) \).

We define the loss function \( L_{KL} \) as

\[
L_{KL}(\tilde{y}_{sem}, y) = L_{cross}(\tilde{y}_{sem}, y) - \lambda_{KL} KL \left( \tilde{y}_{sem}, \frac{\tilde{y}_{spa} + \tilde{y}_{tem}}{2} \right) \tag{4.4.3}
\]

where \( \lambda_{KL} \) is a scaling factor controlling how much the Kullback-Leibler divergence influences the loss.

### 4.4.3 Dot Product Divergence

A more geometric approach is to minimize the dot product between the predicted distributions. If the predictions are viewed in a vector space, the dot-product is minimized when the vectors are orthogonal. A geometric interpretation is that two probabilities are divergent when their angle is as large in the probability distribution vector space. We define the loss function \( L_{dot} \)

\[
L_{dot}(\tilde{y}_{sem}, y) = L_{cross}(\tilde{y}_{sem}, y) + \lambda_{dot} \tilde{y}_{sem} \cdot (\tilde{y}_{spa} + \tilde{y}_{tem}) \tag{4.4.4}
\]

where \( \lambda_{dot} \) is a scaling factor controlling how much the dot product influences the loss.

### 4.4.4 Combining Average with Divergence

We also consider two mixtures of the above techniques — for \( L_{dot} \) and \( L_{KL} \), we replace \( L_{cross} \) with \( L_{average} \). We define \( L_{average+dot} \) and \( L_{average+KL} \) as

\[
L_{average+dot}(\tilde{y}_{sem}, y) = L_{average}(\tilde{y}_{sem}, y) + \lambda_{dot} \tilde{y}_{sem} \cdot (\tilde{y}_{spa} + \tilde{y}_{tem}) \tag{4.4.5}
\]

and

\[
L_{average+KL}(\tilde{y}_{sem}, y) = L_{average}(\tilde{y}_{sem}, y) - \lambda_{KL} KL \left( \tilde{y}_{sem}, \frac{\tilde{y}_{spa} + \tilde{y}_{tem}}{2} \right) \tag{4.4.6}
\]
4.5 Implementation Details

4.5.1 Training

Sampling

The models are trained in a similar procedure as described in [14]. The clips are uniformly sampled across the clip categories. From each sampled clip a single frame is randomly selected uniformly. The single frame is randomly cropped and horizontally flipped.

For training the temporal stream we uniformly sample $L$ consecutive frames of the precomputed optical flow of the clip. The sequence of the optical flows is spatially randomly cropped and horizontally flipped.

Batch Size

We use a batch size of 64 for all experiments. When training the spatial and the temporal spatial stream we divide the batch across two GPUs. The weights for each iteration is updated with the average gradient of the batches across the multiple GPUs. For the semantic stream we use only one GPU.

Epochs

The training is divided into epochs. One epoch corresponds to 100 iterations of SGD parameter update. After each epoch the average validation loss is computed on 12,800 inputs randomly sampled from the validation set.

Learning Rate Decay

The learning rate is decreased by a factor of 0.1 if the validation loss has not decreased for 3 epochs. The training is terminated when the validation loss has not been decreased for more than 10 epochs. A final evaluation on a clip basis is made with the model with the lowest validation loss (see section 4.5.3). Unless mentioned, the starting learning rate is $5 \times 10^{-3}$. 
Weight Decay

The weight decay is set to $5 \times 10^{-3}$ for the weight parameters, and 0.0 for the bias parameters for all trained models.

4.5.2 Datasets

We use two datasets for human action recognition, UCF-101 and HMDB-51. These two are currently the most common sets for human action recognition.

We use the pre-computed optical flow of UCF-101 and HMDB-51 provided by [14].

UCF-101

UCF-101 is a collection of 13,320 clips downloaded from YouTube with a total of 27 hours of video data. The clips are typically in unconstrained environments. With varying lighting, camera motion, partial occlusion and low quality frames makes the dataset challenging and realistic [52].

The clips are labeled with 101 different action categories. The actions are according to the authors divided into five different types: 1) Human-object interaction; 2) Body-motion only; 3) Human-human interaction; 4) Playing musical instruments; 5) Sports. However, there is no additional labeling for these five types. The different action categories are depicted in table A.2.1.

Each action class is divided into 25 different groups, where each group consist of 4-7 clips. The clips in each group share common features such as actors and sceneries. Typically the groups represent clips from the same video. There are 3 training/validation splits for UCF-101.

HMDB-51

Human Motion Database (HMDB-51) is another dataset for human action recognition proven to be more challenging than UCF-101. HMDB-51 consists 6,766 videos labeled with 51 different action categories, which are depicted in table A.1.1 [26]. Each category consists of at least 101 clips.

The actions are according to the authors divided into five types: 1) General facial actions; 2) Facial actions with object manipulation; 3)
General body movements; 4) Body movements with object interaction; 5) Body movements with human interaction.

The videos are collected from digitized movies, public databases, YouTube, Google Videos and other videos collected from the internet.

### 4.5.3 Evaluation

We follow the same evaluation protocol as used in [50]. For each clip, we sample 25 frames with spaced equally temporally. For each frame, we crop 5 different regions: one in each corner and one in the center. We also flip each of the crops so we get in total 10 different inputs for each frame. In total, each stream makes 250 different predictions for each clip.

