BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

SELF-SUPERVISED LEARNING FOR RECOGNITION OF SPORTS POSES IN IMAGE

POUŽITÍ SELF-SUPERVISED LEARNING PRO ROZPOZNÁNÍ SPORTOVNÍCH POZIC V OBRAZE

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. DANIEL KONEČNÝ**
AUTOR PRÁCE

SUPERVISOR **prof. Ing. ADAM HEROUT, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2022

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM) Academic year 2021/2022

Master's Thesis Specification

Student: **Konečný Daniel, Bc.**
Programme: Information Technology and Artificial Intelligence
Specialization: Machine Learning
Title: **Self-Supervised Learning for Recognition of Sports Poses in Image**
Category: Image Processing
Assignment:

1. Study the field of machine learning for computer vision and recognition of sports poses in image and video.
2. Obtain and/or collect a data set (sets) of images of sports poses.
3. Experiment with methods of self-supervised learning on the collected data set (sets).
4. Demonstrate the usability of the developed techniques for recognition of sports poses.
5. Iteratively improve the developed techniques and the data set towards maximal usability.
6. Discuss the achieved results and propose possibilities for future work on the project; create a poster and a short video for presenting the results of the project.

Recommended literature:

- Goodfellow, Bengio, Courville: Deep Learning, MIT Press, 2016
- Bharath Ramsundar, Reza Bosagh Zadeh: TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning, O'Reily Media, 2018
- Gary Bradski, Adrian Kaehler: Learning OpenCV; Computer Vision with the OpenCV Library, O'Reilly Media, 2008
- Richard Szeliski: Computer Vision: Algorithms and Applications, Springer, 2011
- Grill J-B et al.: Bootstrap your own latent: A new approach to self-supervised Learning, NeurIPS 2020, https://arxiv.org/abs/2006.07733
- Caron M et al.: Emerging Properties in Self-Supervised Vision Transformers, https://arxiv.org/abs/2104.14294
- Sermanet et al.: Time-Contrastive Networks: Self-Supervised Learning from Video, ICRA 2018, https://arxiv.org/abs/1704.06888
- Asano et al.: Self-labelling via simultaneous clustering and representation learning, ICLR 2020, https://arxiv.org/abs/1911.05371
- L. Jing, Y. Tian, Self-supervised visual feature learning with deep neural networks: A survey, IEEE PAMI, 2020

Requirements for the semestral defence:

- Items 1 and 2, considerable development on items 3 through 5.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Herout Adam, prof. Ing., Ph.D.**
Head of Department: Černocký Jan, doc. Dr. Ing.
Beginning of work: November 1, 2021
Submission deadline: July 29, 2022
Approval date: November 1, 2021

Abstract

The goal of this thesis is to recognize sports poses in image data with a self-supervised learning approach to achieve high classification accuracy even with a low number of annotated samples. Self-supervision is obtained by using images of the same scene from multiple viewpoints at identical and different times. A convolutional neural network trained with triplet loss learns embedding vectors of sports poses and a dense neural network classifies them. The proposed self-supervised model achieves classification accuracy higher by 30–40 % than a supervised model when there are only tens or ones of annotated training samples from each class. The main contributions of this thesis are a set of semi-automatic tools to prepare a dataset for the specific training process, two datasets with sets of labels for classification, and implemented models for specific self-supervised learning. The results show that self-supervised learning is a meaningful approach for solving classification problems with very few labeled samples.

Abstrakt

Cílem této práce je rozpoznání sportovních pozic v obrazových datech za pomocí přístupu self-supervised learning pro docílení vyšší úspěšnosti klasifikace s použitím malého množství anotovaných vzorků. Učení za pomocí self-supervision je docíleno snímky stejné scény z různých úhlů ve stejných a různých časech. Konvoluční neuronová síť naučená s pomocí funkce triplet loss zakóduje sportovní pozice do latentních vektorů a plně propojená neuronová síť tyto vektory klasifikuje. Model natrénovaný pomocí self-supervised learning dosahuje o 30–40 % vyšší úspěšnosti než supervised model, když je trénovaný pouze na desítkách či jednotkách označených snímků z každé třídy. Hlavními přínosy této práce jsou nástroje pro přípravu datové sady pro tento specifický typ učení, dvě datové sady s více anotacemi a implementované modely využívající self-supervised learning. Výsledky ukazují, že učení za pomocí self-supervision je vhodný přístup pro řešení klasifikace za použití velmi malého množství označených snímků.

Keywords

machine learning, computer vision, recognition, convolutional neural network, image, self-supervised learning, time-contrastive learning, sports pose

Klíčová slova

strojové učení, počítačové vidění, rozpoznání, konvoluční neuronová síť, obraz, self-supervised learning, time-contrastive learning, sportovní pozice

Reference

KONEČNÝ, Daniel. *Self-Supervised Learning for Recognition of Sports Poses in Image.* Brno, 2022. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Ing. Adam Herout, Ph.D.

Rozšířený abstrakt

Práce se zabývá rozpoznáváním sportovních pozic v obrazových datech. Pro naučení modelu, který rozpoznání provádí, je použita metoda self-supervised learning, ve které se využívá nějaké informace, která je v trénovacích datech už obsažena a není třeba ji doplnit dodatečně. Trénovací data pak sama slouží modelu jako supervizor a ten se díky tomu naučí data smysluplně reprezentovat v latentním prostoru. Kvalitní reprezentace pak obsahují informace hodnotné k dalšímu využití a ignorují informace, které v originálních datech nenesly žádnou hodnotu. Sportovní pozice zakódovaná v latentním vektoru je pak jednodušeji klasifikovatelná a je tedy potřeba méně trénovacích snímků.

V této práci je self-supervised learning docílen metodou time-contrastive learning, která využívá nahrávek jedné scény z více různých úhlů. Sportovní pozice v jednom konkrétním čase, ačkoliv je zachycená z různých úhlů, je stále stejná. Zatímco pozice zachycená ze stejného místa, ale po tom, co se odehrál v obraze nějaký pohyb, je jiná. Tato nápomocná informace je ve videích obsažena automaticky, pokud jsou synchronizovaná v čase, a není tedy třeba jakéhokoliv dalšího označování snímků pro naučení dobré reprezentace sportovních pozic.

Pro co nejjednodušší přípravu takových dat je představena sada nástrojů, která videa přepracuje do snímků vhodných přímo k tréninku. Nástroje pracují téměř automaticky, vyžadují pouze minimum interakce s uživatelem. V první řadě uživatel určí začátek a konec jednoho z nahraných videí, aby na začátku a konci nebyly nežádoucí záběry. Jedno video je dostatečné, neboť synchronizační nástroj, který bude představen později, se postará o zkrácení těch ostatních. Následně uživatel vybere ve videu ohraničující obdélník pro oříznutí na oblast zájmu. Ořezávací nástroj se postará o to, aby výsledné snímky měly správné rozlišení pro vstupní vrstvu neuronové sítě. Poté jsou videa plně automaticky synchronizována pomocí dense optical flow, kdy je nalezen překryv videí s největší korelací směru pohybu ve videích a následně jsou videa patřičně zkrácena. Nakonec je použit detektor pohybu pro rozpoznání, zda se v obrazu udál dostatek změn, aby mohly být snímky brány jako rozdílné a uloženy pro trénink jako obrázky. Detektor využívá sparse optical flow a následné zpracování vektorů pohybu, aby co nejpřesněji detekoval snímky s rozdílnými sportovními pozicemi.

Dále jsou prezentovány 2 datové sady, které byly použity při vývoji a následném testování nástrojů a modelů v této práci. První z nich obsahuje pohyby ruky natočené od předloktí níže k prstům s tím, že se ruka pohybuje převážně v zápěstí a v prstech. Tato datová sada byla použita pro vývoj a testování nástrojů pro přípravu datové sady pro time-contrastive learning. Druhá datová sada je tvořena nahrávkami člověka převážně od kolen výše, ve kterých provádí různé sportovní pózy pomocí pohybů pažemi. Ve videích se nijak nemění pozice těla ani hlavy, pouze se ohýbají paže v ramenech a loktech. Videa jsou natáčena ve dvou prostředích a osoba na nich zachycená má různé oblečení pro dosažení větší různorodosti. Tato datová sada byla připravena pomocí zmíněných nástrojů a následně použita pro vývoj a vyhodnocení modelu trénovaného pomocí self-supervised learning.

Model pro klasifikaci sportovních pozic se skládá ze dvou částí: enkodér a klasifikátor. Enkodér je konvoluční neuronová síť a má za úkol nalézt co nejhodnotnější reprezentace sportovních pozic. Jeho architektura je založena na síti ResNet-50 s upraveným výstupem a je učený za pomocí funkce triplet loss. Klasifikátor přijímá na vstup latentní vektory v předem určené dimenzionalitě a jeho výstupem jsou pravděpodobnosti příslušnosti tohoto vektoru k jednotlivým třídám. Jedná se také o neuronovou síť, ale pouze s jednou skrytou plně propojenou vrstvou s nelineární aktivační funkcí, neboť řešený úkol už je jednoduchý a nemusí být poskytnuto mnoho trénovacích dat. Nakonec je také implementován model

konvoluční neuronové sítě s téměř ekvivalentní architekturou, pouze učený se supervizí. Tato síť slouží k porovnání úspěšnosti přístupů self-supervised a supervised learning na různě velkých datových sadách.

Implementované modely jsou vyhodnoceny pomocí různých metrik a následně je diskutována jejich úspěšnost v řešení úkolů, ke kterým byly určeny. Nejprve je provedena vizuální analýza latentního prostoru se zakódovanými pozicemi po tom, co je redukována dimenzionalita pomocí t-distributed stochastic neighbor embedding. Z této vizualizace vyplývá, že se enkodér zvládne naučit reprezentace sportovních pozic i s jejich příslušnými vlastnostmi, avšak na dané datové sadě nedokáže kvalitně generalizovat mezi jednotlivými scénami a osobami. Dále je provedeno vyhodnocení zakódovaných pozic na prostorech o různých dimenzích, ze kterého vyplývá, že danou datovou sadu je nejvhodnější reprezentovat v 64 dimenzích. Nakonec je porovnán model trénovaný pomocí self-supervised learning s modelem trénovaným pomocí supervised learning na různě velkých datových sadách. Z experimentu vyplývá, že při použití pouhých desítek či jednotek označených snímků v jednotlivých třídách má self-supervised model o 30–40 % vyšší úspěšnost rozpoznání sportovní pozice. Při použití vyššího množství trénovacích dat je self-supervised model lepší než supervised model, či alespoň dosahuje stejných výsledků.

Self-Supervised Learning for Recognition of Sports Poses in Image

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of prof. Ing. Adam Herout, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Daniel Konečný
July 23, 2022

Acknowledgements

I would like to express my sincere gratitude to my supervisor prof. Ing. Adam Herout, Ph.D. for his valuable advice, professional guidance, and important feedback. I would also like to thank Mgr. Lada Konečná for helpful consultations of mathematical topics.

Contents

Chapter 1

Introduction

Learning algorithms of many models used for recognition are nowadays relying on large datasets and also high computational power. Whereas the computational power of computers is still expected to increase, the creation of larger and more accurate annotated datasets is a task much harder. Labeling usually requires some sort of manual work that cannot be automated unless the required task was previously solved. This problem can be minimized by using unlabeled data to learn their representations and then needing only a smaller amount of labeled samples to provide good classification results. This method is referred to as self-supervised learning.

The introduction to machine learning and convolutional neural networks is covered at the beginning of Chapter 2. Then, important features of convolutional neural networks that are used in this thesis are described. After that, current advances in sports pose recognition are presented. The following section introduces self-supervised learning and its advantages and disadvantages in deep learning. Finally, various loss functions for self-supervised learning are discussed.

This thesis discusses self-supervision achieved with a time-contrastive approach when one scene is filmed from multiple viewpoints and the difference in time and camera position is used as supervision that needs no labeling. The recorded video footage has to be normalized to the network's input, synchronized, and then motion detection is needed to produce individual training images. This whole process is specified in detail in Chapter 3. A set of tools for semi-automatic preparation of such dataset is proposed. Finally, 2 different datasets with multiple sets of labels are presented. The datasets were recorded for this thesis and were prepared with the introduced tools.

The classifier model can be divided into 2 parts: the first one encodes the image into an embedding and the second one recognizes the sports pose from the embedding. Chapter 4 describes both of these models and their specific parameters. Inputs of the first model are images with sports poses and outputs are their representations – embeddings in a form of latent vectors. The first model is a convolutional neural network trained with triplet loss function, the second model is a simple neural network with densely connected layers, which classifies the embeddings into given categories.

Evaluation of the performance of all described models is provided in Chapter 5. At first, class distribution is visually analyzed with t-distributed stochastic neighbor embedding to better understand the embedding space. Afterward, the models are evaluated on validation data and their different settings are compared to each other. Finally, the overall performance of the self-supervised model on sports poses is measured and compared to the supervised model. All evaluations are discussed and their key takeaways are emphasized.

Chapter 2

Neural Networks in Computer Vision

Machine learning (ML) is playing a great role in solving many tasks proposed by the field of computer vision (CV) [19]. Analyzing objects from images is a task widely considered as not definable by a set of conditions and therefore, some sort of artificial intelligence (AI) has to be used, as is further discussed in Section 2.1. When problems got more complicated, ML models had to be increased in depth, which led to the introduction of the term deep learning (DL).

Neural networks (NNs) became a very powerful computing architecture in machine learning. Section 2.2 informs how the introduction of convolutional layers to neural networks further increased their efficiency in image processing. Various other operations were also implemented into convolutional neural networks (CNNs) to fit the needs of specific CV challenges.

Deep learning has untangled many problems which were not solvable with other methods. Recognition, being one of these problems, requires a large amount of training data when the model is using supervised learning. This matter is further described in Section 2.3. The quality of annotated data available for the training process has a great influence on the model performance and as such became a challenge in the further development of recognition models.

