FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of master's thesis

Title:	Virtual piano using image processing
Student:	Bc. Jiří Hanuš
Supervisor:	Ing. Tomáš Nováček
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2021/2022

Instructions

With the rise of new technologies, digitalization is one of the ways how to save money and make work more efficient. One of the possible goals of digitalization is the conversion of musical instruments into their virtual form:

Goals of the thesis:
• Analyse the possibilities of user interaction with the virtual environment with the use of hand and finger movement detection.

• Design a virtual piano, that will detect user's interaction with the use of image processing (with RGB camera and Leap Motion sensor).

• Compare the accuracy of the proposed approaches.

• Create an application that will create sound files in the MIDI format with the use of proposed virtual piano.

Electronically approved by Ing. Karel Klouda, Ph.D. on 21 October 2020 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Virtual piano using image processing

Bc. Jiří Hanuš

Department of Knowledge engineering
Supervisor: Ing. Tomáš Nováček

May 5, 2021

Acknowledgements

First of all, I would like to thank my supervisor Ing. Tomáš Nováček who gave me this opportunity to write my diploma thesis as his student with topic that I really like. He is really great teacher, actor, writer, organizer, Apakrychle researcher, leader and a very good friend. During my writing, he provided me much new professional knowledge about this field of study and spent many hours with me consulting my work and pushing my limits, thank you!

I also acknowledge all the teachers from Faculty of Information Technology, especially teachers from ImproLab for providing me knowledge about Computer Vision, which was really useful in my work.

I am grateful to Czech Technical University for enabling to study abroad as an exchange student in such a beautiful country as Taiwan where I have got most of my knowledge about Neural Networks and Deep Learning tasks.

I would like to express thanks to my colleagues from Technology Agency of the Czech Republic for giving me the time and support to write my thesis.

Many thanks to my family supporting me during my studies, both in my bachelor studies and now during my masters studies. Sometimes it was hard, but you have always motivated me.

Last but not least, huge thanks to my girlfriend Alenka, who has been supporting and motivating me during the process of writing my thesis, helping me to concentrate and making sure that I did not starve.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 5, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Hanuš, Jiří. *Virtual piano using image processing.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

S vzestupem nových technologií, digitalizace je jedna z cest, jak ušetřit peníze, čas a zároveň zefektivňovat práci. Jedna z možností digitalizace je také převedení hudebních nástrojů do jejich virtuální podoby.

V této práci jsem nejprve shrnul současnou podobu aktuálních technik vytvoření
virtuálního piana a také virtuálních klávesnic obecně. Popsal jsem techniky rozpoznávání ruky, prstů, gest a také různé přístupy snímacích zařízení.

Dále jsem v práci porovnal dva různé přístupy rozpoznávání gest rukou a prstů pro vytvoření virtuálního piana. První přístup je otestován pomocí technik zpracování obrazu z RGB kamery. Druhý přístup je pomocí Leap Motion Controller, což je optický modul pro sledování pohybu rukou. Dále jsem popsal výhody a nedostatky těchto přístupů a experimentálně změřil úspěšnost.

Pro rozpoznání pozic prstů a predikci stisku kláves na piánu využívám hluboké konvoluční neuronové síť (CNN), Stereo IR 170 Camera Module od firmy Leap Motion a další knihovny jazyka Python s předtrénovanými modely.

Na závěr předkládám software virtuálního piána pomocí jedné ze zmíněných technik. Software umožňuje uložit notový zápis hrané hudby do formátu MIDI.

Klíčová slova Virtuální piano, RGB kamera, Leap Motion, počítačové vidění, zpracování obrazu, MIDI soubory

Abstract

With the rise of new technologies, digitalization is one of the ways how to save money, time and make work more efficient. One of the possible goals of digitalization is conversion of musical instruments into their virtual form.

In this work I summarized the state-of-the-art and also provided an analysis of the different approaches about creating virtual piano and virtual keyboards in general. Few techniques how hand and and finger gesture recognition is done nowadays are also described in this work as well as different approaches of several controllers.

Further, I compared two different approaches to hand and finger detection in order to create the virtual piano. One way is making virtual piano using only RGB camera. Second way is using Leap Motion Controller, which is an optical hand tracking module that captures movement of your hands. Further, I described advantages and disadvantages of these approaches and experimentally tested the accuracy.

In the thesis, in order to recognize finger location and predict tapping on the piano keys I am using deep convolutional neural networks (CNN), Stereo IR 170 Camera Module by Leap Motion and other Python libraries with pretrained models.

In the end, I provided a virtual piano software using one of these techniques which is able save output of the piano to a MIDI file.

Keywords Virtual piano, RGB camera, Leap Motion, computer vision, image processing, MIDI files

Contents

List of Figures

List of Tables

Introduction

The digital age is here. People try to live their everyday lives but the pandemic world makes the conditions for them more difficult. However, with the help of digitalization, we are able to enjoy outside world from the comfort of our homes.

In the recent years, there has been a rapid progress in the field of hand tracking, which is a method of Human-Computer-Interaction (HCI). The state-of-the-art techniques are able to track hand motion and also hand gestures accurately. These techniques mostly use depth images to do so.

With this know-how, we should also be able to help the music industry to adapt to the new digital world.

In this work, I focused on creating virtual piano using image processing. The first part is consisted of state-of-the-art for virtual pianos nowadays, the analysis of the given problem and devices that were used.

The datasets and designs chapter contains information about collected RGB datasets and masked datasets. Further, all designs are described – four deep convolution neural network approaches and one approach using Stereo IR 170 Camera Module (similar to Leap Motion) by Leap Motion are tested. And last approach using again only RGB images with open-source library MediaPipe by Google which uses pretrained models to get the hand landmarks.

In the implementation part, I introduced my solution how to create virtual piano. First challenge was accomplishing the task real-time. Another challenge was classify whether more notes were played simultaneously.

First approach is by using only RGB camera. I use this approach, because most of the laptops have their own web camera and thus the resulting piano could be accessible for more people.

Second approach is by using Stereo IR 170 Camera Module controller by Leap Motion, which is able to detect hand gestures by projecting infrared pattern on the hand and it calculates the position of hand components based on the distortion of the captured pattern. Stereo IR 170 Camera Module (or Leap Motion Controller) costs around 260 (or 80) dollars could achieve

better accuracy than only with RGB images.

Third approach will be combination of first two approaches. The main goal is to create a piano which is affordable for everyone. Everybody has RGB camera in their laptops and the Leap Motion controller is cheaper than depth camera.

Most of today's state-of-the-art solutions are able to track hand motion precisely. The solutions comes with the accuracy around 100 micrometers. However, most of the solutions require that hands are in the air and not interacting with other object – for example with tapping on the table.

For more realistic user experience it would be better to interact with something – for example something simple as printed out keyboard on a piece of paper. That is why I am using this semi-virtual approach, where the user will print a piano keyboard. Then click with the cursor on the corners of the piano in the software to create perspective transformation matrix, which is further used in all experiments. In the end, the user can tap on the printed sheet to imitate real piano.

In my work, I will try Deep Convolutional Neural Networks to see and compare how they are good for this kind of problem.

In the end, I will provide a piano using image processing and fingertip recognition. This software will be also able to save the result to a MIDI file.

Chapter 1

Analysis

1.1 State-of-the-art

In this chapter, I compared different state-of-the art solutions of virtual pianos, hand, gesture tracking and finger tracking. There are two different approaches how it is done. Both of them are using depth sensors and one is projected on a display and the other solution is showing the result in virtual reality goggles.

1.1.1 Barehanded Music: Real-time Hand Interaction for Virtual piano

The state-of-the-art solution is the result presented in the article Barehanded Music: Real-time Hand Interaction for Virtual piano [11], where researchers are using fingertip tracking with a DepthSense 325 sensor and also finger tapping tracking.

The paper aimed to create virtual piano application, where when the users put their hands on a table and also tap the fingers on the desk, the piano will start playing. However, they did not use any printed paper sheet with the keyboard in this work. That is why the user can be confused what he is playing.

The researchers trained random regression forest (which is a machine learning model) on 7 200 RGB-D images to predict the position of the hand joints and then used the trajectories to detect finger tapping using support vector machine (SVM).

In the online tracking stage, they segment the hand from the plane by fusing the information from both color and depth images. Afterwards, they trained a random forest model to estimate the 3D position of fingertips and wrists in each frame and predict tapping based on the estimated fingertip motion.

