BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Hladchenko Vladyslav**	Personal ID number:	**466351**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Branch of study:	**Software**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Robot control by mean of gestures

Bachelor’s thesis title in Czech:

Ovládání robotu gesty

Guidelines:

The aim of the task is to use free software to recognize the human pose from the camera image and gestures, which could control the movement of the mobile robot. The choice of suitable gestures and the way of their recognition is a part of the work. Learn the methods of camera image processing and research the available methods to find human pose in the image and recognize the positions of his/her limbs, which will serve as the basis for gesture detection. Focus on methods with freely available implementation. Using the available libraries, create a demonstration application for gesture recognition and test its success and robustness.

Bibliography / sources:

[1] Schneider Pascal, Memmesheimer Raphael, Kramer Ivanna, Paulus Dietrich. “Gesture Recognition in RGB Videos Using Human Body Keypoints and Dynamic Time Warping”.
[2] Oleg Sergiyenko, Wendy Flores-Fuentes, Paolo Mercorelli. “Machine Vision and Navigation 1st ed.“.
[3] Davide Brugali. “Software Engineering for Experimental Robotics”.
[4] Lei Shi, Yifan Zhang, Jian Cheng, Hanqing Lu. “Skeleton-Based Action Recognition with Multi-Stream Adaptive Graph Convolutional Networks”.
[5] Okan Kopuku, Ahmet Gunduz, Neslihan Kose, Gerhard Rigoll. “Real-time Hand Gesture Detection and Classification Using Convolutional Neural Networks”.
[6] Michael A. Goodrich, Alan C. Schultz. “Human–Robot Interaction: A Survey”.

Name and workplace of bachelor’s thesis supervisor:

Ing. Jan Chudoba, Intelligent and Mobile Robotics, CIIRC

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **13.02.2020** Deadline for bachelor thesis submission: **05.01.2021**

Assignment valid until: **30.09.2021**

Ing. Jan Chudoba Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Bachelor thesis

Robot control by mean of gestures

Vladyslav Hladchenko

Supervisor: Ing. Jan Chudoba

Study Programme: Open Informatics

Field of Study: Software

January 5, 2021

Aknowledgements

I want to thank my supervisor Jan Chudoba and all people who participated in creation of gestures dataset.

Declaration

I declare that I elaborated this thesis on my own and that I mentioned all the information sources and literature that have been used in accordance with the Guideline for adhering to ethical principles in the course of elaborating an academic final thesis.

In Prague 5. 1. 2021 ....

Abstract

This work describes the approach of gesture recognition that uses modifyed Openpose algorithm and PoseNet for pose keypoint extraction and Dynamic Time Warping with 1NN Classifier for the classification of keypoint sequences.
The proposal of actions for robot control is made, with certain extensions to enlarge the control possibilities.

Abstrakt

Práce popisuje přístup rozpoznávání gest, který používá modifikovaný algoritmus Openpose a PoseNet pro extrakci pozic kloubů a Dynamic Time Warping s 1NN klasifikátor pro klasifikaci sekvencí pozic kloubů.
Je vytvořen návrh akcí pro řízení robota s určitými rozšířeními.

Contents

List of Figures

List of Tables

Chapter 1

1 Problem research

For computer vision tasks convolutional neural network (a class of deep learning methods) has become dominant method. The purpose of convolution in case of a convolutional networks is to extract features from the input image, it preserves the spatial relationship between pixels by learning image features using smaller pieces of input data. However, CNN is data intensive because of a large number of parameters to estimate (finding kernels in convolution layers and weights in fully connected layers), and therefore is more computationally expensive, requiring GPUs for model training and inference. Kopuku et al. [7] proposed a method of gesture recognition using two convolutional networks: a detector which is a 3D CNN architecture to detect gestures and a classifier which is a deep CNN to classify the detected gestures using NVIDIA Titan Xp GPU.

Research on human motion detection and gesture recognition has also revolved around the use of Hidden Markov Models (HMM). HMMs allow for the encoding of continuous streams of data into perceived gestures and/or behaviours, by identifying the state sequence associated to the motion represented in the observable events. Their main downside is that they require large training databases to train the HMM associated to each particular behavior and are particularly sensitive to segmentation errors. HMM-based clustering is very sensitive to segmentation errors that can easily provide an inefficient or ill generalization from training data. They also require large example datasets.

[8] shows an approach of pose estimation from video that uses convolutional network architecture that is able to benefit from temporal context by combining information across the multiple frames using optical flow.

Some approaches use body markers or CyberGloves. Also depth data is used to simplify the gesture recognition task.

The Kinect provides synchronized depth and color images

These approaches might be unsuitable for real world scenarios where the presence of high performance hardware or mocap systems may be impossible.

Chapter 2

Approach

Approach is based on the method described in [10], which proposed gesture recognition method using only RGB data and not requiring resource-intensive learning for adding/removing new gestures.

Retrieving coordinates of pose key points for each frame, Dynamic Time Warping (DTW) algorithm is then applied to find the closest gesture.

2.1 Pose estimation

For detection of the pose-key points, the Ildoonets tf-pose-estimation [5] based models and tensorflows PoseNet [12] model is used.

Idoonets tf-pose-estimation implements the Openpose 2D human pose estimation algorithm using Tensorflow framework, with changes to the network structure which enables decent FPS for real-time processing on the CPU.

PoseNet is an optimized model for mobile and embedded platforms implemented by tensorflow developers.

