Bachelor Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Action recognition system

Anastasia Ostapenko

Supervisor: Mgr. Michal Vavrečka Ph.D
May 2022

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Ostapenko Anastasia**	Personal ID number:	**483746**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Specialisation:	**Artificial Intelligence and Computer Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Action Recognition System

Bachelor’s thesis title in Czech:

Systém pro rozpoznávání akcí

Guidelines:

The aim of the work is to create a system for action recognition, which is based on the integration of information from two separate modules. The first module recognizes objects and their positions and the second module recognizes and categorizes motion. Information from both modules is passed to a third module that can distinguish the type of action based on the objects in the scene (e.g. action „hammering“ requires object „hammer“). The system will be trained on its own dataset and tested and evaluated in real conditions. The system will be tested on the assembly task actions (hammering, screwing, wrenching etc.) and 20 types of objects (screwdriver, hammer, wrench, screw etc.)
1. Create and annotate an action dataset with assembly actions and objects.
2. Train a visual module for object recognition.
3. Train an action recognition module.
4. Train a module that integrates information from previous modules.
5. Test the system in real conditions.

Bibliography / sources:

[1] Bolya, D., Zhou, C., Xiao, F., & Lee, Y. J. (2019). Yolact: Real-time instance segmentation. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 9157-9166).
[2] Zhang, C., Zou, Y., Chen, G., & Gan, L. (2020). Pan: Towards fast action recognition via learning persistence of appearance. arXiv preprint arXiv:2008.03462.

Name and workplace of bachelor’s thesis supervisor:

Mgr. Michal Vavrečka, Ph.D. Robotic Perception CIIRC

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

Date of bachelor’s thesis assignment: **16.09.2021** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **19.02.2023**

Mgr. Michal Vavrečka, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce her 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

Acknowledgements

I would like to thank my supervisor Mgr. Michal Vavrečka, Ph.D for giving me the opportunity to work in the direction I wanted, and for all the useful advice and help he provided to me.

I am also grateful to my family, especially my mother, who has always supported and encouraged me. A special thanks goes to my friend Nikita Sokovnin, for all the helpful advice.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May , 2022

Abstract

This thesis contributes to the field of computer vision and focuses on the development of an action recognition system based on the integration of information from two separate modules. The first module is responsible for motion detection and categorisation. The second module is an instance segmentation module that recognises objects and their position in the scene. The information from both modules is passed to a classifier that makes the final prediction.

To train the action recognition module, we create our own dataset with eight action types that include assembly actions with tools and also corresponding "fake" actions that have similar motion but where no tools are used. The proposed classifier achieves 95.21% accuracy in this dataset compared to 85.52% for a baseline classifier. Therefore, we demonstrate that combining data from two different sources can improve the overall results of the action recognition task, especially when a small dataset is used.

Keywords: human action recognition, classification, object detection

Supervisor: Mgr. Michal Vavrečka Ph.D

Abstrakt

Tato práce přispívá do oblasti počítačového vidění a je zaměřena na vytvoření systému pro rozpoznávání akcí, který je založen na integraci informací ze dvou samostatných modulů. První modul dokáže rozpoznat a kategorizovat pohyb. Druhý modul rozpoznává objekty a jejich polohu na scéně. Informace z obou modulů je přeposlána navrženému klasifikátoru, který udělá konečnou predikci.

Pro trénování modulu zodpovědného za rozpoznávání akcí jsme vytvořili vlastní dataset který obsahuje osm typů akci. Část těchto akcí vyžaduje nástroj a část jsou odpovídající jim „falešné“ akce, které mají podobný pohyb, ale žádné nástroje se nepoužívají. Navrhovaný klasifikátor dosahuje v tomto datasetu přesnosti 95,21% ve srovnání s 85,52% u základního klasifikátoru. Takovým způsobem demonstrujeme, že kombinaci dat ze dvou různých zdrojů můžeme zlepšit celkové výsledky úlohy rozpoznávání akcí, zvláště v případech kdy je použita malá datová sada.

Klíčová slova: rozpoznávání akcí, klasifikace, detekce objektů

Překlad názvu: Systém pro rozpoznávání akcí

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation

Computer vision is the field of Artificial Intelligence that focuses on recognising and processing objects in visual scenes in the same way as humans do. Nowadays, interest in this field is increasing and several major advances have been made in object recognition in recent years. However, there are still areas that have significant potential but do not have as many research contributions as others. One of these is the area of action recognition, which goal is to recognise and identify human actions in videos automatically without manual operations.

Identification and classification of human actions is one of the most important tasks in video understanding. It also has many real-world applications such as surveillance, behavioural analysis, entertainment, content-based video retrieval and human-computer interaction. For example, if a patient is doing a rehabilitation exercise at home, it would be helpful if her/his robotic assistant could recognise and correct the patient's actions to avoid possible injuries [5]. But even though it seems to be a simple task for a human, action recognition is quite challenging for an artificial system.

Several systems for recognising actions in videos exist, but their capabilities are limited, especially when small datasets are used for training. Therefore, in this work we focus on improving the overall performance of one of these systems.

1.2 Goals

There are several goals of our work:

- Create a dataset containing assembly actions with tools (e.g. hammering, wrenching, etc.) and simple actions without tools (e.g. fake hammering, fake wrenching, etc.).
- Train the action recognition module on the created dataset with different hyperparameters.
- Train the visual module for object detection and extract key features.
- Propose a method to integrate the information from the previous modules.
- Conduct experiments to test the performance of the proposed approach.

1.3 Overview of the chapters

The text is structured as follows:

- In Chapter 2, we define the terms that are important for our work, such as object recognition, instance segmentation and action recognition. We also do research on these topics to provide a theoretical basis for our method.
- The formal definition of the task is given in Chapter 3, where we also discuss our own approach.
- Chapter 4 is devoted to the implementation process. We describe the creation of the dataset, the tools used and also provide details on the proposed method.
- In Chapter 5 we discuss the results achieved and test our solution.
- In the last Chapter 6 we summarise the results of our work and also discuss possible improvements.

Chapter 2

Related work

2.1 Action recognition

The goal of human action recognition is to label image sequences with action labels and extract the information about which human actions occur in videos. Even though action recognition has been an active area of research for more than four decades, the computer vision algorithms used fall far below human performance on this type of task for different reasons. One of these reasons was described in [1]. Most computer vision algorithms perform well on simple lab-recorded datasets, but still have problems recognizing actions in more realistic datasets that come from movies [2], TV shows [3] or web videos [4]. Another frequent problem is described in the next section.

2.1.1 How common action recognition algorithms work

The process of action recognition can be divided into two parts. The first is action localisation, which involves finding the correct location and timestamp in the video (when exactly the action is performed). And the second part is the classification of the action, where the correct label is assigned to a particular image or video.

In order to classify an action in a video, motion information should be modeled. In general, motion is represented by optical flow [6], [7]. The definition of optical flow was given in [7], and it states that "optical flow is the distribution of apparent velocities of movement of brightness patterns in an image." Most of the current action recognition methods compute the EPE loss (the average Euclidean distance between the estimated and the ground-truth flow) and then the estimated optical flow is transmitted to the action recognition module. This means a two-stage procedure should be performed. However, as it was mentioned in [8] and [9], such two-stage paradigms are storage demanding, time-consuming, and cannot be trained consistently.

To solve the above problems, it was recently proposed by the authors of [9] to use Persistence of Appearance (PA) for motion representation to lift the dependence on optical flow and achieve real-time speed. According to [9] and [10], PA derives from optical flow and focuses on the small shifts in motion boundaries, as these are the most important components for action recognition.

The comparison between the described methods (using optical flow or PA) is demonstrated on the Figure 2.1 from [10].