The class scores for the clip is then obtained by average the predictions of the sampled crops.

### 4.5.4 Equipment

We use Keras 2.0 [7] with Tensorflow backend [9] for the implementation, training and evaluation of the models. GeForce GTX 1080 GPUs are used for our experiments.

Our SSD implementation is based on the Keras port by [47]. We ported the YOLO weights from its original implementation in Darknet to Keras [43]. The weights of the spatial and the temporal streams for UCF-101 are ported from its original implementation in MatConvNet to Keras [14]. For HMDB-51 we needed to train the spatial and the temporal stream by ourselves.
Chapter 5

Results

Our experimental results are reported in this chapter. First in section 5.1 we set up the spatial and the temporal stream in our working environment to match the results reported in [14] as closely as possible.

Section 5.2 explores the possibilities to predict human actions with object detections by investigating which kind of objects are detected in for different human actions.

Section 5.3 finally investigates different architectures for the semantic stream. Initially we perform all experiments on UCF-101 split 1. The most promising models will be then used for the remainder of the UCF-101 splits and all of the HMDB-51 splits.

5.1 Setting up the Spatial and the Temporal Streams

Present the results from our trained VGG16 models. Discuss briefly why there might be some differences. Better results were reported in [61] if the temporal stream is counted twice as much as the spatial stream. We do the same for our own experiments.

We use the same weights as twostream fusion paper for the models trained on the UCF-101 dataset. The weights are converted to Keras 2.0 with Tensorflow backend. The original weights were trained with MatConvNet.

Unfortunately we found no publicly available weights for the spatial and the temporal stream pre-trained on the HMDB-51 dataset. We therefore train the weights for HMDB-51 by ourselves. This is done by
Table 5.1.1: Differences between the reported accuracy of the streams in [14] and the observed accuracy with our implementation.

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>T</th>
<th>S + T</th>
<th>S + 2T</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCF-101 split 1 [14]</td>
<td>82.6%</td>
<td>86.3%</td>
<td>90.6%</td>
<td></td>
</tr>
<tr>
<td>UCF-101 split average [14]</td>
<td></td>
<td></td>
<td></td>
<td>91.7%</td>
</tr>
<tr>
<td>HMDB-51 split 1 [14]</td>
<td>46.1%</td>
<td>55.2%</td>
<td>58.2%</td>
<td></td>
</tr>
<tr>
<td>HMDB-51 split average [14]</td>
<td></td>
<td></td>
<td></td>
<td>58.7%</td>
</tr>
<tr>
<td>Ours UCF-101 split 1</td>
<td>82.77%</td>
<td>86.44%</td>
<td>89.93%</td>
<td>90.96%</td>
</tr>
<tr>
<td>Ours UCF-101 split average</td>
<td>81.45%</td>
<td>87.81%</td>
<td>89.75%</td>
<td>91.07%</td>
</tr>
<tr>
<td>Ours HMDB-51 split 1</td>
<td>40.58%</td>
<td>50.45%</td>
<td>54.83%</td>
<td>56.21%</td>
</tr>
<tr>
<td>Ours HMDB-51 split average</td>
<td>39.06%</td>
<td>48.43%</td>
<td>52.57%</td>
<td>54.90%</td>
</tr>
</tbody>
</table>

fine-tuning the pre-trained weights from UCF-101. We compare the weights reported from [14] with our implementation in table 5.1.1.

Our scores a bit off from the scores reported by [14]. It is unknown why the results are worse. A possible cause can be slight implementation differences between Keras/Tensorflow and MatConvNet. Another possible cause can be slight differences in the evaluation scheme.

5.2 Objects Detected in Clips

To get a better idea of how effective YOLO and SSD can detect the actions in the clips, we determine the top 10 most detected objects for each action. A great diversity of objects between the actions indicates that most of the information of the action is maintained through the network.

This is done by sampling 25 frames with equal temporal spacing from each clip. For each frame we compute the final prediction layer of YOLO and SSD. For each detection box we compute a certainty vector by multiplying the predicted probability vector with the certainty scalar. Each certainty vector is then summed across all detection boxes for each action.

To express it more mathematically, let $C_i$ be the number of clips of action class $i$, $D$ be the number of detection boxes generated for each frame, $c_{i,f,d}$ and $p_{c,i,f,d}$ the certainty scalar and probability vector of detection box $d$ in frame $f$ in clip $c$ of action $i$. Then we have the
total certainty $c_{i, tot}$ for action $i$ defined as

$$c_{i, tot} = \sum_{c=0}^{C_i} \sum_{f=0}^{25} \sum_{d=0}^{D_i} c_{c, f, d} \cdot p_{c, f, d}$$  \hspace{1cm} (5.2.1)

We then take the top 10 largest values of each $c_{i, tot}$. Figures 5.2.1 present the object detection matrices.

We can see YOLO detects a more diverse set of objects than SSD in both UCF-101 and HMDB-51. Three classes are always appearing in the top 10 for all actions: ‘background’ (with SSD), ‘person’ and ‘chair’. The fact that ‘person’ is commonly detected in the clips indicates that the object detection systems works as intended; all clips contains at least one person. We are not sure why ‘chair’ is so commonly detected. Many clips do not include a chair at all.

We believe this is due to that SSD is trained with a special target class for backgrounds. Uncertain predictions might be concentrated on the background class. YOLO will always try to predict the target class, even if it is uncertain about the prediction.

