Comenius University in Bratislava

Faculty of Mathematics, Physics and Informatics

Label-efficient learning in artificial neural networks

Dissertation Thesis

2021 **Mgr. Ing. Matúš Tuna**

Comenius University in Bratislava

Faculty of Mathematics, Physics and Informatics

Label-efficient learning in artificial neural networks

Dissertation Thesis

Supervisor: **prof. Ing. Igor Farkaš, Dr.**

Study program: **Computer Science**

Field of study: **2508 Computer Science**

Department: **Department of Applied Informatics**

Bratislava 2021 **Mgr. Ing. Matúš Tuna**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Mgr. Ing. Matúš Tuna
Študijný program: informatika (Jednoodborové štúdium, doktorandské III. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: dizertačná
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Label-efficient learning in artificial neural networks
Učenie umelých neurónových sieti s efektívnym využitím označených dát

Anotácia: Umelé neurónové siete sa osvedčili ako veľmi úspešné modely strojového učenia pri rôznych úlohách. Ich pretrvávajúcim obmedzením, typickým pre klasifikáciu vzorov, je však silná závislosť na rozsiahlom súbore označených príkladov. Jednou z výziev je učenie s efektívnym využitím označených trénovacích príkladov, ktoré by v konečnom dôsledku rozšírili použiteľnosť neurónových sietí na prípady, kde je k dispozícii iba obmedzené množstvo označených dát.

Cieľ: 1. Vypracovať prehľad modelov neurónových (NS) sietí s efektívnym využitím označených dát. 2. Navrhnúť nové modely NS s efektívnym využitím označených dát. 3. Aplikovať existujúce modely ma nové domény. 4. Implementovať a otestovať modely na vybraných úlohách.

Literatúra:
Lake B. et al. "One shot learning of simple visual concepts." Proceedings of the annual meeting of the Cognitive Science Society, 2011.
Koch G., Zemel R., Salakhutdinov R. "Siamese neural networks for one-shot image recognition." ICML deep learning workshop, 2015.
Tarvainen A., Valpola H. "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results." arXiv:1703.01780, 2017.

Kľúčové slová: umelé neurónové siete, hlboké učenie, transferové učenie, augmentácia dát, few-shot učenie, polokontrolované učenie

Školiteľ: prof. Ing. Igor Farkaš, Dr.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 22.01.2016

Dátum schválenia: 16.02.2017 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
školiteľ

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Mgr. Ing. Matúš Tuna
Study programme:	Computer Science (Single degree study, Ph.D. III. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Dissertation thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Label-efficient learning in artificial neural networks

Annotation: Artificial neural networks have proven very successful learning machines, excelling at performance in different tasks. However, their remaining limitation, typically in pattern classification, is the strong dependence on a large dataset of labeled training examples. One of the challenges is label-efficient learning, based on few labelled training examples, that would eventually expand the applicability of neural networks to cases where only limited data resources are available.

Aim: 1. Write an overview of neural networks with effective use of labelled data. 2. Design new models of neural networks with effective use of labelled data. 3. Apply existing models to new domains. 4. Implement and test models on selected tasks.

Literature:
Lake B. et al. "One shot learning of simple visual concepts." Proceedings of the annual meeting of the Cognitive Science Society, 2011.
Koch G., Zemel R., Salakhutdinov R. "Siamese neural networks for one-shot image recognition." ICML deep learning workshop, 2015.
Tarvainen A., Valpola H. "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results." arXiv:1703.01780, 2017.

Keywords: artificial neural networks, deep learning, transfer learning, data augmentation, few-shot learning, semi-supervised learning

Tutor:	prof. Ing. Igor Farkaš, Dr.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	prof. Ing. Igor Farkaš, Dr.
Assigned:	22.01.2016

Approved: 16.02.2017 prof. RNDr. Rastislav Kráľovič, PhD.
Guarantor of Study Programme

.....
Student

.....
Tutor

Declaration of Originality

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text, and a list of references is given in the bibliography.

Bratislava, 6.9.2021 Mgr. Ing. Matúš Tuna

Acknowledgments

I would like to express my gratitude to my supervisor Igor Farkaš for his valuable help, tips, consultations and time that he afforded me during my studies and during the writing of this thesis. I would also like to thank Kristína Malinovská for her valuable discussions and help during my studies.

Bratislava, 6.9.2021 Mgr. Ing. Matúš Tuna

Abstrakt:

Umelé neurónové siete sú v súčasnosti najviac skúmanými modelmi v oblasti strojového učenia s mnohými aplikáciami v oblastiach ako rozpoznávanie objektov, strojový preklad jazyka, robotika a generovanie obrázkov. Avšak aplikácia umelých neurónových sietí môže byť v praxi obmedzená absenciou dát s označením. Témou tejto dizertačnej práce sú algoritmické techniky na zníženie množstva označených dát potrebných na úspešné natrénovanie modelov založených na umelých neurónových sieťach. Medzi tieto techniky patrí transferové učenie, augumentácia dát, few-shot učenie, polokontrolované učenie a modely ktoré využívajú rozsiahle datasety bez označených dát na riešenie rôznych úloh bez učiteľa. V tejto práci prezentujeme výsledky ktoré vylepšujú úspešnosť modelov založených na neurónových sieťach s použitím malého množstva označených dát v rôznych oblastiach vrátane klasifikácie obrázkov, segmentácie obrázkov, odhadovania pózy objektov a detekcie objektov.

Kľúčové slová: umelé neurónové siete, hlboké učenie, transferové učenie, augumentácia dát, few-shot učenie, polokontrolované učenie

Abstract:

Artificial neural networks are currently the most widely used and researched models in machine learning with numerous applications in domains such as object recognition, machine translation, robotics or image generation. However, the application of artificial neural networks can often be hindered by the lack of labeled data. The topic of this thesis are techniques designed to lower the amount of labeled data needed to successfully train artificial neural network models. Among these techniques are transfer learning, data augmentation, few-shot learning, semi-supervised learning and models that use large scale unlabeled datasets to learn to solve various tasks in an unsupervised manner. We present results that improve label efficiency in various domains including image classification, image segmentation, object pose estimation and object detection.

Keywords: artificial neural networks, deep learning, transfer learning, data augmentation, few-shot learning, semi-supervised learning

Contents

Introduction ……..1

1. Artificial neural networks and their applications……... 3

1.1. Neural networks and gradient based learning ……3

1.2. Convolutional neural networks and their use in image recognition ……..5

1.3. Recurrent neural networks ……....8

1.4. Neural network-based reinforcement learning models ……...11

1.5. Generative models based on neural networks ……14

2. Strategies for increasing label efficiency of artificial neural networks ……..18

2.1. Transfer learning and data augmentation strategies ……18

2.2. Few-shot learning strategies ...……21

2.2.1. Siamese neural networks ……..23

2.2.2. Matching networks ……...24

2.2.3. Meta-Learner LSTM ……26

2.2.4. Prototypical networks ……...27

2.2.5. Temporal-convolution-based meta-learner ……..28

2.2.6. MAML ……..28

2.2.7. REPTILE ……..30

2.3. Semi-supervised learning strategies ……31

2.3.1. Ladder Networks, Pi Model and Temporal Ensembling ……..34

2.3.2. Mean Teacher model ……36

2.3.3. Contrastive Predictive Coding ……..38

2.4. Label-efficient neural networks at scale...……...39

3. Experimental results ……...44

3.1. Categorical Siamese Networks ……...44

3.1.1. Method description ……...44

3.1.2. Results ……..47

3.2. Few-shot semantic segmentation using Seg-REPTILE algorithm ……..51

3.2.1. Method description ……...52

3.2.2. Results ……..53

3.3. Detecting Wearable Objects via Transfer Learning ……55

3.3.1. Method description ……...56

3.3.2. Results ……61
3.3.2.1. Hyperparameter search ……61
3.3.2.2. Comparison between setups ……62
3.4. Semi-supervised Learning in Camera Surveillance Image Classification……..…..65
3.4.1. Method description ……...65
3.4.2. Results ……..68
3.4.2.1. Data augmentation experiments ……...68
3.4.2.2. Binary Mean Teacher experiments ……...70
3.5. Dataset and methods for training 6D Object Pose Estimator for Imitation Learning from RGB Video …….....74
3.5.1. Method description ……...77
3.5.2. Results ……..82
3.5.2.1. Hyperparameter search and the impact of data augmentation ……..82
3.5.2.2. Examination of environmental factors ……..83
3.5.2.3. Results summary ……...87
Conclusion ……...89
References ……92

List of Figures

1. Schema of a fully connected multilayer perceptron ……4
2. Example of a convolutional neural network architecture……..6
3. Example of an Inception layer……...7
4. Schema of a residual block……8
5. Example of a recurrent neural network with one hidden layer……..9
6. Schema of long short-term memory cell……10
7. Architecture of the Transformer model……11
8. Neural network architecture used by Deep Q-learning……..13
9. Diagram of the training procedure for GANs……16
10. Examples of images generated from text descriptions by StackGAN++……16
11. Examples of images from Omniglot dataset……..22
12. Schema of the Siamese neural network architecture……..23
13. Schematic representation of MAML training procedure……...29
14. Schematic representation of REPTILE training procedure……31
15. Pi model training procedure……...35
16. Temporal ensambling model training procedure……...36
17. Mean Teacher model training procedure……...37
18. GPT-3 few-shot learning accuracy across 42 language benchmarks……41
19. Sample of outputs from the DALL-E model……..42
20. The architecture of Categorical Siamese Network……44
21. Episode-based categorization procedure……...45
22. Architecture of CSN subnetworks for the Omniglot experiments……..48
23. Architecture of convolutional residual block……48
24. Architecture of CSN subnetworks for the Mini-ImageNet experiments……50
25. High-level schema of Fully Convolutional Network architecture……..53
26. Examples of segmentation from our model……...54
27. Examples of images from re-annotated DukeMTMC dataset……58
28. t-SNE visualization from the last layer of DenseNet161……64
29. Schema of the Binary Mean Teacher model……..67
30. The influence of different augmentation strategies on accuracy……69
31. The influence of the fraction of training data on the fully supervised model accuracy...70

32. Overview of different setups in the test part of the dataset……78
33. Overview of the data acquisition setup……..78
34. Examples of the data from the Train set……79
35. Examples of different augmentation techniques……79
36. Qualitative results on Test dataset……87

List of Tables

1. Classification accuracies for Omniglot in comparison with selected models……49
2. Classification accuracies for Omniglot with small CSN……49
3. Classification accuracies for Mini-ImageNet in comparison with selected models…...50
4. Categories included in various test splits……...54
5. Performance comparison of the models using 1 example per class……54
6. Performance comparison of the models using 5 examples per class……..54
7. Dataset statistics……58
8. Characteristics of neural network architectures used in our experiments……...59
9. BACC for hyperparameter search in setup A using DenseNet161……61
10. BACC for hyperparameter search in setup B using ResNet50……...61
11. Characteristics of the output classifier used in Setup A……..62
12. BACC for all feature extractors and classifiers in setup A……62
13. BACC and ACC of our best models on the test set in setup A……63
14. BACC and ACC of our best models on the test set in setup B……63
15. Naming scheme and statistics of the datasets used in our experiments……...66
16. Parameters and names for augmentation techniques that we use……...68
17. Hyperparameters for Binary Mean Teacher for all setups……..72
18. Optimal hyperparameters for Binary Mean Teacher for specific setups……72
19. Comparison between fully supervised models and Binary Mean Teacher using randomly initialized weights……72
20. Comparison between fully-supervised models and Binary Mean Teacher using ImageNet initialized weights……73
21. Comparison between ADD pass rates for models trained using input images with different resolution……82
22. Comparison between ADD pass rates for different augmentation methods……...83
23. Generalization across demonstrators (subjects)……84
24. Generalization across camera viewpoints……..84
25. Generalization across left and right hand of the demonstrator……...85
26. Generalization across different levels of background clutter……86
27. Generalization across tasks……86

Introduction

Artificial neural networks (ANNs) are currently one of the most widely used and researched models in machine learning. Models based on ANNs have numerous applications in many areas of artificial intelligence where they achieve state-of-the-art results such as object recognition (Krizhevsky et al., 2012), machine translation (Wu et al., 2016), control of artificial agents in complex environments (Mnih et al., 2015), generation of realistic looking pictures from particular domain (Odena et al., 2017) or language modelling (Brown et al., 2020). In some domains, ANNs achieve better results than humans. Examples are object recognition of objects in photos (Szegedy et al., 2017) or the board game Go (Silver et al., 2017).

Learning algorithms in most models based on ANNs are based on gradient descent learning method combined with backpropagation mechanism (Werbos, 1974; Rumelhart et al., 1986). These kinds of algorithms require a cost function that specifies the error rate of an artificial neural network model when solving a particular problem. Using the gradient descent learning method with a backpropagation mechanism we can then change the weights of a particular network in a way that minimizes the cost function, which results in increasing performance of a network on a particular problem. This learning method has in recent years proven to be very effective, but also very demanding on computational resources and the number of labelled training examples needed for convergence to correct solution. This is why ANNs became popular only in recent years when cheap hardware optimized for matrix operations came to the market in the form of graphics processing units (GPUs) and when a source of large amounts of training data became available, otherwise known as the internet.

Because of the need for a large number of labeled training examples for convergence to correct solution, applications of ANNs are currently limited to domains with a large number of labeled training examples. One example is the domain of object recognition, where there is many human annotated photographs and pictures available on image sharing services. Another good example is the domain of computer games where although we do not have access to a large number of annotated training examples, we can utilize the deep reinforcement learning method (Mnih et al., 2013) which enables an artificial agent to learn the correct strategy during large amounts of interactions with a simulated environment.

Unfortunately, in many important problems in artificial intelligence we do not have access to either large amounts of annotated training data or large a number of interactions with a simulated environment. One example is the situation of a robotic agent interacting with a real-

world environment that needs to continuously adapt and learn tasks in its environment. As opposed to a simulated environment, an artificial agent in the real-world environment cannot afford a large number of interactions with an environment to learn some task because of time constraints and because some of these interactions could result in a damage of either the agent or the environment itself. In other words, an artificial agent with limited access to the annotated data or interactions with an environment requires a learning procedure that can converge to a good solution using only a limited number of annotated training data or a limited amount of interactions with the environment.

In chapter 1 we will explore the basic theory behind ANNs and introduce fundamental ANN models used today and their applications. There are several approaches for increasing the label efficiency of artificial neural networks. In chapter 2 we review various approaches to the problem of label efficiency of artificial neural networks. These approaches include standard transfer learning and data augmentation strategies, few-shot learning strategies, semi-supervised learning strategies and self-supervised learning strategies. We mainly focus on label efficiency in the image domain, but these strategies also apply to other domains. In chapter 3 we present several models that improve label efficiency in various domains including image classification, image segmentation, object pose estimation and object detection. In this chapter we also present experimental results and comparisons with other methods.

1. Artificial neural networks and their applications

1.1 Neural networks and gradient based learning

Artificial neural networks (ANNs) in recent years saw a resurgence of interest both in academic and commercial sphere. The reason for this resurgence, as mentioned before, is the performance of ANN models in many challenging tasks. An ANN can be viewed as a connectionist model that consists of a number of basic computational units called neurons that are interconnected with a set of adaptable parameters called weights. Each neuron in an ANN computes an output that is a function of its input. ANNs as a whole can be viewed as a composite function f, or network function, that transforms an input space X to an output space Y.

$$f : X \to Y \quad (1)$$

ANNs learn by observing the examples from an input space X and then adapting their weights so that the network learns a network function f that can correctly transform the input X to output Y. This should be true even for examples of X that the network did not encounter before. The basic computation unit of ANNs, the neuron, can compute different functions. In most ANN models the neuron computes the sum of the input vector x weighted by learnable weights w. Usually there is also a nonlinear activation function σ applied to the internal activity of a particular neuron. The output of a typical neuron in ANN can thus be expressed in the following way, where b represents the bias of a particular neuron and i represents the index of a particular input and the corresponding weight.

$$y = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right) \quad (2)$$

The neurons in ANNs are organized in layers. The simplest ANNs have only an input layer and output layer connected with weights and are called simple perceptrons (Rosenblatt, 1958). Despite the initial promising results of perceptrons when it comes to classification, Minsky and Papert (1969) showed that these models are unable to learn classification tasks in which the classes are not linearly separable and thus cannot be the solution to the general classification problem. Neural networks need at least one hidden layer between the input and output layer in order for them to learn classification problems with classes that are not linearly separable. Basic architecture of an ANN with one hidden layer is illustrated in Figure 1.

The weights in ANNs are adapted using a learning algorithm. Through the history of ANN research, many learning algorithms were developed such as expectation-maximization (Dempster et al., 1977), simulated annealing (Kirkpatrick et al., 1983) or Hebbian learning (Hebb, 1949), but we will focus on an algorithm based on gradient based learning using backpropagation algorithm developed independently by Paul Werbos and David Rumelhart. Backpropagation algorithm enables us to apply the gradient based learning to ANNs with multiple hidden layers and is currently the basis of all state-of-the-art neural network models used today.

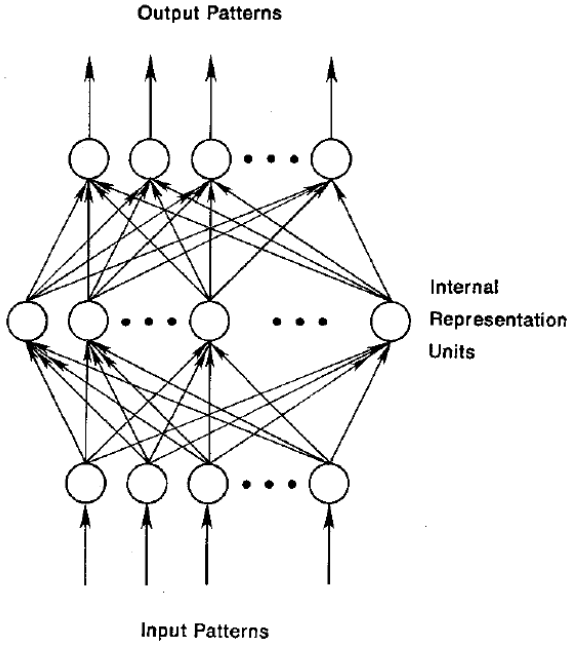

Figure 1. Schema of a fully connected multilayer perceptron with one hidden layer (Rumelhart et al., 1986).

The basic principle of the backpropagation algorithm is to use the gradient descent optimization method for finding such a combination of learnable parameters (weights) that minimizes the cost function, i.e that maximizes the performance measure of particular neural network on some task. The cost function is often fine-tuned to the task, but for most classification problems general cost functions like cross-entropy or mean-square error are used. Given p training examples, correct labels t, and network outputs o, we can define the mean squared error (MSE) and the cross-entropy (CSE) cost functions in the following way:

$$MSE = \frac{1}{2p}\sum_{i=1}^{p}(o(i) - t(i))^2 \quad (3)$$

$$CSE = -\sum_{i=1}^{p} [\, t(i) \ln o(i) + (1 - t(i)) \ln(1 - o(i))] \tag{4}$$

The backpropagation algorithm can be divided into following basic steps:

- output of the network is calculated during the forward pass from a set of training examples,
- cost function E is calculated from the output of the network and ground truth (labels),
- we calculate the gradient ∇E of the cost function $\nabla E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_l})$, with respect to every learnable parameter (weight) of the network using the chain rule,
- we update every weight and bias in the neural network by subtracting the gradient scaled by a learning constant α from the value of the weight $w_l^{t+1} = w_l^t - \alpha \frac{\partial E}{\partial w_l^t}$.

Since the publication of the original backpropagation paper, several new variants of gradient descent optimization algorithm were developed that often exhibit better performance than the standard gradient descent algorithm. Some of the most widely used algorithms include AdaDelta (Zeiler, 2012), AdaGrad (Duchi et al., 2011) or ADAM (Kingma et al., 2014).

In subchapters 1.2-1.5 we will examine some of the most significant neural network architectures and their applications. All of these architectures use gradient descent learning with backpropagation mechanism.

1.2 Convolutional neural networks and their use in image recognition

It is well known that building some constraints (a priori knowledge) relevant for the domain that a machine learning system operates in can often enhance generalization capabilities of that system. This is also true for machine learning systems based on artificial neural networks. Convolutional neural networks (CNNs) developed by LeCun (1989) for use in handwritten image recognition have several important constraints built in its architecture that enables CNNs to perform better than a multilayer perceptron in the image domain.

The main constraint built into CNNs is the receptive field arrangement of the neurons in the individual layers of convolutional networks that resembles the neuron organization in the primary visual cortex which was first discovered in seminal work by Hubel and Wiesel (1962). Unlike neurons in the multilayer perceptron, neurons in CNNs are connected by their weights to only a small subset of their input space. When presented with an input space the weight matrix of every neuron “slides” over the input space in a process known as the convolution.

This process results in feature maps that contain responses of all neurons in a particular layer for all spatial locations in the input space. Typically a CNN contains a multilevel hierarchy of convolutional layers where neurons in convolutional layers farther form input have larger receptive fields. This arrangement of neurons is beneficial in the image domain due to the compositional nature of images i.e. objects in natural scenes can be decomposed into parts (or features), that can also be decomposed into lower level parts all the way down to elementary features (edges). These features as well as objects themselves can appear anywhere in the image, therefore we want the object recognition model to be invariant to change in spatial position. Due to the hierarchical arrangement of CNNs and the convolutional operation, CNNs can learn features that exploit compositional nature of images and features that are invariant to translation (change in spatial position). Architecture of a Convolutional Neural Network from Krizhevsky et al. is illustrated in Figure 2.

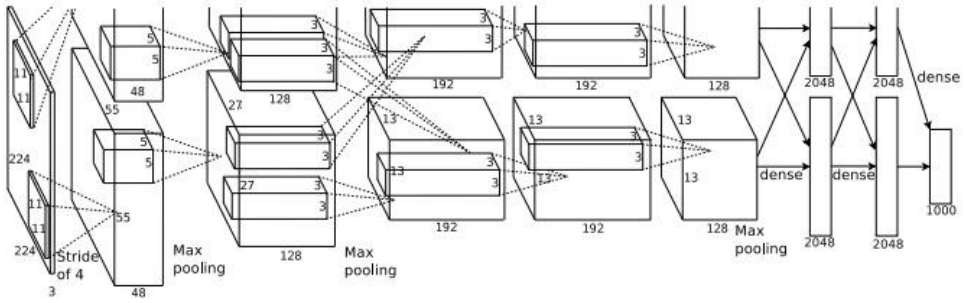

Figure 2. Example of a convolutional neural network architecture (Krizhevsky et al. 2012).

Besides convolutional layers, basic CNNs typically contain max pooling or average pooling layers and fully connected layers that are responsible for computing the output of the network based on the activations of final convolutional layer. Max pooling and average pooling layers aggregate together the outputs of neighboring neurons by selecting only maximum or average activation of neurons in predefined locations in the feature map. Pooling layers are used for downsampling of the input space and are known to improve the invariance characteristics of CNNs. Unfortunately both max pooling and average pooling layers "throw" away some spatial information contained in the feature maps. For this reason max pooling and average pooling layers are in most recent models often replaced by convolutional layers with larger receptive fields which can be used for subsampling while avoiding the loss of spatial information.

CNNs have numerous applications, particularly in image recognition and processing. Alex Krizhevsky, Sutskever and Hinton (2012) first demonstrated that CNNs can outperform

competing machine learning methods in image classification. Since then, CNNs became the standard machine learning method for most image processing tasks including image generation or control of artificial agents using raw visual input. Several architectural changes enabled the design of CNNs with a large number of hidden layers (up to 150) with much higher performance. The first architectural change was the introduction of Inception layers (Szegedy et al., 2015) that replaced convolutional layers with a single filter size with so called Inception layers. These layers combine feature maps with multiple filter sizes which enables each layer to combine information about local and more global features into a simple output while at the same time lowering the computational requirements of very deep networks. Figure 3. illustrates a basic architecture of the Inception layer.

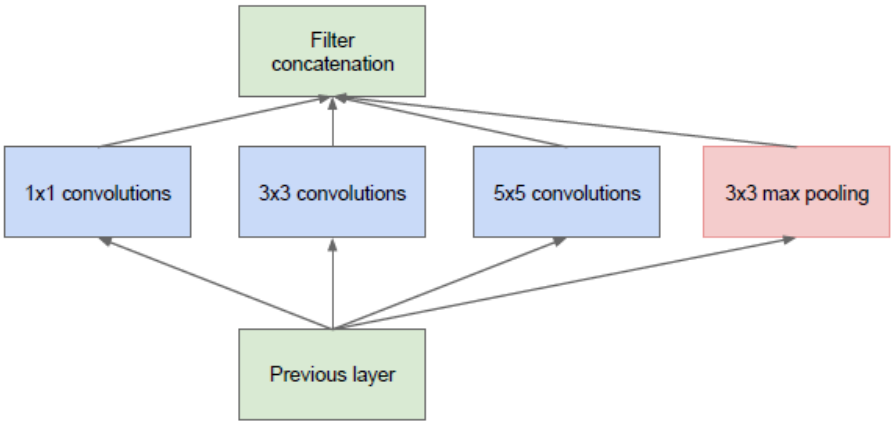

Figure 3. Example of an Inception layer (Szegedy et al., 2015).

Another key insight comes from a ResNet architecture (He et al., 2016). According to the authors ResNet architecture addresses the key problem of constructing very deep architectures which can potentially learn more complex features and thus improve the performance, by alleviating the vanishing gradients problem. Vanishing gradients are a result of gradient multiplication which can result in a situation when gradients go to zero in neural network layers closer to the input to the network which can halt the learning in very deep networks. ResNet architecture addresses this problem by introducing the so-called residual blocks, illustrated in Figure 4. Residual block consists of a number of stacked convolutional (or fully connected) layers and a skip connection that simply copies the input to residual layer and adds it to the output of the residual layer. During backpropagation this enables the gradient from the layer above to pass through skip connection directly, bypassing the weights in the residual block which leads to better propagation of gradients through the network.

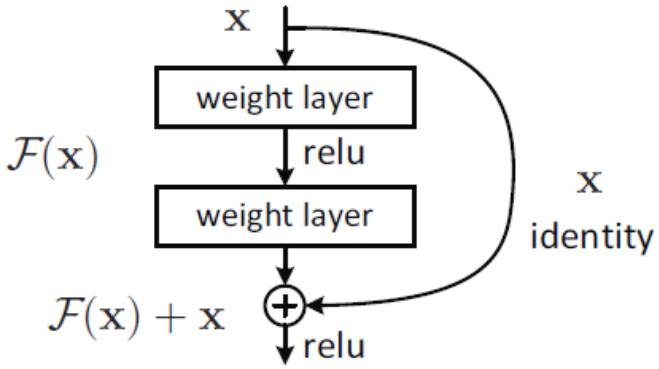

Figure 4. Schema of a residual block (He et al., 2016).

A combination of Inception and ResNet architectures (Szegedy et al., 2017) managed to approach human-like performance on the ImageNet image recognition dataset (Russakovsky et al., 2015) with 3.7% top-5 error rate. This is significant, because of the challenging nature of the dataset that consists of photos of real-world objects collected from the internet belonging to 1000 classes.

Beside these key architectural changes, several other changes to standard CNN architecture were made that contributed to improved performance of CNNs:

- usage of faster optimization algorithms like ADAM or RMSprop,
- more efficient weight initialization techniques like Xavier initialization (Glorot et al., 2010),
- usage of alternative activation functions like ReLu, that enables better propagation of gradients through the network and reduces vanishing gradient problem,
- better normalization methods like batch normalization (Ioffe et al., 2015).

1.3 Recurrent neural networks

The main advantage of recurrent neural networks (RNNs) is that unlike feed-forward models such as multilayer perceptrons and CNNs they can explicitly model sequences of inputs with arbitrary length. They are also capable of generating sequences of outputs. These features make recurrent neural networks ideal for tasks like time series prediction, language translation and language modelling as well as any domains where the input data is of the sequential nature.

The architecture of RNNs is similar to feed-forward models, but with recurrent connections on hidden layers that connect hidden layers to themselves through time steps in the

sequence which enables RNNs to model sequences of data. Most RNNs today are trained using a variants of training procedure known as backpropagation through time (Werbos, 1990). The basic principle behind backpropagation through time is the unrolling of the timesteps during the computation of the output. Unrolling procedure is illustrated in Figure 5. At every timestep a part of the input sequence corresponding to that timestep is inputted to the network together with activations from hidden layers from previous timestep connected to corresponding hidden layers with recurrent weights. Based on the input at the current timestep and activations of hidden layers from previous timesteps the output at current timestep is computed. The value of the cost function and gradients (using gradient descent) are then calculated for that timestep. This process is repeated for every data point in input sequence. When the end of the sequence is reached, the gradients are then accumulated and weights of the network are updated.