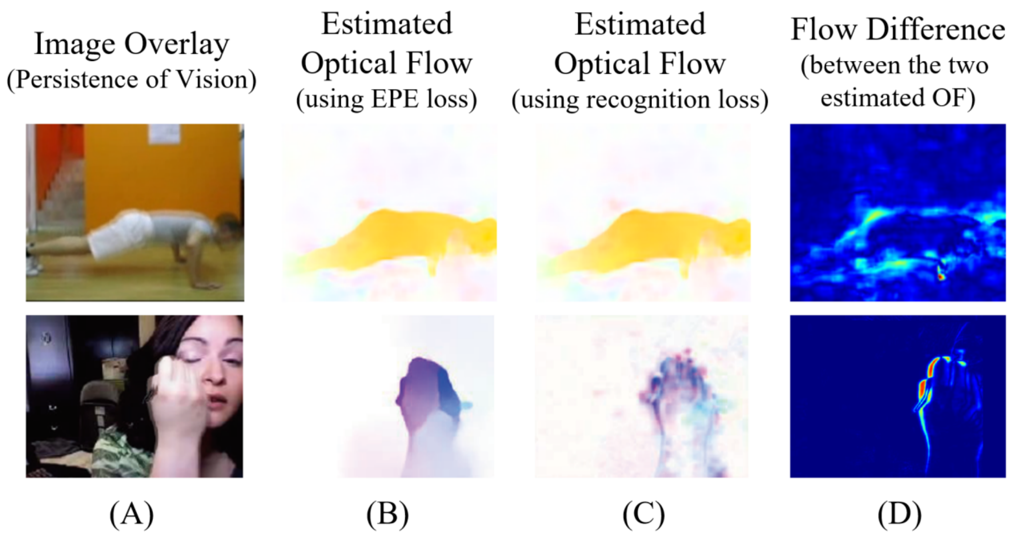
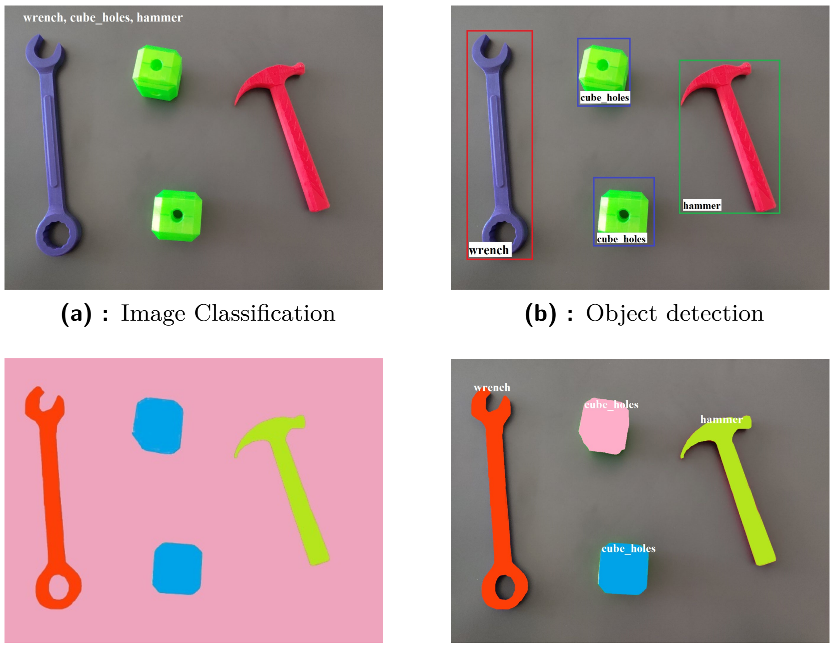


Figure 2.1: Comparison between estimated optical flows using Persistence of Vision (A), the EPE loss (B), the recognition loss (C), and their difference calculated by the Euclidean distance (D), from [10]

The authors of [9] also proved that using PA is more efficient and flexible than what most current action recognition methods suggest. They proposed a unified framework called Persistent Appearance Network (PAN) that achieves the state-of-the-art recognition performance even on challenging datasets from movies, YouTube and Google videos. And for the above reasons, we use the PAN module in our work.

2.2 Object detection and instance segmentation

We begin this section by defining such important computer vision concepts as object detection [11], semantic segmentation [12], and instance segmentation [13].



(a) : Image Classification **(b) :** Object detection

(c) : Semantic segmentation **(d) :** Instance segmentation

Figure 2.2: Comparison of different visual recognition tasks in computer vision

Object detection [11] is the process of detecting instances of a certain class (car, laptop, human) in images and videos and labeling them with the associated class. Also, the location of the object is predicted by assigning a bounding box or centroids and a confidence score.

Semantic segmentation [12] is proposed to label each pixel of an image with a corresponding class without separating instances of the same class. The goal of this process is to provide a better understanding of the image.

Finally, **instance segmentation** [13] is a combination of object detection and semantic segmentation. It aims to separate all objects from the background image and assign each of them a separate categorical pixel-level mask. Moreover, instance segmentation can be viewed as a special kind of object detection. But instead of localising an object by a bounding box, pixel-level localisation is used.

2.2.1 Early object detection methods

There are different approaches to object detection and instance segmentation. In [11] it was pointed out that the first object detection algorithms (before deep learning) consisted of three steps: proposal generation, feature vector extraction, and region classification. Sliding windows [14] were used to suggest the regions where objects might have been located. A sliding window is a rectangular box of fixed width and height that slides around an image and classifies each image crop, as to whether or not it contains an object of interest. To understand if there is such an object a feature vector should be obtained from the sliding window. The feature vector is then encoded by low-level visual descriptors such as SIFT (Scale Invariant Feature Transform), HOG (Histogram of Gradients) or SURF (Speeded Up Robust Features). And finally, region classifiers, such as SVM (Support Vector Machines) [15] are used to learn how to assign categorical labels to the covered regions.

2.2.2 Current object detection methods

With the development of DCNN (Deep Convolutional Neural Networks) [16], there was a significant advance in many pattern recognition tasks, such as image classification and video classification. Current detector frameworks based on deep learning can be divided into two categories: one-stage detectors and two-stage detectors.

Two-stage detectors include, for example, Fast R-CNN (Region-based Convolutional Neural Networks) [18], Faster R-CNN [19] and Mask R-CNN [20]. In the first step, such detectors find regions of interest (ROI) in the image that have a high probability of containing an object. Then, the proposed regions are fed into the R-CNN. Classification score and spatial offsets of the region are obtained and the category of the region is predicted.

One-stage object detectors such as YOLO (You Only Look Once) [21] or SSD (Single Shot Detector) [22] make the categorical prediction of objects at each location of the feature maps directly, without the region classification step [11]. In other words, there is no intermediate task that should be performed before producing an output. As a result, one-stage detectors have a simpler and faster model architecture and can achieve real-time speeds, but the accuracy of predictions also decreases. However such detectors as YOLO or SSD are able to fill the gap in performance in some other ways (e.g., strong data augmentation, anchor clustering, etc.)

A more advanced version of the YOLO detector is YOLACT (You Only Look At Coefficients) which we will use in our work. YOLACT [23] (also YOLACT++ [24]) is a state of the art, real-time object segmentation algorithm, that can perform object detection and segmentation with high accuracy and is much faster than two-stage detectors. YOLACT omits the localisation step and divides the instance segmentation process into two parallel tasks: generating a set of prototype masks and predicting mask coefficients per instance. Then, according to [23] the prototypes for each instance are linearly combined with the corresponding predicted coefficients and then cropped using a predicted bounding box. The prediction of prototype masks is critical to ensure high resolution of the final instance masks. The prototype masks depend only on the input images and are independent of categories and specific instances.

2.3 Multimodal integration

Neural networks [25] [26] have dramatically improved performance beyond the state of the art in the fields of automated driving, medical research, industrial automation, etc. compared to traditional machine learning techniques, such as Naive Bayes or Support Vector Machine. They are widely used to solve problems such as classification, prediction, optimisation, pattern recognition and function approximation.

According to [27], the reason for such success is that conventional computers use an algorithmic approach to problem solving and it restricts us to solving only those problems we already know how to solve. Neural networks, on the other hand, process information in a similar way to the human brain and can also solve more advanced tasks.

In [31] a modular method for combining neural networks was described. The main problem is decomposed in such a way that there are different modular components that perform different subtasks. Therefore, each module is trained to be a specialist for a particular task and then they are integrated into one module. The task decomposition can be done explicitly or automatically. The advantages of this modular approach are: efficiency, as several independent neural networks can be trained simultaneously, reduction of model complexity and better performance.

Chapter 3

Methods

3.1 Problem definition

We start this chapter with formalising our task and describing it in more detail.

Let:

- $\mathbf{K} = \{1, 2, \ldots N\}$ be a set of labels (action types) with N labels
- $\mathbf{X} = \{(x_0, y_0)\ , (x_1, y_1) \ldots (x_m, y_m)\}$ be a dataset with m observations, where $x_i \in \mathbb{R}^p$ is a video sample and $y_i \in \mathbf{K}$ is a corresponding label
- $\mathbf{D} = \{d_0\ , d_1, \ldots d_m\ \}$, $d_1 \in \mathbf{K}$ be a set of decisions
- $\mathbf{R}$: X $\times$ D $\rightarrow \mathbb{R}$ be a reward function, where

$$R((x_i, y_i), d_i) = \begin{cases} 1, & \text{if } y_i == d_i \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

  Therefore, $\mathbf{R}$ represents the reward obtained when the true label of the video is y_i and the decision made is d_i.

The task is to assign decision d_i to each video x_i from $\mathbf{X}$ in such a way that the overall ratio of correctly classified videos is maximised for the entire dataset.

$$r = 100 * \frac{1}{m} \sum_{i=0}^{m} R(x_i, d_i) \tag{3.2}$$

3.2 Action types

For our work we decided to use eight assembly actions: hammering, fake hammering, pliering, fake pliering, wrenching (wrenching the nut with a wrench), fake wrenching, hand-screwing (wrenching the nut by hand), fake hand-screwing. All action types are shown in the Figure 3.1.