The classes of UCF-101, HMDB-51 and COCO are listed in appendix A. The same order of classes are used in figure 5.2.1.

5.3 Experiments

5.3.1 Where to Connect the Semantic Stream

We now investigate how well the different object detection systems represents the human actions by connecting a softmax layer to different layers of the object detection systems (as described in sections 4.3.1 and 4.3.2). This experiment will show how linearly separable the features for object detection are for the action recognition problem.

The initial learning rate for YOLOv2 and SSD is set to 0.0005. YOLO9000 requires an initial learning rate as low as 0.000005. The loss diverges for higher learning rates.

The results are presented in table 5.3.1. Individually, YOLO9000 yields the most accurate prediction. Together with the spatial and the temporal streams, YOLOv2 is consequently more accurate than YOLO9000. Overall, the ‘conv7-1’ layer of YOLOv2 yields the best results when the semantic stream predictions are combined with one spatial stream and two spatial streams.
Figure 5.2.1: Top 10 detections on UCF-101 split 1. Darker squares represent larger representation of the detection class. The rows represents the actions of UCF-101, and the columns represents the objects of COCO.
Figure 5.2.2: Top 10 detections on HMDB-51 split 1. Darker squares represent larger representation of the detection class. The rows represents the actions of HMDB-51, and the columns represents the objects of COCO.
Table 5.3.1: Results from UCF-101 split 1 by connecting a softmax layer to different layers.

SSD on the other hand yields the poorest results. This indicates that the architecture of YOLO is the best feature representation for human action recognition.

For our continued experiments the semantic stream is connected to the ‘conv7-1’ layer of YOLOv2.

5.3.2 Dropout Rates

In this subsection we attempt to improve the generality of the semantic stream by using dropout on its output. The affect of different dropout rates is presented in table 5.3.2.

Individually, the semantic stream performs best with dropout rate 10%. Together with the other two streams, dropout rate 50% consequently outperforms other dropout rates.

5.3.3 Adding Additional Hidden Layers

We explore different architectures for the semantic stream by adding additional hidden layers. Five different architectures are considered. All hidden convolutional and fully connected layers are activated with ReLU.

1. softmax: No hidden layers are used. The model is a dropout layer with dropout rate 50% followed by a softmax layer. The same as the best architecture in section 5.3.2.

\(^1\)O: Semantic stream (with Object detections)
Dropout rate & O & O+S & O+T & O+S+T & O+S+2T \\
0% & 70.95% & 82.84% & 88.90% & 89.24% & 90.80% \\
10% & 72.96% & 83.06% & 88.63% & 89.59% & 90.80% \\
25% & 72.40% & 83.32% & 88.74% & 89.27% & 90.80% \\
50% & 72.59% & 83.32% & 89.03% & 89.61% & 91.14% \\
75% & 72.05% & 82.82% & 88.61% & 89.43% & 90.83% \\
90% & 69.83% & 82.13% & 87.07% & 88.58% & 90.40% \\

Table 5.3.2: The effect of different dropout rates for YOLOv2 with 'conv7-1' as connection point for the semantic stream (UCF-101 split 1).

2. fc1024: Fully connected layer with 1024 nodes is used as a hidden layer. The fully connected layer is followed by a dropout layer with dropout rate 50% and a softmax layer.

3. 2×fc1024: Two fully connected layers with 1024 are used as hidden layers. A dropout layer with dropout rate 50% is used between the two fully connected layers. The hidden layers are followed by a dropout layer with dropout rate 50% and a softmax layer.

4. convolution: One $3 \times 3$ convolutional layer with 1024 filters, one $2 \times 2$ max pooling layer and one fully connected layer with 4096 nodes. The hidden layers are followed by a dropout layer with dropout rate 50% and a softmax layer.

5. 2xconvolution: One $3 \times 3$ convolutional layer with 1024 filters, one $2 \times 2$ max pooling layer, another $3 \times 3$ convolutional layer but with 2048 filters, another $2 \times 2$ max pooling layer and one fully connected layer with 4096 nodes. The hidden layers are followed by a dropout layer with dropout rate 50% and a softmax layer.

The results in table 5.3.3 shows that hidden layers gives small improvements in the if the semantic stream is used individually and in the O + S, O + T and O + S + T settings. Using no hidden layers is best for the O + S + 2T setting.
Table 5.3.3: The effect of different architectures for YOLOv2 with ‘conv7-1’ as connection point for the semantic stream (UCF-101 split 1).

5.3.4 Divergence Enhancement

We explore how the different divergence enhancement methods from section 4.4 affects the training. We initialize the models with the weights trained for YOLOv2 with dropout rate 50% trained in section 5.3.2. The configurations we consider are

1. average: The categorical cross entropy is replaced with the average layer as described in section 4.4.1.
2. dot 0.1: The dot product divergence as described in section 4.4.3 with $\lambda_{dot} = 0.1$.
3. dot 0.5: The dot product divergence as described in section 4.4.3 with $\lambda_{dot} = 0.5$.
4. KL 0.005: The Kullback-Leibler divergence as described in section 4.4.2 with $\lambda_{KL} = 0.005$.
5. KL 0.01: The Kullback-Leibler divergence as described in section 4.4.2 with $\lambda_{KL} = 0.01$.

