Karlsruhe Institute of Technology

Carnegie Mellon University

Application of Neural Networks for Heading Direction Estimation

Bachelor's Thesis of

Xuan Tung Nguyen

at the Department of Informatics
Institute for Anthropomatics and Robotics (IAR)
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany

School of Computer Science
The Robotics Institute (RI)
Canergie Mellon University (CMU)
Pittsburgh, United States

Reviewer:	Prof. Alexander Waibel
Second reviewer:	Prof. Tamim Asfour
Advisor:	Dr. Thanh-Le Ha

01. February 2019 – 21. May 2019

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text, and have followed the rules of the KIT for upholding good scientific practice.

Karlsruhe, 21.05.2019

. . . . . .

(Xuan Tung Nguyen)

Acknowledgements

First and foremost I would like to express my gratitude to Prof. Alexander Waibel and the CLICS exchange program for the opportunity to write this bachelor's thesis at the CMU. I would also like to thank Prof. Tamim Asfour for his constructive reviews and feedback, which helped to complete this work.

I wish to thank Dr. Sebastian Scherer and Rogerio Bonatti for the warm welcome into the Air Lab, where I spent my time at the CMU. I also sincerely appreciate the help of Dr. Wenshan Wang, whose research inspired my thesis.

Many thanks to my advisor Dr. Thanh-Le Ha for his devoted support and guidance, without which this work would not have been possible.

My appreciation for my colleagues in Prof. Waibel's interactive drone research team. Our work together laid the foundation for my own research.

To my friends and family who have always encouraged and supported me ever since I started living on my own, far from home, I am deeply grateful.

Abstract

Recent rapid advancements of drone technology and control has made consumer drones more commonplace. Pedestrian following is an desirable feature for both outdoor and indoor drones. To help with this task, it is important for drones to predict the heading direction of a target person from pure visual input. While learning-based, data-driven approaches like neural networks achieve good results for similar problems in the robotics, they require large dataset and suffer from low generalization capability. Semi-supervised learning can help reduce the dataset constraint. In this work, we investigate the performance of different neural network models in the task of estimating human heading direction, given the a person's bounding box. We show that by integrating semi-supervised learning into a predictor, the overall prediction results and generalization capability can be improved. We measure a predictor model by comparing the performance on three distinct datasets and evaluate the prediction error.

Contents

1 Introduction **1**

2 Background **3**

2.1 Heading Direction Estimation . . . 3
2.1.1 Metrics . . . 3
2.2 Interactive Indoor Drone Assistant Project . . . 4
2.2.1 Motivation . . . 4
2.2.2 Platform . . . 4
2.2.3 Approach . . . 6
2.2.4 Future works . . . 7
2.3 Neural Network . . . 8
2.3.1 Activation function . . . 8
2.3.2 Backpropagation algorithm . . . 9
2.3.3 Batch normalization . . . 10
2.3.4 Time Delay Neural Network - Convolutional Neural Network . . . 11
2.3.5 Recurrent Neural Network . . . 12
2.3.6 Semi-supervised Learning . . . 14

3 Approach **15**

3.1 Supervised Loss . . . 15
3.2 Base Extractor . . . 16
3.3 Mobile Extractor . . . 17
3.3.1 Depthwise separable Convolution . . . 17
3.3.2 Architecture . . . 17
3.4 Vanilla Estimator . . . 17
3.5 RNN Estimator . . . 19
3.6 Semi-supervised learning Predictor . . . 19
3.6.1 Unsupervised loss function for temporal continuity . . . 20
3.6.2 Combined loss function . . . 21
3.7 Data Augmentation . . . 21

4 Experiments **23**

4.1 Datasets . . . 23
4.1.1 DukeMTMC . . . 23
4.1.2 DroLAB . . . 24
4.1.3 COMBI . . . 25
4.2 Implementation . . . 26

5 Evaluation **29**
5.1 Results on DukeMTMC 29
5.1.1 Selection of λ 29
5.1.2 Extractor influence 31
5.1.3 Semi-supervised learning with data constraint 33
5.1.4 Test results 33
5.2 Results on DroLAB 34
5.3 Results on COMBI 36

6 Conclusion **39**
6.1 Summary 39
6.2 Future works 40

Bibliography **41**

List of Figures

2.1 Examples of HDE task . . . 3
2.2 CF2.0, Crazyradio PA, Flow deck . . . 5
2.3 FPV camera with receiver . . . 5
2.4 Customized CF2.0 . . . 5
2.5 Indoor drone assistant system overview . . . 6
2.6 Diagram of laboratory corridor . . . 7
2.7 Output of the obstacle avoidance system . . . 7
2.8 Feed-forward network . . . 8
2.9 Rectifier and hyperbolic tangent functions . . . 9
2.10 Training and validation error graph . . . 10
2.11 Convolution and pooling operation . . . 11
2.12 RNN loop . . . 12
2.13 Backpropagation on unfolded RNN . . . 13
2.14 GRU cell . . . 14

3.1 Predictor model overview . . . 15
3.2 Standard convolution layer . . . 16
3.3 Base extractor architecture . . . 16
3.4 Depthwise separable convolution layer . . . 18
3.5 Mobile extractor architecture . . . 18
3.6 Vanilla estimator . . . 19
3.7 RNN estimator . . . 20
3.8 Combined loss calculation . . . 21
3.9 Data augmentation . . . 22

4.1 Frames extracted from DukeMTMC videos . . . 23
4.2 DukeMTMC sample sequence . . . 24
4.3 DroLAB raw images . . . 25
4.4 DroLab sample sequence . . . 25
4.5 COMBI samples . . . 27

5.1 Angle error for different values of λ . . . 30
5.2 Supervised vs unsupervised loss . . . 31
5.3 Base vs mobile extractor . . . 32
5.4 Angle error with labeled data constraint . . . 34
5.5 Test accuracy on DroLAB . . . 36
5.6 Test accuracy on COMBI . . . 37

List of Tables

3.1 Base extractor body architecture 16
3.2 Mobil extractor body architecture 19

4.1 Ground truth for manual labelling 26
4.2 COMBI dataset composition 26

5.1 Datasets statistics 29
5.2 Validation results on DukeMTMC regarding λ 30
5.3 Extractor-Estimator validation results on DukeMTMC 32
5.4 Vanilla vs SSL results under data constraint 33
5.5 Test results on DukeMTMC 35
5.6 Evaluation on DroLAB 35
5.7 Evaluation on COMBI 37

1 Introduction

Unmanned aerial vehicles (UAVs), or drones, are useful platforms for academic research in a wide variety of fields, including control theory, air navigation, robotics, etc. The quadcopter model is by far the most popular drone variant among both researchers and hobbyists due to its simple mechanical design and dynamics, which lead to easy maintenance and maneuver. In recent years, many researchers have successfully "taught" drones to perform complex aerial moves and flight routines. While drones have a wide range of applications from military surveillance to commercial transport, consumer quadcopters are mainly used for aerial filming and photography. For both indoor and outdoor uses, person tracking is a highly desirable feature. An outdoor security drone needs to be able to follow a suspicious target, while an indoor assistant drone should be able to keep up with its master.

To follow a moving human, a drone has to be able to predict future trajectory of the target based on visual input. This makes heading direction estimation (HDE) an essential component of the path planning system. Accurate heading prediction will help the planner make better forecast of the target's trajectory, thus allowing a better optimized control signal with regards to possible changes in direction of the person being pursued.

HDE is itself a problem in other robotics subfields such as vehicle-pedestrian collision avoidance and human-robot interaction. Machine learning approaches like deep learning have showed good results on specific tasks. However, such approaches are heavily data-driven and rely on large amount of labeled data to achieve high prediction results. State-of-the-art models trained on one dataset tends to yield low performance when tested on another dataset due to mismatch between data distribution. This is true in particular for the task of aerial filming, where image quality can varies significantly with regards to human orientation, height and distance to camera.

Semi-supervised learning (SSL) is an active research area which focuses on using unlabeled data to improve learning accuracy, as opposed to solely labeled data. SSL makes use of some structure to the underlying data distribution to introduce certain constraints to a learning model's parameters. For visual input, temporal continuity is a valid assumption, as consecutive images in a sequence should produce close to continuous outputs.

In this thesis, we investigate the efficiency of different neural network models for the task of predicting human heading direction from images containing a single person's bounding box. The networks are trained and evaluated on open datasets. We then examine how well they generalize when tested on other datasets with limited amount of labeled samples. We show that by utilizing unlabeled data, the required amount of data can be reduced. We

also illustrate that temporal continuity can be used as unsupervised signal to regularize a model and achieve better generalization capability.

- Chapter 2 gives a short overview of the HDE problem, the interactive drone project at our lab as well as some common neural network architectures.
- Chapter 3 proposes three neural networks models. We show how a temporal continuity assumption can regularize a learning model by introducing a unsupervised loss to complement the usual supervised loss.
- Chapter 4 describes the experimental setup. It introduces two open datasets and one dataset created at our institute using a micro drone equipped with nano fisheye camera. This chapter also explains how the training data is generated and labeled.
- Chapter 5 shows and explains the results of the experiments.
- Chapter 6 discusses the results, suggests future works and improvements.

2 Background

2.1 Heading Direction Estimation

For many robotics applications such as pedestrian following, activity forecasting, etc., the planner subsystem needs to predict possible future trajectory of an moving object given visual cues and past knowledge. To do this, at each calculation step the planner needs to predict the most probable direction of the velocity vector of the object being tracked. This is referred to as the heading direction estimation (HDE) problem, which is a widely studied with emphasis on humans and cars. In this work we focus on the human HDE problem.

Let $[p_x, p_y, p_\alpha]$ be the pose of a target actor in the world frame. To estimate the orientation p_α, we need to calculate the heading direction α in the image coordinate system, projected on the ground plane. To avoid angle ambiguity and increase numerical stability, our HDE module predicts $[cos(\alpha), sin(\alpha)]$ and estimate α through these values. The world orientation p_α can be inferred using α and camera parameters.

Figure 2.1: Examples of HDE task. Given the bounding boxes of pedestrians, the predictor should output the direction of corresponding vectors.

2.1.1 Metrics

Let

- I be an input image
- $[cos(\alpha), sin(\alpha)]$ be the expected output
- $[cos(\beta), sin(\beta)]$ be the output of our predictor

We define:

- square error (SE) loss = $(cos(\alpha) - cos(\beta))^2 + (sin(\alpha) - sin(\beta))^2$
- AngleDiff = $|\alpha - \beta|$
- Acc = $\begin{cases} 1, & \text{if AngleDiff} \leq \pi/8 \\ 0, & \text{otherwise} \end{cases}$

2.2 Interactive Indoor Drone Assistant Project

2.2.1 Motivation

Thanks to recent advancements of UAV technology and control, consumer drones have become commonplace with simple flying maneuvers. Human-drone interaction thus becomes an increasingly promising research area. At our Interactive Systems Lab, we work towards a speaking assistant drone that

- is small and quiet
- can navigate and avoid obstacle in indoor environments
- understands and responds to voice control

Out first prototype [14] can receive voice commands to navigate between fixed points, given a schematic map of a corridor environment. It registers a 78% success rate after 27 flight missions.

2.2.2 Platform

2.2.2.1 Crazyflie 2.0

We use the Crazyflie 2.0 (CF2.0) by BitCraze, a open-source, customizable nano quadcopter which is popular for academic researches and experiments [16]. The drone can be controlled remotely from a computer using communication link provided by the Crazyradio PA [4], a 2.4GHz USB dongle. The funtionalities of the main board can be enhanced with extension boards. For our experiment, we utilized the Flow Deck [5] to keep the altitude stable once the drone reaches a desirable height.