In contrast to the other existing hand tracking methods which often need to have hands in the air and can not interact with physical objects, this method

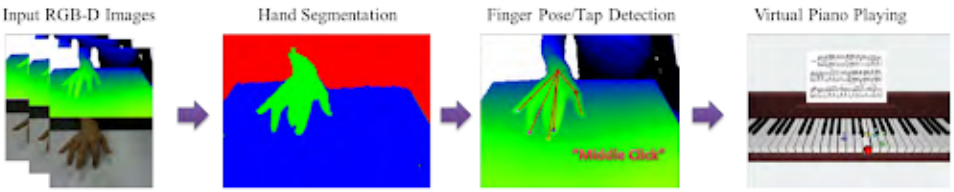


Figure 1.1: Barehanded Music – the workflow

is designed for interaction with planar objects. In my opinion that creates a better user experience.

1.1.2 A New Input Method of Computers with One CCD Camera: Virtual Keyboard

The researchers proposed a keyboard-style input method to computers with one compact charge-coupled device (CCD) camera [12]. First, they used a CCD camera to create video images after that they detected fingertips. If the fingertips were detected, they used a stroke detector which watches the alternation of moving vectors of each fingertip. Last step was the keyboard checker. This method had not been thought of as a practical one.

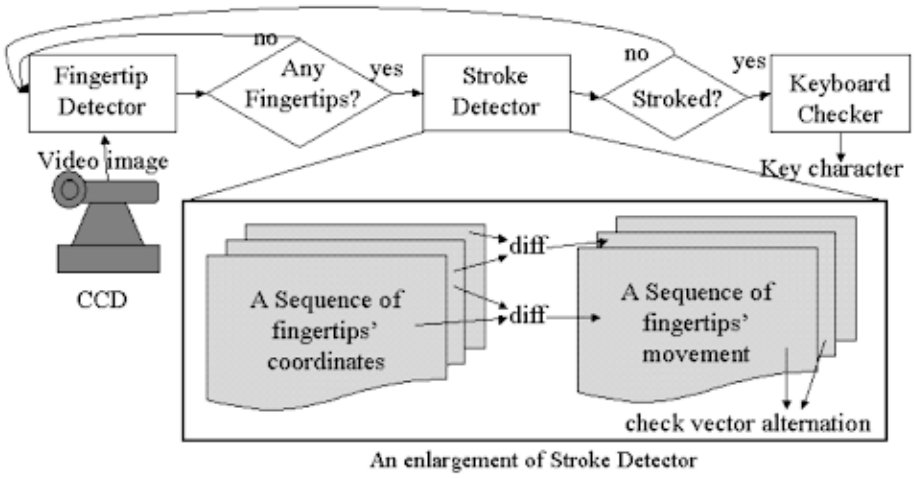


Figure 1.2: Workflow with one CCD camera

1.1.3 Hand Gesture Recognition with Leap Motion

The researchers used Leap Motion Controller which allows them to exploit the distortion of projected pattern to compute the depth information in order

to recognize hand pose very precisely [13]. They extracted a series of features along with a histogram of oriented gradient (HOG) from the controller and used that into a multiclass SVM classifier to recognize performed gestures. They also experimented with different approaches using dimension reduction and feature weighted fusion.

Their proposed architecture consists of tracking data and sensor images of gestures were captured simultaneously. After that, they extracted a series of related features and also the HOG features. Then they applied feature fusion and dimension reduction using principal component analysis (PCA). Finally, they fit the features into One-vs-One multi-class support vector to classify hand gestures.

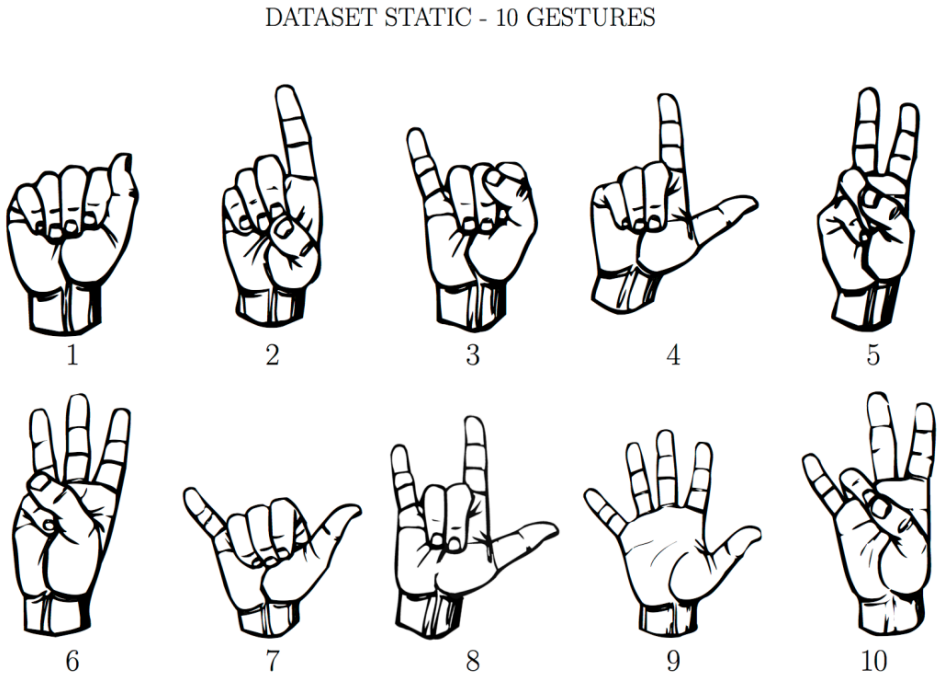


Figure 1.3: Ten hand gestures used

The dataset consisted of 10 gestures performed from 13 people and each gesture from each subject was performed 20 times; they had 2600 samples in total. With the Leap Motion Controller and extraction the HOG features they significantly improved gesture accuracy. The average achieved accuracy with all extracted features used was around 98%.

1.1.4 Hand gesture recognition with Leap motion and Kinect devices

In this article [14], two different gesture recognition algorithms have been proposed. The first uses the Leap Motion Controller and the second one uses Kinect device. According to this article, Leap Motion provides higher level but more limited data description while Kinect provides full depth map. That is probably because Kinect is for full body recognition and Leap Motion Controller specializes in hand, palm and finger recognition.

Leap Motion has not been completely reliable, since some fingers were not detected. Kinect allowed capturing other properties missing in the Leap Motion Controller and by combining the two devices a very good accuracy can be obtained. Experimental results showed also that the assignment of each finger to specific angular region leads to considerable increase of performance. The dataset they used were gestures from American Sign Language (ASL) that had been acquired for experimental results.

They used three major features from the Leap Motion Controller – fingertips distances, fingertips angles and fingertips elevations. For the gestures prediction they have got 76.07%, 74.21% and 73.07% accuracy respectively to the features. Accuracy of features combined together was 80.86%. The features proposed for Kinect were curvature and correlation. For curvature, they have accuracy of 87.28% and correlation 65% combined together 89.71%. Finally combining all various features from both sensors they obtained accuracy of 91.3%.

1.1.5 Motion Capture with Constrained Inverse Kinematics for Real-Time Hand Tracking

The researches researchers tried a marker-based motion capture method for hand tracking [15]. According to this article the method's results are highly accurate and the method is also facile to configure. However, it is sometimes impossible or very difficult to attach more markers on each limb segment. Hence, in this article they attempted attaching a single marker instead of attaching 3 markers on each limb segment as usual.

One marker is attached to each finger, 1 marker is at the chain base and 2 markers at strategic positions to help define the hand orientation.

Every joint of the finger has one or more degrees of freedom (DoF). The DoFs represent the rotation relative to their parent joints up to the root joint. In order to avoid manually setting these degrees of freedom to each joint, researchers employed a simulation mechanism called Inverse Kinematics solver, to help to place limbs according to their known end effector positions (markers on tips of the fingers).

