MEng final project

Imperial College London

Department of Computing

Semi-supervised Learning of Instance-level Recognition from Video

Supervisor:
Prof. Daniel Rueckert
Second Marker:
Dr. Ben Glocker

Author:
Ilija Radosavovic

Date: June 19, 2017

Abstract

Over the recent years, data-driven approaches that take advantage of the availability of large manually-annotated datasets have proven effective at visual perception tasks. We witnessed particularly rapid improvements in the image classification task, where machines now surpass human-level performance. However, the progress in certain instance-level recognition tasks, such as object segmentation and human pose estimation, has not been as rapid. In part, the rate of improvement has been affected by the difficulty of manually annotating very large amounts of data for these tasks.

In this work we explore if semi-supervised learning approaches that utilise unlabelled video data can provide a reasonably effective alternative to the manually annotated data for instance-level recognition. Central to our approaches is the idea that good models should consistently predict the same label for different pose and view variations of the same object instance. We employ a hard example mining heuristic to find video frames in which the model makes mistakes and correct them by combining the information from the remaining video frames. By noting that video can be seen just like a source of transformation, we generalise our approach to unlabelled images and apply it to the human pose estimation task. The resulting technique, which we call keypoint data distillation (KDD), is simple and very effective.

Using a collection of unlabelled video frames, we show that the Mask R-CNN model combined with KDD achieves state-of-the-art results on the COCO Keypoint Challenge, outperforming all other entries by a significant margin.

Acknowledgements

I would like to thank my supervisor Professor Daniel Rueckert for his guidance and support. I would like to thank my collaborators Ross Girshick, Kaiming He and Piotr Dollár for their invaluable advice.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Computers perform exceptionally well at solving tasks which have a clear mathematical specification that can be translated to a sequence of machine instructions. Examples include performing numeric operations on millions of numbers, solving scheduling problems and finding shortest paths between places. However, humans often find such tasks challenging due to the amount of cognitive reasoning required.

Challenges. In contrast, machines programmed by specifying rules struggle at visual perception tasks that humans find easy. For instance, when humans look at the images shown in Fig. 1.1 they can immediately see that the scene contains people and horses. Furthemore, humans require no additional reasoning to localise objects, denote object outlines or identify body parts of the people in the images. However, these are all very challenging tasks for a machine. In particular, computers operate on images encoded using arrays of numbers indicating brightness values at each position. Deriving more abstract meaning from the raw numbers corresponding to an object is challenging due to the almost infinite variability across different object instances and scenes in the natural world. Hence, it is not feasible to come up with a clear mathematical specification for visual perception tasks.

One way to approach this problem is to rely on heuristic specifications. For instance, one of the first computer vision attempts at object recognition proposed modeling objects as collections of rigid pieces connected by 'springs' [25]. Such approaches have had limited success and have not come even close to human-level performance at visual perception tasks. Contrary to this line of thought, humans excel at perception tasks and yet have never been provided with a set of rules of any kind. Instead, humans learn from data, by observing the world and being taught by others.

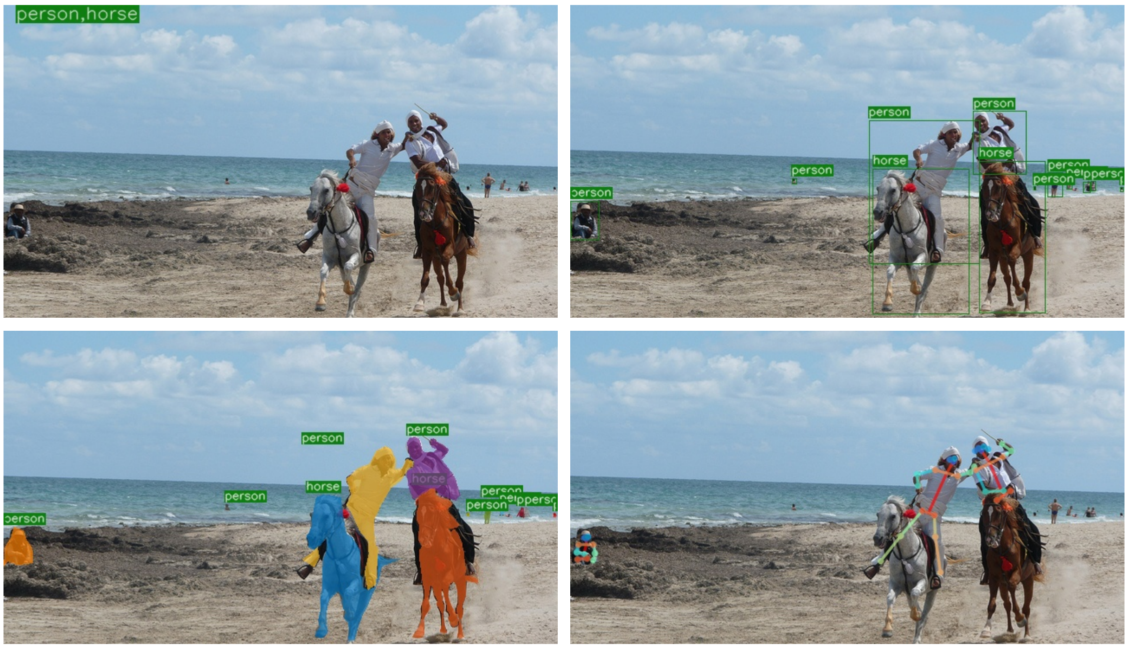

Figure 1.1: *(Top Left)* Image classification. *(Top Right)* Object detection. *(Bottom Left)* Object instance segmentation. *(Bottom Right)* Human pose estimation. Classification is an image-level task while detection, segmentation and pose estimation are instance-level tasks.

An alternative way to tackle the lack of mathematical specification in visual perception tasks is to employ data-driven approaches. In particular, instead of programming machines to perform perception tasks, humans design learning algorithms and let computers discover patterns in data automatically by showing them examples. Although the core algorithmic concepts behind such approaches have been long known [71, 45], we have only recently started to see their effectiveness at perception tasks.

Encouraging Progress. Over the past few years, the rapid increase in performance of learning-based methods at visual perception tasks has been largely driven by more powerful computing platforms and the availability of large manually annotated datasets. We have witnessed particularly dramatic improvements in image classification [43], with machines even surpassing human-level performance [35] on the ImageNet challenge [72].

Further Challenges. However, the progress in instance-level tasks, such as segmentation and pose estimation (Fig. 1.1, bottom), has not been as rapid. Instance-level tasks are more challenging since they require both localising the objects and performing some form of fine-grained classification. Furthemore, manually segmenting objects and locating human keypoints is much more labour-intensive than specifying categories for an image. Hence, creating very large datasets for instance-level tasks is very time-consuming and expensive. Although a relatively large dataset is available [51], it is likely that additional labelled data would speed up the progress in instance-level tasks.

Long-term Goal. An alternative approach to labelling additional data for instance-level tasks is to develop algorithms for learning from unlabelled data. Hence, we could potentially leverage the practically infinite amount of unlabelled data available. There has been some encouraging progress in this direction at certain computer vision tasks. For instance, edge detectors trained using unlabelled data [48] achieve the same level of performance as the ones trained on labelled data. However, in the case of instance-level tasks methods that require no supervision, in the form of manual annotations, are still far from achieving the level of performance of the methods trained with full supervision.

Our Goal. In this work we explore if semi-supervised approaches can provide a reasonably effective alternative to manually annotated examples for instance-level tasks. In particular, we use models trained in a fully-supervised setting to generate annotated examples from unlabelled video data. We then re-train the models using the additional generated data. We choose video as the source of unlabelled data due to its availability and the spatio-temporal information it provides.

1.2 Outline

The rest of this work is organised in five chapters

- In **Chapter 2** we provide the necessary background on classical computer vision techniques and machine learning.
- In **Chapter 3** we start our study by investigating if noisy labels obtained from a model trained in a fully-supervised setting can be used for learning an instance-level task from unlabelled videos. To this aim, we choose the tracking task and develop a framework for learning of tracking.
- In **Chapter 4** we investigate if a model trained with full supervision can bootstrap itself from a collection of unlabelled videos. Namely, we develop a hard example mining heuristic based on dense object tracks and utilise it to generate annotated data from videos. We then use the generated data to re-train the model.
- In **Chapter 5** we generalise the idea from Chapter 4 and simplify our method. In particular, we use a model trained in a fully-supervised setting to generate annotations from still images by combining the model predictions computed at different transformations of an image. We then re-train the model using the generated data.
- In **Chapter 6** we close the exposition and discuss future directions.

1.3 Contributions

We summarise the contributions of our study

- We demonstrate that noisy predictions obtained from a model trained with full supervsion can provide useful signal for learning of tracking from unlabelled videos. As a byproduct of our investigation, we develop a powerful framework for learning of tracking. (Chapter 3)
- We develop an approach for generating hard examples for a model trained in a fully-supervised setting by using spatio-temporal information available in unlabelled videos. (Chapter 4)
- We propose a simple and very effective approach for semi-supervised learning from unlabelled image collections. Using a collection of unlabelled video frames, we show that the Mask R-CNN model trained using our method surpasses the accuracy of all previous approaches for human pose estimation on the COCO Keypoint Challenge. (Chapter 5)

Chapter 2

Background

2.1 Low-level Features

Establishing feature correspondences in image sequences is an essential part of many computer vision algorithms. Examples include panoramic image stitching [9], video stabilisation [42] and 3D reconstruction [3]. This section provides necessary background on feature detection and tracking. For a more thorough introduction we recommend the Chapter 4 of the Computer Vision book by Szeliski [85], on which we based this section.

Point features can be used to establish feature correspondences between different images. However, some features are more distinct and can be identified easier than others. Hence, we want to detect image features that can be matched reliably across different images.

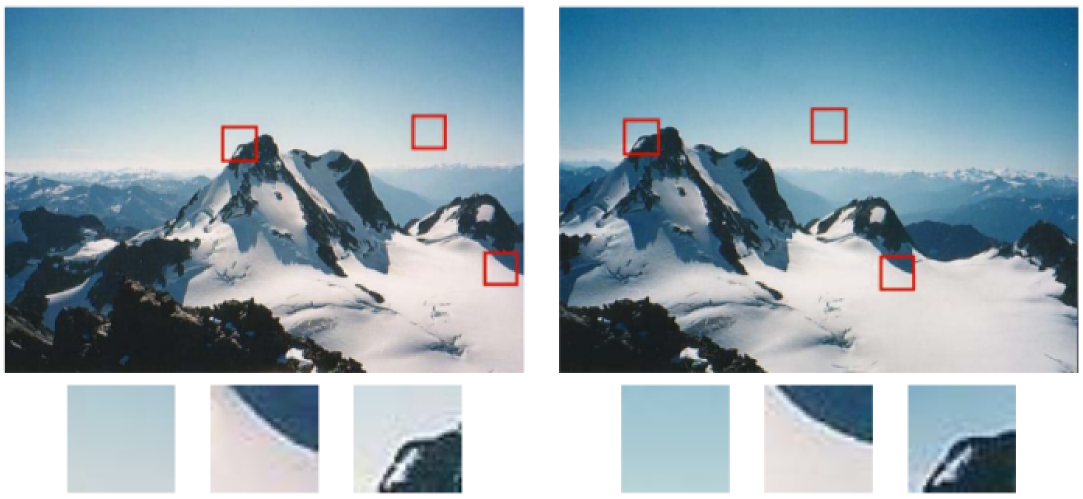

Figure 2.1: Image pair with patches extracted from different types of regions. Source: Figure 4.3 of [85].

2.1.1 Feature Detection

Consider the image pair with three patches shown in Fig. 2.1 to see how well different patches could be matched. Notice that the uniform regions are almost impossible to distinguish and match. The patches containing large gradient changes (edges) are easier to identify. However, there is an ambiguity known as the aperture problem. Namely, we can only align the patches along the direction normal to the edge direction. The third class of patches includes regions with large gradient changes in two directions (corners) which are the easiest to match of the three.

We can formalise our reasoning and quantify the similarity of image patches by defining an appropriate similarity measure. The simlest similarity measure we can use is the sum of squared difference,

$$E_{SSD}(\boldsymbol{u}) = \sum_i (I_0(\boldsymbol{x}_i + \boldsymbol{u}) - I_1(\boldsymbol{x}_i))^2 \,, \tag{2.1}$$

where I_0 and I_1 are the images being compared, $\boldsymbol{u} = [x, y]^\top$ is the displacement vector and i ranges over all the pixels in the patch.

In the feature detection setting we only have access to the image of interest and we cannot know which features might get compared to which patches. Hence, we estimate the stability of image features by comparing the image with itself and computing the sum of squared difference measure in the small neighbourhood $\Delta\boldsymbol{u}$. The resulting function is called the auto-correlation function

$$E_{AC}(\Delta\boldsymbol{u}) = \sum_i (I_0(\boldsymbol{x}_i + \Delta\boldsymbol{u}) - I_0(\boldsymbol{x}_i))^2 \tag{2.2}$$

We can approximate the image intensity in the small neighbourhood $\Delta\boldsymbol{u}$ by using the Taylor series expansion

$$I_0(\boldsymbol{x}_i + \Delta\boldsymbol{u}) \approx I_0(\boldsymbol{x}_i) + \nabla I_0(\boldsymbol{x}_i)^\top \Delta\boldsymbol{u} \tag{2.3}$$

Consequently, we can approximate the auto-correlation function

$$\begin{aligned}
E_{AC}(\Delta\boldsymbol{u}) &= \sum_i (I_0(\boldsymbol{x}_i + \Delta\boldsymbol{u}) - I_0(\boldsymbol{x}_i))^2 && (2.4)\\
&\approx \sum_i \Big(I_0(\boldsymbol{x}_i) + \nabla I_0(\boldsymbol{x}_i)^\top \Delta\boldsymbol{u} - I_0(\boldsymbol{x}_i)\Big)^2 && (2.5)\\
&= \sum_i \Big(\nabla I_0(\boldsymbol{x}_i)^\top \Delta\boldsymbol{u}\Big)^2 && (2.6)\\
&= \Delta\boldsymbol{u}^\top \boldsymbol{A} \Delta\boldsymbol{u} && (2.7)
\end{aligned}$$

where $\boldsymbol{A}$ is the auto-correlation matrix

$$\boldsymbol{A} = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{2.8}$$

A number of classic feature detectors are based on performing eigenanalysis on the auto-crrelation matrix $\boldsymbol{A}$ and finding local maxima in the scalar measures derived from eigenvalues of the auto-correlation matrix [73].

Harris and Stephens [32] propose detecting features by maximising the quantity

$$\det(A) - \alpha\,\mathrm{tr}(A)^2 = \lambda_0\lambda_1 - \alpha(\lambda_0 + \lambda_1)^2 \tag{2.9}$$

with $\alpha = 0.06$. Shi and Tomasi [74] suggest that good features to track can be found by finding maxima in the smaller eigenvalue $\min(\lambda_0, \lambda_1)$. Since aliasing effects at the edges can inflate the smaller eigenvalue, Triggs [88] proposed using the quantity $\lambda_0 - \alpha\lambda_1$ with $\alpha = 0.05$.

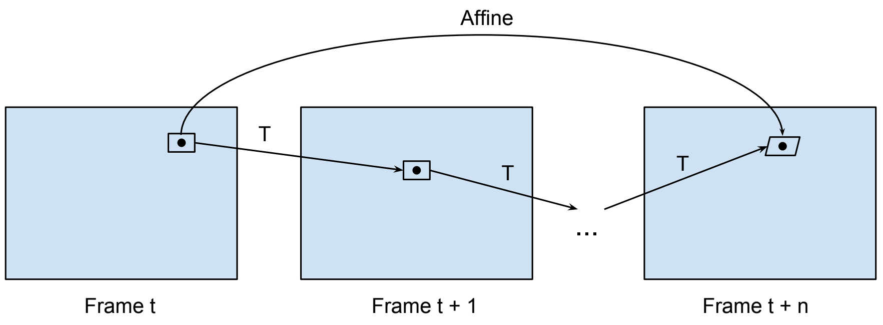

Figure 2.2: Kanade-Lukas-Tomasi (KLT) feature tracking.

2.1.2 Feature Tracking

Given the feature locations in the initial frame of a video, the task of feature tracking is to estimate the feature locations in the remaining frames of the video. The main distinction between feature tracking and general feature matching problem is that the motion between the adjecent frames of the video is expected to be small.

The local motion between features in subsequent frames is mostly translational. Hence, searching for patches with the low sum of squared difference (2.1) metric works well in practice. However, in case of the large motion it is more efficient to use the hierarchical search strategy [54]. Namely, the search is performed at different scales and the matches from lower resolution images are used to initialise the serach for larger resolution images.

After tracking features for longer periods of time, their appearance can change. Hence, comparing potential feature locations with the initially detected patches can result in match failures. One way to overcome this is to re-sample features at newly matched locations. However, this approach can often lead to features drifting from their original locations [74]. This is particularly undesirable when we are interested in tracking features located on a moving object.

A better way to overcome this issue is to estimate an affine transformation between the features in the base frame and the features in the current frame. Shi and Tomasi [74] use the points obtained from the translational model [54] to estimate an affine transformation between the initial feature locatios and the current feature locations. The resulting tracker is often called Kanade-Lukas-Tomasi (KLT) tracker and its steps are shown in Fig. 2.2.

2.2 Motion Estimation

2.2.1 Parametric Motion

For parametric motion, we use a correspondence map $\boldsymbol{M}$, parametrised by $\boldsymbol{\theta}$, to estimate motion that transforms image I_0 to I_1. The map $\boldsymbol{M}$ could be parametrised by any of the standard motion models. Consequently, the number of parameters depends on the chosen motion model. For example, translation would require estimating only two parameters while an affine transformation would require estimating six parameters.