Therefore, learning algorithms that do not require large datasets are being developed. One of these approaches is self-supervised learning introduced in Section 2.4. It extensively uses a large amount of data without any labels as these are easier to obtain. A model trained in this manner needs a smaller amount of annotated data to generalize. Such an attribute is very valuable since labeling can often be done only by hand and is very time-consuming.

2.1 Machine Learning

Computers are very good at solving problems that can be specified with a set of conditions and states, such as a game of chess. The computational power that it possesses allows it to analyze the game many steps ahead. Whereas the human mind finds these tasks very complicated. It is, therefore, no surprise that computers managed to defeat the best chess players in the world. That is why one of the ways to create an artificial intelligence (AI) is focused on the knowledge base approach. The main hypothesis was that every problem

can be described in a formal language. After designing a set of logical inference rules, the problem would be solved with just a simple inference.

Unfortunately, when it comes to solving real-world problems, this approach cannot be applied. Humans have immense knowledge about the world and applying it is very subjective and intuitive. It cannot be formalized in any way. The knowledge base AI often did not understand the problem correctly and provided misleading results. Another disadvantage is that the formalization itself was an unwieldy process requiring a large amount of human staff [5].

Different approaches had to be chosen to solve real-world tasks. Instead of modeling the real world with conditions and rules, probabilistic models with a set of parameters were chosen. Most parameters are to be set automatically based on the data provided to learn the problem's nature. Logistic regression is one of these basic models that provide subjective reasoning based on the information that it learned from previous real-world examples. It finds a correlation between inputs and various outcomes. The computation involves a weighted sum of the inputs and a non-linear transformation of this sum, illustrated in Figure 2.1. Parameters that are adjusted are the weights of each input and a bias, a single number that is added to the sum, which can be also seen as a weight to a constant input of +1.

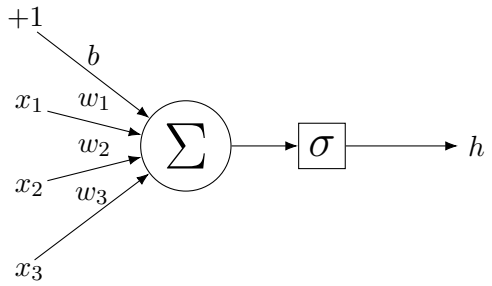


Figure 2.1: Logistic regression as an elementary model in machine learning. $x_k, k \in \{1, 2, 3\}$ are the inputs and h is the resulting output that equals the determined probability of the input.

Inputs of these models are called features of the data and the performance of the model heavily depends on their representation. If the features correlate with the different outcomes, the model is expected to provide good results. If we wanted to recognize a sports pose and the data provided would be positions of all joints and both eyes in a human body, the task would be fairly easy. After normalizing the scene to always have the same scale and point straight to the eyes, the task gets even simpler. Inputs of such model would be coordinates of mentioned objects and outputs probabilities of each sports pose. This solution introduces another lot more complicated challenge – the coordinates are hard to obtain without any special tools. What we would like to have is a model that can work on simple image data since obtaining those is affordable.

Images can be described with pixel values and provided to the input but individual pixels have no direct correlation to sports poses, therefore, the predictions would be useless. There is a number of examples of why this is true but perhaps the easiest one is translational dependence. Having the sports pose moved just a few pixels in any direction from where it is expected to be, makes the results incorrect. Additional problems would be caused by shadows, different clothes that the person is wearing, etc.

This obstacle can be overcome by having the ML model discovering not only the mapping of features to outputs but also finding the useful features in the raw data on its own. This makes the model not only work on raw data but also generalizes it to different tasks. For example, not only recognizing sports poses but also vehicles with the same model only trained on different data. Logistic regression is not capable of doing such predictions, some more complex solution has to be found. In computer science, the concept of building complex structures from simple modules is well known and can be used in machine learning as well. By combining many logistic regressions into a structure, a neural network (NN) is created [5].

These networks consist of neurons – logistic regressions. Its non-linear function can be adjusted to fit the needs of a specific task and it is often referred to as an activation function. The capability to solve complex problems arises from the structuring of simple neurons into groups called layers. The key is not doing the mapping of abstract features in one task but dividing it into multiple simple mappings. The input layer, also called the visible layer because its data can be easily observed, provides data to the following layer. The first simple mapping is done by the second layer and each following layer uses the mappings of its predecessor to obtain more complex information from the data. These following layers are also called hidden because their values are not given in the data, they have to be determined by the model. Finally, the last layer provides outputs in a format specified by its activation function.

With tasks becoming more complicated, the number of layers is growing bigger. As there was an increase in the depth of the graph, such networks were called deep neural networks, and their usage was referred to as deep learning (DL).

Neural network for classification can be seen as function $\boldsymbol{y} = f(\boldsymbol{x}|\boldsymbol{\theta})$ that maps input vector $\boldsymbol{x}$ to a category $\boldsymbol{y}$ with parameters of the network given by $\boldsymbol{\theta}$. It is also possible to decompose the neural network function f to multiple functions, each one representing one layer, applied in the correct order. NN with two hidden layers $f^{(1)}$, $f^{(2)}$ and one output layer $f^{(3)}$ is representing function:

$$\boldsymbol{y} = f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x}))). \tag{2.1}$$

Network parameters $\boldsymbol{\theta}$ are basically weights and biases of each neuron and they are unknown when the model is constructed. The ideal values of parameters cannot be computed in a simple way because of the non-linearity of neural networks that causes most of the loss functions to become non-convex. Loss functions are going to be further explained later in this section. NN parameters have to be somehow initialized and iteratively improved to provide better results. This iterative process is called learning or training and its goal is to approximate some function f^* that provides accurate results for a given problem. That is achieved by finding parameters $\boldsymbol{\theta}$ that result in such approximation [5].

Non-linear results of neural networks are achieved with activation functions. There are lots of various functions with different use cases, but two of them are very common. One is a sigmoid function with Equation 2.2, which maps any real number to a number between 0 and 1. It is often used to represent probability. The other activation function is rectified linear unit (ReLU) (Equation 2.3) that is linear for any positive number and 0 otherwise. ReLU is very often used in later discussed convolutional neural networks.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

$$f(x) = \max(0, x) \tag{2.3}$$

The learning algorithm consists of multiple stages that repeat until the model is producing satisfactory results. These stages are explained in detail in the following paragraphs.

1. Forward propagating data – inference.
2. Computing gradients with back-propagation algorithm.
3. Calculating learning rate.
4. Performing learning step of the model with the optimization algorithm.

Evaluation $\boldsymbol{y}$ of samples $\boldsymbol{x}$ is computed by forward propagating the samples through the network. That means evaluating all layers in the correct order as illustrated in Equation 2.1. Correct values $\boldsymbol{y}^* = f^*(\boldsymbol{x})$ are known because the training data are annotated with them. The loss (or cost) function can be used to compute how good the approximation f of f^* is. Loss functions are designed to fit specific tasks and data distribution. When it is necessary to compute some sort of distance on data that probably come from Gaussian distribution, the Mean Squared Error function is often used.

For classification problems, the typical choice is a measure called cross-entropy. It is based on the Kullback–Leibler divergence, which measures the difference between two probability distributions. Cross-entropy computes the expected number of bits needed to represent data coming from the distribution p while using the distribution q and it is calculated as follows [11]:

$$\mathbb{H}(p, q) \triangleq -\sum_{y} p(y) \log q(y). \tag{2.4}$$

Gradients can be computed in many different ways but the most common one for models that are working on large datasets is stochastic gradient descent (SGD). Therefore, this is the only one discussed in this thesis. Generally, a gradient is a vector pointing in the direction of the steepest ascent. By following such a vector, the local maximum can be reached. In machine learning, the thought is often reversed – the goal is to reach the local minimum, but the main idea remains the same. For the number of samples m and loss function L, gradient $\boldsymbol{g}$ is computed with this equation:

$$\boldsymbol{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}^{(i)}|\boldsymbol{\theta}), y^{*(i)}). \tag{2.5}$$

The direction of the next step is computed but another variable called the learning rate is still unknown. It represents the size of the step and it has a vast impact on the training performance. One possible solution is to keep the learning rate fixed for the whole training but better results can be achieved with more advanced algorithms. The first improvement can be achieved by computing a specific learning rate for each parameter of the network. The second way to achieve better results is by changing the learning rate throughout the training process.

The update of parameters is done with an optimization algorithm. It uses previously computed gradient $\boldsymbol{g}$ and other algorithm-specific parameters to update the network's parameters. It usually incorporates the calculation of the learning rate. Very common is

the use of the Adam optimization algorithm which also uses the previously mentioned improvements for a more useful learning rate. The algorithm uses the mean and uncentered variance of parameters to adapt the learning rates. The computation goes as follows [9].

$$\boldsymbol{s} = \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g} \tag{2.6}$$

$$\boldsymbol{r} = \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g} \tag{2.7}$$

$$\hat{\boldsymbol{s}} = \frac{\boldsymbol{s}}{1 - \rho_1} \tag{2.8}$$

$$\hat{\boldsymbol{r}} = \frac{\boldsymbol{r}}{1 - \rho_2} \tag{2.9}$$

$$\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta} \tag{2.10}$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta} \tag{2.11}$$

Where:

ρ_1, ρ_2 are exponential decay rates for moment estimates (mean and variance, usually initialized to 0.9 and 0.999 respectively),

$\odot$ is an element-wise product,

$\boldsymbol{s}$ is an updated biased first-moment estimate,

$\boldsymbol{r}$ is an updated biased second-moment estimate,

$\hat{\boldsymbol{s}}$ is a correct bias in the first moment,

$\hat{\boldsymbol{r}}$ is a correct bias in the second moment,

ϵ is a step size (usually initialized to 0.001) and

δ is a small constant used for numerical stabilization (usually initialized to 10^{-8}).

Another important part of the training process is how the data samples are handled. It is possible to update the network's parameters after each sample but also with the whole dataset. The ideal solution is to divide the dataset into minibatches of size ranging from lower tens to higher hundreds of samples. Parameters are then updated with each minibatch. After all minibatches of the dataset have been used for training, the process can start again on the previously minibatches. It is also important to shuffle the data in the dataset and in the minibatches. If the same order of samples was used, the network might have problems with not generalizing enough.

Every time the whole dataset has been handled, one epoch has passed. Training can consist of many epochs, depending on the problem difficulty, network size, and dataset. It is important to measure the network's performance on data it has never seen during the training. Once the model's accuracy is not improving and/or loss is approaching nearly zero values, the training will no longer provide better results. Therefore, the dataset should be divided into training and validation data.

2.2 Convolutional Neural Networks

Convolutional neural networks are a special kind of NNs including at least one layer that is computing convolution. These networks are used for processing data with grid-like topologies, such as sequences and images. This thesis focuses on image data and therefore, only

those will be discussed further on, even though the computation can be generalized to other input types.

At first, convolution is discussed as an operation on image data with its important properties. Its usage as a layer in a neural network is explained in detail. Then, other operations important for CNNs are introduced and explained. Once most of the important principles of convolutional neural networks have been mentioned, a specific convolutional neural network architecture is presented.

2.2.1 Convolution on Image Data

Convolution is a mathematical operation of two functions that produces a third function that describes how one modifies the other in shape. This is a very general definition that is not necessary for image processing and can be made more specific. It is only necessary to consider discrete values of inputs, continuous functions are not used in CNNs. Images usually consist of multiple channels (typically red, green, and blue), but for convolutional neural networks, channels are handled separately. For that reason, images will be discussed as 2-dimensional arrays of numbers only.

Convolution computes a weighted sum of values across a fixed-size area of the image. It takes a 2-D image input and a 2-D array of weights called a kernel. Images can be extended on the edges with padding, which are basically pixels with a value of zero. Since convolution changes the size of the input image, padding is often used to equalize the sizes [4]. The resulting 2-D array is often referred to as a feature map and it is computed by multiplying the input value with the corresponding kernel value for all of the overlapping elements and then summed together. After that, the kernel moves one step further on the input and the next value of the feature map is calculated the same way until the whole input is processed. Figure 2.2 illustrates one step of the computation.

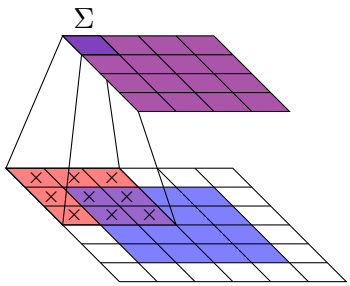


Figure 2.2: Example of 2-dimensional convolution with input size 4 (blue), kernel size 3 (red) and padding size 1 (white). The feature map (dark purple) has the same size as the input because of the padding.

The operation of convolution is often denoted with an asterisk $*$ and for input I of size $m \times n$ and kernel K, feature map FM is calculated as:

$$FM(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{2.12}$$

Layers that perform convolution are not that different from normal dense layers mentioned in the previous section. The input image is the layer's input, weights are the kernel values and the output of the layer is the feature map. When training is performed, the goal is to find kernel values that produce the best results. Kernels are usually called filters in

the CNN context, therefore, this terminology will be used from now on. Each convolutional layer often includes more filters and produces an equal number of feature maps, one feature map from each filter applied to the input. That means 2-D input data are transformed into 3-D data, as there are multiple 2-D feature maps of the same size. The following convolutional layer applies filters to each input and sums the results over each filter. Color images on the input are handled the same way as if each channel was a feature map from a previous convolutional layer, there is no difference between them.

Convolution is a very important operation for image processing because it holds many essential properties. Since it is computed over multiple neighboring values, the context of pixel values is taken into account, not just the single values. That enables pattern recognition in images. Simple patterns such as shadow information and edges are layer by layer combined into more complex patterns until an object detection can be done. Another important property is equivariance, which means that the position of objects in the image plays no role in detection. Finally, convolution has low memory requirements that are not dependent on the input size, only values that are stored are 2-D filter arrays [5].