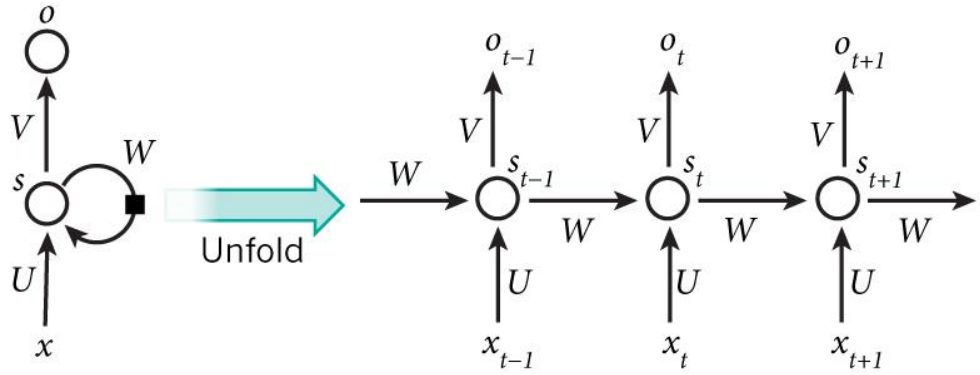

Figure 5. Example of a recurrent neural network with one hidden layer with input x, output o, weight matrices V and U and a recurrent weight matrix W. On the right is the unrolled view of the network (from http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns)

The unrolled RNN can be viewed as one big composite neural network that consists of several copies of a feed-forward network (one for each timestep) connected laterally with recurrent connections. In long sequences with many steps this architecture and learning algorithm suffer from the same vanishing and exploding gradient problem as very deep feed-forward networks. Long short-term memory (LSTM) architecture (Hochreiter et al., 1997) was developed to deal with vanishing and exploding gradient problem in recurrent neural networks when processing long input sequences. In LSTM individual neurons are replaced with the so called cells. For every cell in the hidden layers there are several gates that control the flow of information through the cell called forget gate, input gate and output gate. These cells are responsible for learning which parts of information coming to the cell should be retained, modified and outputted and which should be "forgotten". Figure 6. illustrates the architecture

of the LSTM cell. This allows for more efficient flow of gradient through the network and as a result, allows for the processing of much longer input sequences. There are several variants of the LSTM architecture in use today including LSTM with peephole connections (Gers et al., 2002), Gated Recurrent Units (Chung et al., 2014) or bidirectional LSTM (Graves et al., 2005).

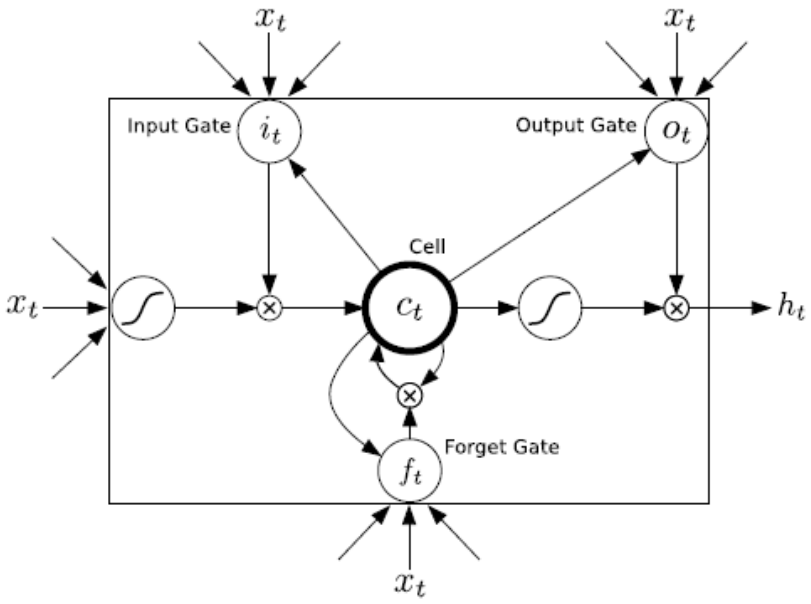

Figure 6. Schema of long short-term memory cell (Graves et al., 2013).

RNNs with LSTM like units have been used to tackle many problems in various areas of artificial intelligence. Bidirectional LSTM was successfully applied to a speech recognition problem while significantly outperforming other models like Hidden Markov Models (Graves et al., 2013).

Recently RNNs were being replaced by a new model called Transformer (Vaswani et al., 2017) for sequence prediction tasks. In Transformer model recurrent connections are replaced by attention module that models relationships between individual elements in the sequence. This attention module only uses feedforward connections which eliminates the vanishing and exploding gradient problem. Information about the position of individual elements in the sequence is provided to the network as an additional input variable. Transformer models currently constitute a backbone of state-of-the-art machine translation models (Liu et al., 2020), language prediction models (Brown et al., 2020) or item recommendation models used in various web services (Kang et al., 2018). Transformer model architecture is displayed in Figure 7.

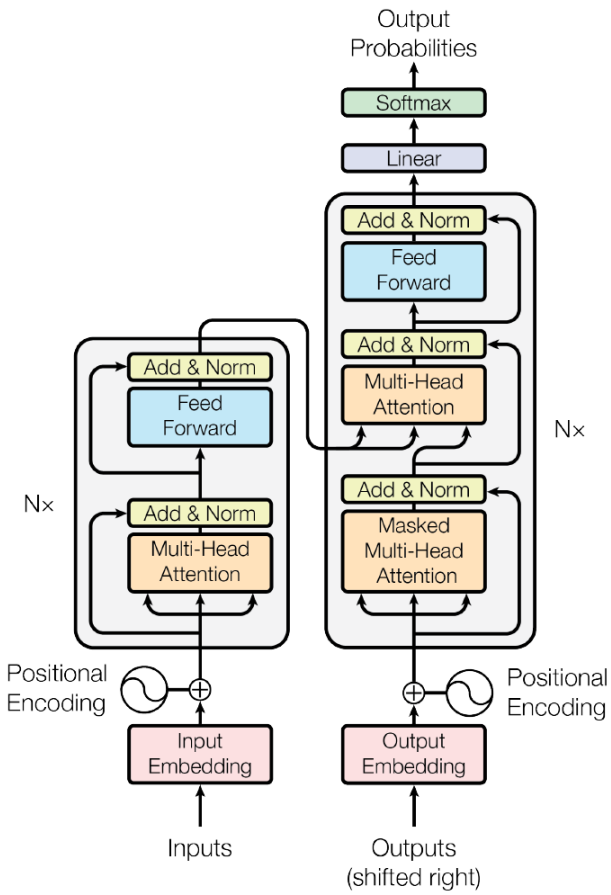

Figure 7. Architecture of the Transformer model (Vaswani et al., 2017).

1.4 Neural network-based reinforcement learning models

Reinforcement learning (RL) is defined as a subfield of machine learning that deals with the question of how should some artificial system, or agent, select actions that maximize rewards that the agent receives from the environment. Reinforcement learning algorithms are often applied to domains where the agent does not have access to a supervisory signal. Instead, the agent only has access to some representation of the state of the agent's environment and to reward from this environment. From the representation of the state of agent's environment and reward signals that the agent receives from the environment during interactions, the agent has to infer a policy that maximizes agent's reward. Thus, the basic reinforcement learning procedure can be defined in the following way:

- at time t the agent receives an observation O_t and a scalar reward R_t from the environment,
- based the observation O_t and a policy the agent selects an action A_t,
- environment receives an action A_t and emits an observation O_{t+1} and a reward R_{t+1},

- the agent receives an observation O_{t+1} and a reward R_{t+1}, selects a new action A_t and, if necessary, modifies its policy.

There are several algorithms that can be used to solve reinforcement learning problem including dynamic programming, Monte-Carlo methods, temporal difference learning (Sutton, 1988) SARSA (Rummery et al., 1994) and Q-learning (Watkins, 1989). These algorithms are based on finding an optimal value function which can then be used to determine optimal policy that maximizes the reward that the agent receives from the environment. Value function can be defined as a measure of "goodness" or "badness" of some state of the environment. There are two basic types of value functions, namely the state-value function $V_\pi(s)$ and the action-value function $Q_\pi(s,a)$. The state-value function $V_\pi(s)$ can be defined as an expected cumulative discounted reward (return) from some state s and then following a policy π. The action-value function $Q_\pi(s,a)$ can be defined as an expected cumulative discounted reward (return) from some state s, when our agent took an action a and then followed a policy π. Optimal state-value function $V^*(s)$ and action-value function $Q^*(s,a)$ can simply defined as a maximum value function over all policies.

$$V^*(s) = \max_\pi V_\pi(s) \tag{5}$$

$$Q^*(s,a) = \max_\pi Q_\pi(s,a) \tag{6}$$

Optimal policy can be determined from the action-value function by simply selecting an action with highest action-value function.

$$a^* = \operatorname*{argmax}_{a \in A} Q^*(s,a) \tag{7}$$

Policies themselves can be divided into deterministic and stochastic policies. In deterministic policy $\pi(s)$, the agent always takes the same action in a given state. In stochastic policy $\pi(a|s)$ the agent takes certain action in the same state with some probability.

$$\pi(s) = a \tag{8}$$

$$\pi(a|s) = p[A_t = a|S_t = s] \tag{9}$$

In most domains like for example control of computer game agents, robotic control or self-driving cars, there is a large number of states which prohibits the use of value function representations by traditional means like for example lookup tables. Instead we have to rely on approximate solutions using value function approximators with some parameters w. Considering that artificial neural networks are currently the state-of-the-art models in many domains, they are ideal models for value function approximation.

Deep Q-learning algorithm developed by Mnih et al. (2013) allows us to train artificial agents in complex environments by combining RL principles with value function approximation using CNNs. The architecture of Deep Q-learning network is illustrated in Figure 8. Using these methods the authors were able to learn successful policies for most of the 50 Atari 2600 games they tested directly from the pixel representation of the game states with changes in the score of the game acting as a reward signal. They used Q-learning update rule together with CNN for action-value function approximation. They also introduced the so-called experience replay mechanism that is used for stabilizing the learning. Experience replay stores a tuples of states, actions and rewards from previous time steps during the training in the memory and during very weight update it randomly samples past transitions and uses them as learning data for value function approximator. It stabilizes the learning procedure by removing correlations between the training data. Deep Q-learning also further stabilizes the learning by using secondary CNN to approximate target action-value function used in the weight updates of primary action value-function approximator.

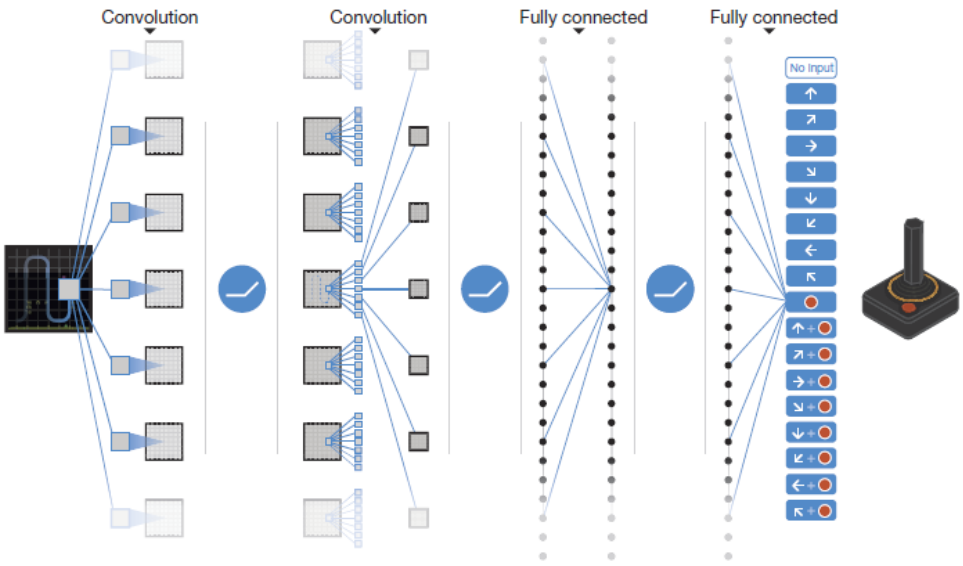

Figure 8. Neural network architecture used by Deep Q-learning based on convolutional neural network. Input to the network consists of 84x84x4 pixel array that contains 4 most recent game screens and the output represents potential actions that the agent can execute in the environment (Mnih et al., 2015).

There are several variants of Deep-Q learning that improve the performance of this algorithm. This includes Double Deep-Q learning (Van Hasselt et al., 2016) which addresses the tendency of the Deep-Q learning to overestimate Q-values, or Deep-Q learning with prioritized experience replay which can sample more “important” tuples of states, actions and rewards more frequently (Schaul et al., 2015). Algorithms based on Deep Q-learning have been

recently applied to much more complex domains than simple Atari games. Deep Q-learning model developed by Lample et al. (2017) have been able to learn successful policy that exhibited performance similar to human players in 3D video game Doom which is partially observable in nature. A model that is a combination of neural network value approximation model with Monte Carlo tree search was used to beat top human players in a game of Go for the first time (Silver et al., 2017).

The main disadvantage of algorithms based on Deep-Q learning is that they are not directly applicable to domains with continuous action spaces control of the limbs of autonomous robots or self-driving car control. In domains with continuous action spaces we need to separately approximate the policy and the value of the action-state pair. This kind of algorithm is called the actor-critic algorithm. The actor part of this algorithm selects an appropriate action based on the current state of the environment. The critic part evaluates the value of the current state of the environment and the current action selected by the actor. Actor-critic algorithm and neural network approximators were combined in the work by Lillicrap et al. (2015). They used Deterministic Policy Gradient update rule (Silver et al., 2014) to update the actor network and Q-learning update for critic network. This architecture managed to successfully learn policies in wide range of continuous control tasks such as vehicle control, monopoded balancing task, reaching task or cart-pole swing-up task.

1.5 Generative models based on neural networks

So far we discussed various variants of discriminative models that can classify, select an action or make certain inferences from the data. Unlike discriminative models, generative models role is not to model a dependence between the input data and some target data, but to generate novel instances of data that are from the same distribution as the training data. Such models have numerous applications in the realm of artificial intelligence that include image generation and modification, simulating models of agent's environment in reinforcement learning that can be used for planning or generation of new training data for improving classification models.

There are several generative models based on artificial neural networks. Some of the more popular models include Deep Boltzmann Machines (Salakhutdinov et al., 2010), Variational Autoencoders (Kingma et al., 2013), PixelRNN (Oord et al., 2016) and Generative Adversarial Networks (Goodfellow et al., 2014). We will focus on Generative Adversarial Networks (GANs) due to the quality of samples they produce that is superior to other generative models

as well as due to fewer architectural limitations and greater flexibility when designing practical models. Also unlike models like PixelRNN they can generate samples in one forward pass of the network which lowers computational requirements of this model.

The basic principle of GANs is to create a game scenario between two artificial neural networks called the generator and the discriminator. In this game the generator $G(z)$ takes as an input a vector of latent variables z from some distribution and tries to generate samples from the same distribution as training data. On the other hand, the discriminator $D(x)$ takes as the input the samples from the generator x and classifies them as either "true", if the discriminator thinks that the samples come from the same distribution as training data, or "fake", if the discriminator thinks that the samples come from the generator distribution. In this setting the discriminator tries to maximize its capacity to tell apart samples from real distribution and samples from generator distribution. On the other hand, the generator tries to produce such samples from its distribution that minimize discriminator's ability to tell them apart, in other words the generator learns to "fool" the discriminator by generating data that are similar to "real" data.

The training process of GANs consists of two simultaneous gradient updates to both generator and discriminator. First, two batches of training data are sampled from the training dataset and from the generator. Usually there are the same number of samples in both batches. When the discriminator receives the batch containing samples from the training dataset the goal of the discriminator is to classify all samples as belonging to class "real", therefore the output of discriminator $D(x)$ should be close to 1. When the discriminator receives the batch of samples from generator the goal of the discriminator is to classify all samples as belonging to class "fake", therefore the output of the discriminator $D(G(z))$ should be close to 0. At the same time the goal of the generator is to produce such samples that are hard to distinguish from samples from the training dataset, i.e. the generator tries to push the output of the discriminator $D(G(z))$ to be close to 1. The cost function that is used for updating the parameters of discriminator $J(D)$ can therefore be expressed the in following way, which is a standard cross-entropy cost function for binary classification problem with one sigmoid output unit.

$$J(D) = -\frac{1}{2}E_x \log D(x) - \frac{1}{2}E_z \log(1 - D(G(z))) \tag{10}$$

The cost function used for updating the parameters of generator $J(G)$ can then be expressed the following way.

$$J(G) = -\frac{1}{2}E_z \log D(G(z)) \tag{11}$$

The game interpretation of this training procedure is that the generator maximizes of log-probability of the discriminator being mistaken which can only be achieved by the generator producing samples that are undistinguishable by the discriminator from samples from the training data. Figure 9. illustrates the training procedure for GANs.

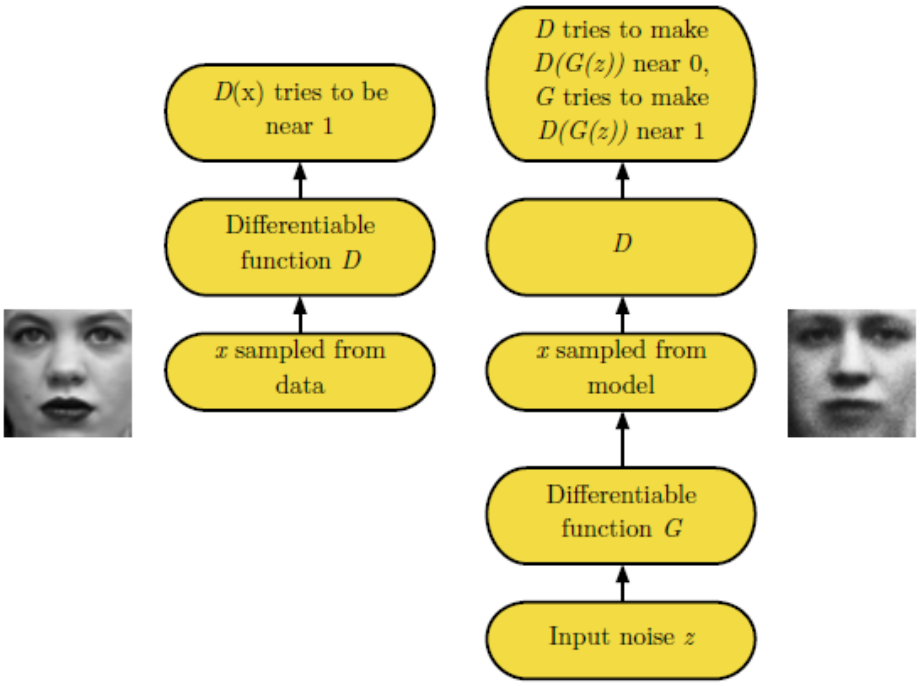

Figure 9. Diagram of the training procedure for GANs (Goodfellow et al., 2016).

GANs have many different applications, particularly in the domain of image generation and processing. Figure 10. displays example outputs of GAN-based image generation model conditioned on text descriptions.

Figure 10. Examples of images generated from text descriptions by StackGAN++ (Zhang et al., 2018).

Most of the image generation models are based on DCGAN architecture (Radford et al., 2015) which uses CNN for discriminator and deconvolutional neural network (Zeiler et al., 2010) for generator. Basic GANs don't have an ability to generate class-specific instances. The architecture of GANs was since then improved in such a way that almost photorealistic images can be generated. One example is Plug and Play GAN model by Nguyen et al. (2017). Image generation conditioned on some other variables besides the latent variables z is also possible. Auxiliary Classifier GAN (Odena et al., 2017) use GAN conditioned on image labels to generate specific classes by the generator. Conditional model by Isola et al. (2017) uses GAN conditioned on images themselves to translate between different representations of images, for example between aerial photos to maps or from black and white images to colored images. Similar model by Ledig et al. (2017) was used for upsampling low resolution images to high resolution photographs with much higher accuracy than traditional interpolation methods. Finally StackGAN++ model (Zhang et al., 2018) uses multistage GAN conditioned on text description of objects to generate near photorealistic photographs of these objects.

2. Strategies for increasing label efficiency of artificial neural networks

2.1 Transfer learning and data augmentation strategies

Transfer learning and data augmentation are probably the most commonly used strategies for increasing label efficiency of artificial neural networks, particularly in the supervised learning setting. It is common both in the research practice and real-world applications to incorporate these strategies in the neural network training pipeline.

Transfer learning is a process of adapting neural network predictive function trained using data from source domain to data in target domain (Weiss et al., 2016). This strategy works very well for target domains that share common low-level structure with the source domain. As an example, we can consider the problem of adapting image classifier trained on a subset of natural image data (for example pictures of cats and dogs) to a new set of natural image data that the classifier was not trained on (for example pictures of horses). Both domains share large number of commonalities, for example the presence of fur, 4 legs and two eyes. Also, all natural images contain contrasting edges. Neural network models such as Convolutional Neural Networks have been shown to learn feature detectors that are sensitive to low level features such as edges of various orientations or changes in color (Krizhevsky et al., 2012) as well as more advanced features such as the presence of body parts or even whole objects (Mordvintsev et al., 2015). Therefore, the goal of transfer learning is to utilize these learned feature extractors and adapt them to the new task. This can be achieved in two main ways. First, we can freeze the parameters of source domain predictor and retrain selected parts of this predictor on data from target domain. Second, we can use pretrained parameters as a starting point and fine-tune all parameters on the target task.

The efficiency of transfer learning in the context of deep neural network models was demonstrated by Oquab et al. (2014) where deep convolutional neural network was pretrained using ImageNet 2012 dataset (Russakovsky et al., 2014). Then the convolutional layers were frozen and output layers were replaced by task specific layers trained using dataset from different modality with relatively small amount of data per class. The classification accuracy on a new modality was substantially higher when they used the pretrained network as opposed to training the whole network from scratch. This method is now standard when training neural network classifier on natural images when large number of training data is not available. This

method is also applicable in other tasks in the domain of natural image processing such as image segmentation (Long et al., 2015), object detection (Girshick et al., 2014) or object pose estimation (He et al., 2017).

Overfitting, i.e the phenomena when the neural network model achieves good performance on training part of a dataset but shows worse performance on testing part of the dataset, is one of the main limiting factors in neural network training process. Mismatch between training data distribution and test data distribution is one of the main causes of overfitting. For example, in the domain of natural images there might be mismatch between the presence of features of objects (red car in test set is not present in train set), position of object (object is centered in the train set but not in test set), lighting of objects (objects in train set are lit by artificial light, objects in test set are lit by natural light) or rotation of object (digits are not rotated in train set, but are in test set). These and many other kinds of mismatches are particularly problematic when the size of training dataset is small or contains a small variability in data (for example as a result of limited data acquisition process), which increases the probability that the data in the train set will not represent the variability of data in the test set. Data augmentation aims to expand the variability of training data without acquiring additional train data, with the aim to increasing the performance of the neural network model. Traditional data augmentation techniques include geometric transformations, color space augmentation, random noise augmentation or random image parts erasure. More advanced data augmentation techniques include synthetic data augmentation by generative neural network models like Generative Adversarial Networks.

Geometric transformations are some of the most widely used techniques to expand variability of training data in the image domain. This family of data augmentation techniques include random cropping, random flipping, random rotation, random translation or more complex transformations like shear mapping or aspect ratio change. Random cropping augments training data by selecting random crop (usually using a fixed aspect ratio) from original training image. Random translation shifts training images up-down and left-right. Random cropping and random translation are effective augmentation strategies in situations where there are object scale and object position biases present in the training data. Random flipping flips the training images around vertical or horizontal axis with randomly with certain probability, which can alleviate horizontal and vertical object orientation bias in training data. Random rotation rotates train images by a random angle which makes the neural network more invariant to object rotation in the absence of unbiased training data. In their seminal work Krizhevsky et al. (2012) showed that the combination of a random image cropping and

horizontal flipping can substantially reduce overfitting. Some combination of geometric transformations are now standard in training large neural network models on image based datasets. Taylor et al. (2017) studied the impact of various types of geometric transformations and found that random flipping, random rotation and random cropping have positive impact on the training of neural network image classifier, with random cropping having a significant positive impact on final accuracy of the tested neural network classifier.

Color space augmentations change the properties of RGB channels during training. Among these augmentations are operations such as color space changes, color temperature changes, color channels dropping, hue changes or saturation changes. These operations aim to reduce color biases in training data. Similarly, random noise augmentation changes pixel values by adding noise (typically from gaussian distribution) to original pixel values, which might help to increase the invariance of feature detectors to changes in various properties of natural images such as texture or color pattern changes. Random image parts erasure works by selecting a small patch or multiple patches from the training images and erasing or replacing the image data in that patch with random noise. This approach was shown to improve the accuracy of neural network models for image classification and object detection (Zhong et al. 2020). Related to random image part erasure is background augmentation method. This method isolates the object of interest in the image (using masking) and replaces the background with a randomized picture. Background augmentation is usually used in robotics setting in order to increase the model invariance to the background change. This is advantageous when robotics model, for example task policy or object pose estimation model, is trained in virtual environment that is visually distinct from test environment (real-world robotics setup). An example of this approach is the work by Tremblay et al. (2018) that uses background augmentation to increase the performance of object pose estimation model trained in the virtual environment.

Synthetic data augmentation takes a more principled approach than previously mentioned methods. The aim of these methods is to train and data generation or data augmentation model that produces data points that are close to distribution of training data and thus expanding the training dataset. DAGAN framework (Antoniou et al., 2017) trains generative adversarial network that outputs augmented versions of training samples and uses these samples to train neural network classifiers. Generative adversarial networks can be considered an unsupervised learning model, in that they do not need class labels to generate new examples from the target distribution. Therefore, DAGAN framework requires a large unlabeled part of the dataset to learn augmentation function, unlike previously mentioned methods. In a training regime with a small amount of labelled training data this augmentation method substantially improved the

performance of the tested image classifier compared to standard image augmentation methods. Using additional training examples generated by DAGAN also improved the performance of a few-shot learning model.

Synthetic data augmentation has also been successfully applied to robotics setting. For example, Randomized-to-Canonical Adaptation Networks (James et al., 2019) combine background augmentation method with a conditional GAN to generate canonical representation of the robotics setup. The authors found that conditional GAN trained on background augmented images can successfully generate canonical representation even from real world robotics setup. This enables the training of robot grasping policy in a simulator environment with abundance of training data and transfer the trained policy to real world environment without extensive fine-tuning.

2.2 Few-shot learning strategies

In the previous section we explored strategies that are aimed to either expand the limited amount of training data or to exploit pretrained models in order to either lower the amount of data needed to train neural network models or to increase the overall accuracy of neural network models. Here we will explore few-shot learning strategies specifically designed to make learning only from few examples possible.

Typically in the image classification setting we have a dataset D of many examples per class that we use for training and testing the classifier. In order to test the generalization performance of a classifier we divide the dataset by class into training subset D_{train}, that is used for updating the model parameters and D_{test} that is used for testing the generalization performance. In the few-shot learning scenario we usually train the model using some meta dataset D_{meta} that is divided into $D_{meta-train}$ and $D_{meta-test}$ training subsets. Both $D_{meta-train}$ and $D_{meta-test}$ contain multiple datasets D, each of which can also be divided into example and testing subsets. The neural network-based classifier in this setting takes as an input individual datasets D from $D_{meta-train}$ and learns correct classification mechanism for every class in D in a way that generalizes well to $D_{meta-test}$ subset with different classes unseen during the training. In the few shot learning setting each dataset from D_{meta} contains k training examples per class and N classes, k usually being a very small number between 1 and 5. The number of classes in each dataset can vary, but due to the difficulty of the problem the number of classes is usually between 5 and 20. Datasets are assigned to $D_{meta-train}$ and $D_{meta-test}$ by

random selection with a condition that none of the classes in $D_{meta-train}$ are also in $D_{meta-test}$. General few-shot learning procedure can thus be summarized in the following way:

- randomly assign classes to train and test subsets,
- construct $D_{meta-train}$ - create T datasets by randomly sampling (without replacement) N classes from training subset for each dataset,
- construct $D_{meta-test}$ - create S datasets by randomly sampling (without replacement) N classes from test subset for each dataset,
- for every dataset in $D_{meta-train}$ randomly select k training examples per class
- update the weights of the model using training examples from $D_{meta-train}$
- test the classification accuracy of the model by averaging the accuracy on datasets from $D_{meta-test}$.