We can generalise the Lukas Kanade algorithm [54] to estimate parametric motion models [85]. In particular, we want to find

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_i \left(I_1(\boldsymbol{M}(\boldsymbol{x}_i;\boldsymbol{\theta})) - I_0(\boldsymbol{x}_i)\right)^2 \tag{2.10}$$

However, it is difficult to optimise the objective function (2.10) directly. Hence, we can apply an iterative scheme

$$\Delta\boldsymbol{\theta}^* = \arg\min_{\Delta\boldsymbol{\theta}} \sum_i \left(I_1(\boldsymbol{M}(\boldsymbol{x}_i;\boldsymbol{\theta}+\Delta\boldsymbol{\theta})) - I_0(\boldsymbol{x}_i)\right)^2 \tag{2.11}$$

and update the parameters at each iteration

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}^* \tag{2.12}$$

By using the Taylor series expansion we can simplify the objective function (2.11) to obtain

$$\Delta\boldsymbol{\theta}^* = \arg\min_{\Delta\boldsymbol{\theta}} \sum_i \left(I_1(\boldsymbol{M}(\boldsymbol{x}_i;\boldsymbol{\theta})) + \nabla I_1 \frac{\partial \boldsymbol{M}}{\partial \boldsymbol{\theta}} \Delta\boldsymbol{\theta} - I_0(\boldsymbol{x}_i)\right)^2 \tag{2.13}$$

2.2.2 Optical Flow

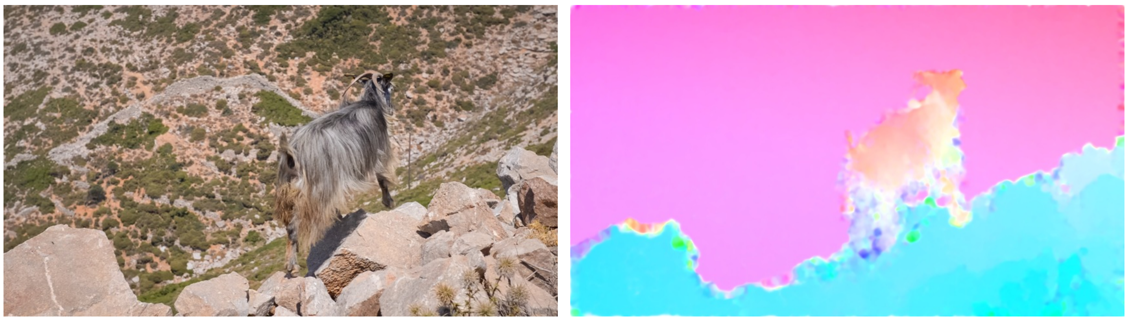

Figure 2.3: Optical Flow computed using Farneback's algorithm [22] and visualised using the colour encoding presented in [2].

Optical flow is the most general way to represent motion and is the most challenging one to estimate [85]. It can be seen as a set of displacement vectors $\{\boldsymbol{d}_i\}$, describing the independent motion at each pixel $\boldsymbol{x}_i$. The optical flow estimation problem can be framed as the minimisation of the colour difference between the corresponding pixels summed over the image

$$E_{SSD-OF}(\{\boldsymbol{d}_i\}) = \sum_i (I_1(\boldsymbol{x}_i + \boldsymbol{d}_j) - I_0(\boldsymbol{x}_i))^2 \tag{2.14}$$

However, the problem is underconstrained since the number of unknowns is twice the number of equations. One way to approach this problem is to minimise the error function locally over the overlapping patches. This approach usually involves performing the Lucas Kanade algorithm [54] to obtain sub-pixel estimates [85].

An alternative way to approach the problem is to constrain the problem and search for the global solution. Horn and Schunck [39] proposed a framework that adds smoothness terms on the $\{\boldsymbol{d}_i\}$ field using regularisation. In their technique, the linearised brightness consistancy constraint is given by

$$E_{HS} = \int (I_x u + I_y v + I_t)^2 \, dx \, dy \tag{2.15}$$

where $(I_x, I_y) = \nabla I_1$ and I_t is the brightness change between images.

2.3 Machine Learning

Machine learning is concerned with developing algorithms that exract patterns from data automatically. We will first provide necessary background on machine learning. For a more thorough introduction to machine learning we recommend textbooks, such as [6] and [57]. For necessary background on mathematics for machine learning, we recommend the lecture notes for CO-496 at Imperial College [16].

Deep learning refers to a type of machine learning and we will provide necessary background on certain aspects of deep learning. For a more comperhensive background on deep learning we recommend the Deep Learning book by Goodfellow, Bengio, and Courville [29].

2.3.1 Supervised Learning

The task of supervised learning is to learn to estimate a function $\boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^t$ from a training set $\mathcal{D}$ consisting of N inputs $\boldsymbol{x}_i \in \mathbb{R}^d$ and their corresponding labells $\boldsymbol{y}_i \in \mathbb{R}^t$, $i = 1, .., N$. We assume thate each $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{D}$ is an independent and identically distributed (i.i.d.) sample from a data generating distribution.

For example, $\boldsymbol{x}_i$ could be a vector of numbers, representing the number of rooms and the area of a flat and $\boldsymbol{y}_i$ could be the estimated price of the flat. The elements of $\boldsymbol{x}_i$ are called features. In general, both $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ could be more complex structured objects. For instance, $\boldsymbol{x}_i$ could be photos of people and $\boldsymbol{y}_i$ could be positions of faces detected in the photos.

We can compactly represent our inputs using a matrix containing a different example in each row

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^\top \\ \boldsymbol{x}_2^\top \\ \vdots \\ \boldsymbol{x}_N^\top \end{bmatrix} \in \mathbb{R}^{N \times D} \tag{2.16}$$

which is commonly called the design matrix. Similarly, if labels are scalars, we can represent them compactly using a vector $\boldsymbol{y} = [y_1, ..., y_N]^\top \in \mathbb{R}^N$.

2.3.2 Linear Regression

We will use the linear regression model to demonstrate the concepts we introduce throughout this section.

In the linear regression setting the task is to predict a continious variable $y \in \mathbb{R}$ from the given input $\boldsymbol{x} \in \mathbb{R}^d$. The model can be written as

$$y = f(\boldsymbol{x}) + \epsilon = \boldsymbol{\theta}^\top \boldsymbol{x} + \epsilon \tag{2.17}$$

where $\boldsymbol{\theta} \in \mathbb{R}^d$ are the parameters and $\epsilon \sim \mathcal{N}\big(0, \sigma^2, \big)$.

Parameters $\boldsymbol{\theta}$ are the values that control the behaviour of the model and that we wish to estimate. A common way to think of $\boldsymbol{\theta}$ is as a set of weights θ_i that determine how each input feature x_i affects the prediction.

Note that the "linear" in the name refers to the fact that the model is linear in the parameters. Linear regression can be made to model non-linear relationships by introducing the basis functions. Namely, we can replace each input $\boldsymbol{x}$ by a non-linear basis function $\boldsymbol{\phi}(\boldsymbol{x})$. For example, the polynomial basis function has the form

$$\boldsymbol{\phi}(x) = [1, x, x^2, x^3, ..., x^k] \tag{2.18}$$

When using the basis functions, the design matrix becomes

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}_0^\top(\boldsymbol{x}_1) & \boldsymbol{\phi}_1^\top(\boldsymbol{x}_1) & .. & \boldsymbol{\phi}_k^\top(\boldsymbol{x}_1) \\ \boldsymbol{\phi}_0^\top(\boldsymbol{x}_2) & \boldsymbol{\phi}_1^\top(\boldsymbol{x}_2) & .. & \boldsymbol{\phi}_k^\top(\boldsymbol{x}_2) \\ \vdots & \vdots & & \vdots \\ \boldsymbol{\phi}_0^\top(\boldsymbol{x}_N) & \boldsymbol{\phi}_1^\top(\boldsymbol{x}_N) & .. & \boldsymbol{\phi}_k^\top(\boldsymbol{x}_N) \end{bmatrix} \tag{2.19}$$

and the model can be written as

$$y = \boldsymbol{\theta}^\top \boldsymbol{\phi}(\boldsymbol{x}) + \epsilon \tag{2.20}$$

2.3.3 Maximum Likelihood Estimation

A common way to estimate the parameters of a model is to perform the maximum likelihood (ML) estimation

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) \tag{2.21}$$

Maximising the log-likelihood is equivalent to maximising the likelihood, since logarithm is a monotonically increasing function. We prefer working with the log-likelihood as it is more numerically stable. The log-likelihood formulation can be written as

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) \tag{2.22}$$

Many software optimisation packages provide procedures to find minima of functions, rather than maxima. Hence, it is often more convenient to turn the log-likelihood maximisation problem into the equivalent problem of minimising the negative log-likelihood

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} -\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) \tag{2.23}$$

We assumed that the training samples are independent and identically distributed (i.i.d.). Thus, we can write our optimisation problem as

$$\begin{aligned} \boldsymbol{\theta}^* &= \arg\min_{\boldsymbol{\theta}} -\log \prod_i^N p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta}) && (2.24) \\ &= \arg\min_{\boldsymbol{\theta}} \sum_i^N -\log p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta}) && (2.25) \end{aligned}$$

To make this more concrete, we perform the maximum likelihood estimation for the linear regression model defined in (2.17). Note that since we assumed Gaussian noise, the likelihood is also Gaussian. The negative log-likelihood (NLL) can be written as

$$\begin{aligned} \mathrm{NLL}(\boldsymbol{\theta}) &= \sum_i^N -\log\left[\left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{N}} \exp\left(-\frac{1}{2\sigma^2}(y_i - \boldsymbol{\theta}^\top \boldsymbol{x}_i)^2\right)\right] && (2.26) \\ &= \frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}) && (2.27) \end{aligned}$$

where we ignored all the terms independent of $\boldsymbol{\theta}$. From (2.27) we see that the NLL of the linear regression is a qudratic in $\boldsymbol{\theta}$. Hence, it has a unique global minimum $\boldsymbol{\theta}^*$

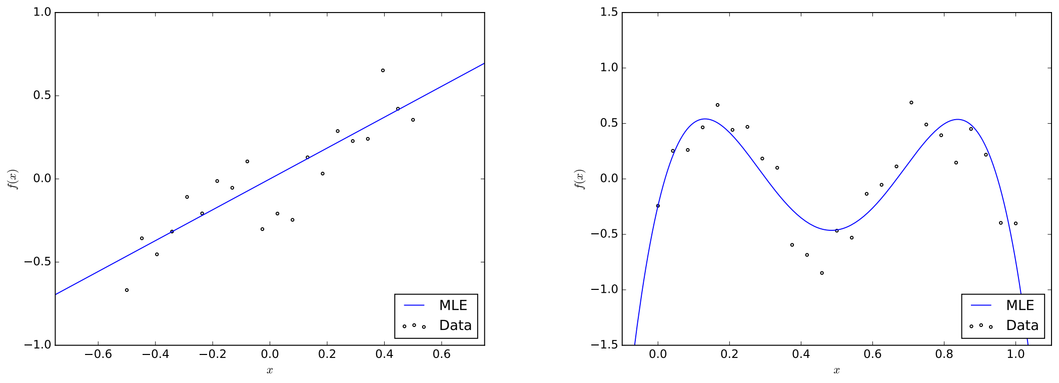

Figure 2.4: Linear regression applied to 1D data. *(Left)* Line determined using maximum likelihood estimation. *(Right)* Polynomial of degree 4 determined using maximum likelihood estimation.

that can be obtained by computing the gradient of NLL and equating it to zero. The gradient of NLL is given by

$$\frac{\partial \mathrm{NLL}}{\partial \boldsymbol{\theta}} = \frac{1}{\sigma^2}(\boldsymbol{\theta}^\top \boldsymbol{X}^\top \boldsymbol{X} - \boldsymbol{y}^\top \boldsymbol{X}) \tag{2.28}$$

Equating to zero we obtain the maximum likelihood estimate for the linear regression model defined in (2.17)

$$\boldsymbol{\theta}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \tag{2.29}$$

By observing that the design matrix is independent of the parameters, we can generalise the computed maximum likelihood estimate (2.29) to the linear regression model with basis functions defined in (2.20) to obtain

$$\boldsymbol{\theta}^* = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\top \boldsymbol{y} \tag{2.30}$$

Chapter 3

Learning to Track

3.1 Introduction

In this chapter we investigate if noisy predictions from a model trained in a fully-supervised setting can provide a sufficient amount of useful signal for learning a non-trivial visual task. To this aim, we choose the tracking task. Given the position of an object in the initial frame of a video, the task of tracking requires predicting the position of the object in the remaining frames of the video.

We approach the tracking task in a data-driven manner. In particular, we use an object segmentation method trained in a fully-supervised setting to generate tracking training examples from unlabelled videos. Next, we propose an approach for training a convolutional network to predict how objects move across the video frames. Finally, we show that our model, trained using noisy segments, outperforms the geometric baselines.

3.2 Related Work

The problem we explore in this chapter has not been addressed much in the literature. Nevertheless, we discuss related elements from the recent work in a number of areas.

Feature Learning from Tasks. Over the recent years a number of approaches for learning features from 'pretext' tasks have been proposed. Examples include, learning to order shuffled video frames temporaly [56], predicting the relative location of cropped image patches [59] and image inpainting [64]. In contrast to our setting, noise-free labells for the aforementioned 'pretext' tasks are easily obtainable using simple algorithmic proceedures. More related to ours is the work of Pathak et al. [63], where the authors show that good features can be learnt from noisy labells obtained by unsupervised motion segmentation. Unlike ours, their approach is unsupervised and the emphasis is on feature learning. Nonetheless, their results support our findings.

Video Classification. Since both video classification and tracking take video data as inputs, we build upon the recent advances in convolutional network architectures for video classification. For instance, Tran et al. [87] propose using 3D convolutions to learn spatio-temporal features and Karpathy et al. [41] present a number of convolutional networks that learn from stacked video frames. Simonyan and Zisserman [78] observe that operating on raw video frames is a difficult task since it requires the network to learn to implicitly represent the spatio-temporal motion-dependent features in the first layers. Consequently, they propose a convolutional network that operates on dense optical flow fields. We experiment with both the stacked raw frame inputs and the optical flow representations proposed in [78], but find that an alternative optical flow representation works best in our setting.

Pixel Map Prediction. Instead of predicting a single score like in classification or a tuple of numbers like in bounding box regression, our setting requires predicting a segmentation mask. Hence, we draw inspiration from the the work that uses convolutional networks to predict pixel maps for various tasks [20, 13, 18]. In order to reduce the computation and help achieve translational invariance, a common approach is to use pooling layers. However, the pooling layers reduce the spatial resolution which, in case of pixel map prediction tasks, needs to be recovered. To this aim, many authors employ the deconvolution layers [96, 52, 69], which often results in fairly complex architectures. Since our network takes relatively small patches as inputs and maintaining spatial information is important for the tracking task, we employ no pooling operations. Similarly, Eigen et al. [19] and Shi et al. [75] apply convolutional networks without pooling to image denoising and super-resolution tasks, respectively. Unlike the two aforementioned approaches, we use ReLU [58] instead of *tanh* nonlinearities for the hidden layers. Using ReLUs avoids saturation [43] and hence allows us to train deeper networks that yield superior performance at our tracking task.

Tracking. Visual tracking has a large body of work; Smeulders et al. [81] povide a comperhensive overview of the area. For a number of years, the most successfull tackers have been based on discriminative approaches that update the model of the tracked object in an online fashion [40, 31, 37]. Following the success of deep learning at various computer vision tasks, trackers based on deep networks are starting to emerge [91, 97, 5]. Unlike most of the tracking approaches, our method tracks segments instead of bounding boxes. Segmentation masks allow for capturing complex deformations, which makes this setting more challenging. We emphasize, however, that our goal in this chapter is not to develop the state-of-the-art tracker but to investigate if our semi-supervised scheme can yield reasonable performance at the tracking task.

3.3 Segmenting Objects

Given a sequence of frames $I_t, I_{t+1}, ..., I_{t+n}$ and a binary mask M_t denoting the position of an object at time t, the task of tracking requires predicting the object position at time $t + n$ represented by a binary mask M_{t+n}. Hence, in order to construct a dataset suitable for learning the tracking task, we need to be able to segment objects from video frames. To this aim, we utilise the DeepMask [66] and SharpMask [67] networks trained for object segmentation in a fully-supervised setting.

DeepMask. Given an image, DeepMask generates a set of segmentation masks with the corresponding set of object score likelihoods. DeepMask employs an approach based on a convolutional network that operates on raw image data.

For a given input patch, DeepMask predicts the object segmentation mask and the score denoting how likely the patch is to contain an object. Both the mask and the score prediction are performed using a single convolutional network. In particular, the network consists of a shared feature extraction trunk which branches into two pathways that handle mask and score prediction. The proposed architecture choice increases the inference speed and serves as a form of regularisation.

The network is trained jointly for the two tasks. The training samples are patch triplets consisting of an image patch, a binary mask corresponding to the patch and a label specifying whether the patch contains an object. In order to achieve generalisation to out-of-training-set objects, only the triplets containing objects are used for training. The generalisation ability of DeepMask is particularly important for our use case, since unlabelled videos are likely to contain object classes not present in the training set.

During full scene inference, the network is applied densely across a number of locations and scales. This ensures that each object is seen in the 'canonical' training view in at least one of the patches. The procedure is performed in a fully convolutional fashion, which makes it fast and computationally efficient.

SharpMask. Object segmentation task requires combining low-level spatial information with the high-level object information. This represents a conflicting goal for the feedforward convolutional networks, since the early layers capture local spatial information while the later layers capture object-level information. To tackle this issue, SharpMask proposes a refinement module that augments the feedforward convolutional networks and results in a bottom-up/top-down approach. In particular, a coarse mask is first predicted in the forward pass and then refined in a backward pass. SharpMask builds on top of the DeepMask architecture and achieves significant accuracy and speed improvements over the baseline DeepMask model. SharpMask's training and inference procedures largely follow the ones of the original DeepMask framework.