2.2.2.2 Modifications

For our purposes, we made some customizations to the platform:

- Replacing the original 250mAh 1S LiPo battery with 2 300mAh 1S LiPo batteries
- Replacing manufacturer 12000Kv motors with 19000Kv motors from BetaFPV
- Attaching a first person view (FPV) nano camera with fisheye lens, which transmits on 5.8GHz band. The signals can be picked up using a 5.8GHz mobile receiver (see figure 2.3)

(a) Mounted Crazyflie 2.0 (b) Crazyradio PA (c) Flow deck

Figure 2.2: CF2.0 for autonomous hovering

(a) GOQOTOMO GD02 FPV camera (b) Happymodel VMR40 Receiver

Figure 2.3: FPV camera with receiver

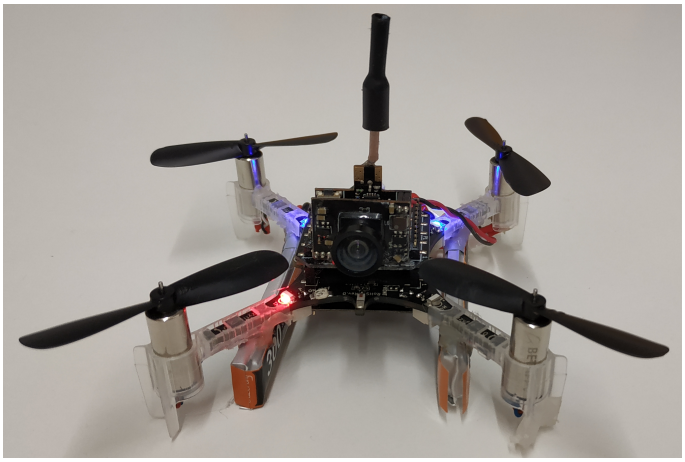


Figure 2.4: Our customized CF2.0 drone

2.2.3 Approach

Our system can recognize commands like “Fly to room 235” or “Fly to Stefan’s office”. An overview is given in figure 2.5.

The automatic speech recognition (ASR) component deciphers the verbal request and forwards the interpretation to the dialogue subsystem, where the target location is decoded. The navigation subsystem finds the closest fixed point in the schematic mapping corresponding to the real-world target. The flight controller then plans a collision-free path using the navigation subsystem and obstacle avoidance algorithm.

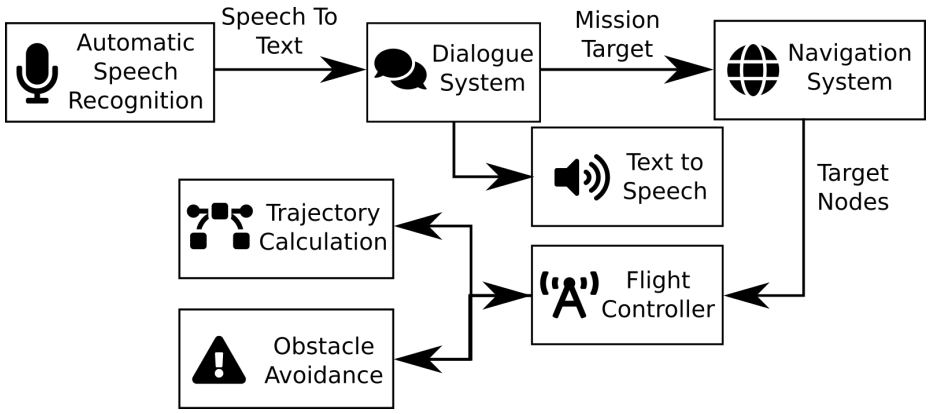


Figure 2.5: Indoor drone assistant system overview

2.2.3.1 Dialogue-based Mission Control

Our dialogue subsystem consists of three components:

- The ASR component transcribes the recorded audio into text. For this task we used the Janus speech recognition toolkit [31]
- The dialogue component extracts the semantic meaning from the transcription. It is based on the attention-based encoder-decoder model [2], using byte pair encoding [45].
- The text-to-speech (TTS) synthesizes an audio respond from the output of the dialogue component.

2.2.3.2 Localization and planning

With the Flow deck attached, the CF2.0 can send its position relative to a fixed starting point back to the control station. Navigation is performed by evaluating a schematic map of the laboratory (see figure 2.6), which contains a few important fixed points. We implemented Dijkstra’s algorithm [13] based on euclidean distance to compute the shortest sequence of points between the current drone position and the target location.

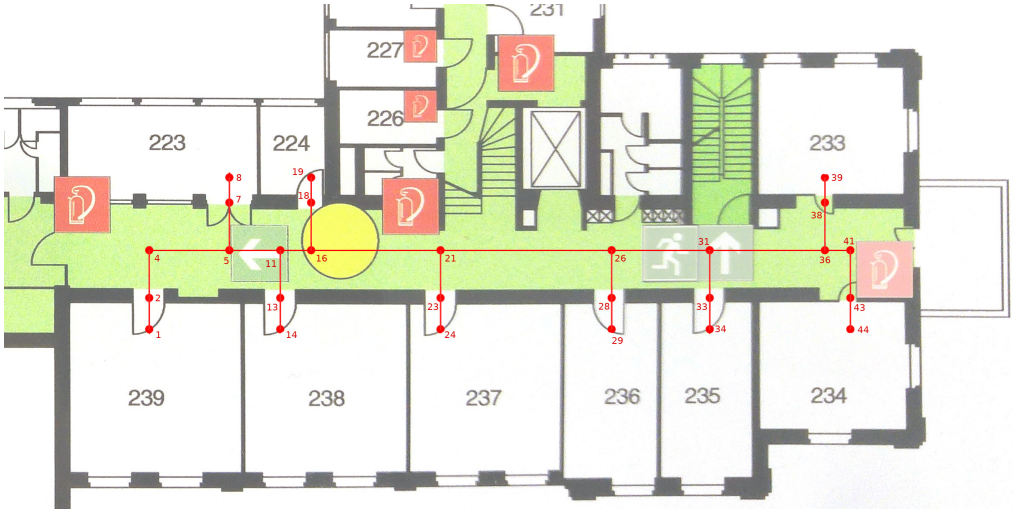


Figure 2.6: Schematics of our laboratory corridor. The fixed points are shown in red.

2.2.3.3 Obstacle avoidance

We employed a convolutional neural network (CNN) to compute a depth map from the RGB image obtained from the camera feed using the method proposed in [28]. A threshold function is then applied to the depth image to compute a binary image representing the free areas. Then a simple heuristic serves to determine in which direction the drone can fly. Figure 2.7 shows a sample of the camera image and corresponding depth map.

(a) Camera image (b) Depth map (c) Bitmap of obstacle-free areas

Figure 2.7: Output of the obstacle avoidance system

2.2.4 Future works

We plan to extend the functionalities of the system to

- Person recognition
- Person tracking and following
- Simultaneous localization and mapping (SLAM)

2.3 Neural Network

Artificial neural networks have been shown to be universal function approximators [10, 11, 22]. A neural network can be visualized as a directed graph and consists of:

- one input layer
- a number of hidden layers
- one output layer

Figure 2.8 shows a simple feed-forward network with 1 hidden layer, 3 input neurons, 4 hidden neurons and 2 output neurons [1].

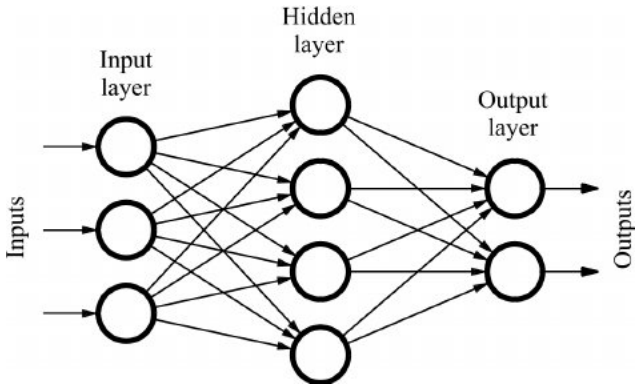


Figure 2.8: A simple feed-forward neural network

Let X^k, Y^k, W^k, σ^k be the input, output, weights and activation function and of layer k respectively. The values of the units in the next layer $k+1$ are computed as follows:

$$\begin{aligned} X^{k+1} &= W^k . Y^k \\ Y^{k+1} &= \sigma^{k+1}(X^{k+1}) \end{aligned} \tag{2.1}$$

with X^k, Y^k as vectors and . denotes a matrix multiplication.

2.3.1 Activation function

An activation function, or transfer function, is a nonlinear function that maps the input of an artificial neuron to its output. As of present, the most widely used function is the rectifier [39]:

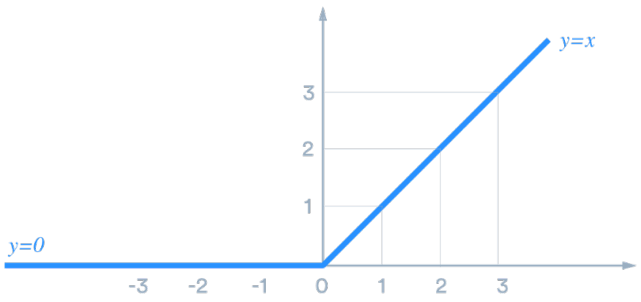
$$f(x) = max(0, x) \tag{2.2}$$

Another popular activation function is the hyperbolic tangent:

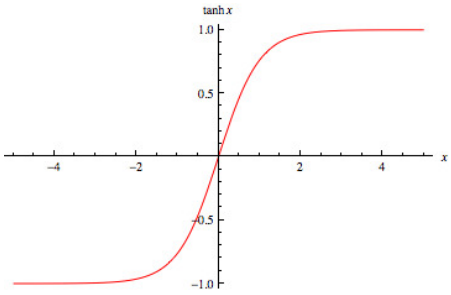
[1] Quiza and Davim, "Computational Methods and Optimization"

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.3}$$

Other popular activation functions include the logistic sigmoid, sinusoid, softmax, etc. Figure 2.9 illustrates the rectifier [2] and hyperbolic tangent [3] functions.



(a) ReLu



(b) tanh

Figure 2.9: Rectifier and hyperbolic tangent functions

2.3.2 Backpropagation algorithm

For supervised learning problems, training data consist of labeled input-output pairs [18]. Given input, the model learns to approximate the desired output by optimizing a loss function L. Hinton et al introduced an algorithm to tune the network weights based on gradient descent and the chain rule by backpropagating the training error E [43].

Let x be the input, f the mapping function represented by the network, y^* the desired output and w the current weights. The learning rate η is a predefined hyperparameter. Each training step works as follows:

- Forward pass:

$$\begin{aligned} Y &= f(x) \\ E &= L(y, y^*) \end{aligned} \tag{2.4}$$

[2] https://www.tinymind.com/learn/terms/relu
[3] http://mathworld.wolfram.com/HyperbolicTangent.html

- Backward pass:

$$w_{new} = w - \frac{\delta E}{\delta w} \tag{2.5}$$

To avoid overfitting, the model's performance should be measured on a validation dataset which is separated from the training dataset after a number of iterations or epoches.

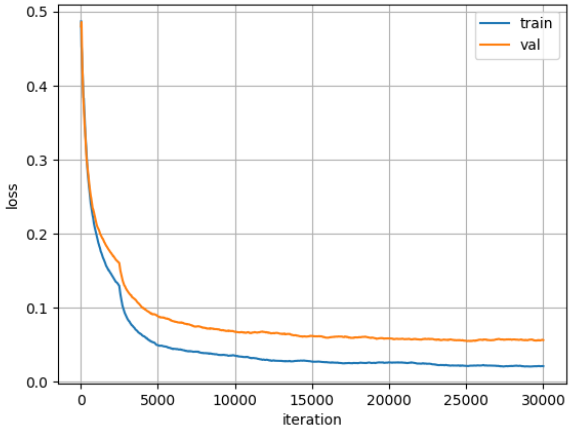


Figure 2.10: Training loss and validation loss converge after 30000 iterations

2.3.3 Batch normalization

Batch normalization [24] is a data normalization technique that has been shown to improve the performance and accelerate the training of neural networks [44, 26].
Let $B = (x_1, x_2, ..., x_m)$ a mini-batch of the training set. The mean and variance of B is:

$$\begin{aligned} \mu_B &= \frac{1}{m}\sum_{i=1}^{m} x_i \\ \sigma_B^2 &= \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \end{aligned} \tag{2.6}$$

Now for an input $x = (x^1, x^2, ..., x^d)$, each dimension of x is normalized separately:

$$\hat{x}_i^k = \frac{x_i^k - \mu_B^k}{\sqrt{{\sigma^k}^2{}_B + \epsilon}} \tag{2.7}$$

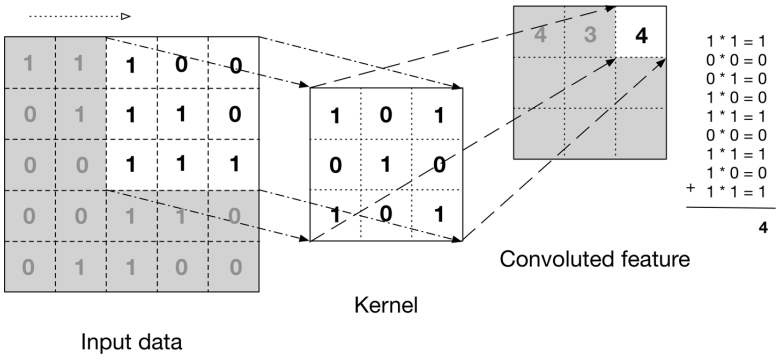
where $k \in [1, d]$ and $i \in [1, m]$, ϵ an arbitrarily small constant added in for numerical stability. The normalized $\hat{x}^k$ has zero mean and unit variance.