To conclude this article, researchers were able to set up a prototype producing visually natural and bio-mechanically correct movements. Their system

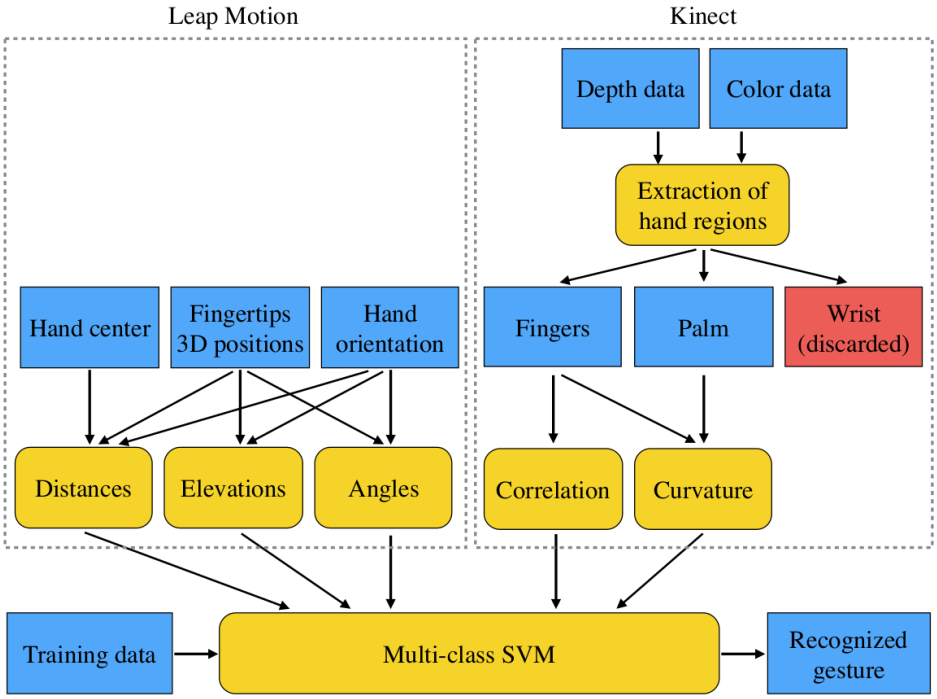


Figure 1.4: Workflows used for either Leap Motion or Kinect

can process up to 76 frames per second, so it could be set up to real-time production. However, they did not provide any accuracy of their proposition.

1.1.6 Keyboards without Keyboards: A Survey of Virtual Keyboards

The authors summarized the state-of-the-art in alphanumeric input interfaces in one article published in 2002 [16]. They described methods using non-virtual and virtual interfaces. In this work, I am going to write about the virtual ones in order to understand the principles of these virtual keyboards to use them in my thesis.

The virtual keyboard is described as a touch-typing device that does not have a physical manifestation of the sensing areas. That is the sensing area which acts as a button is not a button but is programmed to act as one instead. This sensing area could be for example a photo-electric sensor, a finger tracking method or a touch pad.

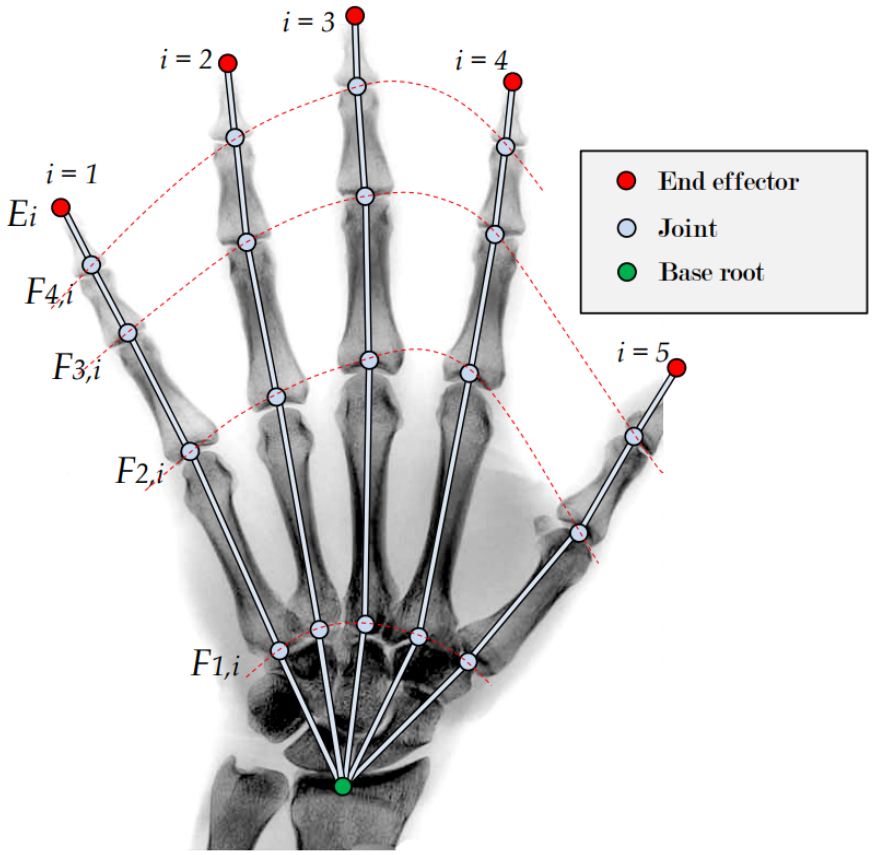


Figure 1.5: Hand description used for Motion Capture with Constrained Inverse Kinematics for Real-Time Hand Tracking

1.1.6.1 Visual panel

The visual panel consists of a camera and a sheet of paper. The location of the extended index finger in reference to the paper sheet is located with computer vision. Pressing a button in this method is done by resting fingertip in its current position for three seconds.

1.1.6.2 Finger-Joint Gesture Wearable Keypad

In this method, the phalanges of the fingers (besides the thumb) on one hand are meant to be the keys on a phone keypad

Figure 1.6: The ”Thumbcode” method

1.1.6.3 FingeRing

FingeRing uses accelerometers on each finger to detect surface impacts. There is a wireless version that communicates with a wrist-mounted device where the processing unit is.

1.1.6.4 Multi-Point Touchpad

Multi-Point Touchpad is a device offered by DSI Datotech Systems, which can report up to ten surface contacts and their pressure forces independently and simultaneously.

1.1.6.5 VType

VType detects the key stroke of each finger “in the air” with a data glove, which uses fiberoptical curvature detection. Locations of the keystrokes are not distinguished, only which finger pressed a key. The mapping of the input is done by statistical methods on the word and sentence level.

1.1.6.6 VKB Projection

This virtual keyboard is a tabletop unit that projects a laser image of a keyboard on any flat surface. Infrared cameras detect keystrokes of all fingers. With fixed detection areas, the detection of keys should result fairly good. The surface impact of the fingers is also detected and serves as a typing feedback.

Figure 1.7: VKB Projection

1.1.6.7 VKey

VKey by Virtual Devices Inc. is a device with combined projection and recognition, however no more information was provided. The device should detect the movement of all ten fingers. Just as the VKB Projection, the Vkey also consists of a tabletop unit. The feedback of hitting the surface is also measured.

1.1.6.8 Scurry

Scurry from Samsung is a device with tiny gyroscopes on each finger. The prototype suggested that these finger rings communicated with a wrist-mounted unit where data were processed.

1.1.6.9 Senseboard

Senseboard consists of two rubber pads that are wearable on the user's hands. Muscle movements in the palm are sensed and translated into keystrokes with pattern recognition methods.

Figure 1.8: Scurry

1.1.7 Real-Time Hand Gesture Spotting and Recognition Using RGB-D Camera and 3D Convolutional Neural Network

In this article from December 2019 [17], researchers proposed a novel method for fingertip detection and hand gesture recognition in real-time using RGB-D camera and 3D convolution neural network (3DCNN).

In their method the hand region of interest is extracted using in-depth skeleton-joint information from a Microsoft Kinect Sensor v2. After that, by using K-cosine corner detection, the fingertips are detected. The result of fingertip detection is transformed into the gesture initialization in order to spot hand gestures. Finally, the gesture is recognized based on the 3DCNN.

The hand region of interest and the center of the palm are first extracted from depth images provided by Kinect skeletal tracker. After that, the hand contours are extracted and described using a border-tracing algorithm.
The hand contours are computed using the Moore-Neighbor algorithm [18].

The K-cosine corner detection is an algorithm used to detect the shapes of certain objects. In this work it attempts to determine the angle between vectors of a finger.

The real-time hand gesture recognition is challenging because it is difficult to determine when a gesture begins and when it ends. Researchers had only six hand gestures. Swipe left, swipe right, swipe down, swipe up, zoom in and zoom out and the remaining gesture which was meaningless.

In this step, they used 3DCNN with six convolutional layers, three max-pooling layers and two fully-connected layers before softmax output. The fully-connected layers had 512 nodes with 50 % dropout. They had some issues with

Figure 1.9: Senseboard

different videos using different frame rates, so they decided to down-sample the input video to 20 frames per second in order to normalise the dataset.