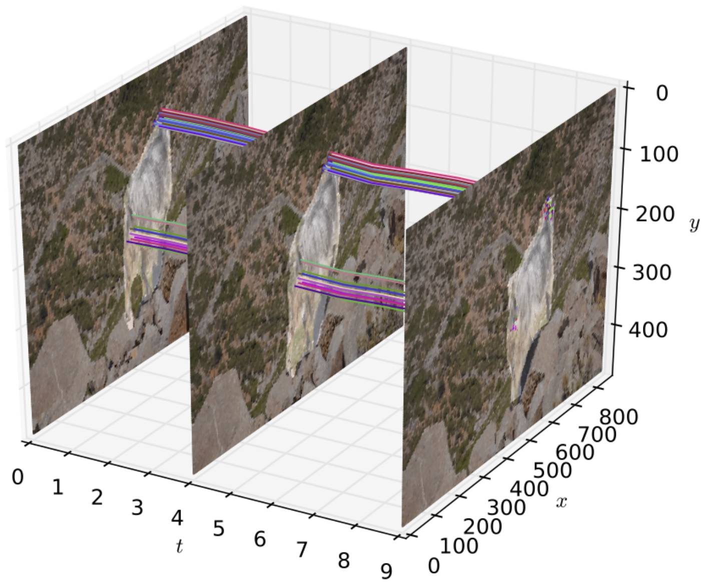

Figure 3.1: Mask matching process. First, features are detected within the mask area in the initial frame. Next, the detected features are tracked across the frame sequence. Finally, the matching heuristic is applied to select the matching mask from the last frame.

3.4 Mask Matching

Given a sequence of frames $I_t, ..., I_{t+n}$, we use the SharpMask [67] network to obtain two sets of masks, $\{M_t\}$ and $\{M_{t+n}\}$, representing objects in frames I_t and I_{t+n}, respectively. However, in order to construct the training examples suitable for learning the tracking task, we need to establish correspondences between pairs of masks. In particular, we require a set of pairs of the form (M_t^i, M_{t+n}^j), such that the i-th mask from the frame I_t and the j-th mask from the frame I_{t+n} correspond to the same object.

To this aim, we devise a mask matching procedure based on low-level feature tracking. Namely, for each mask M_t^i from the frame I_t, we detect a set of feature points P_t^i. Then, we perform feature tracking to obtain a set of feature locations P_{t+n}^i within the frame I_{t+n}. Finally, we employ a mathcing heuristic to determine the best matching mask M_{t+n}^j from the frame I_{t+n} (if any). The process is depicted in Fig. 3.1.

Feature Detection. In order to perform feature tracking successfully, we first need to detect features that can be tracked reliably across the frames. In practice, good features to track usually correspond to 'corners' and are derived by eigenanalysis of the auto-correlation matrix (2.8). We utilise the Shi and Tomasi [74] feature detection algorithm to detect a set of feature points P_t^i within each of the masks M_t^i.

Feature Tracking. After detecting features P_t^i in the initial frame, we want to estimate their corresponding locations P_{t+n}^i in the last frame. Since the motion and the appearance deformation between the consecutive frames is expected to be small, the features detected in the initial frame can be searched for in the subsequent frames. Feature movement between consecutive frames can be modelled using a translational model and solved for my minimising the squared difference (2.1) in the local neighbourhoods. However, to account for occasional larger movement efficiently, it is beneficial to incorporate a pyramidal search strategy. Thus, we use the Lucas-Kanade algorithm [54] to find feature correspondances between the consecutive frames.

The aforementioned procedure works well for short sequences, but becomes less reliable with the increase in the length of the sequence. In particular, in longer sequences the feature appearance undergoes larger changes which cause feature missmatches and drift. To alleviate this issue, we use RANSAC [24] to estimate a homography between the current feature locations and the feature locations in the initial frame and remove the outliers. Note that we are only interested in identifying the outliers in the estimation process and that the computed homography transformation is unused. The resulting feature tracking procedure is inspired by the KLT tracker [85].

Match Selection. Starting from a mask M_t^i we obtain a sparse set of feature locations P_{t+n}^i within the frame I_{t+n} and proceed to determine the best matching mask M_{t+n}^j. We compute the number of features that fall within each of the masks and take the mask M_{t+n}^j that maximises this count as the best candidate match. In order to verify the match, we employ the method proposed in [21] and require that the number of features within the mask is large relative to the total number of features

$$\frac{\text{within}(P_{t+n}^i, M_{t+n}^j)}{|P_{t+n}^i|} > \text{match\_threshold} \tag{3.1}$$

Backward Matching. The mask matching procedure works well when segmentation masks corresponding to the same object are present in both sets of masks $\{M_t\}$ and $\{M_{t+n}\}$. However, this is not always the case since (1) an object may not be visible in both frames due to occlusion or going out of view and (2) SharpMask may fail to detect the same object in both frames. In such scenarios, the matching procedure outlined so far produces false positives.

In order to account for false positives, we repeat the entire matching process in the backward direction and take as final matches only the pairs of masks that agree in both directions. We found this step to be critical for achieving a very robust mask matching procedure. Note that the doubled computation time is not a concern since the mask matching process is performed only once as part of the data generation process.

3.5 Tracking Network

Using SharpMask and the mask matching procedure we can construct a set of examples of the form $(M_t, I_t, ..., I_{t+n}, M_{t+n})$ from a collection of unlabelled videos. Given such examples, we design and train a convolutional network to perform the tracking task.

Frame Stacking. One way to approach the problem would be to design a network that takes the mask M_t and stacked video frames $I_t, ..., I_{t+n}$ as inputs and predicts the mask M_{t+n} as the output. As noted by Simonyan and Zisserman [78], operating on raw video frames is a difficult task since it requires the network to implictly represent spatio-temporal motion-depent features in the first layers.

A common way to represent motion is to use optical flow. In particular, a dense optical flow field can be seen as a set of displacement vectors $\boldsymbol{d}(u, v)$ between the consecutive video frames. Using optical flow field instead of the raw frames makes the learning task easier as the network does not need to learn to represent motion implicitly. However, there is more than one way to encode optical flow information into a format suitable for an input into a convolutional network.

Flow Stacking. We consider the optical flow stacking approach proposed by [78]. In particular, for each pair of the consecutive frames the horizontal and vertical components of the flow field $F^x \in \mathbb{R}^{h \times w}$ and $F^y \in \mathbb{R}^{h \times w}$ are considered as image channels. Then, for a sequence $I_t, ..., I_{t+n}$ the flow fields for consecutive frames are stacked into a volume $F_{t \to t+n} \in \mathbb{R}^{h \times w \times 2(n-1)}$ and used as network input.

We found this approach to give better signal than using raw video frames. However, the resulting performance was still unsatisfactory. Furthermore, since the depth of the input volume depends on the length of the video sequence, the flow stacking approach requires having different networks for different temporal offsets n.

Flow Accumulation. To overcome the limitations of the flow stacking approach, we utilise the flow accumulation procedure used for weak video stabilisattion by [62]. In particular, the horizontal and vertical components of the flow fields for the subsequent frames of the sequence are accumulated into a single flow field $F_{t \to t+n} \in \mathbb{R}^{h \times w \times 2}$.

$$F^x_{t \to t+n} = \text{accumulate}(F^x_t, ..., F^x_{t+n-1}) \tag{3.2}$$

$$F^y_{t \to t+n} = \text{accumulate}(F^y_t, ..., F^y_{t+n-1}) \tag{3.3}$$

Two main benefits of this approach are: (1) it makes the learning task easier as the network does not need to learn to combine flow fields implicitly and (2) a single network operating on fixed size inputs is used to handle all temporal offsets. We found this approach to yield superior performance to the aforementioned alternatives and require a network of smaller capacity that takes less time to train and is faster to run.

Confidence Maps. Dense optical flow estimation is usually framed as an energy minimisation problem and solved for a displacement field by incorporating assumptions such as smoothness or brightness consistency. Those approaches are not perfect and produce noisy estimates. However, the estimates are often more reliable in textured regions that contain good low-level features.

In order to help the network, we experimented with using a confidence map that would give higher weight to flow in the areas of good features. In particular, we tried using the harris responses [32] and minimal eigen values of the auto-correlation matrix [74] as additional input channels. However, we found neither of those made a significant difference, which may suggest that the network learns to ignore noise in the flow field. Consequently, we omitted the confidence map input from our network architecture.

Patch Prediction. Inspired by DeepMask [66], we adapt the patch-based prediction and training approach. In particular, for a binary mask and the flow field corresponding to an object-centered patch in the initial frame, our network predicts a binary mask for the object patch in the last frame.

Each training sample is a tuple containing (1) a binary mask corresponding to the input patch m_{source} (2) the horizontal component of the flow field f_x, (3) the vertical component of the flow field f_y and (4) the binary mask m_{target} corresponding to the output patch. We use lowercase symbols to emphasize that tuple elements correspond to patches rather than entire frames. In order for a patch to be valid we require that

(i) the patch contains an object in the center
(ii) the object is fully within the patch
(iii) the moved object is fully within the patch

All the patches that do not meet the aforementioned constraints are discarded from training. The procedure used to construct the tuples from full masks and flow fields is given in the implementation details section.

conv1	conv2	conv3	conv4	conv5	conv6	conv7	conv8
3x32	32x32	32x32	32x32	32x32	32x32	32x32	32x1
3x3	3x3	3x3	3x3	3x3	3x3	3x3	3x3
ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	Sigmoid

Table 3.1: Network configuration of the SegTrack model. Each convolutional layer uses a stride of one and performs zero padding, which is not shown for brevity.

Network Architecture. To construct network inputs we resize the mask m_{source}, the horizontal flow field f_x and the vertical flow field f_y to a fixed spatial resolution $h^p \times w^p$ and stack them together to obtain an input volume of size $3 \times h^p \times w^p$. Given the input volume, our network predicts a mask of size $h^p \times w^p$ with continuous values between zero and one. To binarize the continuous mask output, we threshold the values using a global threshold. The configuration of our model, which we call *SegTrack*, is outlined in Table 3.1. We next describe our architectural choices.

The spatial information contained in the convolutional feature maps is important for the tracking task. Hence, we chose not to include any pooling layers and pad all the convolutional layers to maintain the same spatial resolution throughout the network. Note that this approach would be very computationally expensive in case of full images, but is fairly cheap in our setting since the network operates on small fixed-size patches.

Following the VGG networks [79] philosophy, we experimented with not overly deep networks that use only 3×3 convolutional filters. In terms of the effective receptive field size, a stack of filters of smaller size is equivalent to a single filter of larger size. However, using a stack of smaller filters has two benefits: (1) it incorporates non-linearities in between which make the decision functions more descriminative and (2) it reduces the number of parameters which can be seen as a form of regularisation. Consistent with evidence presented for other tasks [84], we found that the network depth is of crucial importance for our tracking task.

All hidden layers of our network utilise ReLU non-linearities. In contrast to *tanh* non-linearities, using ReLUs avoids saturation and hence allows us to train deeper networks that yield superior performance in our setting. Since we are interested in predicting binary masks, we chose the *sigmoid* non-linearity for the last network layer.

We experimented with using a larger number of filters in the hidden layers and varying the number of filters between the layers. However, we found this to have little impact on the accuracy while increasing the training and inference times significantly. Hence, we chose to limit the number of filters in all the hidden layers to a relatively small number.

During training, we minimise the pixel-wise binary cross entropy loss between the target mask and the predicted mask.

Full Scene Inference. During full scene inference, for each object mask we construct a patch, satisfying the first two assumptions made during training, and run it through the network. Each prediction is then 'inpainted' in place of the corresponding patch, to obtain the full scene prediciton. A limitation of this approach is that the patch prediction is performed sequentially, which causes the inference time to grow linearly with the number of objects in the scene. However, this is not a concern in practice since the number of tracked objects is small in most cases.

A pragmatic way to improve the full scene inference speed would be to take advantage of the GPU capabilities and perform prediction in batches. Alternatively, an algorithmic way to improve the full scene inference time would be to run the network in a fully-convolutional fashion, like is done in DeepMask. In contrast to DeepMask, at full-scene inference time the input to the tracking framework is a set of dense object masks that could be overlapping, which makes running the network fully-convolutionally non-trivial. Nevertheless, it could be done by incorporating the ideas presented in [49]. However, this is not the focus of our tracking task investagtion presented in this chapter.

3.6 Implementation Details

Given a collection of unlabelled videos, we first perform the data generation procedure to construct a set of training samples for the tracking task. The data generation involves segmenting objects from video frames, computing optical flow, matching masks between frames and constructing the training tuples of the appropriate format. The obtained dataset is stored to disk and used for training and evaluation. We next describe the implementation of the required steps in more detail.

Mask Matching. To segment objects in video frames we use the pre-trained SharpMask model provided by the authors.[1] For each frame we select top 10 masks according to the objectness scores. To binarize the continious SharpMask outputs we apply a threshold of 0.2. We experimented with applying non-maximum suppression (NMS) to the mask outputs but found that it made no impact on the overall performance. Likely since the mask matching process serves as a good way to filter out bad masks. In our feature tracking algorithm, we use the implementations of feature detection [74] and pyramidal Lucas-Kanade [54] from the OpenCV toolbox [8]. We use a thresold of 0.5 for the match selection heuristic.

[1] `https://github.com/facebookresearch/deepmask`

Optical Flow. We compute optical flow using the popular method of [22], available from the OpenCV toolbox. Its fast computation time allows us to perform OpticalFlow estimation on-the-fly during the data generation process. To accumulate optical flow across multiple frames, the flow at integer locations at each stage is obtained using the nearest neighbor interpolation. Note that there is little storage overhead since we only store flow patches of small resolution.

Sample Creation. Given a binary mask M_t, the flow field $F_{t \to t+n}$ and the binary mask M_{t+n}, a training tuple is constructed as follows. First, we compute the bounding box for the object in mask M_t of size $h_{box} \times w_{box}$. We next construct a square patch of size $p_{size} = 1.4 * \max(h_{box}, w_{box})$ enclosing the centered object. Provided that the patch satisfies the necessary constraints for a training sample, the patch areas from the binary masks and the flow field are cropped and resized to 64×64 resolution. Note that special care needs to be taken when resizing the flow fields since the intensities need to be scaled accordingly. In particular, we scale the flow patch intensities by $\frac{64}{p_{size}}$.

Training. We adapt online stochastic gradient descent for training. This scheme is optimal for stochastic optimisation with good starting points [89] and we found it to lead to better convergence than gradient descent with mini batches in our setting. However, the disadvantage of this choice is that we do not fully exploit the parallelism available in modern architectures [46]. We use weight decay of 0.0005 and momentum of 0.9. The starting learning rate is set to 0.001 and adjusted according to a fixed learning rate schedule. In particular, after the first 50 and 120 epochs we reduce the learning rate to 0.0005 and 0.0001, respectively. We train for 200 epochs and use early stopping to help avoid overfitting. Our models take around 7 hours to train on a single Nvidia TitanX GPU. All training experiments were conducted using Torch7 [14].

Inference. At test time, we construct the patches using the aforemention procedure and run each patch through the network. To binarize the continious mask otuputs we use a threshold of 0.5.

3.7 Experiments

Dataset. We perform our experiments using a collection of videos obtained by combining the BVSD [82], FBMS [60] and DAVIS [65] video datasets. The resulting video collection consists of 209 videos. Since we are not interested in the original tasks for which those datasets were created, we discard all the labells provided by the datasets. We shuffle the video collection randomly and use 60% of the videos for training, 20% for tunning the hyper parameters and leave out 20% for testing. Note that we perform the split assignement at the video level rather than the sample level in order to ensure that different splits do not share samples constructed from the same video.

Preprocessing. Sample generation process is applied to each of the splits individually to construct the patch samples. We created a separate dataset for each of the temporal offsets used in our experiments. The sizes of the splits vary slightly depending on the temporal offset. Typically, the number of samples in the training splits is around 22k while the validation and test splits contain around 7k samples each.

Metrics. We measure the similarity between the predicted and 'ground truth' masks by utilising the commonly used Intersection over Union (IoU) metric. IoU is computed by taking the interesection of the predicted mask and the 'ground truth' mask and dividing it by their union. When the IoU score between the predicted and the 'ground truth' mask is above a thresold we consider the prediction to be correct. For all of our experiments, we report the mean and median IoU scores as well as the accuracy at the IoU thresholds of 0.5 and 0.7.

Baselines. To assess the tracking performance of our approach, we compare our learnt model to four geometric baselines

(i) Identity: copies the mask
(ii) T mean: translates the mask by the mean flow vector from the mask area
(iii) T median: translates the mask by the median flow vector from the mask area
(iv) Warp: warps the mask using the flow field

Noisy Ground Truth. Due to the lack of ground truth labels we use the noisy segmentations as ground truth during evaluation. Consequently, achieving the perfect score is not feasible because of the noise present in the 'ground truth' data. As will be seen later, in some cases our model predicts masks considerably better than the 'ground truth'. However, those predictions may get penalised in the quantitative evaluation. Neverthless, the setup serves as a good indicator of the tracking performance.

	IoU mean	IoU median	acc@0.5	acc@0.7
Identity	0.816	0.865	94.175	80.455
T mean	0.871	0.899	98.992	92.602
T median	0.871	0.899	99.006	92.767
Warp	0.874	0.904	98.910	92.698
SegTrack	**0.879**	**0.912**	**99.006**	**93.126**

Table 3.2: SegTrack vs. geometric baselines results for the temporal offset of 1.

Next Frame. We first evaluate the performance at predicting the object position in the next frame (Table 3.2). Note that the identity baseline achieves very high scores. Although our model outperforms the baselines, the amount of movement between consecutive frames is small and makes it hard to draw strong conclusions.