2.3.4 Time Delay Neural Network - Convolutional Neural Network

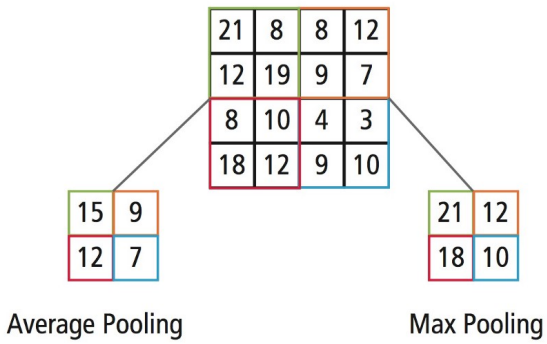
Time delay neural networks (TDNNs) [49] and Convolutional neural networks (CNNs) are neural network architectures which can learn translation invariance. Both are based on the neocognitron model [15]: each convolutional layer is followed by a downsampling (pooling) layer.

TDNN was the first architecture to achieve shift invariance by sharing weights along temporal dimension. This makes TDNNs suitable for speech processing and they achieve high performance when applied to far distance speech recognition tasks [25].

CNN on the other hand finds widespread use in computer vision applications. In image processing, to apply a filter on an image means to do a convolution between a kernel and the image. A pooling operation downsamples the image by replacing each subregion by an value, usually the maximum or average. Figure 2.11 gives an example of convolution [4] and pooling operations [5].



(a) Convolution



(b) Pooling

Figure 2.11: Convolution and pooling operation

Let **F** be an square input feature map of size $D_F \times D_F \times M$, where D_F is the spatial width and height, M the input depth (number of input channels). Applying a convolutional kernel **K**

[4] http://www.davidsbatista.net/blog/2018/03/31/SentenceClassificationConvNets
[5] https://medium.com/@Aj.Cheng/convolutional-neural-network-d9f69e473feb

of size $D_K \times D_K \times M \times N$ on **F** results in an output feature map **G** of size $D_G \times D_G \times N$ as follows:

$$G_{k,l,n} = \sum_{i,j,m} K_{i,j,m,n}.F_{k+i-1,l+j-1,m} \tag{2.8}$$

with $i, j \in [1, D_F]$, $k, l \in [1..D_G]$, $m \in [1..M]$, $n \in [1..N]$

LeCun et al. introduced a backpropagation method to learn the kernel coefficients [29] and the first CNN to classify hand-written digits [30]. Some prominent CNN architectures that achieved state-of-the-art results in the ImageNet visual recognition challenge [12] are: VGGNet [33], AlexNet[27], Google Inception[48], ResNet [17].

2.3.5 Recurrent Neural Network

Recurrent neural networks (RNNs) are neural networks with internal memory capability which allows them to learn temporal dynamic behaviour from sequential input. This make RNNs ideal for tasks like speech recognition and time-series forecasting.

For a simple RNN unit, the memory capability is realized by a feedback loop from the hidden state to itself, as shown in figure 2.12. At time step t, let x_t be the input vector and s_{t-1} the previous hidden state. Then the new hidden state s_t and output y_t are computed as follows:

$$\begin{aligned} s_t &= \phi(W.x_t + U.s_{t-1}) \\ y_t &= \sigma(V.s_t) \end{aligned} \tag{2.9}$$

where ϕ and σ are activation functions, U, W, V are weight matrices.

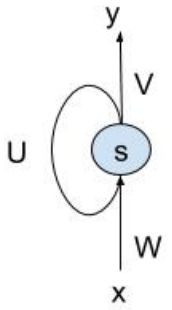


Figure 2.12: RNN loop

2.3.5.1 Backpropagation through time

Backpropagation through time (BPTT) is an extension of the backpropagation algorithm described in section 2.3.2 used to train RNNs [52, 41, 36].

To perform a backward pass over n input-output pairs $(x_0, y_0), (x_1, y_1), (x_2, y_2), .., (x_{n-1}, y_{n-1})$, the network is unfolded into n connected feed-forward components. The backpropagation algorithm is applied on the unfolded network. The forward pass computes the errors term $E_0, E_1, ..., E_{n-1}$. The error derivatives are backpropagated in reverse order, starting from E_{n-1} downto E_0. A schema for $n = 4$ is shown in figure 2.13 [6].

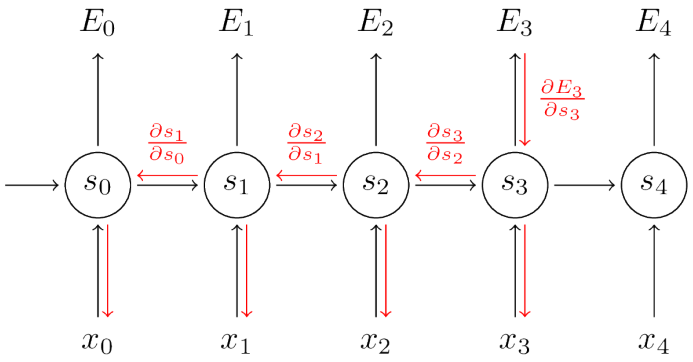


Figure 2.13

Since the weight matrices U, V, W are shared across all recurrent units, the weight derivatives are summed up and averaged by n. For large n, the gradients at earlier step decay due to long backward pass. This lead to influence of the front part of the input sequences being diminished in the training process, which is known as the vanishing gradient problem [21]. This implies that more complex memory units are needed to model long-term dependencies.

2.3.5.2 Gated Recurrent Unit

Gated recurrent unit (GRU) [8] is a gating mechanism in RNN that aims to mitigate the vanishing gradient problem. GRU can be seen as a lighter version of the powerful but more computational expensive Long-short term memory (LSTM) architecture [20].

A GRU cell contains an update gate $\mathbf{z}$ and reset gate $\mathbf{r}$, each with its own weight matrix. At time step t, the hidden state h_t is computed through current cell state $\tilde{h}_t$, input x_t and the gate values as follows:

[6] http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients

$$
\begin{aligned}
z_t &= \sigma(W_z.[h_{t-1}, x_t]) \\
r_t &= \sigma(W_r.[h_{t-1}, x_t]) \\
\tilde{h}_t &= tanh(W.[r_t * h_{t-1}, x_t]) \\
h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t
\end{aligned} \tag{2.10}
$$

where σ denotes the rectifier function, $*$ the element-wise multiplication, [,] the vector concatenation. Figure 2.14 shows a simple GRU cell [7].

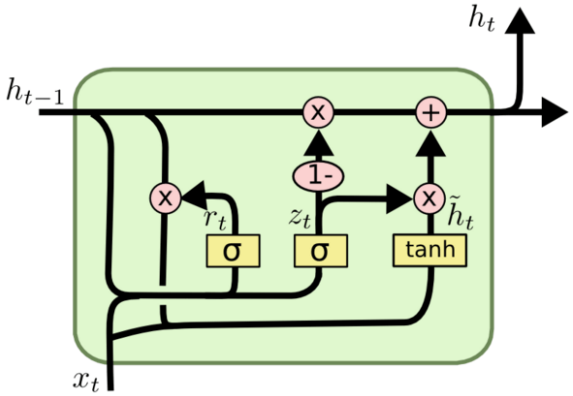


Figure 2.14: GRU cell

2.3.6 Semi-supervised Learning

SSL is a class of machine learning methods that utilizes unlabeled data, in conjunction with a limited amount of labeled data, to improve learning accuracy. It creates a bridge beetween supervised learning (all data labeled) [18] and unsupervised learning (no labeled data) [19]. SSL algorithms make use of one of the following assumptions [7]:

- Continuity: *Points which are close to each other are more likely to share a label*
- Cluster: *The data tend to form discrete clusters, and points in the same cluster are more likely to share a label*
- Manifold: *The data lie approximately on a manifold of much lower dimension than the input space*

For image and video data, **temporal continuity** is a valid assumption. Mohabi et al. [35] used temporal continuity as supervisory signal over unlabeled video data to boost object recognition performance. Temporal continuity can also be exploited to learn feature representations [47, 51].

[7] `https://www.data-blogger.com/2017/08/27/gru-implementation-tensorflow`

3 Approach

All our neural networks consist of 2 main components:

- the Feature Extractor takes as input an RGB image of size $192x192$ and outputs a fixed size 256-dimensional feature vector
- the Estimator network transforms the resulting feature vector into an angle prediction $[sin(\alpha), cos(\alpha)]$

We experimented with 2 CNN architectures for the extractor component: the base extractor (section 3.2) and the Mobile extractor (section 3.3). For the estimator network, we constructed 3 different models: a vanilla model (section 3.4), a RNN-based model (section 3.5) and a SSL-based model (section 3.6).

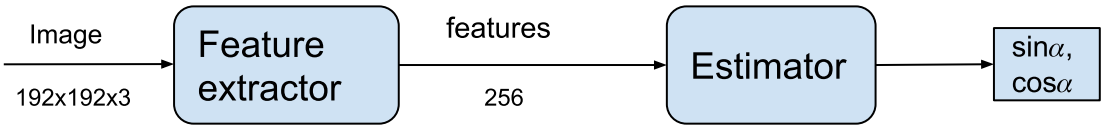


Figure 3.1: Predictor model overview

3.1 Supervised Loss

For our HDE supervised learning regression task, we used the MSE loss (also known as $L2$ loss).
Suppose we have labeled dataset $D = \{(x_1, y_1), ..., (x_m, y_m)\}$ where x_i denotes an input image and $y_i = [sin(\alpha_i), cos(\alpha_i)]$ the ground truth angle. Our predictor model is $\mathbf{M}$ and $\mathbf{f}$ is the approximation function represented by $\mathbf{M}$. The $L2$ loss is defined as

$$L = L_s = \frac{1}{2m} \sum_{i=1}^{m} ||f(x_i) - y_i||^2 \tag{3.1}$$

The goal is to find the model parameter set Θ which minimizes this loss function. For this task we used the backpropagation algorithm (see section 2.3.2).

3.2 Base Extractor

Our base extractor is a deep CNN with 5 convolution layers, each followed by ReLu activation (see figure 3.3. This is inspired by the work of Choi et al.[9].

Figure 3.2 visualizes a standard convolution layer[1]. The full details of all extractor layers are given in table 3.1

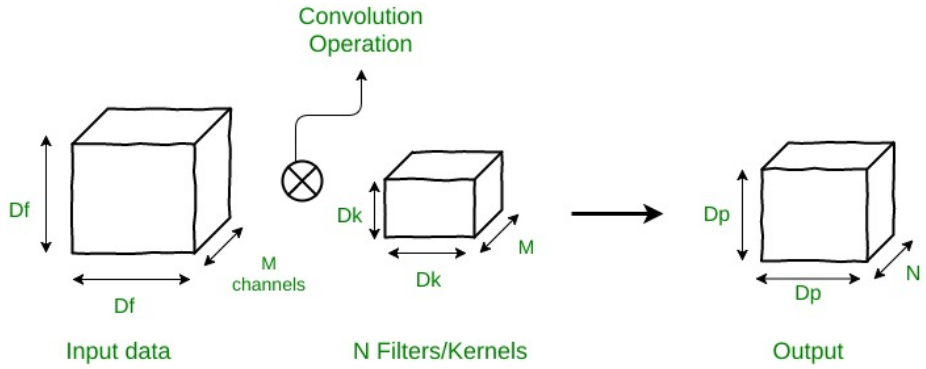


Figure 3.2: Standard convolution layer

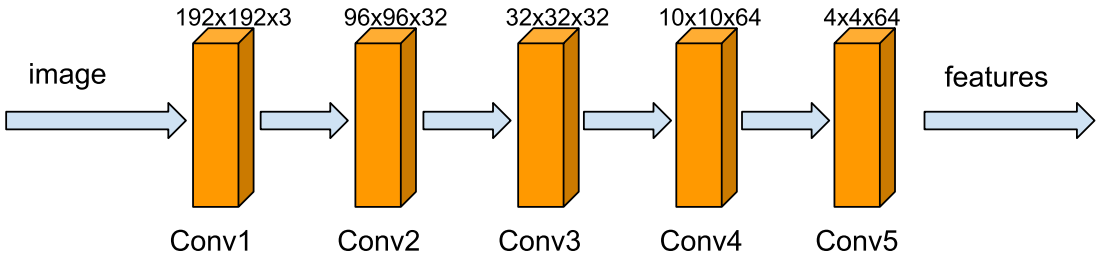


Figure 3.3: Base extractor architecture

Layer	Input size	Kernel shape	Stride	Padding
Conv1	192 x 192 x 3	3 x 3 x 3 x 32	2	1
Conv2	96 x 96 x 32	5 x 5 x 32 x 32	3	1
Conv3	32 x 32 x 32	5 x 5 x 32 x 64	3	0
Conv4	10 x 10 x 64	3 x 3 x 64 x 64	3	1
Conv5	4 x 4 x 64	4 x 4 x 64 x 256	1	0
Output	1 x 1 x 256			

Table 3.1: Base extractor body architecture

[1] https://www.geeksforgeeks.org/depth-wise-separable-convolutional-neural-networks

3.3 Mobile Extractor

3.3.1 Depthwise separable Convolution

The depthwise separable convolution was introduced along with the MobileNet architecture[23]. Let F be an input feature map of size $D_F \times D_F \times M$ and K a kernel of size $D_K \times D_K \times M \times N$. A depthwise separable convolution factorizes a standard convolution with kernel K into a depthwise convolution with kernel $\hat{K}$ of size $D_K \times D_K \times M$, followed by a pointwise convolution with kernel $\tilde{K}$ of size $1 \times 1 \times M \times N$.