Researchers prepared a dataset containing the 7 mentioned gestures. Each gesture was performed 15 times by 50 participants resulting in 5250 videos in total. Videos were also collected in three different places in order to have different light conditions in the dataset.

With three different models the best average accuracy of 92.6 % achieved was by the proposed 3DCNN. With adding ensemble models with different 3DCNN they got the best average accuracy of 97.12 %.

1.1.8 A real-time virtual piano based on gesture capture data

In the article presented by authors from Beijing Institute of Technology [19] the researchers used Leap Motion Controller to create virtual piano. The resulting piano can be played in virtual reality. In order to create the piano itself, they were obliged to prepare the 3D model.

After that, they used a physical model created in Unity named hinge to give the key a basic model as the rule of a movement. The hinge joint is a physics engine component which allows objects to rotate round the axis of hinge, just like a door and doorframe.

When they had the virtual piano created, they added colliders on each key in order to detect the collision between virtual finger and piano key. When

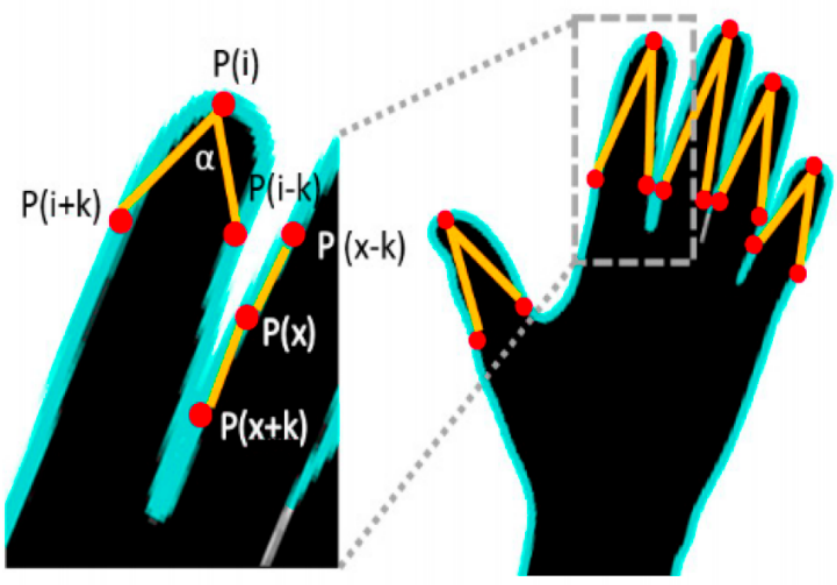


Figure 1.10: Fingertip detection using K-cosine algorithm used in their article

Models	Training Time	Testing Time	Accuracy (%)
SVM	21.94 m	2.57 s	60.50
2DCNN	50.00 m	3.46 s	64.28
3DCNN	1h35	5.29 s	92.60

Models	Accuracy (%)
3DCNN	92.60
Ensemble model with 5 different 3DCNN networks	96.42
Ensemble model with 10 different 3DCNN networks	96.82
Ensemble model with 15 different 3DCNN networks	97.12

Figure 1.11: Results of different models and with ensemble models

the virtual finger hits the key, the collision is going to be detected and the key is going to rotate as a hinge under the control of the physics engine.

The researchers also added some constraints in order to avoid users hitting the key from the side, so they allowed to press the key in one axis only. Also they added constraint to trigger the sound of the piano if the key is rotated at least 4 degrees.

In the testing phase, they compared five different gestures to evaluate accuracy. The gesture represented one finger bending. A demonstration of

their work is in figure 1.11.

The conclusion described that the piano has a good performance, but also that the user had no force feedback. This causes users to try moving their hands to find the location of their virtual hands and it is not easy to find the right key.

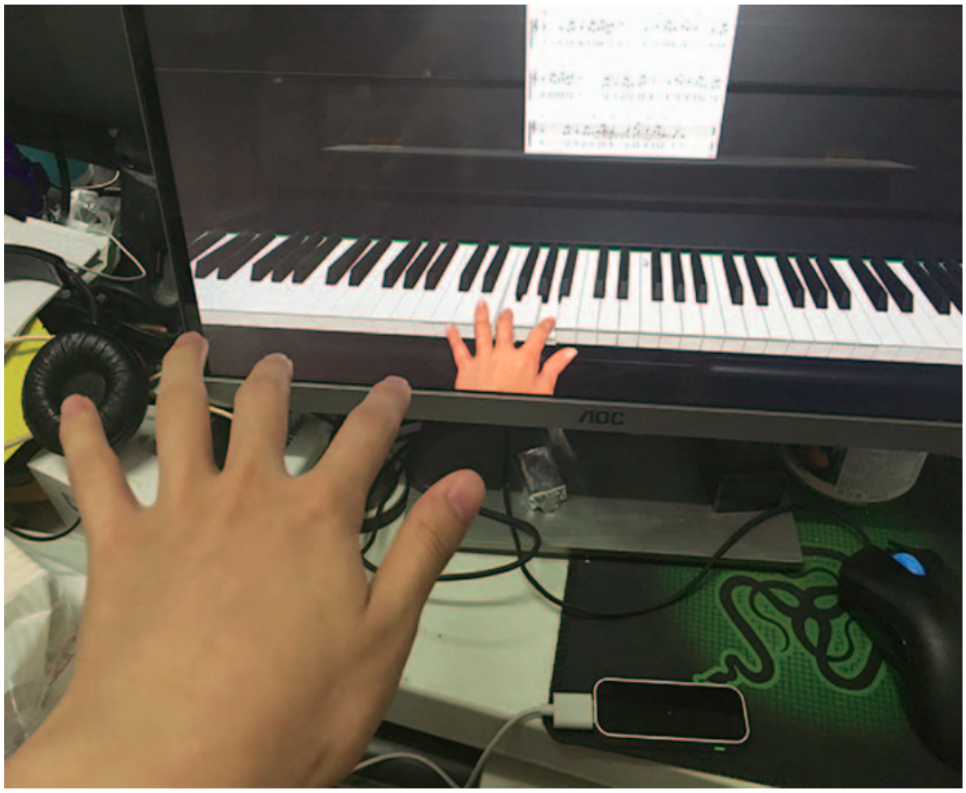


Figure 1.12: Virtual piano from an article A real-time virtual piano based on gesture capture data

1.1.9 "Virtual Keyboard" Controlled by Spontaneous EEG Activity

This article describes a "virtual keyboard" using the EEG [20], whereby the EEG is modulated by mental hand and leg motor imagery. Researchers report on three able-bodies subjects, operating the virtual keyboard.

This keyboard is used to help patients suffering from late stage of amyotrophic lateral sclerosis causing a locked-in syndrome. The virtual keyboard allows patients to write with a speed approximately one letter every 2 minutes and classification accuracy of 70-80 %.

I did not find this article not useful for my work, but it is a fresh view, how the human-computer interaction can also be done.

1.1.10 A Virtual Keyboard Based on True-3D Optical Ranging

In this paper is presented a complete system which mimics a QWERTY keyboard on an arbitrary surface [21]. The system consists of a pattern projector and a true-3D range camera to detect typing events.

They used the depth information acquired with the 3D range camera and detected the hand region. After that, the fingertips are discovered analysing the hands' contours and fitting the depth curve with different feature models. In order to detect keystroke, they analyzed the feature of the depth curve and mapped it back to the global coordinate system to find which key was pressed.

According to the article, their system could be also extended to the application of a virtual mouse because the finger tracking method can precisely locate the position of a moving finger in the working area, and detect the click event in the same way as the detection of keystroke event.

However, the challenge for their solution is tracking multiple fingers and record their traces in the scene at once.

1.1.11 Overview of controllers of user interface for virtual reality

This overview is a summary of many possibilities for human-computer interaction, especially controllers for virtual reality, but it also describes other usable possibilities such as wearable interfaces, eye tracking, haptic suits or even for virtual locomotion [22]. The article is divided into five main parts where it is described, how different body-parts are tracked – Positional tracking, Hand-based controllers, Body tracking and wearable body haptics, Gaze for eye tracking and Locomotion which specializes in treadmills.

According to this article, hand-based controllers are separated into handheld controllers, hand motion tracking and wearable devices. The most useful part for my thesis is the hand motion tracking. It can be done by optical tracking which is used by Leap Motion Controller [23] (2016), which uses two infrared cameras to compute distortion of projected IR grid by the controller. It also provides API about the hand segments.