The difference between "normal" and "fake" actions is that fake actions do not require a tool. However, the motion of these two types of actions is very similar, which is confusing for ordinary action recognition systems because they are designed to recognise the action motion and not the objects around. Here it especially helps to add a second module that is responsible for object detection.

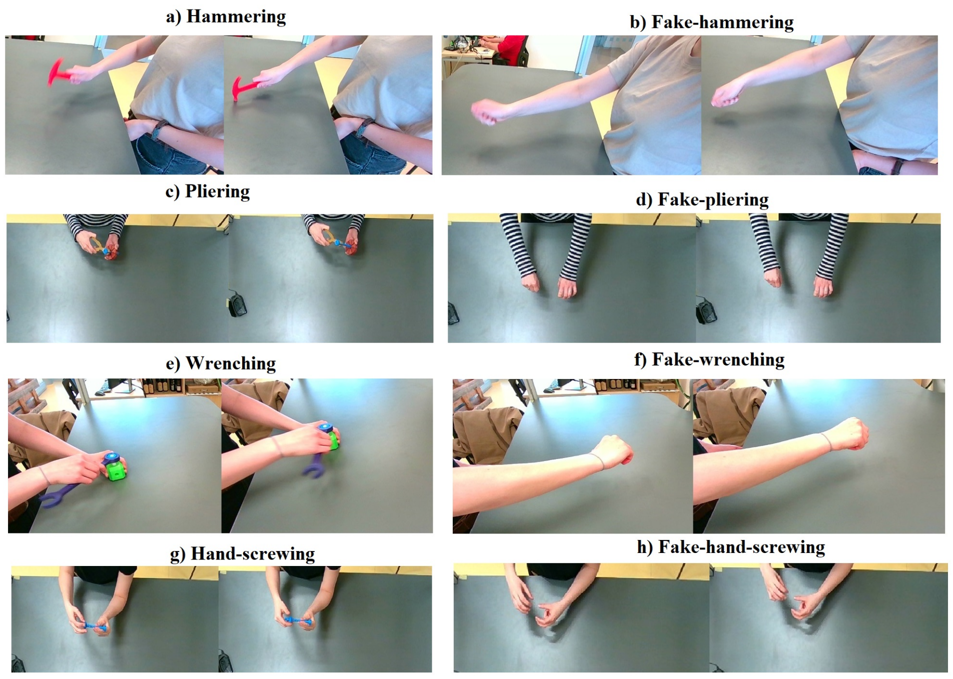


Figure 3.1: Comparison between actions with tools and fake actions

3.3 Suggested method

The main idea of our approach is based on the above fact. We assume that associating an action with a tool can increase the overall accuracy in cases where the action type is not well identifiable, e.g., hammering and false hammering or wrenching and hand-screwing.

To test this assumption, we propose the following method.

First, the YOLACT module is trained on the dataset containing images of all the tools needed for the selected actions, e.g. hammer, wafer etc. Then, video frames from the action dataset are fed into the trained YOLACT network. As an output of this step, we obtain the labels of all objects present in a single video, their centroids and scores (probabilities with which the objects were detected). This data is analysed again and, according to a certain algorithm, an object is selected to represent the action. In addition, the average distance that the object travels between two frames is calculated.

$$dist = \frac{\sum_0^{N-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}}{N-1}$$

Here (x_i, y_i) are centroids of the object in frame i, and (x_{i+1}, y_{i+1}) are centroids of the object in frame i+1, N is the number of frames in the video.

The PAN module is also trained with the same action dataset. Each video is separated to image frames. Then the trained model is applied to the dataset. As a result, we get the predicted action and the probability of this prediction.

The data from both modules are combined and stored as a new dataset, which is then used to train the proposed classifier.

3.3.1 Multimodal integrator

The final and most important part of our algorithm is a classifier that takes all the preprocessed data as input and produces the final classification as output, Figure 3.4. For this step, we have chosen to use a neural network consisting of several layers (input, hidden and output) that convert an input vector into an output. Each layer consists of neurons, and neurons in different layers are connected with each other. Each of these connections has an associated weight that represents the importance of this relationship between the units, and it also has a bias, Figure 3.2.

Training the neural network (well described in [29]) is the process by which the neural network learns the values of the parameters - weights and biases. It consists of two stages: forward propagation and backward propagation. During forward propagation the input values are fed into the neural network and the output or predicted value is received. In general, each unit in the layer takes an input, applies an activation function to it, and then passes the output on to the next layer. The activation function here is a non-linear function added to help the network learn complex patterns in the data. Some of the most commonly used activation functions are Sigmoid 3.3 and ReLU 3.4.

$$sigm(VN) = \frac{1}{1+e^{-VN}} \tag{3.3}$$

$$ReLU(VN) = max(0, VN) \tag{3.4}$$

How neural networks learn

To explain the learning process, let us assume that our neural network has a similar structure to the one in the Figure 3.2.

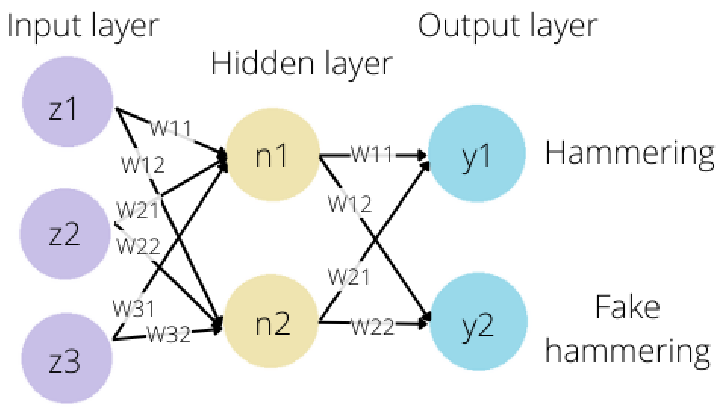


Figure 3.2: Example of a neural network architecture

Then the value transmitted from the input layer to the unit j in the hidden layer during forward propagation can be expressed as:

$$n_j = \sum_i z_i w_{ij} + b_j \tag{3.5}$$

Let's also assume the ReLU activation function is used as it is less susceptible to the vanishing gradients problem [28].

$$ReLU(n) = max(0, n) \tag{3.6}$$

After that, the values should be forwarded to the output layer. This is done in a similar way:

$$y_j = \sum_i n_i w_{ij} + b_j \tag{3.7}$$

And then the output with the higher value is chosen using the *argmax* function:

$$output = argmax(y) \tag{3.8}$$

Since in our problem we also need to get the probability value of the predicted result, we apply the softmax function at the end, which normalises the output vector into a probability distribution.

$$softmax(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \tag{3.9}$$

After the result of the forward propagation step is obtained, we need to calculate the error, that indicates the difference between the expected and the obtained outcome. To calculate the error, the loss function is used. As it was pointed out in [30], cross-entropy is usually preferred while solving classification problems. Therefore:

$$E = L(\hat{y}, y) = -\sum_{i=1}^{K} y_i^{(k)} log\, \hat{y_i}^{(k)} \tag{3.10}$$

Here, $y^{(k)}$ is 0 or 1, indicating whether class label k is the correct classification.

The next step is back propagation. It aims to minimise the cost function by adjusting the weights and biases of the network. For this purpose, the gradient of the error value with respect to these parameters is calculated. And then each weight is updated by an amount proportional to the partial derivative of E with respect to the weight.

$$\begin{aligned} w_{ij} &= w_{ij} - \alpha \frac{\partial E}{\partial w_{ij}} \\ b_j &= b_j - \alpha \frac{\partial E}{\partial b_j} \end{aligned} \tag{3.11}$$

Where α is the learning rate.

The idea behind this is that the gradient indicates the slope of a function at a certain point. And it always points in the direction in which the value of the loss function increases. Therefore, the values of the parameters should be updated in the opposite direction to the direction indicated by the gradient. Also illustrated in Figure 3.3 [1].

[1]Inspired by `https://towardsdatascience.com/learning-process-of-a-deep-neural-network-5a9768d7a651`

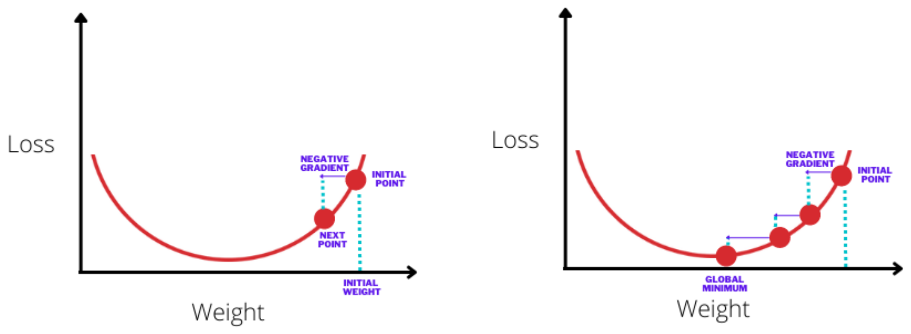


Figure 3.3: Adjusting weights

This process is repeated until the required number of iterations or the target error is reached.