Depthwise convolution applies a single filter per input channel. Thus, the output feature map has the same depth as the input feature map and can be written as:

$$\hat{G}_{k,l,m} = \sum_{i,j} \hat{K}_{i,j,m}.F_{k+i-1,l+j-1,m} \tag{3.2}$$

A pointwise convolution is a standard convolution with filters of size 1×1, so in essence each layer of the output feature map is a linear combination of the input feature map layers. The output of the pointwise convolution following the depthwise convolution is:

$$\tilde{G}_{k,l,m} = \sum_{n} \tilde{K}_{1,1,m}.\hat{G}_{k,l,m} \tag{3.3}$$

The total computational cost of a depthwise separable convolution is calculated from equations 3.2 and 3.3:

$$D_K.D_K.M.D_F.D_F + M.N.D_F.D_F \tag{3.4}$$

this is considerably less the cost of a standard convolution based on equation 2.8:

$$D_K.D_K.M.N.D_F.D_F \tag{3.5}$$

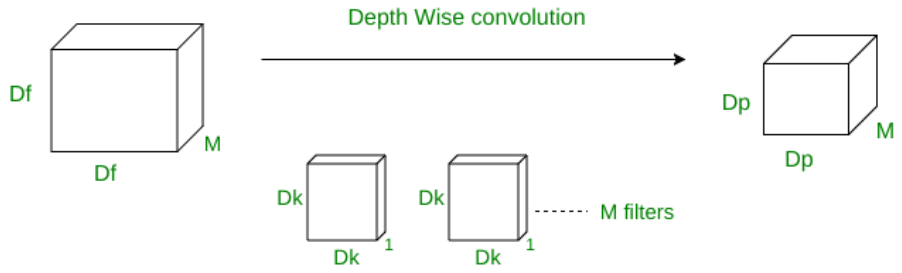
Figure 3.4 visualizes the depthwise and pointwise convolution layers [1].

3.3.2 Architecture

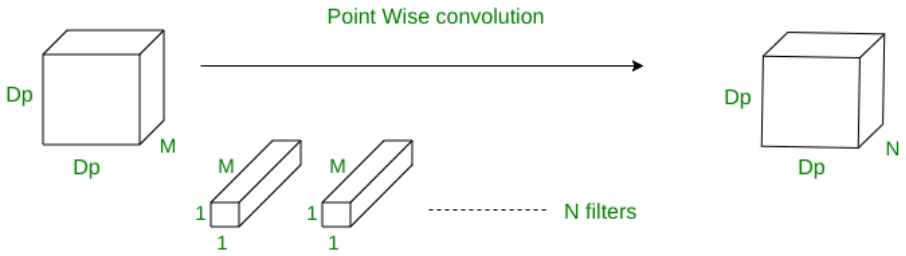
Our mobile extractor design is based on the MobileNet architecture [23]. The first layer is a standard convolution, which is followed by 11 depthwise separable convolutions. For the first 12 layers, convolution is followed by batch normalization ($\epsilon = 0.001$) and ReLU activation. Standard and depthwise convolutions use 3×3 filter with padding size 1.The depthwise stride interchanges between 1 and 2. All other convolutions have stride 1. A final 3×3 convolution layer outputs $1 \times 1 \times 256$ feature map without normalization and activation.
Figure 3.5 shows a sketch of the architecture. Full details of all layers are given in table 3.2.

3.4 Vanilla Estimator

We designed a simple feedforward neural network with 1 a single fully connected (FC) hidden layer and ReLU activation, inspired by orientation classification network proposed by Choi et al.[9].

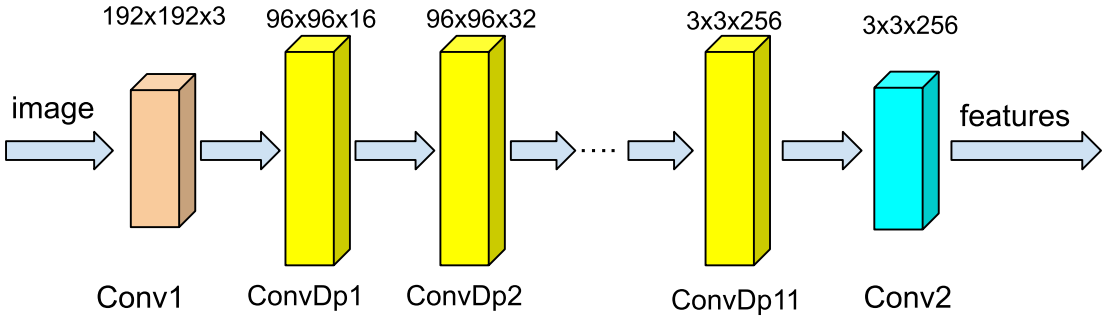


(a) Depthwise convolution

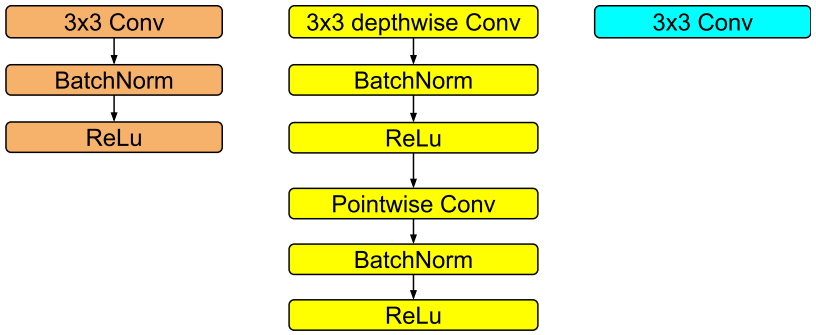


(b) Pointwise convolution

Figure 3.4: Depthwise separable convolution layer



(a) Data flow



(b) Left: standard convolution (Conv) with BatchNorm and ReLU (layer 1). Middle: depthwise separable convolution (ConvDps) withBatchNorm and ReLU (layers 2-12). Right: 3x3 convolution (layer 13)

Figure 3.5: Mobile extractor architecture

Layer	Input size	Kernel depth	Stride
Conv1	192 x 192 x 3	3 x 16	2
ConvDps1	96 x 96 x 16	16 x 32	1
ConvDps2	96 x 96 x 32	32 x 64	2
ConvDps3	48 x 48 x 64	64 x 64	1
ConvDps4	48 x 48 x 64	64 x 128	2
ConvDps5	24 x 24 x 128	128 x 128	1
ConvDps6	24 x 24 x 128	128 x 256	2
ConvDps7	12 x 12 x 256	256 x 256	1
ConvDps8	12 x 12 x 256	256 x 256	2
ConvDps9	6 x 6 x 256	256 x 256	1
ConvDps10	6 x 6 x 256	256 x 256	2
ConvDps11	3 x 3 x 256	256 x 256	1
Conv2	3 x 3 x 256	256 x 256	1
Output	1 x 1 x 256		

Table 3.2: Mobil extractor body architecture

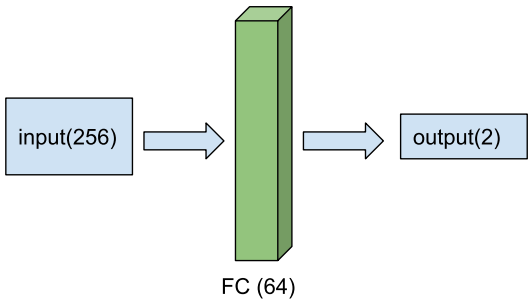


Figure 3.6: Vanilla estimator

3.5 RNN Estimator

For our RNN estimator model, we exploited the possible sequential dependecy of features. The 256d input feature vector F is splitted into four 64d subfeature vectors f_1, f_2, f_3, f_4. We used a single GRU cell S with 256 hidden units as memory storage. Let h_t denotes the memory content (values of hidden unit) and S^t the GRU cell at time t. We set $h_0 = 0$ by default. The last fully connected layer maps the 256d memory vector h_4 to 2d output. Figure 3.7 explains the data flow.

3.6 Semi-supervised learning Predictor

For this section we implemented the approach proposed by Wenshan et al [50].

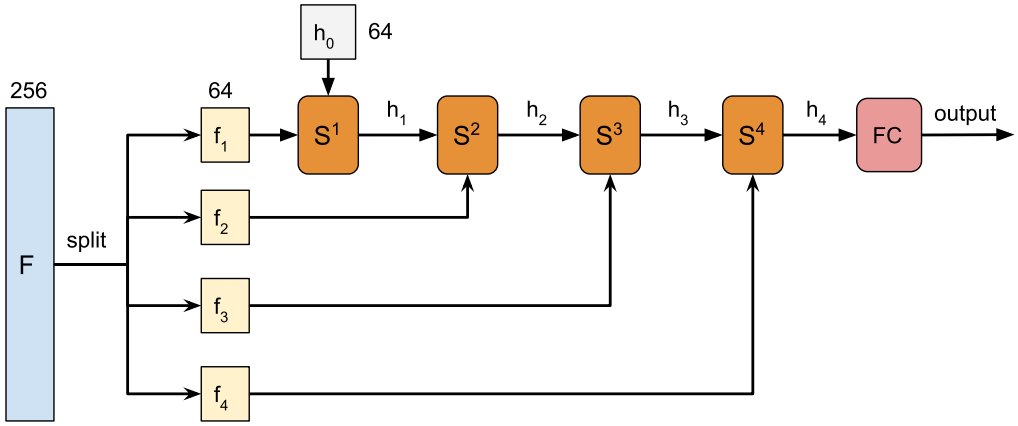


Figure 3.7: RNN estimator

3.6.1 Unsupervised loss function for temporal continuity

In order to exploit temporal continuity to improve prediction accuracy, we need to define a **unsupervised loss** L_u for unlabeled data. Let $(x_1, x_2, ..., x_n)$ be an image sequence of an unique actor in chronological order. Then $Q = \{x_1, x_2, ..., x_n\}$ is an unlabeled image set. We make use of the temporal continuity assumption (section 2.3.6): if two frames are close regarding temporal aspect, the actor's heading direction predictions should also be close. The unsupervised loss can be generalized as:

$$L_u = \sum_{x_1, x_2 \in Q} \tau(x_1, x_2)\Omega(x_1, x_2, f) \tag{3.6}$$

where f is the mapping function of model M, $\tau(x_1, x_2)$ is the similarity of two samples and $\Omega(x_1, x_2, f)$ is the difference between their outputs regarding f and can be defined as half the square difference:

$$\Omega(x_1, x_2, f) = \frac{1}{2}(f(x_1) - f(x_2))^2 \tag{3.7}$$

We assume that $\tau(x_1, x_2)$ is inverse proportional to the temporal displacement between x_1 and x_2. This implies that if $x_1 < x_2 < x_3$ then $\tau(x_1, x_2) > \tau(x_1, x_3)$. Equation 3.6 suggests that $\Omega(x_1, x_2, f) < \Omega(x_1, x_3, f)$ to keep L_u low. If this is not the case, the model should be punished by an error amount equal to $E = \Omega(x_1, x_2, f) - \Omega(x_1, x_3, f)$
We can now formulate L_u as:

$$L_u = \sum_{\substack{x_1, x_2, x_3 \in Q \\ x_1 < x_2 < x_3}} max(0, \Omega(x_1, x_2, f) - \Omega(x_1, x_3, f) - \epsilon) \tag{3.8}$$

where ϵ is an arbitrarily small threshold. For our experiment, we chose $\epsilon = 0.005$.