2.2.2 Additional Important Parts of Convolutional Neural Networks

Convolutional layers are usually followed by *pooling* layers in CNN architecture. Pooling is a function that for each value of the input grid computes a summary statistic of its nearby values. The most common statistics are maximum and average. For example, the max pooling layer with 2×2 pool size takes a maximum of every 2×2 region in the image and creates a new image constructed out of the maximum values. A simplified version of this operation is in Figure 2.3. In this case, the output size will be smaller than the input size. To keep the size uniform, padding must be added the same way it was added during convolution.

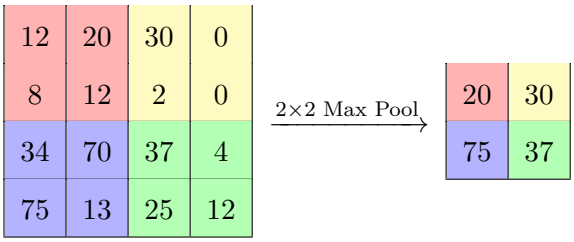


Figure 2.3: Max pooling of 4×4 grid into a 2×2 grid with pool size also 2×2. From every region, the maximal value is taken to the output.

Pooling layers make the model invariant to small translations. Even when the input has moved a few pixels in some direction, pooled outputs should not change much. Pooling can be also easily used for downsampling of images when the stride is set to 2 or more. Another use case of pooling is handling images of varying sizes because classifiers are accepting only images with fixed size [5].

Residual blocks made a vast impact on the development of convolutional neural networks. While the recognition problems got more complicated and datasets enormous, the need to make CNNs deeper arose. Unfortunately, the performance of the networks did not improve by just adding more layers. Gradients could not be back-propagated all the way to initial layers.

The solution to this problem came with residual blocks that introduced a simple connection that bypassed blocks with convolution and pooling, as shown in Figure 2.4. This

connection symbolizes a simple identity function, it takes the input and outputs it unchanged. When such an identity connection bypasses every convolutional block, the neural network can basically work as an identity function. The same principle can be applied when gradients are computed, therefore, larger gradients are back-propagated to the initial layers [6].

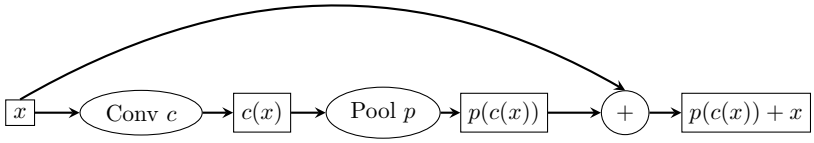


Figure 2.4: Residual block with bypassed convolutional and pooling layers. Rectangular nodes symbolize data and ellipsoidal nodes are operations.

Dropout layer is used to deactivate some neurons during the training process and by that, it is trying to simulate different models. For each mini-batch of data, some percentage of neurons have their activation function set to zero. The training step is done as usual: inference, back-propagation, and weight update. Then, for the next mini-batch, different neurons are chosen to produce output with a value of zero. Dropout is a computationally inexpensive way to regularize models [5].

2.3 Recognition as a Supervised Learning Task

Recognition is one of many computer vision tasks and it can be further divided into multiple more specific categories. The most common one is classification – the recognized image is supposed to be assigned a class from a previously known set. This thesis focuses specifically on sports poses classification, however, the other recognition varieties are worth mentioning as well. Detection and segmentation challenges are trying to localize objects in images with a bounding box or pixel-wise, respectively. A very specialized task is a pose estimation where the model is trying to assign a specific structure of connected joints to a human body.

Supervised learning is one of the most general training methods. The main prerequisite is generally a dataset with annotated samples. For the classification task, each data point has to have a class assignment. During the training procedure, the model is trying to assign the correct class to each sample in a mini-batch and compare it to ground truth – the real class of the sample saved in the dataset. If the model is not successful, the information is back-propagated through the network to improve on the next mini-batch inference.

Nowadays, models are trained on datasets of millions or even billions of data points. Networks with a number of layers well over 100 have enough parameters to be able to classify very complicated images into a large number of categories (even tens of thousands of hierarchically divided ones). An important property of each classifier is its ability to generalize – to classify correctly images it has never seen. Generalization is achieved with training on immense datasets with a large variety of images.

2.3.1 ResNet Architecture of Convolutional Neural Networks

This thesis implements residual network ResNet-50 from [6] as a backbone, a structure of convolutions, pooling, and other operations to obtain information from images. This

network was chosen because of its well-known architecture which still provides good results. It can be easily compared to other results since it is widely used across the whole computer vision field. Its depth and number of parameters are not as high as the newer architectures have and therefore, might be easier to train on data. As an alternative, a MobileNet-V2 architecture was also tested [13]. Although it learned faster, its results did not achieve the ResNet's accuracy. It is easily possible to use other CNN architecture as a backbone, the only difference might be the input size that has to be adjusted.

The network uses a well-known structure of convolutional layers followed by pooling layers, where the number of convolutional filters corresponds to the size of the stride and the number of filters in the previous layer. When the stride size is 1, the input and outputs of convolution have also the same size and therefore, the number of filters stays the same. If the image size is decreased with stride size equal to 2, the number of filters doubles. In the beginning, convolution with filter size 7×7 is used but afterward, standard 3×3 convolutions are implemented.

The head of the network has a densely connected layer with a softmax activation function. This network introduces residual blocks further explained together with the other mentioned concepts in Section 2.2. This allows the network to have higher tens or even lower hundreds of layers and still be able to learn well. The network also uses batch normalization after each convolution but before the activation. With the exception of the network's output, Rectified Linear Unit is used as an activation function. An example of the architecture in a simplified form is in Table 2.1.

2.3.2 Current Advances in Sports Pose Recognition

Most of the current research focuses on sports pose estimation which is a different task than sports pose classification done in this thesis. Researchers also tried yoga pose recognition from body contours but the variety of poses did not draw near to all possible options [2]. However, with the publishing of the Yoga-82 dataset, results of different supervised-trained models were also analyzed [20]. Summarized in Table 2.2 are the results of classifying images from the Yoga-82 dataset with various CNN models.

The Yoga-82 dataset contains 82 third-level classes of yoga poses grouped into 20 second-level classes that are further merged into 6 first-level classes. The poses are grouped according to the posture and pose look. Of course, not all poses can be easily assigned to one of the 82 third-level classes, some variation has to be taken into account. The hierarchy of classes can be used to improve the classification or to estimate a pose type with higher accuracy – reported first, second, and third-level accuracy is 89.81 %, 84.59 %, and 79.08 %, respectively, for top-1 accuracy on DenseNet-201.

Yoga as a sport includes an extensive amount of poses with a variety that stands out amongst other sports. The poses can be also sorted into groups and thus create a hierarchy that can be further used for classification as can be seen in [20].

2.4 Self-Supervised Learning for Computer Vision

Self-supervised learning is a method of training a model first to learn data representations on unannotated data and then to use annotated data to train another model for classification of the representations. The first model learns patterns in the data and how to represent the needed information in it with a latent vector. There is no need for any labels since the data itself is used for supervision. Means of obtaining the supervision differ upon the task

Repeated	Layer	Settings
1×	Input	Size: (224, 224,3)
1×	Conv2D	Filter count: 64 Kernel size: (7, 7) Stride size: (2, 2)
6×	Conv2D	Filter count: 64 Kernel size: (3, 3) Stride size: (1, 1)
1×	Conv2D	Filter count: 128 Kernel size: (3, 3) Stride size: (2, 2)
7×	Conv2D	Filter count: 128 Kernel size: (3, 3) Stride size: (1, 1)
1×	Conv2D	Filter count: 256 Kernel size: (3, 3) Stride size: (2, 2)
11×	Conv2D	Filter count: 256 Kernel size: (3, 3) Stride size: (1, 1)
1×	Conv2D	Filter count: 512 Kernel size: (3, 3) Stride size: (2, 2)
5×	Conv2D	Filter count: 512 Kernel size: (3, 3) Stride size: (1, 1)
1×	Dense	Units: 1000 Activation: Softmax

Table 2.1: ResNet-34 architecture with only the main layers mentioned together with their settings. Some layers are repeatedly used after each other, the number of repetitions is in the first column. Residual connections are over every 2 convolutional layers except the first one.

and data available. The second model needs to use an annotated dataset to assign classes to latent vector output by the first model. Classification is made easier with the data being represented efficiently and there is no need for large annotated datasets to achieve good generalization properties of the model.

There are various ways to use the image data itself as supervision. For instance, it is possible to use a small distortion on the original data and expect it to not change its meaning. With this, different images that are bound together are created automatically. Self-supervision can also be used for colorization tasks, the original color images are easily converted into grayscale images and the model's goal is to colorize it to match the original sample. Another common challenge is generating missing image data, which is done with context encoders. Some part of the image is cropped out and the encoder is trying to fill it in to its previous form.

This thesis uses models called time-contrastive networks (TCNs) introduced in [15]. Self-supervision is achieved by using multiple cameras to film a scene from different viewpoints. After the videos are synchronized, frames with the same timestamp but from different cameras should still produce latent vectors fairly close to each other. When the timestamps are different (and the scene changed), latent vectors should be further from each other even when filmed from an identical viewpoint.

Architecture	Depth	Parameters	Top-1 Accuracy	Top-5 Accuracy
ResNet-101	101	42.72 M	65.84	84.21
DenseNet-169	169	12.60 M	74.73	**91.44**
DenseNet-201	201	18.25 M	**74.91**	91.30
MobileNet-V2	88	2.33 M	71.11	88.50
ResNeXt-50	50	23.15 M	68.45	86.42

Table 2.2: Performance of widely-known CNN architectures on Yoga-82 dataset using third-level classes from [20].

Self-supervised models use different loss functions that suit their specific approach to solving the problem. As described in the previous paragraphs, self-supervised learning has many different forms and, therefore, architectures vary in many ways. The time-contrastive network has to learn to represent the object in the image independently of the viewpoint. Images from the same viewpoint can differ just a little in time but their image embeddings should be different if the observed object changed. Whereas images from different viewpoints at the same time can be entirely different, only the observed object is constant. Therefore, their embeddings should be reasonably similar. Such a challenge can be solved with a loss function called the triplet loss.

2.4.1 Triplet Loss Function

The triplet loss function pushes embeddings of similar data closer together and pulls embeddings of diverse data further apart. Its main goal is learning data representation in a d-dimensional Euclidean space. Inputs of the function are 3 data embeddings: anchor, positive, and negative. Anchor and positive should be closer to each other than anchor and negative. It can be thought of as anchor and positive belonging to the same class and negative to a different one [14].

Data point x has an embedding $f(x) \in \mathbb{R}^d$ which is additionally constrained to live on a unit hypersphere – $||f(x)||_2 = 1$. Triplet i consists of anchor image x_i^a, positive image x_i^p and negative image x_i^n. The goal of the network is for inequality 2.13 to held true with condition 2.14.

$$||f(x_i^a) - f(x_i^p)||_2^2 + \alpha < ||f(x_i^a) - f(x_i^n)||_2^2 \tag{2.13}$$

$$\forall (f(x_i^a), f(x_i^p), f(x_i^n)) \in \mathcal{T} \tag{2.14}$$

Where α is a margin that is enforced between positive and negative pairs and $\mathcal{T}$ is the set of all possible triplets, $|\mathcal{T}| = N$.

The triplet loss to be minimized is then:

$$L = \sum_i^N \left[||f(x_i^a) - f(x_i^p)||_2^2 - ||f(x_i^a) - f(x_i^n)||_2^2 + \alpha \right]_+ . \tag{2.15}$$

And more often it is used as:

$$L = \sum_i^N \max\left(0, ||f(x_i^a) - f(x_i^p)||_2^2 - ||f(x_i^a) - f(x_i^n)||_2^2 + \alpha\right). \tag{2.16}$$

The triplet selection is an important part of the whole training process. When the constraint 2.14 is easily met, the triplet has not improved the model at all and therefore, it will converge slower. The ideal triplets (hard positive and hard negative, respectively) are satisfying these two conditions [14]:

$$x_{hard}^{p} = \underset{x_i^p}{\operatorname{argmax}} \, ||f(x_i^a) - f(x_i^p)||_2^2, \tag{2.17}$$

$$x_{hard}^{n} = \underset{x_i^n}{\operatorname{argmin}} \, ||f(x_i^a) - f(x_i^n)||_2^2. \tag{2.18}$$

In this thesis, with the time-contrastive learning method, hard triplets cannot be computed in any way. Hard positive pair is enforced by making the viewpoints of the images as different as possible. Since anchor and positive images are taken from different viewpoints, the background of the scene and light conditions can vary. Also, just by filming the scene from different angles, the object can look completely different. The hard negative pair condition can be fulfilled by choosing an image from the same viewpoint as for the anchor image but with a slightly different timestamp. How much should the timestamp differ, depends on the scene itself. If it is very dynamic, the images can be just a few frames apart, if it stays the same for a longer time, a different approach has to be chosen. One possibility to choose a hard negative pair from a video is by computing optical flow from the video and using it to detect movement. This procedure is thoroughly discussed in Section 3.1.3.

2.4.2 Other Metric Learning Loss Functions

Triplet loss is not the only possible loss function that can be used to construct an embedding space with a neural network. There are several other functions for metric learning that can be applied to the same task that is being solved in this thesis. These alternative functions are proposed as a possibility for future work on the project and are not implemented.

Lifted Structure Loss is a good candidate for an alternative to the triplet loss function [18]. It uses a similar format to the triplet loss's training data: anchor, positive, and negative samples. The difference is that it utilizes multiple negative samples at once and thus provides faster convergence. It is fairly easy to provide a higher number of negative samples since frames before and after the timestamp of anchor-positive pairs from all viewpoints are candidates for negatives.

Multi-Class N-Pair Loss is very similar to lifted structure loss in the sense that it uses multiple negative samples but it differs in what it tries to optimize [17]. It computes cosine similarity between features of the data points and tends to be scale-invariant.