The whole process of video processing and producing the result is shown in Figure 3.4. It does not include the training steps. Implementation details, such as the training process, data collection, feature extraction, etc., are discussed in the next chapter.

3.3.2 Hyperparameters and optimisation methods

There are various parameters and optimisation methods that can influence the learning process and it is important to understand them in order to achieve better results. A hyperparameter [35] is a parameter that cannot be estimated directly from data learning (such as weights or bias) and must be set before training a model. It is important to choose hyperparameters wisely as their values control the learning process.

Learning rate

The learning rate is one of the most important hyperparameters in Machine Learning, that determines how much the weights and bias of the network should change at each iteration 5.1. The learning rate controls the speed at which the model learns. Thus, a high learning rate speeds up the learning process, but can also cause the model to converge too quickly to a suboptimal solution. On the other hand, a small learning rate converges smoothly but can lead to a long training process that could get stuck. With the right learning rate, the objective function should be able to converge to a global minimum in an acceptable amount of time.

Number of epochs

An epoch in machine learning is defined as one complete pass of the training dataset through the algorithm. After one epoch, the model updates its parameters. The number of epochs depends on the dataset. If the model is not trained long enough it will be underfitted and will not be able to capture the underlying trend of the data. However, if the number of epochs is too high, the opposite problem arises. The model is then overfitted, i.e. it learns patterns that are highly specific to the sample data. This leads to high accuracy in the training dataset but low accuracy in the validation dataset. Therefore, the number of epochs should be adjusted by slowly increasing its value until validation accuracy starts to decrease.

Mini-batch size

The mini-batch size represents the number of processed samples from the training dataset used to estimate the error gradient. While training PAN classifier we always used batch size 2 due to limitations on the GPU side.

Dropout rate

Dropout is a regularisation method for neural network models in which randomly selected neurons are removed from training. The method is used to prevent overfitting. A good value for dropout in a hidden layer is between 0.5 and 0.8, where 0.0 means the layer is completely excluded from training and 1.0 means no dropout.

Optimisation methods

An optimizer [36] is a function or algorithm that changes the neural network parameters and learning rate to reduce loss. Some of the most popular optimizers are SGD (Stochastic Gradient Descent) and Adam (Adaptive Moment Estimation). The idea behind SGD was explained earlier in Section 3.3.1. Adam is a combination of the two SGD extensions Root Mean Square Propagation and Adaptive Gradient Algorithm. It calculates individual adaptive learning rates for different parameters. Adam is faster than SGD, but should be fine-tuned, otherwise convergence problems may occur.

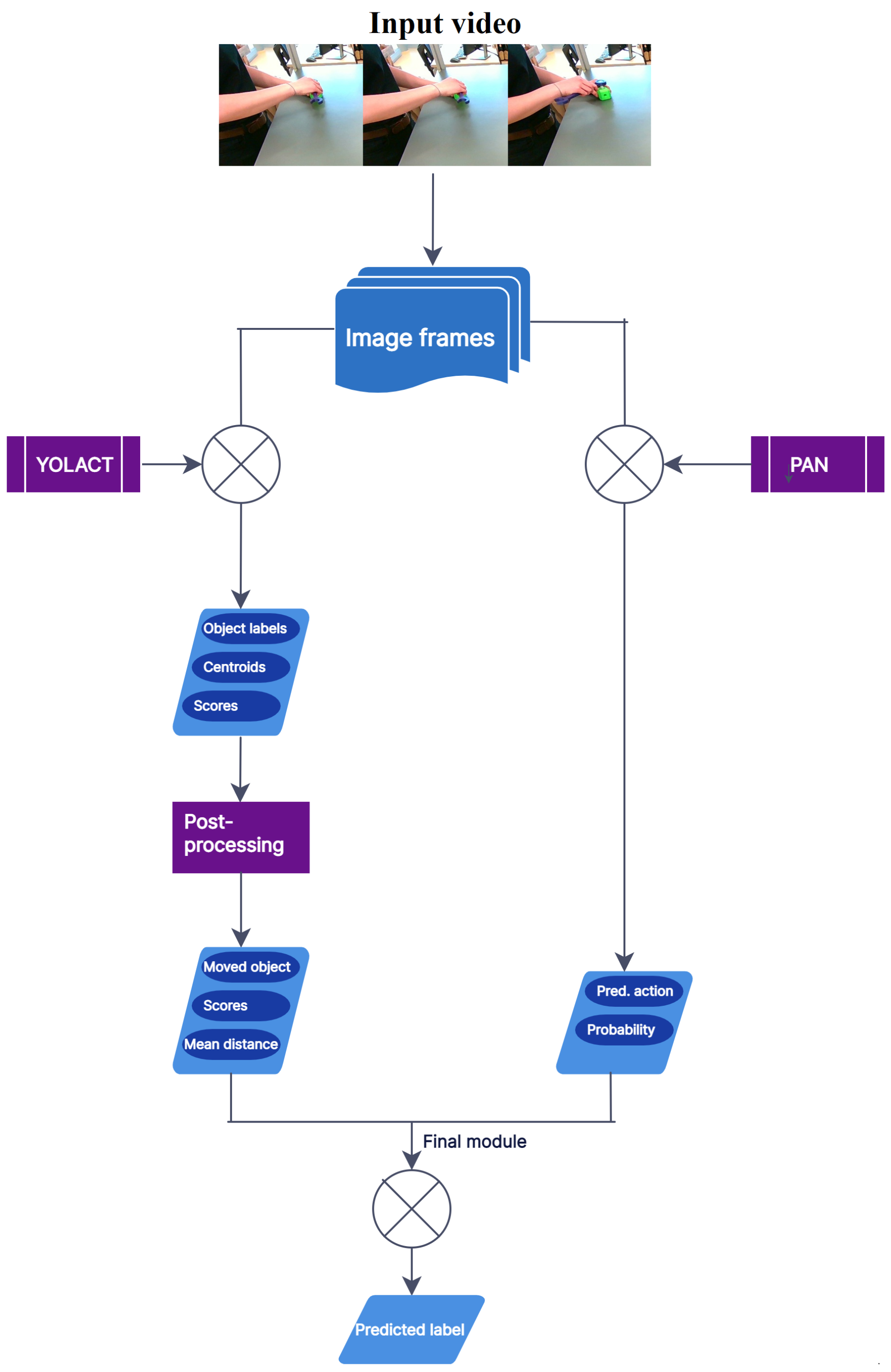


Figure 3.4: Flowchart of the action recognition process according to our approach

Chapter 4

Implementation

The solution is implemented in the Python 3.9 programming language. We used an open-source machine learning framework based on the Torch library - PyTorch [1] for the development of the learning algorithm. Python packages seaborn [2] and matplotlib [3] were also used for visualisation of the metrics.

4.1 Basic action recognition with PAN

Based on the information described in Chapter 2 we have chosen PAN algorithm for action recognition. The reason for this is that PAN is much faster than other action recognition methods and in theory can reach 8000 frames per second.

4.1.1 Dataset preparation for PAN

The dataset was collected using the Intel RealSense D435 cameras. We aimed to increase variability and reduce the overall recording time, so five cameras were used at the same time. They recorded each action from five different angles.

The dataset contains eight types of actions: hammering, fake hammering, pliering, fake pliering, wrenching, fake wrenching, hand-screwing, fake hand-screwing. And each type of action is represented in the dataset with about 200 videos. The recorded video is then processed and splitted into single frames. Also, each video has about 30 frames, but 3 is the minimum number of frames a video is allowed to have in order to be processed correctly

[1]Available at `https://pytorch.org/`

[2]For further references see `https://seaborn.pydata.org/`

[3]For further references see `https://matplotlib.org/`

by PAN. Therefore, videos that do not have enough frames or have a low quality (e.g. camera did not capture a tool) are filtered.

4.1.2 Training PAN and predicting actions in videos

The training of the module PAN was done according to the tutorial on the Github page of the project. The main module is responsible for the training. As input it takes the csv file representing the dataset. This file has three columns: video path, number of frames in the video and the true action. As output we get another csv file with the video path, the number of frames, the predicted action and the probability of the prediction.

4.2 Detecting tools with YOLACT

The first step of the whole process is to train YOLACT on the dataset of objects that will later be used in the action video dataset. Then the trained model is used to recognise tools in that video dataset.

4.2.1 Dataset preparation for YOLACT

CROW dataset is used in our work. It was created by a team of researchers from the Czech Institute of Informatics, Robotics and Cybernetics and consists of high quality base objects shown in the Figure 4.1. The objects were designed to be used in modular assembly and have a unique visual appearance to prevent misclassification. Also, each object has a fixed color, however the models have been trained without the texture to remain invariant to the exact colors of the objects.

Creating an image dataset of fourteen different classes with sufficient variability requires to take thousands of photos of different objects and label them manually. This approach is highly inefficient, so the toolkit myGym [34] was used instead. myGym is a modular framework for developing and comparing reinforcement learning algorithms and imitation learning tasks trained in a 3D simulator. The toolkit allows to generate synthetic datasets that can be used for YOLACT. The dataset generator uses PyBullet [4] to construct photo-realistic scenes, with rich annotations.