Since Q contains n samples, the computational complexity of equation 3.8 is $O(n^3)$ which would slow down training considerably. To solve this, we can generate randomly a set of triplets $U = \{(i_1, j_1, k_1), ..., (i_p, j_p, k_p)\}$ where $i_r < j_r < k_r$ and $i_r, j_r, k_r \in [1..n]$. The mean value of L_u can be computed stochastically on U:

$$L_u = \frac{1}{r} \sum_{(i,j,k) \in U} max(0, \Omega(x_i, x_j, f) - \Omega(x_i, x_k, f) - \epsilon) \tag{3.9}$$

3.6.2 Combined loss function

For the SSL estimator network, we used the same architecture as the vanilla model in section 3.4.

Each training batch B consists of a labeled images batch B_l and a sequence of unlabeled images from one unique actor B_u. Thus we have $B = (B_l, B_u)$. The combined loss L is made up from the supervised loss (3.1) and unsupervised loss (3.8):

$$L(B) = L(B_l, B_u) = L_s(B_l) + \lambda . L_u(B_u) \tag{3.10}$$

where λ is a hyperparameter which serves to balance the supervised and unsupervised losses. The selection of λ plays an important role in model performance, which we discuss in section 5.1.1

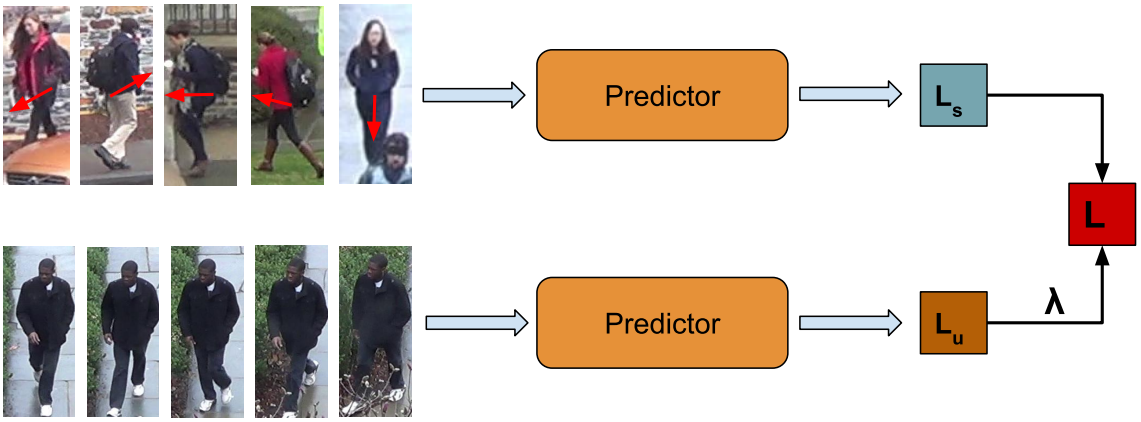


Figure 3.8: Combined loss calculation

3.7 Data Augmentation

For data-driven machine learning approaches like neural networks, models tend to produce better results with larger amount of training data available. Data augmentation means to artificially generate more training data from existing data. This is done by applying domain-specific techniques to training samples in order to create different but plausible

new samples.

In image processing, one common augmentation technique is to add a small amount of noise and distortion to images which result in slightly modified versions of themselves. This is particularly helpful when the size of training set is limited. For our approach, we applied the following popular augmentation methods in the neural network training process:

- RandomCrop: the image's top and bottom are randomly cut off up to 20% of its height. Its left and right sides are also randomly cut off up to 20% the width.
- Random Hue, Saturation, Value: to each of the H,S,V channel, a value is randomly chosen from a fixed interval and added to the current value. The interval is fixed to $[-10, 10]$ for the H channel and $[-60, 60]$ for the S and V channels.
- RandomMirror: an image is flipped left-right with 50% probability. If a labeled image is flipped, the correspondind ground truth angle is also corrected accordingly.

Figure 3.9 compares a sequence in the DukeMTMC dataset before and after augmentation is applied.

(a) no augmentation

(b) with augmentation

Figure 3.9: A DukeMTMC sample sequence before and after data augmentation. Corresponding images are aligned vertically.

4 Experiments

In this section we discuss 3 datasets and how we generated network input from them.

4.1 Datasets

4.1.1 DukeMTMC

The DukeMTMC [40] is a large multi-target tracking dataset, consisting of 1080p videos recorded at 60FPS from 8 static cameras on the Duke University campus during daytime. It recorded more than 2 million frames and 2700 identities. All trajectories were manually annotated. Figure 4.1 shows 2 example frames.

We used videos from cameras 1, 3, 4, 7, 8 to generate training data, camera 5 for validation and camera 2 for testing.

(a) Camera 1 (b) Camera 5

Figure 4.1: Frames extracted from DukeMTMC videos

4.1.1.1 Data generation

The algorithm to generate ground truth of heading direction was provided by the courtesy of Dr. Wenshan Wang[1]. We downsample all videos from 60FPS to 5FPS. Each tracked person **P** is assigned an unique identifier I_P. Suppose person **P** appears in a video **V**. We define the position of **P** to be the center of two feet. The dataset annotations for **P** are given as a sequence of tuple in the form $(F_i, B_i, w_{xi}, w_{y_i}, f_{xi}, f_{y_i})$ where F denotes the frame number, B the bounding box, w_x, w_y the position in camera coordinate system and f_x, f_y the pixel coordinate.

[1]Wenshan Wang, AIR Lab, The Robotics Institute, Canergie Mellon University

We discard frame F_i if the square position displacement to the previous frame F_{i-1} is smaller than a threshold δ, i.e

$$(w_{xi} - w_{xi-1})^2 + (w_{y_i} - w_{y_{i-1}})^2 < \delta$$

If F_i is not discarded, we extract the subimage defined by B_i and assign the angle α closed by the vector spans from $(f_{x_{i-1}}, f_{y_{i-1}})$ to (f_{x_i}, f_{y_i}) and the x-axis as the heading direction:

$$\alpha = arctan(\frac{f_{y_i} - f_{y_{i-1}}}{f_{x_i} - f_{x_{i-1}}})$$

Figure 4.2: Sample sequence of DukeMTMC dataset

4.1.2 DroLAB

We created an indoor dataset by using our customized CF2.0 to follow and film moving actors at various locations inside building 50.20 of the Karlsruhe Institute of Technology. The drone is flown manually using Bitcraze Python client [2]. This guarantees a high level of diversity within the dataset with regards to background settings, actor's scale to image size, the camera's height and orientation. To further increase dataset variance we also had actors put on different outfits. Figure 4.3 shows some examples of the raw image data.
In total we filmed 86 videos, divided into 10 subsets, each filmed with one unique actor. We use 4 subsets as training data, 3 subsets for validation and 3 subsets for testing purpose.

4.1.2.1 Data generation

To extract the actor's bounding box from raw images, we used Single shot multibox detector (SSD) method [34] with MobileNet as feature extractor [23]. We used the implementation by Adrian Rosenbrock [42]. We sampled all videos with 5FPS frequency and set the confidence threshold to 0.8 for the task of person recognition. See Figure 4.3 for example of actor bounding box extraction.
Labeled data were created by classifying the heading direction of the actor in bounding box as a cardinal direction (north, south, east, west) or intercardinal direction (northeast, northwest, southeast, southwest). The angle and sin, cos values were assigned to the selected samples according to Table 4.1. We manually labeled 2000 samples for training, 500 for validation and 500 for testing.

[2] https://github.com/bitcraze/crazyflie-lib-python

Figure 4.3: Images extracted from videos recorded for DroLAB dataset. Each actor is enclosed in a light green rectangular bounding box.

Figure 4.4: Sample sequence of an actor walking in an office. The images were resized for better visualization due to large difference of bounding box size according to the actor's proximity to the camera.

4.1.3 COMBI

We created a new dataset from multiple subdatasets to measure the generalizability of models trained on the DukeMTMC and DroLAB datasets, respectively. We named this dataset COMBI. Since this dataset serves only testing purposes, it is not splitted into training/validation/test subsets.

Direction	Angle(α)	$cos(\alpha)$	$sin(\alpha)$
E	0	1	0
NE	$\pi/4$	0.7071	0.7071
N	$\pi/2$	0	1
NW	$3\pi/4$	-0.7071	0.7071
W	π	-1	0
SW	$-3\pi/4$	-0.7071	-0.7071
S	$\pi/2$	0	-1
SE	$-\pi/4$	0.7071	-0.7071

Table 4.1: Ground truth for manual labelling

The subdatasets are as follows:

- 3DPeS [3]: dataset for people re-identification, created by using 6 different surveillance cameras to monitor a section of the campus of the University of Modena and Reggio Emilia.
- Microsoft-COCO [32]: a large-scale object detection, segmentation, and captioning dataset. This is a popular benchmark for machine learning models.
- UCF101 [46]: contains 13000 videos of human actions in the wild.
- CMU-Drone: created as part of the drone autonomous cinematography project at the CMU [6, 50]. Human actors were filmed by flying a DJI M210 quadcopter outdoor. This dataset was provided by Dr. Wenshan Wang.

For the MS-COCO and UCF101 datasets, bounding boxes of actors were obtained using the method described in section 4.1.2. The authors of the 3DPeS dataset provided a number of person bounding boxes. From each subdataset, a small number of samples are randomly selected and manually labeled. For the exact composition of COMBI, see table 4.2.

Subdataset	Number of samples
3DPeS	1618
MS-COCO	524
UCF101	1324
CMU-Drone	593

Table 4.2: COMBI dataset composition

4.2 Implementation

Our neural networks are implemented in Python3.6 using the Pytorch machine learning library [38]. We used Anaconda to manage Python packages and distribution [1]. To

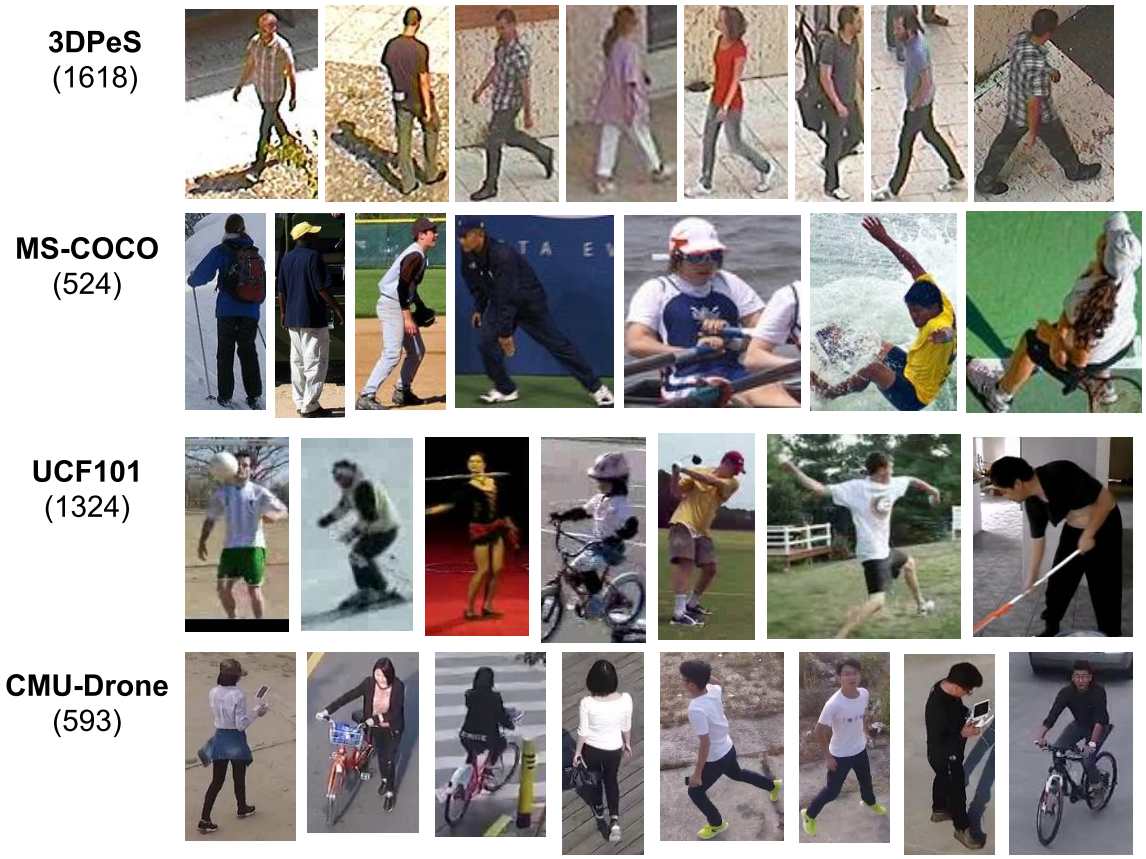


Figure 4.5: Samples of COMBI subdatasets. The images were resized for better visualization due to having different scales.

speedup training, we trained all NNs on our institute's Nvidia Tesla K20 GPU cluster using Nvidia CUDA Toolkit 8.0 [37].