While triplet and lifted structure losses both use relative distance as a metric, angular loss accounts for the angle at the negative edge of the triplet triangle [21]. It drags negative data points away from the anchor-positive pair. The pair is on the other hand pushed closer together. This metric also benefits from scale invariance. The advantage over triplet loss is an easier setting of margin as a hyperparameter. The margin for triplet loss depends on the intra-class variance of data while the margin angle for angular loss is invariant of such property.

Chapter 3

Obtaining Datasets for Self-Supervised Learning

Datasets for self-supervised learning, in general, can be very different from each other, their form depends on the task that is being solved and the way of achieving the self-supervision. Although, all of them have one thing in common – they mainly consist of unannotated data and in the end, they need a smaller amount of annotated data to be able to classify from learned embeddings.

As is mentioned in Section 2.4, this thesis focuses on Time-Contrastive Learning (TCL), which means that it achieves supervision with multiple viewpoints of the same scene concurrently as is displayed in Figure 3.1. While the filmed object may look very different when filmed from different angles, it is still the same object if the timestamps are identical. It is also possible to use moving cameras for the filming of the scene, although this might introduce some inaccuracy to movement detection. On the other hand, even when the viewpoint is equivalent, the object can be altered only after a short time has passed. This characteristic holds supervision when TCL is used and because no extra work (e.g. labeling) has to be done, the data is supervised by itself – self-supervised. The only restriction is the need to have multiple videos of the same scene synchronized in time.

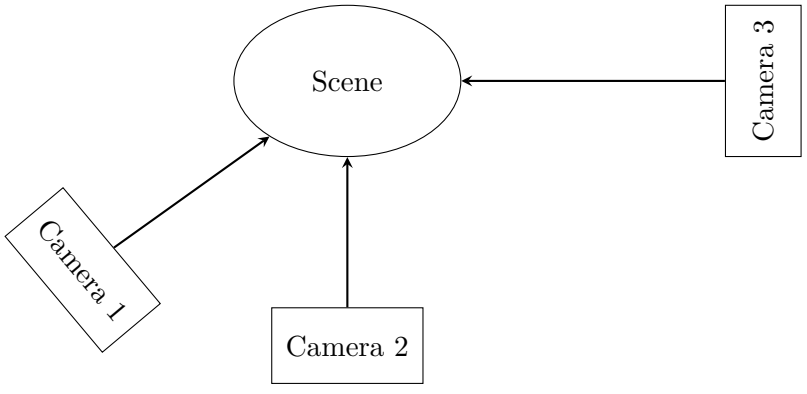


Figure 3.1: Example of a scene filmed from different viewpoints with 3 cameras. The higher number of viewpoints ensures greater variability in the dataset for time-contrastive learning.

It is necessary to have the process of dataset creation as automated as possible. Otherwise it would have been easier to just label the data and simply use a supervised learning approach. Therefore, I propose a set of tools in Section 3.1 that creates a dataset ready for

TCL with just a small amount of user interaction. After that, a simple tool for labeling images is presented in Section 3.1.5, because at least a small number of images with an assigned class is always necessary.

At the end of this chapter, in Section 3.2, a basic dataset of sports poses is presented. It contains scenes with a solid background and sports pose with variance only in arm movement and it was captured and prepared especially for this thesis. The basic sports pose dataset was demonstratively prepared only with the tools described below. Finally, possible directions of the development of the dataset with advanced sports poses are discussed.

3.1 Creating Dataset for Time-Contrastive Learning

Dataset for Time-Contrastive Learning (TCL) is created from synchronized videos of the same scene filmed from different angles. I propose a tool for semi-automatic preparation of such a dataset, illustrated in Figure 3.2. It offers a few simple editing features such as cropping and trimming. A very important feature it provides is the automatic synchronization of multiple videos. The second necessary component is a movement detector that estimates how much movement happened between frames of the video. This information is essential to achieve the time contrast that TCL relies on. Finally, the tool exports chosen video frames with their timestamps to simplify the creation of triplets for model training.

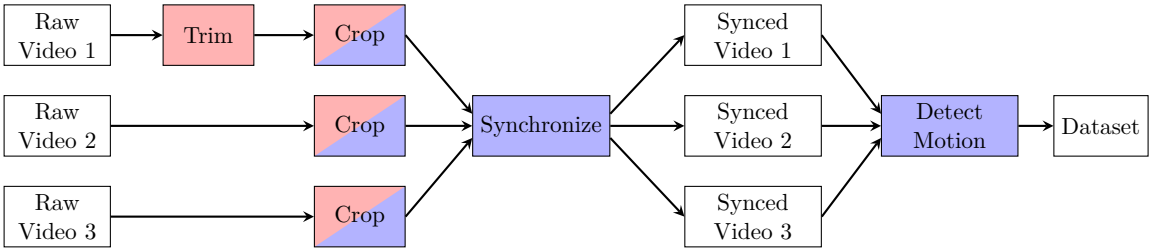


Figure 3.2: Tools for construction of dataset presented in the order of their usage. White boxes represent data and colored boxes the tools. The red color symbolizes tools that need some user interaction whereas blue-colored tools are fully automatic. The tool for cropping needs a user to select a specific area but then it automatically adjusts the selection to fit all needs.

Cross-platform video conversion solution `FFmpeg` is used to handle all video modifications effectively. Videos can be either processed directly with `FFmpeg` or a script is created that does the identical operations but can be launched later. The individual parts of the editing tool are presented in the following subsections in the order of their execution.

3.1.1 Preparing Videos Filmed with Various Cameras

The first step in dataset preparation is to trim the start and the end of the video. It is almost certain that the video contains a little bit of inapplicable footage at the beginning and at the end. Therefore, a simple tool that allows users to select the trim range with sliders is developed for the purpose of this thesis. Because of how the synchronization tool works, a user only needs to trim one of the videos, the others will be trimmed automatically when being synchronized. This is further described in Section 3.1.2.

In most cases, the video's resolution does not match the input of the model and has to be scaled down, and is often also cropped to the correct ratio. The tool allows users to

select a bounding box around the scene which will always be included in the cropped video and non-important parts of the scene will mostly be deleted. Correct crop coordinates are automatically computed to match the input of the network and all other constraints. The computation consists of operations shown in Figure 3.3 and described in detail in the following enumerated list, their order is important.

1. A view of the video as a single image is constructed from 10 merged frames taken out of the whole video in order to provide the user with enough information about the range of motion.
2. User selects the part of the scene that has to be included in the cropped video with a bounding box, these are the initial crop coordinates.
3. Crop coordinates are adjusted to match the $height \times width$ ratio of the model input.
4. If crop selection has a lower resolution than the network input, the selection is equally extended.
5. If crop selection exceeds the frame size, it is decreased to the closest possible value.
6. If crop selection is positioned out of the frame, it is moved to the closest correct position.
7. Video is cropped to the computed crop selection.

Crop coordinates are correctly computed to match the model input $height \times width$ ratio but the resolution will most likely not match. Therefore, the video has to be scaled down or (in the case of a video having lower resolution than model input) scaled up. Lastly, the framerate of all of the videos has to be unified to a previously chosen fixed value to ensure the correct run of synchronization and motion detection algorithms.

3.1.2 Synchronizing Videos by using Dense Optical Flow

The main requirement for TCL to work is the synchronization of all used videos. It is very likely that not all used videos are perfectly synchronized and manual synchronization would not be very precise nor effortless. Therefore, I present an automatic tool that determines the correct synchronization and trims all videos at the beginning and at the end so that all are the same length and synchronized.

The synchronization is done by optical flow, which is information about the movement of each pixel between video frames [8]. Dense version of optical flow from `OpenCV` library is used [1]. Visualization of dense optical flow is shown in Figure 3.4. The assumption behind using dense optical flow for synchronization is that videos of the same scene have a correlative amount of movement in similar directions at the same time. The precise movement of each pixel cannot be easily computed, it is only possible to approximate it. However, the precise values are not necessary for synchronization purposes, rough values are accurate enough.

Movement vector of each pixel is from two reasons too specific for this task. First reason is that each video displays the scene from different angle and their optical flows will most likely be different. It is more useful to have general information about movement in the whole frame than to have it pixel-wise to eliminate small discrepancies. The second reason to aggregate information over the whole frame is growing computational complexity. If the

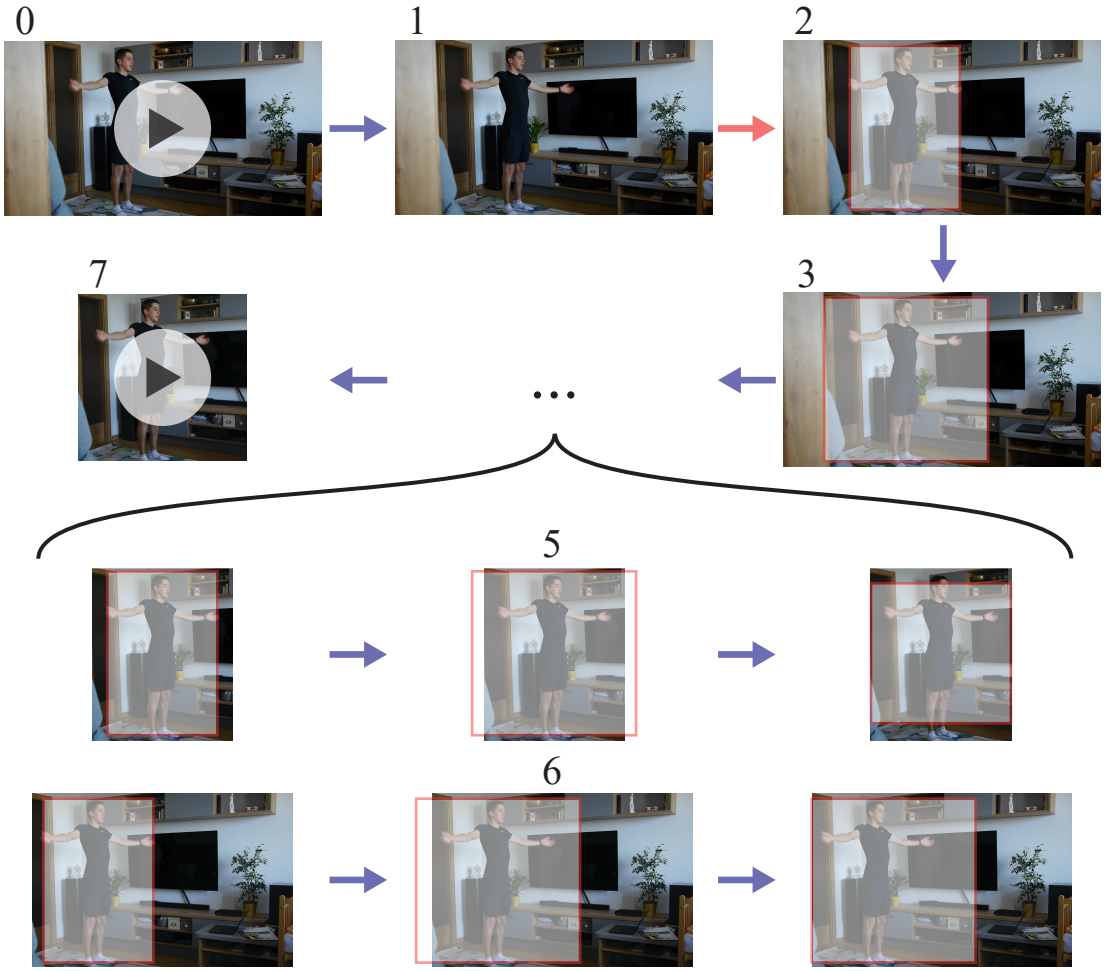


Figure 3.3: Process of cropping a video with the semi-automatic tool. The user selects an area of interest with a bounding box and the tool performs cropping and resizing to a given resolution. The numbering of individual steps refers to the previously mentioned description of the tool. Blue arrows symbolize automatic steps and a red arrow manual steps. Steps 5 and 6 are only needed for some specific cases displayed in the figure, step 4 is not shown.

information is accumulated over all pixels into a fixed number of values, the computational complexity stays constant, whereas it grows when pixel motion values are used individually.

Since all pixels can move in two dimensions, it would make sense to gather information about the horizontal and vertical movement by simply summing up all the values. The problem with this approach is that when some pixels move to the right and some to the left, their movement vectors subtract from each other in that dimension and a lot of information is lost. For that reason, I propose to sum separately positive and negative values in each dimension and obtain information about the amount of motion in 4 directions: up, down, left, and right. Each frame (except the first one) of each video is assigned these 4 values describing the optical flow from the previous frame to the current one.

After that, Pearson correlation of the aggregated optical flows of video pairs has to be done. These are not computed for all pair combinations, all flows are only compared to the shortest one to perform a smaller number of computations but still guarantee to get

(a) (b) (c) (d)

Figure 3.4: Dense Optical Flow visualized on multiple frames from the same video where a person is moving his arms. Visualization 3.4a shows both arms moving up while 3.4b captures both arms moving down. As can be deduced, visualization 3.4c is a combination of the previous two: one arm is moving up and the other one down. Graphics 3.4d displays only forearms moving closer to each other while the elbows stay in place. All other parts of the frames stay steady.

the best possible synchronization. Each flow pair is compared to get an overlap with the highest correlation, which means the correlation for each possible overlap is computed. The only restriction is that the overlap has to be at least a certain number of frames long to eliminate corner cases where for example only one frame has the best correlation (the first frame from one video and the last one from another). This minimal overlap length can be a little lower than what the expected length of synchronized videos is to optimize for the lowest number of correlations that has to be done. By default, it is set to 1,000 frames.