In the few-shot learning scenario, defined in the previous paragraph, we want our ANN model to learn a function that maps any possible dataset from some domain to any possible labels corresponding to individual classes from that dataset. This requires a dataset with different characteristics than datasets for traditional classification tasks. Specifically a dataset with a large number of classes and a small number of examples per class is preferable as opposed to the dataset with a small number of classes and large number of examples per class. The standard dataset for training and testing few-shot learning models is the Omniglot dataset (Lake et al., 2011). This dataset was nicknamed the "transpose" of MNIST because it contains 1623 classes with 20 examples per class as opposed to MNIST dataset that contains 10 classes with 6000 examples per class. Omniglot dataset consists of pictures of handwritten characters from various alphabets around the world. Each class represents one character from one of the alphabets with each example handwritten by a different subject. Figure 11. displays examples from the Omniglot dataset.

Figure 11. Examples of images from Omniglot dataset (Koch et al., 2015).

Another dataset often used for few shot learning is the miniImageNet dataset (Vinyals et al., 2016) which contains a subset of ImageNet dataset with 100 classes and 600 examples per class. This dataset is much more challenging than Omniglot dataset due the nature of images which in the case of miniImageNet are colour photographs of objects in the real world.

In subchapters 2.2.1-2.2.7 we will discuss various approaches based on ANNs to the few-shot image classification problem. Multiple few-shot learning models were developed in recent years. We will discuss some of the most successful models for few-shot image recognition. These include Siamese neural networks, matching networks, meta-learning long short-term memory, Prototypical networks, Temporal-convolution-based meta-learner, MAML and REPTILE.

2.2.1 Siamese neural networks

Siamese neural networks (Bromley et al., 1993) consist of two subnetworks that share the same set of weights and that are connected at the output using a common hidden layer or layers. During the training each subnetwork extracts some features from their input, while the common hidden layer combines the outputs of the subnetworks. The output of a common hidden layer is then used as an input to the subsequent fully connected layers. The final output of Siamese neural network usually represents some measure of semantic distance between the pair of inputs. Figure 12. depicts a basic schema of Siamese neural network.

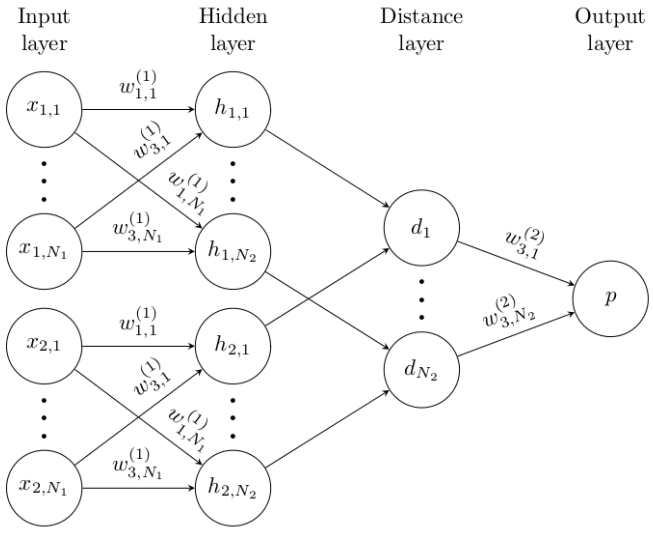

Figure 12. Schema of the Siamese neural network architecture. Distance layer refers to the common hidden layer (Koch et al., 2015).

SNNs were first applied to the few-shot classification problem by Koch et al. (2015). They used a convolutional neural network architecture for subnetworks combined with a distance layer that computed the L1 distance between the flattened output of both subnetworks. The final output neuron represents the probability that the two inputs are from the same category. Few-shot learning task is therefore reduced to a binary classification problem where half of the input pairs belongs to the same category and the other belongs to different categories. The network is trained using binary cross-entropy cost function with weight regularization. Trained Siamese neural network can then be used as a basis for few-shot image classifier. In order to classify a test image X into one of the C possible categories we are given some set of example images $\{x_c\}_{c=1}^{C}$ for every category. The number of examples per category is usually referred in the literature as "shot" and the number of classes as "way". We can then use the trained SNN to calculate the similarity score between the test image X and example images for every class. Based on the similarity score we can predict the class of the test image X by selecting a class with a maximum similarity score. The classification decision can then be defined as

$$C^* = argmax_c(f(X, \{x_c\})) \tag{12}$$

where $f(X, \{x_c\})$ is the output of the network for test image and examples for category $\{x_c\}$.

SNN few-shot learning accuracy was tested on Omniglot dataset. To test the few-shot classification accuracy of the SNN, the network had to correctly classify images into 20 possible categories with one example per class. Multiple test datasets were generated and the classification accuracy was averaged across these datasets. In this setting the SNN model achieved an accuracy of 92%. For comparison a simple nearest neighbor classifier achieved accuracy of only 21.7% and a CNN only 65.2% with one example per class. Big advantage of this approach to the few-shot learning problem is that we do not need to change the parameters of the network when we want to classify a different set of images, because the trained SNN is able to calculate a similarity score for any pair of images, even when they are from classes that were not a part of the training data.

2.2.2 Matching networks

Matching networks (Vinyals et al., 2016) is a model that was purposely built for a few-shot learning task. Similarly to SNNs, we do not need to change the parameters of Matching networks to classify images that are different from training images. Matching network training

procedure closely resembles the general meta-learning procedure. During the training we are given a support set $S = \{x_c, y_c\}_{c=1}^{C}$ that contains support images x_c and corresponding labels y_c from C possible categories. We are also given a training set $\hat{X} = \{\hat{X}_c, \hat{Y}_c\}_{c=1}^{C}$ that contains training images $\hat{X}_c$ and corresponding labels $\hat{Y}_c$ from C possible categories. During the training we want our network to learn a classifier that given any support set S can map a sample from a training set $\hat{x}$ to a corresponding correct label $\hat{y}$. More formally, we want the network to learn $P(\hat{y}|\hat{x}, S)$, where P is parameterized by a neural network model. We also want our model to generalize well on unseen training sets and support sets. Matching networks compute the $P(\hat{y}|\hat{x}, S)$ in a following way

$$P(y|\hat{x}, S) = \sum_{i=1}^{C*k} a(\hat{x}, x_i) y_i \tag{13}$$

$$P(\hat{y}|\hat{x}, S) = argmax_y(P(y|\hat{x}, S)) \tag{14}$$

where x_i is a sample from the support set S, y_i is the corresponding label a is a so called attention mechanism and y is the output vector of a network.

The attention mechanism $a(\hat{x}, x_i)$ is the core feature of matching networks. Generally $a(\hat{x}, x_i)$ computes the softmax of some distance measure between the embedding of the training image and the support image. Both embeddings are parameterized using an appropriate ANN model. Intuitively a Matching network classifier computes the distance measure between the data we want to classify and every support example. The support example that has the lowest distance between it and data we want to classify will be assumed to belong to the same category as the data. Because the labels corresponding to support examples are known, we can therefore infer the label of the data. Matching networks take two approaches for the design of the attention mechanism. One approach is to define attention mechanism using cosine distance between the embeddings of the support example and the training example. Formally the attention mechanism $a(\hat{x}, x_i)$ is defined as

$$a(\hat{x}, x_i) = \frac{\exp\left(cos\big(f(\hat{x}), g(x_i)\big)\right)}{\sum_{j=1}^{k} \exp\left(cos\Big(f(\hat{x}), g\big(x_j\big)\Big)\right)} \tag{15}$$

where cos is the cosine distance, $f(\hat{x})$ is the neural network embedding of training example and $g(x_i)$ is the neural network embedding of particular sample from the support set.

Neural network embedding functions $f(\hat{x})$ and $g(x_i)$ use architecture based on CNNs. Matching networks use cross-entropy cost function for training.

The main disadvantage of the attention mechanism is that the embeddings $f(\hat{x})$ and $g(x_i)$ are only dependent on current support example x_i. It would be beneficial if the support example embedding would also depend on other support examples. Authors therefore introduced so called fully context embeddings which used bidirectional LSTM to embed both the training example $\hat{x}$ and support example x_i conditioned on the whole support set S.

For evaluation purposes Vinyals et al. (2016) used Omniglot dataset and miniImageNet dataset. Omniglot dataset was divided to 1200 training classes and 423 test classes independent of alphabet. Classes were augmented by rotating the image by multiples of 90 degrees. In 5-way experiments Matching networks achieved 98.1% accuracy in 1-shot setting and 98.9% accuracy in 5-shot setting. In 20-way experiments the accuracy was 93.8% accuracy for 1-shot setting and 98.7% for 5-shot setting. In 5-way experiments using miniImageNet dataset Matching networks achieved 46.6% accuracy in 1-shot setting and 60.0% accuracy in 5-shot setting.

2.2.3 Meta-Learner LSTM

In their work Ravi et al. (2017) took a very different approach to the few-shot learning problem. They designed a RNN-based meta learning model that learns an optimization algorithm for updating a separate classifier. Their model therefore has two parts. The first part is called the learner that takes as an input individual training datasets from $D_{meta-train}$ and classifies individual datapoints from these datasets. The second part is called the meta-learner that takes as an input the loss and the gradients from the learner and outputs weights that should improve the performance of the learner. The weights of the meta-learner are updated based on the performance on separate part of $D_{meta-train}$ dataset. A trained meta-learner should therefore be able to output the weights for an untrained learner that maximize the performance of the learner even for the unseen classes. CNN is used for the learner and long short-term memory is used for the meta-learner.

The performance of this model was tested on miniImageNet dataset. The authors reported 43.44% accuracy in 5-way 1-shot experiments and 60.6% accuracy in 5-way 5-shot experiments. Performance of this model is therefore comparable to the performance of the Matching networks.

2.2.4 Prototypical networks

Prototypical networks by Snell et al. (2017) is an approach to the few-shot learning problem similar to Matching networks. Prototypical networks learn an embedding function that maps the input data into a high-dimensional metric space in which classification is possible by calculating the distance between query data and prototypes for every category. The embedding function is parameterized using ANN. As in Matching networks, during the training we are given a support set $S = \{x_c, y_c\}_{c=1}^{C}$ and a training set $\hat{X} = \{\hat{X}_c, \hat{Y}_c\}_{c=1}^{C}$. Then using the embedding neural network f_θ we can create a prototype for every category in a given dataset in the following way

$$c_k = \frac{1}{|S_k|} \sum_{x_i \in S_k} f_\theta(x_i) \tag{16}$$

where c_k is the prototype for the category k, S_k is the set of examples for the category k and x_i is a sample from the set of examples S_k.

The prototype for every category is therefore simply a mean of all support examples embedded by the neural network f_θ. When we have computed prototype for every class in a dataset we can classify a sample from a training set $\hat{x}$ by embedding the sample using the same neural network f_θ, calculating the distances between the sample and prototypes for every category using some distance function d and applying softmax function to those distances. The probability of sample $\hat{x}$ with label $\hat{y}$ belonging to category k can therefore be expressed as:

$$p(\hat{y} = k | \hat{x}) = \frac{\exp\left(-d(f_\theta(\hat{x}), c_k)\right)}{\sum_{j=1}^{k'} \exp\left(-d(f_\theta(\hat{x}), c_j)\right)} \tag{17}$$

Prototypical networks use CNN for embedding network f_θ together with cross-entropy cost function. For a distance measure the squared Euclidean distance was found to work best in practice as opposed to cosine distance used in Matching networks. In terms of accuracy Prototypical networks managed to overcome both Matching networks and Meta-learner long short-term memory model by Ravi et al. (2017). For testing purposes the authors used Omniglot dataset and miniImageNet dataset. In 5-way Omniglot experiments Prototypical networks achieved 98.8% (1-shot) and 99.7% accuracy (5-shot). In 20-way Omniglot experiments Prototypical networks achieved 96% (1-shot) and 98.9% (5-shot) accuracy. In 5-way miniImageNet experiments the accuracy was 49.72% (1-shot) and 68.2%.

2.2.5 Temporal-convolution-based meta-learner

Temporal-convolution-based meta-learner (Mishra et al., 2017) model is based on embedding the input data and corresponding labels into a high-dimensional embedding space. The sequence of embedded tuples of input data and labels is then used as an input to meta-learning model based on 1-D temporal convolutions with an attention mechanism. Based on these inputs the meta-learning model should find correct mappings between labels and data even when the classes of input data were not encountered during training. More formally the Temporal-convolution-based meta-learner receives a sequence of example-labels pairs $(x_1, y_1), \dots, (x_{t-1}, y_{t-1})$ that belong to classes sampled from $D_{meta-train}$. Based on these example-label pairs the network is supposed to classify an example x_t. CNN with residual connections is used for the embedding function.

In 5-way Omniglot experiments Temporal-convolution-based meta-learner achieved 98.96% (1-shot) and 99.9% accuracy (5-shot). In 20-way Omniglot experiments this model achieved 97.64% (1-shot) and 99.36% (5-shot) accuracy. In 5-way miniImageNet experiments the accuracy was 55.71% (1-shot) and 68.88%.

2.2.6 MAML

Model-Agnostic Meta-Learning model (Finn et al., 2017), or simply MAML, is a few-shot learning model that aims to train a task and neural architecture-agnostic model that can be retrained on a new task only using few examples. This is achieved by using a training procedure in which model parameters are learned in such a way that changes in parameter space using only small number of training examples from a new task results in large increase in performance. Due to task and model invariance of MAML training setup, this few-shot learning method can be applied to few-shot image classification problem as well as other image processing problems, such as image segmentation or object localization. Few-shot reinforcement learning was also successfully demonstrated by authors.

Training procedure of MAML model consists of two optimization steps that are called inner and outer loop learning steps. In the inner loop learning step a batch of tasks S_b is sampled from a distribution of tasks (e.g. multiple sets of 5 random categories are sampled from a set of possible categories), classification accuracy is evaluated using cost function C_{S_b} and starting

network parameters θ for every set, then n steps of stochastic gradient descent are performed on all tasks task resulting in a new set of parameters $\tilde{\theta}$. This step can be summarized as

$$\tilde{\theta} = \theta - \alpha \nabla_{\theta} C_{S_b}\ (f_{\theta}) \tag{18}$$

where α is the inner loop learning rate, ∇_{θ} is the derivative of the cost function w.r.t to starting network parameters θ and f_{θ} is the neural network.

In the outer loop learning step, a different set of training examples T_b is sampled for tasks in S_b. Then the classification accuracy is evaluated using cost function C_{T_b} and updated network parameters $\tilde{\theta}$. Starting network parameters θ are updated by computing gradient of the cost function C_{T_b} w.r.t to θ. Outer loop learning step can be summarized as

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{b=1}^{B} C_{T_b}\ (f_{\tilde{\theta}}) \tag{19}$$

where β is the inner loop learning rate, B is the number of tasks in the batch, and $f_{\tilde{\theta}}$ is the neural network with updated parameters from inner loop learning step.

This update effectively pushes the starting network parameters θ to a position in parameter space that is close to effective parameters (in terms of performance) for all tasks in the batch of tasks. After convergence, the final network parameters θ should be located in such a place in the parameter space, that is close to optimal parameters for all tasks from the same distribution as the training tasks. Only a few training examples for a new task are therefore needed to fine-tune the network for new task, which makes MAML a good few-shot learning model. Schematic representation of MAML training procedure is displayed in Figure 13.

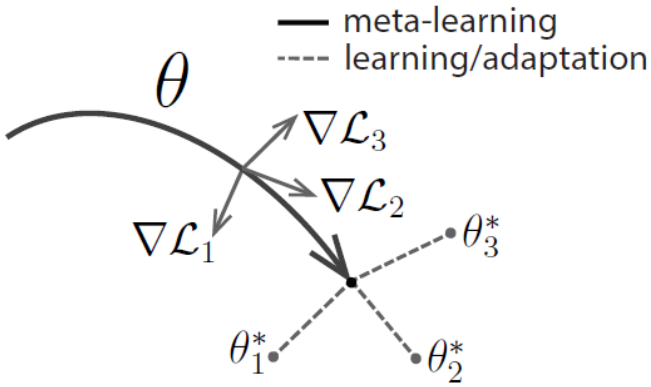

Figure 13. Schematic representation of MAML training procedure (Finn et al., 2017).

In 5-way Omniglot experiments MAML achieved 98.7% (1-shot) and 99.9% accuracy (5-shot). In 20-way Omniglot experiments this model achieved 95.8% (1-shot) and 98.9% (5-shot) accuracy. In 5-way miniImageNet experiments the accuracy was 48.7% (1-shot) and 63.11%.

The outer loop update can alternatively be expressed as:

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{b=1}^{B} C_{T_b} \left(f_{\theta} - \alpha \nabla_{\theta} C_{S_b} \left(f_{\theta} \right) \right) \tag{20}$$

From the above expression it is apparent that the MAML update requires secondary backward pass through the computational graph of the network and computation of Hessian vector products, which impacts the computational efficiency of the model compared to standard backpropagation training procedure. This can be alleviated by using first order approximation of MAML algorithm proposed by Finn et al. (2017) in the same paper, which achieves comparable performance to original MAML algorithm. Alternatively, there are algorithms that use similar principle as MAML that do not require computing second order derivatives, such as REPTILE algorithm (Nichol et al., 2018).

2.2.7 REPTILE

REPTILE algorithm (Nichol et al., 2018) belongs, together with MAML, to a family of meta-learning based few-shot learning methods. Unlike MAML, REPTILE algorithm does not calculate the second-order derivatives of a cost function which substantially lowers the computational requirements of this model compared to MAML.

REPTILE training procedure, like MAML, consists of two optimization steps. First a task is sampled from a distribution of tasks. Then n steps of stochastic gradient descent are performed on this task resulting in a new set of parameters $\tilde{\theta}$. Then starting parameters θ are moved towards the new set of parameters $\tilde{\theta}$. This procedure can be summarized in the following way.

- Initialize network parameters θ,
- for i iterations do:
    - sample task τ and compute loss L_{τ}
    - perform n steps of SGD: $\tilde{\theta} = SGD(\theta, L_{\tau}, n)$
    - REPTILE update with learning rate ε : $\theta \leftarrow \theta + \varepsilon(\tilde{\theta} - \theta)$
- end for.

The intuition behind REPTILE is that the network parameter vector θ should converge to a state that is close (in a Euclidian sense) to manifolds of optimal solutions for every possible task. In this state, only a small number of gradient steps using a small number of new training examples are required to fine-tune the network to a new task. Figure 14. depicts this intuitive representation.

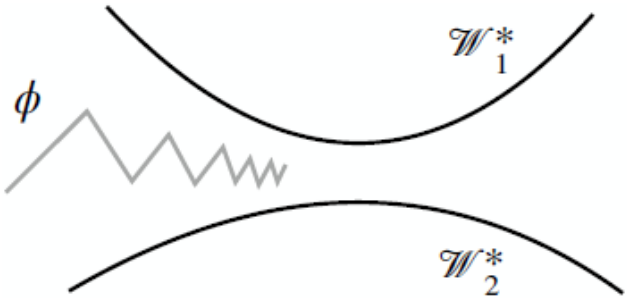

Figure 14. Schematic representation of REPTILE training procedure (Nichol et al., 2018).

The performance of REPTILE model is comparable to performance of MAML across benchmark classification tasks. In 5-way Omniglot experiments MAML achieved 97.68% (1-shot) and 99.48% accuracy (5-shot). In 20-way Omniglot experiments this model achieved 89.43% (1-shot) and 97.12% (5-shot) accuracy. In 5-way miniImageNet experiments the accuracy was 49.97% (1-shot) and 65.99%.

2.3 Semi-supervised learning strategies

Semi-supervised learning (SSL) has a similar goal as few-shot learning, namely improving the label efficiency of artificial neural networks, although the approaches taken by these two frameworks differ. In few-shot learning framework the aim is to lower the amount of labeled training data needed to train neural network model on a new task to minimum, but the tradeoff is that we need to provide a large number of training tasks with labelled data in order to train a general model that can learn a similar task quickly. Similarly to few-shot learning, the main aim of semi-supervised learning is to lower the number of labelled data needed to train a model, but instead of using fully-supervised tasks with fully labelled examples as training data, in semi-supervised learning scenario we utilize a large number of unlabeled training data to improve the performance of a model trained using a small amount of labelled data.

For many real-world problems obtaining a large number of unlabeled data from the same distribution as a smaller labelled portion of data is much easier than obtaining labelled data for the entire dataset. Image classification, image segmentation or object detection are good examples for this principle due to abundance of pictures that can be obtained on the internet. Semi-supervised learning procedure (in the classification context) can therefore be summarized in the following way:

- acquire labelled part of the dataset D_l and unlabeled part of the dataset D_u,
- assign part of the labelled data into test set, thus creating a training part of the labelled data D_l^{train} and test part of the labelled data D_l^{test},
- update the weights of the model using both the labelled part of the training data D_l^{train} and unlabeled part of the data D_u,
- test the performance of the model using test part of the labelled data D_l^{test}.

Semi-supervised learning approaches can be broadly divided into four separate categories, namely consistency regularization methods, proxy-label methods, graph neural network-based methods and generative model-based methods.

Consistency regularization methods rely on applying various perturbations to the input of the model that do not change the label of the input and then force the output of the model to stay consistent across various perturbations across the same inputs. Examples of this approach are Ladder networks (Rasmus et al., 2015), Pi model (Laine et al., 2016), Temporal Ensembling (Laine et al., 2016), Virtual Adversarial Training (Miyato et al., 2018) and Mean Teacher model (Tarvainen et al., 2017).

Proxy-label methods aim to expand the labeled portion of the dataset by labeling the unlabeled portion of the dataset using a neural network trained using a smaller labeled portion of the dataset. These predictions can be generated using a single model or by combining multiple predictions for every data example using an ensemble of different models and data augmentation techniques. Examples of methods that use this approach are Pseudo-label (Lee, Dong-Hyun 2013), Deep clustering for unsupervised learning of visual features (Caron et al., 2018), Billion-scale semi-supervised learning for image classification (Yalniz et al., 2019) and MixMatch (Berthelot et al., 2019).

Models based on graph neural networks represent the problem of a semi-supervised learning using a graph-based representation where the nodes represent the labeled or unlabeled datapoints and the edges represent the semantic similarity between the datapoints. Unlabeled

datapoints in the dataset are classified by propagating the labels from labeled nodes to unlabeled nodes or by applying classifier to node embeddings. Methods like DeepWalk (Perozzi et al., 2014), Graph convolutional networks (Kipf et al., 2016) or Graph attention networks (Velickovic et al., 2017) can be used for semi-supervised learning.

Semi-supervised methods based on generative models use models such as Generative Adversarial Networks to learn a function that approximates the generative distribution of the dataset and utilizes this function to improve classification accuracy in semi-supervised learning regime. SGAN model (Odena et al., 2016) used GAN model to learn the generative distribution of data and adds supervised component to discriminator output that predicts the category of examples from the labeled part of the dataset. Bidirectional GAN (Donahue et al., 2017) augments the GAN framework by adding an encoder component that learns to invert the output of the generator, i.e. it learns to predict the latent vector from the unlabeled examples. This latent vector contains information about the semantics of the data, including the class of the individual datapoints. This training procedure can be utilized in the semi-supervised learning setting. An example of Bidirectional GAN being used in the semi-supervised can be found in Kumar et al. (2017).

A closely related problem to semi-supervised learning is the self-supervised learning scenario. In self-supervised learning scenario, a proxy tasks are devised that are similar to the task of interest (for example image classification). A good proxy task must be devised in a way that the unlabeled training data themselves provide implicit labels needed to learn this task. A neural network model trained using this task should develop feature extractors that are useful for learning the task of interest. Self-supervised learning can therefore be used for improving the performance of fully supervised models and also in semi-supervised learning scenario. Examples of this approach are image context prediction model (Doersch et al., 2015), colorization model (Zhang et al., 2016), image rotation model (Gidaris et al., 2018), S4L model (Zhai et al., 2019), Contrastive Predictive Coding (Oord et al., 2018), Contrastive Multiview Coding (Tian et al., 2019) or SimCLR (Chen et al., 2020).

Self-supervised models are also important in robotics setting where proxy tasks are used to improve generalization capabilities and performance of agents performing tasks in virtual or real-world environments. UNREAL agent (Jaderberg et al., 2016) uses several proxy tasks such as maximizing change in pixel space or future reward prediction to increase the frequency and quantity of the feedback signal for reinforcement learning agents. Intrinsic Curiosity Module (Pathak et al., 2017) uses intrinsic reward based on maximizing the novelty of visual appearance of the environment experienced by the agent to increase the exploratory behavior of the agent

and as a result, a lower the number of training steps. Hindsight Experience Replay (Andrychowicz et al., 2017) generates proxy tasks by selecting goals and setting synthetic rewards from a database of visited environment states, which increases the frequency of rewards available to RL system and substantially lowers the number of actions needed to learn an object manipulation policy in robotic environment.

In the next sections we will take a closer look on some widely used semi-supervised learning models. In section 2.3.1 will examine three closely related semi-supervised learning methods that constitute a backbone of semi-supervised learning methods based on consistency regularization. Section 2.3.2 deals with Mean Teacher model. In section 2.3.3 we will examine one of the most influential self-supervised learning model, namely Contrastive Predictive Coding.

2.3.1 Ladder Networks, Pi Model and Temporal Ensembling

Ladder networks (Rasmus et al., 2015) use two network paths to generate predictions for unperturbed input and perturbed input. Model predictions for unperturbed inputs are used as proxy labels for the consistency cost between the outputs of the perturbed and unperturbed paths of the model. Standard classification cost is applied for the labelled part of the dataset. In addition to the output consistency cost, the more complex version of Ladder networks uses denoising function that takes as an input the feature vector from particular layer of perturbed path of the network and reconstructs the feature vector from the corresponding layer of the unperturbed path. Separate cost function is utilized to minimize the difference between denoised feature vectors and feature vectors from the unperturbed path.

The Pi model (Laine et al., 2016) is similar to Ladder networks in that it uses category preserving perturbations of the same input. Unlike Ladder networks the Pi model applies two sets of perturbations to the input. Two different dropout conditions are also applied to both paths of the model. Consistency cost in the form of mean squared error is then applied to the outputs of both prediction paths. The strength of the consistency cost, relative to cross-entropy cost computed from labelled examples is controlled by time-dependent weight parameter that ramps up from zero as the training progresses. Pi model cost function can be expressed as

$$C(\theta) = \frac{1}{m}\sum_{j}^{m}[-\log P_f(y_j|x_j,\theta,\eta)] + w_t\frac{1}{n}\sum_{i}^{n}\|f(x_i,\theta,\eta) - f(x_i,\theta,\eta')\|^2 \qquad (21)$$

where f denotes the neural network model, θ are the parameters of the neural network model, η is the noise applied to the model input, η' is the different noise applied to the model input, x_i is the input in the batch with index i, n is the number of examples in the unlabeled batch, m represents the size of supervised batch, y_j represents the label for the input x_j and j represents the index of the labelled input. Figure 15. displays the schematic representation of Pi model training procedure.

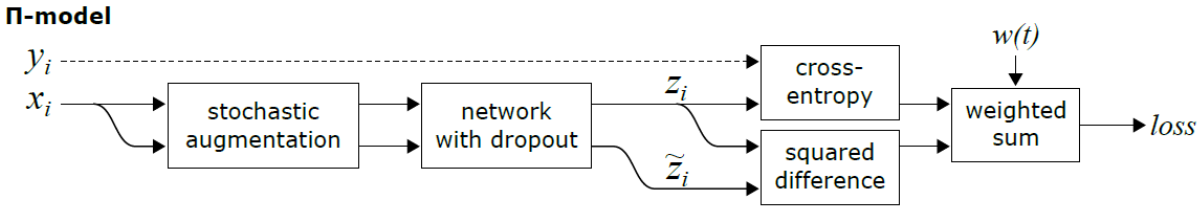

Figure 15. Pi model training procedure (Laine et al., 2016).