2.1.1 Openpose

CMU-Perceptual-Computing-Lab Openpose [1] takes the image, extracts feature maps **F** with VGG-19 network. The VGG-19 network is an architecture of convolutional neural network created by Visual Geometry Group at Oxford, with the depth of 19 layers. The output of VGG-19 is passed to the Stage 1 of the two-branch multi-stage CNN (shown in the Figure 2.1) that simultaneously predicts a set of 2D confidence maps S^1 of body part locations for each person (Branch 1) and a set of 2D vector fields L^1 part affinities, which encode the degree of association between body parts (Branch 2). Then stages 2-6 follow, each stage takes the original image features **F** concatanated with the outputs of Branches 1 and 2 from the previous stage to produce more refined predictions and eliminate false part-to-person associations.

The part affinity is a 2D vector field for each limb, which for each pixel belonging to a person encodes the direction as a unit-vector that points from one part of limb to the other.

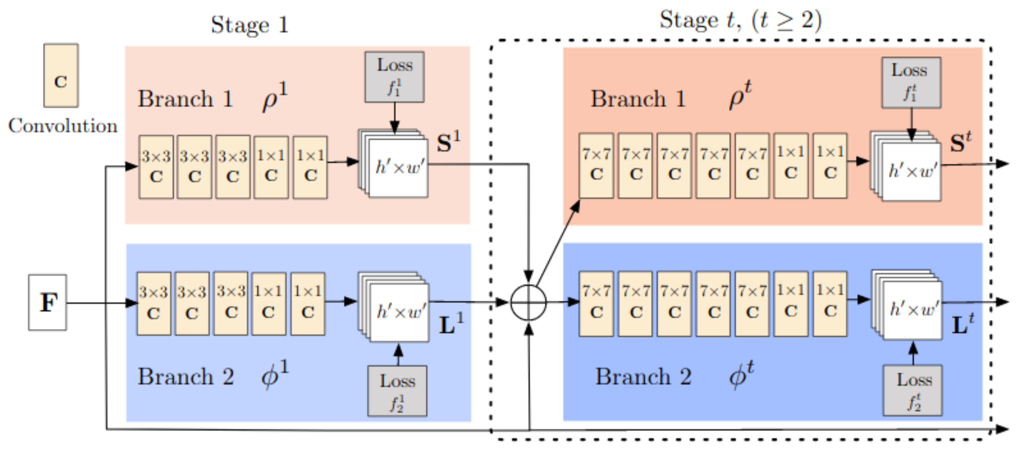

Figure 2.1: Stages of the two-branch multi-stage CNN used by Openpose, taken from the Openpose paper.

2.1.2 MobileNets

MobileNet [4] is an architecture of efficient CNN models for mobile and embedded vision applications, created by Google, that uses depthwise separable convolutions and introduces two simple global hyper-parameters that efficiently trade off between latency and accuracy.

Convolution is an operation that extracts features (such as specific shapes, color, etc.) performing dot product of kernels (also called filters, typically smaller than image, e.g. of size 3×3×depth, 5×5×depth) and parts of the input tensor of the same size (as kernel), for each such tensor part.

Standard convolution takes an $h_i \times w_i \times d_i$ input tensor, and applies convolutional kernel of size $k \times k \times d_i \times d_j$ to produce an $h_i \times w_i \times d_j$ output tensor. The computational cost is $h_i \cdot w_i \cdot d_i \cdot d_j \cdot k \cdot k$.

Depthwise separable convolution is a depthwise convolution followed by a 1×1 convolution, also called a pointwise convolution.

The depthwise convolution applies a single filter to each input channel. The computational cost is $h_i \cdot w_i \cdot d_i \cdot kk$ (d_j is eliminated).

The pointwise convolution applies a standard 1×1 convolution to combine the outputs the depthwise convolution. The computational cost is $h_i \cdot w_i \cdot d_i \cdot d_j$.

Thus, the computation cost of a depthwise separable convolution is $h_i \cdot w_i \cdot d_i \cdot (k^2 + d_j)$, which is $k^2 \cdot d_j / (k^2 + d_j)$ less operations than the standard one.

The two hyper-parameters: width multiplier and resolution multiplier are further used to make the model smaller.

The width multiplier $\alpha \in (0, 1]$ reduces the number of channels (depth) of each layer.

The resolution multiplier $\rho \in (0, 1]$ reduces the width and height of tensors.

2.1.3 Idoonet tf-pose-estimation

Ildoonet replaces the VGG-19 network used by Openpose for feature maps **F** extraction with the MobileNets shown in table 2.1.

name	parameters
MobileNetV2_0.75	$\alpha = 0.75, \rho = 0.75$
MobileNetV2_1.4	$\alpha = 1.4, \rho = 1$
MobileNetV1_0.5	$\alpha = 0.5, \rho = 0.5$

Table 2.1: Different MobileNets used by Ildoonet.

2.1.4 Training the MobileNets

Mobilenet V2 Imagenet Checkpoints [11] are chosen as pre-trained checkpoints for training on COCO-2017 dataset. Details of person category (pose keypoints) of COCO-2017 dataset are shown in table 2.2.

The weights of the pre-trained checkpoints were obtained by training on the ILSVRC-2012-CLS dataset for image classification ("Imagenet").

The parameters and performance of this checkpoint is shown in table 2.3. MAC is the number of multiply-accumulates needed to compute an inference on an image.