Besides from the above listed configurations, we also consider a combination with the average layer together with the dot product divergence and the Kullback Leibler divergence as described in section 4.4.4. The same values of $\lambda_{dot}$ and $\lambda_{KL}$ is used as above.

The results in table 5.3.4 shows that dot product divergence with $\lambda_{dot} = 0.5$ is the most effective on the O + S + 2T setting.
Table 5.3.4: The effect of the different divergence enhancement methods on UCF-101 split 1. YOLOv2 with ‘conv7-1’ as connection point and dropout rate 50% is used.

<table>
<thead>
<tr>
<th>Divergence Enhancement</th>
<th>O</th>
<th>O + S</th>
<th>O + T</th>
<th>O + S + T</th>
<th>O + S + 2T</th>
</tr>
</thead>
<tbody>
<tr>
<td>no divergence enhancement</td>
<td>72.59%</td>
<td>82.32%</td>
<td><strong>89.03%</strong></td>
<td>89.61%</td>
<td>91.14%</td>
</tr>
<tr>
<td>average</td>
<td>71.72%</td>
<td>82.87%</td>
<td>88.63%</td>
<td>89.45%</td>
<td>90.80%</td>
</tr>
<tr>
<td>dot 0.1</td>
<td>72.59%</td>
<td>83.21%</td>
<td>88.98%</td>
<td>89.74%</td>
<td>91.22%</td>
</tr>
<tr>
<td>dot 0.5</td>
<td>72.48%</td>
<td>83.27%</td>
<td>88.95%</td>
<td><strong>89.80%</strong></td>
<td><strong>91.28%</strong></td>
</tr>
<tr>
<td>KL 0.005</td>
<td>73.46%</td>
<td><strong>83.29%</strong></td>
<td><strong>89.03%</strong></td>
<td>89.37%</td>
<td>91.01%</td>
</tr>
<tr>
<td>KL 0.01</td>
<td><strong>73.51%</strong></td>
<td>83.03%</td>
<td><strong>89.03%</strong></td>
<td>89.56%</td>
<td>90.99%</td>
</tr>
<tr>
<td>average + dot 0.1</td>
<td>72.53%</td>
<td>83.19%</td>
<td>88.69%</td>
<td>89.58%</td>
<td>90.91%</td>
</tr>
<tr>
<td>average + dot 0.5</td>
<td>68.01%</td>
<td>82.32%</td>
<td>88.08%</td>
<td>89.37%</td>
<td>91.20%</td>
</tr>
<tr>
<td>average + KL 0.005</td>
<td>72.24%</td>
<td>82.98%</td>
<td>88.87%</td>
<td>89.43%</td>
<td>91.07%</td>
</tr>
<tr>
<td>average + KL 0.01</td>
<td>71.98%</td>
<td>83.14%</td>
<td>88.50%</td>
<td>89.53%</td>
<td>90.85%</td>
</tr>
</tbody>
</table>

5.3.5 Experiments on HMDB-51

We now investigate how well the most promising models on UCF-101 performs on HMDB-51. Four models for the semantic stream are investigated.

1. YOLOv2 conv7-1: A softmax layer directly connected to the ‘conv7-1’ layer of YOLOv2.

2. Dropout 50%: Same as above, but with a dropout layer added between ‘conv7-1’ and the softmax layer.

3. Dot 0.5: Same as above, but trained with dot product divergence as described in section 4.4.3 with $\lambda_{dot} = 0.5$.

4. KL 0.001: Same as 2, but trained with Kullback-Leibler divergence as described in section 4.4.2 with $\lambda_{KL} = 0.001$.

The results in table 5.3.5 indicate that using dot product divergence is not as effective as in UCF-101. Instead, only using a dropout layer 50% seem to be most effective. We can also observe that as in UCF-101, KL 0.001 gives small improvements to the semantic stream individually.
5.3.6 Mean Accuracy over all Three Splits

Finally we evaluate the following three models over all three splits on UCF-101 and HMDB-51: 'YOLOv2 conv7-1', 'Dropout 50%' and 'Dot 0.5'. The results are depicted in table 5.3.6.

5.3.7 Per Class Accuracy

We now study which actions perform well under the semantic stream, and which that does not. We use YOLOv2 'conv7-1' with dropout 50% as a semantic stream for the following experiments. Figure 5.3.1 depicts the recall rate of the semantic stream. We see that the semantic stream perfectly recognizes the classes 'Cutting in kitchen', 'Surfing', 'Knitting', 'Writing on board', 'Billiards', 'Playing dhol', 'Playing tabla', 'Fencing', 'Basketball', 'Diving', 'Horse riding', 'Drumming', 'Volleyball spiking', 'Baby crawling' and 'Breaststroke'.

The three worst recognized classes are in order 'Jumping jack', 'Nunchucks' and 'Boxing punching bag'. 'Jumping jack' contains in general no human-object interaction, so it is understandable that the semantic stream has difficulties to recognize the action. 'Nunchucks' is an human-object interaction, which should be an action the semantic stream should be good at classifying. However, there are no observed nunchuck represented in the COCO dataset, which might cause some confusion. 'Boxing punching bag' is in general mixed up with 'Boxing speed bag' as can be seen in the confusion matrix in figure 5.3.2.