Temporal ensembling (Laine et al., 2016) was introduced in the same paper as the Pi model and can be thought as a variation of the Pi model. Like in the Pi model the input data is perturbed using label preserving augumentations. The main difference between the Pi model and Temporal ensembling lies in the way the target values for consistency cost are generated. In Temporal ensembling the model is not evaluated two times for different input perturbations, but only once and the target values for consistency cost are ensembled by tracking exponential moving average of models past predictions. Because at the start of the training the exponential moving average will be biased to zero (since the predictions are initialized to zero vector) this is corrected by bias correcting factor inspired by Adam optimizer (Kingma et al., 2014). Then the consistency cost is evaluated using these exponential moving averages as the target value. Because the targets were generated by the model at different time steps during the training, the target will be less noisy than in the Pi model, which might lead to more stable training and subsequently to better performance, which was demonstrated in Laine et al. (2016).

The loss function for Temporal ensembling can therefore be expressed as

$$C(\theta) = \frac{1}{m}\sum_{j}^{m}[-\log P_f(y_j|x_j,\theta,\eta)] + w_t\frac{1}{n}\sum_{i}^{n}\|f(x_i,\theta,\eta) - \tilde{z}_i\|^2 \qquad (22)$$

where target values for input x_i are denoted as $\tilde{z}_i$.

Target values $\tilde{z}_i$ are calculated by accumulating past model predictions for given example, which can be expressed as

$$Z_i \leftarrow \alpha Z_i + (1-\alpha) f(x_i, \theta, \eta) \quad (23)$$

$$\tilde{z}_i \leftarrow Z_i / (1-\alpha^t) \quad (24)$$

where Z_i represents accumulated predictions for input x_i, α represents parameter that controls the ensembling momentum, and t represents the index of current epoch of training. Temporal ensambling training procedure is displayed in Figure 16.

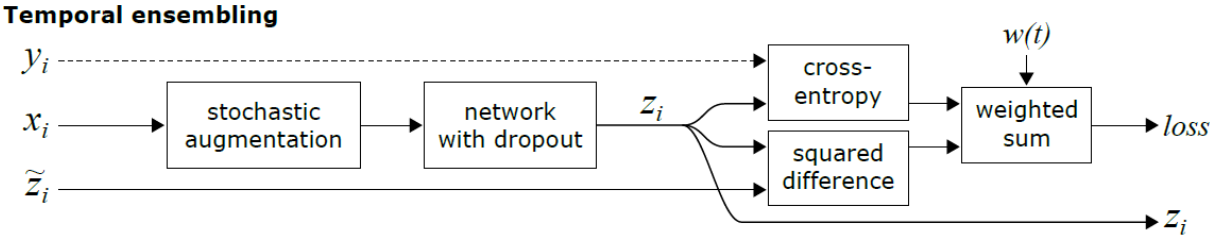

Figure 16. Temporal ensambling model training procedure (Laine et al., 2016).

2.3.2 Mean Teacher model

The Mean Teacher model (Tarvainen et al., 2017) leverages a combination of various semi-supervised learning techniques to dramatically improve generalization capability in the semis-supervised learning context. First, like Ladder networks, Mean Teacher predicts the class of the unsupervised data points using two sets of noise applied to unsupervised data points. Noisy batches are evaluated using two neural networks called the student and the teacher. Predictions for every data point in each of the noisy batches should be consistent between the student and the teacher, which is achieved using a consistency cost between student and teacher predictions during training. Consistency cost can be expressed as

$$J(\theta) = \frac{1}{n} \sum_i^n \| f(x_i, \theta', \eta') - f(x_i, \theta, \eta) \|^2 \quad (25)$$

where f denotes the neural network model, θ are the parameters of student network, θ' are the parameters of the teacher network, η' is the noise function applied to teacher input, η is the noise applied to student network, x_i is the input in the batch with index i and n is the number of examples in the unlabeled batch.

Mean Teacher uses mean squared error as the consistency cost, although different cost functions, such as cross-entropy are also viable. In addition to consistency cost, supervised cost is also evaluated for labeled portion of the input batch. Supervised cost can be expressed as

$$S(\theta) = \frac{1}{m}\sum_{j}^{m}[-\log P_f(y_j|x_j,\theta,\eta)] \tag{26}$$

where m represents the size of supervised batch, y_j represents the label for the input x_j and j represents the index of the labelled input.

Consistency cost in the Mean Teacher model effectively acts as a regularization technique that enforces the outputs of the model to be consistent across similar data points, which leads to improvement in generalization capabilities of the model. Because the consistency cost uses self-generated targets and because in the semi-supervised setting there are usually many more unlabeled examples than labelled examples, during the training the consistency cost might come to dominate the training process. Mean Teacher model mitigates this problem by introducing dynamic consistency cost weight that balances the contribution from consistency cost and supervised cost. Total loss optimized in the Mean Teacher training procedure can be expressed as

$$Loss(\theta) = S(\theta) + w_t J(\theta) \tag{27}$$

where w_t is the consistency cost weight.

Mean Teacher also incorporates model parameters ensembling technique similar to the PI model. In order to improve targets produced by the teacher network, the parameters of the teacher network are set as an exponential moving average of the student model. Model parameter ensembling procedure can be expressed as

$$\theta'_t = \alpha\theta'_{t-1} + (1-\alpha)\theta_t \tag{28}$$

where α represents parameter that controls the ensembling momentum. Figure 17. displays Mean Teacher training procedure.

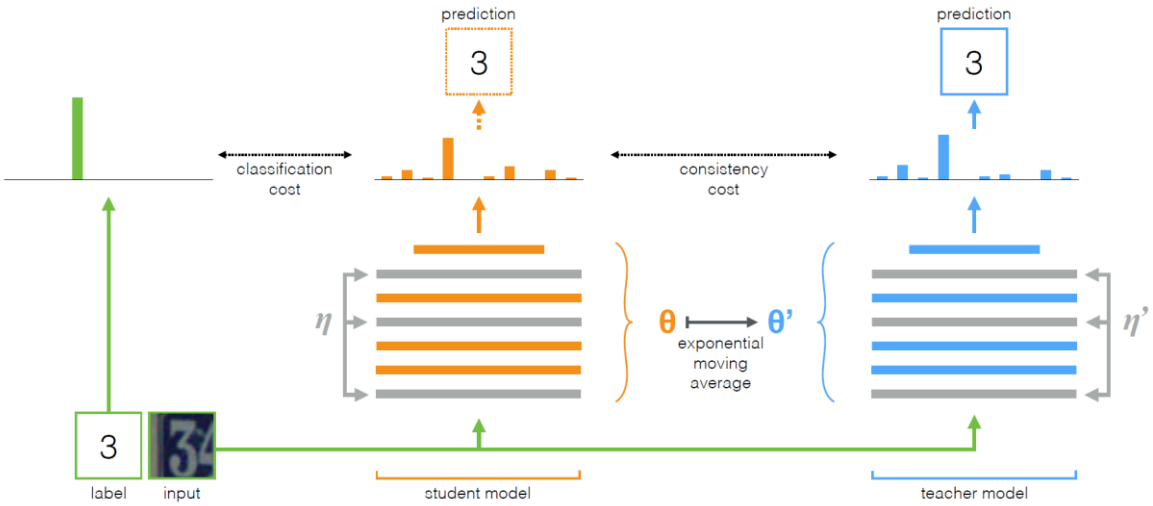

Figure 17. Mean Teacher model training procedure (Tarvainen et al., 2017).

2.3.3 Contrastive Predictive Coding

Contrastive Predictive Coding (Oord et al., 2018) is a self-supervised model that uses spatial structure of images to pretrain a neural network feature extractor that is useful in both fully-supervised setting as well as in semi-supervised setting. Henaff et al. (2020) further adapted Contrastive Predictive Coding to semi-supervised image recognition. Using a very small number of labelled examples (less than 20 per class), the pretrained feature extractor combined with relatively small output classifier is able to match the performance of fully supervised AlexNet model on the ImageNet dataset.

Contrastive Predictive Coding achieves this level of performance by utilizing spatial prediction task trained using contrastive loss. More specifically, in a self-supervised training phase each training image is divided into several overlapping image patches $x_{i,j}$, where i and j are the indices of the image patch. Then a feature vector $z_{i,j} = f_\theta(x_{i,j})$ is generated for every image patch using a neural network-based feature extractor f_θ. A fully convolutional neural network $g_{context}$ is then applied to resulting feature vector map. This network has a masked input that only takes into account image patches that lie directly above and below the feature vector of interest. The spatial prediction task is then based on predicting the feature vector of an image patch separated from the image patch of interest by some spatial gap k. The prediction can be expressed as

$$\hat{z}_{i+k,j} = W_k g_{context}(z_{i,j}) \tag{29}$$

where $\hat{z}_{i+k,j}$ is the prediction of patch separated by spatial gap k and W_k is the prediction matrix. To increase the quality of resulting features augmentation techniques such as color channel dropping, random flips and random crops are applied to image patches. Contrastive loss is used in the self-supervised training phase to estimate the quality of the spatial predictions. More specifically the distance between the feature vector of the patch of interest $z_{i,j}$ and the predicted feature vector of spatially distant patch $\hat{z}_{i+k,j}$ is compared to the distance between the patch of interest and feature vectors of patches randomly sampled from other images in the dataset z_l using cross-entropy loss. The distance between patches from the same image $\hat{z}_{i+k,j}^T z_{i,j}$ should be lower than distances between patches from the separate images $\hat{z}_{i+k,j}^T z_l{}'$. Therefore the contrastive loss can be expressed as:

$$C_{CPC} = -\sum_{i,j,k} \log \frac{\exp\left(\hat{z}_{i+k,j}^T z_{i,j}\right)}{\exp\left(\hat{z}_{i+k,j}^T z_{i,j}\right) + \sum_l \hat{z}_{i+k,j}^T z_l{}'} \tag{30}$$

After the self-supervised training phase the feature vectors $z_{i,j}$ are used as an input to a fully supervised output classifier. The output classifier is then fine-tuned using a small number of labelled images using cross-entropy cost function.

2.4 Label-efficient neural networks at scale

In sections 2.1, 2.2 and 2.3 we introduced some of the most widely used methods for increasing label efficiency of artificial neural networks. These methods, although successful, can only be applied to problems where there are numerous examples of similar problems, as in few-shot learning problems, or a large number of unlabeled examples for certain problem, as in semi-supervised learning problems. If fully supervised training of artificial neural networks could be characterized as a problem specific method, few-shot learning methods and semi-supervised learning methods can therefore be specified as domain-specific methods. Neural network models that can easily be adapted across different domains, could substantially increase the cognitive flexibility of current machine learning systems and thus decrease the number of domain-specific models and datasets needed to create a complex machine learning pipelines. This is especially important in robotics where an artificial agent can encounter a vast number of different types of problems that are difficult to anticipate beforehand. More general models are therefore required for the creation of artificial agents that can effectively operate in complex real-world environments.

User-created content on the internet provides vast amount of training data. The problem with user-created content, like blog posts, videos or public forum posts, is that most of this data is unlabeled, which implies that learning setup based on highly general self-supervised task needs to be applied. In the natural language processing domain GPT-2 model (Radford et al., 2019) aims to exploit abundant user generated content to learn a general-purpose language model that solves multiple language tasks without fine-tuning. A language model can be formalized as a self-supervised prediction task where given a sequence of language tokens, such as words, the model predicts next token in the sequence. More formally, given an example sequence x composed of language tokens $(s_1, s_2, \dots, s_n)$ the language model predicts the probability of every token in the sequence given previous tokens in the sequence which can be expressed as a product of conditional probabilities.

$$p(x) = \prod_{i=1}^{n} p(s_n|s_1, s_2, \ldots, s_{n-1}) \quad (31)$$

GPT-2 uses Transformer network architecture for its language model architecture. The main change compared to previous Transformer-based architectures is its unprecedented number of parameters with up to 1.5 billion weights and 48 layers. Smaller versions of this architecture were also tested in order to approximate the impact of the size on the performance of this model. Training dataset contained more than 40 GB of text data, mainly sourced from online massage board Reddit. The authors then tested resulting language model on several language-based tasks, including common sense reasoning task like Winograd Schema Challenge (Levesque et al., 2012), reading comprehension, summarization, translation and question answering. For every task separate standardized dataset was used. GPT-2 model achieved state of the art results in language modelling task it was directly trained on as well as in some other tasks that it was not explicitly trained on, like for example Winograd Schema Challenge. Special prompts were used to bias the network to perform specific tasks it was not trained on. For example although the network was not explicitly trained on text summarization, simply adding a text of an article, followed by "TL;DR:" as an input to the model, the model was able to output short summary of said article. Another example is the translation between languages where when adding several inputs in the form "*sentence in language 1 = sentence in language 2"* followed by "*prompt target sentence in language 1 ="* the model was able to translate the target sentence. The authors argue that this behavior emerged as a consequence of the size of the dataset and generality of the language modelling task, where if the network "wants" to maximize the performance on this task, it has to learn to recognize all the various subtasks that are prevalent in the training corpora and solve them. Then to perform a few-shot learning task, the user just needs to find correct prompt that signals the network the task user wants it to perform. In this setting the number of prompts that are used as an input to the model is analogous to number of training examples in a traditional few-shot learning setting. The authors also found that substantially increasing the number of parameters of the model improves the few-shot learning performance of GPT-2, which hints at further improvements in few-shot learning performance of neural-network models that can be realized just by increasing the number of parameters of the model.

GPT-3 model (Brown et al., 2020) expands on the ideas explored in the GPT-2 paper. In this work the authors expanded both the number of parameters of the Transformer language model and the size of training dataset. The number of parameters of the model was expanded

from 1.5 billion parameters to 175 billion parameters to further explore the impact of the size of the model to a few-shot learning performance. The training dataset was expanded from 40 GB of text data to 570 GB of test data. The number of tasks that was evaluated also expanded. In addition to tasks that have been used in GPT-2 paper, in this work tasks such as common sense reasoning about physical world, arithmetic operations, word scrambling and manipulation task, SAT analogy task or a novel word usage task were added. On multiple tasks including common sense reasoning tasks, language translation to English or open domain question answering task GPT-3 either approached or surpassed performance of state-of-the-art models that were fine-tuned on a particular task. Figure 18. displays the performance of the GPT-3 model across 42 language benchmarks as a function of the number of model parameters. The authors also tested whether text generated by GPT-3 can fool human judges to think that the text was generated by human being by generating short news articles by the model. On average human judges were able to distinguish between human generated articles and GPT-3 generated articles with 52% accuracy which is very close to random chance. The authors also tested the impact of the model size on few-shot learning performance and confirmed the continual scaling of the few-shot learning performance when the number of trainable parameters is increased.

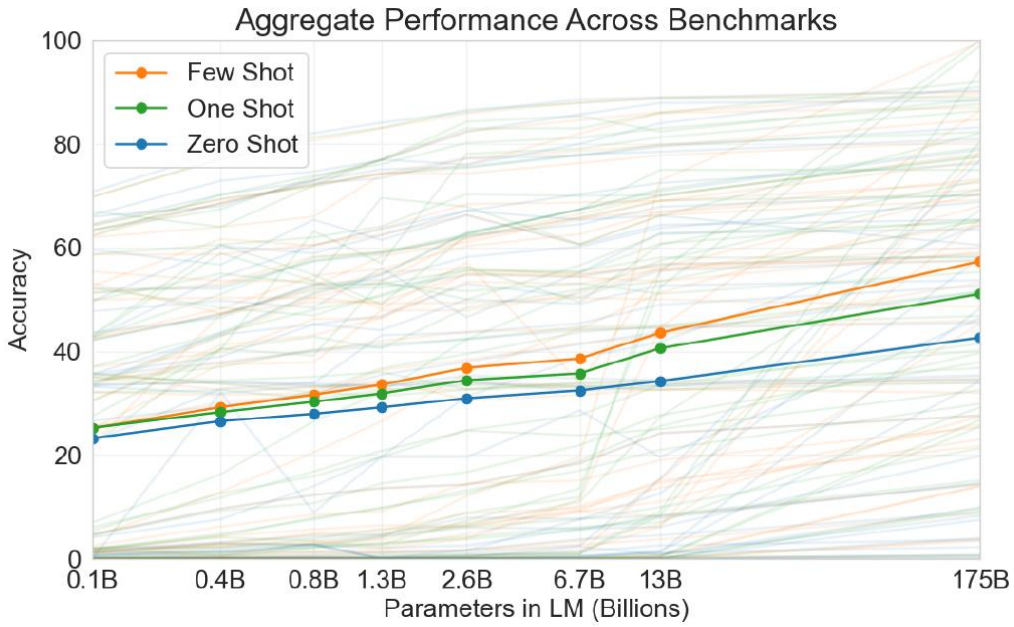

Figure 18. GPT-3 few-shot learning accuracy across 42 language benchmarks as a function of the number of parameters (Brown et al., 2020).

DALL-E model (Ramesh et al., 2021) extends the principles explored in GPT-2 and GPT-3 papers to the domain of text-to-image generation. Instead of just predicting the next token in

a sequence of text tokens, this model uses a combination of Variational autoencoder (Kingma et al., 2013) combined with transformer-based autoregressive model to generate novel images based on the text description. DALL-E uses a large-scale model with 12 billion parameters combined with 250 million text-image pairs collected on the internet. The training procedure consists of two main steps. First a Variational Autoencoder is trained to compress input images to a 32x32 grid of discrete image tokens. This 32x32 grid can be described as a "visual codebook" that encodes the content of the image into discrete one-hot vectors that can be used to reconstruct given image. In a second step image tokens are combined with text tokens and the transformer model is used to predict combined text-image tokens is similar fashion to GPT models. Trained transformer model can then predict most probable sequence of image tokens from starting text-image token. Resulting image tokens are then used as an input to pretrained Variational Autoencoder that outputs reconstructed image that should correspond to the text tokens used as an input.

DALL-E model exhibits similar ability to adapt to new tasks to GPT models. Besides generating novel images based on the content of the text description, DALL-E can do style transfer between images, manipulate various visual attributes of the images, changing pose of the object in the image, combine visual concepts in a plausible manner, incorporate geographical or temporal knowledge into generated images or even basic forms of common-sense reasoning. Example outputs of DALL-E model across 4 different tasks are shown in Figure 19.

(a) a tapir made of accordion. a tapir with the texture of an accordion.

(b) an illustration of a baby hedgehog in a christmas sweater walking a dog

(c) a neon sign that reads "backprop". a neon sign that reads "backprop". backprop neon sign

(d) the exact same cat on the top as a sketch on the bottom

Figure 19. Sample of outputs from the DALL-E model that demonstrates the ability of combining visual concepts, manipulating visual attributes of a scene and style transfer. (Ramesh et al., 2021).

Models like GPT-3 and DALL-E demonstrate that large-scale models and datasets can learn multiple tasks without any fine-tuning on these tasks or without designing special-purpose

architectures for various tasks. The main disadvantage of these models is the amount of computational resources needed to train these models which makes them impractical to train for small research institutions and commercial subjects. Also some tasks, such as arithmetic based tasks, are still hard to solve using these methods. Despite these disadvantages this approach might be very effective in learning complex cross-domain relationships within datasets. Adding more data domains like sound, environmental maps, video, or 3D objects, combined with an increasing model size might lead to a development of powerful world-models that can guide learning and decision making of general-purpose artificial agents.

3. Experimental results

3.1 Categorical Siamese Networks

Categorical Siamese Networks (Tuna et al., 2018) extend the Siamese networks model, which detects semantic similarity among pairs of images, to output directly the best matching category. Our approach has also been inspired by the Prototypical Networks model with the main difference being that instead of the arbitrary distance function used in PN, we parametrize the distance function by an ANN. Conceptually, unlike the PN, based on the prototype theory of categorization, our model can be seen as instantiating the exemplar view (Medin et al., 2018). In this view there is no single category prototype, but instead, categories are formed by representations of their instances called exemplars. Our hypothesis was that including the categorical decision process in the training of our model will increase the classification accuracy in comparison to the SNN. We also hypothesized that our model would perform better than the PN, due to our ANN-parameterized distance function. Another important distinguishing feature of our model, apart from the architecture, is the training procedure described below.

3.1.1 Method description

The architecture of the Categorical Siamese Networks (CSN) model, displayed in Figure 20, contains two main parts. The first part is the embedding sub-network f_θ that projects the input images into a high-dimensional embedding space using multiple convolutional layers similarly to the SNN. The concrete architecture of f_θ depends on the task being solved and we further describe it in results section.

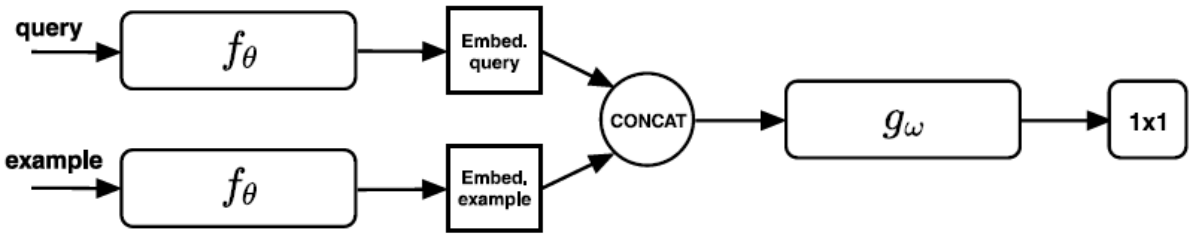

Figure 20. The architecture of Categorical Siamese Network.

The second part is a convolutional neural network g_ω that takes as an input concatenated feature maps for the query image $\hat{x}$ and a category example image x_i. The network output g_ω, given by the activation of a single output unit, represents the semantic similarity as the distance between the query and the support example.

The architecture is used in an episode-based categorization procedure displayed in Figure 21 where outputs of g_ω, each indicating the similarity of the given query $\hat{x}$ and examples $x_i \in S_k$ are averaged over each category k producing the output Y_k. The output values Y_k for each category are then processed using the SoftMax function as described below and the category with a maximum value is selected as the final output of the classication process (winner).

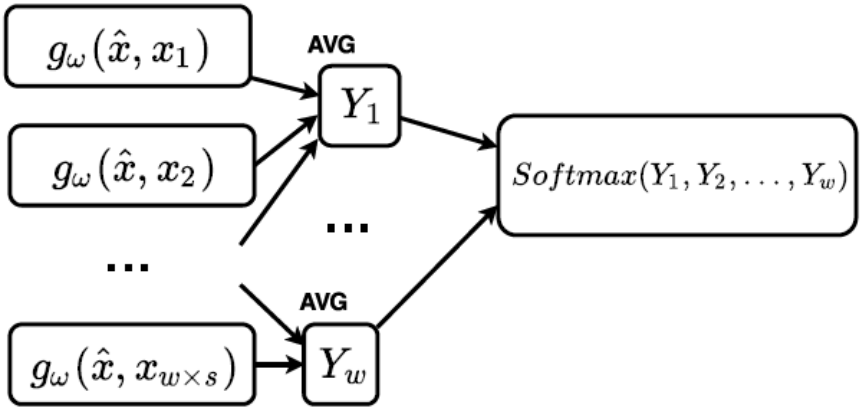

Figure 21. Episode-based categorization procedure outputting the winning category, where w denotes the number of categories and s the number of examples in each support set.

The training procedure of our CSN model is displayed in the following algorithm. The RandSel(M; n) operation denotes the process of randomly selecting n elements from the set M. D_{train} denotes the training subset and D_{test} the testing subset of the initial dataset.

The output for category k is denoted as Y_k, and computed as an average from all the outputs from the network for the given query sample $\hat{x}$ and all support examples x_i from the class k, as displayed in Eq. 32, where $cat(f_\theta(\hat{x}), f_\theta(x_i))$ is the concatenation of embeddings for $\hat{x}$ and x_i, outputted from the embedding network f_θ , and S_k are randomly selected training examples of a category k.

$$Y_k(\hat{x}, S_k) = \frac{1}{|S_k|} \sum_{x_i \in S_k} g_\omega(\, cat(f_\theta(\hat{x}), f_\theta(x_i))\,) \tag{32}$$

Furthermore, $P(\hat{y} = k|\hat{x})$ expresses the probability that the label $\hat{y}$ of the query sample $\hat{x}$ is the label of category k, where $\sum_{j=1}^{|V|} \exp(-Y_k(\hat{x}, S_j))$ is the sum of the distances for all other categories in D_{train}.

$$P(\hat{y} = k|\hat{x}) = \frac{\exp(-Y_k(\hat{x}, S_k))}{\sum_{j=1}^{|V|} \exp(-Y_k(\hat{x}, S_j))} \quad (33)$$

Weights of the network components f_θ and g_ω are then updated using the stochastic gradient descent minimizing the negative log-likelihood loss J according to Eq. 34.

$$J \leftarrow J + \frac{1}{wq}(-\log p(\hat{y} = k|\hat{x})) \quad (34)$$

The training procedure for CSN, which closely fallows the training procedure for can be summarized in the following way.

- Divide the classes in the initial dataset by random sampling into training subset D_{train} containing N_c classes, and testing subset D_{test} containing N_t classes,
- for n episodes do:
    - $V \leftarrow RandSel(D_{train}, w)$
    - $for\ K \in V\ do$:
        - $S_k \leftarrow RandSel(K, s)$
        - $Q_k \leftarrow RandSel(K \backslash S_k, q)$
    - end for
    - for $k = 1$ to $|V|$ do:
        - for $(\hat{x}, \hat{y}) \in Q_k$ do:
            - compute $Y_k(\hat{x}, S_k)$ using Eq. 32
            - compute $P(\hat{y} = k|\hat{x})$ using Eq. 33
            - update loss using Eq. 34
        - end for
    - end for
    - update weights of f_θ and g_ω using gradient descent to optimize Eq. 34
- end for.

3.1.2 Results

We tested our model on two benchmark datasets, the Omniglot (Lake et al., 2011) and the Mini-ImageNet (Vinyals et al., 2016). For each dataset we created a suitable CSN model architecture described below. In order to explore the hypothesis that our model would perform better than the PN model because it uses a distance function parameterized by the network rather than an arbitrary one, we created a smaller CSN network with parameters comparable to the PN architecture.

In all experiments we used the Xavier method (Glorot et al., 2010) for initializing the weights and Adam optimization algorithm for updating the weights. Apart from the SoftMax activation function used for final evaluation, all layers of all our networks use leaky ReLU (Maas et al., 2013) activation function. Similarly to related models, we used a step-wise learning rate annealing. Our CSN models differed in the architecture used, initial learning rate and annealing schedule, and in the number of training epochs.

The Omniglot dataset contains 1623 classes of handwritten characters from 50 different alphabets from around the world with 20 examples per character. We resized the images to 28×28 pixels and divided the dataset to 1200 training classes and 423 test classes independent of the alphabet. We also augmented the classes by rotating the images by multiples of 90 degrees resulting in 4800 and 1692 classes as in Vinyals et al. (2016).

The architectures of networks f_θ and g_ω are displayed in Figure 22. To get the most accurate comparison with similar models we used an embedding network architecture with the same number of parameters as Vinyals et al. [68], and most importantly, the number and size of the convolutional layers. Mimicking the architecture of Vinyals et al. (2016), all 4 convolutional layers of f_θ contained 64 neurons with a 3×3 filter size followed by batch normalization. In order to prevent information loss at this level of processing, we did not use the 2×2 max-pooling layers and just took advantage of the convolutional subsampling using stride 2 in the second and fourth layers of the network (layers 1 and 3 had stride 1).

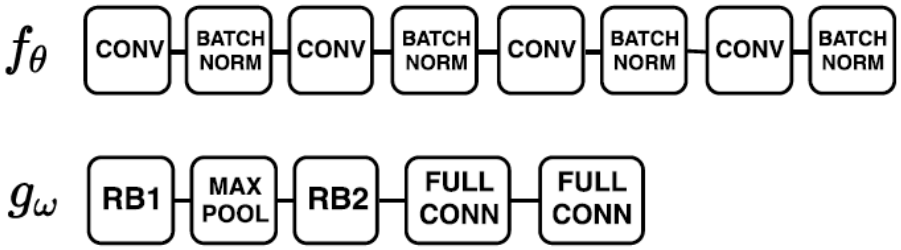

Figure 22. Architecture of f_θ and g_ω networks for the Omniglot experiments. RB in g_ω represents a residual block depicted in Figure 23.