Number of training images	Number of validation images	Number of test images
118287	5000	40670

Table 2.2: Details of COCO-2017 dataset.

name	Classification Checkpoint	MACs (M)	Parameters (M)	Top 1 Accuracy	Top 5 Accuracy
MobileNetV2_0.75	`float_v2_1.4_224`	582	6.06	75.0	92.5
MobileNetV2_1.4	`float_v2_0.5_224`	97	1.95	65.4	86.4
MobileNetV1_0.5	`MobileNet_v1_0.75_224`	317	2.59	68.4	88.2

Table 2.3: MobileNet chekpoints.

Training is done on AWS ml.p2.8xlarge instance with details shown in table 2.4.

GPUs	vCPUs	RAM (GiB)
`8x NVIDIA K80`	32 (Intel Xeon E5-2686 v4)	488

Table 2.4: AWS ml.p2.8xlarge instance details.

The parameters of training are shown in table 2.5.

batch size	epochs	steps	start learning rate	image resolution
128	20	37000	0.0001	432 × 368

Table 2.5: training parameters.

Heatmap and PAF losses graphs in figures 2.2 2.5 show the process of training `MobileNetV2_1.4`. Graphs and images were generated during training and visualised in Tensorboard.

Figures 2.6 - 2.8 show the evolution keypoint heatmaps and PAF-vactor maps for one of the validation images during training.

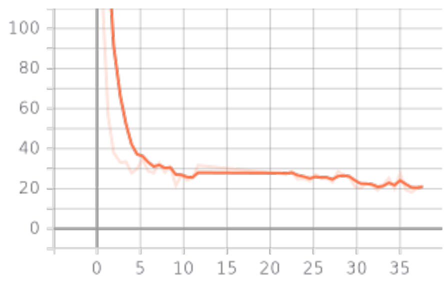
Figure 2.2: Heatmap Loss.

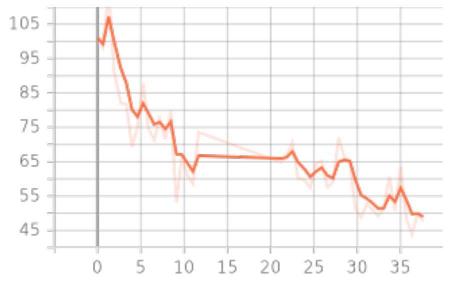
Figure 2.3: PAFmap(Part Affinity Field) Loss.

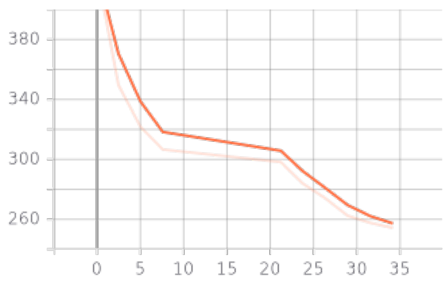
Figure 2.4: Validation Heatmap Loss.

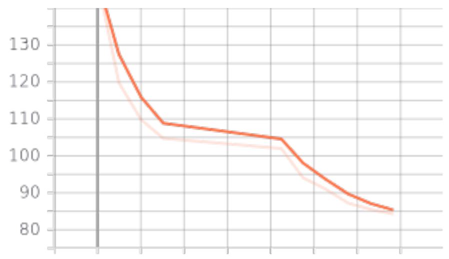
Figure 2.5: Validation PAFmap(Part Affinity Field) Loss.

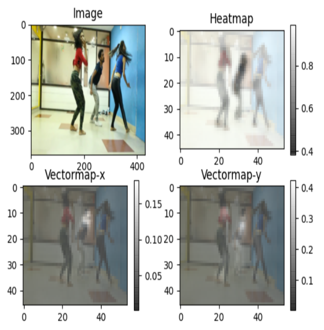

Figure 2.6: Training step 2000.

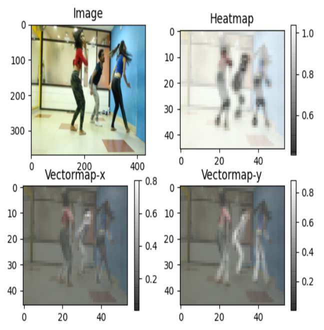

Figure 2.7: Training step 9000.

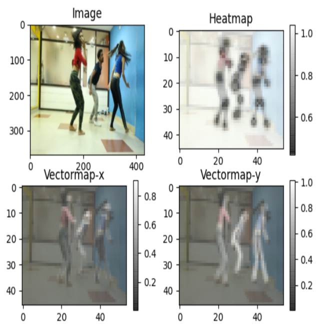

Figure 2.8: Training step 27000.

2.2 Gesture classification

For gesture classification Dynamic Time Warping algorithm is used in conjunction with One-Nearest-Neighbor classifier, which simply means that the closest gesture is taken.