Each overlap is assigned a specific index that will be used to describe it in the following text. The problem is that the overlap index with the highest correlation might not be the correct one that synchronizes the videos because of some inaccuracy. To eliminate this problem, one additional property can be used – if the correlations are computed accurately, the highest ones will be of overlap indices from a similar range, just a few frames apart. To use this property, overlap indices are sorted descending by correlation, and the top 10 are taken. The standard deviation of these samples is computed and if it exceeds 10, it means that there is at least one overlap index among them that does not fit the majority. All these indices are checked and if they are further than one standard deviation from the mean, they are removed from this list. This operation is repeated until the standard deviation of the whole list is lower than 10. After that, the index with the highest correlation that is still on the list is the best one for synchronization. Real data example of this algorithm is provided in Table 3.1.

When the best overlap indices for the flow pairs are computed, flows are shortened to the same length where all of them overlapped. Their respective videos are trimmed at

State	Best overlap indices (descending by correlation)	Standard deviation (SD)	Allowed range
Initial	2882, 1410, 1543, 1692, 1691, 1693, 1690, 1694, 1388, 1695	$398.7 > 10$	$(1339, 2137)$
1st epoch	1410, 1543, 1692, 1691, 1693, 1690, 1694, 1388, 1695	$122.3 > 10$	$(1488, 1733)$
2nd epoch	1543, 1692, 1691, 1693, 1690, 1694, 1695	$52.3 > 10$	$(1618, 1724)$
3rd epoch	1692, 1691, 1693, 1690, 1694, 1695	$1.7 \not> 10$	Any – SD condition satisfied

Table 3.1: Example of real data from the algorithm that tries to find the correct overlap index of two flows to have them correlated as much as possible. The algorithm needed 3 epochs to remove indices that had no other surrounding ones and, therefore, were detected as false findings. These indices are out of the allowed range, while the other were close to each other and in the allowed range. Index 1692 had the best correlation and was chosen as the best overlap index.

the beginning and at the end to have the same length as well and are thus synchronized because of the correct trim times.

3.1.3 Detecting Movement with Sparse Optical Flow

After videos of a scene are synchronized, the last step in creating the dataset is choosing the correct frames to be used for training a neural network model. These frames have to satisfy one property – there has to be enough movement between them for a model to be able to recognize the difference. The less movement is between the frames, the harder it will be for the model to learn embeddings of the filmed object but also the more precise the embeddings might be. It is necessary to set the movement threshold to a correct value since the dataset quality is crucial for the model's performance.

The first chosen approach was taking every k-th frame where k was manually set by the amount of movement in the video. Since the amount of movement varies in different timestamps of each video, this approach did not produce good enough frames for training. The next chosen tactic was using Dense Optical Flow for motion detection but this algorithm tracks every pixel in the video and the necessary information about the movement of the followed object is lost in the amount of unnecessary data.

Therefore, I decided to use Sparse Optical Flow to fulfill the movement detection task. Unlike the Dense Optical Flow, the sparse variant chooses only some pixels with the Shi-Tomasi corner detector and those are being tracked [16]. This is a specific implementation of the sparse optical flow from `OpenCV` library [1]. The amount of movement has to be ideally aggregated into a single number and summed frame after frame until it exceeds a certain threshold. Then, enough motion has been detected and the given frame is selected and motion detection continues again from zero to detect another frame. This procedure dynamically chooses the gap between chosen frames, which is essential for sports pose recognition. It is possible that at some times, a few seconds of no movement are followed by a lot of motion in just one second.

The challenging task is aggregating the information from sparse optical flow to a valuable metric. Since each scene can be different, the number of tracked pixels can also vary. Therefore, the distances cannot be easily summed but rather averaged. Another problem is introduced when the background of the scene is not solid and some of the tracked pixels are in the background. Those do not move and they should not influence the average distance of pixel movement. This is solved by calculating the average of only those pixels, whose movement is above a certain threshold.

One more problem that was encountered during the development of the motion detection algorithm was the vanishing of monitored pixels. After a higher amount of movement, the pixels detected to be followed might get lost and there are not enough pixels left to precisely detect motion. When this happens, the Shi-Tomasi corner detector has to be run again to detect new pixels for the sparse optical flow.

The last feature that the motion detector uses to provide more accurate results is accounting only for unique moves. If all pixels move the same way, that means the scene has changed but the sports pose probably did not change at all. This problem also has to be addressed. The motion detector does so by computing a cosine similarity between all motion vectors produced by sparse optical flow and ignores those vectors that are too similar.

Detected frames from the video are saved as images and will be used for self-supervised learning of a neural network. The motion detection has to be run prior to the learning procedure to enable for shuffling of training data and also to make the loading of the dataset less computationally demanding.

3.1.4 Building Triplets from Video Frames

The neural network used in this thesis is trained with triplet loss function and, therefore, it has to be provided with 2 data samples of the same pose from a different viewpoint and 1 sample of a different pose from the same viewpoint as one of the previous two. The goal is to provide the network with batches of such triplets.

At first, file paths to the correct images are formed into triplets and then into batches. After that, file paths are replaced with images that they were pointing at. Dataset of batches of triplets is then shuffled and split into training and validation subsets. Before each epoch, the training subset is always shuffled again to provide for higher variability.

3.1.5 Tool for Labeling Sports Poses in Dataset

Even though the main advantage of self-supervision is that very few annotated training samples are needed, there is still a need for some of them. That is why I decided to also develop a tool for very fast and easy labeling of training samples. This labeling tool takes a directory with unsorted images as input and moves them to their respective directories named after their labels.

If launched for the first time, new classes have to be assigned to specific keyboard keys and then with just a single press of the key, the displayed image is assigned to its class. This procedure makes image labeling as minimalistic as possible. Key-class pairs are saved as a dictionary to a file that can be loaded at any time to continue annotating of images.

3.2 Dataset of Sports Pose Images

Multiple sets of videos were recorded for developing tools for a dataset suitable for time-contrastive learning and then for the sports pose recognition itself. At first, the goal was to obtain a dataset consisting of a simple scene that presents an easy challenge for both the dataset preparing tools and for recognition models. Afterward, a more difficult task can be presented to the dataset tools and encoding and classifying models, e.g. a dataset with sports poses recorded in multiple environments. Sports poses from yoga are proposed for future development as one of the most challenging tasks possible in this field.

3.2.1 Hand Poses as a Simple Testing Data

One hand in different poses is a real-world situation with pretty low variability, especially if the background is solid. Therefore, it is a good candidate for testing data for the dataset preparation tools and can be used for the testing of a self-supervised trained model during its development. The original videos are cropped so that only the forearm and hand with fingers are in the frame to make them as elementary as possible. This dataset will be referred to as the Hand Dataset. Examples of its images are shown in Figure 3.5.

Figure 3.5: Hand Dataset images from different scenes. Some images have a fairly solid background while the others have a very heterogenous one to simulate various possible situations for the dataset preparation tools.

Trimming and cropping of the video are done completely manually and basically are not dependent on the dataset, the first real challenge comes with the automatic synchronization of videos from multiple viewpoints. The task was easily fulfilled on videos with a solid background and not so different viewpoints but once these two conditions were disrupted, an incorrect synchronization could be found. Therefore, an algorithm that searched for the top 10 best synchronization timestamps, not only the best one, was developed. This algorithm is fully described in the previous section.

The following challenge was to obtain specific frames from the video used for training. Videos without a solid background showed the importance of only taking into account the moving pixels because the Shi-Tomasi corner detector used in Sparse Optical Flow

computation selects also pixels from the background, not only those related to the followed object. Another task that this dataset exposed was the vanishing of the followed pixels. The last and most difficult challenge to solve was the detection of translation where the pose actually does not change. Movement without any pose adjustment is a phenomenon normally present in this dataset. Another instance of the same problem is when the camera is moving and the pose stays in the same position.

3.2.2 Sports Poses with Upper Body Movement

The Hand Dataset served its purpose in the making of the dataset preparation tools and the next task is the development of the self-supervised model. I recorded and prepared a dataset of simplified sports poses that are less complicated than what would for example yoga poses look like but still have the character of sports poses. An example of poses can be seen in Figure 3.6. All of the recorded poses hold these conditions. The person is recorded from knees up and moving only his arms, while any bending of shoulders, elbows, wrists, and fingers is allowed (and advised).

The dataset videos include one person in two scenes with different backgrounds. There are 5 recordings of each scene with the person wearing various clothing in each of them to increase variability in the dataset. That means 10 recordings in total. Performed poses are chosen randomly. Each scene was filmed with 3 cameras with different lenses. The camera angles were chosen so that one is facing straight from the front side and one is on each side at approximately 45 degrees angle from the front one. The total number of images in the dataset is 3,804 but each of those is one of 1,268 potentially different poses captured with 3 cameras. Distribution of the poses count across all ten recordings is following: 34, 49, 67, 99, 96, 51, 204, 128, 295, 245.

3.2.3 Sorting Upper Body Dataset into Classes

The recorded dataset of arm movement is prepared for self-supervised training of an encoder model but for real recognition, some amount of labeled samples is also necessary. There are many different ways to sort all the poses into classes. I decided to assign two sets of labels to all the samples to have enough data for experiments, one with a lower number of classes that presents an easier recognition task and the other with a higher number of classes to demonstrate the model's performance.

At first, I divided the data into 4 classes to create an easier task. Since arms are the only moving entities in the image, only those are taken into account and the rest of the body is ignored. Each arm can be either in an upward or downward direction but when various elbow bendings are taken into account, resolving whether the arm is pointing up or down is ambiguous. Therefore, a strict criterion has to be set. I decided to use the height of the wrist and shoulder to be the determining factor. If a wrist is above shoulder height, the arm is in the upward position. Finally, each arm is dealt with separately and that means 4 classes emerge: both arms down (down-down – 1,146 samples), left arm down and right arm up (down-up – 813 samples), left arm up and right arm down (up-down – 795 samples) and both arms up (up-up – 1,050 samples). Dataset with basic sports poses divided into these 4 classes will be from now on referred to as Directions Dataset.

For a more complex and convincing evaluation of developed models, I prepared one more set of labels with finer separation. The main thought is the same as for the Directions Dataset – each arm is either pointing up or down, but one more feature was added. Each arm can be either bent in the elbow or not – if the elbow angle is smaller than 135 degrees,

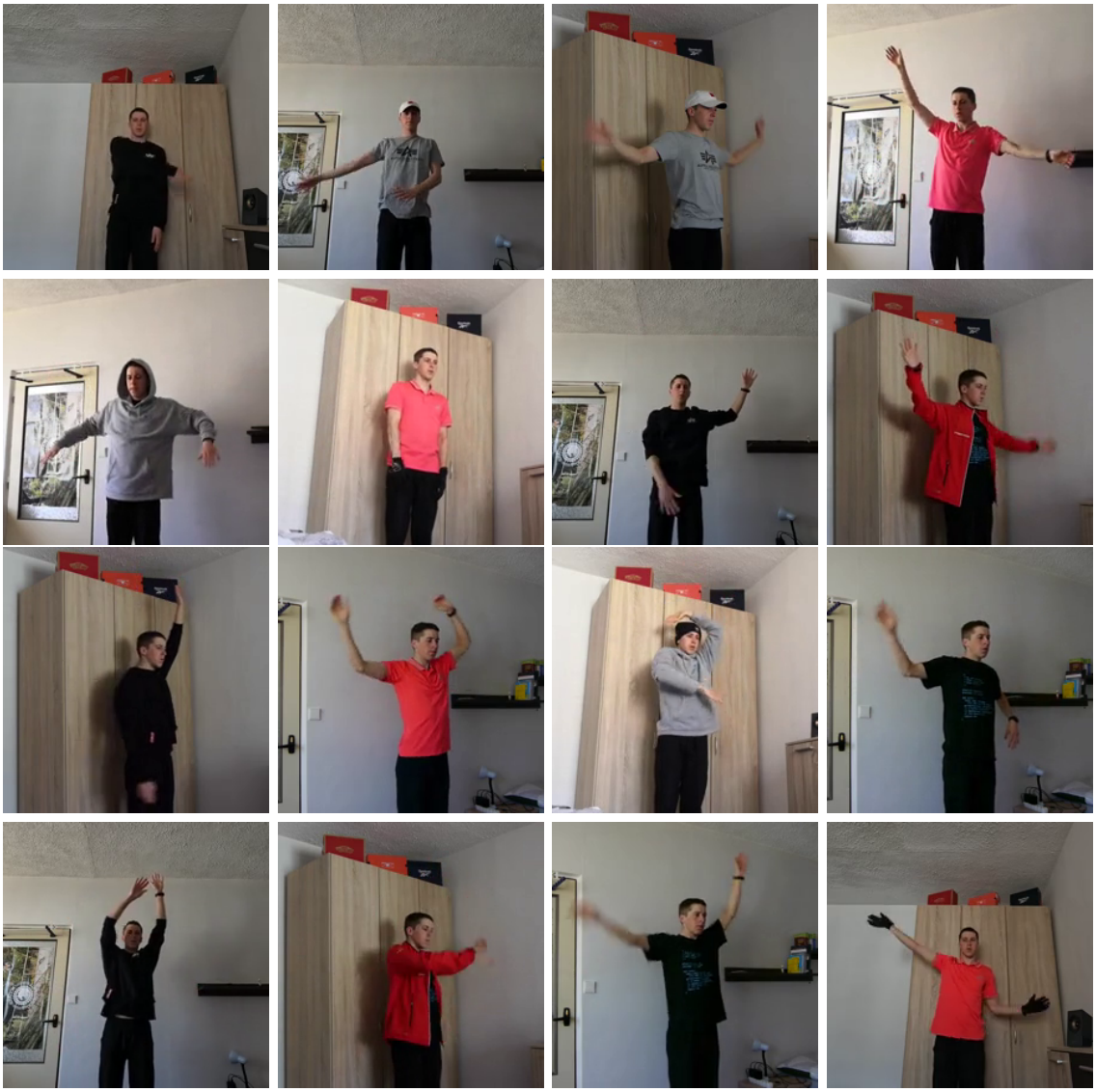


Figure 3.6: Samples from the dataset with upper body poses. Recordings come from 2 places, each with 5 videos with different clothing. The scenes are recorded with 3 cameras from different viewpoints.

the arm is considered bent. All of the Directions Dataset classes have 2 possible options which create 16 classes in total. A higher number of classes not only allows for more valuable testing of the model but also brings the option of evaluating not only top-1 accuracy but also top-3 accuracy. When the correct class of a certain pose is not the one with the highest probability according to the classifier but still is the second or third, the model shows some ability to recognize poses too. Counts of samples for each class of the Bent Dataset are presented in Table 3.2. Class names are derived from the Directions Dataset, the only change is that when an arm in the given direction is bent, the letter 'b' is placed in front of the direction.