5 Evaluation

The following experiments show how the models proposed in chapter 3 perform on the datasets described in chapter 4, as well as the impact of their feature extractor and angle estimator components. Model performance is evaluated using the 3 metrics introduced in section 2.1.1.

For training phase, data are augmented as described in section 3.7. Model parameters are saved in checkpoint after a fix number of iterations. We selected the final parameters and hyperparameters which result in the lowest supervised loss measured on the validation set. Final results are reported on the test set.

For all models, labeled batch size is fixed at 128. For SSL models, we used unlabeled batch size (unlabeled image sequence length) 48 on the DukeMTMC dataset and 32 on the DroLAB dataset.

Table 5.1 gives an overview of each dataset's size and training/validation/test split.

Dataset	Training	Validation	Test	Total
DukeMTMC	305943	42557	90294	438794
DukeMTMC (unlabeled)	546668	56914	115247	718829
DroLAB	2000	500	500	3000
DroLAB (unlabeled)	12138	3166	5426	20550
COMBI			4059	4059

Table 5.1: Datasets statistics

5.1 Results on DukeMTMC

Due to the fact that DukeMTMC is our largest and most intensively labeled dataset, we use AngleDiff as the main evaluation metric.

5.1.1 Selection of λ

For SSl models, the hyperparameter λ balances the supervised and unsupervised losses, as stated in equation 3.10:

$$L(B) = L(B_l, B_u) = L_s(B_l) + \lambda . L_u(B_u)$$

If λ is set too high, the model will prioritize minimizing L_u by generating trivial unchanged outputs for the unlabeled batch B_u. If λ is too low, the potential gain by using unlabeled samples is diminished.

We selected an optimal value by trying out different values of λ in range $[0.001, 1.5]$. We trained SSL predictor using Mobile feature extractor through at least 20000 iterations. Results on the DukeMTMC validation set are reported in table 5.2.

λ	MSE	AngleDiff (rad)	Accuracy (%)
0.001	0.042	0.208	88.94
0.005	**0.039**	**0.196**	**89.81**
0.01	**0.038**	**0.199**	**89.62**
0.05	0.049	0.208	89.01
0.1	0.119	0.238	84.60
0.5	0.055	0.204	89.37
0.9	**0.038**	**0.197**	**89.71**
1.2	0.054	0.242	84.28
1.5	0.061	0.237	85.39

Table 5.2: Validation results for λ. Top 3 models have $\lambda \in \{0.005, 0.01, 0.9\}$, $\lambda = 0.005$ yields best score in angle difference and accuracy and second best MSE loss

Based on results showed in table 5.2, we use $\lambda = 0.005$ in all SSL models in later sections.

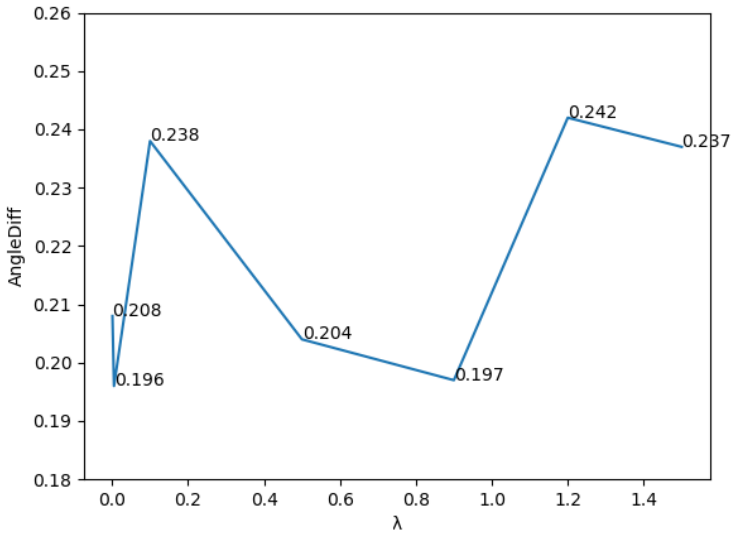


Figure 5.1: Angle error for different values of λ

Figure 5.2 shows the supervised vs unsupervised error graph with $\lambda = 1.5$.

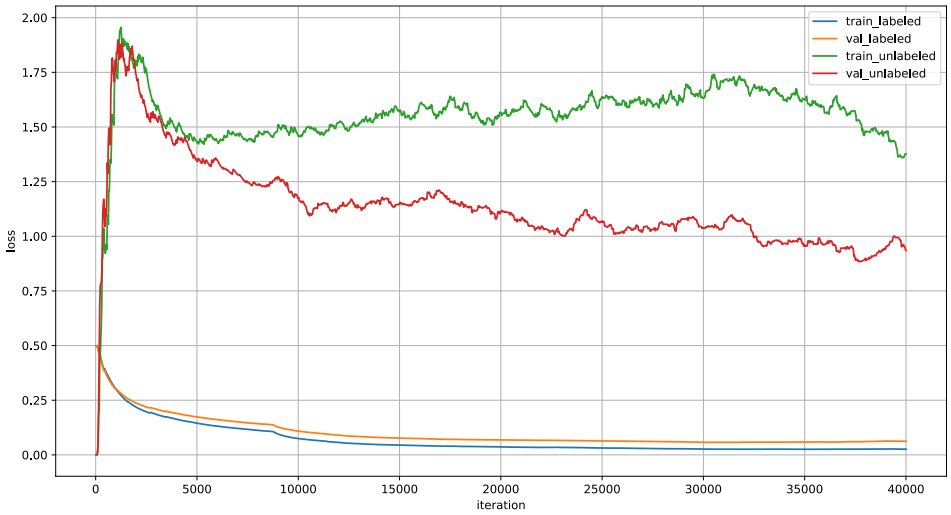


Figure 5.2: $\lambda = 1.5$. The convergence of training and validation supervised loss (train_labeled, val_labeled) and generally falling trend of unsupervised loss (train_unlabeled, val_unlabeled) suggests that the model learned to predict accurate heading direction as well as learned about temporal continuity restriction at the same time.

5.1.2 Extractor influence

We compared the effectiveness two extractor architectures on the overall predictor model performance in order to decide on a suitable design for the HDE task. We trained and evaluated all three estimator models in combination with two feature extractor models. Full results are obtained on validation set and shown in table 5.3.

For all models, the mobile extractor outperformed the base extractor by a noticeable margin on every metric. We hypothesize that this comes from 2 factors: higher network depth and use of batch normalization.

The vanilla model achieved best results compared to the other two using the base extractor with 0.218 angle error score and 87.11% accuracy. Upgrading to the more powerful mobile extractor, the RNN model received highest performance boost and became the best performing model: angle error dropped from 0.250 downto 0.191 while accuracy increased from 83.98% to 90.43%. We believe that higher grained features extracted by the mobile component allows the RNN model to make use of the more complex estimator subnetwork, in particular the recurrent connections and memory content of the GRU cell.

While the SSL estimator ranks behind the vanilla design when combined with base extractor, it outperforms the latter slightly when mobile extractor is put to use.

A side by side comparison of the impact of feature extractor designs on angle prediction accuracy is presented in figure 5.3.

Extractor	Model	MSE	AngleDiff (rad)	Accuracy (%)
Base	Vanilla	**0.047**	**0.218**	**87.11**
	RNN	0.057	0.250	83.98
	SSL	0.051	0.237	84.74
Mobile	Vanilla	0.038	0.200	89.57
	RNN	**0.036**	**0.191**	**90.43**
	SSL	0.039	0.196	89.81

Table 5.3: Extractor-Estimator validation results on DukeMTMC

.

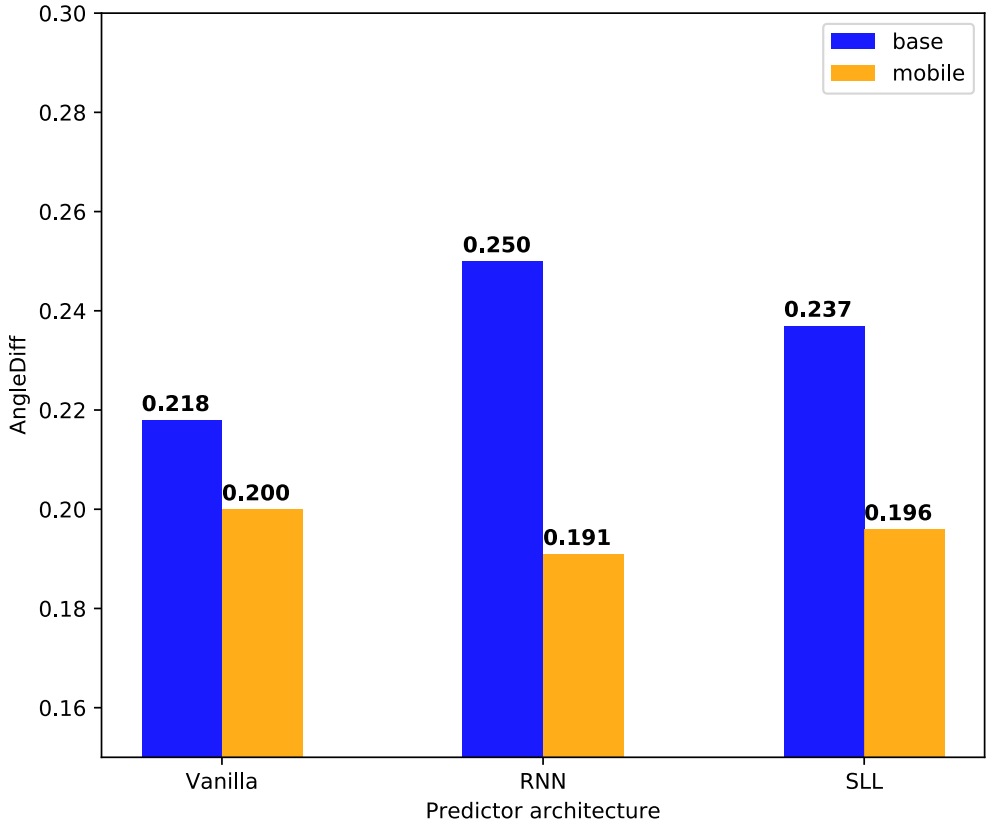


Figure 5.3: Angle error comparison between base and mobile extractors. RNN and SSL models experience higher gains from mobile extractor component compared to Vanilla model.

Based on the results in this section, we set mobile extractor as the default design for the following sections.

5.1.3 Semi-supervised learning with data constraint

We investigated the advantage of semi-supervised over supervised approach under various conditions of labeled data.

We trained Vanilla and SSL predictor using 1%, 5%, 20% and 100% labeled samples in the training dataset, respectively. All models were evaluated on test set. Evaluation results are shown in table 5.3. Figure 5.4 compare the test angle error score between 2 approaches.

Extractor	Data (%)	MSE	AngleDiff (rad)	Accuracy (%)
Vanilla	1	0.152	0.417	69.89
	5	0.100	0.323	76.90
	20	0.078	0.281	78.49
	100	**0.069**	**0.266**	**81.25**
SSL	1	0.128	0.387	70.15
	5	0.099	0.325	75.09
	20	0.076	0.277	79.34
	100	**0.069**	**0.263**	**79.60**

Table 5.4: Results on test set of Vanilla and SSL predictors trained on different scales of labeled training data

.