Two mentionable sensors apart the ones that are using are Okuli (2015) by researchers from the University of Wisconsin-Madison which is a system for tracking fingers using visible light. It uses LED lights and two light sensors. Canesta Projection Keyboard (2003) – which is a system projected virtual keyboard on a plane surface and when the user crosses the IR light, the keystroke is detected.

The positional tracking consists of two major types – optical tracking and non-optical tracking. According to this article, optical tracking, in general, is the most common positional tracking for virtual reality. However, most modern systems use a combination of optical and non-optical tracking.

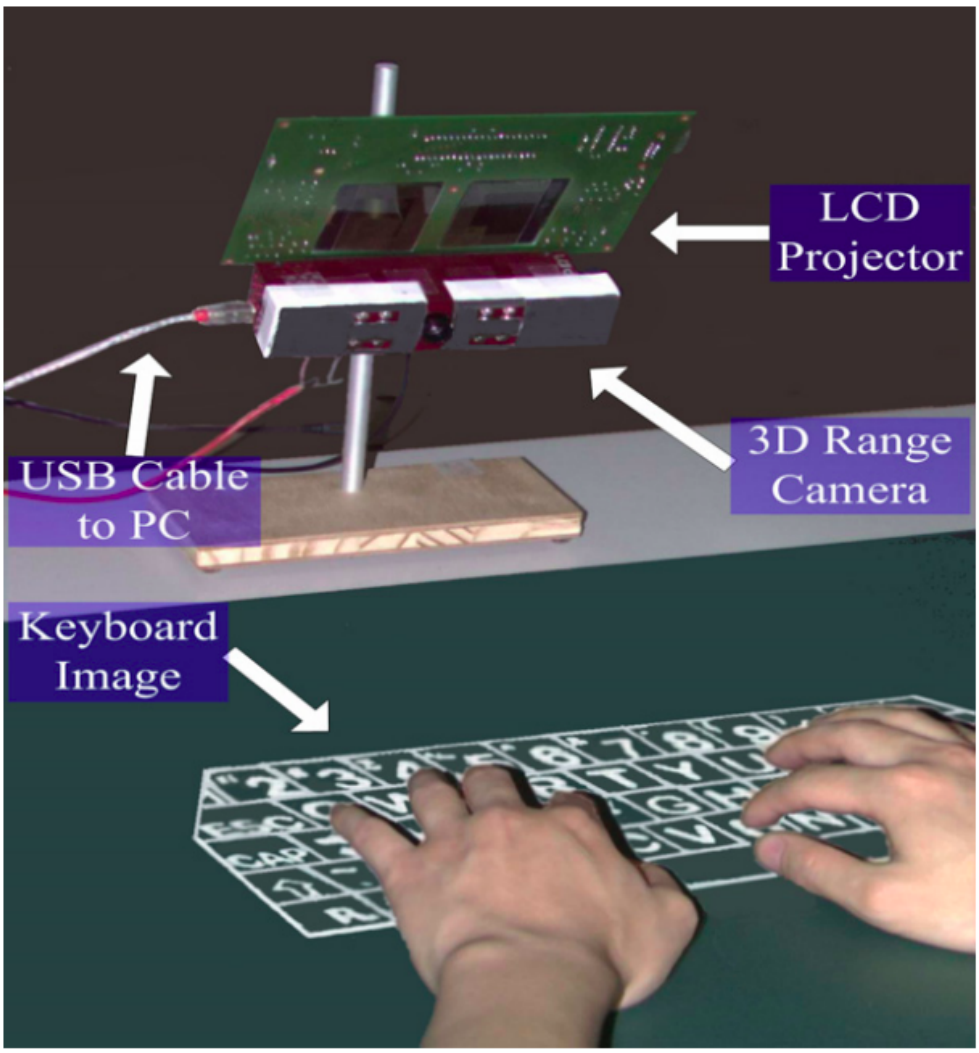


Figure 1.13: A Virtual Keyboard Based on True-3D Optical Ranging

Optical tracking could be divided into two methods. Passive optical tracking where the position of the user is usually calculated by the reflection of optical (e.g. infrared light) or reflection of special markers. The optical tracking is called active optical tracking if the signal is generated by special markers (e.g. LED lights).

Another different method is used by markerless tracking which uses model-based approaches or image processing to extract features of the space – edges, corners.

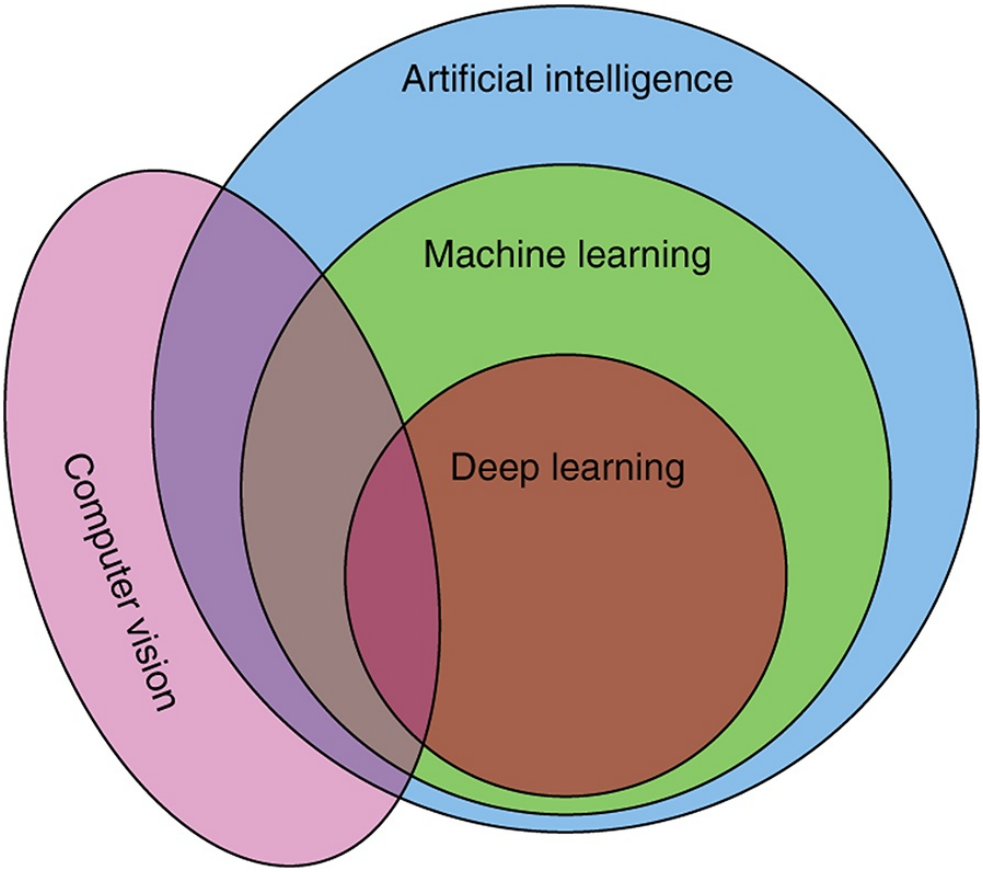


Figure 1.14: Venn diagram of computer vision with respect of artificial intelligence [1].

1.2 Computer vision

Computer vision is a set of algorithms to process various types of image data in order to describe the world that we see [24]. With this image, data we are trying to reconstruct its properties, such as shape, color or illumination. The main goal is to automate tasks that could be really simple for humans such as counting fingers in the image, but for computer the process is challenging.

This is why researchers came with many techniques to describe the image properties. Thanks to this the computers are now able to recognize people in the image, classify various shapes, detect and localize object in the image. The state-of-the-art solutions are combining computer vision and machine learning techniques [25].