In Omniglot experiments, g_ω consisted of two residual blocks (256 neurons) with architecture displayed in Figure 23 with 2×2 max-pooling between them and two fully connected layers in the outer-most part of the network (with 512 hidden and 1 output neurons). The same residual block architecture was also used in Mini-ImageNet experiments and contained three 3×3 convolutional layers with batch normalization and one 1×1 residual convolutional layer.

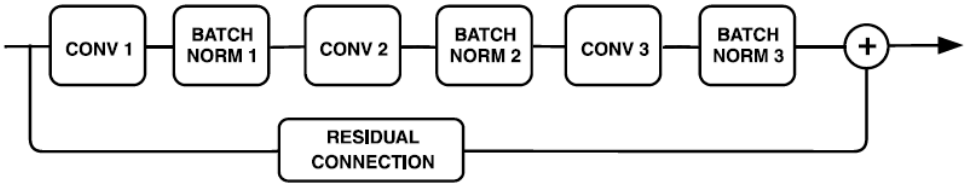

Figure 23. Architecture of convolutional residual block.

We trained our model for 30,000 episodes using the training procedure outlined in previous section. Each episode contained 5 query points per class. During training we tested the models on 1000 testing episodes after each 2000 training episodes. The initial learning rate was 0.0005 and we decreased it to a half every 2000 episodes until episode 10,000. The results in terms of classification accuracy for 5-way and 20-way classification each with 1-shot and 5-shot tasks are displayed in Table 1.

	5-way		20-way	
Model	**1-shot**	**5-shot**	**1-shot**	**5-shot**
Siamese Networks (Koch et al., 2015)	97.3%	98.4%	88.2%	97.0%
Matching Networks (Vinyals et al., 2016)	98.1%	98.9%	93.8%	98.5%
Prototypical Networks (Snell et al., 2017)	97.4%	99.3%	96.0%	98.9%
Relation Networks (Sung et al., 2018)	**99.6% ± 0.2%**	**99.8% ± 0.1%**	97.6% ± 0.2%	99.1% ± 0.1%
SNAIL (Mishra et al., 2017)	99.07 ± 0.16%	99.78% ± 0.09%	97.64% ± 0.30%	**99.36% ± 0.18%**
CSN (ours)	98.16 ± 0.08%	99.52% ± 0.04%	**97.7 ± 0.13%**	99.23% ± 0.06%

Table 1. Classification accuracies for Omniglot in comparison with selected models with 95% confidence intervals where available.

In order to compare our model with Prototypical Networks we used the same f_θ as in all other Omniglot experiments. To consistently compare our results with the PN, the g_ω network contained only two fully-connected layers as in outermost part of g_ω in our Omniglot architecture described above (512 and 1 neurons). Additionally, we observed the same training practice as Snell et al. (2017) using 60 instead of 20 classes in each training episode. The results for 20-way classification with 1-shot and 5-shot tasks are displayed in Table 2.

	20-way	
Model	**1-shot**	**5-shot**
Siamese Networks (Koch et al., 2015)	92.0%	NA
Prototypical Networks (Snell et al., 2017)	96.0%	98.9%
Categorical Siamese Networks (ours)	**96.81%**	**98.97%**

Table 2. Classification accuracies for Omniglot with small CSN in comparison with Prototypical Networks and Siamese Networks.

Mini-ImageNet dataset contains a subset of ImageNet dataset with 100 classes and 600 examples per class. This dataset is much more challenging than the Omniglot dataset due to the nature of images which in the case of Mini-ImageNet are color photographs of objects in the real world. The images in this dataset were resized to 84×84 pixels. Similarly, to related works, we did not use any image preprocessing. We used the standard data split by Ravi and Larochelle (2017) to 64 classes used for training, 16 for validation, and 20 for testing.

The architectures of networks f_θ and g_ω are displayed in Figure 24. The embedding network f_θ, consisted of 4 residual blocks (displayed in Figure 23) with a gradually growing size (64-96-128-256 neurons) each followed by 2×2 max-pooling with stride 1. g_ω had the same architecture as for Omniglot (two 256-neuron RBs with pooling between and 2 fully connected layers).

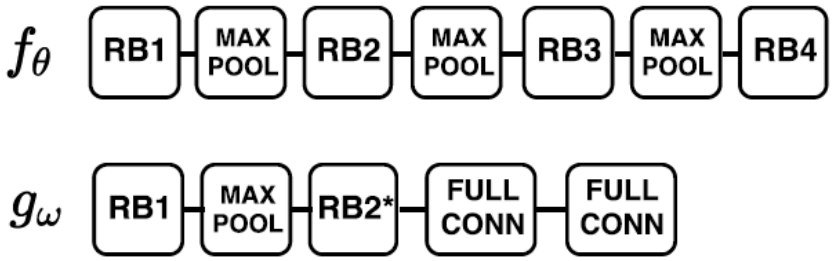

Figure 24. Architecture of f_θ and g_ω networks for the Mini-ImageNet experiments. RB2 does not have the 1×1 convolution in the residual connection because the number of neurons does not change from RB1 to RB2.*

We trained the network for 30,000 episodes for 1-shot experiments and for 20,000 episodes for 5-shot experiments. Each episode contained 15 query points per class. During the training we tested the models on 1000 testing episodes after each 1000 training episodes. The initial learning rate was 0.0001 and annealed to a half every 8000 episodes until episode 16,000. The results for 5-way classification with 1-shot and 5-shot tasks are displayed in Table 3.

	5-way	
Model	**1-shot**	**5-shot**
Matching Networks (Vinyals et al., 2016)	43.6%	55.3%
Prototypical Networks (Snell et al., 2017)	46.61% ± 0.78%	65.77% ± 0.70%
Relation Networks (Sung et al., 2018)	50.44% ± 0.82%	65.32% ± 0.70%
SNAIL (Mishra et al., 2017)	55.71% ± 0.99%	68.88% ± 0.92%
CSN (ours)	52.17% ± 0.43%	66.01% ± 0.52%

Table 3. Classification accuracies for Mini-ImageNet in comparison with selected models with 95% confidence intervals where available.

The results of our experiments show that the performance of our model is comparable with the state-of-the-art models. Compared to model by Mishra et al. (2017), our model has slightly worse performance, but on the other hand, it consists of a smaller number of layers and has fewer parameters.

The classification accuracy of our model in 5-way Omniglot experiments is also slightly lower compared to the Relational Network model by Sung et al. (2017). However, there was a difference between us and Sung et al., who used many more query points, namely 19 for 1-shot and 15 for 5-shot learning. In order to reach the most accurate comparison with the related work and also to compensate for our hardware limitations, we used just 5 query points for all Omniglot experiments.

By outperforming the Prototypical networks in the Omniglot experiments, we have shown that including the categorical decision process in the training has indeed been beneficial for the model's classification accuracy.

Similarly to other deep learning architectures for few-shot learning, our model is very computationally demanding and cannot be operated without a powerful GPU with large memory capacity. Due to the computational complexity which makes experimentation quite a lengthy process, we have not yet tested all possible options of our model. For instance, we could have given the model much more than 30 thousand training episodes whereas Sung et al. (2017) had reported more than 200,000 training episodes. A more detailed search of the model hyperparameters might also lead to better model performance.

3.2 Few-shot semantic segmentation using Seg-REPTILE algorithm

In recent years Deep learning was successful in producing models that can achieve high performance on several challenging tasks including image classification, image segmentation, or instance segmentation. Deep learning models based on Convolutional Neural Networks (CNNs) such as Fully Convolutional Networks (FCN) (Long et al., 2015), Deeplab (Chen et al., 2017) or Dual Attention Networks (Fu et al., 2019) are currently the state-of-the art models for semantic image segmentation. Like other Deep learning models these models require a large number of annotated examples. Due to the high annotation cost of the data required for training such models, publicly available datasets for semantic image segmentation such as Pascal VOC (Everingham et al., 2015), MS COCO (Lin et al., 2014) or Cityscapes (Cordts et al., 2016) contain only a fraction of training examples in the large image classification datasets such as ImageNet (Russakovsky et al., 2015). Because of the difficulty of obtaining annotations in the semantic segmentation class few-shot learning methods, that aim to reduce the necessary number of training examples, could potentially be even more beneficial in the context of semantic segmentation than they are in the context of image classification.

Semantic segmentation task is inherently more difficult than category classification, because instead of one label, it requires a 2D map of labels, separating a target object from the background. Few-shot classification models such as Siamese neural networks (Koch et al., 2015), Prototypical networks (Snell et al., 2017) or MAML (Finn el al., 2017) achieve classification accuracy approaching fully supervised models, using only a fraction of labelled

data. Due to an inherently higher difficulty of semantic segmentation and limited availability of training data, few-shot approaches to semantic segmentation seem viable, but they have not yet been sufficiently explored, compared to their use for categorization. Among the first attempts to few-shot semantic segmentation is Shaban et al. (2017) who combined a segmentation model based on a fully-convolutional network (Long et al., 2015) with a model trained to generate a set of parameters conditioning the segmentation model to segment objects of particular category using only a few examples. Dong et al. (2018) introduced a metric learning approach (similar to Prototypical networks) that relies on creating per class prototypes from a limited amount of training data that can be used for final segmentation. Guided networks by Rakelly et al. (2018) use a small number of training images with limited annotation (one point per object) to perform semantic segmentation. This model also uses two networks, one for generating latent representation of a task from a small number of sparsely annotated images and one that uses this representation to generate a segmentation map.

We explore a few-shot segmentation model we call Seg-REPTILE (Tuna et al., 2019), based on the REPTILE meta-learning algorithm. The main advantage of our approach is that our approach is architecture agnostic. Therefore, our approach can be easily applied to future innovative semantic image segmentation architectures. We evaluate the accuracy of our model using standard semantic segmentation architecture.

3.2.1 Method description

In our experiments, to be consistent with the setup used by Shaban et al. (2017), we used the network architecture FCN32 (Long et al. 2015). High level schema of the Fully Convolutional Network architecture is depicted in Figure 25. The only difference is that we chose Adam optimizer (Kingma et al., 2014). We also adapted the Adam optimizer to the REPTILE update by treating the term $(\tilde{\theta} - \theta)$ as a gradient. The convolutional feature extractor was pretrained using the ImageNet dataset.

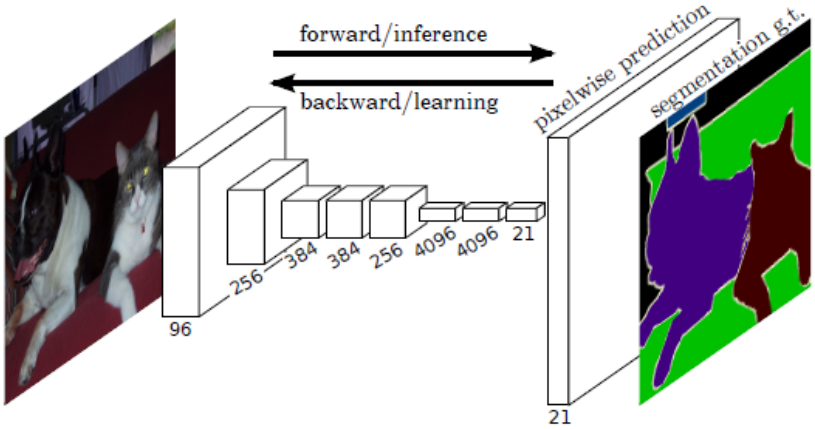

Figure 25. High-level schema of Fully Convolutional Network architecture (Long et al., 2015).

We used PASCAL VOC 2012 image segmentation dataset (Everingham et al., 2015) and applied the same category splits as Shaban et al. (2017). The performance was evaluated on a one way 5-shot tasks, which means that during training every iteration a random category is sampled from a set of 15 training categories from the training subset. Then every single SGD iteration, 5 supports images that contain at least one pixel from that category are sampled from the training data. All pixels belonging to categories that are different from the training category are relabeled as background. During testing the similar procedure is performed, although the categories are taken from the validation subset and the network is retrained using only the 5 support examples selected at the beginning of the test episode. After each test episode the network parameters and Adam optimizer statistics are reset to the state before the test. We used the same performance metric as Shaban et al. (2017), namely the per class mean intersection over union computed across all 5 test classes (excluding background). The learning rates for Adam optimizer and for REPTILE were set to 0.0001. For every REPTILE update during training we performed 24 Adam updates, and during testing 64 Adam updates. The network was trained for 1000 REPTILE updates.

3.2.2 Results

In Table 5 and Table 6 we compare our model with Shaban et al. (2017) using the same 4 test splits (named PASC5-0, PASC5-1, PASC5-2 and PASC5-3). In Table 4 we show which categories are included in different test splits.

Test split	Categories
PASC5-0	['aeroplane', 'bicycle', 'bird', 'boat', 'bottle']
PASC5-1	['bus', 'car', 'cat', 'chair', 'cow',]
PASC5-2	['diningtable','dog', 'horse', 'motorbike', 'person']
PASC5-3	['potted plant','sheep', 'sofa', 'train', 'tv-monitor']

Table 4. Categories included in various test splits

We have not compared our model to Dong et al. (2018) and Rakelly et al. (2018) due to a different metric used in their work. Accuracy of our model for most test splits, except PASC5-1, is comparable to the model by Shaban et al. There is a large difference in the number of parameters between the two models. Our model does not contain the conditional model that contains similar number of parameters as the segmentation model which might explain the difference in segmentation accuracy. Example segmentations from our model are shown in Figure 26.

Method (1-shot)	PASC5-0	PASC5-1	PASC5-2	PASC5-3	Mean
(Shaban et al., 2017)	33,6	55,3	40,9	33,5	40,8
Ours	**39,8**	**47,6**	**38,08**	**34,1**	**39,9**

Table 5. Performance comparison of the models using 1 example per class. We used the mean intersection over union (MIOU) metric from Shaban et al (2017).

Method (5-shot)	PASC5-0	PASC5-1	PASC5-2	PASC5-3	Mean
(Shaban et al., 2017)	35,9	58,1	42,7	39,1	43,9
Ours	**40,9**	**50,5**	**39,9**	**36,1**	**41,9**

Table 6. Performance comparison of the models using 5 examples per class.

Figure 26. Examples of segmentation from our model.

3.3 Detecting Wearable Objects via Transfer Learning

Transfer learning is a well-known technique how to tackle the problem of limited amount of labelled data in training deep neural networks. It has been successfully used in the field of camera surveillance image processing which suffers from poor data quality and quantity. In our work (Kraus et al., 2019) we investigated this technique in our task of wearable object detection and investigated generalization capabilities of well-known neural network architectures such as VGG (Simonyan et al., 2014), ResNet (He et al., 2016), and DenseNet (Huang et al., 2017). Our task was to distinguish if people in the images were or were not wearing a backpack. We created new annotations for the DukeMTMC-attribute dataset (Lin et al., 2019) to overcome the discrepancies among the attributes. Within the transfer learning we explored a setup with frozen feature weights, but also the model fine-tuning, which turned out to perform much better.

There are many open challenges in the domain of camera surveillance. For example, the identification of individual people from photos or videos, or other vision tasks such as the attribute recognition. Distinguishing wearable objects or outfits of the observed people is used in the cases when the identity is unknown or cannot be distinguished. Detecting the objects carried by people is a difficult task even if the quality of images is quite high, but in practice surveillance systems are often very heterogeneous. They use cameras with different resolution and positioning, the lighting conditions are often changing during the recording and the field of view recorded by the cameras is not continuous in general. Thus the image quality may vary significantly.

Our motivation is to overcome these difficulties. To process videos from individual cameras and to detect the characteristic wearable objects and clothes, such as the person is wearing a red jacket and a black backpack. This high-level information may be used for finding correspondences in the multiple camera images. Having a system detecting such attributes might be utilized in forensic search as well.

The transfer learning paradigm has been successfully applied in the domain of human re-identification. Li et al. (2012) proposed a metric-transfer approach and outperformed former methods based on the visual features extraction and matching. Geng et al. (2016) enhanced the performance of the transferred model by employing person identification and person classification (distinguishing one person from others). Chen et al. (2017) proposed joint training of the identification subnet with an additional network performing a ranking task.

Lin et al. (2019) came with an idea to include feature recognition to enhance the performance of the neural model in the identification task. Within their work, they re-annotated two re-identification benchmarks: DukeMTMC-reID and Market-1501 (Zheng et al. 2015). Their results show that learning through optimizing a composite loss of both re-identification and attribute classification leads to better performance in re-identification task. Wang et al. (2018) applied semi-supervised learning paradigm to the attribute annotation problem. Their assumption was that attributes might be used as defining features for learning person identities in the unlabeled data.

Yu et al. (2016) showed that a weakly-supervised deep learning architecture with special detector layers can be used to classify pedestrian attributes without the need of bounding box annotations. This approach, which does not use any form of transfer learning, would indeed require a large dataset, which is usually a problem for real-world camera surveillance applications. Xiang et al. (2019) show how incremental few-shot learning can overcome the problems with the amount of labeled data and predefined feature sets needed for majority of the classification systems. An interesting example of deep learning in attribute recognition task is the deep hierarchical contexts model by Li et al. (2016). It uses a custom neural network architecture that builds up pose-based deep representations of humans, compares them, and provides a scene-level feature description, from which it derives attributes. For example it detects the activity people are doing and estimates the wearable attributes semantically related to the activity in all people from the group. Thus it can detect attributes that might, in some cases, not be detectable by a human observer. Another novel model was proposed by Ji et al. (2017), which combined the standard CNN architecture with recurrent neural networks that captures the sequential relations among various pedestrian attributes.

3.3.1 Method description

In this work we focus on building a system for feature recognition for automatic camera surveillance. Such a system has to be compatible with different camera types, resolutions, view-points, etc. Since there is a limited number of suitable datasets for the task we chose to work with the available DukeMTMC-attribute dataset (Lin et al., 2019), which adds additional annotations to the DukeMTMC-reID dataset (Lin et al., 2019). DukeMTMC-reID is a subset of DukeMTMC (Ristani et al., 2016) re-identification dataset created from 85-minute long high-resolution video clips from 8 different cameras. Images are captured every 120 frames and

cropped to bounding boxes containing people. DukeMTMC-attribute consists of 702 training and 1100 testing entities (individual people) from DukeMTMC-reID annotated with 24 additional features related to visible properties, such as the gender (male/female), the properties of outfit (upper-part-black) or whether they are carrying items such as bags or backpacks. For the purposes of our work, we chose to focus on just the backpack attribute. Our first experiments with original DukeMTMC-attribute dataset showed that the default annotations are not fully suitable for our purpose, because the original attributes are assigned based on the entities, rather than visual assessment of the attribute in the individual images. Such annotations are beneficial for enhancing the performance in the re-identification task as shown by Lin et al. (2016), but the attribute relevance might be restricted when we use it for recognition of carrying a backpack on separate images. There are some mismatches in annotation relevant to our task that can negatively influence the training process, such as occlusion by objects, occlusion by person with different attribute value, visibility of a backpack, and few real annotation errors. Therefore, we decided to re-annotate the dataset.

Based on our observation, we re-annotated the images into three classes: with a backpack, without a backpack, and unresolved. We put the image to an unresolved class either when there were more people in an image or when the backpack was not well observable. From the total of 34,183 original annotations: 61.8% were unchanged, 37.0% were marked as undecidable, and 1.2% were changed to the opposite category. It means that 13,045 annotations were modified, of which 3.0% were changed into the opposite category and 97% were marked as undecidable. Examples of images with our annotations for the three categories are displayed in Figure 27. Note that the original images in the dataset are already cropped out of the camera footage and thus have different sizes and resolutions.

Figure 27. Examples of images from re-annotated DukeMTMC dataset.

For the purposes of our work, we exclude the unresolved class from our dataset. Our task can be described as a binary classification problem, the positive class means that the person is wearing a backpack and the negative class means the opposite. We display the numbers of training and testing images in the dataset after our reannotation in Table 7.

	Positive	**Negative**	**All**
Train	6540	3593	10133
Test	6911	4491	11402
All	13451	8084	21535

Table 7. Dataset statistics.

In our re-annotated data there is noticeable imbalance between positive and negative examples. We decided not to compensate for it either in terms of data augmentation or in designing special loss function in our models. However, we take this imbalance into account and use the balanced accuracy (BACC) as a performance measure for our experiments. Mathematically, balanced accuracy can be expressed as:

$$BACC = (\frac{TP}{P} + \frac{TN}{N})/2 \tag{35}$$

where TP stands for classified true positives, P for all positive examples, and TN for classified true negatives and N for all negative class samples. Nevertheless, for possibility of comparison

with other models, we also display the standard accuracy (ACC) in the final results from the best models.

In our experiments we worked with three widely-used neural network architectures for image classification: the VGG, the ResNet, and the DenseNet. We used models pretrained using the ImageNet dataset. We evaluated two different depths of each architecture. That brings us diverse sizes of feature vectors as well as a possibility to make decision about generalization capacity based on depth of the model. The complete list of the 6 architectures with the most important characteristics is displayed in Table 8.

Architecture	Layers all	Fully connecetd	Feature size
VGG16	16	4	4096
VGG19	19	4	4096
ResNet34	34	1	512
ResNet50	50	1	2048
DenseNet121	121	1	1024
DenseNet161	161	1	2208

Table 8. Characteristics of neural network architectures used in our experiments.

We chose to experimentally evaluate two different paradigms in transfer learning. In setup A, we used a feature extractor with frozen weights, where the weights of the convolutional part of the original network trained on the ImageNet dataset are left intact, while the topmost fully connected layer or layers of the model (the classifier) was replaced by a new classifier network tailored for the new task. We gathered the features of the images from our dataset using the feature extractor. Subsequently, we trained a new classifier from the scratch to perform the binary classification.

In the case of transfer learning with model fine-tuning, we also started with a network trained on the ImageNet, but unlike the first setup, the whole model was further trained to perform the new task on the target dataset. The architecture of the feature extractor part of the pre-trained network remained the same and only the topmost layer of the classifier was adapted to fit the new task. In the case of model fine-tuning, we used only a small learning rate to prevent losing the feature structure from the rich ImageNet dataset. Nevertheless, we expected that the network would quite soon converge in the new task. We labeled this scheme as setup B.

For the requirements of setup A and to speed up the network convergence in setup B, we normalized our dataset using standard normalization setup used for ImageNet training. Given the fact that the dimensionality of images in the ImageNet dataset is 224×224 pixels and the

dimensionality of images in our dataset varies between approximately 100×240 pixels, we also needed to adjust the size of the images. We experimentally evaluated several techniques, for example positioning the image in one corner of the 224×224 square and filling the remainder with the mean value, so the network would ignore it. Finally, we decided to use the same method as Lin et al. (2019), which was just to stretch the image to the desired size. Albeit the stretched images did not look very natural for a human, there were no significant differences in the network performance.

To compensate for the relatively small size of our dataset, we decided to augment our data by flipping each image horizontally as suggested by Krizhevsky et al. (2012). We consider horizontal flipping as not harmful to the plausibility of our dataset. We used the same operation on both training and testing sets to keep similar data distribution.

In both setups of our experiments, we compared all six architectures from Table 8 pre-trained on the ImageNet dataset. We used the same data pre-processing pipeline in all experiments. In setup A, we evaluated several classification architectures as described below, while in setup B only the last fully-connected layer was replaced with a layer connected to one output neuron with sigmoid activation function. In setup A, we initialized weights using He's normal initialization (He et al., 2015) which limits the range of the random initialization of weights based on the size of the previous layer, and thus speeds up the convergence of the model. This is useful for us since hidden layer sizes in our tested classifiers vary. In setup B, we used standard Xavier weight initialization (Glorot et al., 2010). For all experiments, we used Binary Cross Entropy (BCE) cost function:

$$BCE = -\sum_{n}^{N} [t_k^n \ln y_k^n + (1 - t_k^n) \ln(1 - y_k^n)] \tag{36}$$

where t_k^n is the label and y_k^n is the network output. We trained all our networks using Adam optimizer algorithm (Kingma et al., 2014) with the same parameters across all experiments. These parameters were $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$. We did not use any weight decay.

3.3.2 Results

3.3.2.1 Hyperparameter search

The first step in our experimental setup was to determine the optimal combination of minibatch size and learning rates. For every combination of selected minibatch sizes and learning rates, we repeated the experiment 10 times in setup A and 5 times in setup B and averaged the results across runs for the particular combination of hyperparameters. We show our results on the DenseNet161 architecture in setup A in Table 9 and for ResNet50 architecture in setup B in Table 10.

	Batch size		
Learning rate	**4**	**16**	**64**
0,01	79,12 ± 0,36	79,31 ± 0,31	79,31 ± 0,26
0,001	79,34 ± 0,4	79,54 ± 0,29	79,89 ± 0,31
0,0001	80,04 ± 0,22	80,21 ± 0,22	80,2 ± 0,21
0,00001	80,04 ± 0,15	79,91 ± 0,11	79,71 ± 0,03

Table 9. BACC for hyperparameter search in setup A using DenseNet161.

	Batch size		
Learning rate	**4**	**16**	**64**
0,001	81,68 ± 0,53	84,98 ± 0,62	87,73 ± 0,76
0,0001	90,25 ± 0,61	91,36 ± 0,22	91,24 ± 0,25
0,00001	91,57 ± 0,25	90,17 ± 0,24	88,45 ± 0,28

Table 10. BACC for hyperparameter search in setup B using ResNet50.

We identified different combinations of learning rates and minibatch sizes that performed the best on the test set. Surprisingly, we were able to find a combination that performs reasonably well in both setup A and setup B, which was learning rate 0.0001 and minibatch size 16. We used these hyperparameters in all subsequent experiments. Even though that this combination was not the best in setup B (the best result was achieved with learning rate 0.00001 and minibatch size 4), we chose minibatch size 16 because of larger minibatches lead to faster training (wall clock time), especially when computing on the GPUs.

3.3.2.2 Comparison between setups

In setup A, we experimented with the architecture of the trained classifiers. We used only fully-connected feedforward networks with different number of hidden layers and their sizes. As well as in setup B, the output of all classifiers is one neuron with sigmoidal activation function, while for hidden layers we used RELU. We experimented with different size of the hidden layer derived from the size of the input layer which varies with a particular feature extractor architecture. The names and parameters of tested classifiers are displayed in Table 11 where N denotes the number of neurons in the input layer. We summarize the best results in terms of balanced accuracy for all six feature extractors and all five classifiers in Table 12.

Classifiier name	layers	hidden
Linear1	2	---
Linear2a	3	N/4
Linear2b	3	N/2
Linear2c	3	3N/4
Linear2d	3	N

Table 11. Characteristics of the output classifier used in Setup A. Layers denotes the number of hidden layers in the classifier. 'Hidden' denotes the number of neurons per hidden layer. N denotes the size of the feature vector used as an input to output classifier (see Table 8).

	Feature extractor					
Classifier name	ResNet34	ResNet50	VGG16	VGG19	Dense121	Dense161
Linear1	75,58	77,27	75,42	73,55	77,85	79,53
Linear2a	77,42	78,64	74,54	73,05	79,03	80,09
Linear2b	77,66	78,58	74,35	73,12	79,25	80,07
Linear2c	77,73	78,39	74,32	73,15	79,18	80,25
Linear2d	77,71	78,35	74,18	72,91	79,01	80,21

Table 12. BACC for all feature extractors and classifiers in setup A.

In order to find the optimal model, we compared the above-mentioned architectures using previously determined learning rate and minibatch size. The highest ACC and BACC on the testing data for every network architecture was averaged across 10 runs in setup A and 5 runs in setup B. We show the results in Table 13 and 14. The network architecture with the highest accuracy on test data was DenseNet161. This is consistent with the performance of deep neural networks on other image-based datasets, where increased depth of neural network architecture generally leads to better generalization.

Architecture	Classfier	BACC	ACC
ResNet34	Linear2c	77,73 ± 0,19	79,03 ± 0,14
ResNet50	Linear2a	78,64 ± 0,16	80,01 ± 0,16
VGG16	Linear1	75,42 ± 0,08	76,88 ± 0,09
VGG19	Linear1	73,55 ± 0,12	75,29 ± 0,09
DenseNet121	Linear2b	79,25 ± 0,2	80,56 ± 0,15
DenseNet161	Linear2c	80,25 ± 0,22	81,47 ± 0,15

Table 13. BACC and ACC of our best models on the test set in setup A.