[4]See `https://pybullet.org/wordpress/`

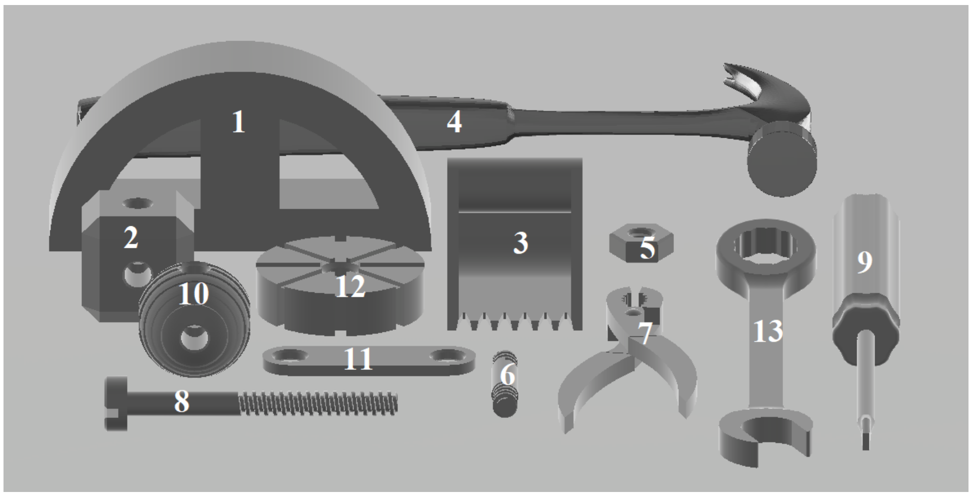


Figure 4.1: `1.car_roof, 2.cube_holes, 3.ex_bucket, 4.hammer, 5.nut, 6.peg_screw, 7.pliers, 8.screw_round, 9.screwdriver, 10.sphere_-holes, 11.wafer, 12.wheel, 13.wrench; modified figure from [33]`

The final dataset consisted of 36k training and 4k validation images with segmentation masks.

4.2.2 Training YOLACT

As mentioned in chapter 2, YOLACT [5] is a one-stage instance segmentation model that generates a prototype mask and predicts the mask coefficients for each instance based on the entire image. The fact that YOLACT belongs to one-stage models makes the process of object detection fast, and yet the accuracy remains high. In our work, the speed at which objects are detected is of great importance, since even a small video dataset contains a large number of frames.

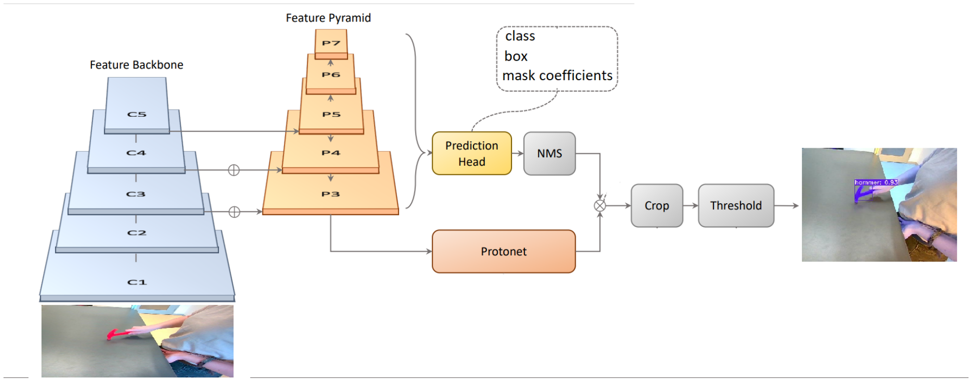


Figure 4.2: YOLACT Architecture based on [23]

[5]The code is available at `https://github.com/dbolya/yolact`

Hardware requirements

The computer on which the training takes place must be equipped with a CUDA [6] capable GPU (graphics processing unit). This is important because GPUs can perform calculations simultaneously, increasing the speed of computations and therefore speed computer vision applications. This fact was also pointed out by the authors of [32].

YOLACT was trained for 24 epochs and mean average precision (mAP) for box is 78.78% and for mask - 67.47%.

4.2.3 Detecting objects in videos

Once the YOLACT module is trained, we need to feed it the action dataset to recognise objects in videos. The objects we used for the dataset were 3D printed and therefore have the same visual appearance as the objects YOLACT was trained on. The process of labelling a video is as follows:

```
movements = {};
frames = get_video_frames();
for frame in frames do
    detected_objects, centroids, scores = apply_yolact(frame);
    for o, c, s in zip(objects, centroids, scores) do
        if o not in movements then
            init_instance(movements, o, c, s);
        else
            all_instances = get_instances(movements, o);
            for instance in all_instances do
                old_c = get_old_centroids(movements, instance);
                distance = get_distance(old_c, c);
                if distance < threshold then
                    update_instance(movements, o, instance, c, s);
                else
                    add_new_instance(movements, o, instance, c, s);
                end
            end
        end
    end
end
```

Algorithm 1: Labeling a video

[6]For further references see `https://developer.nvidia.com/cuda-toolkit`

Using the algorithm, we try to recognise all the objects that appear in the video. And we also keep track of the different instances, because there can be several hammers in one video or part of the environment can be misclassified with one of the objects. At the end, the movements dictionary contains all the objects that were recognised by YOLACT, probabilities with which they were detected and the distance they moved on average between two adjacent frames.

4.3 Combining outputs from YOLACT and PAN

This is the last stage of preprocessing the information from the videos for the multimodal integrator. The steps for each video are as follows:

1. Using *movements* from the algorithm 1, select the object (if any was detected) with the largest moved distance. Only objects associated with specific actions should be analysed.
2. Get PAN action prediction and probability.
3. Write the received information in the file as: [video_number, pan_prediction, pan_probability, yolact_prediction, yolact_probability, mean_moved_distance].

4.4 Multimodal integrator and advanced action recognition

The multimodal integrator consists of two main parts. The first one reads the csv dataset and loads it to the code. And the second one is a neural network that is trained and tested on this dataset.

4.4.1 Dataset for the multimodal integrator

The dataset for the multimodal integrator is prepared during the previous steps. It is a csv file with six columns: video id, action prediction made by PAN, probability of this prediction, the object detected by YOLACT, probability of this detection, the distance it moved on average between two adjacent frames. The *CSVDataset* class is responsible for processing the csv file and loading it into the code as a python dictionary. The neural network is later trained on this data. Video ids are removed during preprocessing because they are unique and should not be learned by a neural network.

4.5 Structure of the multimodal integrator

The neural network for the multimodal integrator was implemented using the pytorch framework. It has a similar structure to the one shown in Figure 3.2. Our neural network has an input layer, two hidden layers and an output layer. The size of the input sample in the first layer is five, since we have five features for each video. The activation function used is ReLU.

We have also created a separate module that does training or testing depending on the requirements. As input for training it takes:

- Path to the training and validation data and path to the true values
- Number of classes
- Name of the output model

As output we get the best recall achieved during the training process for both training and validation data.

For testing, the input parameters are similar. But as output we calculate the accuracy, recall and precision for the whole dataset and also for each class individually. These metrics are discussed in the next chapter.

Chapter 5

Results

In this chapter we summarise the results achieved using the method proposed in Chapter 3 and implemented according to Chapter 4. The evaluation is done using the data presented in the next section. We also compare the results of the base classifier and the results of the multimodal integrator.

5.0.1 Dataset

We trained and evaluated our baseline and proposed modules on two datasets. A larger dataset contains eight types of actions and is described in Section 4.1.1. A smaller dataset is a part of the larger dataset from which we excluded two groups of actions that gave the best and the worst results. Therefore, in the resulting smaller dataset we have four action labels: pliering, fake pliering, wrenching, fake wrenching. We used two different datasets to observe how the number of different actions in the dataset affects the results of the classification.

5.1 Training and test results

5.1.1 Evaluation Metrics

Before we discuss the results, we need to explain the metrics we used to evaluate our model. For the following examples, we consider 'hammering' as positive prediction and everything else as negative predictions.

The metrics used to evaluate the classifiers are:

- True Positive (TP) - is an outcome when the model makes a correct positive prediction. Example: true label is 'hammering' and the prediction of the model is 'hammering'.

- False Positive (FP) - is the case when the model makes an incorrect positive prediction. Example: true label is 'fake hammering' and model's prediction is 'hammering'.
- False Negative (FN) - is a result when the model incorrectly predicts the negative class. Example: true label is 'hammering' and model's prediction is 'fake hammering'.
- True Negative (TN) - is the case when the model correctly predicts the negative class. Example: true label is 'fake hammering' and model's prediction is 'fake hammering'.