The results on HMDB-51 are evidently worse. Two actions were never correctly recognized: 'Smile' and 'Ride bike'. It is not surprising that 'Smile' is difficult to recognize for the semantic stream — the semantic stream is unable to capture facial expressions. However, it is quite surprising that 'Ride bike' is difficult to recognize, especially since 'Bicycle' is in the COCO dataset. YOLO is also able to recog-
<table>
<thead>
<tr>
<th>Experiment</th>
<th>O</th>
<th>O+S</th>
<th>O+T</th>
<th>O+S+T</th>
<th>O+S+2T</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>YOLOv2 conv7-1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCF-101 Split 1</td>
<td>70.95%</td>
<td>82.84%</td>
<td>88.90%</td>
<td>89.24%</td>
<td>90.80%</td>
</tr>
<tr>
<td>UCF-101 Split 2</td>
<td>65.91%</td>
<td>80.02%</td>
<td>88.51%</td>
<td>88.16%</td>
<td>90.17%</td>
</tr>
<tr>
<td>UCF-101 Split 3</td>
<td>70.31%</td>
<td>81.85%</td>
<td>90.77%</td>
<td>89.29%</td>
<td>92.18%</td>
</tr>
<tr>
<td>UCF-101 Split Average</td>
<td>69.06%</td>
<td>81.57%</td>
<td>89.39%</td>
<td>88.90%</td>
<td>91.05%</td>
</tr>
<tr>
<td>HMDB-51 Split 1</td>
<td>42.68%</td>
<td>45.62%</td>
<td>54.97%</td>
<td>54.44%</td>
<td>57.25%</td>
</tr>
<tr>
<td>HMDB-51 Split 2</td>
<td>36.93%</td>
<td>43.01%</td>
<td>52.48%</td>
<td>50.52%</td>
<td>54.11%</td>
</tr>
<tr>
<td>HMDB-51 Split 3</td>
<td>39.09%</td>
<td>42.35%</td>
<td>52.16%</td>
<td>50.91%</td>
<td>54.77%</td>
</tr>
<tr>
<td>HMDB-51 Split Average</td>
<td>39.57%</td>
<td>43.66%</td>
<td>53.20%</td>
<td>51.96%</td>
<td>55.38%</td>
</tr>
<tr>
<td><strong>Dropout 50%</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCF-101 Split 1</td>
<td>72.59%</td>
<td>83.32%</td>
<td>89.03%</td>
<td>89.61%</td>
<td>91.14%</td>
</tr>
<tr>
<td>UCF-101 Split 2</td>
<td>71.00%</td>
<td>80.24%</td>
<td>88.00%</td>
<td>87.60%</td>
<td>90.23%</td>
</tr>
<tr>
<td>UCF-101 Split 3</td>
<td>70.81%</td>
<td>81.95%</td>
<td>89.61%</td>
<td>89.12%</td>
<td>92.10%</td>
</tr>
<tr>
<td>UCF-101 Split Average</td>
<td>71.47%</td>
<td>81.84%</td>
<td>88.88%</td>
<td>88.78%</td>
<td>91.16%</td>
</tr>
<tr>
<td>HMDB-51 Split 1</td>
<td>43.86%</td>
<td>45.49%</td>
<td>54.57%</td>
<td>53.79%</td>
<td>57.32%</td>
</tr>
<tr>
<td>HMDB-51 Split 2</td>
<td>38.43%</td>
<td>42.94%</td>
<td>50.33%</td>
<td>50.59%</td>
<td>54.18%</td>
</tr>
<tr>
<td>HMDB-51 Split 3</td>
<td>39.15%</td>
<td>42.42%</td>
<td>51.31%</td>
<td>51.05%</td>
<td>54.90%</td>
</tr>
<tr>
<td>HMDB-51 Split Average</td>
<td>40.48%</td>
<td>43.62%</td>
<td>52.07%</td>
<td>51.81%</td>
<td>55.47%</td>
</tr>
<tr>
<td><strong>Dot 0.5</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCF-101 Split 1</td>
<td>72.48%</td>
<td>83.27%</td>
<td>88.95%</td>
<td>89.80%</td>
<td>91.28%</td>
</tr>
<tr>
<td>UCF-101 Split 2</td>
<td>72.36%</td>
<td>80.29%</td>
<td>88.46%</td>
<td>87.81%</td>
<td>90.28%</td>
</tr>
<tr>
<td>UCF-101 Split 3</td>
<td>71.94%</td>
<td>81.93%</td>
<td>90.21%</td>
<td>89.29%</td>
<td>91.94%</td>
</tr>
<tr>
<td>UCF-101 Split Average</td>
<td>72.26%</td>
<td>81.83%</td>
<td>89.21%</td>
<td>88.97%</td>
<td>91.17%</td>
</tr>
<tr>
<td>HMDB-51 Split 1</td>
<td>43.46%</td>
<td>46.80%</td>
<td>53.99%</td>
<td>53.92%</td>
<td>56.41%</td>
</tr>
<tr>
<td>HMDB-51 Split 2</td>
<td>39.02%</td>
<td>42.94%</td>
<td>50.33%</td>
<td>51.18%</td>
<td>53.99%</td>
</tr>
<tr>
<td>HMDB-51 Split 3</td>
<td>38.43%</td>
<td>41.57%</td>
<td>50.98%</td>
<td>50.39%</td>
<td>54.64%</td>
</tr>
<tr>
<td>HMDB-51 Split Average</td>
<td>40.30%</td>
<td>43.77%</td>
<td>51.77%</td>
<td>51.83%</td>
<td>55.01%</td>
</tr>
</tbody>
</table>

Table 5.3.6: Collective results over all datasets and splits.
Figure 5.3.1: Recall rate of different classes with YOLOv2 ‘conv7-1’ dropout 50% on UCF-101 split 1.

nize bicycles in the ‘Ride bike’ actions of HMDB-51 according to the detection matrix in figure 5.2.2.
Figure 5.3.2: Confusion matrix with YOLOv2 ‘conv7-1’ dropout 50% on UCF-101 split 1.
Figure 5.3.3: Recall rate of different classes with YOLOv2 'conv7-1' dropout 50% on HMDB-51 split 1.