Both of the presented datasets cannot be perfectly divided into classes without any discrepancy. Followed features are in some cases right in between the available classes. The arm can be almost perfectly horizontal with its wrist at the same height as the shoulder.

Class	Sample count	Class	Sample count	Class	Sample count	Class	Sample count
down-down	1,005	bdown-down	63	down-bdown	48	bdown-bdown	63
down-up	399	bdown-up	21	down-bup	285	bdown-bup	102
up-down	399	bup-down	282	up-bdown	33	bup-bdown	90
up-up	495	bup-up	135	up-bup	99	bup-bup	285

Table 3.2: Distribution of sports pose images between all classes of the Bent Dataset. The first word shows the direction that the left arm is heading, and the second word represents the right arm. If the arm is bent, the letter 'b' precedes the direction.

Correspondingly, the elbow angle can be precisely 45 degrees or very close to it and it is not possible to distinguish this difference from a single image. Therefore, some error rate is almost inevitable and has to be taken into account during the model evaluation.

Directions Dataset has fairly equally distributed images between classes whereas Bent Dataset has significant disproportions in the counts. This presents another challenge for the classifier that is trying to learn from the data. In general, self-supervised models should have better performance on such data because they first learn the data embeddings without any labels and then solve the fairly simple task of classifying those. In contrast to a supervised classifier that is solving the difficult task directly on the labeled data and might not update its weights enough because other classes in the batch are more significant.

3.2.4 Yoga Sports Poses for Future Development

Yoga sports poses are one the most complex among all sports poses, there are hundreds of possible poses and their variants. They can be also sorted into different sets according to their similarities. All these attributes make them the perfect candidate for a very difficult sports pose recognition task and, therefore, I propose yoga poses as a benchmark for future development in this field.

Verma et al. in [20] present a new dataset Yoga-82 for human pose classification that is based on yoga poses. It contains over 28 400 annotated samples of 82 different yoga poses. The classes are sorted into a 3-level hierarchy where each of the 82 poses is assigned a second and first-level class as well. This structure can be further used for the recognition of poses that are projected into an embedding space since embeddings of poses from the same higher-level class can have similarities in the space.

The dataset is available to download in form of URL links to each image sorted into files according to their classes. The images are under different creative commons licenses. There is no script for downloading of the images as a part of the publication. Since the images are from various sources on the internet, their availability is out of reach of the dataset authors. At the time of publishing this thesis, there are already hundreds of images not available.

The images differ in resolution and aspect ratio, therefore, some sort of preprocessing is necessary. Their variability is very high, they are captured both indoors and outdoors and with very different backgrounds. The displayed people differ in their gender, skin color, clothing, and other visible characteristics. Some images even contain multiple people, text added over the pose in postproduction or the images are just a simple illustration of the pose, not a real photo. These additional features might not serve well for better generalization of the trained model, but rather for confusion because of the unrealism.

These images could be used for the classification of the poses from embeddings but the more complicated task is learning the sports pose embeddings with a time-contrastive network. This process requires synchronized videos or at least images of the same scene from different viewpoints. I have not found such a dataset online and therefore, I suspect it has to be recorded specifically for this task. One possible solution for making the dataset creation more feasible would be to partner with some yoga video producers. There is a chance they are filming their videos from multiple angles and could offer their raw recordings for such a project. The variability of all recordings has to be taken into account. If they are from the same environment (e.g. indoor gym), the model will probably not generalize well to other surroundings (for example outdoors).

Chapter 4

Recognition of Sports Poses from Images

Recognition or classification is a task of assigning a class from a defined set of classes to an image according to what is displayed in it. When a video is on the input, it is often divided into single frames that are handled and classified individually. The most common approach to processing image data is with convolutional neural networks that are explained in depth in Section 2.2. Models in this thesis also use an architecture that relies on such networks and they were implemented in `TensorFlow 2` library [12].

Whereas supervised learning is done in most cases with a single model that has an image on the input and outputs probabilities of the image belonging to available classes. Self-supervised learning usually requires two models that are trained separately and are working together after they are fitted to the data. Therefore, the input and output of the self-supervised model are the same as for the supervised trained model once it is fitted. The first of two models that form the described architecture is usually called the encoder and its goal is to find the most valuable representation of the input in an embedding space. Implementation of this model is described in depth in Section 4.1. The embeddings produced by the encoder are used as an input to the second model, the classifier. Its objective is to find the most probable class the embedding is representing. A thorough description of the second model can be found in Section 4.2.

Another model for sports pose recognition but trained with supervision was also implemented to provide a comparison in evaluation. This network is introduced in Section 4.3 in contrast to the models proposed before. Finally, Section 4.4 discusses how the sport pose embeddings could be used in future research on this topic.

4.1 Representing Sports Poses in Latent Space

An encoder is the crucial part of a model that is trained with self-supervision. It uses some information that is naturally contained in the dataset as a supervisor during the learning process. In the case of this thesis, the supervision is provided with multiple synchronized videos of the same scene. Its target is to find the most efficient yet the most descriptive embedding of the input. If the goal is to recognize sports poses, the best embedding describes the whole body in the correct position but ignores all the specifics of the person and the environment around.

On the input of the model is an image with a specific resolution and channels in the correct format. On the output is an embedding vector describing the input image in the set dimensionality. Section 4.1.1 specifies the model's architecture into detail.

4.1.1 Architecture of the Encoder

The input is always a single image that needs no preprocessing because all the necessary operations were already done with the dataset preparation tool from the previous chapter. In case of a smaller dataset size, to gain more generalization, data augmentation is also implemented. It is not recommended to use any rotation or horizontal/vertical flipping augmentation because of the model's dependence on positions of body parts and distinguishing between left and right-hand sides. Augmentations that alter colors, brightness, and contrast are a favorable option used in this thesis.

To obtain embeddings of the images, a convolutional neural network is used. This thesis uses a ResNet-50 architecture from [6] with weights trained on ImageNet dataset [3]. The head of the network is replaced to provide embeddings as vectors in d-dimensional latent space. This is done with a single dense layer after the data from the last convolution are processed by average pooling and flattening layers. The number of units of the dense layer and the dimensionality of the embedding space is equal.

The embedding vectors are sometimes restricted by the condition to sit on a unit hypersphere. That means that squared values in all dimensions of the vector have to sum up to 1. This is done to provide normalization of the individual values in all dimensions. This restriction can be fulfilled with L2 normalization used as the last layer after the previously mentioned dense layer. This is the output layer of the whole model.

The model was trained with Adam optimizer. Underlying concepts and the calculation of Adam are presented in Section 2.1. Parameters were configured to typical values: learning rate $= 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-7}$.

The network is trained on triplet loss in the self-supervised manner [14]. This loss function is described in Section 2.4.1 and Section 2.4.2 proposes possible improvements in this direction that were not implemented. Since the triplet loss uses Euclidean distance to compare embeddings, its effective calculation is crucial to the good performance of the model. The function is implemented to compute loss over the whole batch of triplets. It uses simple subtraction and squaring in each dimension and then a sum to reduce all the dimensions into a single number. The loss is only influenced by the triplets that do not have the positive sample closer to the anchor than the negative sample by a set margin. Value of the loss is a sum of their differences in positive-anchor and negative-anchor distances. When computing accuracy, the margin is not taken into account.

The training of the model can be divided into two parts: fitting and fine-tuning. When the encoder is fitted, only the head of the network, and weights of the last dense layer, are adjusted. The ResNet-50 backbone has its weights locked to the ImageNet-pre-trained values. After that, a fine-tuning process can be turned on as well. Fine-tuning starts with unfreezing all the weights of the backbone except the ones used for batch normalization. Then, a learning rate is changed from 10^{-3} to 10^{-5} to prevent large changes and possible loss of information already acquired from fitting and pre-training. After that, the model's weights from the epoch that provided the best results on the validation dataset during fitting are restored and fine-tuning is launched as a casual fitting. From the experiments done, it seems the encoder can provide very good results just with fitting and fine-tuning provides almost no improvement in the model's accuracy.

Fitting of the model is done in epochs with the dataset divided into mini-batches (further only as batches). Each batch contains a fixed number of anchor-positive-negative triplets, only the last batch of the epoch can be smaller. The batch size can be set according to the memory constraints of the training machine, in most cases between 32 and 256. Since the network is designed to only accept one image as an input, the triplets have to be merged together into a "merged„ batch. Its size is correspondingly 3× as large. After all the individual images are encoded into embeddings, they can be split into the original triplets again. The merging and splitting algorithms have to be deterministic and mutually reversed to ensure all triplets stay the same. Only after that, the loss of the whole batch can be computed. Finally, gradients are computed from the loss and applied to the network's weights.

The model reports loss and accuracy on training and validation datasets in a such format that can be further analyzed with TensorBoard. It also saves the model's weights after each epoch to allow for restoring the best-performing model. The implementation also allows for restoring weights and continuing fitting and with that divide the training process into multiple sessions.

4.2 Sports Pose Classification from Embeddings

After the sports poses are encoded into a d-dimensional embedding vectors, various operations can be done with them. This thesis only implements classification, the other possibilities are discussed in Section 4.4. The main advantage of a classifier that has vector embeddings on the input instead of images is that the important information is already extracted and, therefore, the classifying is a lot easier task.

The classifier in this thesis is a simple neural network with one hidden dense layer. The input layer has an identical size to the dimensionality of the embedding space and the output layer corresponds to the number of classes the sports pose can be classified to.

The size of the dense layer (number of units) is a hyperparameter that can be tuned according to the difficulty of the task that is being solved. Since all the needed information is already effectively encoded into the embedding, it is not advised to use a dense layer with more units than the input layer has. Likely, no other information will be gathered from the data and, therefore, there is no need to represent it with more values. After the experiments were done, one dense layer performed on par with networks with two or three hidden dense layers. Thus, I chose a single dense layer with 64 units for the classifier model implemented in this thesis. To introduce some non-linearity to the model, a Leaky ReLU with $\alpha = 0.01$ is used as an activation function of this layer.

The output of the classifier uses the softmax activation function to output the probabilities of each class that sum up to one. The training is optimized with Adam optimizer with parameters set to learning rate $= 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-7}$ and as a loss function is used categorical cross-entropy which corresponds to classifying tasks with more than two possible outcomes.

4.3 Classifier Trained with Supervision

To evaluate the effectiveness of self-supervision, a supervised-trained model is implemented as a comparison. The main condition is to make both models as identical as possible to not distort the experiments with model dissimilarities. The model is presented in comparison to the self-supervised model that was introduced earlier.

The main parts of both networks are completely the same, they both have ResNet-50 as a backbone. This means that the inputs of the networks are also identical. They only differ in the network heads – self-supervised model needs more dense layers to account for the embeddings. The architecture comparison can be seen in Table 4.1. The model's optimizer is Adam with the same parameters as for both of the self-supervised model: learning rate $= 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-7}$. The loss function is identical to the classifier from the self-supervised model – categorical cross-entropy.

Self-Supervised Model		Supervised Model	
Description	Layer – Shape	Layer – Shape	Description
Image	Input – (224, 224, 3)	Input – (224, 224, 3)	Image
Backbone ResNet50	Padding – (230, 230, 3) ⋮ Pooling – (2048)	Padding – (230, 230, 3) ⋮ Pooling – (2048)	Backbone ResNet50
	Dense – (64)	Dense – (4)	Label
Embedding	L2 Normalize – (64)		
	Dense – (64)		
Label	Dense – (4)		

Table 4.1: Comparison of architectures of self-supervised and supervised models. Their input, backbone, and output are identical, only the top of the self-supervised model is adjusted for the self-supervised training.

Although the models trained with supervision and self-supervision have almost the same architecture, the training process vastly differs. With a different approach to learning of the data structure, the number of parameters that have to be fitted is also different. Table 4.2 illustrates the contrast between them.

Model	Self-Supervised Encoder (fit)	Self-Supervised Encoder (fine-tune)	Self-Supervised Classifier	Supervised
Trained Parameters	131,136	23,665,728	4,420	23,542,788

Table 4.2: Different training procedures require a different number of model parameters to be trained. This table compares them. The self-supervised model encodes the input into a 64-dimensional embedding space and the number of classes on the output is 4, which also affects the parameter count. The self-supervised model mostly trains parameters of the encoder with fitting (fine-tuning did not bring significantly better results) and then parameters of the classifier. Their sum is the best comparison to the supervised model's number of parameters: 135,556 and 23,542,788.

In this chapter, only static data were shown. The performance of each model on the validation dataset is presented and discussed in the following Chapter 5. Both models perform very differently when only a lower number of training samples is introduced to them. The results of these experiments are shown in Section 5.3.

4.4 Additional Usage of Sports Pose Encodings

This thesis only discusses the classification of sports poses from their embedding vectors but this is not the only possible usage of such information. In this section, I propose several other possibilities for how the information could be processed.

While classification assigns a class to an embedding vector, other information could be assigned as well. A very common task in this field is the pose estimation, which can include several different information about a human pose such as joint position and orientation of different body parts. Obtaining this information just from an embedding could be very useful since labeling the pose estimation dataset is even more time-consuming than the labeling of a simple classification dataset.

There might be also a possibility to perform operations on embedding vectors such as addition or subtraction to obtain embeddings of poses that are not captured. This could be practically used to classify poses that are not even part of the training data and the model has not seen them or at least it could help lower even more the required number of training samples.

Generally, the implemented tools could also be used for another classification problem that includes an object that changes poses or a similar challenge. It could not only capture humans but also animals, robots, or machines.