From these we compute precision and recall as:

$$\begin{aligned} precision &= \frac{TP}{TP + FP} \\ recall &= \frac{TP}{TP + FN} \end{aligned} \tag{5.1}$$

So precision measures the number of correctly identified positive cases out of all predicted positive cases. And recall is a measure of the correctly identified positive cases out of all actual positive cases.

Finally, we can compute accuracy as:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.2}$$

Accuracy is one of the most popular metrics in classification tasks. However, we do not consider accuracy to be very representative in our case. It can reach high values even if the number of positive predictions is low, because negative classes are always predicted.
We also use confusion matrices to better visualize the numeric results.

5.1.2 Training and evaluation of the baseline action recognition method

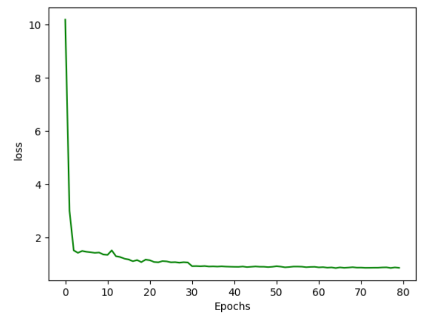
We consider the PAN module as our baseline solution. First, we trained the PAN classifier with the hyperparameters given in [9]. Initial learning rate 0.01, number of epochs 80 and dropout 0.5. The optimisation method used is an SGD algorithm with a weight decay 1e-4 and a mini-batch size of 2. ImageNet [1] pre-trained weights are used for initialisation. In the original implementation, recall is used instead of accuracy to track progress, so we decided to use this metric as well.

[1]Details here `https://image-net.org/`

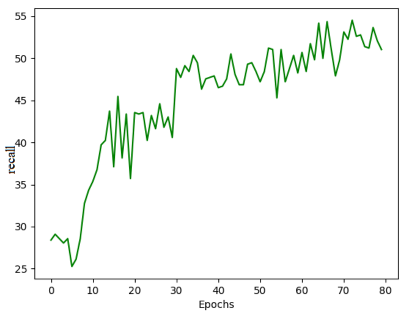
	4 classes	8 classes
Recall on training data	55.5%	37.82%
Recall on test data	55.0%	37.08%
Training time [h]	3.7	7.4
Best epoch	80	65

Table 5.1: Results obtained with PAN, trained for 80 epochs, with a learning rate of 0.01

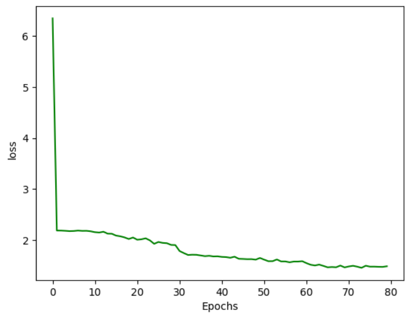
Figure 5.1 visualises the process of learning with the hyperparameters described above. It can be seen that PAN needs about 2 times more epochs for a larger dataset until the loss function and the recall value stabilise.



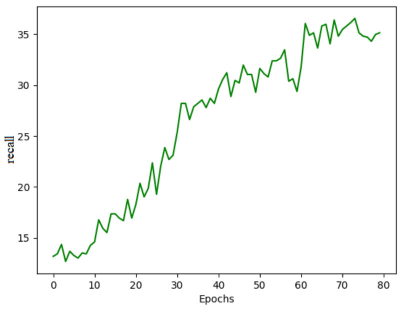
(a) : Loss, 4 classes dataset



(b) : Recall, 4 classes dataset



(c) : Loss, 8 classes dataset



(d) : Recall, 8 classes dataset

Figure 5.1: Loss and recall progress during training PAN on both datasets for 80 epochs, with a learning rate of 0.01

We have also experimented with the number of epochs and the learning rate. The best result is obtained when PAN is trained for 120 epochs with a learning rate of 0.01. The recall value on the test data increased by 5% for both datasets in comparison with the standard hyperparametres described earlier. However, it can be concluded that changing these hyperparameters does not drastically improve the performance of our baseline classifier.

We then used the best performing model to evaluate the results for each class separately. The results are presented in Table 5.2 and Table 5.3.

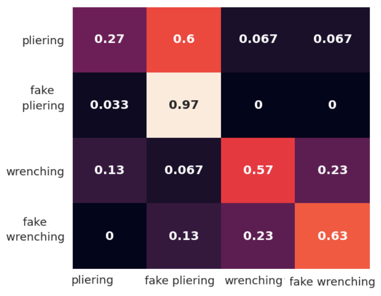
	Accuracy [%]	Recall [%]	Precision [%]
Pliering	77.5	26.67	61.54
Fake pliering	79.17	96.67	54.72
Wrenching	81.67	56.67	65.38
Fake wrenching	83.33	63.33	67.86
Total	80.41	60.83	62.38

Table 5.2: Metrics calculated for the best performing model PAN for the 4-class validation dataset

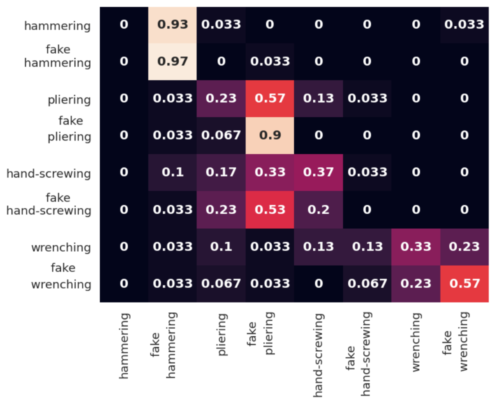
	Accuracy [%]	Recall [%]	Precision [%]
Hammering	87.50	0.0	N/A
Fake hammering	84.58	96.67	44.63
Pliering	82.08	23.33	25.93
Fake pliering	79.58	90.0	36.99
Hand-screwing	86.25	36.67	44.0
Fake hand-screwing	84.17	0.0	0.0
Wrenching	88.75	33.33	58.82
Fake wrenching	91.25	56.67	68.0
Total	85.52	42.08	39.76

Table 5.3: Metrics calculated for the best performing model PAN for the 8-class validation dataset

Finally, we calculated confusion matrices for both datasets, to visualise the results. Several things can be observed from them. First, it is interesting to note that overall, fake actions are better detected than actions requiring a tool. This could be due to the fact that there are more fake actions in the datasets. However, the difference between the number of fake actions and actions with a tool is less than 10%. Secondly, actions with tools are often confused with the corresponding fake actions, which is reasonable since their motion is similar.



(a) : 4 classes



(b) : 8 classes

Figure 5.2: Confusion matrices evaluated on the validation datasets using the best performing PAN model

5.2 Detecting objects with YOLACT

In this section we use the already trained YOLACT model to annotate videos in which actions are performed with tools. For each video, YOLACT detects objects and stores the average distance they moved in the json file, as described in Section 4.2.3. The entire evaluation was carried out with the 8-class dataset only, as the size of the dataset in this case has no influence on the recall for each individual class. At this stage we were interested in two things. First, how much each class of object moves, because tools naturally move different distances for different types of actions. And secondly, how well the different objects are recognised. This data is important because it directly affects the results of the multimodal integrator.

We used a boxplot to visualise the distance each object class moves on average between two image frames (Figure 5.3). The boxplot was created from the entire action dataset. We included only the objects relevant to the actions classification to make the plots clearer. However, if plotting all the objects it would be noticed that the graph includes distances for objects that never appeared in the dataset, such as wafer, ex_bucket or screwdriver. This happens because of misclassification. For example, a hammer is misclassified as a screwdriver and a wrench is misclassified as a wafer because they have similar shape. It can also be noticed from the boxplot that all tools reasonably have different distances because they are used for different actions. The cube_-holes, for example, has a small average range of movement, since it is used in wrenching and remains in the same place. The hammer, however, has a large range because the movement can be done in slightly different ways (e.g. moving arms wildly or hammering a small object into the same place).

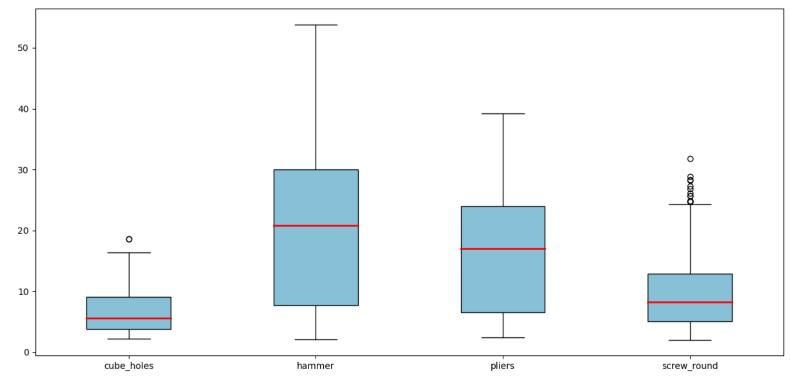


Figure 5.3: Boxplot comparison of the moved distance of different object classes

We also created a histogram showing the number of detected objects of the different classes in the entire dataset (Figure 5.4). We have limited this to only one class instance per video. Thus, if the algorithm detected two different hammers in a video due to misclassification, they are counted as one in the plot. It should also be noted that our dataset contains about 200 videos per action and we did not use the same tools for different actions. Ideally, the bar in the graph should be around 200 high for each object type used in the dataset. In our case, however, this is true only for the hammer and pliers. Other classes are represented less or, on the contrary significantly more. The wheel class, for example, was recognised over 800 in the dataset and can be considered an outlier in our case because of its high value. This is again due to a misclassification, because some parts of the working environment were wrongly classified as the class wheel. For this reason we have not used such object classes as wheel or wrench to "represent" the action. In the case of the action "wrenching", for example, we have chosen the object cube_holes to represent it, because cube_holes is better recognised by YOLACT than wrench.