2.2.1 Dynamic Time Warping

Dynamic Time Warping is an algorithm for measuring similarity between two temporal sequences, which may vary in speed. Given two sequences $X = (x_1, x_2, ..., x_n)$,$Y = (y_1, y_2, ..., y_m)$, the algorithm finds optimal warping path and optimal distance between the sequences. Example on two sequence of 1-dimensional data is shown in Figure 2.9

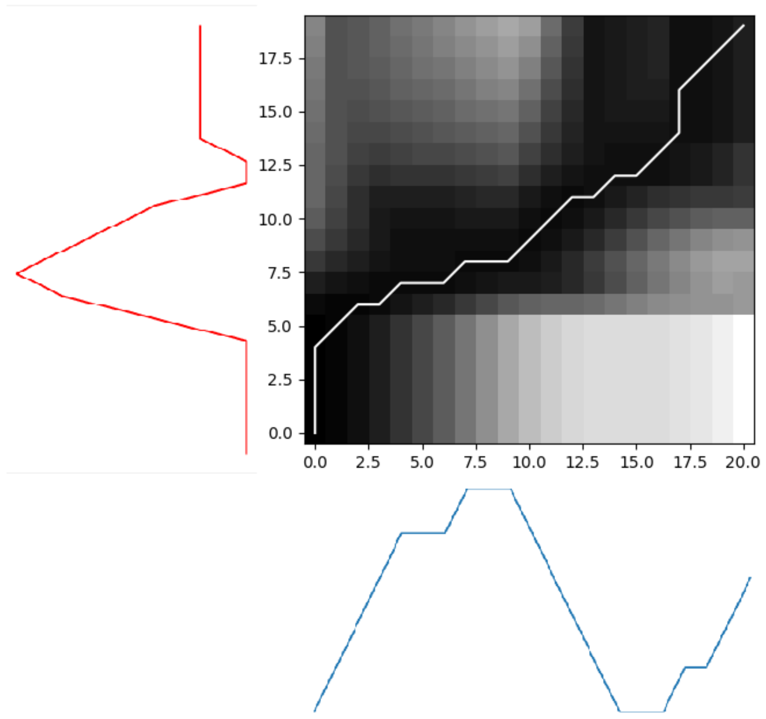

Figure 2.9: Optimal alignment (white graph) found with Dynamic time Warping on two sequences (red and blue colored), The cost of each step in optimal alignment graph is encoded as shades of gray: the darkest - the cheapest.

Optimal warping path is a sequence of points $p = (p_1, p_2, ..., p_l)$, with $p_k = (p_i, p_j) \in [1..n] \times [1..m]$ for $k \in [1, l]$, which defines correspondence of an element $x_{p_i} \in X$ to $y_{p_j} \in Y$. Distance between two sequences is then given by $\sum_{i=1}^{l} dist(p_i)$.

First, the local cost matrix C is constructed using the distance function:

$C(i, j) = dist(x_i, y_j), i \in [1, n], j \in [1, m]$.

Then the accumulation matrix A, based on C:

$$A(i,j) = \begin{cases} \sum_{k=1}^{j} C(x_1, y_k), & \text{if i =1} \\ \sum_{k=1}^{1} C(x_k, y_1), & \text{if j =1} \\ minA(i-1, j-1), A(i-1, j), A(i, j-1) + C(x_i, y_j), & \text{otherwise} \end{cases}$$

for $i \in [1, n], j \in [1, m]$

The cost of an optimal warping path (optimal distance between sequences) is given by $A(n, m)$.

2.2.2 Dynamic Time Warping application

DTW is applied to sequences of ordered sets of key points yielded by the pose estimator.

To achieve scale and translational invariance all keypoints are shifted so that Neck keypoint is in the origin, further, distance between shoulders is normalized.

Only 6 of 18 key points returned by the pose estimator are relevant in classifying a hand gesture, namely: left shoulder, right shoulder, left elbow, right elbow, left wrist and right wrist. Neck keypoint is always in the origin, it gains no additional information about the gesture. Distance between two elements of sequences is given by the sum of squares of the Euclidean distances between corresponding key points:

$$dist(a, b) = \sum_{i=1}^{6} \|a_i - b_i\|_2^2 \tag{2.1}$$

2.2.3 DTW customisation and result usage

For every gesture threshold is empirically acquired so that no gesture is recognised when no gesture is performed in reality.

Since after processing frames $[n, n + k]$, where k is the length of a particular gesture, the only new information in frames $[n + 1, n + k + 1]$ is positions of key points in the last frame, there is no need to compute cost matrix C for each gesture anew. When receiving a new frame, cost matrix is obtained by truncating the first row of the old one and adding new row of distances from the new frame (set of key point positions) to each frame in gesture. This reasoning speeds up computing of cost matrix C by $k - 1$ times. Also computation of the accumulated cost matrix stops in case if minimum value in the current row is above threshold for a given gesture.

By properties of DTW algorithm, for some number $l > 0$ of frames, right after particular gesture recognition in frame n, sequences of frames $[n - k + l, n + l]$ (where k is the length of particular gesture) might be still recognized as this gesture. To avoid this, the feature cost matrix is erased, so that it will be computed anew from frame $n + 1$ and information about recognition will be given only once.

2.3 Robot control proposal

2.3.1 Proposed gestures

This section describes the proposal for gestures and some extensions that might be used for the robot control.

For robot control following gestures can be used, mapped to actions. The proposed gestures are sufficently diverce to be predcted with the leats miss rate:

- left hand to the side (Figure 2.10) - turn left
- right hand to the side (Figure 2.11) - turn right
- left hand forward (Figure 2.12) - stop/start moving
- right hand forward (Figure 2.13) - stop/start moving

- left wrist forward with fixed elbow (Figure 2.14)- stop/start moving
- right wrist forward with fixed elbow (Figure 2.15) - stop/start moving
- right wrist clockwise motion around the elbow (Figure 2.17) - rotate right 90 degrees
- right wrist anticlockwise motion around the elbow (Figure 2.19) - rotate left 90 degrees
- left wrist clockwise motion around the elbow (Figure 2.16) - rotate right 180 degrees
- left wrist anticlockwise motion around the elbow (Figure 2.18) - rotate left 180 degrees

The last two gestures can be mapped to any action, e.g. making photo or end effector action. Alternatively, one gesture can be used to drive the robot into specific mode e.g. effector control, in which other gestures are mapped to effector actions, and another one is used to return the robot to the previous mode.