Figure 5.3.4: Confusion matrix with YOLOv2 'conv7-1' dropout 50% on HMDB-51 split 1.
Chapter 6

Discussion

Overall our experiments indicate that the semantic stream achieves a small improvement to the two-stream convolutional network. We believe the results could potentially be further improved if the architectures are further developed.

A potential drawback of the semantic stream is its inaccuracy when used individually.

6.1 Comparison of Object Detection Systems

Our experiments indicate that SSD is not suitable for predicting action recognition. A possible cause is that SSD will predict a region as a background if all of other object categories are unlikely. Important information for action recognition purposes might be filtered out by this behavior.

The YOLO models might be better in this case because a prediction is always made, even if the prediction is uncertain. For example, YOLO might incorrectly detect a certain object as a cat. This certain object might not be included in the COCO dataset. The semantic stream could without loss of performance associate the action with the appearance of a cat — even if no cat actually appears in the videos.

6.2 Comparison with State-of-the-Art

Our results do not break the current state-of-the-art in human action recognition, but is comparable at least for UCF-101 (table 6.2.1. A
60  CHAPTER 6. DISCUSSION

<table>
<thead>
<tr>
<th>Method</th>
<th>UCF-101</th>
<th>HMDB-51</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two-Stream [50]</td>
<td>88.0%</td>
<td>59.4%</td>
</tr>
<tr>
<td>Two-Stream [14]</td>
<td>91.7%</td>
<td>58.7%</td>
</tr>
<tr>
<td>Two-Stream Fusion [14]</td>
<td>92.5%</td>
<td>65.4%</td>
</tr>
<tr>
<td>Temporal Segment Networks [59]</td>
<td>94.2%</td>
<td>69.4%</td>
</tr>
<tr>
<td>Chained Multi-Stream Networks [63]</td>
<td>91.1%</td>
<td>69.7%</td>
</tr>
<tr>
<td>Key Volume Mining Framework [62]</td>
<td>93.1%</td>
<td>63.3%</td>
</tr>
<tr>
<td>Ours (Dropout 50%)</td>
<td>91.16%</td>
<td>55.47%</td>
</tr>
<tr>
<td>Ours (Dot 0.5)</td>
<td>91.17%</td>
<td>55.01%</td>
</tr>
</tbody>
</table>

Table 6.2.1: Mean classification accuracy over three splits in some current state of the art methods.

reason why HMDB-51 is very low compared to the state-of-the-art is because our spatial and temporal streams are inaccurate in our setup.

It is unknown why our spatial and temporal streams does not perform well on HMDB-51, while the streams perform well on UCF-101. A potential cause is that there are slight differences in the training procedure from [14]. However, we managed to train the spatial and the temporal stream to similar levels as [14] on UCF-101, so it is possible there might be some other cause for the differences.

6.3 Directions of Improvement

We believe there is much room for improvement. Due to time constraints we were not able to fully investigate the proposed models and techniques in this thesis. We suggest a few interesting directions to further improve our work.

6.3.1 Temporal Setting

We only applied the semantic stream on a single-frame basis. Further progress can be made if the semantic stream predictions are made on a multi-frame basis in a temporal setting. This would allow the semantic stream to capture the evolution of the object detections over a sequence of frames.

A potential challenge with the temporal setting is to correctly associate the object detections between the frames. We are not sure how
important this issue may be. It is possible that the semantic stream is able to learn the associations by itself.

### 6.3.2 Very Deep Semantic Streams

Our semantic streams were rather shallow. At most three hidden layers were used in our experiments. A potential improvement is to construct even deeper networks for the semantic stream. Our experiments indicated small improvements in some cases when using hidden layers. The very deep networks can be fine tuned using pre-trained weights learned from large-scale datasets like ImageNet.

### 6.3.3 Further Investigations on Divergence Enhancement

The divergence enhancement had little effect on the semantic stream, but the results are still promising. Interestingly, the accuracy of the semantic streams were not decreased significantly after training with divergence enhancement. In some cases the semantic streams even became more accurate.

Our initial intention with the divergence enhancement was that the semantic stream would sacrifice some of its accuracy in order to enhance the accuracy in a three-stream case. This does not seem to be the case except for the ‘average + dot 0.5’ configuration.

Interestingly the dot product divergence did not have any positive effect on HMDB-51. A possible cause for this is the inaccuracy of the spatial and the temporal stream on the dataset. A deeper investigation is required to determine when dot product divergence works, and when it does not.

But possibly most interesting is how the Kullback-Leibler divergence improved the semantic streams on an individual basis. This raises the question if this is always the case, and if the Kullback-Leibler divergence can be used to further enhance trained models.