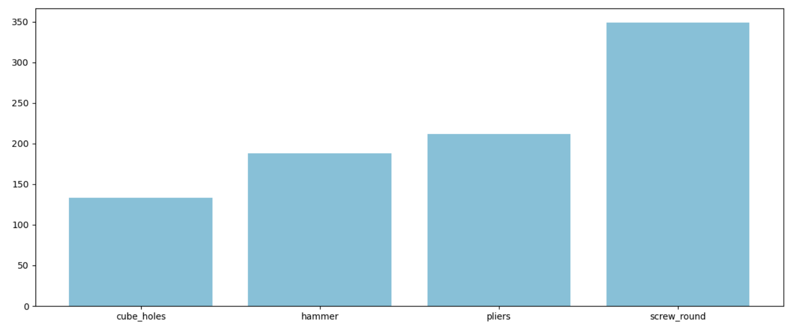


Figure 5.4: Histogram by class of the number of objects and videos containing at least one object of the corresponding class

Last but not least, we evaluated YOLACT's predictions using the same metrics we used to evaluate PAN. This time, however, we take a different approach. The YOLACT module can only predict five labels for our 8-class dataset because half of the actions are "fake" and the object is not used. Therefore, in this case, the YOLACT module classifies the video as "hammering" if it detected a hammer, as "pliering" if it detected pliers in the scene, as "hand-screwing" if it detected screw_round, as "wrenching" if there was cube_-holes in the video and as "fake" if none of these objects were present in the scene.

The results of the evaluation are shown in Table 5.4 and are also illustrated by the confusion matrix in Figure 5.5.

	Accuracy [%]	Recall [%]	Precision [%]
Hammering	97.5	100.0	83.33
Pliering	88.33	46.67	53.85
Hand-screwing	91.25	90.0	60.0
Wrenching	96.67	73.33	100.0
Fake actions	90.42	86.67	93.69
Total	92.83	79.33	78.17

Table 5.4: Metrics calculated with YOLACT for the 8-class validation dataset, where all fake actions are grouped into a single category

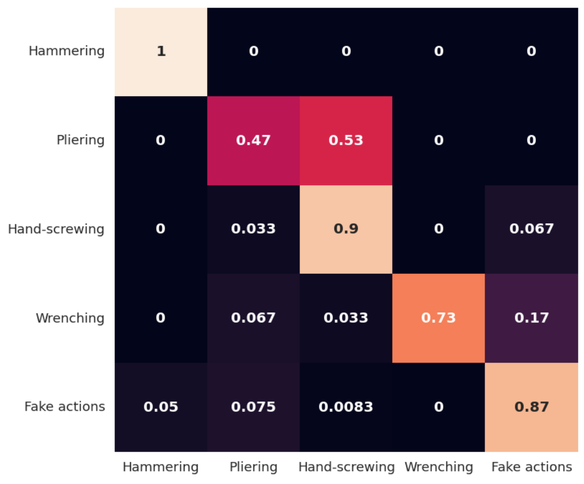


Figure 5.5: Confusion matrix evaluated with YOLACT on the the 8-class validation dataset, where all fake actions are grouped into a single category

5.3 Training and evaluation of the proposed method

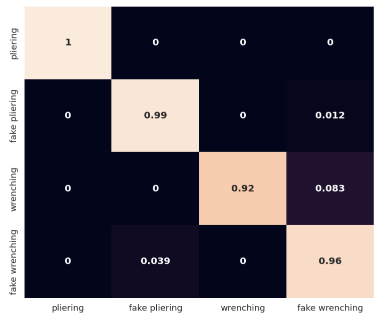
As the last step of our solution, we trained and tested the multimodal integrator on both datasets (four classes and eight classes). The structure of the integrator was described in Section 4.5, however, we experimented with the number and type of hidden layers, the type of loss function and optimizers. The highest recall in training and testing results is obtained when our neural network has two linear hidden layers, abd when CrossEntropy loss and Adam optimizer are used. The best hyperparameters found are: number of epochs - 300, learning rate - 0.002, batch size - 8. The results of this training session can be found in Table 5.5.

	4 classes	8 classes
Recall on training data	97.21%	85.57%
Recall on test data	89.17%	80.83%
Best epoch	272	268

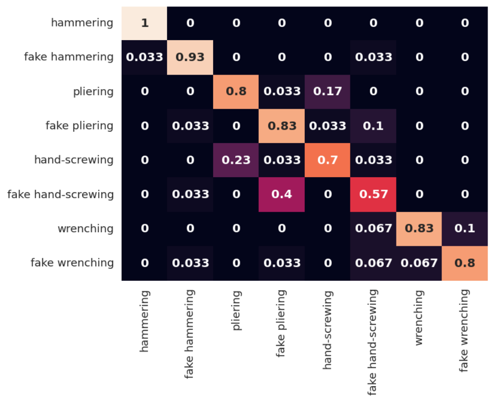
Table 5.5: Results obtained with the proposed classifier, trained for 300 epochs, with a learning rate of 0.002

The learning process is visualised in Figure 5.7. The plot shows that validation loss increases slightly after 250 epochs. For this reason, we limit the training to 300 epochs to prevent overfitting.

We also calculated the accuracy, recall and precision for each class separately. The results can be found in Table 5.6 for the 4-class dataset and in Table 5.7 for the 8-class dataset. They are also visualised with the help of confusion matrices.

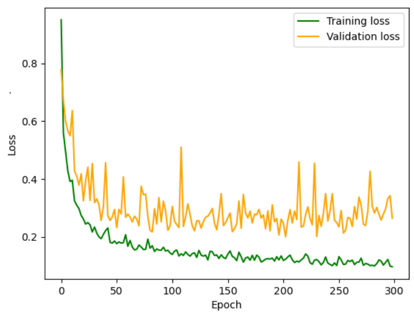


(a) : 4 classes

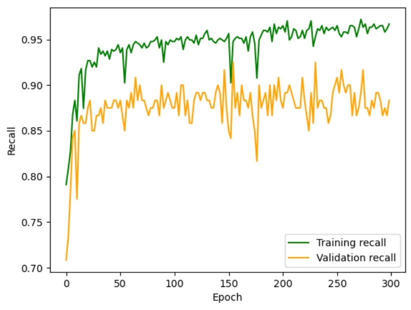


(b) : 8 classes

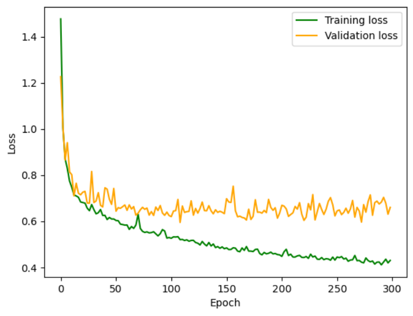
Figure 5.6: Confusion matrices evaluated on the validation datasets with the best performing model of the proposed classifier



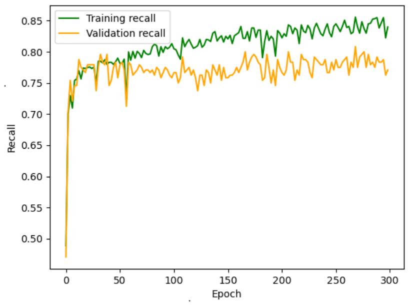
(a) : Loss, 4 classes dataset



(b) : Recall, 4 classes dataset



(c) : Loss, 8 classes dataset



(d) : Recall, 8 classes dataset

Figure 5.7: Evolution of loss and recall during training of the proposed classifier on both datasets with optimal hyperparameters

5.4 Comparison of the proposed method and baseline

As discussed in Section 5.1.2, the baseline method does not perform well when many fake actions are present in the dataset. This happens due to the high level of similarity between fake actions and corresponding actions with tools. By adding the second module, which is responsible for object recognition, the number of correctly classified videos increases noticeably. Recall value for the 4-class validation dataset increases from 59.26% to 89.17% and for the 8-class validation dataset from 41.2% to 80.83%. It can be seen that the best improvement is obtained in the classification of non-fake actions where a tool is present (Table 5.6 and Table 5.7). On the other hand, the recall value for some fake actions (fake hammering, fake pliering) slightly decreases compared to the results of the baseline classifier. Another disadvantage of our solution is, of course, the fact that it is only applicable when actions with tools (or a mixture of actions with tools and without tools) are used.