Architecture	BACC	ACC
ResNet34	90,99 ± 0,31	91,88 ± 0,28
ResNet50	90,43 ± 0,16	91,3 ± 0,27
VGG16	88,8 ± 0,52	89,61 ± 0,39
VGG19	87,52 ± 0,99	88,4 ± 1,16
DenseNet121	91,55 ± 0,31	92,34 ± 0,25
DenseNet161	91,88 ± 0,22	92,65 ± 0,14

Table 14. BACC and ACC of our best models on the test set in setup B.

To take a closer look at the extracted features that we use for classification we used the well-known t-SNE algorithm (van der Maaten et al., 2008). We used a freely available t-SNE implementation from the Python-based Scikit-learn library (Pedregosa et al., 2011) with the default parameters. We randomly selected 1000 samples from each class, i.e. images of people with backpacks and without backpacks separately from the testing set. Note that our choice to use a fixed number of random samples from each class does not respect the real distribution of the classes in the dataset which is slightly imbalanced. Figure 28 illustrates the t-SNE for the DenseNet161 feature vectors from setup A and setup B. In the figures it is clearly visible that the features from the feature extractor with frozen weights are very different from the fine-tuned ones and that it is obviously a very hard task to separate the two classes, which explains the noticeably worse models performance in setup A.

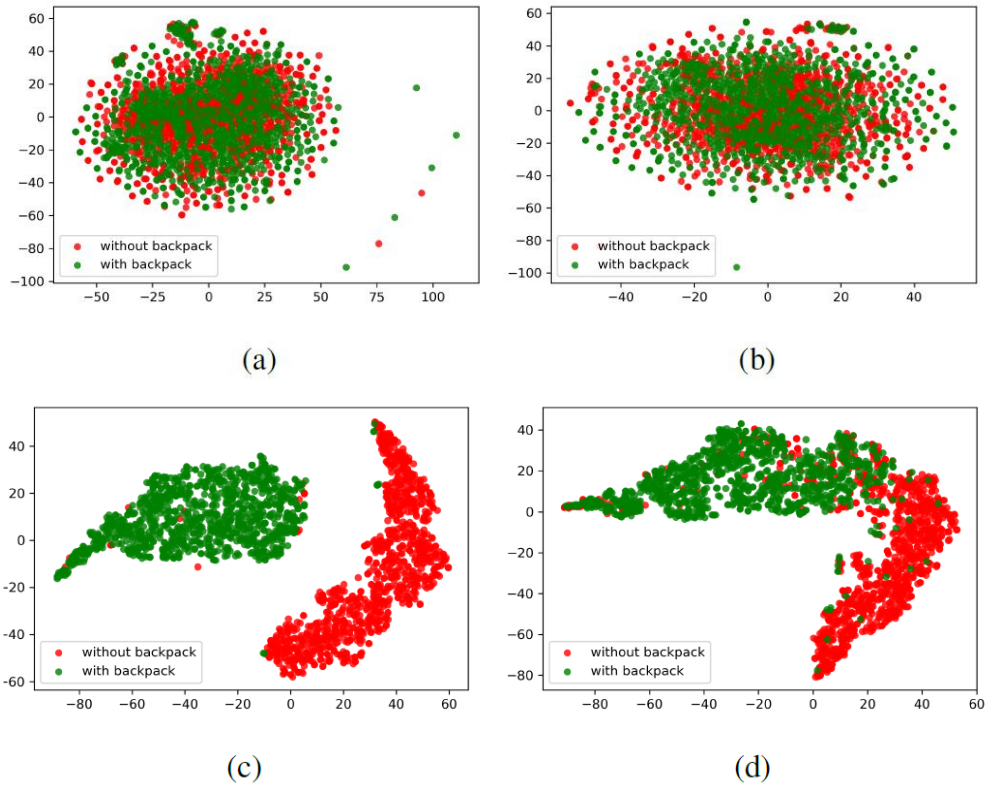

Figure 28. t-SNE visualization for 1000 randomly chosen features from the last layer of DenseNet161. Pictures a) and b) depict setup A with frozen feature extractor weights for the train set and test set, respectively. Pictures c) and d) depict setup B with fine-tuned feature extractor weights for the train set and test set.

In setup A, we are testing how well these deep architectures generalize. In setup B we fine-tuned the feature extractors themselves which turned out to be crucial for successful transfer to our target dataset and task. As we intuitively expected, the setup B that uses fine-tuned convolutional feature extractors achieved much better performance on both testing and training data compared to setup A. It is most likely due to the fact that the distribution of the data in our dataset is substantially different from the distribution of data in the ImageNet dataset that was used for the pre-training of the tested feature extractors. For example, our data have different aspect ratio than the typical image from ImageNet, have lower resolution and lower visual quality. This is evident from the differences in the BACC as well as from the feature visualizations using the t-SNE method. These visualizations show that features from the fixed feature extractor are difficult to linearly separate, which explains the performance gap between the setups.

3.4 Semi-supervised Learning in Camera Surveillance Image Classification

In this work (Tuna et al., 2021) we build upon our previous work in the domain of pedestrian attribute recognition (Kraus et al. 2019). Our work unfolds in two directions, the use of data augmentation to reach better performance in the supervised learning paradigm as a continuation of our previous work, but also a new path in which we experiment with the semi-supervised learning models. Namely, we propose a novel semi-supervised model for binary classification of wearable objects that we call Binary Mean Teacher, based on the Mean Teacher (MT) model (Tarvainen et al., 2017).

We evaluate our models in several different setups. First we evaluate the influence of data augmentations in the supervised setup with full label set, but also with very limited small subset of the training data. Next, we explore and evaluate our Binary Mean Teacher (BMT) model that we adapted for our task and dataset and compare the performance of BMT in the limited data case, i.e. just one thousand labels. We evaluate this semi-supervised model with various quantity and quality of the unlabeled data.

3.4.1 Method description

For the experiments with a simple supervised learning both with and without transfer learning we used our DukeMTMC-backpack dataset as described in Kraus et al. (2019). In this case we kept the samples with correct yes and no labels and left the uncertain class out. Note that, as we kept the original data split from DukeMTMC-attribute and left out the uncertain cases, our testing set remained slightly larger than the training set, which renders the task more difficult than with usual benchmarks.

For experiments with semi-supervised learning we took our DukeMTMC backpack and used either the full labeled training set or kept only a portion (2%, 8%, 10%) of the labels and removed the labels from the remainder of the dataset. The case in which there is only data from DukeMTMC-backpack with a particular portion of labels removed will be referred to as DS0. In order to explore the influence of the magnitude of the unlabeled part of the training set on the network performance we decided to enhance our unlabeled parts of the dataset in two ways. First, we enhanced the unlabeled part of DS0 by 6389 images that we relabeled as uncertain in the DukeMTMC-backpack dataset and named it DS1. Additionally, we decided to use a completely different, yet a similar dataset, namely both training and testing sets of the Market-

1501 dataset (Zheng et al., 2015). The combination of DukeMTMC-backpack with Market-1501 will be referred to as DS2. It is important to note that in all DS0, DS1 and DS2 cases the amount of the labeled data remains the same, therefore the actual ratio between labeled and unlabeled data varies among these datasets. Note that our dataset mixing does not oblige the original assumption for the SSL that labeled and unlabeled parts of the dataset must have the same distribution. However, we believe that since there is a strong similarity among DukeMTMC and Market-1501 datasets, that they are both made out of camera-surveillance images by cropping out people and used for attribute recognition, the distributions will be reasonably similar. We noticed that the class imbalance for the backpack attribute in the Market-1501 dataset is much bigger, we can observe whether such imbalance can disrupt the performance of the BMT model. We provide the dataset statistics in Table 15.

Labels	**DS0: DukeMTMC-backpack**	**DS1: DS0 + uncertain**	**DS2: DS0 + Market-1501**
10133	100%	61,33%	28%
1000	10%	6,05%	2,76%
800	8%	4,84%	2,21%
200	2%	1,21%	0,55%

Table 15. Naming scheme and statistics of the datasets used in our experiments.

Note that, the case of DS0 with 100% labels is not a real case of semi-supervised learning since there are no unlabeled data. However, it can work nicely as another supervised baseline that uses the same augmentation technique and distance-based regularization, yet with no contribution of the consistency regularization.

As a standard procedure, we used contrast normalization per channel. We adapted the size of images in our dataset, which usually have size of approximately 100×240 pixels to ImageNet size, which is 224×224 pixels by stretching. Although the stretched images did not look very natural, there were no significant differences in the network performance.

For our semi-supervised learning experiments we created a semi-supervised model based on the Mean Teacher (MT) model. To change the MT model for our purposes we not only needed to change the shape of the last fully connected layer. Since our task is binary, we only have one output neuron and our primary (supervised) cost function is the Binary Cross Entropy (BCE) so our supervised cost is

$$S(\theta) = -\sum_{j}^{m} [y_j \log \hat{y}_j + (1 - y_j) \log(1 - \hat{y}_j)] \tag{37}$$

where y_j stands for the binary label and $\hat{y}_j$ stands for the network output $f(x_j, \theta, \eta)$ for labeled input x_j given noise η.

In our preliminary experiments we realized that the MSE cost used for the unsupervised part of the model given just one output neuron does not lead to successful learning. Therefore we altered the model to compute the unsupervised consistency cost directly as MSE from the last convolutional layers of the student and the teacher networks as displayed in Figure 29. Our unsupervised cost is then computed as:

$$J(\tau) = \frac{1}{n}\sum_{i}^{n} \|g(x_i, \tau', \eta') - g(x_i, \tau, \eta)\|^2 \tag{38}$$

where x_i are the training samples augmented using noise η for the student network and η' for the teacher network. Parameters $\tau \subset \theta$ and $\tau' \subset \theta'$ are the weights of the student network and the teacher network up to the last convolutional layer. The output of the whole student network can therefore be expressed as $f(x_j, \theta, \eta) = h(g(x_j))$ where g is the convolutional feature extractor and h is the fully connected layer atop of the architecture. Similarly, the output of the whole teacher network can be expressed as $f(x_i, \theta', \eta') = h(g(x_i))$. We combined the two losses $S(\theta)$ and $J(\tau)$ in the following way:

$$Loss(\theta) = S(\theta) + w_t J(\tau) \tag{39}$$

where w_t is the consistency cost weight.

Following the original formulation of MT model, the parameters of the teacher network are set using parameters ensembling procedure as an exponential moving average of the student model. Model parameter ensembling procedure can be expressed as

$$\theta'_t = \alpha\theta'_{t-1} + (1-\alpha)\theta_t \tag{40}$$

where α represents parameter that controls the ensembling momentum.

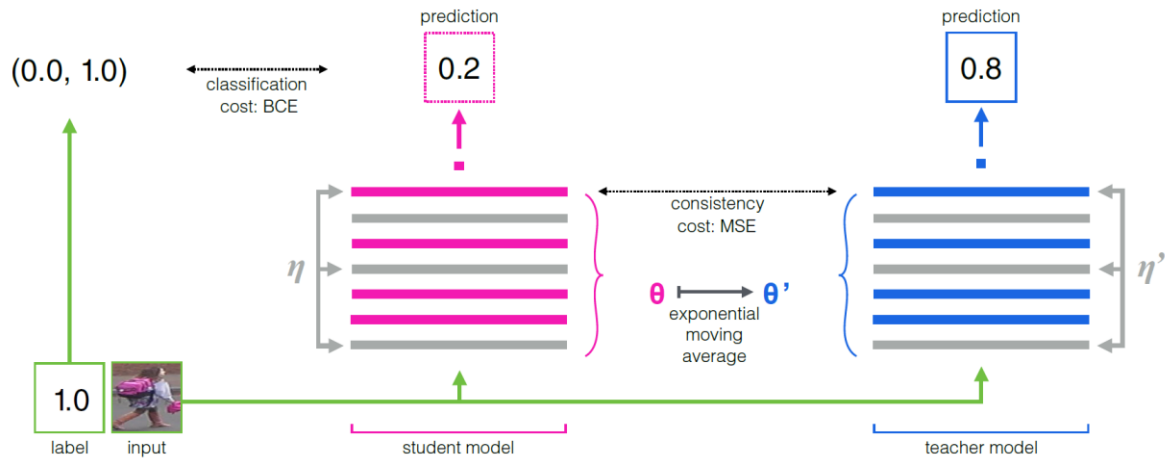

Figure 29. Schema of the Binary Mean Teacher model. Adopted from Tarvainen et al. (2017).

In all our experiments we used DenseNet121 architecture adapted for our binary classification task. The output layer contains one output neuron representing whether the person on the image is with or without a backpack. In all our experiments where there was no pretraining we used the Xavier weight initialization (Glorot et al., 2010). As described in previous section we used the composite loss function that contains supervised loss (Eq. 37) and unsupervised loss (Eq. 38) weighted with parameter w_t. We trained all our networks using well established Adam optimizer algorithm (Kingma et al., 2014) with standard parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$ and no weight decay. Due to high computational demand we used exactly 100 training epochs in all experiments including the hyperparameter search. As the performance measure we use standard accuracy.

3.4.2 Results

3.4.2.1 Data augmentation experiments

In our first set of experiments we explored the influence of data augmentations in the case of fully supervised learning with a limited dataset and with full dataset. Our assumption behind these experiments was that even with a very limited amount of data, using a reasonable augmentation technique, we can achieve a performance comparable to the full dataset training. We explored the full model training as well as model fine-tuning with the model pretrained using ImageNet. For our experiments we chose to use the same set of augmentations as the original MT for the ImageNet and evaluate them separately and in combination. The only difference in our augmentation strategies compared to the original MT is in the crop and resize, which was slightly adapted to keep the backpack and the torso of the person in the image intact. For the sake of clarity we label the augmentations using arbitrary codes as shown in Table 16.

Name	Technique	Parameters
A	Random rotation	max. 10 degree
B	Varying aspect ratio crop and resize	224x224, scale=(0,8;1), ratio=(0,8;1,2)
C	Random horizontal flip	
D	Random color jitter	brightness=0,4; contrast=0,4; saturation=0,4; hue=0,1

Table 16. Parameters and names for augmentation techniques that we use.

In search for optimal hyperparameters we experimented with a varying learning rate and minibatch size for various sizes of training data sets in both setups: full training and model fine-tuning. We tested batch sizes of 4–64 samples and initial learning rates from 10^{-2} to 10^{-5}. While for the learning rate we identified the best value over all tested combinations and learning setups, which was 10^{-4}, ideal minibatch sized varied from 4 to 16. However, the differences in performance were rather small so we decided to settle with minibatch size 8 as best value. Note that such a small minibatch size could not be feasible for the Binary Mean Teacher, where a certain ratio of unlabeled and labeled samples has to be kept in each minibatch. Unlike our small optimal minibatch size, the SSL uses larger minibatches of approximately 128 or more samples. We assume this is necessary due to the nature of the dataset which can in some cases contain only several labeled samples per minibatch which would not be feasible to make a data split with such a small size. The influence of the augmentation strategy on the accuracy of our models is shown in Figure 30.

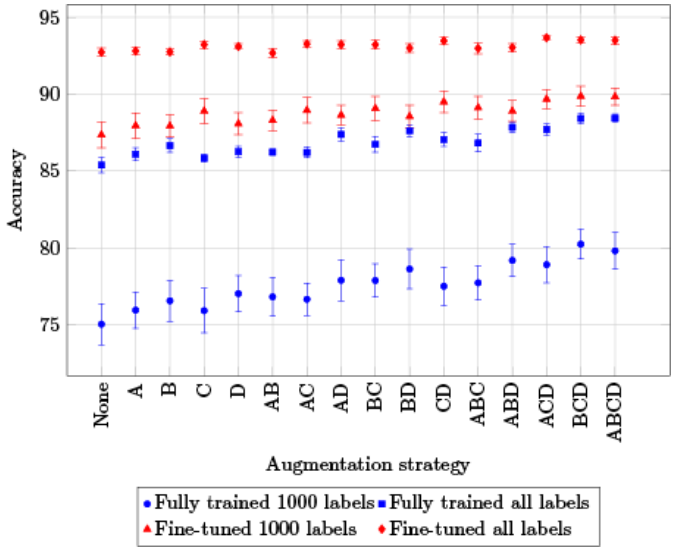

Figure 30. The influence of different augmentation strategies and their combinations on accuracy.

The best performing models were the ones that used model fine-tuning and a combination of 3 and more augmentations. It seems that the cropping and resizing (B) as well as the color jitter (D) have the best overall influence on the performance in our task.

In the context of our domain, the well-labeled images are not in abundance. Therefore, we were interested how well a standard supervised model can generalize over the whole testing set when trained with only a small amount of labeled data. We chose small portions in accordance with the semi-supervised learning experiments which go up to one third of the dataset and compared it with full dataset. Results are displayed in Figure 31.

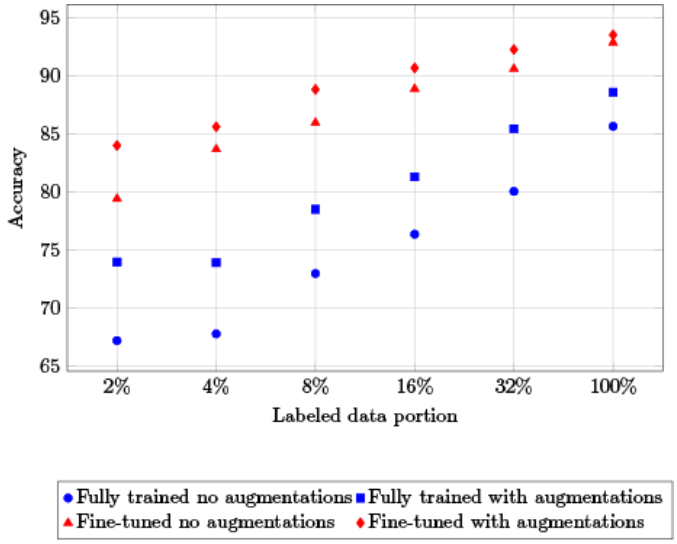

Figure 31. The influence of the fraction of training data (of the original training dataset) on the fully supervised model accuracy.

Note that the testing set we used was the same as in another task and contained more than 11,000 images. The smallest portion of labels, which was 2% of our training set equals to approximately 200 images. This graph illustrates how augmentations can compensate for the lack of data even when no unlabeled data are used. The effect of augmentation is much more visible in the case of full training which however, performs generally worse than the fine-tuned setup. If pre-trained models are used, the weights already respond well to low-level features so the augmentations that generally increase the variability of the training data have smaller influence on the model performance.

3.4.2.2 Binary Mean Teacher experiments

In the second set of experiments we tested the accuracy of our Binary Mean Teacher model in semi-supervised setting. We used model fine-tuning with ImageNet, but also trained the models from the scratch. Adding more complexity we tested the model on three variations of the labeled and unlabeled dataset listed in Table 15, starting from the basic DukeMTMC-backpack and then adding additional unlabeled data. We also took varying numbers of labeled data from our train set (e.g. one thousand or the whole 10,000) while the remainder was added to the unlabeled part of the training set. Namely the excluded part which was hard to label by hand (uncertain category as described in Table 15 as DS1), which basically matches the

distribution of the labeled part of the dataset, but is inconclusive for categorization (contains occlusions, multiple persons,... etc). Then in DS2 we combine two different, yet similar datasets which both contain images of people from surveillance cameras, DukeMTMC and Market-1501. Although there are strong similarities, the distribution of backpack vs. no-backpack cases in Market-1501 is different than in our dataset. Nevertheless, we presume that higher number of unlabelled data will improve the accuracy of our model despite the different dataset distributions. This can be very useful in practical applications where there is an abundance of data, but the labels for the data are hard to achieve. Therefore a reasonable small set of good examples along with well-tailored augmentation strategies can help balancing this problem.

We have found the Binary Mean Teacher model to be very sensitive to hyperparameter selection. Therefore we paid particular attention to hyperparameter tuning process. Because of the large number of hyperparameters that can influence the final accuracy, we focused on exhaustive parameter search only for a small subset of hyperparameters that we found most affected the final accuracy. These include learning rate, unsupervised weight, EMA decay rate and ramp-up length. We performed hyperparameter search on these parameters with range [0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.00001] for learning rate, [20, 30, 40, 50, 60, 70, 80] for unsupervised weight, [0.9, 0.95, 0.99, 0.995, 0.999, 0.9995, 0.9999] for EMA decay rate and [1, 5, 10, 15, 20] for ramp-up length. We performed the hyperparameter search using 1000-label subset of DS2 and then reused these hyperparameters in subsequent BMT experiments.

We performed the hyperparameter tuning process separately for fine-tuning and full training setup. Other hyperparameters were fine-tuned using various heuristics with original parameters used in the original MT research by Tarvainen et al. (2017) as a starting point. We used minibatch of size 64 due to the memory limitations of our hardware. In each minibatch we used 48 unlabeled and 16 labeled examples (8 examples with backpack and 8 without backpack). We found that using minibatch composition with a fixed number of labeled and unlabeled examples led to higher classification accuracy than sampling labeled and unlabeled examples randomly due to high imbalance between labeled and unlabeled examples. We also explored different minibatch compositions but we found previously mentioned composition to work best across multiple experiments. We fixed the ramp-down length at the value 20 in the initial exploratory experiments and then left it fixed because we had found this parameter did not significantly alter the performance. We used all the previously mentioned data augmentations at once, which is the same as in original MT setup for the ImageNet dataset. In Table 17 and 18 we summarize the final hyperparameter selection.

Hyperparameter	Value
Minibatch size	64
Minibatch composition	16 labelled, 48 unlabeled
Ramp up length	10 epochs
Ramp down length	20 epochs
Data augmentation	ABCD

Table 17. Hyperparameters for Binary Mean Teacher for all setups.

Setup	Learning rate	Consistency cost	Ensembling momentum
Full training, all labels	0,001	50	0,995
Full training, 1000 labels	0,0005	70	0,999
Fine-tuning, all labels	0,0001	70	0,9999
Fine-tuning, 1000 labels	0,001	60	0,99

Table 18. Optimal hyperparameters for Binary Mean Teacher for specific setups.

Next, we compare the classification accuracy of our Binary Mean Teacher model with strong supervised baseline. We used the optimal set of hyperparameters that we determined in previous experiments. In Table 19 we compared the classification accuracy of our model to a supervised baseline in a scenario when both supervised baseline and BMT model is trained using randomly initialized weights. In Table 20 we do the same comparison but instead of randomly initialized weights we use weights initialized using the ImageNet dataset. In both experiments we provide a varying number of labelled examples. More specifically we measure the accuracy of both the supervised baseline and BMT model using 2%, 8%, 10% and 100% of all labelled data in our dataset.

Learning type	Dataset	Augmentations	100%	10%	8%	2%
Supervised	DS0	None	85,39	74,96	72,35	63,1
Supervised	DS0	ABCD	88,44	78,47	77,95	69,38
BMT	DS0	ABCD	83,28	89,14	81,77	75,33
BMT	DS1	ABCD	83,73	88,53	82,45	75,56
BMT	DS2	ABCD	85,09	89,57	83,17	75,67

Table 19. Comparison between fully supervised models and Binary Mean Teacher using randomly initialized weights.

Learning type	Dataset	Augmentations	100%	10%	8%	2%
Supervised	DS0	None	92,73	86,96	86,19	78,29
Supervised	DS0	ABCD	93,5	89,91	88,89	82,07
BMT	DS0	ABCD	92,85	90,72	90,04	79,21
BMT	DS1	ABCD	92,98	90,47	89,75	78,96
BMT	DS2	ABCD	93,53	91,75	90,83	77,7

Table 20. Comparison between fully-supervised models and Binary Mean Teacher using ImageNet initialized weights.

The main advantage of the Binary Mean Teacher compared to the baseline appears when we train both models from randomly initialized weights. In this setting we observe 5-10% difference in classification accuracy in favor of the Binary Mean Teacher model. In tests using 1000 labeled training images (8% labelled images), randomly initialized Binary Mean Teacher model achieved 90% classification accuracy. Randomly initialized baseline without any image augmentations achieved 75% accuracy using the same amount of training images which demonstrates the superiority of our approach compared to baseline in this setting.

The difference is much smaller when we initialize the weights of both supervised baselines and Binary Mean Teacher using the weights pretrained on the ImageNet dataset. In this setting Binary Mean Teacher achieved 92% accuracy and the baseline achieved 90% accuracy. This suggests that a supervised model pretrained on a suitable large dataset should be used as a baseline when testing semi-supervised models. In this setting the Binary Mean Teacher model has only a slight accuracy advantage compared to supervised baseline, which demonstrates the power of well-established fine-tuning paradigm. This finding is contrary to most semi-supervised research where the gap between supervised models and semi-supervised models is much higher because of absence of baselines pretrained on large-scale image datasets such as ImageNet. We suspect that much larger unlabeled datasets would be needed to fully leverage the potential of semi-supervised models like our Binary Mean Teacher model in binary classification task studied in this work.

In 2% labeled examples setting the Binary Mean Teacher model achieved slightly lower accuracy than the baseline. This can be attributed to the hyperparameter tuning procedure that used the 1000 label (8%) setting. This suggests that settings with a different number of labeled examples requires different hyperparameter settings to achieve optimal results. We did not perform exhaustive hyperparameter tuning for every setting with a different number of labeled examples due to computational constraints.

3.5 Dataset and methods for training 6D Object Pose Estimator for Imitation Learning from RGB Video

Traditional programming of robots to perform manipulation tasks is expensive and time consuming. An alternative approach is to use machine learning methods to generate an initial solution to the task at hand, based on a demonstration by a skilled human or by another robot. This initial solution consists of trajectories in the space of observed variables. The appropriate actions needed to steer the robot along these trajectories are then computed by using a model of the robot's dynamics or they can be learnt, typically by reinforcement learning.

In this work (Tuna and Sedlar et al., 2021), we focused on using data acquired by observing routine manual work without a necessity to set up expensive motion tracking devices or to install on-teacher sensors or markers. Therefore, we aim to develop methods to extract useful information from recordings by a standard RGB camera which provides a third-person view of the scene. We assume that the human demonstrator uses a hand tool to perform a task, e.g., a glue gun to apply glue along specified trajectories. The problem we address is how to detect the tool and how to track its 6D pose, which refers to position of the object in the 3D space relative to some reference point (for example camera) and pitch, yaw and roll of the object. The ability to train a 6D pose estimator for a new object without the need of its 3D model is useful also in other areas such as virtual reality or object grasping.

The key challenge of the above approach is to accurately and reliably learn the 6D trajectory of the tool manipulated by the demonstrator during the demonstration of a task. Tracked objects typically lack distinctive textures and are partly occluded by the human body. This makes the training of the pose estimator difficult and it is a priori unclear whether the tracking accuracy will be sufficient for the robotic task.

To address these challenges we first have collected, annotated and published a dataset for manipulation tasks with 3 tools: glue gun, grout float, and roller. The dataset consists of RGBD camera recordings along with ground truth 6D pose trajectories of each tool, calibration data, and computed tool bounding boxes, as well as segmentation masks for the training set. Then we adapted state-of-the-art 6D pose estimation method DOPE (Tremblay et al., 2018) and used this methods to evaluate the accuracy of a 6D pose estimation algorithm using restricted amount of training data, as well as generalization to a new environment, tasks, camera viewpoints, demonstrating subjects or left/right hand. We also evaluated the benefits of training data augmentation and proposed augmentation methods suitable for improving the generalization

capability of 6D model when tested on new environments, different tasks, demonstrators hand or camera viewpoints. We devised a novel augmentation method based on the blending of original background and random background (called BgBlend) and showed its effectiveness compared to standard augmentation methods.

Motivated by applications in robotics, 6D object pose estimation has recently attracted significant attention. In case of richly textured objects, methods based on matching of local invariant features such as SIFT (Munich et al. 2006) or SURF (Bay et al. 2006) produce reasonable results.

The more complicated 6D estimation of texture-less objects can be handled by models based on mixing of real and synthetic training data such as SSD-6D (Kehl et al. 2017) or DOPE. Methods such as BB8 (Rad et al., 2017) and PoseCNN (Xiang et al., 2017) typically proceed by first localizing and segmenting the object in the image and proposing a rough estimate of the object 6D pose based on its location. This initial estimate is then optionally refined using an iterative refinement procedure. In Pauwels et al. (2015), parallel processing of multiple camera viewpoints and interaction with a 3D model of the scene and objects was used for real-time object detection and tracking. DPOD (Zakharov et al., 2019) uses a deep convolutional encoder-decoder architecture to estimate a correspondence map between the object in the 2D image and the 3D model of the object, which is then used to compute an 6D pose of an object via a PnP algorithm (Lepetit et al., 2009). Pix2Pose (Park et al., 2019) uses similar correspondence mapping between the 2D image and the 3D model combined with generative adversarial training to improve the models handling of occlusions. CorNet (Pitteri et al., 2019) detects poses of object corners in the input image and uses poses of these corners and the poses of corners in a 3D model of an object to compute the 6DoF via a PnP algorithm.