	IoU mean	IoU median	acc@0.5	acc@0.7
Offset 3				
Identity	0.698	0.732	83.201	56.280
T mean	0.814	0.851	96.749	80.879
T median	0.816	0.853	97.020	80.937
Warp	0.810	0.847	95.704	80.472
SegTrack	**0.823**	**0.868**	**95.762**	**83.182**
Offset 6				
Identity	0.619	0.625	73.795	38.247
T mean	0.750	0.775	93.358	66.210
T median	0.753	0.777	93.700	65.502
Warp	0.733	0.754	89.886	60.039
SegTrack	**0.765**	**0.813**	**92.072**	**72.102**
Offset 9				
Identity	0.568	0.562	62.028	29.622
T mean	0.698	0.711	88.094	52.485
T median	0.704	0.716	88.624	53.324
Warp	0.672	0.680	81.710	47.294
SegTrack	**0.714**	**0.759**	**86.923**	**62.006**

Table 3.3: SegTrack vs. geometric baselines results for the temporal offsets of 3, 6 and 9.

Larger Offset. We turn to evaluating our model at predicting masks for frames further apart (Table 3.3). Our model outperforms the baselines in all metrics and for all settings. Furthermore, as the temporal offset increases and the task becomes more challenging, the gap between the tracking model and the baselines increases. This clearly shows that the model manages to learn something more than a standard geometric transformation.

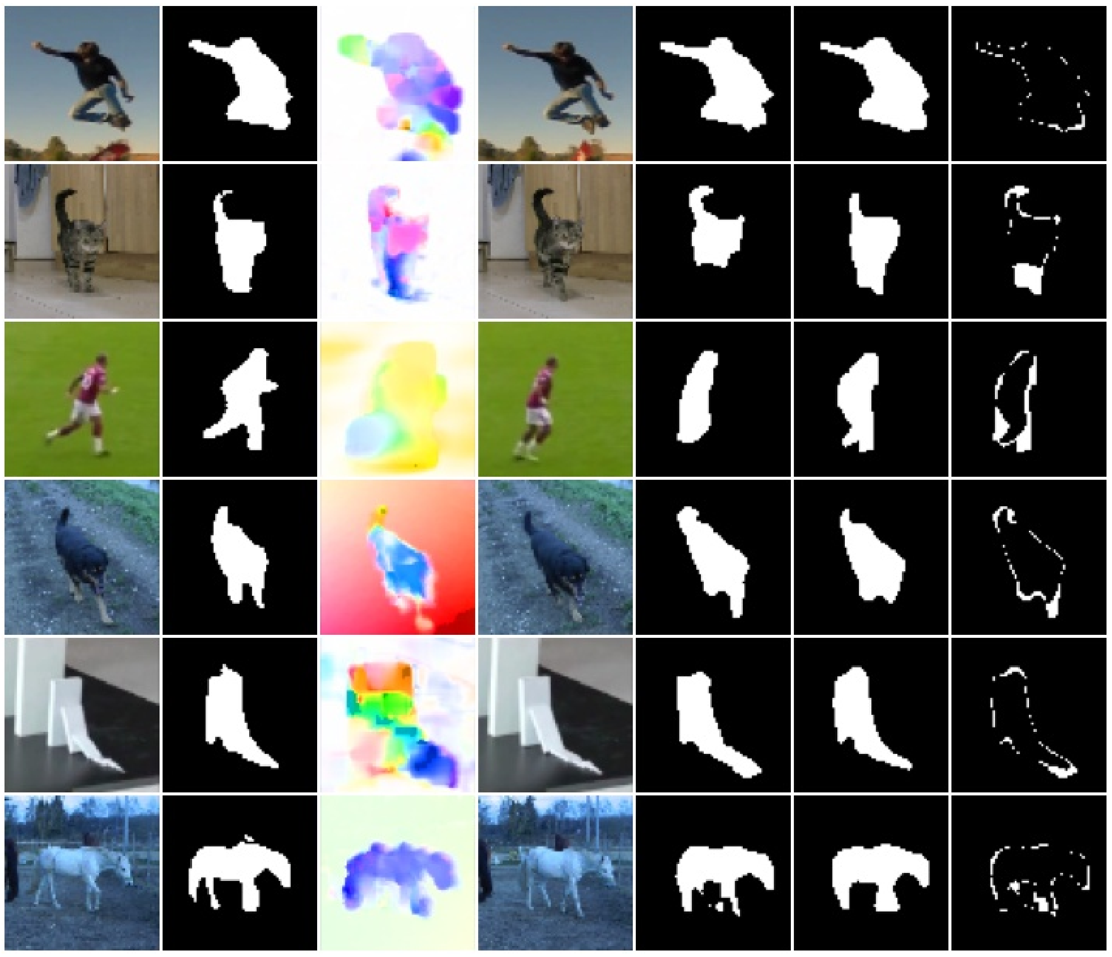

Figure 3.2: SegTrack example predictions on the selected test patches. From left to right: input frame, input mask, input flow, target frame, target mask, predicted mask, absolute error. Flow channels are visualised as one image using the color encoding from [2]. Video frames are shown only for illustrative purposes.

Qualitative Fig. 3.2 shows the SegTrack model outputs on the selected test patches. The absolute difference between the ground truth and predicted masks is shown in the rightmost column. The SegTrack model achieves good results in all cases. It is particularly interesting to note the cases where the predicted masks are of better quality than the noisy ground truth masks. For instance, the ground truth mask for the cat example is lacking a portion of the mask corresponding to the legs of the cat. However, since the source mask is complete the SegTrack model is able to predict a significantly better mask than the noisy ground truth.

3.8 Discussion

We developed a semi-supervised approach for learning of tracking from unlabelled video data. Our experiments demonstrate that the model trained on noisy segments outperforms the geometric baselines. Furthemore, in some cases, the model manages to learn to produce predictions better than the noisy segments it was trained on. Those results are encouraging and show that useful signal can be extracted from noisy predictions.

Although the tracking task was not our focus, we created an interesting framework for learning of tracking, as a byproduct of our investigation. It is not unlikely that the proposed approach could be extended further to deliver superior tracking performance.

Following the observation that the tracking predictions are in some cases better than the predicted object segmentations, it should be possible to use the tracking network to generate additional training data for re-training the object segmentation network. The entire process can then be repeated in an iterative fashion. We leave this direction for future work.

Chapter 4

Video Bootstrapping

4.1 Introduction

In the previous chapter we have shown that noisy predictions from a model trained in a fully-supervised setting can provide sufficient amount of useful signal for learning a non-trivial visual task from unlabelled videos. In this chapter we investigate if a model trained using labelled data can bootstrap itself using a collection of unlabelled videos.

In order to improve the model, we want to find examples on which the model does not perform well. Suboptimal performance is manifested either by the model predicting the correct label with low confidence or by the model predicting an incorrect label. Examples that cause such predictions are hard examples in the context of the given model and would provide useful signal for improving the model. In contrast, examples for which the model predicts the correct labels with high confidence are already mastered by the model and hence would provide no additional information.

The aforementioned observation is not new and has been used in the computer vision community for decades. In particular, it forms the basis of the bootstrapping technique that iteratively updates the model by selecting hard training examples from a pool of labelled examples. However, when trying to apply this idea to unlabelled images we face two challenges: (1) given a prediction for an image we cannot automatically determine whether the prediciton is correct or not and (2) assuming we can somehow establish that a prediction is incorrect we do not know what the correct label is. We tackle those issues by taking advantage of the spatio-temporal information available in videos.

A good model should predict a consistent label for the same object across the video frames with high confidence. However, object pose and scene conditions may vary significantly across the frames and hence make predicting the correct label more difficult in certain frames. Therefore, it is likely that the model would predict correct labels with high confidence in easier frames and be less confident or misspredict in harder frames.

In contrast to the still image case, we can identify the low-confidence predictions and mispredictions in hard frames using the high-confidence predictions for the same object made in other video frames. Furthermore, the correct labels for the incorrect predictions can be approximated by interpolating the labels from the frames with high-confidence predictions. This can be seen as a form of spatio-temporal ensembling, where the predictions for the same object in different frames are combined to obtain a superior prediction.

We use an object instance segmentation model trained in a fully-supervised setting to obtain a set of predictions for each of the video frames. Next, we develop a method for linking the individual frame predictions into a set of dense object tracks. The linking method is then augmented with a hard example mining heuristic and used to extract hard examples from unlabelled videos. Finally, we re-train the model using the generated examples and present the experimental results.

4.2 Related Work

Bootstrapping. Origins of bootstrapping can be traced back to the early work on face detection [83, 70], where bootstrapping was applied in order to cope with the imbalance between the number of positive (face) and negative (background) examples present in the datasets. Instead of starting with all of the negative examples from the dataset, the idea was to gradually expand the training set with the negatives for which the model predicts a face (hard negatives). This led to an algorithm that alternates between two steps: (1) using a fixed model to mine hard negatives to add to the active training set and (2) training the model on the fixed active training set.

Thereafter bootstrapping has been generalised to object detection and applied to a variety of models. Examples include training SVMs [15] and boosted decision trees [17] for pedestrian detection. Specialisations of bootsrapping for certain classes of models have been developed as well. For instance, Felzenszwalb et al. [23] proposed a version of bootstrapping for SVMs and proved that it converges to the global optimal solution.

Simlilar to the traditional bootstrapping, our method relies on mining hard examples and using them to update the model. Unlike the traditional bootstrapping, we mine hard examples from unlabelled data and use generated labels for training.

Bootstrapping in Deep Learning. Perhaps surprising at first, bootstrapping in its traditional form has not seen a widespread use in the modern object detection algorithms that employ deep networks. The main reason for this is likely that deep networks are trained using descent algorithms on datasets with millions of examples, which results in training times that typically range from days to weeks. Hence, performing the alternating bootstrapping steps would slow down the development process significantly.

Our method suffers from the same limitation. However, unlike the traditional bootstrapping, it provides additional labelled data and is still much faster than the alternative of annotating the data manually. Hence, we believe this is a reasonable trade-off.

In order to overcome the long iteration times, a number of approaches for training deep networks using bootstrapping in an online fashion have been proposed. A common theme in these methods is that hard examples are selected based on the current loss of each datapoint. For instance, image patches with high loss are selected when learning image descriptors [80] and good features [92] using siamese networks. [53] and [76] select hard examples for each mini-batch based on the loss when training convolutional networks for image clasification and object detection, respectively. Online bootstrapping techniques are complementary to our method.

4.3 Mask R-CNN

We apply our video bootsrapping method to the recently proposed Mask R-CNN [33] object instance segmentation network. We chose Mask R-CNN since it is a conceptually simple, fast and very general framework. Hence, it allows us to easily generalise our method to other tasks, such as human pose estimation. However, the downside is that Mask R-CNN is a very strong baseline and hence makes seeing positive signal from our method more challenging. We briefly review Mask R-CNN and its predecessors.

R-CNN. Most of the modern object detectors are based on the two-stage approach embodied by the R-CNN [27] framework. The first stage of the R-CNN pipeline uses a class-agnostic region proposal method [90] to compute a set of candidate object locations. In the second stage, a convolutional network is used to extract a fixed-size feature vector from each of the proposed regions. The feature vectors are then classified using the class-specific SVMs and the refined boxes are regressed using the class-specific bounding-box regressors. R-CNN was the first to show that features obtained using deep networks can outperform hand-engineered features at object detection by a large margin.

There are several limitations of the original R-CNN pipeline. Firstly, training is a three-stage process and involves training a convolutional network, a set of class-specific SVMs and a bounding-box regressor per-class. Secondly, training is very slow and requires a lot of space for storing the intermediate training data to disk. Finally, inference is slow since features need to be extracted for each of the region proposals in the test image.

SPPnet. He et al. [36] propose SPPnet to speed up R-CNN by sharing computation. In particular, SPPnet computes a single shared feature map for the entire image and max-pools a portion of the feature map inside each proposal into a fixed-size output. SPPnet achieves fast inference times but is still limited by the three-stage training process.

Fast R-CNN. In order to overcome the limitations of the original R-CNN and the SPPnet, a single-stage training algorithm was proposed in the Fast R-CNN [26] framework. Given an image and a set of object proposal regions of interest (RoIs), a Fast R-CNN network first computes a single convolutional feature map for the entire image. Then, for each object proposal the RoI-pooling layer extracts a fixed-size feature vector from the shared feature map. Each feature vector is fed to a sequence of fully-connected layers that branch into two sibling output layers for classification and bounding box regression. Hence, the entire procedure is performed using a single convolutional network that can be trained end-to-end with a multi-task loss.

Faster R-CNN. Fast R-CNN framework assumed a set of object proposals are given as input. Most proposals computation methods leverage low-level grouping and silency cues [90, 98] and have become the test-time bottleneck of the Fast R-CNN system. In order to address this, Ren et al. [68] proposed the region proposal network (RPN) for computing the object proposals. Since RPN is a deep network itself, the convolutional layers can be shared between the RPN and Fast R-CNN. Hence, proposal computation comes almost for free. The resulting system formed the Faster R-CNN framework.

Faster R-CNN introduced a four-step training process for learning shared features using alternating optimisation. In the first stage, an RPN network is trained and used to compute a set of region proposals. In the second stage, the computed proposals are used for training a Fast R-CNN network. After the second stage, the two networks are independent and share no features. In the third stage, the layers specific to RPN are fine-tuned starting from the Fast R-CNN network from the second stage and keeping the convolutional layers fixed. Finally, the layers unique to the Fast R-CNN are fine-tuned starting from the third-stage RPN network with the shared layers frozen. Consequently, the networks share the convolutional layers and form a unified framework.

Mask R-CNN. In principle, Mask R-CNN [33] is an intuitive extension of the Faster R-CNN framework to object instance segmentation. Mask R-CNN employs the same two-stage process. The first stage is identical to Faster R-CNN and utilises the RPN. In the second stage, in parallel to predicting the class and the bounding box, a binary mask is predicted for each RoI. Since the spatial layout is important for instance segmentation, RoIPool is replaced with RoIAlign which preserves spatial information.

Mask R-CNN architecture consists of a backbone network used for feature extraction and the task-specific heads that are applied to each RoI separately. Its general design allows for the backbone network to be instantiated with a number of recent architectures [34, 94, 50]. The classification and bounding-box regression heads are straight-forward extensions of the standard Faster R-CNN heads while the mask head is a small fully-convolutional network that predicts a segmentation mask in a pixel-to-pixel manner.

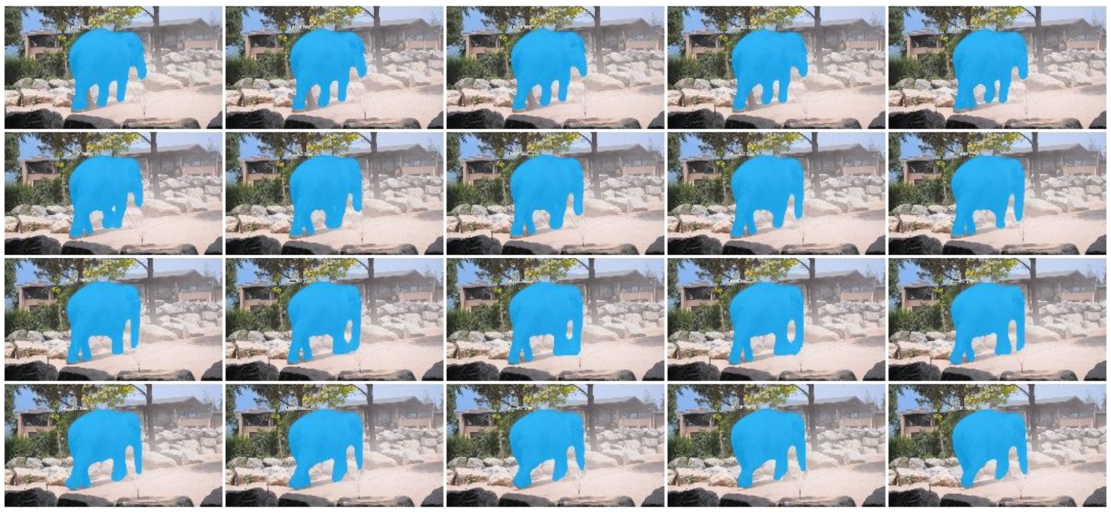

Figure 4.1: Example object track computed using hysteresis linking.

4.4 Object Tracks

Given a sequence of video frames, we use Mask R-CNN to obtain a set of object masks for each of the video frames. In order to be able to apply our video bootstrapping idea, we require a method for computing dense object tracks from individual frame predictions. Specifically, a dense object track consists of a set of masks predicted in contiguous frames that correspond to the same object.

Feature Tracking. One way to approach this problem is to generalise the mask matching method presented in Chapter 3 to dense object tracks. Namely, for each of the masks in the initial frame of the sequence a set of features is detected and tracked across the sequence. Then, for each of the intermediate frames the best matching mask is selected by applying the mask matching heuristic. In essence, for a sequence of length n a dense track is constructed by solving $n - 1$ mask matching problems each of which uses the initial frame as the source frame.

Although this method works well for some scenes, we found that it produces false positives in cluttered scenes and fails to capture more challenging movement. In contrast to the two-frame mask matching setting, we have access to the masks predicted for each of the intermediate frames. The masks output by Mask R-CNN provide much richer information than the low-level features. A limitation of the approach based on feature tracking is that it does not take advantage of the information available in masks.

IoU Linking. Instead of linking masks indirectly via the tracked features, we can directly link the masks according to a similarity measure. Since the amount of motion between the consecutive frames is expected to be small, the intersection over union (IoU) metric is well-suited for this use case.