	Accuracy [%]	Recall [%]	Precision [%]
Pliering	100.0	100.0	100.0
Fake pliering	98.95	98.81	97.65
Wrenching	98.26	91.67	100.0
Fake wrenching	97.21	96.08	89.09
Total	98.6	97.21	96.68

Table 5.6: Metrics calculated for the best performing final model for the 4-class validation dataset

	Accuracy [%]	Recall [%]	Precision [%]
Hammering	99.58	100.0	96.77
Fake hammering	97.92	93.33	90.32
Pliering	94.58	80.0	77.42
Fake pliering	91.67	83.33	62.5
Hand-screwing	93.75	70.0	77.78
Fake hand-screwing	90.83	56.67	65.38
Wrenching	97.08	83.33	92.59
Fake wrenching	96.25	80.0	88.89
Total	95.21	80.83	81.46

Table 5.7: Metrics calculated for the best performing final model for the 8-class validation dataset

Overall, it can be concluded that the combination of two modules for action recognition, where the first module is a basic classifier for actions and the second module is responsible for object recognition, can improve the classification results in the datasets where actions requiring tools are present.

Chapter 6

Conclusion and future work

6.1 Discussion and conclusion

In this thesis, we propose and develop a method to combine an action recognition module that is described in [9] with an instance segmentation module from [23] and create an advanced version of the action classifier.

To train the first module, we create a dataset with eight different action types. The dataset contains both assembly actions with tools and corresponding "fake" actions without tools. We train the first module with different hyperparameters to achieve the best performance. We also modify the second module so that it can distinguish between different instances of the same class. We use the information about the centroids of each object instance in the video frames to calculate the average distance each object travels between two frames when the action is performed. Using the data obtained from the first two modules (predicted action by a baseline classifier, detected objects, the distance the objects move on average) we train our final classifier.

We achieve a recall/precision score of 80.8%/81.5% for the 8-class dataset compared to 42.0%/39.8% for a baseline solution. And a score of 97.2%/96.7% for the 4-class dataset compared to 60.9%/62.4%. Thus, we prove that combining an action recognition model that performs at an average level with an object recognition model can improve the overall action classification.

6.2 Future work

Larger dataset

One of the most constructive work to be done is probably creating a larger dataset for training the baseline action recognition module. It would be beneficial to observe whether the basic action recognition module can be trained well enough with a larger dataset if it still contains many actions with a similar motion. And it would also be interesting to see how a better trained action recognition module would affect the results of the final classifier.

Train the object recognition module better

Also, it would be worthwhile to train the object recognition module better. When analysing the YOLACT predictions in Section 5.2, we found that some object classes are poorly predicted. This could be due to the imbalanced dataset or insufficient training. Increasing the size of the object dataset and re-training YOLACT may solve this problem, which should also lead to improved accuracy of the proposed classifier.

Real time action recognition

Apart from this, it would be interesting to see if our solution is able to classify actions in real time. On the one hand, both PAN and YOLACT are designed to work in real time. On the other hand, the combination of these two modules can significantly slow down the multimodal classifier. So further experimentation in this area is needed.

Appendix A

Bibliography

[1] H. Jhuang, J. Gall, S. Zuffi, C. Schmid, M. J. Black. Towards understanding action recognition. In *IEEE International Conference on Computer Vision*, pp. 3192-3199, 2013.

[2] I. Laptev, M. Marszałek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-8, 2008.

[3] A. Patron-Perez, M. Marszalek, I. Reid, and A. Zisserman. Structured learning of human interactions in TV shows. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 12, pp. 2441-2453, 2012.

[4] K. Reddy and M. Shah. Recognizing 50 human action categories of web videos. In *Machine Vision and Applications 24*, pp.971-981, 2013.

[5] Yu Kong, Yun Fu. Human Action Recognition and Prediction: A Survey. *arXiv:1806.11230* 2018.

[6] Fleet D., Weiss Y. Optical Flow Estimation. In *Handbook of Mathematical Models in Computer Vision*, 2006

[7] Berthold K.P. Horn, Brian G. Schunck. Determining optical flow. In *Artificial Intelligence, Volume 17, Issues 1–3*, pp 185-203, 1981.

[8] J. Y.-H. Ng, J. Choi, J. Neumann, and L. S. Davis. Actionflownet: Learning motion representation for action recognition. In *IEEE Winter Conference on Applications of Computer Vision*, pp. 1616–1624, 2018.

[9] Z. Can, Y. Zou, G. Chen, and L. Gan. Pan: Towards fast action recognition via learning persistence of appearance. In *arXiv preprint arXiv:2008.03462*, 2020.

[10] Z. Can, Y. Zou, G. Chen, and L. Gan. PAN: Persistent Appearance Network with an Efficient Motion Cue for Fast Action Recognition. In

Proceedings of the 27th ACM International Conference on Multimedia, pp. 500-509, 2019.

[11] X Wu, D Sahoo, SCH Hoi. Recent advances in deep learning for object detection. In *Neurocomputing*, 2020.

[12] P. Arbeláez, B. Hariharan, C. Gu, S. Gupta, L. Bourdev and J. Malik. Semantic segmentation using regions and parts. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3378-3385, 2012.

[13] Hafiz A.M., Bhat, G.M. A survey on instance segmentation: state of the art. In *Int J Multimed Info Retr*, pp 171–189, 2020.

[14] C. H. Lampert, M. B. Blaschko and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-8, 2008.

[15] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt and B. Scholkopf. Support vector machines. In *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 18-28,1998.

[16] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu and S. Liu. Towards Better Analysis of Deep Convolutional Neural Networks. In *IEEE Transactions on Visualization and Computer Graphics*, no. 1, pp. 91-100, Jan. 2017.

[17] Z. -Q. Zhao, P. Zheng, S. -T. Xu and X. Wu. Object Detection With Deep Learning: A Review. In *IEEE Transactions on Neural Networks and Learning Systems*,vol. 30, no. 11, pp. 3212-3232, 2019.

[18] R. Girshick. Fast R-CNN In *Proceedings of the IEEE international conference on computer vision*,pp. 1440-1448, 2015.

[19] S. Ren, K. He, R. Girshick, J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems 28*, 2015.

[20] K. He, G. Gkioxari, P. Dollar, R. Girshick. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2961-2969, 2017.

[21] J. Redmon, S. Divvala, R. Girshick, A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779-788, 2016.

[22] L.Wei, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A.C. Berg. SSD: Single Shot MultiBox Detector. In *European conference on computer vision*, pp. 21-37, 2016.

[23] D. Bolya, C. Zhou, F. Xiao and Y. J. Lee. YOLACT. Real-time Instance Segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp.9157-9166, 2019.

[24] D. Bolya, C. Zhou, F. Xiao and Y. J. Lee. YOLACT++ Better Real-Time Instance Segmentation. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 2, pp. 1108-1121, 2022.

[25] B. Abul. Survey on evolving deep learning neural network architectures. In *Journal of Artificial Intelligence and Capsule Networks*, pp. 73-82, 2019.

[26] SC. Wang. Artificial Neural Network. In *Interdisciplinary Computing in Java Programming. The Springer International Series in Engineering and Computer Science*, vol 743, 2003.

[27] S. B. Maind, P. Wankar. Research Paper on Basic of Artificial Neural Network. In *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 2, no. 1, pp. 96-100, 2014.

[28] B. Hanin. Which Neural Net Architectures Give Rise to Exploding and Vanishing Gradients? In *Advances in Neural Information Processing Systems*, 2018.

[29] G. E. Hinton. How Neural Networks Learn from Experience. In *Scientific American*, pp 144–151, 1992.

[30] K. Janocha, W. M. Czarnecki. On Loss Functions for Deep Neural Networks in Classification. In *arXiv:1702.05659v1*, 2017.

[31] A. Sharkey. On Combining Artificial Neural Nets. In *Connection Science*, 1996.

[32] A. Mouna, Y. Said, and M. Atri. Computer vision algorithms acceleration using graphic processors NVIDIA CUDA. In *Cluster Computing 23, no. 4*, pp. 3335-3347, 2020.

[33] M. Vavrecka, G. Sejnova, M. Mejdrechova, N. Sokovnin. myGym's documentation. In *https://mygym.readthedocs.io/en/latest/index.html*, 2020.

[34] M. Vavrecka, G. Sejnova, M. Mejdrechova, N. Sokovnin. myGym: Modular Toolkit for Visuomotor Robotic Tasks. In *arXiv:2012.11643*, 2020.

[35] L. Yang, A. Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. In*Neurocomputing*, pp. 295-316, 2020.

[36] D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, G. E. Dahl. On Empirical Comparisons of Optimizers for Deep Learning. In*arXiv:1910.05446*, 2020.