Another recent method called CosyPose (Labbe et al., 2020) estimates the 6D poses from RGB images of multiple objects captured from unknown viewpoints. CosyPose and in more general, other render and compare methods such as DeepIM (Li et al., 2018), require a known 3D model of the object, which is used to synthesize millions of training examples and is also used for alignment at test time. However, obtaining accurate 3D models quickly is a non-trivial task and hence here we investigate a scenario more practical for an imitation learning set-up, which only requires the user to record few short training videos with the hand-held tool.

The DOPE algorithm (Tremblay et al., 2018) predicts the object 3D bounding box vertices and centroid locations in the 2D coordinate system of the input RGBimage. These 2D predictions are then passed to the EPnP algorithm (Lepetit et al., 2009) to find a transformation

of the 3D bounding box into the camera coordinate system. This approach was shown to outperform other models like the PoseCNN.

Several datasets for 6D pose estimation have been collected in recent years. One of the frequently used static datasets for 6D estimation is Linemod (Hinterstoisser et al., 2012), which consists of 15 texture-less household objects with annotations and test set that include these objects in cluttered scenes. An extended version called Linemod-Occluded (Brachmann et al., 2014) also provides annotations for all objects in one of the test sets, introducing a more challenging, occluded testing scenario. The T-LESS dataset (Hodan et al., 2017) features 30 objects, all from industrial environment, which lack an easily discriminative color and are symmetrical along one or more axes. Another such industry-oriented dataset is ITODD (Drost et al., 2017), the ground truth annotations are however not available for the test images. One of the recent ones, the YCB-M dataset, consists of real-world static scenes recorded on 7 different cameras (Grenzdörffer et al. 2020). The authors trained the DOPE algorithm and compared its accuracy on RGBD data from each of the cameras, concluding that the most accurate 6D estimation was obtained from Asus Xtion and Intel RealSense RGBD sensors, which was also used for our dataset collection.

All of the above-mentioned datasets capture static objects with only the camera moving around different angles. However, for a more realistic real-world 6D pose estimation, it is beneficial to train the estimators on datasets depicting manipulated objects. Creating such datasets is even more technically challenging, the existing datasets are thus small, or employ methods that simplify the annotation task. For example, Marion et al. (2018) created the LabelFusion pipeline which uses 3D reconstructions of real-world scenes to generate a large number of synthetic RGB-D data with pose annotations. Alternatively, the authors of the YCB-Video dataset (Xiang et al., 2017) avoided full manual annotation by keeping the recorded objects at fixed positions and moving the camera only, so that most of the 6D poses could be calculated through global optimization. However, as the objects were fixed in one position lying on the table, the bottom viewpoints are missing from the dataset, whereas we cover most of the object viewpoints in our dataset as we manipulate them in hand. A real human activity RGB dataset was proposed by Kokic et al. (2020) and is focused on task-oriented grasping. A part of the dataset is synthetic, another part consists of real RGB-D videos of manipulated objects. However, the annotations are mainly focused on the hand joints positions and only a small part of the objects are provided with their meshes and 3D poses.

Probably the most related to ours is the work of Krull et al. (2014) who created a dataset consisting of three objects recorded with Kinect. While we also used an RGB-D camera for

object detection, we obtained the reference 6D object pose from a HTC Vive controller attached to the object, whereas Krull et al. (2014) used manual annotation. The biggest difference between this dataset and ours is in the size. The dataset from Krull et al. (2014) provides 3187 images in total, whereas we provide nearly 40,000 frames for the train dataset and over 60,000 frames for the test set. We are unaware of any other 6D pose estimation video dataset besides ours that would enable evaluation of trained models for so many different levels of generalization, i.e., across different human operators, left-handed and right-handed manipulation, task variations, camera viewpoints, occlusions and backgrounds. To demonstrate its utility, we measure the impact of each of these challenges on the accuracy of 6D pose estimation provided by the DOPE algorithm.

3.5.1 Method description

Our Imitrob dataset provides annotated videos of various manipulation tasks with hand-held tools in a real-world environment. The dataset is designed for application of 6D pose estimators in industrial tasks, such as imitation learning or object grasping. It consists of two main parts:

- Test dataset and evaluation metrics for benchmarking 6D pose estimation methods,
- Train dataset and augmentation methods for training 6D pose estimators that do not require a 3D model.

The Test dataset provides real-world benchmarking data for 6D pose estimation. It enables evaluation of various setup combinations, including the camera viewpoint, demonstrating subject, performed task, manipulated tool, the hand used for manipulation, or presence of clutter in the scene. Inspired by common trajectory-dependent industrial tasks, we focus on the scenario where the robot is observing manipulation with a tool and imitates the trajectories presented by a human operator. The operator is holding the tool in one hand and performs various tasks, such as application of hot glue with a glue gun along various trajectories on a frame, polishing a surface with a grout float, or flattening a cloth with a roller. In order to learn from such demonstrations, the robot has to identify the tool's 6D pose. A successful task imitation requires the robot to replicate the trajectory with its own end effector. Different setups provided in our dataset are displayed in Figure 32.

The Imitrob dataset features three tools (glue gun, grout float, and roller), four demonstrators (subjects S1-S4), and manipulations by each hand (left LH and right RH). The 6D poses of the tools were measured by the HTC Vive and the image data were recorded using

two RGB-D cameras from front (C1) and right-hand side (C2) viewpoints. Figure 33 displays the data acquisition setup we used. While we do not utilize the depth component in this work, it is included in the dataset, along with the raw data and the code for custom data extraction.

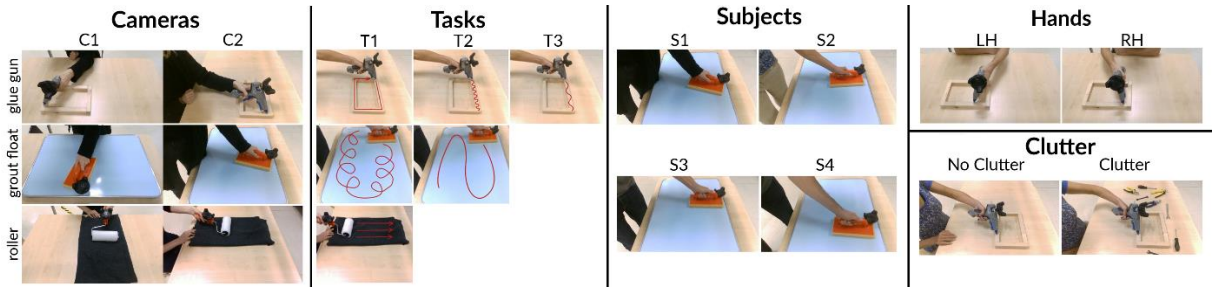

Figure 32. Overview of different setups in the test part of the dataset.

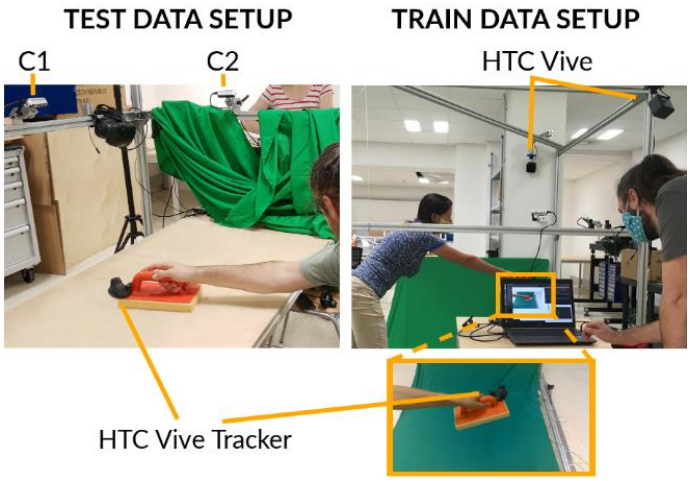

Figure 33. Overview of the data acquisition setup.

The Test dataset contains six tasks with different tool trajectories: three for glue gun (frame, dense wave, sparse wave), two for grout float (round, sweep), and one for roller (press) (see Tasks in Figure 32). In addition, the glue gun frame task was recorded in two environments: with only the gluing frame on the table (NoClutter, default) and with a clutter of other objects around the frame (Clutter) (see Clutter in Figure 32). Each task was performed by all four demonstrators (S1-S4) to simulate the variability of tool manipulation by humans. The dataset could thus be used also for learning task-specific motions from human demonstration.

All the RGB-D images collected in the Imitrob dataset are accompanied by a reference 6D pose of the tool. The 6D poses were acquired from HTC Vive at 30Hz frequency and interpolated to match the time stamps of the camera frames. The Test dataset contains 45 233, 12 048, and 4 379 annotated frames for glue gun, grout float, and roller, respectively.

The Train dataset is designed for training 6D pose estimation methods that do not require a 3D model of the object. Instead of creating a complex 3D model, the dataset captures the tools in various orientations to provide sufficient viewpoint variability for 6D pose training. Each tool was moved randomly in one hand for a short time (20-40s) to simulate the range of possible 6D poses during manipulation. For the purpose of training with data augmentation, the tools were recorded in front of a green background, which enables automatic object segmentation. We include the segmented data (as well as the reference 6D pose from HTC Vive) in the Train dataset. Figure 34 displays examples of the data from the Train part of the dataset.

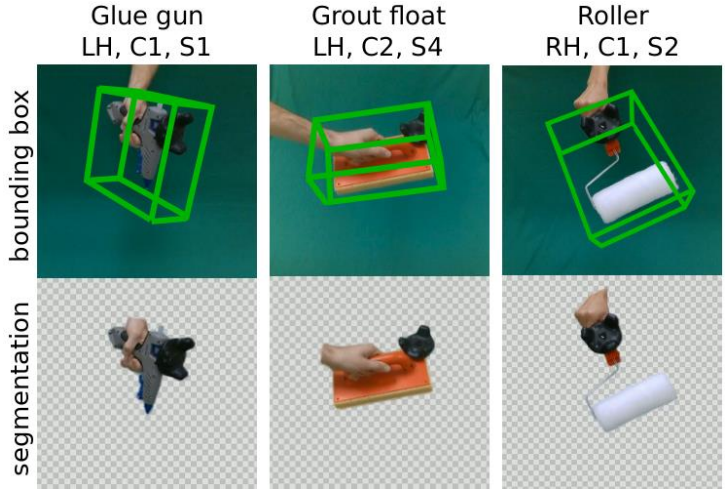

Figure 34. Examples of the data from the Train set.

We provide a collection of augmentation methods convenient for the Train dataset. Augmentation enhances training data and thus allows training of better 6D pose estimation models. The following sections describe the proposed augmentation techniques (we will later evaluate their performance on the Test dataset and propose the best combination). For comparison, NoAug denotes no augmentation of the training data. Overview of different augmentation techniques is provided in Figure 35.

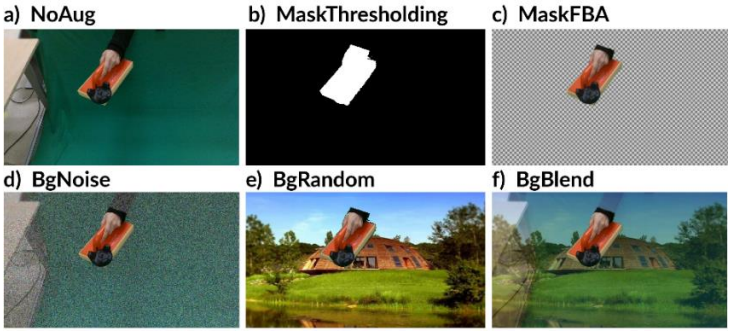

Figure 35. Examples of different augmentation techniques.

First, we segment each image into foreground (in our case the tool and hand) and background. We leverage the green background in the Train dataset for simple thresholding and crop the segmentation mask by the convex hull of the bounding box vertices projected into the 2D image. The result is a binary mask of the tool and hand within the projected bounding box (Mask-Thresholding, see Figure 35b). We enhance the segmentation by F, B, Alpha Matting (FBA) (Forte et al., 2020), which in addition estimates the foreground opacity and color along the boundaries. The output is an RGBA image with the foreground opaque, background transparent, and boundaries convenient for background augmentation (MaskFBA, default, see Figure 35c).

Background augmentation increases robustness of trained models to new environments. We examine several background randomization techniques. BgRandom replaces the background with a random image sampled from the Mini-ImageNet dataset (60 000 images). The remaining three methods keep the original background for 25% of the training set examples; for the remaining 75%. BgAlternate replaces the background with a random image. BgBlend (default) is a novel augmentation method that blends the background with a random image (blending ratio 0.5). BgNoise blends the background with random color noise (blending ratio 0.5). See Figure 35d-f for comparison of the background augmentation methods.

In all augmentation setups (with the exception of NoAug), we apply geometric randomization, namely a random crop and horizontal flip of the image. The crop is constrained to keep all vertices of the 3D bounding box inside the image.

We chose DOPE as our 6D pose estimation model because unlike methods such as CosyPose and DeepIm, DOPE does not need a 3D model of an object to estimate the pose. Instead, only visual data and ground truth 6D pose data are needed. We created a dataset that reflects our scenario in which we demonstrate a task with new object in a real-world setting without training in a simulator. Although training in a simulator has been demonstrated to be effective for training accurate 6D pose estimator, data acquired in a simulator do not contain realistic interaction between the demonstrator, object of interest and the surrounding environment, which might introduce various unwanted biases in the data. We use DOPE to experimentally determine which aspects of real-world environment, such as the change of the human demonstrator, the camera view or the task variability, impact the generalization capabilities of 6D estimation methods the most. We also investigate the impact of various augmentation strategies on the generalization capabilities of the DOPE model across different aspects of the real-world environment.

We estimate the object pose separately in each frame and concatenate the individual frame predictions into a complete trajectory. Since we focus our evaluation on the 6D object pose estimation, we do not apply any postprocessing to the individual frame predictions with any temporal or dynamic model.

Our implementation of the DOPE method was based on the referenced PyTorch implementation by Tremblay et al. (2018). We trained our models for 10 epochs using the Adam optimizer, batch size 16 and learning rate 0.0001. We downsized the input frame dimensions by half to enable training with the larger batch size, which decreased the training time without negative impact on the accuracy. The ground truth belief maps we used for the DOPE training contain a 2D Gaussian with 2 pixel standard deviation and 2 pixel radius at each point (bounding box vertices and centroid).

We evaluate the performance of a 6D pose estimator on the benchmarking Test dataset using three measures. The first one computes the displacement of object bounding box vertices and the centroid while the other two measure the rotation and translation errors, respectively.

ADD pass rate (Tremblay et al., 2018) is defined as the average 3D Euclidean distance of the corresponding bounding box vertices and centroid between the predicted and ground truth 6D poses. The ADD_t (ADD pass rate) measures the percentage of model predictions m^p with ADD lower than the selected threshold t:

$$ADD_t = \frac{\#m^p[where\ ADD(m^p) < t]}{\#m^p} * 100 \tag{41}$$

A higher ADD pass rate for a given threshold indicates better prediction accuracy of the object bounding box. In our experiments, we report ADD pass rates for thresholds t = 2 cm (ADD_2), 5 cm (ADD_5) and 10 cm (ADD_{10}). For comparison of models trained with and without augmentation, we use the ratio of their ADD pass rates:

$$ADD_t^{ratio} = \frac{ADD_t^{aug}}{ADD_t^{noaug}} \tag{42}$$

where ADD_t^{aug} and ADD_t^{noaug} correspond to ADD pass rates for training with and without augmentation.

Rotation error E_{rot} measures the angle between the predicted and ground truth orientation in the following way:

$$E_{rot} = arcos\left(\frac{trace\left(R_p^{-1}R_{gt}\right) - 1}{2}\right) \tag{43}$$

where R_p corresponds to the predicted part of rotation-translation matrix and R_{gt} corresponds to ground truth rotation-translation matrix.

The translation error E_{tra} measures the Euclidean distance between the ground truth (t_{gt}) and predicted (t_p) translation vectors:

$$E_{tra} = \left\| t_{gt} - t_p \right\|_2 \quad (44)$$

3.5.2 Results

We present results of the DOPE 6D pose estimator on our Imitrob dataset for various experimental setups. We train all models on the Train dataset and evaluate their performance on the Test dataset using the ADD pass rate measure with 2 cm (ADD_2), 5 cm (ADD_5), and 10 cm (ADD_{10}) thresholds. First, we show ablation studies to motivate our choice of selected parameters, namely the input image resolution and training batch size as well as the data augmentation method. Then we focus on factors affecting the quality of the results, including the difference between the demonstrating subject, hand, or camera viewpoint in the train and test data, as well as the effect of clutter and manipulation task in the test data.

3.5.2.1 Hyperparameter search and the impact of data augmentation

First we explore the impact of downsizing the input images and increasing the batch size on the quality of DOPE 6D pose estimation. Table 21 presents a comparison of ADD pass rates for the original (848×480 pixels, batch size 8) and downsized (424×240 pixels, batch size 16) frames. While the similarity in performance for the 2 cm threshold could be attributed to a trade-off between a larger batch size and loss of details, the increased batch size clearly improved the accuracy for the 5 cm and 10 cm thresholds. We therefore use the 424×240 pixel resolution and batch size 16 for all other experiments.

	848 x 480 pixels			**424 x 240 pixels**		
ADD threshold	2 cm	5 cm	10 cm	2 cm	5 cm	10 cm
gluegun	7,4	49,8	75,6	**9,9**	**57,8**	**79,8**
grout float	**12,5**	**75,6**	93,0	9,7	73,9	**97,7**
roller	**4,5**	40,3	67,4	4,3	**48,6**	**84,9**
average	**8,1**	55,2	78,7	8,0	**60,1**	**87,5**

Table 21. Comparison between ADD pass rates for models trained using input images with different resolution.

After we determined the impact of input image resolution and batch size in the next set of experiments we compared the effect of different augmentation techniques on the 6D accuracy.

Table 22 shows results for the background augmentation method. Real-world images (BgRandom, BgAlternate, BgBlend) clearly outperform color noise (BgNoise) as random background for augmentation. Also, it is beneficial to keep the original background for a portion (in our case 25%) of training images (BgAlternate, BgBlend) rather than to replace the background everywhere (BgRandom).The best results were achieved on average by BgBlend, which blends the original background and a random image together. The use of BgBlend augmentation (ADD_5 = 60.1%) vastly increased the accuracy compared to training without augmentation (NoAug, ADD_5 = 29.2%). In all other experiments, we therefore use the BgBlend background augmentation.

	NoAug			***BgNoise***			***BgRandom***			***BgAlternate***			***BgBlend***		
ADD threshold	2 cm	5 cm	10 cm	2 cm	5 cm	10 cm	2 cm	5 cm	10 cm	2 cm	5 cm	10 cm	2 cm	5 cm	10 cm
gluegun	1,1	17,0	36,3	1,1	18,9	41,7	6,7	36,5	53,4	5,6	45,4	72,2	**9,9**	**57,8**	**79,8**
grout float	5,3	45,4	80,3	3,3	43,6	86,1	7,7	60,4	84,2	9,6	71,8	95,8	**9,7**	**73,9**	**97,7**
roller	3,4	25,3	56,6	3,8	26,3	52,9	2,9	39,8	85,5	**6,3**	**52,2**	**85,6**	4,3	48,6	84,9
average	3,3	29,2	57,8	2,7	29,6	60,2	5,8	45,6	74,4	7,2	56,5	84,5	**8,0**	**60,1**	**87,5**

Table 22. Comparison between ADD pass rates for different augmentation methods.

3.5.2.2 Examination of environmental factors

To be transferable, the 6D pose estimation algorithm should be invariant to the subject that manipulates the tool. We examine the generalization of the DOPE estimator across 4 different subjects (S1-S4) using the Train dataset for training and the Test dataset for testing. We used three different test setups. In the "All subjects" setup we train one model on all 4 subjects (i.e., S1-S4) and test it on each subject separately (i.e., S1, S2, S3, and S4). In the "Different subjects" setup we train one model for each combination of 3 subjects (e.g., S1-S3) and test it on the remaining subject (i.e., S4, in this example). Finally, in the "Same subject" setup we train one model for each subject (e.g., S1) and test it on the same subject (i.e., S1, in this example). In Table 23 we average the model accuracy for each setup across all test subjects (i.e.,S1, S2, S3, and S4). Table 23 shows that the "All subjects" setup (ADD_5 = 58.8%) outperformed the "Different subject" setup (ADD_5 = 52.0%), which in turn clearly outperformed the "Same subject" scenario (ADD_5 = 31.1%). Additionally, using the ADD_t^{ratio} metric that is defined as a ratio of model trained with augmentation and without augmentation, we can observe that the augmentation improves performance more for "Different subject" (ADD_5^{ratio} = 2.3) scenario than for "All subjects" (ADD_5^{ratio} = 2.0) and "Same subject" (ADD_5^{ratio} = 1.7) scenarios.

	All subjects			Different subject			Same subject		
ADD threshold	2 cm	5 cm	10 cm	2 cm	5 cm	10 cm	2 cm	5 cm	10 cm
gluegun	**10,1**	**57,2**	**80,1**	6,2	52,6	78,2	3,4	32,8	69,4
grout float	**9,5**	**77,2**	**97,5**	6,0	64,1	95,3	1,8	38,8	81,6
roller	3,9	**41,8**	**78,1**	**4,4**	39,1	77,3	0,7	21,8	53,8
average	**7,9**	**58,8**	**85,2**	5,6	52,0	83,6	2,0	31,1	68,3
NoAug	3,1	28,9	57,7	1,9	22,3	52,3	1,2	18,7	51,6

Table 23. Generalization across demonstrators (subjects).

In the next set of experiments we study robustness of the 6D pose estimator to change of the camera viewpoint between training and testing (inference) stage. Our setup contains one front camera (C1) and one right-hand side camera (C2). We trained models using either C1 or C2 or both cameras in the Train dataset and evaluated their performance separately for C1 and C2 in the Test dataset. Table 24 compares results for three scenarios. In the "Same" scenario we train and test on the same camera. In the "Other" scenario we train on one camera and test on the other camera. In the "Both" scenario we train on both cameras simultaneously. For all three tools, the accuracy of using different camera viewpoints during training and testing was very low; on average, the results were better for the transfer from C1 to C2 (ADD_2 = 0.0%, ADD_5 = 12.1%, ADD_{10} = 11.2%) than for the transfer from C2 to C1 (ADD_2 = 0%, ADD_5 = 0.1%, ADD_{10} = 1.0%). The accuracy was significantly higher when the camera used for testing was included in the training. Whereas the best results for evaluation on C1 were achieved by models trained on both C1 and C2 (Both scenario), for evaluation on C2 the best results were achieved by models trained only on C2 (Same scenario).

		Test C1			Test C2		
ADD threshold	Training camera	2 cm	5 cm	10 cm	2 cm	5 cm	10 cm
	Same	8,7	61,6	**84,6**	5,8	**53,3**	**82,6**
	Other	0,0	0,2	1,8	0,1	2,2	12,0
gluegun	Both	**13,7**	**63,6**	83,4	**6,3**	50,6	75,0
	Same	2,4	50,7	90,1	**6,7**	66,0	96,2
	Other	0,0	0,0	0,1	0,0	1,2	21,5
grout float	Both	**13,2**	**78,1**	**97,8**	**6,7**	**77,3**	**97,3**
	Same	5,9	56,7	80,9	**0,4**	**37,2**	**89,0**
	Other	0,0	0,1	1,2	0,0	0,0	0,0
roller	Both	**8,2**	**66,3**	**85,8**	0,0	19,1	72,8
	Same	5,7	56,3	85,2	4,3	**52,2**	**89,3**
	Other	0,0	0,1	1,0	0,0	1,1	11,2
average	Both	**11,7**	**69,3**	**89,0**	**4,4**	49,0	81,7
	Same	1,7	28,7	66,0	3,2	41,8	74,3
	Other	0,0	0,0	0,6	0,0	0,8	8,0
NoAug	Both	3,0	23,5	54,3	3,5	34,8	61,1

Table 24. Generalization across camera viewpoints.

In imitation learning setting where the subject holds the object in their hand while demonstrating certain task, the object of interest can be held in different hand during the training and testing phases. Therefore we devised an experiment that explores generalization of the 6D pose to holding the tool in the left (LH) or right (RH) hand. We trained models on the Train dataset using either LH or RH or both and evaluated them on the Test set separately for LH and RH. Three different scenarios are compared in Table 25. In the "Same" scenario we train and test on the same hand. In the "Opposite" scenario we train on one hand and test on the other. In the "Both" scenario we train on both hands. While training and testing on the same hand ("Same" scenario, ADD_5 = 48.7%) is clearly better than the opposite hand (Opposite scenario, ADD_5 = 24.0%), using both LH and RH for training further improved the accuracy (Both scenario, ADD5 = 58.8%). The data augmentation was more beneficial for training on both hands (Both scenario, ADD_5^{ratio} = 2.1) than for training only on the same (Same, ADD_5^{ratio} = 1.7) or opposite hand (Opposite scenario, ADD_5^{ratio} = 1.5).

		Test LH			**Test RH**		
ADD threshold	Training hand	2 cm	5 cm	10 cm	2 cm	5 cm	10 cm
	Same	5,6	**60,9**	77,6	1,8	25,6	56,9
	Opposite	0,7	8,6	25,0	1,1	22,8	54,8
gluegun	Both	**8,3**	60,8	**87,0**	**8,2**	**51,4**	**74,9**
	Same	3,2	60,7	95,3	3,4	57,1	93,7
	Opposite	1,2	31,1	74,5	2,3	53,4	81,2
grout float	Both	**7,9**	**70,8**	**97,8**	**11,0**	**83,8**	**98,3**
	Same	5,9	**49,9**	83,0	1,6	**37,9**	73,4
	Opposite	0,8	23,3	45,1	0,3	4,9	22,7
roller	Both	**9,4**	**49,9**	**89,3**	**4,9**	35,7	**75,4**
	Same	4,9	57,2	85,3	2,3	40,2	74,6
	Opposite	0,9	21,0	48,2	1,2	27,0	52,9
average	Both	**8,5**	**60,5**	**91,4**	**8,0**	**57,0**	**82,8**
	Same	1,9	29,4	57,1	1,5	22,5	51,9
	Opposite	0,9	13,7	35,0	0,8	18,4	47,6
NoAug	Both	4,0	31,3	58,4	2,0	25,6	52,6

Table 25. Generalization across left and right hand of the demonstrator.

In Table 26 we show generalization of the 6D pose estimator to clutter in the test data. We compare evaluation for the glue gun task frame with only the gluing frame on the table (NoClutter, default) and with a clutter of other objects around the frame (Clutter). In both the "clutter" and "no clutter" scenarios the final accuracy using the augmentation is comparable. We have found that our augmentation technique (BgBlend) was significantly more beneficial in the "Clutter" scenario compared to "NoClutter" scenario as seen by comparing the ADD ratios for these two scenarios. The ADD_2^{ratio}, ADD_5^{ratio}, and ADD_{10}^{ratio} improvement ratios

were, respectively, 5.2, 2.7, and 1.9 for “NoClutter” scenario, compared with 33.2, 12.5, and 4.2 for “Clutter” scenario.

	NoClutter			**Clutter**		
ADD threshold	2 cm	5 cm	10 cm	2 cm	5 cm	10 cm
gluegun (frame)	**8,6**	**61,8**	**90,1**	**11,0**	**61,5**	**83,7**
NoAug	1,7	22,8	47,7	0,3	4,9	19,8

Table 26. Generalization across different levels of background clutter.

Table 27 shows ADD pass rates as well as rotation and translation errors for individual tasks. While the 6D pose estimator performed comparably on different tasks of the same tool, the best results were achieved mostly for the grout float. Overall, the average rotation and translation errors were E_{rot} = 6.5 degrees and E_{tra} = 3.4 cm, respectively.