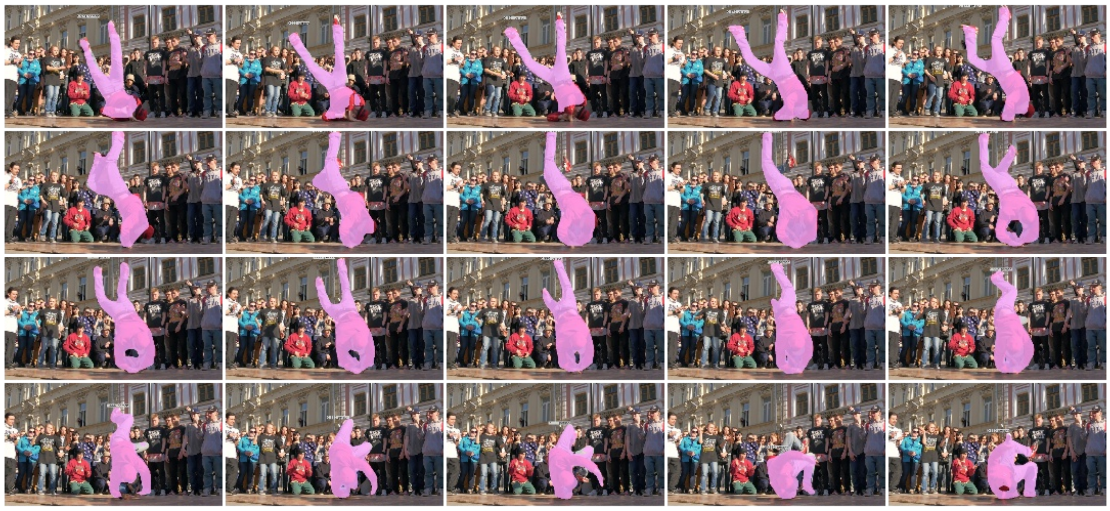

Figure 4.2: Example object track for a breakdancer performing challenging movement in a cluttered scene. The track is computed using hysteresis linking. Note that the tracks computed for the remaining objects in the scene are not shown for clarity.

Using IoU linking, mask tracks can be computed by starting from the initial frame and picking the masks in the subsequent frames that maximise the IoU metric greedily. We found this approach to work considerably better than the feature-based linking. However, it has two limitations: (1) it does not take the predicted scores into account and (2) it needs to be run on a large number of sampled sub-sequences in order to achieve dense scene coverage.

Hysteresis Linking. Inspired by hysteresis thresholding used for edge linking [11], we devise a mask linking algorithm that employs hysteresis thresholds. In particular, we use three thresholds: score-low, score-high and iou-low. The high score threshold is used to initiate the tracks while the low thresholds are used to extend the tracks.

The algorithm proceeds as follows. First, the mask with the highest score of at least score-high is chosen as the seed mask. The mask is then propagated forward and backward as long as it can be linked with another mask whose score is above score-low and whose IoU overlap with the current mask is above iou-low. Once the track cannot be extended further, the masks belonging to the track are pruned from the pool of available masks. The whole process is iterated until no more tracks can be constructed.

We found that the mask linking approach based on hysteresis thresholding works extremely well in practice. An important additional benefit of this method is that it provides a natural way of achieving dense scene coverage. Hence, the algorithm is run once per video and requires no sampling of sub-sequences.

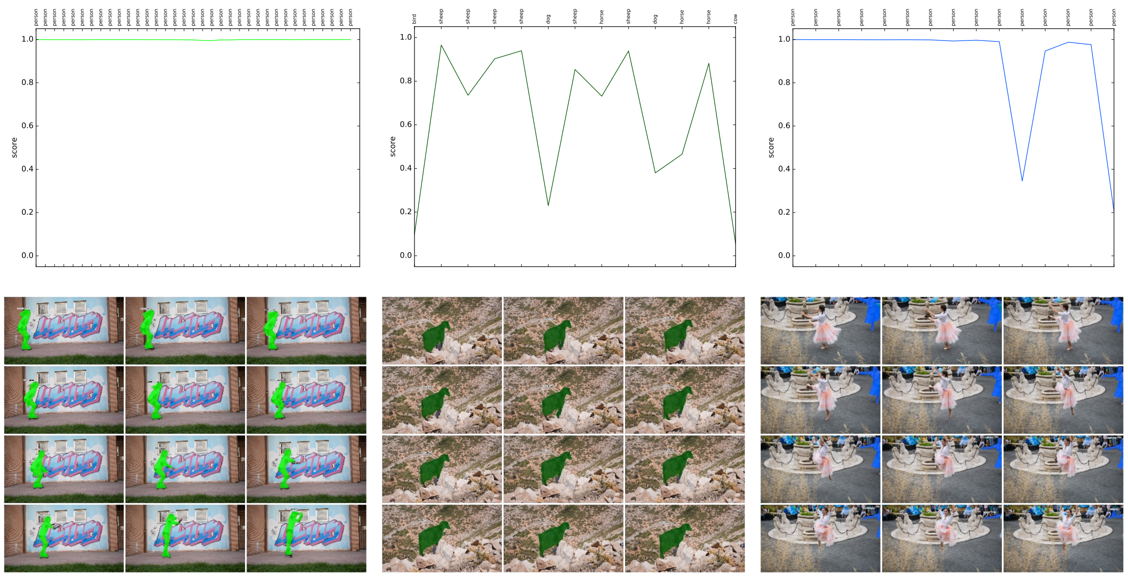

Figure 4.3: *(Top)* Each subfigure shows the classification score over time. The argmax class for each time step is shown on the top x-axis. *(Bottom)* Mask tracks corresponding to the score plots shown in the top row. *(Left)* The model is always confident. *(Middle)* Out-of-trianing-set category. *(Right)* The model is not confident in certain frames.

4.5 Hard Example Mining

We discuss three charactersitic types of object tracks observed while experimenting with mask linking on videos sampled from DAVIS [65] and YFCC100m [86] datasets.

Confident Tracks. The first category consists of mask tracks where the model is confident throughout the track. An example is shown in Fig. 4.3, left. Object poses whose masks form the tracks in this category are already mastered by the model and hence would provide no additional information for improving the model.

Out-of-training-set Tracks. Fig. 4.3, middle, shows an object track corresponding to a goat. Since the goat category is not present in the labelled training set, the model predicts various other categories, such as dog and sheep. Surprisingly, the masks found in the tracks of this type are still of high quality, which suggests that Mask R-CNN masks generalise well. Handling out-of-training-set categories is beyond the scope of this work. Hence, we focus on the categories present in the labelled training set.

Tracks with Hard Examples. In order to improve the model we want to identify hard examples using the object tracks. An example track that allows us to that is shown in Fig. 4.3, right. In particular, such tracks mainly consist of high-confidence predictions for the same category but also include some object views for which the category is predicted with lower confidence or the predicted category differs from the one predicted in the high-confidence frames. We are interested in finding object tracks of this type.

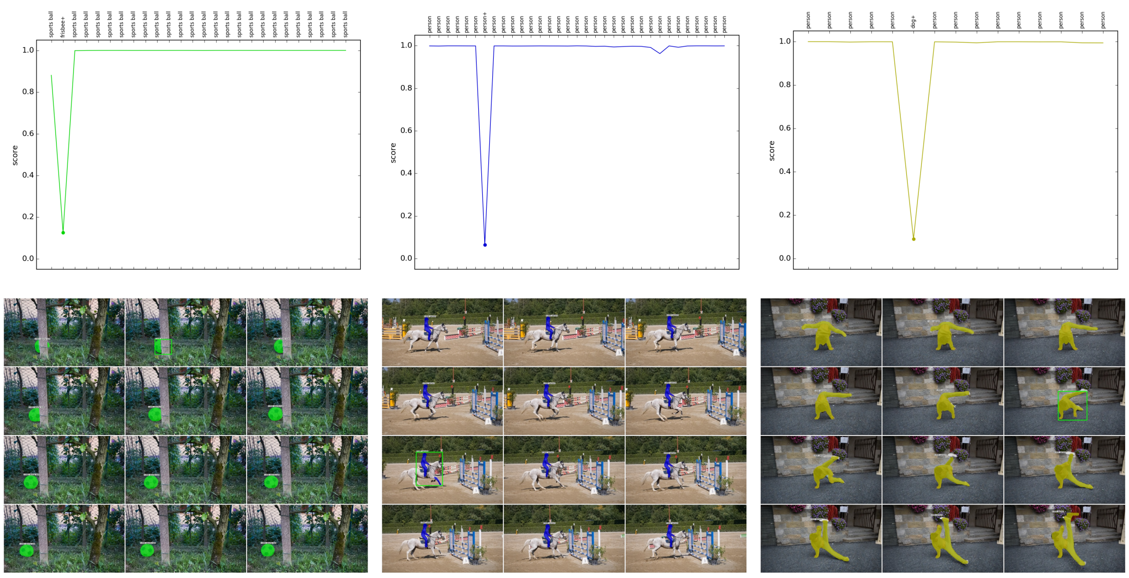

Figure 4.4: Example tracks found using the hard example mining heuristic. *(Top)* Scores predicted for the hard examples are marked with dots in the score plots and the corresponding argmax classes are suffixed with pluses. *(Bottom)* Masks predicted for hard examples are denoted using bounding boxes.

Mining Heuristic. In order to find tracks that contain hard examples, we augment our hysteresis linking method with a mining heuristic. In particular, when propagating the seed mask forward and backward we consider a window of size W around each prediction to determine if the prediction corresponds to a hard example.

Suppose $W = 3$, for simplicity. Then, in order for a mask at time t to be considered as a hard example we require that (1) the mask score is below score-low, (2) its IoU overlaps with the masks from the frames $t-1$ and $t+1$ are at least iou-low and (3) the masks chosen at times $t-1$ and $t+1$ have the scores above score-high and the same argmax class. Note that we do not enforce that the hard example prediction is of a class different from the neighboring predictions, in order to allow for finding low-confidence predictions of the likely correct class. When no mask satisfying the aforementioned conditions is found in a window, the mining heursitic falls back to the standard hysteresis linking. Example tracks obtained using the mining heuristic are shown in Fig. 4.4.

Label Generation. Given the tracks containing hard examples, we want to generate target labels for the hard examples. Each label is a triplet consisting of a binary mask, a bounding box and a class name. For each hard example we first generate the mask by computing the union of the two neighboring masks from the track. The bounding box is then constructed by taking the tight box around the generated mask. Finally, in case the predicted class differs from the class of the neighboring predictions from the track, the class of the neighbors is used as the target class.

4.6 Implementation Details

Given a collection of unlabelled videos, we first run Mask R-CNN inference to obtain a set of predictions for each of the frames. Next, we employ our mining heuristic to find object tracks that contain hard examples. Labels for the hard examples are then generated from the object tracks. Finally, we re-train the model using the generated labels. We next describe the implementation details of the required steps.

Mask Inference. The first step of our hard example mining process involves running Mask R-CNN inference on each of the video frames. We use Mask R-CNN implementation kindly provided by the authors. We consider Mask R-CNN variations with ResNet backbones and train feature sharing versions using the four-step alternating optimisation process, described earlier in the Mask R-CNN section. All of the hyper-parameters are set following the original Mask R-CNN.

Video Indexing. To ease the process of working with arbitrary collections of videos, we designed a COCO-style video annotation format. In particular, we extended the COCO annotation format by adding a video level to the annotation hierarchy. Using the developed annotation format has two additional benefits: (1) it allows us to trivially parallelise Mask R-CNN inference over the video frames and (2) it enables us to efficiently store and re-use the computed annotations.

Motion Blur Filtering. Since a significant number of unlabelled videos in the wild contains motion blur that could introduce bias, we experimented with filtering out the blurry videos. In particular, we implemented a motion blur meter based on the spread of edges [55]. However, we found that Mask R-CNN was very robust to motion blur. Consequently, we decided not to use motion blur filtering as part of our pipeline.

Mask Linking. Mask linking is performed on run-length encoded (RLE) masks. In addition to reducing the space required for storing the masks, the RLE format allows us to perform IoU computations directly on the encoded masks. Hence, the RLE eliminates the need to decode the masks and makes the linking procedure very fast. We use the optimised RLE IoU meter implementation available from the COCO toolbox.[1]

Example Mining. When mining hard examples we consider a temporal window of size three around each of the predictions. We set the hysteresis thresholds iou-low, score-low and score-high to 0.5, 0.5 and 0.8, respectively. To avoid construction of trivial tracks, we impose a minimal track length of three.

Training. When training with generated data, we simply treat the generated labels as ground truth and follow the original Mask R-CNN training process. All training experiments were conducted using the Caffe2 framework.

[1] `https://github.com/pdollar/coco`

4.7 Experiments

Labelled Data. As the source of labelled data, we use the MS COCO object instance segmentation dataset [51]. COCO includes objects from 80 categories and contains 80k training and 40k validation images with over 850k annotated objects. Following the common practice [4], we use the union of 80k training and 35k validation images (trainval35k) for training and report results on the remaining 5k validation images (minival).

Video Data. We use the YFCC100m [86] dataset as the source of unlabelled video data. The YFCC100m dataset contains around 700k videos that have been uploaded to Flickr over the span of 10 years. Hence, the dataset contains videos that differ significantly in both quality and content. In order to experiment with our method, we sampled a subset of 1.5k videos containing around 200k frames in total. Note that although there are more frames in this sample than there are images in COCO trainval35k, the frames come from fewer videos and are thus much more correlated than COCO images.

Preprocessing. Since some of the YFCC100m videos are of very high resolution (4K), the decoded frames take a large amount of space. Furthermore, storing the frames in the very high resolution would be wasteful since the frames would get downscaled to the Mask R-CNN 'canonical' resolution at inference time. Hence, when decoding the videos we downscale the frames such that the longer side of each frame is at most 800 pixels. We perform no motion blur filtering as part of the video data preprocessing.

Metrics. We evaluate our models on COCO minival (5k) using the standard COCO detection evaluation metrics. We report the average precision (AP) for different threshold settings including AP (averaged over ten IoU thresholds), AP_{75} (IoU 0.75) and AP_{50} (IoU 0.5). Additionally, we report AP for small, medium and large objects denoted by AP_S, AP_M and AP_L, respectively. Unless otherwise specified, the AP is averaged over all the categories. Note that the AP is computed using mask IoU in all cases.

Models. For all the experiments we use the Mask R-CNN model with ResNet-50-C4 backbone architecture. In order to simplify the training process and create a more controlled setup, instead of using an RPN that shares features with the Mask R-CNN, we use a separately trained RPN in all of the experiments.

Training Hyperparameters. All of the models used in our experiments are trained using the same hyperparameters. We train for 80k iterations with a learning rate of 0.02 which is divided by 10 at iteration 60k. All other hyper-parameters follow the settings used for the ResNet-50-C4 backbone in the Mask R-CNN paper.

Baselines. We consider two baselines: (1) the model with a backbone pre-trained on ImageNet1k classification set [72] and then fine-tuned on COCO trainval35k and (2) the model from (1) fine-tuned on COCO trainval35k once more.

	init	train	#gen	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AP^p
Baseline	In-1k	coco	n/a	28.7	47.8	30.1	9.2	32.4	47.4	37.1
Vid-25k	In-1k	coco+gen	25k	27.5	46.1	29.0	8.8	30.9	46.3	36.2
Vid-50k	In-1k	coco+gen	50k	26.7	45.2	28.8	8.7	29.9	44.7	35.6
Baseline	coco	coco	n/a	29.8	49.1	31.5	9.7	33.4	50.0	38.1
Vid-25k	coco	coco+gen	25k	29.1	47.8	30.7	9.2	32.3	48.3	37.1
Vid-50k	coco	coco+gen	50k	28.3	47.1	29.7	9.2	31.8	47.0	36.5

Table 4.1: Instance segmentation evaluation results on COCO minival (5k).

Generated Examples. We mine hard examples from 1.5k videos sampled from the YFCC100m dataset. In our experiments we use only the examples found for person category. Focusing on person category has three benefits: (1) it avoids the issue of out-of-training-set categories, (2) since Flickr videos are mainly people-centric ensures that there will be a reasonable number of examples in a relatively small video sample and (3) makes the experiments more controlled and easier to analyse. We consider two samples of generated person annotations, containing 25k and 50k person instances.

Training Procedure. When training with generated data we combine the generated examples with the labelled COCO examples and treat the resulting collection as a single dataset. Combining the labelled and generated data, instead of fine-tuning on generated data only, has two advantages: (a) it provides better gradient estimates and (b) it helps prevent overfitting the noise in the generated data. We experiment with initialising weights from models pre-trained on ImageNet1k classification and COCO.

Results. We compare the models trained with additional generated data to baselines trained using labelled data only in Table 4.1. We notice that for both weight initialisation strategies, the baselines outperform the models trained with generated data. Furthermore, as the amount of generated data increases the model performance decreases.

Limitations. Although the results as presented above show no positive signal, we emphasise two important limitations of our training procedure.

Firstly, both the baselines and the models that use additional generated data were trained for the same number of iterations. This implies that the baselines have on average seen each training example more times than the models trained with additional data. Hence, instead of allowing the models to take advantage of the additional data we effectively substituted some of the manually annotated examples with generated data. Therefore, it is not realistic to expect that the models trained with generated data using our procedure could achieve performance any higher than the baselines. In light of this observation, the results showing the drop of 1-2 AP could be seen as encouraging.

The second limitation of our training procedure concerns handling of negative examples. In particular, images containing objects with generated annotations are likely to contain additional objects that have not been identified as hard by our example mining heuristic. However, our training procedure does not account for unlabelled objects present in the scenes and treats them as background regions. Consequently, during the optimisation process the model gets tasked with conflicting goals which likely has a negative effect on the resulting learnt features.

4.8 Discussion

We developed a semi-supervised approach for bootstrapping models trained in a fully-supervised setting using unlabelled video data. We base our approach on the observation that models trained using the available amounts of labelled data are not fully invariant to a variety of appearance changes objects undergo in large video collections.

Examples generated using our method demonstrate that spatio-temporal information available in video data can be used to identify hard examples in unlabelled videos. However, our quantitative evaluation has not shown a clear positive signal. We hypothesise that surpassing the performance of the baseline models requires addressing the previously outlined limitations of our training procedure for generated data.

Taking everything into account, we have seen some positive signal and believe that the main idea presented in this chapter is promising. In Chapter 5 we generalise our main idea and simplify the methodology. Furthermore, we address the shortcomings of the training procedure applied to generated data in this chapter.