Chapter 5

Evaluation of Models Trained with Self-Supervision

Models trained with self-supervision can be evaluated at two stages. Obviously, the accuracy of the classification of a given input is one way to do so. The other way is to evaluate embeddings either with loss function used for training of the encoder or visually after dimensionality reduction. All of these approaches are discussed in this chapter.

At first, the sports pose embeddings are visually analyzed in Section 5.1. Then, Section 5.2 presents the encoder's accuracy on the validation dataset and how it is affected by the number of dimensions of the embedding space. Finally, the accuracy of the classification itself on the validation dataset is compared to a supervised model with the same architecture in Section 5.3. The results of each evaluation are discussed in their respective sections. Experiments were done fairly and no results were cherry-picked.

5.1 Visual Analysis of Latent Space

The latent space has well over 3 dimensions and therefore cannot be easily visualized. Typically, embeddings of more complex information such as sports pose can range from 64 to 512 dimensions. Vectors representing them usually satisfy the constraint of living on a unit hypersphere. Analyzing data visually can help understand patterns in them and detect emerging problems. When the dimensionality is decreased to only 2 dimensions, a lot of information can be lost. Therefore, the challenging task for the projecting algorithm is to drop the non-necessary information and preserve the patterns in the data.

The elemental method for dimension reduction is Principal Component Analysis (PCA). It computes a new basis of the vector space to maximize the data variance. After projecting the data into the new basis, only 2 or 3 dimensions with the highest variance can be taken into account and the rest is ignored. Finally, such data can be plotted and reviewed. Another possible projection is Linear Discriminant Analysis (LDA) which also takes into account the label of each data point and is trying to find a basis that allows for the best linear separation of classes.

Whereas the previously mentioned algorithms allowed for computing the precise results, more complex methods for dimension reduction are based on iterative approaches to find the best approximation of the ideal state since it cannot be computed directly. A widely used algorithm for this task is t-distributed Stochastic Neighbor Embedding (t-SNE). It puts data points into pairs and tries to attract those that are similar and repel the dissim-

ilar ones. Another iterative method is Uniform Manifold Approximation and Projection (UMAP) which also non-linearly projects data into 2D or 3D. I chose to use t-SNE as the dimensionality reduction algorithm because it was able to find patterns in the data embeddings better than the other algorithms.

5.1.1 Dimensionality Reduction with t-distributed Stochastic Neighbor Embedding

t-distributed Stochastic Neighbor Embedding (t-SNE) is a non-linear dimensionality reduction method suited for displaying embedding vectors in two or three-dimensional space [10]. It is based on Stochastic Neighbor Embedding (SNE) but it uses also t-distribution instead of only Gaussian distribution [7]. t-distribution has heavier tails in comparison to Gaussian distribution and therefore, it solves one of the problems of SNE, which was centering the data points into one place in the low dimensions and not preserving the gaps between them.

The t-SNE algorithm starts with random initialization of projected data points in the targeted 2 or 3-dimensional space. It places them fairly close to each other to allow for patterns to emerge on a higher scale. Then two similarity distributions are constructed: one from points in the source high-dimensional space, the other from points in the destination low-dimensional space. Both distributions are constructed from distances between all pairs of data points in their respective spaces. Then, Kullback-Leibler divergence of joint distribution P in the high-dimensional space and Q in the low-dimensional space is minimized:

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{5.1}$$

Distances of data points from themselves p_{ii} and q_{ii} are set to zero. p_{ij} and p_{ji} are averaged in order to preserve symmetry $p_{ij} = p_{ji}$. Each data point pair is assigned a probability from Gaussian distribution with mean set to coordinates of point i and variance computed from the density of other points around it. The distance of point j from i is projected to the Gaussian distribution and p_{ij} equals the given probability, calculated as:

$$p_{ij} = \frac{e^{-||y_i - y_j||^2}}{\sum\limits_{k \neq l} e^{-||y_k - y_l||^2}}. \tag{5.2}$$

Probabilities q_{ij} are obtained from Student's t-distribution with one degree of freedom with a similar approach to the p_{ij}. The formula is as follows:

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum\limits_{k \neq l} (1 + ||y_i - y_j||^2)^{-1}}. \tag{5.3}$$

Finally, a gradient of the Kullback-Leibler divergence between P and Q is computed with:

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1}. \tag{5.4}$$

5.1.2 Analysis of Embeddings with t-distributed Stochastic Neighbor Embedding

The encoder presented in Section 4.1 transforms image of sports pose into 64-dimensional vector embedding. The goal is to have this embedding describe only the sports pose and ignore the background of the scene and the look of the person doing the pose. Sports poses similar to each other should be closer to each other in the embedding space than poses that are completely different. If only one arm moved from one image frame to another, their embeddings should be very similar. The same pose performed by another person in a different place and even photographed from a different angle should have the same or at least very similar embedding.

The whole Directions Dataset introduced in Section 3.2.3 includes 3,804 divided into 4 classes according to arm positions of the person – each arm is either pointing down or up and therefore, the corresponding classes are named: `down-down`, `down-up`, `up-down` or `up-up`. Positions, where both arms are in a downward direction, should be relatively far from each other while the other 2 classes can be placed somewhere in between the edge cases. The dataset consists of 10 scenes, each filmed from 3 angles. Projected embedding are displayed in Figure 5.1.

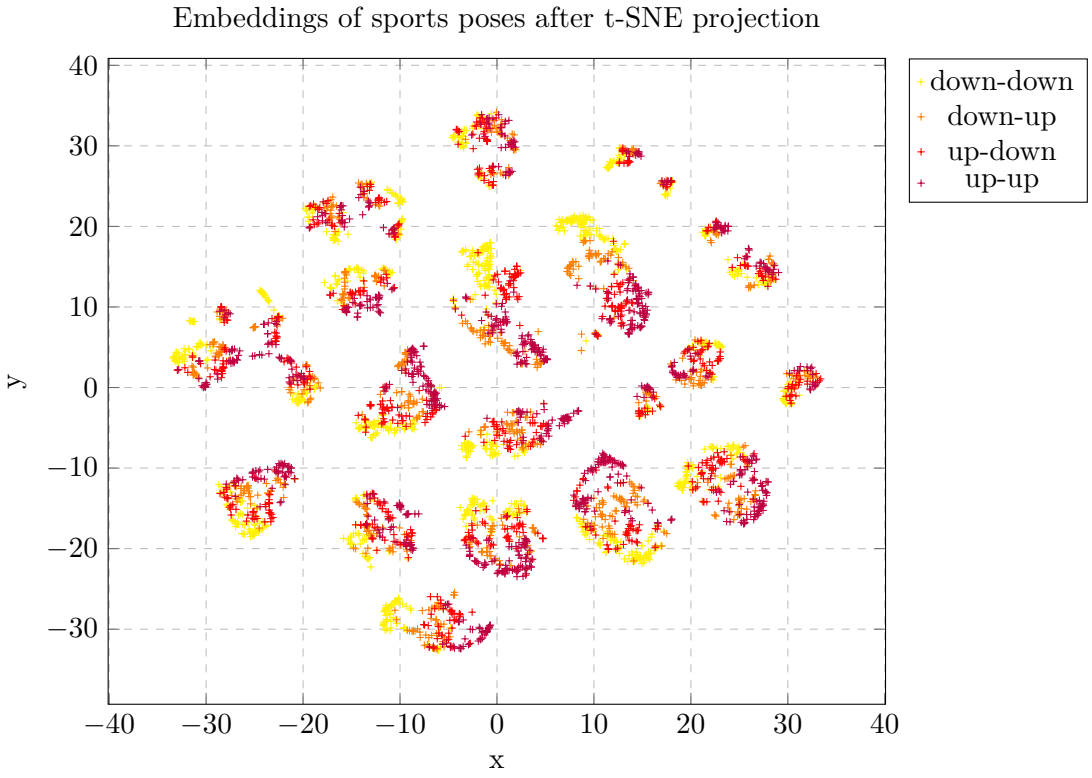


Figure 5.1: Embeddings of all 3,804 samples from the Directions Dataset projected from 64 dimensions to 2D with the t-SNE algorithm. Data points are colored according to their class. Parameters of the t-SNE were 600 iterations, perplexity 32, learning rate 10 and the algorithm ran without any supervision based on sample labels.

The projection clearly shows a number of clusters of different sizes, each consisting of data points from all 4 classes. Each cluster is probably a representative of a single

viewpoint of a scene with some of them being closer to each other or even almost merged together. This shows that the encoder model is not capable of generalizing over different scenes or viewpoints. One possible explanation for such behavior is not using a diverse enough dataset.

When focusing on each cluster individually, the data point distribution holds relations for similar sports poses. Classes `down-down` and `up-up` are usually far apart from each other within the cluster and while the `down-up` and `up-down` are between them. Some images of poses contain arms pointing almost perfectly horizontally and their classification cannot be precise. These cases have to be taken into account.

5.2 Evaluation of Encoder on Validation Dataset

The encoder itself is just a single component in the whole model that performs the classification. Its performance cannot be easily measured like a normal classifier – by counting how many of the validation data were correctly assigned their class. There are no ground truths to the inputs, no image of a sports pose has a correct nor false embedding. The only way to measure the encoder's performance is by comparing one embedding to another. If the objective is to have similar sports poses close to each other in the embedding space, the distances of embeddings can be compared.

The encoder is trained with a triplet loss function whose aim is to have two embeddings of different images of the same sports pose closer to each other than two embeddings of a distinct pose. The distance between correct and false pairs should also be greater than some fixed value called margin. Therefore, the same function can be also used to evaluate the encoder. The only difference is that the margin is set to 0, whereas during the learning process, the value is above 0.

The objective of this experiment was to evaluate the performance of the encoder model on different dimensionalities of the embedding space. The numbers of dimensions used for testing were 16, 32, 64, 128, 256, and 512. The margin of the triplet loss was set to 0.1. The model was trained for 50 epochs on the same training data and evaluated on the validation subset after each epoch. The dataset used for this experiment is Upper Body Dataset from Section 3.2.2. Dataset was split so that 90 % of it was used for training and 10 % for validation. Each model was trained 5× on the same dataset but shuffled with a distinct seed. To provide consistency of training data between different embedding space dimensionalities, the seed had the same value from 0 to 4 in the 5 runs. The best validation accuracy of all epochs was taken as the model's accuracy. The obtained results are presented in form of boxplots in Figure 5.2. The format of the boxplots is from the bottom: minimum, first quartile, median, third quartile, and maximum.

The highest median accuracy on the validation dataset achieved a model that encoded the sports pose images into 64-dimensional vectors. Embedding spaces with 32 and 16 dimensions might achieve comparable accuracy in some runs but their variance is very high. This suggests that the model is not always capable of finding efficient enough encoding to store all the information about the pose, even though it might be possible. Models producing encodings with 64 and more dimensions show less variance in accuracy which advocates for their ability to save all the necessary information in the embedding. Their median accuracy declines with rising dimensionality. That is a corresponding incident since encoding pose into a higher-dimensional space is a more difficult task and with rising complexity, the accuracy drops. From these assumptions, an embedding space with 64 dimensions provides the best results on Upper Body Dataset.

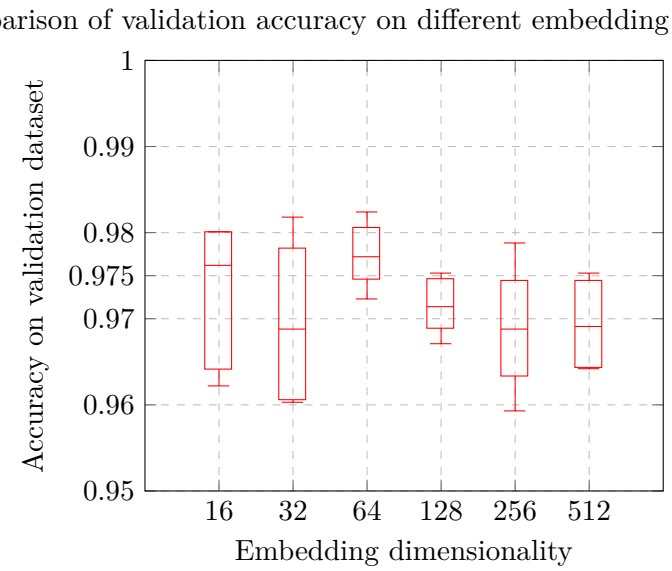


Figure 5.2: When an image of a sports pose is encoded into a vector in embedding space, its dimensionality can play a role in the performance of the model. The encoder model was tested on a number of dimensions between 16 and 512. The best median accuracy on validation data had an embedding space with 64 dimensions.

Dataset with very high diversity in sports poses requires more information to be stored and therefore an embedding space with more dimensions. The correctly chosen size of the embedding space can influence the performance of the model and the time it requires for fitting on the dataset. Therefore, it is advised to tune this hyperparameter to match the dataset complexity.

5.3 Comparison of Self-Supervised and Supervised-Trained Models

The main advantage of models trained in a self-supervised manner is their ability to perform well with datasets containing a smaller number of labeled data than what would supervised training needed. This advantage is shown in the evaluation done on different-sized datasets in the following experiments.

The encoder and classifier models described in sections 4.1 and 4.2 is used as the self-supervised learning benchmark. For the supervised learning representative, the most similar network is chosen. This model and its comparison to the self-supervised one is in Section 4.3.

Since each model is trained in a different way, it is not trivial to set the borderline for the number of epochs used for training. For that reason, each model was trained for a sufficient number of epochs after which it no longer improved on validation data. The self-supervised model consists of two parts – the encoder which creates the embedding from an image and the recognizer which classifies the pose from the embedding. The encoder was trained for 30 epochs and the recognizer for 20 epochs. The encoder was trained once and stayed the same for the whole experiment while the recognizer was fitted for every dataset sample. The supervised model was trained for 50 epochs on each dataset sample.