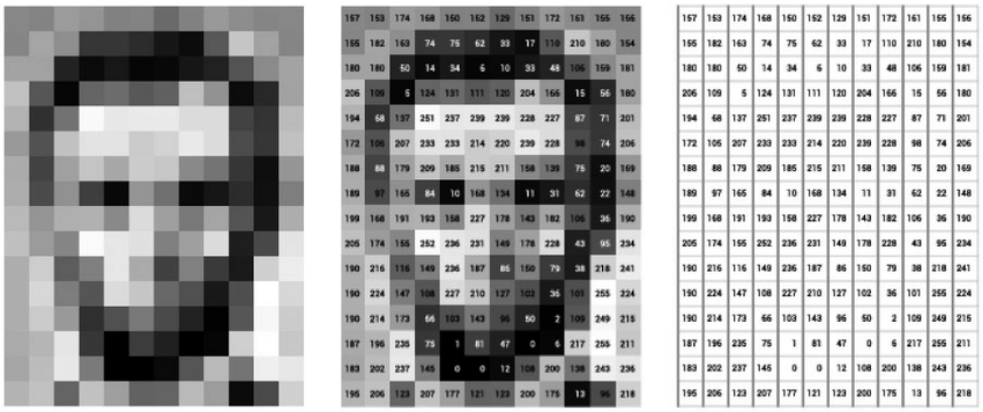


Figure 1.15: Grayscale image representation [2].

1.2.1 Image representation and RGB camera

To use computer vision techniques, we must first understand the image representation. Image in general is a 2D matrix of pixels. The value of the pixels represents how bright the pixel is. You can see an example of representation in the figure 1.15. The more quality picture you want, the more pixels is needed.

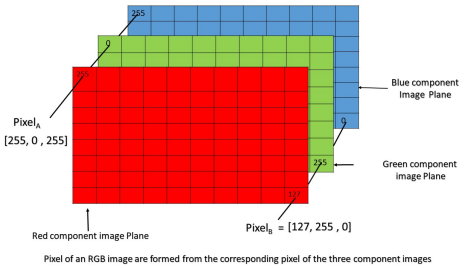
RGB image is similar to grayscale. The only difference is that there are three matrices stacked on top of each other. One for every color red, green and blue – and the value of pixel represents how bright the pixel for each color should be. By stacking red, green and blue values together, it is possible to create almost every color.

Alternative representation of color image are HSV, which means hue, saturation and value. The hue element represents which color is used. Saturation describes the saturation of the color and the value determines how bright should be the color.

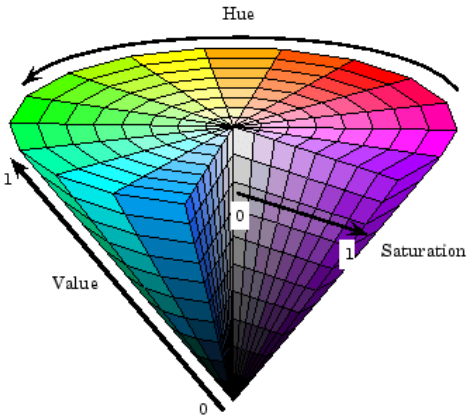
1.2.2 Leap Motion

The Leap Motion Controller by Leap Motion (formed from Leap Motion and Ultrahaptics in 2019) is a device for hand gesture controlled user interfaces with declared sub-millimeter accuracy [3]. It is a sensor device that aims to translate hand movements into computer commands. The controller itself is an eight by three centimeter unit that plugs into USB on a computer. Placed face up on a plane, the controller senses the area above it to a range of approximately one meter [23].

The heart of the device consists of two cameras and three infrared LEDs. These track infrared light with wavelength of 850 nanometers which is outside the visible light spectrum. The LEDs pulse is in sync with the camera framerate allowing for significantly lower power use and increased intensity.



(a) RGB image representation [26].



(b) HSV image representation [27].

Figure 1.16: RGB and HSV color spaces

There are two main hand tracking modules from Leap Motion company, the Leap Motion Controller and Stereo IR 170 Camera Module. They both work on the same principle. The Leap Motion Controller has interactive zone that extends from 10cm to 60cm or more in a 140x120° but the Stereo IR 170 Camera Module has a larger interaction zone extending from 10cm to 75cm or more with a 170x170° field of view. The range is limited by LED light propagation through space, as it becomes much harder to infer your hand's position in 3D beyond certain distance. The LEDs intensity is limited by the maximum current that can be drawn over the USB connection.

The Leap Motion Service feeds the result – expressed as a series of frames, or snapshot containing all of the tracking data – into a transport protocol. With this protocol, the service communicated with Leap Motion Control Panel, as well as native and web client libraries, through a TCP or WebSocket connection. The client library organizes the data into an object-oriented API structure.

Leap Motion has two modes – Desktop mode and Headset mode. The Desktop mode is optimized to when the controller is placed on a plane and detect hand gestures from below. The Headset mode is used when the controller is attached on VR headset in order to detect hands for virtual reality tasks.

1.2.3 MediaPipe

MediaPipe from Google offers cross-platform, customizable machine learning (ML) solutions for live and streaming media. The MediaPipe Holistic provides simultaneous perception of human pose, face landmarks and hand tracking. The software itself is an open-source [4, 28].

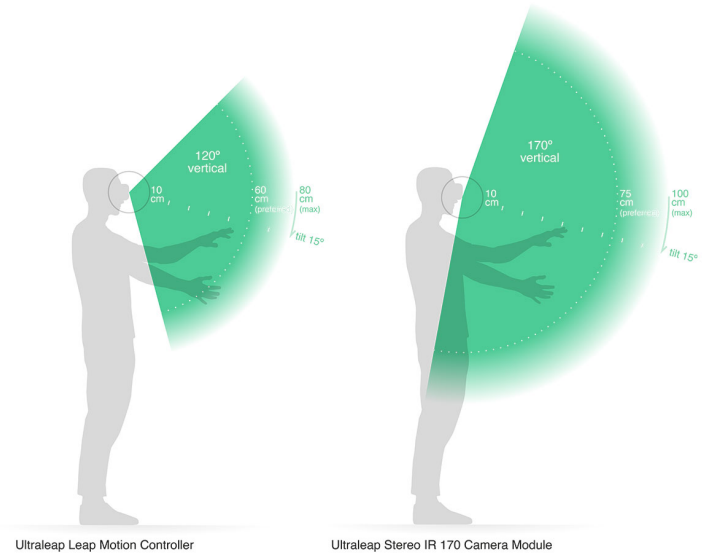


Figure 1.17: Leap Motion Controller and Stereo IR 170 Camera Module [3].

MediaPipe Holistics provides a unified topology for groundbreaking 540+ keypoints (33 pose, 21 per-hand and 468 facial landmarks). It is released for mobile phones (Android, iOS) and desktop. They also provided ready-to-use API for Python and JavaScript.

The MediaPipe Holistics integrates separate models for pose, face and hand components, each of which are optimized for their domain. Because models specialize in different recognition, the input for one model is not suitable for other one. Therefore, they designed it as multi-stage pipeline, which treats the different regions separately with appropriate image resolutions.

The MediaPipe Hands is a high-fidelity hand and finger tracking solution. It employs machine learning to infer 21 3D landmarks of a hand from a single frame. First, the software detects the palm and returns an oriented hand bounding box.

After the palm detection is used, the Hand Landmark Model that localizes hand-knuckle coordinates inside the detected hand region via regression. The researchers manually annotated ground truth data, which were 30K real-world images with 21 3D coordinates. The Z-value was obtained from image depth map if it existed.

You can see example images of landmarks of the palm and the connection between hand-knuckle in the figure 1.18.

1.2.4 Perspective transformation

When human eyes sees a near things they look bigger than compare to those who are far away. In general, this is called the perspective. The perspective

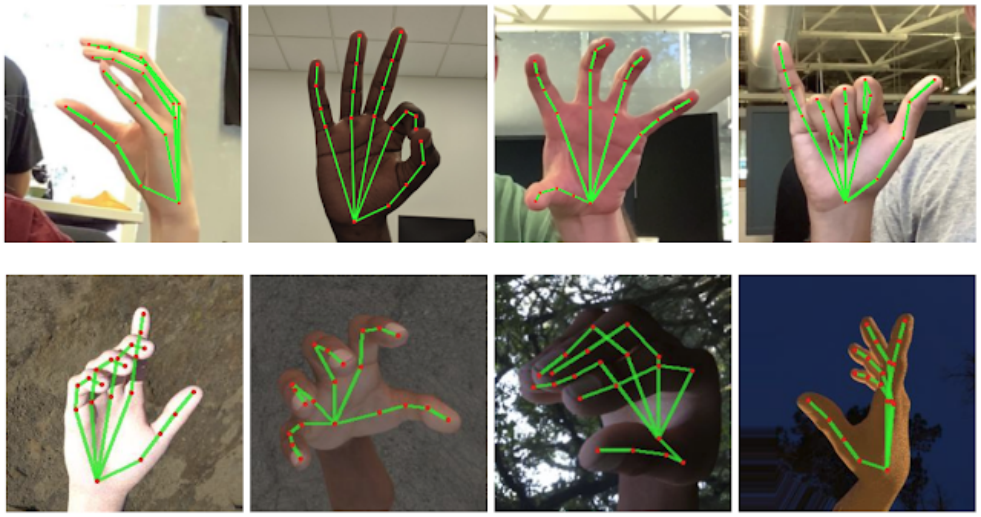


Figure 1.18: MediaPipe hand tracking [4]

transformation deals with the conversion of 3D world info 2D image. Same principal is used in human eye as in the camera.