Chapter 5

Data Distillation

5.1 Introduction

In Chapter 4 we developed a semi-supervised approach for bootstrapping models trained in a fully-supervised setting using unlabelled video data. Our idea was based on the hypothesis that a good model should predict the same label for the same object throughout a video. The proposed method was to find cases in which the model makes mistakes and correct them. In order to identify such cases, we relied on heuristic example mining which is less than ideal. In this chapter we take a more general view on the main idea and simplify our method.

One way to think about video is just like a source of transformation. Hence, ensuring that the model predicts the same label for the same object throughout a video amounts to making the model invariant to the video transformation. It follows that our idea can be generalised to still images. In particular, given an unlabelled image, irrespective of what the correct image label is, a good model should predict the same label for different transformations of the image.

Since our models trained using the available amounts of labelled data are likley not fully invariant to such transformations, some transformed views will allow for better predictions than others. Hence, much like in the case of object track from the previous chapter, we can combine the predictions from the transformed image views to assemble a superior prediction for the original image.

We adopt the generalised view and test our hypothesis on unlabelled images as follows. Given an unlabelled image we run inference on a number of different transformed versions of the image and combine the obtained predictions to generate a label for the image. We perform the aforementioned labelling procedure on a collection of unlabelled images and use the generated annotations to re-train the model.

We apply our approach, which we call data distillation, to the human pose estimation task using the Mask R-CNN framework. In order to analyse different aspects of our approach, we perform a sequence of experiments on the COCO keypoints task. Finally, we show that our simple method is very effective and combined with Mask R-CNN leads to state-of-the-art performance on the COCO Keypoint Challenge.

5.2 Related Work

Model Ensemble. A well known way to imrove performance of a machine learning model is to train a number of variations of the model on the same data and average their predictions at test time [30, 44]. Model ensembling is extremely effective and is heavily utilised in commercial systems [38] and academic challenges [72, 51]. However, it has two limitations: (1) training an ensemble of models is very cumbersome and (2) performing inference is computationally expensive.

Knowledge Distillation. Caruana and collaborators [10] show that test time limitations of model ensembling can be overcome by compressing the knowledge of an ensemble of models into a single smaller model. Their approach was extended by Hinton and collaborators [38] who proposed an alternative way of compressing the ensemble knowledge, which they called distillation. Instead of distilling knowledge of many models on the same data, we distill knowledge of the same model under different views of the data.

Data Augmentation. A standard way to improve model generalisation is to employ data augmentation [95, 77], which effectively increases the size of a labelled training set. In particular, both training inputs and labels are transformed using the same label-preserving transformations and the transformed samples are used as additional training examples. In contrast, we apply transformations to unlabelled data and use the combined transformed predictions as training labels.

Multi-transform Testing. A possible way to improve test time performance of a fixed model is to average its predictions for different transformations of the test input. Examples include averaging predictions for several image crops, multiple scales and horizontally flipped images [43, 79, 35]. At a high-level, our label prediction procedure does the same thing. However, instead of using this procedure to enhance the test time performance of our model we utilse it for generating labels for unlabelled data.

5.3 Keypoint R-CNN

We test our data distillation idea by applying it to the recently proposed Mask R-CNN model [33]. Mask R-CNN is a conceptually simple, fast and very general framework. Hence, it makes it easy to experiment with our approach on any of the instance-level tasks. To this aim, we choose the human pose estimation task and refer to the corresponding Mask R-CNN instantiation as Keypoint R-CNN. We briefly review Keypoint R-CNN, focusing on aspects of training important for our approach. An overview of the Mask R-CNN framework and its predecesors can be found in Chapter 4.

Human Pose Estimation. Keypoint R-CNN tackles the human pose estimation task by utilising the standard two-stage approach. In the first stage, a set of object proposal regions of interest (RoIs) are predicted using the region proposal network (RPN). In the second stage, in parallel to predicting the class and the bounding box, a set of keypoint locations is predicted for each RoI.

Keypoint R-CNN models keypoints using one-hot encoded heatmaps. In particular, for each of the K keypoint types of an instance (e.g. nose, right ear, etc.) the training target is a $m \times m$ binary mask with only a single pixel set. Hence, the keypoint types are treated independently and a heatmap is predicted for each type. Note that this is, up to a small modification, identical to how object masks for instance segmentation are predicted using Mask R-CNN.

Hierarchical Sampling. Keypoint R-CNN training procedure largely follows the one of its predecessors [26, 68]. In order to save computation by taking advantange of feature sharing, a hierarchical mini-batch sampling is employed. In particular, each mini-batch of size R is constructed by first sampling N images and then by sampling N/R RoIs from each image. Note that in case of Keypoint R-CNN only the images that contain at least one visible keypoint are used for training.

RoI Sampling. Each image contains a significantly larger number of background RoIs than foreground RoIs. Hence, using a uniform RoI sampling strategy would result in an inefficient learning process. To overcome the imbalance problem, the background RoIs are subsampled at random to achieve a ratio of $1 : 3$ of foreground to background RoIs. Thus, a quarter of each mini-batch constitutes of foreground RoIs.

Foreground vs. Background. RoIs are classified as foreground or background based on their IoU overlap with ground truth boxes. Each RoI with an overlap greater than or equal to 0.5 is considered as foreground. RoIs with an overlap in $[0, 0.5)$ are treated as backgound. Note that using a non-zero lower bound can be thought of as a form of hard negative mining under the assumption that RoIs that have at least some overlap with ground truth boxes are more likely to represent hard examples [26, 36].

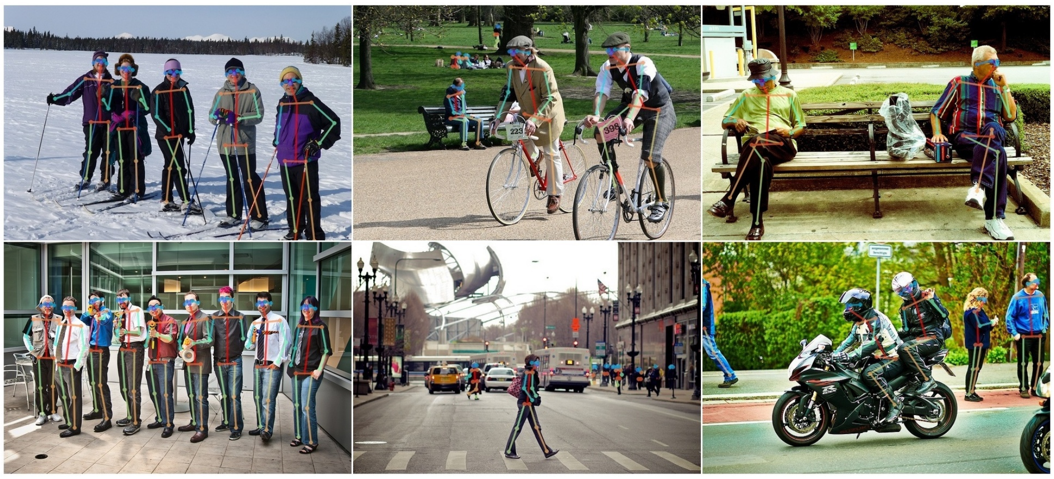
Figure 5.1: Keypoint R-CNN with KDD results on COCO test-dev (Section 5.7.3).

5.4 KDD Inference

In order to test our hypothesis, we apply the data distillation idea to the human pose estimation task. We chose this task for two reasons. Firstly, there is less labelled data available for it than for the other instance-level tasks, such as detection or segmentation. Hence, we would likely require less additional generated data to see positive signal, which would allow us to iterate faster. Secondly, the instance-level predictions for the keypoints task consist of a number of individual keypoint types (e.g. nose, left ear, etc.), which allows for cherry-picking individual keypoints from different predictions.

Prediction Matching. Given an image, the idea is to run inference on the transformed versions of the image to obtain a set of keypoints predictions per transformation. For each person, the predictions from the transformed images should then be combined to obtain a single prediction. However, to be able to combine the predictions at the instance-level, we need to determine which predictions correspond to the same instance.

To this aim, we utilise the region-based nature of the Keypoint R-CNN. In particular, given a set of RoIs for the original image we apply a transformation to the RoIs to obtain a set of corresponding RoIs in the transformed image space. Next, we run the transformed image through the network feature extraction trunk to obtain the transformed feature map.[1] We then use the transformed RoIs to select features from the transformed feature map and run the keypoint network head to obtain the keypoint predictions in the transformed image space. Finally, the keypoint predictions in the transformed image space are projected to the original image space by applying the inverse transform.

It follows that all the transformed predictions obtained using the same RoIs from the original image are in correspondence. In practice, we take the bounding boxes predicted for the original image as the set of RoIs used for computing the transformed predictions.

[1]We sometimes loosely use transformed X to mean X for the transformed image.

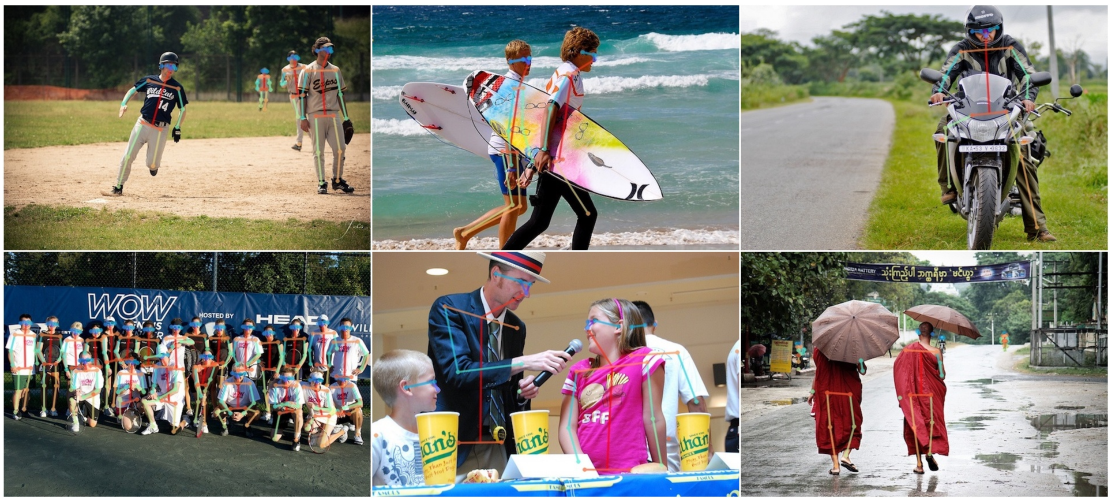

Figure 5.2: More results of Keypoint R-CNN with KDD on COCO test-dev (Section 5.7.3).

Combining Predictions. Given a set of predictions computed under various transformations and corresponding to the same instance, we want to combine them into a single prediction. To this aim, we consider a number of heuristics, such as taking the keypoints with the maximal scores or averaging the keypoint locations. Furthermore, since the predictions can be combined by either combining the predicted heatmaps or the image locations obtained from the heatmaps, we consider heuristics that operate on both of those representations.

Transformations. In order for a transformation to be applicable within our data distillation framework, we require that the transformation (a) is label-preserving and (b) has a corresponding geometric inverse. We employ a range of geometric image transformations. Examples include scale, aspect ratio and horizontal flip. We also consider transformations that operate on image intensities, such as histogram equalisation.

Size-dependent Scaling. Many transformation have the same effect on all objects, irrespective of the object size. For example, performing horizontal flip results in superior predictions for both small and large objects. However, it is important to notice that some transformations have varying effects depending on the object size. For instance, upscaling an image helps with predictions for small objects. In contrast, upscaling has little effect on large objects. In case of downscaling the difference is more extreme. In particular, downscaling helps with predictions for large objects while it hurts the predictions for small objects. In order to account for this behaviour, we selectively apply transformations that alter image scale by object size.

We elaborate on the specific heuristics and transformations in the implementation details section. Furthermore, we evaluate their effectiveness as part of our experiments.

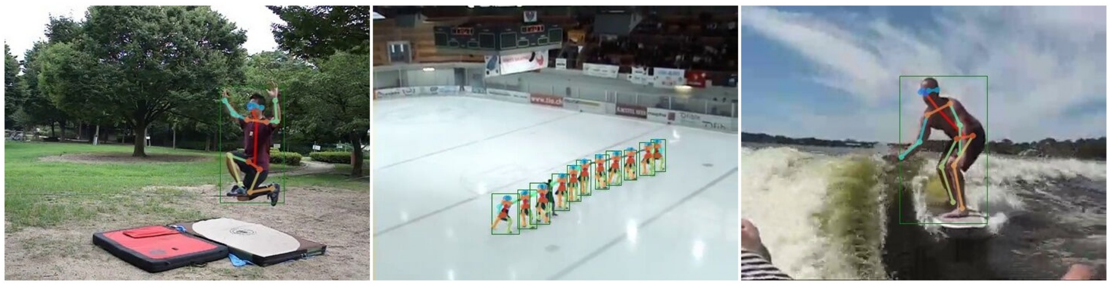
Figure 5.3: Examples of annotations generated from Sports-1M video frames [41].

5.5 KDD Training

By employing the keypoint data distillation (KDD) inference procedure, we compute a set of predictions for each of the unlabelled images. We next use the obtained predictions to generate labels for re-training the model.

Prediction Selection. Since the computed predictions inevitably include predictions of poor quality, generating labels from all of the predictions would lead to the undesirable training outcome. Ideally, we would like to generate labels only from the correct predictions. However, since we have no way of knowing if a prediction for an unlabelled image is correct, we use the predicted scores as a proxy for prediction quality. In particular, we apply a global threshold to the computed predictions and treat only the predictions whose score is above the threshold as 'ground truth'. Note that since the prediction selection is performed at the instance-level we use the bounding box scores.

Although the predicted scores range from zero to one, they have little to do with the notion of probability in practice. In particular, throughout our experiments we observe that models trained using varying amounts of data predict scores of different distributions. Typically, models trained using more data tend to become overly confident and consistently predict very high scores. Therefore, the same score value has different implications on prediction quality in the context of different models. Consequently, the score threshold for the prediction selection should be set dynamically per model.

The raw score threshold can be treated as a hyperparameter and tuned per model. However, in order to reduce the search space we experiment with setting the score threshold as a function of the number of ground truth instances, unlabelled images or predictions. Although all three options have drawbacks, they make reasoning about the threshold setting significantly easier and are straightforward to set in practice.

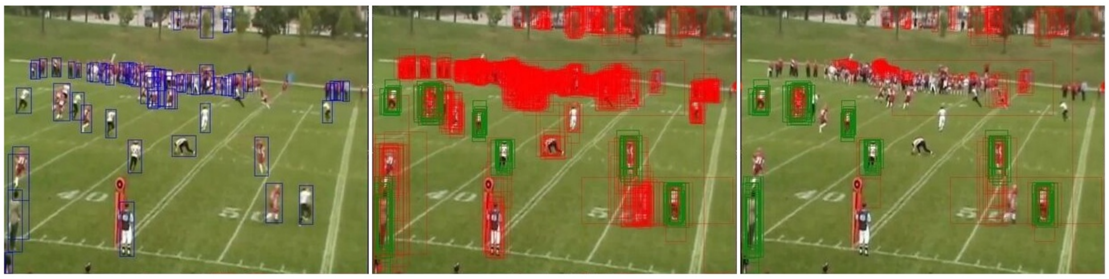

Figure 5.4: *(Left)* All the detected objects (blue). *(Middle)* Bg RoIs (red) and Fg RoIs (green) obtained by applying a single threshold on the detected objects. Note how the crowd of people is wrongly treated as Bg. *(Right)* Bg RoIs (red) and Fg RoIs (green) obtained using the 'gray region' approach. Most of the wrong Bg RoIs are filtered out.

Keypoint Visibility. Selected instance-level predictions consist of K individual keypoints, corresponding to each of the keypoint types (e.g. left elbow, nose, etc.). However, in many of the object views captured by real-world images not all of the keypoint types are visible. Consequently, a large number of the selected instance-level predictions will contain false positive keypoint predictions. In order to account for this, we employ a visibility threshold and consider only the keypoints whose score is above the thresold as visible when generating the object labels.

Background RoIs. We treat the generated object keypoint annotations as ground truth and proceed to re-train the Keypoint R-CNN model. Recall that a RoI is considered foreground if its IoU overlap with a ground truth box is at least 0.5 and background otherwise. Hence, all the RoIs that have no overlap with ground truth boxes are treated as background. In case of images with generated ground truth labels, there may be person instances without the corresponding annotations. Consequently, the RoIs overlapping with such instances would be treated as background, which would likely have a negative effect on the learning process.

An instance may not have a corresponding generated annotation because (a) the instance has not been detected at all or (b) the prediction for the instance has been filtered out based on the score threshold. Although we cannot do much about (a), we can account for RoIs that are considered as background due to (b). Namely, instead of discarding all the predictions with the score below a threshold we introduce a 'gray region' consisting of predictions we are unsure about. Concretely, if the score for a prediction falls within $[\text{score-lo}, \text{score-hi})$ we say that the prediction belongs to the 'gray region'. We then disallow classifying RoIs that have a certain overlap with the predictions from the 'gray region' as background. Similarly as before, the predictions whose score is below score-lo are discarded while the predictions with the score of at least score-hi are treated as ground truth. Note that using a single threshold is a special case of this approach in which score-lo and score-hi are set to the same value.

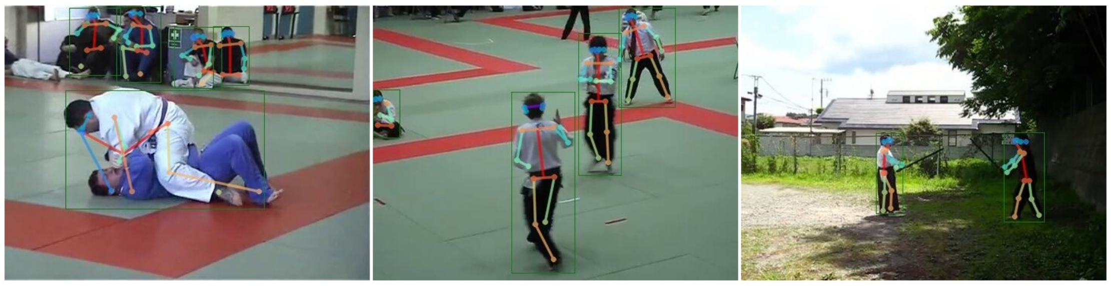

Figure 5.5: More examples of generated data from Sports-1M dataset. Notice the noisy annotation spanning two instances (left). Predicting keypoints correctly for cases like this one is challenging due to the ambiguity.