The methods for divergence enhancement presented here are still in their infancy, and we believe further investigations can be interesting. For example, the spatial and the temporal streams can also be trained with the divergence enhancement techniques. The spatial and the temporal streams are both very accurate already. It would be interesting to investigate how divergence enhancement affects models
which are already accurate.

The divergence enhancement is also applied on data which are already seen by the models. We did not have any unseen data available for our experiments. Training with divergence enhancement on already seen data can give skewed results since the spatial and the temporal streams are both very accurate on the training set. This is a possible cause of why the average layer performed poorly.

### 6.3.4 More Object Categories

One major drawback of the object detection systems used in this thesis is that the object categories in COCO is not very representative of the human actions in UCF-101 and HMDB-51. For example, COCO has the object categories elephant and teddy bear, but no hammer or nunchucks. Some important object categories for certain actions in UCF-101 and HMDB-51 are missing from the COCO dataset.

YOLO9000 partially resolves this issue by being trained on multiple datasets and able to determine the relation between categories thanks to word-net. However, YOLO9000 is only trained on a subset of the object categories it is capable of detecting.

Object detection for human action recognition could probably be improved once it is possible to train object detection systems to recognize a very large number of object categories.
Chapter 7

Conclusion

From our work we can conclude slight improvements when using object detections for human action recognition. We have compared three different one-pass object detection systems. YOLOv2 and YOLO9000 both seem promising for human action recognition. SSD on the other hand is not a very good feature representation of human actions.

Different architectures of the semantic stream were tried. Non-linear models did not provide a big overhead over linear models when the semantic stream is used together with the spatial and the temporal streams.

Our divergence enhancement techniques gave slight improvements. The Kullback-Leibler even had the effect of improving the semantic stream on an individual basis.

We believe we have not reached the full potential of the semantic stream. Deeper networks pre-trained on ImageNet and predicting in a temporal setting are some potential directions to bring the work further. There is definitely room for more work to fully develop the semantic stream.
Bibliography


Appendix A

Dataset Classes

A.1 HMDB-51

<table>
<thead>
<tr>
<th>Action</th>
<th>Action</th>
<th>Action</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brush hair</td>
<td>Fencing</td>
<td>Pull up</td>
<td>Smile</td>
</tr>
<tr>
<td>Cart wheel</td>
<td>Flic flac</td>
<td>Punch</td>
<td>Smoke</td>
</tr>
<tr>
<td>Catch</td>
<td>Golf</td>
<td>Push</td>
<td>Somersault</td>
</tr>
<tr>
<td>Chew</td>
<td>Handstand</td>
<td>Push up</td>
<td>Stand</td>
</tr>
<tr>
<td>Clap</td>
<td>Hit</td>
<td>Ride bike</td>
<td>Swing baseball</td>
</tr>
<tr>
<td>Climb stairs</td>
<td>Hug</td>
<td>Ride horse</td>
<td>Sword exercise</td>
</tr>
<tr>
<td>Climb</td>
<td>Jump</td>
<td>Run</td>
<td>Sword</td>
</tr>
<tr>
<td>Dive</td>
<td>Kick ball</td>
<td>Shake hands</td>
<td>Talk</td>
</tr>
<tr>
<td>Draw sword</td>
<td>Kick</td>
<td>Shoot ball</td>
<td>Throw</td>
</tr>
<tr>
<td>Dribble</td>
<td>Kiss</td>
<td>Shoot bow</td>
<td>Turn</td>
</tr>
<tr>
<td>Drink</td>
<td>Laugh</td>
<td>Shoot gun</td>
<td>Walk</td>
</tr>
<tr>
<td>Eat</td>
<td>Pick</td>
<td>Sit</td>
<td>Wave</td>
</tr>
<tr>
<td>Fall floor</td>
<td>Pour</td>
<td>Sit up</td>
<td></td>
</tr>
</tbody>
</table>

Table A.1.1: The 51 different actions of the HMDB-51 human action recognition dataset.
### A.2 UCF-101