These gestures are shown in following figures 2.10-2.19 as keypoint movements. Keypoint positions are normalized, centered and mirrored. Arrows represent the movements of body keypoints, one arrow per keypoint. Keypoints without the arrows are expected to be still during the gesture performance. Start position of keypoint locates at the arrow start, the final potition on its end.

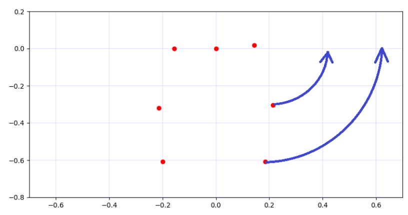
Figure 2.10: Left hand to the side gesture.

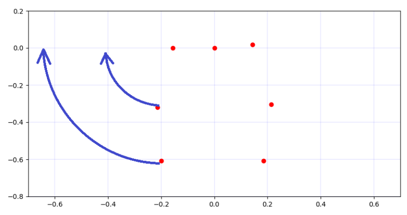
Figure 2.11: Right hand to the side gesture.

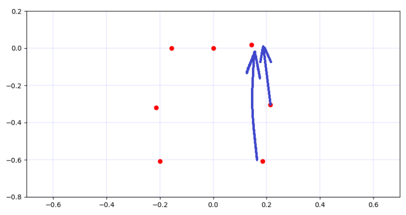
Figure 2.12: Left hand forward gesture.

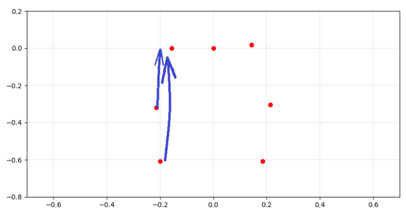
Figure 2.13: Right hand forward gesture.

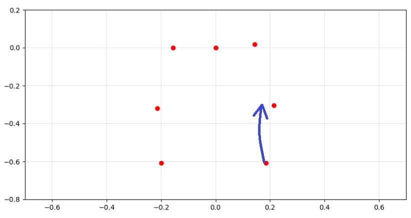

Figure 2.14: Left wrist forward with fixed elbow gesture.

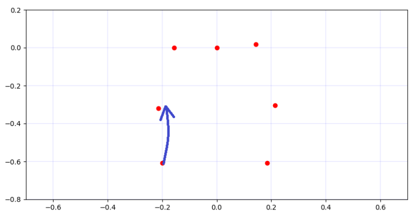

Figure 2.15: Right wrist forward with fixed elbow gesture.

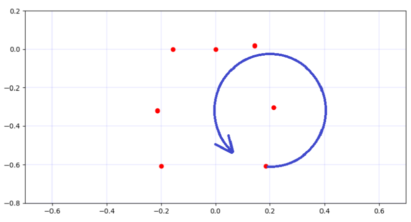

Figure 2.16: Left wrist clockwise motion around the elbow gesture.

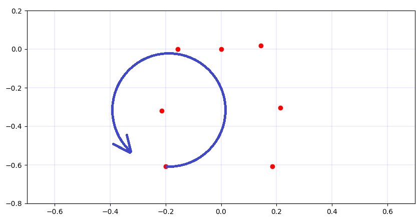

Figure 2.17: Right wrist clockwise motion around the elbow gesture.

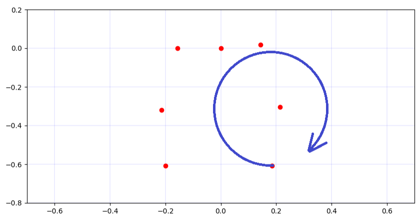

Figure 2.18: Left wrist anticlockwise motion around the elbow gesture.

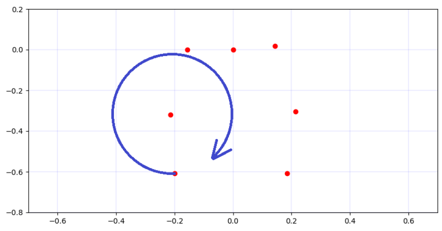

Figure 2.19: Right wrist anticlockwise motion around the elbow gesture.

2.3.2 Gesture holding

Gesture holding can be used to expand the number of possible commands to robot. After some gesture is recognised, it can be "held" until the "main part" of the gesture is being performed. For instance, "left arm to the side" gesture can be used to command the robot to turn left until the left arm - the "main part" of gesture - stays in final position of gesture. Also, the right arm can be used to control the angle of the rotation: angle is inversely proportional to the distance of the right arm keypoints (wrist, elbow and shoulder) to the corresponding keypoints of the final position of "right arm to the side" gesture. "Right arm forward" gesture can be used to stop robot while the right hand remains extended and subsequent optional raising of the other hand will stop the robot completely, Then "Right arm forward" gesture can be used to command the robot to start moving. While the gesture is being held, distance function (equation 2.1) will be applied separately to the "main" and "optional" parts of the gesture until the distance from "main" part is below the threshold.