In the OpenCV library [29] there are many geometric transformations functions and one of them is the perspective transformation where you need a 3 x 3 transformation matrix M. To find matrix M, you need four points on the input image and corresponding points on the output image. Among these 4 points, 3 of them should not be collinear. The matrix M could be further used to any point on the input image to find corresponding coordinates on the ouput image.

1.3 Musical Instrument Digital Interface – MIDI

MIDI is a specification of a communication scheme for digital music devices [30, 31]. But rather, it is an agreement among manufacturers of music equipment, computers, and software that describes a means for music systems and related equipment to exchange information and control signals [32].

The difference between audio information and MIDI data is similar to a record of musician playing piano and the music sheet of the same music. The record stores the information about the sound itself, but the sheet, representing the MIDI, stores only the information which tone and when it is played.

But MIDI does not represent only the tone, but also its length, volume etc. It can also represent which instrument is playing. The MIDI protocol specifies the meaning of each data value and provides means to store, manipulate, transmit, and re-create the information using symbolic data [32].

The data composed via the sequenced MIDI recordings can be saved as a standard MIDI file (SMF). This files usually uses *.mid* extension and are broadly spread in computers, old mobiles ringtones and also metadata for karaoke devices [33].

1.4 Neural Networks

"Any AI smart enough to pass Turing test is smart enough to know to fail it." – Ian McDonald

Neural networks were developed to simulate human nervous system for machine learning tasks. This is done by treating computational units in a learning model in a manner similar to human neurons [7]. This is not a simple task and human brains are still much faster than the fastest computer today.

First concepts of neural network appeared in 1950s, but in 1958 Frank Rosenblatt created the first perceptron algorithm which caused a great initial excitement about artificial intelligence. However, computers in 1960s were not fast, neural networks could not learn enough and there were also not sufficient amount of data for these data hungry algorithms.

At the end of the century, the computational force has grown and the data storage system grew with it too. These new possibilities led to new excitement over neural networks after 30 years of deep sleep and got the new label of "deep learning".

We are still far away from general artificial intelligence but there are many fields where neural networks dominate over simple statistical models, such as image recognition, classification, self-driving cars or time-series prediction.

Neural networks are theoretically capable of learning any mathematical function with sufficient data, and some variants like recurrent neural networks are known to be Turing complete, which refers to the fact that a neural network can simulate any learning algorithm, given sufficient training data [7].

1.4.1 Rosenblatt's perceptron

Simplest neural network is referred to as the perceptron. This neural network has a single input layer and an output layer. The architecture is visible in figure 1.19. The training sequence is formed of instances $(\bar{X}, y)$, where each $\bar{X} = [x_1, ..., x_k]$ contains k feature variables, and $y \in \{-1, +1\}$ (or $\{0, 1\}$) contains the observed value of the binary class variable. For example, in a medical field, it is possible to collect data about the patient (height, weight, temperature, age, ...) and the observed binary value could be if the patient is sick or not. The task is to predict the patient health state based on new data.

The input layer contains k nodes that transmit k features $\bar{X} = [x_1, ..., x_d]$ with edges of weight $\bar{W} = [w_1, ..., w_k]$ to and output node. The input layer does not do any computation. The linear function is $\bar{W} \cdot \bar{X} = \sum_{i=0}^{k} w_i x_i$

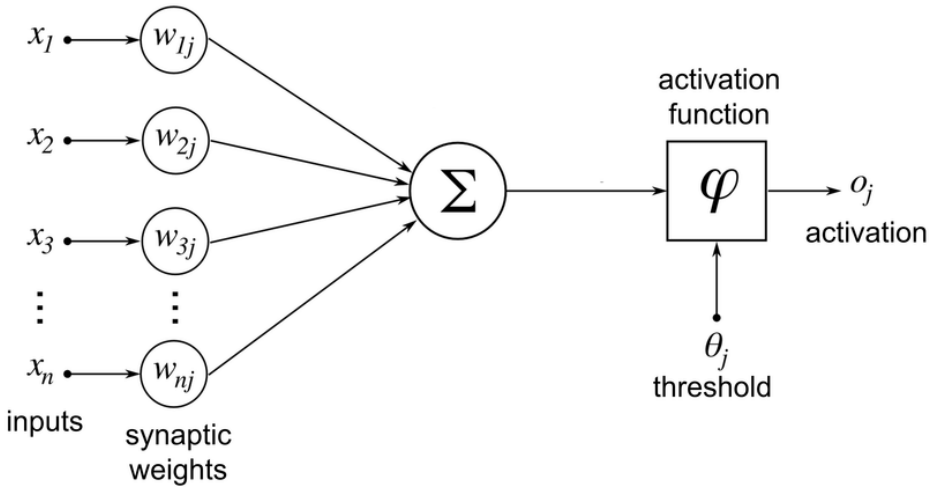


Figure 1.19: Rosenblatt's perceptron [5]

and subsequently, the sign of this real value is used. The prediction of $\hat{y}$ is computed as follows:

$$\hat{y} = sign\{\bar{W} \cdot \bar{X}\} = sign\{\sum_{i=0}^{k} w_i x_i\} \tag{1.1}$$

The sign function maps a real value to either +1 or -1, which is appropriate to binary classification. The error of the prediction is $E(\bar{X}) = y - \hat{y}$, where y is the real value and $\hat{y}$ is the predicted value. Output for this error is in the set of $\{-2, 0, +2\}$. In cases where $E(\bar{X})$ is nonzero, weights in the neural network need to be updated in the (negative) direction of the error gradient [7].

The sign function serves here as the activation function. Different choices of activation function can be used to simulate different types of models used in machine learning.

In some cases, there is an invariant part of the prediction, which is referred to as the bias. For example some variables are mean centered, but the mean of the binary class prediction from $\{-1, +1\}$ is not 0. This causes that the binary class distribution imbalance. That is why we are adding additional bias variable b that captures this invariant part of the prediction.

$$\hat{y} = sign\{\bar{W} \cdot \bar{X} + b\} = sign\{\sum_{i=0}^{k} w_i x_i + b\} \tag{1.2}$$

The main goal is to minimize the error in prediction, that is why we are introducing the loss function. For single perceptron model, the loss function can be represented in a similar way as least-squares algorithm with respect to all training instances in dataset $\mathcal{D}$:

$$Minimize_{\bar{W}} L = \sum_{(\bar{X},y)\in\mathcal{D}} (y - \hat{y})^2 = \sum_{(\bar{X},y)\in\mathcal{D}} (y - sign\{\bar{W} \cdot \bar{X}\})^2 \tag{1.3}$$

However, for continuous values target variables and the corresponding loss is a smooth and continuous function of the variable. The sign function is not differentiable and the loss function resemble staircase. This means that the gradient descent is not possible. As a result, the perceptron algorithm uses a smooth approximation of the gradient of this objective function with respect to each data:

$$\Delta L_{smooth} = \sum_{(\bar{X},y)\in\mathcal{D}} (y - \hat{y})\bar{X} \tag{1.4}$$

Eventhough the objective function is defined over the entire training data, the training itself of the neural networks works by feeding each input data instance $\bar{X}$ into the network one by one (or in small batches) in order to create prediction $\hat{y}$. After that we need to update the weights $\bar{W}$ based on the error $E(\bar{X}) = (y - \hat{y})$. Specifically, when the data point $\bar{X}$ is put into the neural network, the weight vector $\bar{W}$ is updated:

$$\bar{W} \Leftarrow \bar{W} + \alpha(y - \hat{y})\bar{X} \tag{1.5}$$

The parameter α is the learning rate. The perceptron algorithm cycles through the training data in random order and adjusts weights until the convergence is reached. One cycle is reffered to as an epoch. This basic perceptron algorithm can be considered a stochastic gradient-descent method. Overall, the perceptron model is good at classifying datasets which are linearly separable. On non-linearly the perceptron algorithm is not guaranteed to converge.