For both type of models, testing scores increase with amount of labeled data available. This confirms the general belief that more training data result in better learning models. At 1% labeled data (approximately 3060 samples), the SSL model outperforms the Vanilla model in all three metrics: MSE by 0.024, AngleDiff by 0.03 and Accuracy by 0.25%. We argue that this is a result of the SSL approach making use of the abundance on unlabeled data in the absence of labeled data. This gain diminishes and flattens out as the amount of labeled data increases. At 5% or more, both models yield comparable testing results. This can be explained by equation 3.10 and our choice of $\lambda = 0.005$: the combined loss is dominated by the term supervised term. We conclude that though unsupervised learning brings new domain knowledge to our model, the main determinator is the supervised part.

5.1.4 Test results

We evaluated 3 models on the test set (see table 5.5). The RNN predictor lags behind the other two in terms of performance. While the SSL model scores slightly better than the Vanilla model in angle error (0.263 to 0.266), it is surpassed by 1.65% in accuracy. This implies that simple supervised approach is suitable for the HDE problem on the DukeMTMC datasets. We believe therefore that selection of a suitable network model is a dataset specific task.

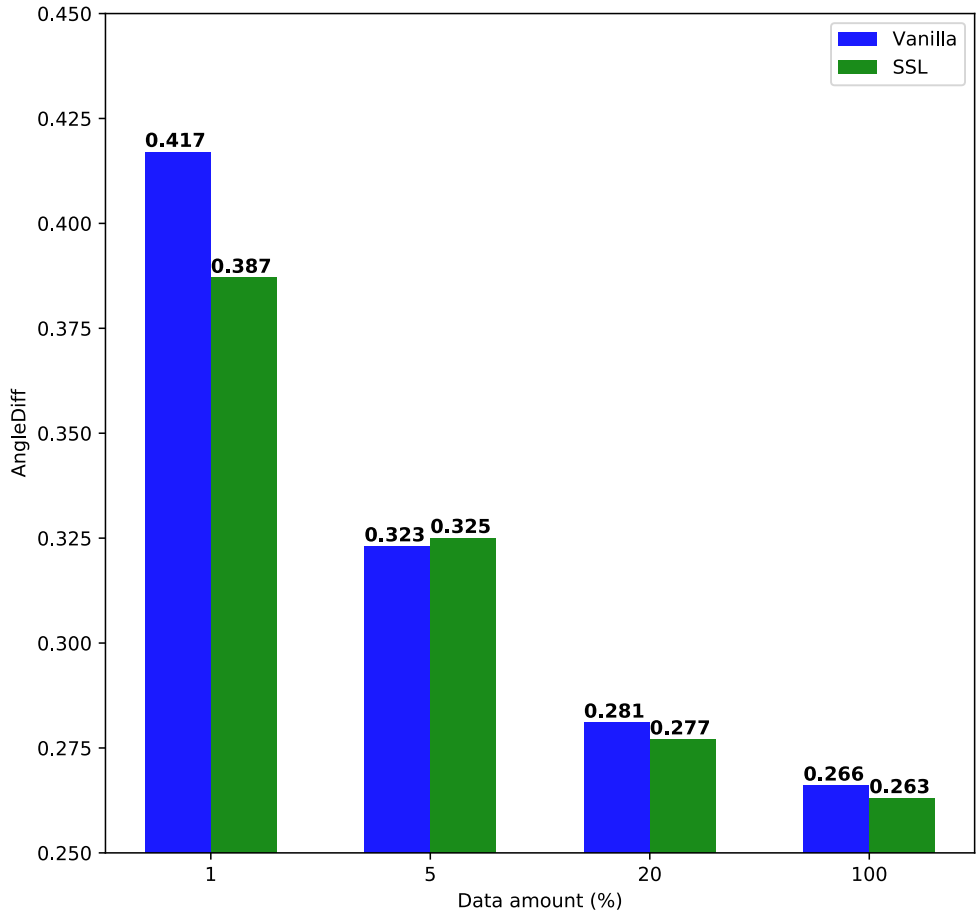


Figure 5.4: In general, SSL achieve lower angle error than Vanilla model. The difference is most evident at 1% labeled training data.

The possible characteristics of DukeMTMC dataset that encourages a supervised approach are:

- large amount of labeled samples
- fixed camera angle
- static background
- low variance in human size scale relative to background

5.2 Results on DroLAB

This section focus on evaluation of different predictor models on our indoor dataset. The models are grouped by their training datas as follows:

Model	MSE	AngleDiff (rad)	Accuracy (%)
Vanilla	**0.069**	0.266	**81.25**
RNN	0.086	0.300	78.15
SSL	**0.069**	**0.263**	79.60

Table 5.5: Test results on DukeMTMC

.

- trained on DukeMTMC (transfer learning)
- trained on DroLAB
- trained on DukeMTMC and finetuned on DroLAB

As described in section 4.1.2, we annotated our indoor dataset by manual labelling using a discrete set of angle values, which leads to high degree of ground truth ambiguity due to human interpretation. For this reason, relative accuracy is a more ideal metric than absolute angle difference. An angle prediction $\hat{\alpha}$ is considered correct with regard to ground truth α if $|\alpha - \hat{\alpha}| < \pi/8$ (section 2.1.1)
Full results are shown in table 5.6. Figure 5.5 compare the models by test accuracy.

Trained on	Model	MSE	AngleDiff (rad)	Accuracy (%)
DukeMTMC	Vanilla	0.236	0.604	48.05
	RNN	**0.217**	**0.576**	47.27
	SSL	0.238	0.593	**49.02**
DroLAB	Vanilla	**0.102**	**0.290**	**70.11**
	RNN	0.130	0.353	65.04
	SSL	0.134	0.337	67.38
DukeMTMC, DroLAB	Vanilla	0.081	0.252	73.44
	RNN	**0.070**	0.244	**76.367**
	SSL	0.081	**0.234**	**76.367**

Table 5.6: Test results on DroLAB. Vanilla model achieves best scores if trained solely on DroLab. SSL model gives best results if it is pretrained on the DukeMTMC dataset

.

All models trained on the DukeMTMC dataset achieved accuracy over 47% at angle difference at most 0.604. Figure 5.5 shows that for all architecture, highest accuracy score is gained by training on DukeMTMC and finetuning on DroLAB, with DroLAB contributing a major part to that score. This indicates that domain knowledge gained on the DukeMTMC dataset is transferable to the DroLAB dataset. Since accuracy score is higher for models trained on DroLAB compared to those trained on DukeMTMC, this suggests that the DroLAB labeled samples have a more prominent role in test results.

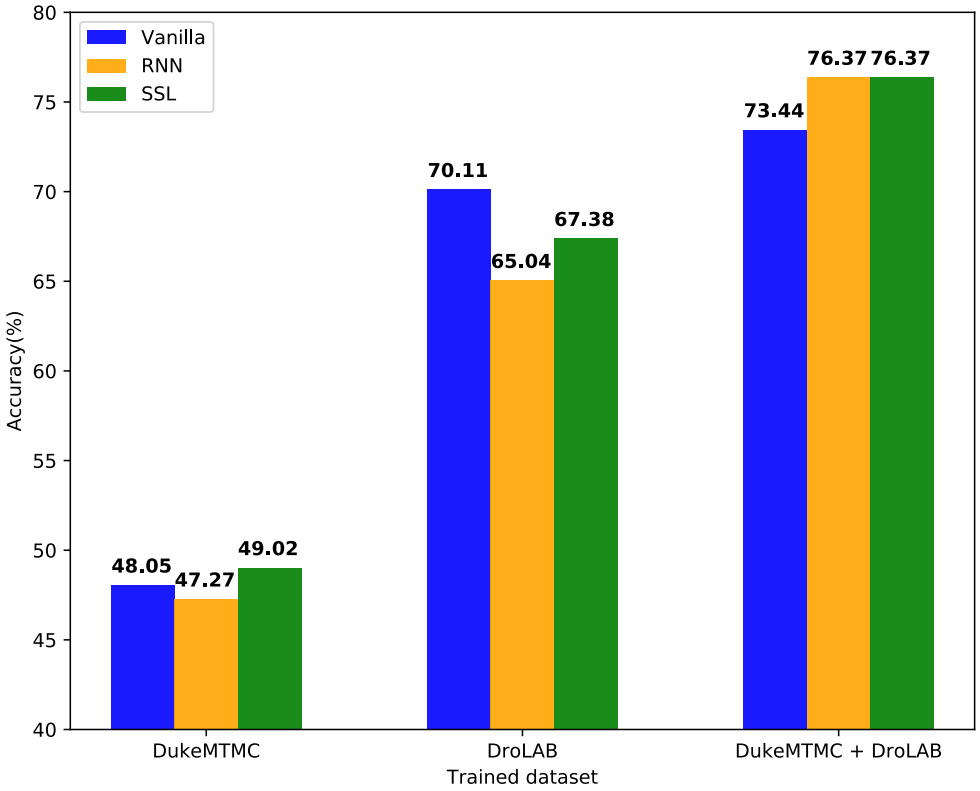


Figure 5.5: Test accuracy on DroLAB

SSL model outscores others in accuracy when trained on DukeMTMC and DukeMTMC + DroLAB. This suggests SSL approach can generalize better to out-of-domain and unseen data.

5.3 Results on COMBI

Similar to section 5.2, we perform a similar experiment and use Accurracy as the main evaluation metric. Results are shown in table 5.7. Figure 5.6 compare the models by test accuracy.

All models score at least 35% Accuracy which indicates a certain level of transfer learning. SSL predictors trained on DukeMTMC or DroLAB have best test scores in 3 metrics, which exhibits a good generalization capability.

Results obtained by training models on DukeMTMC are generally higher than on DroLAB. Among possible reasons, we suspect the followings:

- DukeMTMC training set contains far more samples than DroLAB (305943 vs 2000)

Trained on	Model	MSE	AngleDiff (rad)	Accuracy (%)
DukeMTMC	Vanilla	0.325	0.792	35.72
	RNN	0.324	0.767	38.79
	SSL	**0.307**	**0.752**	**39.20**
DroLAB	Vanilla	0.506	1.006	35.91
	RNN	0.501	1.00	35.64
	SSL	**0.481**	**0.962**	**36.13**
DukeMTMC, DroLAB	Vanilla	**0.288**	**0.664**	**49.44**
	RNN	0.308	0.715	48.19
	SSL	0.312	0.694	47.97

Table 5.7: Evaluation on COMBI

.

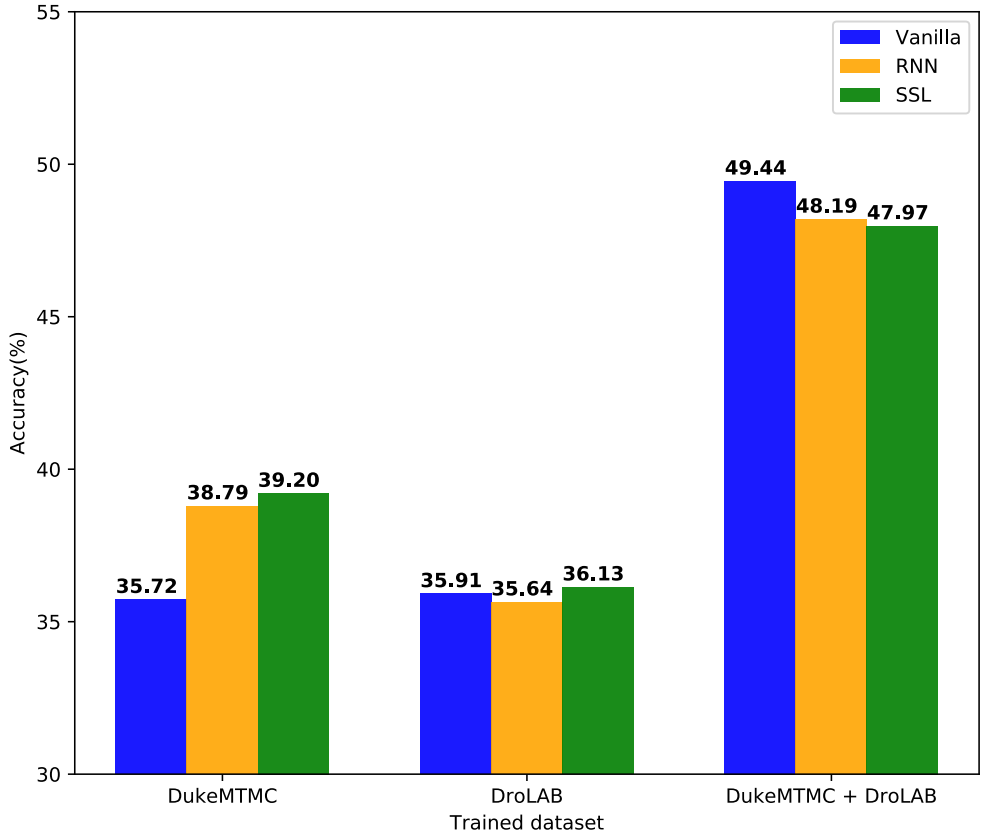


Figure 5.6: Test accuracy on COMBI

- DukeMTMC images are captured by cameras from above, which leads to samples being more similar to those from subdatasets 3DPeS and CMU-Drone. Training on DukeMTMC thus brings in more domain knowledge. On the other hand, DroLAB is

the only dataset using fisheye camera with distortion for close distance and filmed by a drone flying close to ground and human height.