2.3.3 Hand shape classification

While gesture is being held, the classification of hand shape is possible to further extend the control possibilities. For example purposes two shapes from NUS hand posture datasest [6] where chosen, namely "fist" and "palm", as seen at 2.20

For classification the MobileNetV2 [9] is used. It is trained with transfer learning [13] from a network, that was previously trained on a large and general dataset (in this case ImageNet [3]). This method allows to avoid retraining of the entire model, the base convolution layers already contains features that are useful for images classification.

The NUS dataset contains 274 images of size 160x120 pixels for each hand shape. To increase the number of images for training, the data augmentation is performed: rotations and flips for each image.

Testing accuracy of the resulting model is 93.33%.

For model to work correctly the hand must occupy at least 50% of the image as shown on figure 2.20.

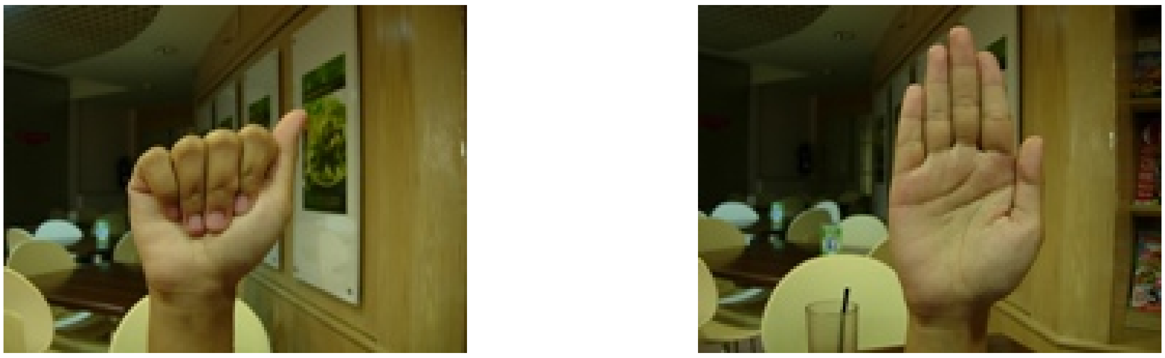

Figure 2.20: Wrist and palm examples from the NUS hand posture datasest.

Chapter 3

Testing and performance

3.1 Custom dataset

The gestures described in chapter 2.3.1 were performed by 6 people, each gesture 5 times, one person in frame. Gestures are recorded with the resolution of 640x480 pixels and framerate 30 FPS in a well lit environment.

Tables 3.1 - 3.4 show classification results using CMU original architecture and different MobileNet architectures for pose keypoint inference. The images are resized to size 432×368 pixels before fed to the network. For each gesture one instance is chosen as base for comparrison with the DTW method. Pose keypoints for base gesture and classified gestures are inferenced by the same network.

Figures 3.3 - 3.3 show gestures confusion.

Gesture	True positives	False positives	False negatives
Right hand forward	29 - 100%	0	0
Right hand to the side	29 - 100%	0	0
Right hand half forward	29 - 100%	0	0
Right hand clockwise	29 - 100%	1	0
Right hand anticlockwise	28 - 96%	0	1
Left hand forward	26 - 100%	0	0
Left hand to the side	28 - 100%	0	0
Left hand half forward	27 - 100%	4	0
Left hand clockwise	28 - 100%	0	0
Left hand anticlockwise	22 - 84%	0	4

Table 3.1: Testing chosen gestures with CMU original network for keypoints position inference.

Gesture	True positives	False positives	False negatives
Right hand forward	28 - 96%	0	1
Right hand to the side	29 - 100%	0	0
Right hand half forward	29 - 100%	1	0
Right hand clockwise	29 - 100%	1	0
Right hand anticlockwise	28 - 96%	0	1
Left hand forward	25 - 96%	0	1
Left hand to the side	28 - 100%	0	0
Left hand half forward	27 - 100%	5	0
Left hand clockwise	28 - 100%	0	0
Left hand anticlockwise	22 - 84%	0	4

Table 3.2: Testing chosen gestures with `MobileNetV2_1.4` network for keypoints position inference.

Gesture	True positives	False positives	False negatives
Right hand forward	23 - 79%	2	6
Right hand to the side	28 - 96%	5	1
Right hand half forward	25 - 86%	7	4
Right hand clockwise	26 - 89%	3	3
Right hand anticlockwise	23 - 79%	0	6
Left hand forward	23 - 88%	0	3
Left hand to the side	27 - 96%	3	1
Left hand half forward	24 - 88%	19	3
Left hand clockwise	26 - 92%	1	2
Left hand anticlockwise	15 - 57%	0	11

Table 3.3: Testing chosen gestures with `MobileNetV1_0.75` network for keypoints position inference.

Gesture	True positives	False positives	False negatives
Right hand forward	18 - 62%	0	11
Right hand to the side	28 - 96%	0	1
Right hand half forward	26 - 89%	69	3
Right hand clockwise	1 - 3%	1	28
Right hand anticlockwise	2 - 6%	2	27
Left hand forward	4 - 15%	1	22
Left hand to the side	3 - 10%	1	25
Left hand half forward	26 - 96%	51	1
Left hand clockwise	26 - 92%	0	2
Left hand anticlockwise	21 - 80%	0	5

Table 3.4: Testing chosen gestures with `MobileNetV2_0.5` network for keypoints position inference.