<table>
<thead>
<tr>
<th>Action</th>
<th>Action</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply eye makeup</td>
<td>Handstand walking</td>
<td>Punch</td>
</tr>
<tr>
<td>Apply lipstick</td>
<td>Head massage</td>
<td>Push ups</td>
</tr>
<tr>
<td>Archery</td>
<td>High jump</td>
<td>Rafting</td>
</tr>
<tr>
<td>Baby crawling</td>
<td>Horse race</td>
<td>Rock climbing indoor</td>
</tr>
<tr>
<td>Balance beam</td>
<td>Horse riding</td>
<td>Rope climbing</td>
</tr>
<tr>
<td>Band marching</td>
<td>Hula hoop</td>
<td>Rowing</td>
</tr>
<tr>
<td>Baseball pitch</td>
<td>Ice dancing</td>
<td>Salsa dancing</td>
</tr>
<tr>
<td>Basketball</td>
<td>Javelin throw</td>
<td>Shaving beard</td>
</tr>
<tr>
<td>Basketball dunk</td>
<td>Juggling balls</td>
<td>Shotgun</td>
</tr>
<tr>
<td>Bench press</td>
<td>Jumping jack</td>
<td>Skiing</td>
</tr>
<tr>
<td>Biking</td>
<td>Jump rope</td>
<td>Skijet</td>
</tr>
<tr>
<td>Billiards</td>
<td>Kayaking</td>
<td>Sky diving</td>
</tr>
<tr>
<td>Blow dry hair</td>
<td>Knitting</td>
<td>Soccer diving</td>
</tr>
<tr>
<td>Blowing candles</td>
<td>Long jump</td>
<td>Soccer penalty</td>
</tr>
<tr>
<td>Bodyweight squats</td>
<td>Military parade</td>
<td>Still rings</td>
</tr>
<tr>
<td>Brushing teeth</td>
<td>Mixing</td>
<td>Sumo wrestling</td>
</tr>
<tr>
<td>Clean and jerk</td>
<td>Moping floor</td>
<td>Surfing</td>
</tr>
<tr>
<td>Cliff diving</td>
<td>Nunchucks</td>
<td>Swing</td>
</tr>
<tr>
<td>Cricket bowling</td>
<td>Parallel bars</td>
<td>Table tennis shot</td>
</tr>
<tr>
<td>Cricket shot</td>
<td>Pizza tossing</td>
<td>Tai chi</td>
</tr>
<tr>
<td>Cutting in kitchen</td>
<td>Playing cello</td>
<td>Tennis swing</td>
</tr>
<tr>
<td>Diving</td>
<td>Playing daf</td>
<td>Throw discus</td>
</tr>
<tr>
<td>Drumming</td>
<td>Playing dhol</td>
<td>Trampoline jumping</td>
</tr>
<tr>
<td>Fencing</td>
<td>Playing flute</td>
<td>Typing</td>
</tr>
<tr>
<td>Field hockey penalty</td>
<td>Playing guitar</td>
<td>Uneven bars</td>
</tr>
<tr>
<td>Floor gymnastics</td>
<td>Playing piano</td>
<td>Volleyball spiking</td>
</tr>
<tr>
<td>Frisbee catch</td>
<td>Playing sitar</td>
<td>Walking with dog</td>
</tr>
<tr>
<td>Front crawl</td>
<td>Playing tabla</td>
<td>Wall pushups</td>
</tr>
<tr>
<td>Golf swing</td>
<td>Playing violin</td>
<td>Writing on board</td>
</tr>
<tr>
<td>Haircut</td>
<td>Pole vault</td>
<td>YoYo</td>
</tr>
<tr>
<td>Hammering</td>
<td>Pommel horse</td>
<td></td>
</tr>
<tr>
<td>Hammer throw</td>
<td>Pull ups</td>
<td></td>
</tr>
<tr>
<td>Handstand pushups</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A.2.1: The 101 different actions of the UCF-101 human action recognition dataset
### A.3 COCO

<table>
<thead>
<tr>
<th>Class</th>
<th>Class</th>
<th>Class</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>Elephant</td>
<td>Wine Glass</td>
<td>Dining Table</td>
</tr>
<tr>
<td>Bicycle</td>
<td>Bear</td>
<td>Cup</td>
<td>Toilet</td>
</tr>
<tr>
<td>Car</td>
<td>Zebra</td>
<td>Fork</td>
<td>TV</td>
</tr>
<tr>
<td>Motorcycle</td>
<td>Giraffe</td>
<td>Knife</td>
<td>Laptop</td>
</tr>
<tr>
<td>Airplane</td>
<td>Backpack</td>
<td>Spoon</td>
<td>Mouse</td>
</tr>
<tr>
<td>Bus</td>
<td>Umbrella</td>
<td>Bowl</td>
<td>Remote</td>
</tr>
<tr>
<td>Train</td>
<td>Handbag</td>
<td>Banana</td>
<td>Keyboard</td>
</tr>
<tr>
<td>Truck</td>
<td>Tie</td>
<td>Apple</td>
<td>Cell Phone</td>
</tr>
<tr>
<td>Boat</td>
<td>Suitcase</td>
<td>Sandwich</td>
<td>Microwave</td>
</tr>
<tr>
<td>Traffic Light</td>
<td>Frisbee</td>
<td>Orange</td>
<td>Oven</td>
</tr>
<tr>
<td>Fire Hydrant</td>
<td>Skis</td>
<td>Broccoli</td>
<td>Toaster</td>
</tr>
<tr>
<td>Stop Sign</td>
<td>Snowboard</td>
<td>Carrot</td>
<td>Sink</td>
</tr>
<tr>
<td>Parking Meter</td>
<td>Sports Ball</td>
<td>Hot Dog</td>
<td>Refrigerator</td>
</tr>
<tr>
<td>Bench</td>
<td>Kite</td>
<td>Pizza</td>
<td>Book</td>
</tr>
<tr>
<td>Bird</td>
<td>Baseball Bat</td>
<td>Donut</td>
<td>Clock</td>
</tr>
<tr>
<td>Cat</td>
<td>Baseball Glove</td>
<td>Cake</td>
<td>Vase</td>
</tr>
<tr>
<td>Dog</td>
<td>Skateboard</td>
<td>Chair</td>
<td>Scissors</td>
</tr>
<tr>
<td>Horse</td>
<td>Surfboard</td>
<td>Couch</td>
<td>Teddy Bear</td>
</tr>
<tr>
<td>Sheep</td>
<td>Tennis Racket</td>
<td>Potted Plant</td>
<td>Hair Dryer</td>
</tr>
<tr>
<td>Cow</td>
<td>Bottle</td>
<td>Bed</td>
<td>Toothbrush</td>
</tr>
</tbody>
</table>

Table A.3.1: The classes of COCO object detection dataset.