Results by models trained on DroLAB vary by small margin. This is likely caused by samples from 2 datasets of vastly different distribution and characteristics.

All models give best testing score after being trained on DukeMTMC and finetuned on DroLAB. The accuracy compared to training only on one dataset is significant: 13.53% for Vanilla model, 9.4% for RNN model and 8.77% for SSL model.

To our surprise, the Vanilla predictor produced highest results after training and finetuning, followed by RNN model and SSL model in third place. This finding requires further investigation to explain.

6 Conclusion

6.1 Summary

Future unmanned aerial vehicles should be able to interact with human in booth indoor and outdoor environments. This necessitates a drone's capability to predict future movement and trajectory of humans, which makes heading direction estimation a fundamental task. This bachelor's thesis presented a solution to this problem by utilizing deep neural networks.

We proposed a simple predictor model that consists of a feature extractor and direction estimator. We investigated two machine learning architectures for the extractor component and three for the estimator subnetwork. To train and evaluate our models, we introduced a large scale outdoor dataset, an indoor dataset and a combination of various open subdatasets. We came up with three evaluation metrics to measure model performance.
Experimental results show that the MobileNet convolutional architecture is suitable to extract feature from person bounding boxes. A simple feed forward estimator achieves good results on the DukeMTMC dataset, given that the amount of labeled data is sufficiently high. All our models show generalization capability when tested on out-of-domain data to a certain extent.

Due to high cost of obtaining labeled samples, we are motivated to make use of unlabeled samples to increase model performance. We experimented with a semi-supervised loss based on a imperfect assumption about temporal continuity constraint. Our results clarify that the semi-supervised loss increase testing accuracy over supervised, especially in absence of data or domain knowledge. By leveraging training data on DukeMTMC, we could improve models accuracy on the DroLAB dataset by a noticeable margin. Our SSL model scores the lowest angle error and highest direction accuracy when evaluated on unknown datasets in general. Test results on COMBI dataset indicate that having more out-of-domain knowledge actually helps boost model accuracy, as the best performing predictor networks were trained on DukeMTMC and finetuned on DroLAB. This suggests there might be a general optimal model to solve the HDE problem.

We have not been able to explain all results, for example why a Vanilla model outperformed a RNN model and a SSL model on the COMBI dataset when all three were trained on DukeMTMC and DroLAB. This leaves room for investigation into dataset and model relations.

6.2 Future works

A long-term goal of our indoor drone project is to have a drone proactively detects and avoids moving person on its way to a destination. To do this we need to integrate the HDE module into the obstacle avoidance subsystem. For future research we plan to extend our indoor dataset by adding more actors and at the same time increase the maximal height above the ground. The drone should adjust to the altitude to find best tracking manuevers.

We would like to integrate more large scale datasets that offer automatic or semi-automatic labelling. A feasible approach is to simulate the drone and actor in a photo-realistic simulator like Microsoft AirSim.

We also want to further explore the potential of semi-supervised learning by trying out different temporal assumptions, for example using half an image sequence to predict feature vector or heading direction.

Bibliography

[1] Inc. Anaconda. *Anaconda Software Distribution*. Version Vers. 2-2.4.0. Nov. 1, 2017. URL: https://www.anaconda.com.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *Proceedings of the Third International Conference on Learning Representations (ICLR)*. 2015.

[3] Davide Baltieri, Roberto Vezzani, and Rita Cucchiara. "3DPes: 3D People Dataset for Surveillance and Forensics". In: *Proceedings of the 1st International ACM Workshop on Multimedia access to 3D Human Objects*. Scottsdale, Arizona, USA, Nov. 2011, pp. 59–64.

[4] Bitcraze. *Crazyradio PA*. https://www.bitcraze.io/crazyradio-pa. 2012.

[5] Bitcraze. "Flow deck | Bitcraze". In: (2018). URL: https://www.bitcraze.io/flow-deck/.

[6] Rogerio Bonatti et al. "Autonomous Cinematography using Unmanned Aerial Vehicles". In: IROS, 2018.

[7] Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. *Semi-Supervised Learning*. 1st. The MIT Press, 2010.

[8] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

[9] Jinyoung Choi, Beom-Jin Lee, and Byoung-Tak Zhang. "Human Body Orientation Estimation using Convolutional Neural Network". In: *CoRR* abs/1609.01984 (2016). arXiv: 1609.01984. URL: http://arxiv.org/abs/1609.01984.

[10] Balázs Csanád Csáji. "Approximation with Artificial Neural Networks". MA thesis. Eötvös Loránd University, 2001.

[11] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *MCSS* 2 (1989), pp. 303–314.

[12] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *In CVPR*. 2009.

[13] Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[14] Tino Fuhrman et al. "An interactive indoor drone assistant". Unpublished. 2019.

[15] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological cybernetics* 36 (Feb. 1980), pp. 193–202. DOI: 10.1007/BF00344251.

[16] Wojciech Giernacki et al. “Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering”. In: *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. IEEE. 2017, pp. 37–42.

[17] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[18] Hinton et al. *Artificial Intelligence: A Modern Approach, Third Edition*. Prentice Hall, 2010. ISBN: 978-0136042594.

[19] Hinton et al. *Unsupervised Learning: Foundations of Neural Computation*. MIT Press, 1999. ISBN: 978-0262581684.

[20] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://doi.org/10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[21] Sepp Hochreiter et al. *Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies*. 2001.

[22] Kurt Hornik. “Approximation Capabilities of Multilayer Feedforward Networks”. In: *Neural Netw.* 4.2 (Mar. 1991), pp. 251–257. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-T. URL: http://dx.doi.org/10.1016/0893-6080(91)90009-T.

[23] Andrew G. Howard et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[24] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[25] Tom Ko et al. “A study on data augmentation of reverberant speech for robust speech recognition”. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2017), pp. 5220–5224.

[26] Jonas Moritz Kohler et al. “Exponential convergence rates for Batch Normalization: The power of length-direction decoupling in non-convex optimization”. In: 2019.

[27] 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 et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[28] Iro Laina et al. “Deeper depth prediction with fully convolutional residual networks”. In: *3D Vision (3DV), 2016 Fourth International Conference on*. IEEE. 2016, pp. 239–248.

[29] Yann LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1 (1989), pp. 541–551.

[30] Yann LeCun et al. “Gradient-Based Learning Applied to Document Recognition”. In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[31] Lori Levin et al. “The Janus-III Translation System: Speech-to-Speech Translation in Multiple Domains”. In: *Machine Translation* 15.1 (June 2000), pp. 3–25. ISSN: 1573-0573. DOI: 10.1023/A:1011186420821. URL: https://doi.org/10.1023/A:1011186420821.

[32] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[33] S. Liu and W. Deng. “Very deep convolutional neural network based image classification using small training sample size”. In: *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*. Nov. 2015, pp. 730–734. DOI: 10.1109/ACPR.2015.7486599.

[34] Wei Liu et al. “SSD: Single Shot MultiBox Detector”. In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325.

[35] Hossein Mobahi, Ronan Collobert, and Jason Weston. “Deep Learning from Temporal Coherence in Video”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML ’09. Montreal, Quebec, Canada: ACM, 2009, pp. 737–744. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553469. URL: http://doi.acm.org/10.1145/1553374.1553469.

[36] Mozer and Michael. “A Focused Backpropagation Algorithm for Temporal Pattern Recognition”. In: *Complex Systems* 3 (Jan. 1995).

[37] John Nickolls et al. “Scalable Parallel Programming with CUDA”. In: *Queue* 6.2 (Mar. 2008), pp. 40–53. ISSN: 1542-7730. DOI: 10.1145/1365490.1365500. URL: http://doi.acm.org/10.1145/1365490.1365500.

[38] Adam Paszke et al. “Automatic differentiation in PyTorch”. In: (2017).

[39] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. “Searching for Activation Functions”. In: *CoRR* abs/1710.05941 (2017). arXiv: 1710.05941. URL: http://arxiv.org/abs/1710.05941.

[40] Ergys Ristani et al. “Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking”. In: *CoRR* abs/1609.01775 (2016). arXiv: 1609.01775. URL: http://arxiv.org/abs/1609.01775.

[41] A. J. Robinson and Frank Fallside. *The Utility Driven Dynamic Error Propagation Network*. Tech. rep. CUED/F-INFENG/TR.1. Cambridge, UK: Engineering Department, Cambridge University, 1987.

[42] Adrian Rosenbrock. *Object detection with deep learning and OpenCV*. 2017. URL: https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv (visited on 05/07/2019).

[43] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536.

[44] Shibani Santurkar et al. “How Does Batch Normalization Help Optimization?” In: *arXiv e-prints*, arXiv:1805.11604 (May 2018), arXiv:1805.11604. arXiv: 1805.11604 [stat.ML].

[45] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL).* 2016.

[46] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. "UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild". In: *CoRR* abs/1212.0402 (2012). arXiv: 1212.0402. URL: http://arxiv.org/abs/1212.0402.

[47] David Stavens and Sebastian Thrun. "Unsupervised learning of invariant features using video". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2010), pp. 1649–1656.

[48] Christian Szegedy et al. "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842 (2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.

[49] Alexander H. Waibel et al. "Phoneme recognition using time-delay neural networks". In: *IEEE Trans. Acoustics, Speech, and Signal Processing* 37 (1989), pp. 328–339.

[50] Wenshan Wang et al. "Improved Generalization of Heading Direction Estimation for Aerial Filming Using Semi-supervised Regression". In: *CoRR* abs/1903.11174 (2019). arXiv: 1903.11174. URL: http://arxiv.org/abs/1903.11174.

[51] Xiaolong Wang and Abhinav Gupta. "Unsupervised Learning of Visual Representations Using Videos". In: *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 2794–2802.

[52] Werbos and Paul. "Generalization of Backpropagation with Application to a Recurrent Gas Market Model". In: *Neural Networks* 1 (Dec. 1988), pp. 339–356. DOI: 10.1016/0893-6080(88)90007-X.

Acronyms

ASR automatic speech recognition. 6

BPTT backpropagation through time. 13

CF2.0 Crazyflie 2.0. 4–6, 24

CMU Canergie Mellon University. 26

CNN convolutional neural network. 7, 11, 12, 15, 16

CUDA Compute Unified Device Architecture. 27

FC fully connected. 17, 19

FPS frame per second. 23, 24

FPV first person view. 4

GPU graphics processing unit. 27

GRU gated recurrent unit. 13

HDE heading direction estimation. 1–3, 15, 31

KIT Karlsruhe Institute of Technology. 24

LSTM long-short term memory. 13

MSE mean square error. 15, 30

NN neural network. 1, 2, 8, 10–12, 17, 21, 26, 27

ReLU linear rectifier unit. 17, 18

RGB red, green, blue. 15

RNN recurrent neural network. 12, 13, 15

SE square error. 4

SLAM simultaneous localization and mapping. 7

SSD single shot multibox detector. 24

SSL semi-supervised learning. 1, 14, 15, 33, 36

TDNN time delay neural network. 11

TTS text-to-speech. 6

UAV unmanned aerial vehicle. 1, 4

Glossary

Acc Accuracy. 4

AngleDiff absolute angle difference. 4

BatchNorm batch normalization. 18

drone unmanned aerial vehicle. 1

epoch number of times the algorithm sees the entire data set. 10

gradient descent a first-order iterative optimization algorithm for finding the minimum of a function. 9

iteration number of times a batch of data passed through the algorithm. 10, 30

manifold a topological space that locally resembles Euclidean space near each point. 14

normalization adjusting values measured on different scales to a notionally common scale. 10

overfitting a statistical model overfits when it contains more parameters than can be justified by the data. 10

quadcopter a multirotor helicopter which is lifted and propelled by four rotors. 1, 4

ResNet residual neural network. 12