Dataset used for these experiments is thoroughly described in Section 3.2 – Upper Body Dataset. It contains 3,804 images of 1,268 poses, each captured from 3 different angles. The poses are not necessarily unique but the images differ in background and clothes of the person. Overall, there are 10 different scenes with 2 possible backgrounds and different clothing of the person in each of the scenes. The poses represent possible movements of a person's arms in all directions and joints' bendings, other parts of the body such as the torso, head, or legs are not moving.

The first experiment is done on the Directions Dataset which contains 4 classes that differ in the position of arms – left or right arm is pointing either down or up. Since some positions may be questionable, the rule of thumb during the labeling of data was whether the wrist is above or below the corresponding shoulder. The second experiment was done on Bent Dataset with 16 classes that extended the Directions Dataset with one more attribute – whether the arms are bent or not.

While the first experiment only evaluates the accuracy of choosing the correct class (top-1 accuracy), the second one evaluates also the accuracy of whether the correct class is within 3 of the most probable outcomes (top-3 accuracy). The decision to provide results in this format was made based on the number of classes in used datasets.

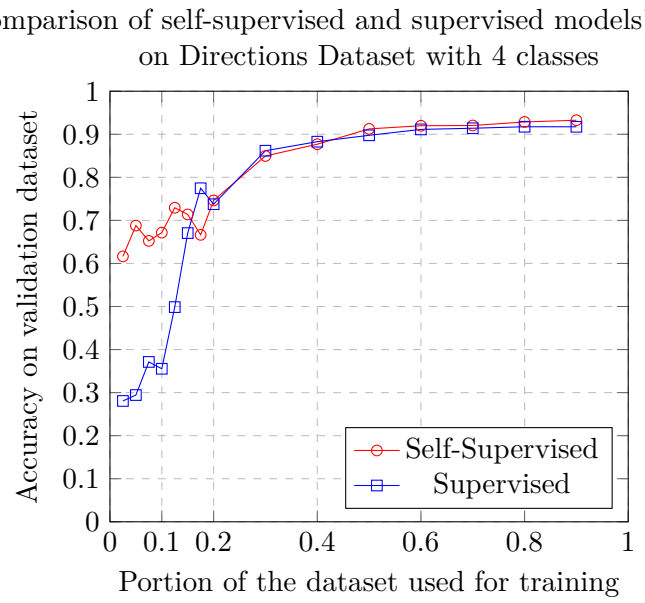
Both of tested models were fitted on datasets of different sizes and then their accuracy on never-seen validation data was evaluated. Before each training, the whole dataset was shuffled and then divided into training and validation subsets. The portion of data used for training was the changing variable and it ranged from 0.9 to 0.025. The rest of the data were always used for validation. This approach decreases variance in the training dataset and concurrently increases it in the validation dataset. Because of this, the model's ability to generalize well is displayed.

To keep the experiment fair, the dataset for each experiment was shuffled with the same seed for both models and, therefore, they had the same data for training and evaluation. For each dataset split portion, 10 runs of fitting and evaluating were done, each one with a different seed. Seeds were chosen deterministically as integers from 0 to 9. Accuracy on validation data was then averaged over all 10 runs to get the final accuracy of the model for a given fraction of training data.

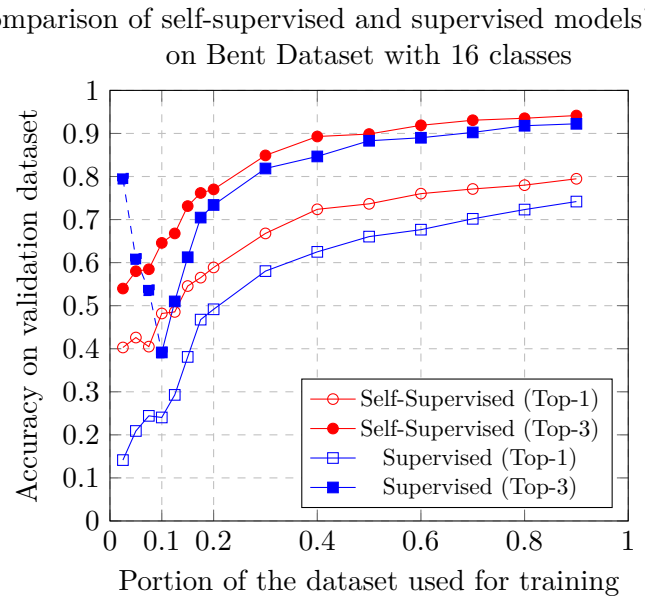
The results of the experiment on the Directions Dataset are shown in Figure 5.3a. For training dataset portions down to 15 % (which equals 570 of the 3,804 images used for training), the performances of self-supervised and supervised models are on a par. When the training dataset portion decreases to 12.5 % (475 of 3,804 images) the supervised model starts to degrade and with just 2.5 % images it approaches accuracy 25 % which is for 4 classes basically a random guess. While the self-supervised model keeps its accuracy above 60 % even when trained on 2.5 % data which equals to 95 images for training and 3,709 for validation.

Experiment number 2 with training and evaluation done on Bent Dataset provides not only top-1 but also top-3 accuracy. The results are presented in Figure 5.3b. While the previous experiment included only 4 classes, the Bent Dataset consists of 16 classes and this difference made an impact on the results. The accuracy of both models dropped down by approximately 0.2 overall. The self-supervised model performed better than the supervised model in all provided dataset splits. This is a display of one of the advantages of self-supervised learning, it adapts better to a higher number of classes since it already learned the key features on unlabeled data. When the amount of training samples approaches less than 10 for each class overall (the portion of 0.025), the accuracy of the supervised model drops significantly while the self-supervised model's accuracy stabilizes.

The top-3 accuracy reveals important information about the supervised model when trained on 10 % of the Bent Dataset – part of the plot with the dashed line. The predicted results degrade in a way that the top-3 accuracy equals 1.0 for more and more of the experiment runs and the top-3 accuracy suddenly rises. Since the top-1 accuracy declines, the model is not performing better, rather it found some workaround that produces these improbable results. The model is also not trained to maximize the top-3 accuracy, it is trying to minimize the loss function. Self-supervised learning clearly provides results with higher or equal accuracy for all experiments and shows its advantages mainly on datasets with a low number of training samples.



(a) Self-supervised model displays better accuracy for small training datasets. With the dataset size getting larger, both models perform similarly.



(b) Self-supervised model performs better overall for all dataset splits in top-1 and top-3 accuracy, especially when the number of training samples approaches an average of fewer than 10 samples for a class with a portion equal to 0.025. For the training portion of less than 0.1, the top-3 accuracy is degraded. This section of the plot is marked with a dashed line.

Figure 5.3: Performance of self-supervised and supervised models on validation dataset based on what fraction of the whole dataset was used for training. The rest of the dataset was used for validation. Each data point is an average of 10 runs.

Chapter 6

Conclusion

The goal of this thesis was to develop a model that can classify sports poses from images and uses self-supervised learning to achieve better results on datasets with a small amount of annotated samples. Time-contrastive learning was chosen as the approach to achieve self-supervision of the data. For that, a set of tools for video preparation had to be implemented and tested on various scenes. After collecting and preparing a dataset of sports poses, a self-supervised model consisting of an encoder and classifier was developed. The self-supervised model was evaluated in comparison to the model trained with supervision and results were presented and discussed.

A set of tools for dataset preparation was developed and it is effectively working on any number of videos of different scenes. The tools can trim and crop videos easily with the least amount of manual work. The synchronization tool can automatically adjust the video start times and lengths to align any number of videos. A useful dataset of images for time-contrastive learning is then detected by using sparse optical flow and exported for future use. A simple tool for labeling images is also implemented.

Two datasets were recorded and prepared with the mentioned tools. The first one contains hand gestures with different backgrounds and was mainly used for testing the dataset preparation tools. The second one contains recordings of the upper body with diverse arm movements. This dataset was used to train and evaluate the self-supervised model. It contains 3804 images of sports pose with two sets of annotations of 4 and 16 classes.

Two models constructing the self-supervision architecture were developed: encoder and classifier. The encoder uses ResNet-50 architecture together with a triplet loss function to provide embeddings of sports poses. The classifier is a simple dense neural network that takes embedding vectors and classifies them. Another model doing the same task but with supervision was also developed to have a comparison between the two architectures.

The embedding space of the encoding is visually analyzed with t-distributed stochastic neighbor embedding and the resulting visualization shows the upsides and downsides of the encodings. Possible settings of encoding dimensionality are evaluated on the validation dataset and the results are compared according to the median accuracy and the variance of results. Finally, a self-supervised model is compared to a model trained with supervision. Datasets with different amounts of annotated samples were used for training and the validation accuracy of both models was compared. The self-supervised model performs similarly on dataset with hundreds of samples from each class but when the amount of samples drops to lower tens or even under 10 per class, the self-supervised model outperforms the supervised one by tens of percent on the validation accuracy.

Future work on the project was proposed in various directions. To increase the variability in sports poses, yoga poses are recommended together with possible sources of data and related work in this direction. Numerous other loss functions that can be used for self-supervision are presented together with their advantages. The obtained embeddings can not only be used for classification but also for other computer vision challenges. These possibilities such as pose estimation or embedding vector operations are discussed. Lastly, the implemented work could also be used for other object classification, not only for sports poses.

Bibliography

[1] Bradski, G. and Kaehler, A. *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*. 2nd ed. O'Reilly Media, Inc., 2013. ISBN 1449314651.

[2] Chen, H.-T., He, Y.-Z., Hsu, C.-C., Chou, C.-L., Lee, S.-Y. et al. Yoga Posture Recognition for Self-training. In: Gurrin, C., Hopfgartner, F., Hurst, W., Johansen, H., Lee, H. et al., ed. *MultiMedia Modeling*. Cham: Springer International Publishing, 2014, p. 496–505. ISBN 978-3-319-04114-8.

[3] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. et al. ImageNet: A Large-Scale Hierarchical Image Database. In: *2009 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2009.

[4] Dumoulin, V. and Visin, F. *A guide to convolution arithmetic for deep learning*. 2016. Available at: https://arxiv.org/abs/1603.07285.

[5] Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning*. 1st ed. MIT Press, 2016. ISBN 0262035618. http://www.deeplearningbook.org.

[6] He, K., Zhang, X., Ren, S. and Sun, J. *Deep Residual Learning for Image Recognition*. 2015. Available at: https://arxiv.org/abs/1512.03385.

[7] Hinton, G. E. and Roweis, S. Stochastic Neighbor Embedding. In: Becker, S., Thrun, S. and Obermayer, K., ed. *Advances in Neural Information Processing Systems*. MIT Press, 2003, vol. 15. ISBN 0-262-02550-7. Available at: https://proceedings.neurips.cc/paper/2002/file/6150ccc6069bea6b5716254057a194ef-Paper.pdf.

[8] Horn, B. K. and Schunck, B. G. Determining optical flow. *Artificial Intelligence*. 1981, vol. 17, no. 1, p. 185–203. DOI: https://doi.org/10.1016/0004-3702(81)90024-2. ISSN 0004-3702. Available at: https://www.sciencedirect.com/science/article/pii/0004370281900242.

[9] Kingma, D. P. and Ba, J. *Adam: A Method for Stochastic Optimization*. 2017. Available at: https://arxiv.org/abs/1412.6980.

[10] Maaten, L. van der and Hinton, G. Visualizing Data using t-SNE. *Journal of Machine Learning Research*. 1st ed. 2008, vol. 9, no. 86, p. 2579–2605. Available at: http://jmlr.org/papers/v9/vandermaaten08a.html.

[11] Murphy, K. P. *Probabilistic Machine Learning: An introduction*. 1st ed. MIT Press, 2022. ISBN 0262046822. Available at: https://probml.github.io/pml-book/book1.html.

[12] Ramsundar, B. and Zadeh, R. B. *TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning*. 1st ed. O'Reilly Media, Inc., 2018. ISBN 1491980451.

[13] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. and Chen, L.-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, p. 4510–4520. DOI: 10.1109/CVPR.2018.00474.

[14] Schroff, F., Kalenichenko, D. and Philbin, J. FaceNet: A unified embedding for face recognition and clustering. In: Google Inc. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, p. 815–823. DOI: 10.1109/CVPR.2015.7298682. ISSN 1063-6919. Available at: http://dx.doi.org/10.1109/CVPR.2015.7298682.

[15] Sermanet, P., Lynch, C., Chebotar, Y., Hsu, J., Jang, E. et al. *Time-Contrastive Networks: Self-Supervised Learning imafrom Video*. 2018. Available at: https://arxiv.org/abs/1704.06888.

[16] Shi, J. and Tomasi. Good features to track. In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, p. 593–600. DOI: 10.1109/CVPR.1994.323794. Available at: https://ieeexplore.ieee.org/document/323794.

[17] Sohn, K. Improved Deep Metric Learning with Multi-class N-pair Loss Objective. In: Lee, D., Sugiyama, M., Luxburg, U., Guyon, I. and Garnett, R., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2016, vol. 29. Available at: https://proceedings.neurips.cc/paper/2016/file/6b180037abbebea991d8b1232f8a8ca9-Paper.pdf.

[18] Song, H. O., Xiang, Y., Jegelka, S. and Savarese, S. Deep Metric Learning via Lifted Structured Feature Embedding. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, p. 4004–4012. DOI: 10.1109/CVPR.2016.434. Available at: https://ieeexplore.ieee.org/document/7780803.

[19] Szeliski, R. *Computer Vision: Algorithms and Applications*. 1st ed. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN 1848829345.

[20] Verma, M., Kumawat, S., Nakashima, Y. and Raman, S. *Yoga-82: A New Dataset for Fine-grained Classification of Human Poses*. 2020. Available at: https://arxiv.org/abs/2004.10362.

[21] Wang, J., Zhou, F., Wen, S., Liu, X. and Lin, Y. Deep Metric Learning with Angular Loss. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, p. 2612–2620. DOI: 10.1109/ICCV.2017.283. Available at: https://ieeexplore.ieee.org/document/8237545.