Data Mixing. Annotations obtained using our data generation procedure are noisy. Thus, when re-training the model using generated data, it is beneficial to also include the clean data used to train the original model. In particular, including the clean data results in better gradient estimates and helps prevent overfitting of random noise structures in the generated data.

One way to achieve this is to combine the clean data and the generated data into a single dataset and perform training as usual. This approach works very well when the amount of data of both types is roughly the same. However, when there is significantly more generated data than there is clean data, the generated data overwhelms the clean data leading to a less efficient learning process.

To overcome the aforementioned limitation we employ a different strategy. In particular, we keep the two types of data separate and construct each mini-batch by sampling images from the two pools in accordance to the desired proportion. Hence, irrespective of what the total amount of data of each type is, we achieve the desired ratio. This training procedure results in a significantly more efficient learning process. An additional benefit of this approach is that it allows us to iterate faster by using shorter learning schedules.

Iterative Training. The whole data distillation process can be performed in a bootstrapping fashion. In particular, the two steps of generating the data and re-training the model can be alternated until convergence.

Note that in each training iteration the weights can be initialised either from the original initial weights or using the model from the previous iteration. When starting from the original weights, the same hyperparameters can be used for each iteration. In contrast, the hyperparameters need to be tuned for each iteration when starting using the model from the previous iteration. Although starting from the previous iteration weights could yield higher performance in theory, we found the alternative of starting from the original weights to work better in practice possibly due to an easier optimisation process.

5.6 Implementation Details

Base Models. We use the Keypoint R-CNN with ResNet50-C4 and ResNet50-FPN backbones as base models for the KDD procedure. Keypoint R-CNN implementation was kindly provided by the authors. We train the base models by using the hyperparameters outlined in the Mask R-CNN work [33].

Inference. At test time, we use 300 and 1000 proposals for the C4 [68] and FPN [50] backbones, respectively. We then run the bounding box branch on these proposals to obtain the initial boxes. The set of boxes is further refined by applying a score threshold of 0.05, non-maximum suppression (NMS) [28] with a threshold of 0.5 and taking the 100 highest scoring boxes. The refined set of boxes is used for applying the keypoint branch on the original image. For a transformed image, the features are first computed by applying the feature extraction trunk. The refined boxes are then transformed and used to apply the keypoint head on the features for the transformed image.

Heuristics. In order to combine the predictions made for the same instance under different transformations, we consider a number of heuristics. For combining the predicted keypoint locations in the image space our heuristics include taking the keypoints with the maximal score (Spatial Max), averaging the keypoint coordinates (Spatial Average) and using the weighted average of the spatial coordinates (Spatial Weighted Average). Note that each heuristic is applied per keypoint type (e.g. nose, left elbow, etc.). Furthermore, we consider heuristics that combine the predicted heatmaps by taking the pixel-wise max across each of the heatmap types (Heatmap Max) or by averaging the heatmaps of each type (Heatmap Average). The keypoint locations in image space are then derived from the combined heatmaps.

Transformations. We implement a range of image transformations each subset of which can be employed during the KDD inference procedure. Aspect ratio of an image is altered by setting the desired absolute aspect ratio or by changing the relative aspect ratio to achieve shrinking/stretching effects. To apply scaling we specify the length of the shorter side in pixels and scale the image while preserving the aspect ratio. For both aspect ratio and scaling transformations we utilise the bilinear interpolation. We further allow flipping the image horizontally and vertically at each of the aspect ratios and scales. Elastic deformation of an image is performed by smoothing a small randomly generated flow field with a Gaussian kernel and using it to warp the image [77]. To perform histogram equalisation of an RGB image we convert the image to the YCbCr color space and apply the histogram equalisation implementation from the OpenCV toolbox [8] to the Y channel.

Size-dependent Scaling. We support applying scaling transformations selectively by object size. However, since KDD inference is performed on unlabelled images, we cannot know the true sizes of the objects in the scenes. Hence, we approximate object sizes using the areas of the predicted bounding boxes. In particular, we use a box area threshold of 180^2 pixels and consider objects as small/medium if their area is below the threshold and as large otherwise. Consequently, scaling up and scaling down are applied only to small/medium and large objects, respectively. In order to take advantage of the shared computation we run inference for each object size at each scale, but only consider the predictions from the appropriate scales when combining the predictions. Note that although we determined the box area threshold empirically, we found that it matches the statistics of the object sizes in the labelled training data we considered quite well.

Label Generation. To generate object keypoint annotations, we treat keypoints with logit (before softmax) scores of at least 1.0 as visible. Unless otherwise stated, we set the instance-level score-hi threshold such that only N of the computed predictions have scores of at least score-hi, where N is twice the number of manually-annotated objects used to train the base model. Furthemore, when the 'gray region' strategy is employed we set the lo-score threshold to 0.7 and filter out all the background RoIs whose IoU overlap with a box from the 'gray region' is at least 0.1.

Data Balancing. When re-training our models with generated data, we set the target ratio of annotated-to-generated images in each mini-batch to $6 : 4$. Following the common practice [7], instead of sampling each mini-batch individually, we maintain two queues of data and iterate over the permutations of the samples with a fixed stride.

Training. We consider a RoI with an overlap of at least 0.5 with a ground truth box foreground and background otherwise. RoIs are sampled to ensure the target foreground-to-background ratio of $1 : 3$. Each mini-batch consists of 2 images per GPU and N RoIs per image, where N is 64 for the C4 backbone [68] and 512 in case of the FPN backbone [50]. We adopt synchronised gradient descent training using a 8-GPU machine. The training experiments are conducted using the Caffe2 framework.[1]

For training both the C4 and the FPN backbones we use the starting learning rate of 0.02. In case of the C4 backbone the learning rate is divided by 10 after 3/4 of the total number of iterations. When training the FPN backbone we divide the learning rate by 10 after 2/3 and then again after 8/9 of the total number of iterations. For both backbones, we use a weight decay of 0.0001 and a momentum of 0.9.

We use the RPN with anchors that span 5 scales and 3 aspect ratios, as in [50]. In all of our experiments we train a separate RPN that does not share features with the Keypoint R-CNN, which results in a more controlled setup and allows for convenient ablation. Nevertheless, the equivalent feature-sharing models can be trained.

[1] `https://github.com/caffe2/caffe2`

5.7 Experiments

Labelled Data. We use the MS COCO dataset [51] as the source of labelled data for the KDD procedure. COCO contains around 80k training and 40k validation images with over 850k annotated objects from 80 categories. However, only the person category has annotated keypoints. In particular, the keypoints are annotated for around 200k person instances. COCO contains 17 person keypoint types (e.g. left hip, right ear, nose, etc.), not all of which are visible for every instance. Following the common practice [4], we use the union of 80k training and 35k validation images (trainval35k) as the source of training data and leave out the remaining 5k validation images (minival) for validation. Note that many of the training and validation images contain no people.

Unlabelled Data. In order to construct collections of unlabelled images we utilise frames from the Sports-1M video dataset [41]. The dataset consists of 1.1 million YouTube videos belonging to 487 categories of sports. This dataset is well suited for our KDD approach since most of the sport categories contain people. Hence, compared to a dataset where people are not as common, using Sports-1M will likely require fewer inference runs, and thus less computation, to generate a certain number of annotations.

Image collections are created from the Sports-1M videos using hierarchical sampling. In particular, N sport categories are first sampled from the dataset, M videos are then sampled from each of the selected categories and finally K frames are sampled from each of the selected videos, to obtain a collection of $N \times M \times K$ images. Note that the frames from the same video are more correlated than the frames from different videos of the same category and even more so than the frames from videos of different category. Hence, choosing larger values for N and M, and smaller values for K is generally preferred.

Preprocessing. Some of of the YouTube videos that comprise the Sports-1M dataset contain large amounts of motion blur, which could potentially introduce bias. However, we found Keypoint R-CNN to be suprisingly robust to motion blur and not to exhibit any systematic errors. Thus, we do not perform any form of data filtering or preprocessing.

Metrics. We evaluate our approach on the COCO Keypoint Challenge using the standard COCO metrics for characterising the performance of a keypoint detector. In order to be able to quantify the similarity between the ground truth and the predicted keypoints COCO defines an object keypoint similary (OKS) measure. The OKS plays a role analogous to the IoU for boxes/segments. Consequently, thresholding the OKS defines the matches between the ground truth and predicted keypoints and allows computing average precision (AP). We report the AP for different threshold settings including AP (averaged over ten OKS thresholds), AP_{75} (OKS 0.75), and AP_{50} (OKS 0.5). We also report the AP for medium and large objects denoted by AP_M and AP_L, respectively. Note that small objects in COCO contain no keypoint annotations.

backbone	scales	HF	SD	heuristic	AP	AP_{50}	AP_{75}	AP_M	AP_L
ResNet50-C4	800			n/a	58.6	82.7	63.6	52.6	68.6
ResNet50-C4	800	✓		hm avg	60.5	83.3	65.6	54.8	70.2
ResNet50-C4	400-1200	✓		hm avg	62.5	84.4	68.2	57.4	71.7
ResNet50-C4	400-1200	✓	✓	hm avg	**63.2**	**84.9**	**69.0**	**58.5**	**72.0**
ResNet50-FPN	800			n/a	64.2	86.4	69.2	59.1	72.6
ResNet50-FPN	800	✓		hm avg	65.4	87.0	70.8	60.4	73.6
ResNet50-FPN	400-1200	✓		hm avg	66.6	87.5	72.5	61.7	74.7
ResNet50-FPN	400-1200	✓	✓	hm avg	**67.0**	**87.8**	**73.0**	**62.3**	**74.9**

Table 5.1: COCO minival (5k) keypoint detection AP (%). All models are trained on COCO trainval35k. The scales denote the lengths of shorter image sides in pixels. Ranges are in increments of 100. Entries that employ KDD inference use heatmap averaging heuristic. Legend: HF: applying horizontal flip at each scale, SD: using size-dependent scaling.

5.7.1 Transformation Selection

In order to determine the optimal set of transformations and the heuristic to use for the KDD label generation procedure, we employ a data-driven approach. In particular, we train a standard Keypoint R-CNN model, apply KDD with different combinations of transformations and heuristics at test time and select the set of transformations and the heuristic that yield the highest accuracy.

Experimental Setup. We use a standard Keypoint R-CNN model with ResNet50-C4 backbone trained on COCO trainval35k and evaluate different combinations of transformations and heuristics at test time on COCO minival. We performed a thorough evaluation of different KDD settings and report only the most effective ones in Table 5.1.

Optimal Combination. For all of the combinations of transformations considered we found the heatmap averaging heuristic to consistently perform the best. Out of all the image transformations, horizontal flipping and scaling stood out as the most effective ones. Namely, including horizontal flip in any combination increases the overall accuracy. Although applying standard scaling on its own improves the accuracy, scaling objects depending on their size is superior. Our best performing combination, shown in Table 5.1, brings an absolute 4.8 AP improvement over the ResNet50-C4 baseline.

Generalisation. We performed our search for the optimal KDD inference setting using the C4 model. To see if our findings generalise to FPN models, we evaluated the most effective combinations found for the C4 model using the FPN equivalent. The best performing combination gives an absolute 2.8 AP improvement over the FPN baseline, as shown in Table 5.1. However, the relative improvement from the baseline is smaller, likely due to the invariance added by the feature pyramids utilised in the FPN.

backbone	GT	GEN	AP	AP_{50}	AP_{75}	AP_M	AP_L
ResNet50-C4	val35k	n/a	47.7	75.3	50.0	41.9	57.0
ResNet50-C4	val35k	train	**54.7**	**79.8**	**58.7**	**49.0**	**64.6**
ResNet50-C4	trainval35k	n/a	58.6	82.7	63.6	52.6	68.6
ResNet50-FPN	val35k	n/a	54.9	80.5	59.0	50.1	62.8
ResNet50-FPN	val35k	train	**60.2**	**83.8**	**65.4**	**55.2**	**68.4**
ResNet50-FPN	trainval35k	n/a	64.2	86.4	69.2	59.1	72.6

Table 5.2: COCO minival (5k) keypoint detection AP (%). Legend: GT: training data with ground truth annotations, GEN: training data with generated data.

5.7.2 Controlled Setting

In order to quantify how the quality of the labels obtained using our approach compares to the manually-annotated labels, we designed a controlled experiment. In particular, we use a subset of the images from a labelled dataset to train a base model. We then consider the rest of the images as unlabelled and apply the KDD procedure. Hence, we can see if the model trained using KDD improves and, if so, how close it comes to the upper bound obtained by training an equivalent model using all of the labelled images.

Experimental Setup. We perform this experiment using the COCO dataset. In particular, we use the images from val35k as labelled data and treat the images from train as unlabelled. We perform the same experiment with the ResNet50-C4 and ResNet50-FPN backbones. In both cases we employ the set of best performing transformations, outlined in the revious section, for KDD inference during label generation.

Results. We present the results on COCO minival in Table 5.2. In case of the ResNet50-C4 backbone, the KDD model improves over the val35k model by absolute 7.0 AP. Furthermore, it comes surprisingly close to the trainval35k upper bound (3.9 AP). The ResNet50-FPN version achieves similar results by bringing an improvement of 5.3 AP over the val35k model and getting to 4.0 AP from the trainval35k upper bound.

Although we have seen very positive signal in the experiments presented in this section, it is important to keep in mind the differences between this setting and the more general setting where arbitrary image collections are used as a source of unlabelled data. Firstly, in this setting both the labelled data and the unlabelled data come from the same distribution. Secondly, COCO images are considerably cleaner than some of the potential unlabelled data sources, such as Sports-1M frames coming from YouTube videos.

We emphasise that the reported results, in this section and the following ones, do not employ KDD inference at test time. Utilising KDD inference at test time has a potential to further improve the performance but is not the focus of our work.

backbone	GT	GEN	AP	AP_{50}	AP_{75}	AP_M	AP_L
ResNet50-C4	trainval35k	n/a	58.6	82.7	63.6	52.6	68.6
ResNet50-C4	trainval35k	Sports-1M	**61.1**	**83.4**	**66.2**	**55.4**	**71.0**
ResNet50-FPN	trainval35k	n/a	64.2	86.4	69.2	59.1	72.6
ResNet50-FPN	trainval35k	Sports-1M	**66.7**	**87.5**	**72.9**	**61.4**	**75.3**

Table 5.3: COCO minival (5k) keypoint detection AP (%). Legend: GT: training data with ground truth annotations, GEN: training data with generated data.

5.7.3 Large Dataset

Encouraged by the positive results obtained in the controlled setting, we proceed to see if our approach generalises to a large dataset of unlabelled images coming from a different distribution than the labelled data.

Experimental Setup. We construct a collection of around 180k unlabelled images using the frames from the Sports-1M videos. In particular, we first randomly select 100 sport categories and then sample a single frame from around 1800 randomly sampled videos from each of the selected categories. As the source of labelled data we use all of the COCO images. The ResNet50-C4 and the ResNet50-FPN base models are trained on trainval35k for 80k and 90k iterations, respectively. When training on the union of labelled and generated data we increase the maximal number of iterations for both models four times. All other hyperparameters are set following Section 5.6.

Results. We compare the models trained using the KDD approach to the baseline Keypoint R-CNN models and report the results on COCO minival in Table 5.3. The models trained using KDD outperform the baselines for both of the backbone architectures. Interestingly, the AP increase of absolute 2.5 is the same in both cases. We note that the performance increase is significantly larger in the AP_{75} than in the AP_{50} metric, which is where most of the improvement in the AP comes from.

The equal AP improvement of the two backbone versions is in contrast with the controlled setting, where C4 had a larger increase in the AP than the FPN, which might suggest that in the context of a large dataset the FPN backbone benefits from greater representational capacity.

In order to see if the KDD approach is affected by the variance in the sampled collection of Sports-1M frames, we experimented with using image collections of similar size sampled from the varying number of categories and videos. We obtained similar results in all cases, which suggests that the KDD approach is robust to the variance in the Sports-1M distribution. Surprisingly, we found that when applied to a large dataset the KDD method is insensitive to the lower bound used for the 'gray region' of the detections.

	GT	GEN	AP	AP_{50}	AP_{75}	AP_M	AP_L
CMU-Pose+++ [12]	COCO	n/a	61.8	84.9	67.5	57.1	68.2
G-RMI [61]	COCO+MPII	n/a	62.4	84.0	68.5	59.1	68.1
KRCNN [33]	COCO	n/a	62.7	87.0	68.4	57.4	71.1
KRCNN w KDD	COCO	Sports-1M	**64.7**	**88.0**	**70.9**	**59.8**	**72.6**

Table 5.4: COCO test-dev keypoint detection AP (%). CMU-Pose+++ is the 2016 competition winner. G-RMI uses two models (Inception-ResNet-v2 and ResNet-101) trained on COCO and MPII [1] (40k labelled instances). Both of the Keypoint R-CNN (KRCNN) models use the ResNet50-FPN backbone. Legend: GT: training data with ground truth annotations, GEN: training data with generated annotations.

Following the significant improvements achieved over the Keypoint R-CNN (KRCNN) baselines on COCO minival, we compare the KRCNN trained using the KDD approach to the state-of-the-art keypoint detectors on COCO test-dev, which has no disclosed labels. We use the same ResNet50-FPN model as in the COCO minival experiments.