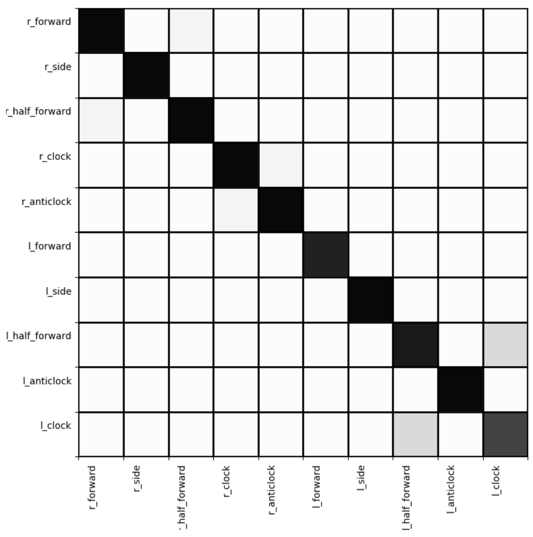

Figure 3.1: Confusion matrix for classification based on `MobileNetV2_1.4`.

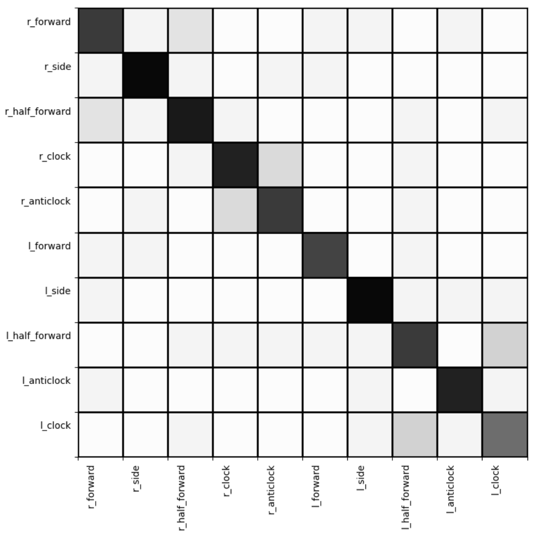

Figure 3.2: Confusion matrix for classification based on `MobileNetV1_0.75`.

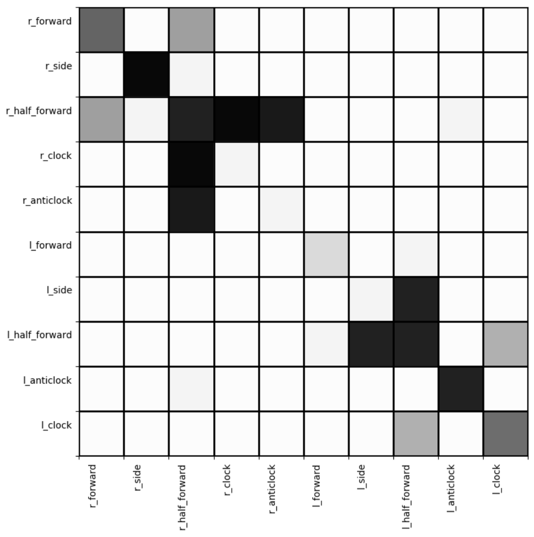

Figure 3.3: Confusion matrix for classification based on `MobileNetV2_0.5`.

Table 3.5 shows result of classification with DTW on images resized to 208×176 pixels.

Given this data it is clear that `MobileNetV2_1.4` architecture for feature extraction is sufficient for successful classification of chosen gestures, while taking less time on pose keypoints inference (see Tables 3.8 and 3.8 in section 3.2.1 Performance)

Gesture	True positives	False positives	False negatives
Right hand forward	29 - 100%	0	0
Right hand to the side	29 - 100%	0	0
Right hand half forward	29 - 100%	0	0
Right hand clockwise	29 - 100%	0	0
Right hand anticlockwise	29 - 100%	0	0
Left hand forward	26 - 100%	0	0
Left hand to the side	28 - 100%	0	0
Left hand half forward	27 - 100%	0	0
Left hand clockwise	28 - 100%	0	0
Left hand anticlockwise	26 - 100%	0	0

Table 3.5: Testing chosen gestures with `MobileNetV2_1.4` network for keypoints position inference. Images are resized to 208×176 pixels.

3.2 Multimodal Human Action Dataset

Gesture classification is tested on publically available dataset UTD-MHAD (Multimodal Human Action Dataset) [2] on chosen gestures that differs from each other the most. One recording from each gesture type is taken as reference - it is compared with all the other gestures.

Gesture	True positives out of 32	True positives percentage
Arm cross	28	87.5
Swipe left (left hand)	21	56.6
Swipe right (right hand)	26	81.3
Catch (left hand)	21	25.6
Catch (right hand)	17	56.1
Draw X	15	46.8
Push	13	50.6

Table 3.6: Testing 7 gestures from UTD-MHAD on tf-pose-estimation `MobileNetV2_1.14`.

"Draw X" was confused the most with "Catch left hand" gesture (13 False Positives). "Push" gesture was confused the most with "Draw X" (11 False Positives).

After adding "Throw (left hand)" and "Throw (right hand)" gestures, which are similar to "Catch" and "Draw X" gestures, TP of this gestures has reduced, while FP with "Throw" gestures has increased, as shown in the next table.

Gesture	True positives out of 32	True positives percentage
Arm cross	28	87.5
Swipe left (left hand)	20	56.6
Swipe right (right hand)	25	78.1
Catch (left hand)	20	62.5
Catch (right hand)	10	31.25
Draw X	6	18.75
Push	13	40.6
Throw (left hand)	6	18.75
Throw (right hand)	6	40.6

Table 3.7: Testing 9 gestures from UTD-MHAD on `MobileNetV2_1.14`.