		ADD threshold				
tool	**task**	2 cm	5 cm	10 cm	E-rot (deg)	E-tra (cm)
gluegun	frame	8	53,3	77,1	11,8	5
	densewave	**10,6**	61,9	88,6	5	3,6
	sparsewave	8,3	66	91	5	3,4
grout float	round	9,2	74,4	**98,7**	**3,9**	2,7
	sweep	9,3	**82,7**	98,1	4,3	**2,2**
roller	press	4,3	50,5	86,3	8,7	3,7

Table 27. Generalization across tasks.

In Figure 36 we present example qualitative results of the DOPE 6D pose estimator trained on Train dataset using the default data augmentation (i.e., segmentation by MaskFBA, background randomization BgBlend, and random crop and flip) and tested on Test set. The predicted bounding box is visualized in red while the ground truth bounding box, acquired through the camera-to-tracker and tracker-to-tool calibration is visualized in green.

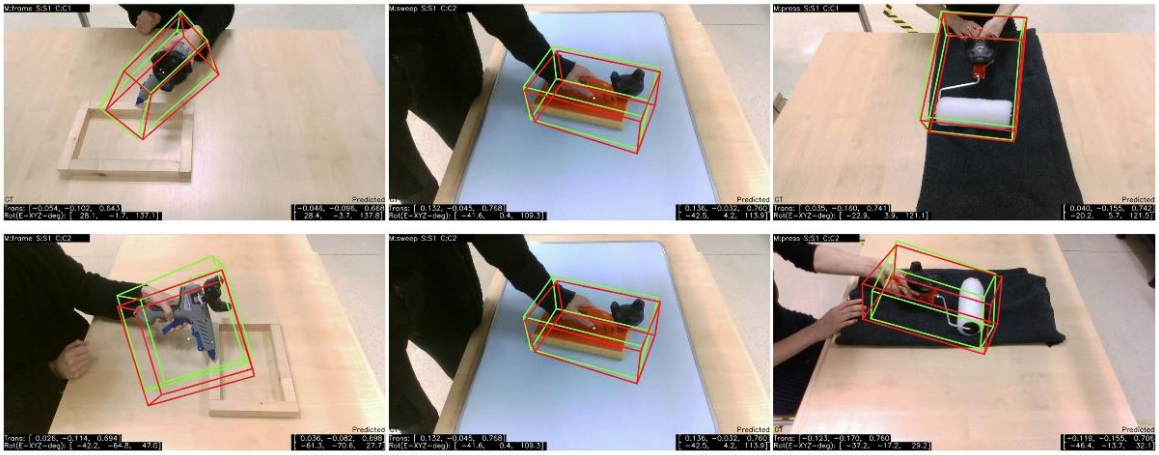

Figure 36. Qualitative results on Test dataset. Ground truth bounding box bounding box is depicted in green, prediction in red. From left to right: glue gun, grout float, roller.

3.5.2.3 Results summary

In this work we have provided a dataset designed for the purposes of training a 6D object pose estimator in the imitation learning setting. We have identified multiple factors that can change between the training phase and testing phase of a 6D pose estimator, including the appearance of the demonstrator, the handedness of the demonstrator, the appearance of the background, the camera angle, the amount of the background clutter and the type of task performed. If any of these factors change between the testing phase and the training phase, this can adversely affect the performance of the 6D object pose estimator. Therefore, our dataset is composed of subsets that reflect these different factors. This enables us to quantify the impact of various factors on the 6D performance which we tested using state-of-the-art 6D object pose estimator called DOPE. We have also tried to increase the invariance of the 6D object pose estimator to changes in various environmental factors using multiple input augmentation methods, including a novel augmentation method (BgBlend) that works by blending the original background with a random image while keeping the original background for a portion (in our case 25%) of the training set.

Based on the quantitative results we can conclude that BgBlend augmentation method is superior to other augmentation methods that we tested including augmentation with random noise (BgNoise), full background replacement (BgRandom) and partial background replacement (BgAlternate). We hypothesize that the reason why BgBlend works better than full or partial background replacement is that when we fully replace the background we remove environmental clues, such as position of the arm or the subtle shadow of the object, which the

6D estimator might use to determine the object pose. Using image background blending instead of background replacement preserves these environmental clues and at the same time changes the appearance of the background enough to increase the invariance of the 6D pose estimator to changes in various environmental factors. We have left future quantitative analysis of which environmental clues contribute to the performance of 6D object pose estimator to future work. Our results demonstrate that BgBlend data augmentation increases the ability of the 6D pose estimator to generalize to a new subject, opposite hand or a cluttered environment. Biggest improvement was achieved in the cluttered environment case in which various distractor objects are inserted in the test scene. This suggest that our data augmentation method is particularly suited for situations where the working environment of the demonstrator changes between the training and testing phase. The benefit of a data augmentation is limited when the angle between training and test camera is large (as in the case of a front and a side camera), because a 2D augmentation does not compensate for a lack of test set tool 3D orientations in the training set. The performance on different manipulation tasks with the same tool is comparable provided that the training data covers the range of tool orientations in the tasks reasonably well.

We have also found that the amount of occlusion of the tool by the manipulating hand in the training data influences the quality of the 6D pose estimation. The experiments using the opposite hand or a different camera angle imply that a front camera (C1) is better suited for recognition of the tools 6D pose than a side camera (C2), probably due to less occlusion of the tool by the hand. These results also imply that the tool is held in the right hand, the side camera should be placed on the left-hand side, and vice versa, to minimize the occlusion.

Conclusion

In this thesis we explored various approaches for improving label efficiency of neural networks. We presented contributions to several approaches including, few-shot image classification, few-shot semantic image segmentation, transfer learning and semi-supervised learning in pedestrian attribute recognition domain and augmentation techniques for 6DOF object pose estimation.

In Chapter 1 we examined basic theory of artificial neural networks and basic types of neural network models and their applications. In Chapter 2 we introduced various approaches for improving label efficiency of neural networks, including transfer learning, data augmentation, few-shot learning and semi-supervised learning. We also introduced concept of label efficiency as a result of training a sequential model on large scale unlabeled datasets. In Chapter 3 we present our algorithmic and experimental contributions.

In subchapter 3.1 we proposed a Categorical Siamese Networks model as an extension of Siamese Networks which directly outputs the categorical decision and utilizes it during training. Although simpler in terms of an architecture and the number of model parameters compared to other state-of-the-art models, our model reaches comparable to these models.

In subchapter 3.2 we proposed the Seg-REPTILE algorithm that adapts the REPTILE few-shot learning algorithm to a problem of semantic image segmentation. We compared the performance of this algorithm to at the time only comparable few-shot semantic segmentation algorithm. In certain subsets of the test data Seg-REPTILE surpassed the performance of comparable algorithm and overall, we achieved comparable performance to the state-of-the-art algorithm.

In subchapter 3.3 we explored the deep transfer learning paradigm in the task of pedestrian attribute recognition. We have created a re-annotated DukeMTMC-attribute dataset that removes ambiguous and incorrect data from the original dataset that could lead to lower classifier performance. We used transfer learning approach to train three image classification architectures pretrained on the ImageNet dataset, namely the VGG, the ResNet and the DenseNet to perform our binary classification task. We compared the performance of two different transfer learning approaches, the fixed feature extractor approach and the model finetuning. We have found the model finetuning to be far superior to the fixed feature extractor approach. DenseNet161 architecture reached the best performance in both tested setups, namely about 80% BACC using the fixed feature extractor approach and about 92% BACC using the

model finetuning approach. In conclusion, we found that transfer learning is a good approach to tackle the lack of quality and quantity of data in the attribute recognition task in the domain of video surveillance.

We continued the exploration of improving the label efficiency in a pedestrian attribute recognition domain in subchapter 3.4, where we explored the application of semi-supervised learning paradigm in this domain. We proposed a semi-supervised model called Binary Mean Teacher for the task of detecting the presence of wearable objects in surveillance data. We also explored the influence of various combinations of input data augmentation techniques on performance of both supervised baselines and our semi-supervised learning model. We show that the lack of labelled training data can be compensated by both data augmentations in case of simple supervised learning as well as with use of the semi-supervised learning paradigm. In tests using 1000 labeled training images, randomly initialized Binary Mean Teacher model achieved much higher classification accuracy compared to randomly initialized baseline without any image augmentations which demonstrates the superiority of our approach compared to baseline in this setting. The difference between Binary Mean Teacher and the baseline was much lower when both models were pretrained using the ImageNet dataset. This demonstrates both the strength of traditional fine-tuning in our task and the need for much higher number of unlabeled examples for the Binary Mean Teacher model to fully show its performance advantage.

Finally in subchapter 3.5 we addressed the problem of 6D object pose estimation of hand tools manipulated by human demonstrators in an industrial environment from RGB image data. To investigate this problem, we have collected a challenging real-world benchmark video dataset, called the Imitrob dataset, of four human demonstrators performing six manipulation tasks with three different tools recorded from two camera viewpoints (front and side). Using state-of-the-art object pose estimation algorithm we examined various factors that can change between the training phase and testing phase of a 6D pose estimator that can affect the generalization of the 6D estimator to unseen environmental factors. We experimentally tested the impact of various factors on 6D estimator performance. We evaluated multiple augmentation techniques to see to what degree these augmentation techniques can increase the invariance of the 6D estimator to changes in various environmental factors. We experimentally showed that using augmentation based on blending the original background with random background sampled from dataset of real-world images is superior to full background replacement and random noise augmentation. We also determined which factors can augmentation techniques be applied to effectively. We conclude that augmentation techniques

can compensate very well to changes in the presence of background clutter or to a person that manipulates the object but cannot compensate for large change in the camera angle.

References

Andrychowicz, Marcin, et al. "Hindsight experience replay." arXiv preprint arXiv:1707.01495 (2017).

Antoniou, Antreas, Amos Storkey, and Harrison Edwards. "Data augmentation generative adversarial networks." arXiv preprint arXiv:1711.04340 (2017).

Bay, Herbert, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features." European conference on computer vision. Springer, Berlin, Heidelberg, (2006).

Berthelot, David, et al. "Mixmatch: A holistic approach to semi-supervised learning." arXiv preprint arXiv:1905.02249 (2019).

Brachmann, Eric, et al. "Learning 6d object pose estimation using 3d object coordinates." European conference on computer vision. Springer, Cham, (2014).

Bromley, Jane, et al. "Signature verification using a "siamese" time delay neural network". *International Journal of Pattern Recognition and Artificial Intelligence*, 07(04), pp.669-688, (1993).

Brown, Tom B., et al. "Language models are few-shot learners." arXiv preprint arXiv:2005.14165 (2020).

Brown, Tom B., et al. "Language models are few-shot learners." arXiv preprint arXiv:2005.14165 (2020).

C. Watkins, "Learning from delayed rewards", *Ph.D thesis*, King's College, (1989).

Caron, Mathilde, et al. "Deep clustering for unsupervised learning of visual features." Proceedings of the European Conference on Computer Vision (ECCV). (2018).

Cordts, Marius, et al. "The cityscapes dataset for semantic urban scene understanding." Proceedings of the IEEE conference on computer vision and pattern recognition. (2016).

Dempster, Arthur P., Nan M. Laird, and Donald B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm." Journal of the Royal Statistical Society: Series B (Methodological) 39.1: 1-22, (1977).

Doersch, Carl, Abhinav Gupta, and Alexei A. Efros. "Unsupervised visual representation learning by context prediction." Proceedings of the IEEE international conference on computer vision. (2015).

Donahue, Jeff, Philipp Krähenbühl, and Trevor Darrell. "Adversarial feature learning." arXiv preprint arXiv:1605.09782 (2016).

Dong, Nanqing, and Eric P. Xing. "Few-Shot Semantic Segmentation with Prototype Learning." BMVC. Vol. 3. No. 4. (2018).

Drost, Bertram, et al. "Introducing mvtec itodd-a dataset for 3d object recognition in industry." Proceedings of the IEEE International Conference on Computer Vision Workshops. (2017).

Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." Journal of machine learning research 12.7 (2011).

Everingham, Mark, et al. "The pascal visual object classes (voc) challenge." International journal of computer vision 88.2: 303-338, (2010).

Finn, Chelsea, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." International Conference on Machine Learning. PMLR, (2017).

Forte, Marco, and François Pitié. "F ,B Alpha Matting." arXiv preprint arXiv:2003.07711 (2020).

Fu, Jun, et al. "Dual attention network for scene segmentation." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. (2019).

Geng, Mengyue, et al. "Deep transfer learning for person re-identification." arXiv preprint arXiv:1611.05244 (2016).

Gers, Felix A., Nicol N. Schraudolph, and Jürgen Schmidhuber. "Learning precise timing with LSTM recurrent networks." Journal of machine learning research 3.Aug: 115-143, (2002).

Gidaris, Spyros, Praveer Singh, and Nikos Komodakis. "Unsupervised representation learning by predicting image rotations." arXiv preprint arXiv:1803.07728 (2018).

Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. (2014).

Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, (2010).

Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems 27 (2014).

Goodfellow, Ian. "Nips 2016 tutorial: Generative adversarial networks." arXiv preprint arXiv:1701.00160 (2016).

Graves, Alex, et al. "Speech recognition with deep recurrent neural networks." 2013 IEEE international conference on acoustics, speech and signal processing. Ieee, (2013).

Graves, Alex, and Jürgen Schmidhuber. "Framewise phoneme classification with bidirectional LSTM networks." Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.. Vol. 4. IEEE, (2005).

Grenzdörffer, Till, Martin Günther, and Joachim Hertzberg. "YCB-M: a multi-camera RGB-D Dataset for object recognition and 6doF pose estimation." 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, (2020).

He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. (2016).

He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." Proceedings of the IEEE international conference on computer vision. (2015).

He, Kaiming, et al. "Mask r-cnn." Proceedings of the IEEE international conference on computer vision. (2017).

Hebb, Donald Olding. The organisation of behaviour: a neuropsychological theory. New York: Science Editions, (1949).

Henaff, Olivier. "Data-efficient image recognition with contrastive predictive coding." International Conference on Machine Learning. PMLR, (2020).

Hinterstoisser, Stefan, et al. "Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes." Asian conference on computer vision. Springer, Berlin, Heidelberg, (2012).

Hodan, Tomáš, et al. "T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects." 2017 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, (2017).

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8: 1735-1780, (1997).

Huang, Gao, et al. "Densely connected convolutional networks." Proceedings of the IEEE conference on computer vision and pattern recognition, (2017).

Hubel, David H., and Torsten N. Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex." The Journal of physiology 160.1: 106-154, (1962).

Chen, Liang-Chieh, et al. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs." IEEE transactions on pattern analysis and machine intelligence 40.4: 834-848, (2017).

Chen, Ting, et al. "A simple framework for contrastive learning of visual representations." International conference on machine learning. PMLR, (2020).

Chen, Weihua, et al. "A multi-task deep network for person re-identification." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 31. No. 1., (2017).

Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." arXiv preprint arXiv:1412.3555, (2014).

Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." International conference on machine learning. PMLR, (2015).

Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." Proceedings of the IEEE conference on computer vision and pattern recognition. (2017).

Jaderberg, Max, et al. "Reinforcement learning with unsupervised auxiliary tasks." arXiv preprint arXiv:1611.05397, (2016).

James, S. et al.: Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 12627-12637), (2019).

Ji, Zhong, Weixiong Zheng, and Yanwei Pang. "Deep pedestrian attribute recognition based on LSTM." 2017 IEEE International Conference on Image Processing (ICIP). IEEE, (2017).

Kang, Wang-Cheng, and Julian McAuley. "Self-attentive sequential recommendation." 2018 IEEE International Conference on Data Mining (ICDM). IEEE, (2018).

Kehl, Wadim, et al. "Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again." Proceedings of the IEEE international conference on computer vision. (2017).

Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980, (2014).

Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." arXiv preprint arXiv:1312.6114, (2013).

Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907, (2016).

Kirkpatrick, Scott, C. Daniel Gelatt, and Mario P. Vecchi. "Optimization by simulated annealing." science 220.4598: 671-680, (1983).

Koch, Gregory, Richard Zemel, and Ruslan Salakhutdinov. "Siamese neural networks for one-shot image recognition." ICML deep learning workshop. Vol. 2. (2015).

Kokic, Mia, Danica Kragic, and Jeannette Bohg. "Learning task-oriented grasping from human activity datasets." IEEE Robotics and Automation Letters 5.2: 3352-3359, (2020).

Kraus, Svatopluk, et al. "Detecting Wearable Objects via Transfer Learning." 2019 IEEE 15th International Conference on Intelligent Computer Communication and Processing (ICCP). IEEE, (2019).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25: 1097-1105, (2012).

Krull, Alexander, et al. "6-dof model based tracking via object coordinate regression." Asian Conference on Computer Vision. Springer, Cham, (2014).

Kumar, Abhishek, Prasanna Sattigeri, and P. Thomas Fletcher. "Semi-supervised learning with gans: Manifold invariance with improved inference." arXiv preprint arXiv:1705.08850 (2017).

Labbé, Yann, et al. "CosyPose: Consistent multi-view multi-object 6D pose estimation." European Conference on Computer Vision. Springer, Cham, (2020).

Laine, Samuli, and Timo Aila. "Temporal ensembling for semi-supervised learning." arXiv preprint arXiv:1610.02242 (2016).

Lake, Brenden, et al. "One shot learning of simple visual concepts." Proceedings of the annual meeting of the cognitive science society. Vol. 33. No. 33. (2011).

Lample, Guillaume, and Devendra Singh Chaplot. "Playing FPS games with deep reinforcement learning." Thirty-First AAAI Conference on Artificial Intelligence. (2017).

LeCun, Yann, et al. "Backpropagation applied to handwritten zip code recognition." Neural computation 1.4: 541-551, (1989).

Ledig, Christian, et al. "Photo-realistic single image super-resolution using a generative adversarial network." Proceedings of the IEEE conference on computer vision and pattern recognition. (2017).

Lee, Dong-Hyun. "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks." Workshop on challenges in representation learning, ICML. Vol. 3. No. 2. (2013).

Lepetit, Vincent, Francesc Moreno-Noguer, and Pascal Fua. "Epnp: An accurate o (n) solution to the pnp problem." International journal of computer vision 81.2: 155, (2009).

Levesque, Hector, Ernest Davis, and Leora Morgenstern. "The winograd schema challenge." Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning. (2012).

Li, Wei, Rui Zhao, and Xiaogang Wang. "Human reidentification with transferred metric learning." Asian conference on computer vision. Springer, Berlin, Heidelberg, (2012).

Li, Yi, et al. "Deepim: Deep iterative matching for 6d pose estimation." Proceedings of the European Conference on Computer Vision (ECCV). (2018).

Li, Yining, et al. "Human attribute recognition by deep hierarchical contexts." European Conference on Computer Vision. Springer, Cham, (2016).

Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).

Lin, Tsung-Yi, et al. "Microsoft coco: Common objects in context." European conference on computer vision. Springer, Cham, (2014).

Lin, Yutian, et al. "Improving person re-identification by attribute and identity learning." Pattern Recognition 95: 151-161, (2019).

Lin, Yutian, et al. "Improving person re-identification by attribute and identity learning." Pattern Recognition 95: 151-161, (2019).

Liu, Yinhan, et al. "Multilingual denoising pre-training for neural machine translation." Transactions of the Association for Computational Linguistics 8: 726-742, (2020).

Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. (2015).

Maas, Andrew L., Awni Y. Hannun, and Andrew Y. Ng. "Rectifier nonlinearities improve neural network acoustic models." Proc. icml. Vol. 30. No. 1. (2013).

Marion, Pat, et al. "Label fusion: A pipeline for generating ground truth labels for real rgbd data of cluttered scenes." IEEE International Conference on Robotics and Automation, (2018).

Medin, D. L., & Schaffer, M. M.. Context theory of classification learning. Psychological review, 85(3), 207, (1978).

Minsky, Marvin, and Seymour Papert. "An introduction to computational geometry." Cambridge tiass., HIT (1969).

Mishra, Nikhil, et al. "A simple neural attentive meta-learner." arXiv preprint arXiv:1707.03141 (2017).

Miyato, Takeru, et al. "Virtual adversarial training: a regularization method for supervised and semi-supervised learning." IEEE transactions on pattern analysis and machine intelligence 41.8: 1979-1993, (2018).

Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540: 529-533, (2015).

Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

Mordvintsev, Alexander, et al. "Inceptionism: Going deeper into neural networks." (2015).

Munich, Mario E., et al. "SIFT-ing through features with ViPR." IEEE robotics & automation magazine 13.3: 72-77, (2006).

Nguyen, Anh, et al. "Plug & play generative networks: Conditional iterative generation of images in latent space." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2017).

Nichol, Alex, et al. "On first-order meta-learning algorithms." arXiv preprint arXiv:1803.02999 (2018).

Odena, Augustus, et al. "Conditional image synthesis with auxiliary classifier gans." International conference on machine learning. PMLR, (2017).

Odena, Augustus. "Semi-supervised learning with generative adversarial networks." arXiv preprint arXiv:1606.01583 (2016).

Oord, Aaron van den, et al. "Representation learning with contrastive predictive coding." arXiv preprint arXiv:1807.03748 (2018).

Oquab, Maxime, et al. "Learning and transferring mid-level image representations using convolutional neural networks." Proceedings of the IEEE conference on computer vision and pattern recognition. (2014).

Park, Kiru, et al. "Pix2pose: Pixel-wise coordinate regression of objects for 6d pose estimation." Proceedings of the IEEE/CVF International Conference on Computer Vision. (2019).

Pathak, Deepak, et al. "Curiosity-driven exploration by self-supervised prediction." International Conference on Machine Learning. PMLR, (2017).

Pauwels, Karl, and Danica Kragic. "Simtrack: A simulation-based framework for scalable real-time object pose detection and tracking." 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, (2015).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12: 2825-2830, (2011).

Perozzi, Bryan, et al. "Deepwalk: Online learning of social representations." Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. (2014).

Pitteri, Giorgia, et al. "CorNet: generic 3D corners for 6D pose estimation of new objects without retraining." Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops. (2019).

Rad, Mahdi, and Vincent Lepetit. "Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth." Proceedings of the IEEE International Conference on Computer Vision. (2017).

Radford, Alec, et al. "Language models are unsupervised multitask learners." OpenAI blog 1.8: 9, (2019).

Radford, Alec, et al. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv preprint arXiv:1511.06434 (2015).

Rakelly, Kate, et al. "Few-shot segmentation propagation with guided networks." arXiv preprint arXiv:1806.07373 (2018).

Ramesh, Aditya, et al. "Zero-shot text-to-image generation." arXiv preprint arXiv:2102.12092 (2021).

Rasmus, Antti, et al. "Semi-supervised learning with ladder networks." arXiv preprint arXiv:1507.02672 (2015).

Ravi, Sachin, and Hugo Larochelle. "Optimization as a model for few-shot learning." (2016).

Ristani, Ergys, et al. "Performance measures and a data set for multi-target, multi-camera tracking." European conference on computer vision. Springer, Cham, (2016).

Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review 65.6: 386, (1958).

Rumelhart, David E., et al. "Learning representations by back-propagating errors." nature 323.6088: 533-536, (1986).

Rummery, Gavin A., and Mahesan Niranjan. On-line Q-learning using connectionist systems. Vol. 37. Cambridge, England: University of Cambridge, Department of Engineering, (1994).

Russakovsky, Olga, et al. "Imagenet large scale visual recognition challenge." International journal of computer vision 115.3: 211-252, (2015).

Salakhutdinov, Ruslan, and Hugo Larochelle. "Efficient learning of deep Boltzmann machines." Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, (2010).

Sedlar, Jiri and Matus Tuna et al. “Dataset and methods for training 6D Object Pose Estimator for Imitation Learning from RGB Video”, submitted to IEEE Transactions on Robotics, (2021).

Shaban, Amirreza, et al. "One-shot learning for semantic segmentation." arXiv preprint arXiv:1709.03410 (2017).

Schaul, Tom, et al. "Prioritized experience replay." arXiv preprint arXiv:1511.05952 (2015).

Silver, David, et al. "Deterministic policy gradient algorithms." International conference on machine learning. PMLR, (2014).

Silver, David, et al. "Mastering the game of go without human knowledge." nature 550.7676: 354-359, (2017).

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

Snell, Jake, et al. "Prototypical networks for few-shot learning." arXiv preprint arXiv:1703.05175 (2017).

Sung, Flood, et al. "Learning to compare: Relation network for few-shot learning." Proceedings of the IEEE conference on computer vision and pattern recognition. (2018).

Sutton, Richard S. "Learning to predict by the methods of temporal differences." Machine learning 3.1: 9-44, (1988).

Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. (2015).

Szegedy, Christian, et al. "Inception-v4, inception-resnet and the impact of residual connections on learning." Thirty-first AAAI conference on artificial intelligence. (2017).

Tarvainen, Antti, and Harri Valpola. "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results." arXiv preprint arXiv:1703.01780 (2017).

Taylor, Luke, and Geoff Nitschke. "Improving deep learning using generic data augmentation." arXiv preprint arXiv:1708.06020 (2017).

Tian, Yonglong, et al. "Contrastive multiview coding." arXiv preprint arXiv:1906.05849 (2019).

Tremblay, Jonathan, et al. "Deep object pose estimation for semantic robotic grasping of household objects." arXiv preprint arXiv:1809.10790 (2018).

Tuna, Matus et al. “Semi-supervised Learning in Camera Surveillance Image Classification”, submitted to IEEE International Conference on Intelligent Computer Communication and Processing, (2021).

Tuna, Matus, et al. “Few-shot semantic segmentation using REPTILE meta-learning approach” Kognícia a umelý život. KUZ, (2019).

Tuna, Matus, et al. “Kategorické Siamské Siete” Kognice a umělý život. KUZ, (2018).

Van der Maaten, Laurens, and Geoffrey Hinton. "Visualizing data using t-SNE." Journal of machine learning research 9.11 (2008).

Van Hasselt, Hado, et al. "Deep reinforcement learning with double q-learning." Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1. (2016).

Van Oord, Aaron, et al. "Pixel recurrent neural networks." International Conference on Machine Learning. PMLR, (2016).

Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. (2017).

Veličković, Petar, et al. "Graph attention networks." arXiv preprint arXiv:1710.10903 (2017).

Vinyals, Oriol, et al. "Matching networks for one shot learning." Advances in neural information processing systems 29: 3630-3638, (2016).

Wang, Jingya, et al. "Transferable joint attribute-identity deep learning for unsupervised person re-identification." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2018).

Weiss, Karl, et al. "A survey of transfer learning." Journal of Big data 3.1: 1-40, (2016).

Werbos, Paul J. "Backpropagation through time: what it does and how to do it." Proceedings of the IEEE 78.10: 1550-1560, (1990).

Werbos, Paul. "Beyond regression:" new tools for prediction and analysis in the behavioral sciences." Ph. D. dissertation, Harvard University (1974).

Wu, Yonghui, et al. "Google's neural machine translation system: Bridging the gap between human and machine translation." arXiv preprint arXiv:1609.08144 (2016).

Xiang, Liuyu, et al. "Incremental few-shot learning for pedestrian attribute recognition." arXiv preprint arXiv:1906.00330 (2019).

Xiang, Yu, et al. "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes." arXiv preprint arXiv:1711.00199 (2017).

Yalniz, I. Zeki, et al. "Billion-scale semi-supervised learning for image classification." arXiv preprint arXiv:1905.00546 (2019).

Yu, Kai, et al. "Weakly-supervised learning of mid-level features for pedestrian attribute recognition and localization." arXiv preprint arXiv:1611.05603 (2016).

Zakharov, Sergey, et al. "Dpod: 6d pose object detector and refiner." Proceedings of the IEEE/CVF International Conference on Computer Vision. (2019).

Zeiler, Matthew D. "Adadelta: an adaptive learning rate method." arXiv preprint arXiv:1212.5701 (2012).

Zeiler, Matthew D., et al. "Deconvolutional networks." 2010 IEEE Computer Society Conference on computer vision and pattern recognition. IEEE, (2010).

Zhai, Xiaohua, et al. "S4l: Self-supervised semi-supervised learning." Proceedings of the IEEE/CVF International Conference on Computer Vision. (2019).

Zhang, Han, et al. "Stackgan++: Realistic image synthesis with stacked generative adversarial networks." IEEE transactions on pattern analysis and machine intelligence 41.8: 1947-1962, (2018).

Zhang, Richard, Phillip Isola, and Alexei A. Efros. "Colorful image colorization." European conference on computer vision. Springer, Cham, (2016).

Zheng, Liang, et al. "Scalable person re-identification: A benchmark." Proceedings of the IEEE international conference on computer vision. (2015).

Zhong, Zhun, et al. "Random erasing data augmentation." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 34. No. 07. (2020).