Main Results. We report the results on COCO test-dev in Table 5.4. KRCNN trained using the KDD approach outperforms all previous state-of-the-art models. This includes CMU-Pose+++ the winner of the COCO 2016 Keypoint Challenge, G-RMI [61] trained using an additional labelled dataset and the standard version of the KRCNN [33]. Furthermore, KRCNN with KDD does so by a margin of 2.0 AP, which is considerably larger than the difference in performance between the other entries (at most 0.6 AP).

Medium Objects. Although KRCNN outperforms G-RMI in the overall AP, the G-RMI model performs better on medium objects. Both CMU-Pose+++ and KRCNN models have roughly the same accuracy on medium objects and were trained using only the COCO dataset, which has no annotations for small objects. Since KDD procedure places no limitations on object sizes when generating labels, it is likely that the generated data used for training the KRCNN with KDD model includes small objects. Thus, it is possible that the significant increase in the medium object accuracy between the KRCNN and the KRCNN with KDD models was stimulated by the generated annotations for small objects.

Large Objects. The KRCNN model outperforms the CMU-Pose+++ and G-RMI models on large objects by a wide margin. KRCNN with KDD widens this gap by bringing an additional improvement over the KRCNN performance on large objects.

Like in the case of COCO minival, the KRCNN with KDD model improvement over the base KRCNN model comes mostly from the AP_{75} metric. In contrast, KRCNN makes a difference over the CMU-Pose+++ and G-RMI models in the AP_{50} metric.

5.8 Discussion

We proposed a simple approach for semi-supervised learning from a collection of unlabelled images. Our method combines model predictions computed at different image transformations to generate labels for unlabelled images that are then used to re-train the model. We applied our approach to the human pose estimation task using the Keypoint R-CNN model. We conducted experimental analysis on the COCO keypoints dataset to demonstrate the effectiveness of our approach. Furthermore, we reported state-of-the-art results on the COCO Keypoints Challenge.

Our data distillation approach is very general and can be applied to other instance-level tasks, such as detection and segmentation. We leave this direction for future work.

Chapter 6

Conclusion

Over the recent years, we witnessed rapid progress in visual perception tasks that has been mostly driven by the availability of large manually annotated datasets. Machines have become particularly effective in the image classification task, even surpasing human-level performance. However, the increase in performance in the instance-level tasks, such as object segmentation and human pose estimation has not been as rapid. In part, due to the difficulty of manually annotating very large datasets for these tasks.

In our study we explored if semi-supervised learning approaches could help advance object-instance recognition without relying on additional manually annotated data. Concretely, in Chapter 3 we have shown that models trained in a fully-supervised setting can provide useful signal for learning an instance-level task from unlabelled videos. In Chapter 4, we developed a hard example mining approach that relies on the spatio-temporal information available in videos. We then generalised our approach in Chapter 5 and proposed a simple but effective method for semi-supervised learning from unlabelled images. By combining our method with the Mask R-CNN model we achieved state-of-the-art performance in human pose estimation.

Over the course of our investigation we have seen some promising results. We believe that this is an interesting research direction and hope that our work may inspire future research in semi-supervised learning of instance-level recognition.

Although our study has provided us with valuable insights, it has also posed many questions that we would like to address in future work.

1. The training procedure for generated data used in Chapter 4 suffered from significant limitations. We addressed those limitations in Chapter 5 by using data mixing strategies and employing regional thresholding. Although, the approach presented in Chapter 5 is generally more compelling, we would still like to go back to the setting from Chapter 4 and apply our updated training procedure.
2. We have seen very positive signal from applying our approach to a relatively large sample of Sports-1M frames. However, in the context of the entire Sports-1M dataset, we used less than 10% of the available data. Hence, we would like to investigate what are the limits of our approach, both in terms of the training duration and the size of the unlabelled dataset.
3. In this study we applied the data distillation approach to human pose estimation. To determine if our approach generalises to other instance-level tasks, we would like to apply the data distillation procedure to boxes and masks.
4. One way to generalise our data distillation approach is to consider different instance-level tasks. Another path forward is to consider different levels of abstraction. In particular, in the same way data distillation is applied to model outputs it could be applied to features. In fact, model outputs are 'features', just low dimensional and interpretable. One of the challenges in this direction is coming up with inverse feature transformations (in case of geometric transforms inverses were available in closed-form). One way to overcome this issue would be to train another model to serve as the inverse. The model could be trained using data generated from dense object tracks, similarly to how we approached learning of tracking. This direction is also related to slow feature analysis [93] and equivariant features [47]
5. We implemented the data distillation approach in a bootsrapping-style fashion. An alternative approach would be to think about data distillation as a form of regularisation and apply it during training. In case of Mask R-CNN, one way to accomplish this would be to add model heads corresponding to different transformations and form a multi-transformation loss.

Bibliography

[1] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. “2D Human Pose Estimation: New Benchmark and State of the Art Analysis”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2014.

[2] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. “A database and evaluation methodology for optical flow”. In: *International Journal of Computer Vision* 92.1 (2011), pp. 1–31.

[3] Paul Beardsley, Phil Torr, and Andrew Zisserman. “3D model acquisition from extended image sequences”. In: *European conference on computer vision*. Springer. 1996, pp. 683–695.

[4] Sean Bell, C Lawrence Zitnick, Kavita Bala, and Ross Girshick. “Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2874–2883.

[5] Luca Bertinetto, Jack Valmadre, João F Henriques, Andrea Vedaldi, and Philip HS Torr. “Fully-convolutional siamese networks for object tracking”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 850–865.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[7] Léon Bottou. “Stochastic gradient descent tricks”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[8] G. Bradski. “OpenCV”. In: *Dr. Dobb’s Journal of Software Tools* (2000).

[9] Matthew Brown and David G Lowe. “Automatic panoramic image stitching using invariant features”. In: *International Journal of Computer Vision* 74.1 (2007), pp. 59–73.

[10] Cristian Buciluă, Rich Caruana, and Alexandru Niculescu-Mizil. “Model compression”. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2006, pp. 535–541.

[11] John Canny. "A computational approach to edge detection". In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), pp. 679–698.

[12] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. "Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields". In: *arXiv preprint arXiv:1611.08050* (2016).

[13] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. "Semantic image segmentation with deep convolutional nets and fully connected crfs". In: *arXiv preprint arXiv:1412.7062* (2014).

[14] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. *Torch7: A Matlab-like Environment for Machine Learning.*

[15] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection". In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on.* Vol. 1. IEEE. 2005, pp. 886–893.

[16] Marc Deisenroth and Stefanos Zafeiriou. *Mathematics for Inference and Machine Learning.* `https://www.doc.ic.ac.uk/~mpd37/teaching/2016/496/notes.pdf`. Accessed: 2017-01-25.

[17] Piotr Dollár, Zhuowen Tu, Pietro Perona, and Serge Belongie. "Integral channel features". In: (2009).

[18] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. "Flownet: Learning optical flow with convolutional networks". In: *Proceedings of the IEEE International Conference on Computer Vision.* 2015, pp. 2758–2766.

[19] David Eigen et al. "Restoring an Image Taken through a Window Covered with Dirt or Rain". In: *The IEEE International Conference on Computer Vision (ICCV).* Dec. 2013.

[20] David Eigen, Christian Puhrsch, and Rob Fergus. "Depth map prediction from a single image using a multi-scale deep network". In: *Advances in neural information processing systems.* 2014, pp. 2366–2374.

[21] Mark Everingham, Josef Sivic, and Andrew Zisserman. "Hello! My name is... Buffy"–Automatic Naming of Characters in TV Video." In: *BMVC.* Vol. 2. 4. 2006, p. 6.

[22] Gunnar Farnebäck. "Two-frame motion estimation based on polynomial expansion". In: *Scandinavian conference on Image analysis.* Springer. 2003, pp. 363–370.

[23] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. "Object detection with discriminatively trained part-based models". In: *IEEE transactions on pattern analysis and machine intelligence* 32.9 (2010), pp. 1627–1645.

[24] Martin A Fischler and Robert C Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[25] Martin A Fischler and Robert A Elschlager. "The representation and matching of pictorial structures". In: *IEEE Transactions on computers* 100.1 (1973), pp. 67–92.

[26] Ross Girshick. "Fast r-cnn". In: *Proceedings of the IEEE International Conference on Computer Vision.* 2015, pp. 1440–1448.

[27] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2014, pp. 580–587.

[28] Ross Girshick, Forrest Iandola, Trevor Darrell, and Jitendra Malik. "Deformable part models are convolutional neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2015, pp. 437–446.

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

[30] Lars Kai Hansen and Peter Salamon. "Neural network ensembles". In: *IEEE transactions on pattern analysis and machine intelligence* 12.10 (1990), pp. 993–1001.

[31] Sam Hare, Stuart Golodetz, Amir Saffari, Vibhav Vineet, Ming-Ming Cheng, Stephen L Hicks, and Philip HS Torr. "Struck: Structured output tracking with kernels". In: *IEEE transactions on pattern analysis and machine intelligence* 38.10 (2016), pp. 2096–2109.

[32] Chris Harris and Mike Stephens. "A combined corner and edge detector." In: *Alvey vision conference.* Vol. 15. 50. Manchester, UK. 1988, pp. 10–5244.

[33] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask r-cnn". In: *arXiv preprint arXiv:1703.06870* (2017).

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* June 2016.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision.* 2015, pp. 1026–1034.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Spatial pyramid pooling in deep convolutional networks for visual recognition”. In: *European Conference on Computer Vision.* Springer. 2014, pp. 346–361.

[37] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. “High-speed tracking with kernelized correlation filters”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.3 (2015), pp. 583–596.

[38] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the knowledge in a neural network”. In: *arXiv preprint arXiv:1503.02531* (2015).

[39] Berthold KP Horn and Brian G Schunck. “Determining optical flow”. In: *Artificial intelligence* 17.1-3 (1981), pp. 185–203.

[40] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. “Tracking-learning-detection”. In: *IEEE transactions on pattern analysis and machine intelligence* 34.7 (2012), pp. 1409–1422.

[41] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. “Large-scale video classification with convolutional neural networks”. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition.* 2014, pp. 1725–1732.

[42] Johannes Kopf, Michael F Cohen, and Richard Szeliski. “First-person hyper-lapse videos”. In: *ACM Transactions on Graphics (TOG)* 33.4 (2014), p. 78.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25.* Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105.

[44] Anders Krogh, Jesper Vedelsby, et al. “Neural network ensembles, cross validation, and active learning”. In: *Advances in neural information processing systems* 7 (1995), pp. 231–238.

[45] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551.

[46] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[47] Karel Lenc and Andrea Vedaldi. "Understanding image representations by measuring their equivariance and equivalence". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015, pp. 991–999.

[48] Yin Li, Manohar Paluri, James M Rehg, and Piotr Dollár. "Unsupervised learning of edges". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016, pp. 1619–1627.

[49] Yi Li, Kaiming He, Jian Sun, et al. "R-fcn: Object detection via region-based fully convolutional networks". In: *Advances in Neural Information Processing Systems.* 2016, pp. 379–387.

[50] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. "Feature Pyramid Networks for Object Detection". In: *arXiv preprint arXiv:1612.03144* (2016).

[51] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context". In: *European Conference on Computer Vision.* Springer. 2014, pp. 740–755.

[52] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2015, pp. 3431–3440.

[53] Ilya Loshchilov and Frank Hutter. "Online batch selection for faster training of neural networks". In: *arXiv preprint arXiv:1511.06343* (2015).

[54] Bruce D Lucas, Takeo Kanade, et al. "An iterative image registration technique with an application to stereo vision". In: (1981).

[55] Pina Marziliano, Frederic Dufaux, Stefan Winkler, and Touradj Ebrahimi. "A no-reference perceptual blur metric". In: *Image Processing. 2002. Proceedings. 2002 International Conference on.* Vol. 3. IEEE. 2002, pp. III–III.

[56] Ishan Misra, C Lawrence Zitnick, and Martial Hebert. "Shuffle and learn: unsupervised learning using temporal order verification". In: *European Conference on Computer Vision.* Springer. 2016, pp. 527–544.

[57] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective.* The MIT Press, 2012.

[58] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[59] Mehdi Noroozi and Paolo Favaro. "Unsupervised learning of visual representations by solving jigsaw puzzles". In: *European Conference on Computer Vision*. Springer. 2016, pp. 69–84.

[60] Peter Ochs, Jitendra Malik, and Thomas Brox. "Segmentation of moving objects by long term video analysis". In: *IEEE transactions on pattern analysis and machine intelligence* 36.6 (2014), pp. 1187–1200.

[61] George Papandreou, Tyler Zhu, Nori Kanazawa, Alexander Toshev, Jonathan Tompson, Chris Bregler, and Kevin Murphy. "Towards Accurate Multi-person Pose Estimation in the Wild". In: *arXiv preprint arXiv:1701.01779* (2017).

[62] Dennis Park, C Lawrence Zitnick, Deva Ramanan, and Piotr Dollár. "Exploring weak stabilization for motion feature extraction". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2013, pp. 2882–2889.

[63] Deepak Pathak, Ross Girshick, Piotr Dollár, Trevor Darrell, and Bharath Hariharan. "Learning Features by Watching Objects Move". In: *CVPR*. 2017.

[64] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. "Context encoders: Feature learning by inpainting". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2536–2544.

[65] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung. "A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation". In: *Computer Vision and Pattern Recognition*. 2016.

[66] Pedro O Pinheiro, Ronan Collobert, and Piotr Dollar. "Learning to segment object candidates". In: *Advances in Neural Information Processing Systems*. 2015, pp. 1990–1998.

[67] Pedro O Pinheiro, Tsung-Yi Lin, Ronan Collobert, and Piotr Dollár. "Learning to refine object segments". In: *European Conference on Computer Vision*. Springer. 2016, pp. 75–91.

[68] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[69] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention.* Springer. 2015, pp. 234–241.

[70] Henry A Rowley, Shumeet Baluja, and Takeo Kanade. “Neural network-based face detection”. In: *IEEE Transactions on pattern analysis and machine intelligence* 20.1 (1998), pp. 23–38.

[71] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning internal representations by error-propagation”. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1.* Vol. 1. 6088. MIT Press, Cambridge, MA, 1986, pp. 318–362.

[72] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[73] Cordelia Schmid, Roger Mohr, and Christian Bauckhage. “Evaluation of Interest Point Detectors”. In: *Int. J. Comput. Vision* 37.2 (June 2000), pp. 151–172. ISSN: 0920-5691.

[74] J. Shi and C. Tomasi. “Good features to track”. In: *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on.* IEEE. 1994, pp. 593–600.

[75] Wenzhe Shi, Jose Caballero, Ferenc Huszar, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. “Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* June 2016.

[76] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. “Training region-based object detectors with online hard example mining”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016, pp. 761–769.

[77] Patrice Y Simard, David Steinkraus, John C Platt, et al. “Best practices for convolutional neural networks applied to visual document analysis.” In: *ICDAR.* Vol. 3. 2003, pp. 958–962.

[78] Karen Simonyan and Andrew Zisserman. “Two-stream convolutional networks for action recognition in videos”. In: *Advances in neural information processing systems.* 2014, pp. 568–576.

[79] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[80] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, and Francesc Moreno-Noguer. "Fracking deep convolutional image descriptors". In: *arXiv preprint arXiv:1412.6537* (2014).

[81] Arnold WM Smeulders, Dung M Chu, Rita Cucchiara, Simone Calderara, Afshin Dehghan, and Mubarak Shah. "Visual tracking: An experimental survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.7 (2014), pp. 1442–1468.

[82] Patrik Sundberg, Thomas Brox, Michael Maire, Pablo Arbeláez, and Jitendra Malik. "Occlusion boundary detection and figure/ground assignment from optical flow". In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on.* IEEE. 2011, pp. 2233–2240.

[83] Kah-Kay Sung. "Learning and example selection for object and pattern detection". In: (1996).

[84] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2015, pp. 1–9.

[85] Richard Szeliski. *Computer vision: algorithms and applications.* Springer Science & Business Media, 2010.

[86] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. "Yfcc100m: The new data in multimedia research". In: *Communications of the ACM* 59.2 (2016), pp. 64–73.

[87] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. "Learning spatiotemporal features with 3d convolutional networks". In: *Proceedings of the IEEE International Conference on Computer Vision.* 2015, pp. 4489–4497.

[88] Bill Triggs. "Detecting keypoints with stable position, orientation, and scale under illumination changes". In: *European conference on computer vision.* Springer. 2004, pp. 100–113.

[89] Mark Tygert. "Poor starting points in machine learning". In: *arXiv preprint arXiv:1602.02823* (2016).

[90] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. "Selective search for object recognition". In: *International journal of computer vision* 104.2 (2013), pp. 154–171.

[91] Lijun Wang, Wanli Ouyang, Xiaogang Wang, and Huchuan Lu. “Visual Tracking With Fully Convolutional Networks”. In: *The IEEE International Conference on Computer Vision (ICCV).* Dec. 2015.

[92] Xiaolong Wang and Abhinav Gupta. “Unsupervised learning of visual representations using videos”. In: *Proceedings of the IEEE International Conference on Computer Vision.* 2015, pp. 2794–2802.

[93] Laurenz Wiskott and Terrence J Sejnowski. “Slow feature analysis: Unsupervised learning of invariances”. In: *Neural computation* 14.4 (2002), pp. 715–770.

[94] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. “Aggregated residual transformations for deep neural networks”. In: *arXiv preprint arXiv:1611.05431* (2016).

[95] Larry Yaeger, Richard Lyon, and Brandyn Webb. “Effective training of a neural network character classifier for word recognition”. In: *NIPS.* Vol. 9. 1996, pp. 807–813.

[96] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision.* Springer. 2014, pp. 818–833.

[97] Gao Zhu, Fatih Porikli, and Hongdong Li. “Robust visual tracking with deep convolutional neural network based object proposals on PETS”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops.* 2016, pp. 26–33.

[98] C Lawrence Zitnick and Piotr Dollár. “Edge boxes: Locating object proposals from edges”. In: *European Conference on Computer Vision.* Springer. 2014, pp. 391–405.