3.2.1 Performance

Perforamnce is tested on which supports AVX-512 and FMA3. FMA (fused multiply–add) can speed up linear algebra computations such as dot-product, matrix multiplication, convolution, etc. Tenforflow 1.15 for tf-pose-estimation and Tensorflow 2.4 for PoseNet was built from source with bazel - opensorce build tool with -march=native for gcc which enables the instruction subsets supported by the local machine.

The performance of chosen models is summarized in Tables 3.8 and 3.9.

model	input image resolution	inference time	resulting FPS potential
CMU original	432x368 208x176	1s 0.5s	1 2
`MobileNetV1_0.75`	432x368 208x176	0.2s 0.04s	5 25
`MobileNetV2_1.4`	432x368 208x176	0.3s 0.05s	3 20
`MobileNetV2_0.5`	432x368 208x176	0.15s 0.04s	6 25
PoseNet moilenetV2 for hand estimation	257x257 160x160	0.013 0.02	76 50

Table 3.8: Performance on Intel(R)Core i7-1065G7 CPU.

model	input image resolution	inference time	resulting FPS potential
CMU original	432x368 208x176	0.43s 0.1s	2 10
`MobileNetV1_0.75`	432x368 208x176	0.12s 0.05s	8 20
`MobileNetV2_1.4`	432x368 208x176	0.15s 0.07s	6 14
`MobileNetV2_0.5`	432x368 208x176	0.1s 0.06s	10 17
PoseNet moilenetV2 for hand estimation	257x257 160x160	0.013 0.02	76 50

Table 3.9: Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz

Chapter 4

Conclusion

MobileNets enables the pose recognition to be performed without GPUs, on which most of approaches shown in papers rely. DTW showed good results (96%-100% on custom data) for recognising a small number of dissimilar gestures. New gestures could be added simply by adding an example of it, it does not require relearning after adding/removing the gesture. Also DTW is not computationally expensive.

However, adding more gestures could be a problem. For overlapping gestures, e.g. 'arm forward' and 'arm up', without explicitly declaring the dependency between gestures and special handling the approach will not work, 'arm forward' gesture will be recognized, leaving out the longer gesture. Also the problem of slow and noisy pose estimation from RGB images in dark environment remains.

Chapter 5

Attachments

`dataset.tar.gz` contains inferenced keypoints used for testing on custom dataset in section 3.1. Image data is included on CD.

Script `test_dtw.py` compares gestures using DTW method and plots the confusion matrix. It takes path to the data (e.g. `dataset/cmu`) as first argument. Add `-v` as second argument for verbose output.

Script `animate_gesture.py` animates gesture from dataset. Path to gesture shall be passed as first argument (e.g. `dataset/cmu/1_1_1.p`).

Folder `tf_gesture_recognition` contains code for gesture recognition, is uses tf-pose-estimetion [5] (which shall be installed onto the system with all its dependencies).

Folder `tf2_posenet` contains example of pose estimation using posenet, uses Tensorflow v2.4, to run execute main.py

Folder `hand_estimation` contains hand shape estimation (fist of palm), uses Tensorflow v2.4, to run execute main.py

Bibliography

[1] CAO, Z. et al. Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. In *CVPR*, 2017.

[2] Chen, C. – Jafari, R. – Kehtarnavaz, N. UTD-MHAD: A multimodal dataset for human action recognition utilizing a depth camera and a wearable inertial sensor. In *2015 IEEE International Conference on Image Processing (ICIP)*, s. 168–172, 2015. doi: 10.1109/ICIP.2015.7350781.

[3] DENG, J. et al. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[4] HOWARD, A. G. et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, 2017.

[5] Ildoo Kim. *tf-pose-estimation* [online]. 2020. [cit. 5. 2. 2021]. Available at: <https://github.com/ildoonet/tf-pose-estimation>.

[6] KUMAR, P. P. – VADAKKEPAT, P. – POH, L. A. The NUS hand posture datasets I, 2017.

[7] Köpüklü, O. et al. Real-time Hand Gesture Detection and Classification Using Convolutional Neural Networks. In *2019 14th IEEE International Conference on Automatic Face Gesture Recognition (FG 2019)*, s. 1–8, 2019. doi: 10.1109/FG.2019.8756576.

[8] PFISTER, T. – CHARLES, J. – ZISSERMAN, A. Flowing ConvNets for Human Pose Estimation in Videos, 2015.

[9] Sandler, M. et al. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, s. 4510–4520, 2018. doi: 10.1109/CVPR.2018.00474.

[10] SCHNEIDER, P. et al. Gesture Recognition in RGB Videos Using Human Body Keypoints and Dynamic Time Warping. In CHALUP, S. et al. (Ed.) *RoboCup 2019: Robot World Cup XXIII*, s. 281–293, Cham, 2019. Springer International Publishing. doi: 10.1007/978-3-030-35699-6_22. ISBN 978-3-030- 35699-6.

[11] TensorFlow. *MobileNet checkpoints* [online]. 2020. [cit. 5. 1. 2021]. Available at: <https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet>.

[12] TensorFlow. *Pose estimation* [online]. 2020. [cit. 5. 1. 2021]. Available at: <https://www.tensorflow.org/lite/models/pose_estimation/overview>.

[13] TensorFlow. *Transfer learning and fine-tuning* [online]. 2020. [cit. 5. 1. 2021]. Available at: <https://www.tensorflow.org/tutorials/images/transfer_learning>.