1.4.2 Multi-layer neural networks

Multi-layer neural networks contain more than one computational layer. This architecture is also called feed-forward networks because they are feeding the output of the layer to the next layer. As it is possible to see in figure 1.20, this neural network contains three hidden layers. As the output layer computation is visible to the user, computations performed in hidden layers are not visible to the user.

The multi-layer neural network is not learning the same way as the original perceptron algorithm. For these types of neural network, we use the back-propagation algorithm. It contains two phases – forward and backward phase. In the forward phase, the inputs are fed into the neural network and this causes the neural network trying to predict the output. Thanks to the loss function, we know how the neural network performed. Using the backward phase, the neural network is changing the weights with respect of the calculated gradients.

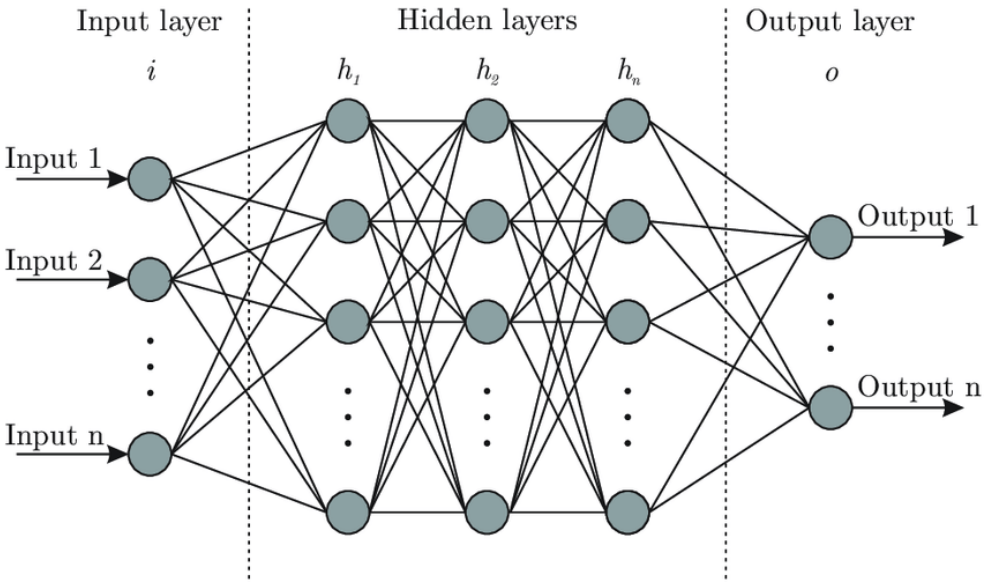


Figure 1.20: Neural network with 3 hidden layers [6]

1.4.3 Autoencoder and U-net

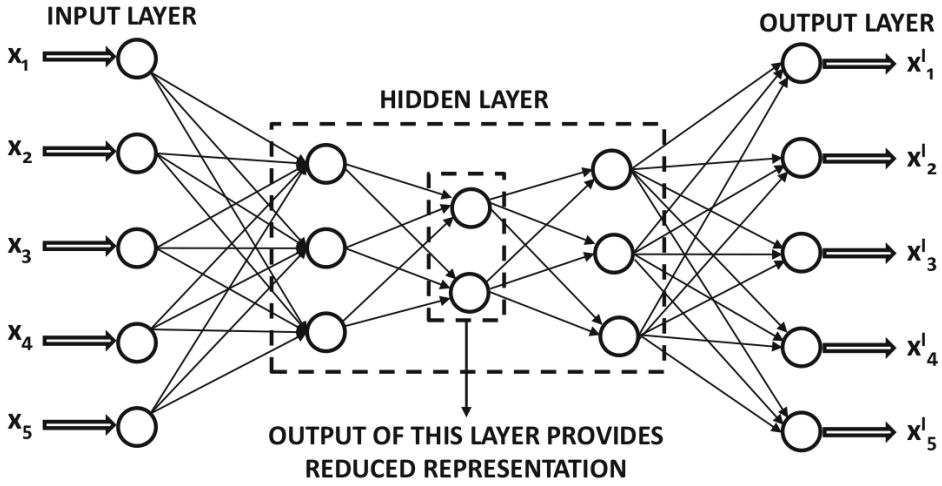


Figure 1.21: Autoencoder architecture [7]

The main idea of autoencoders is simple. We have the same output from the neural network as we have the input to the NN. There is a smaller number of neurons in the hidden layers and so the neural network is trying to fit the data into lower feature space and from the smaller amount of data predict the input. Thanks to this, autoencoders could be used in dimension reduction, similar to for example principle component analysis (PCA).

The autoencoder consists of three parts: Encoder, latent vector and decoder. The encoder part is reducing the dimension and encoding the data into the latent vector. The output of this latent vector could be also used as the output with reduced dimensionality. Third part is the decoder which is stretching the data back to its initial shape.

An example of basic autoencoder is in figure 1.21.

A similar architecture to autoencoder is the U-net. The only two differences are that U-net does not have the latent vector in the middle and the output of this network does not have to have the same shape as the input.

The U-net architecture is widely used in medical image segmentation [34, 35, 36, 37].

1.4.4 Convolutional neural network

Convolutional neural networks are inspired by nature. They are networks that are widely used for computer vision tasks, such as image classification or object detection. This type of neural network is based on the research by Hubel and Wiesel about the functionality of the cat's visual cortex and activation of the neuron by the [38].

Input for CNN are usually images, 3-dimensional for colored images (represented as RGB) or 2D image for grayscale image. Then each hidden layer of convolutional neural network is 3-dimensional, because the convolution uses a 3D filter. The layer itself contains various features of encoded shapes in the image. Typical CNN has two types of layers – convolution layer and subsampling (or pooling), respectively.

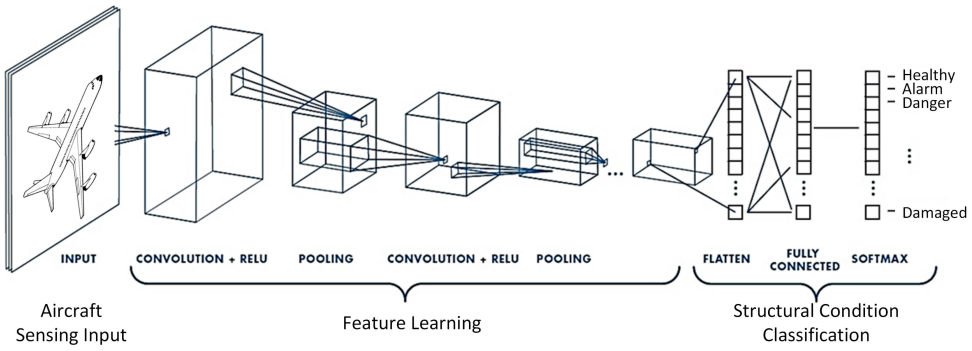


Figure 1.22: Convolutional neural network for image classification [8]

In the convolution layer, various filters are applied to the input image and thanks to that a new features are created from the previous layer. The filters are usually smaller n x n matrices (i.e. 3x3) and they are extracting information about the spatial region (i.e. edges, shapes).

The subsampling (pooling) layers simply averages (or picks a maximum) values in local regions of size typically 2x2. This technique is performed in order to compress the spatial information in the layer by a factor of 2.

The architecture of convolution and subsampling layers is used for example by LeNet-5 [39]. Other famous architectures of CNN are for example ResNet [40], VGG [41], GoogLeNet [42] or AlexNet [43].

1.4.5 Recurrent neural network

Recurrent neural networks (RNN) are designed for sequential data such as time-series or text sentences. The difference between feed-forward neural networks and recurrent neural networks is that in recurrent neural network the output of the layer is also used as an input to the same or former layer.

In this manner, the recurrent neural network is able to keep some information in memory. For example, if we are collecting temperature data each hour and feeding this information into the RNN, the network is able to comprehend some periodicity of time series. For example, that it is colder at night and warmer in the day. With this information, we are able to predict future values better.

The recurrent neural network is not learning by simple backpropagation algorithm but it uses its special type of it called backpropagation through time. According to the book Neural Networks and Deep Learning [7], RNN are Turing complete which means that with enough data and computational resources, they are able to simulate any algorithm.

1.4.6 Long short term memory – LSTM

Long short term memory is a special type or recurrent neural network. Simple RNNs have problem with vanishing and exploding gradient [44]. This causes that simple RNN is decent at learning short sequences and so it has good short-term memory but poor long-term memory.

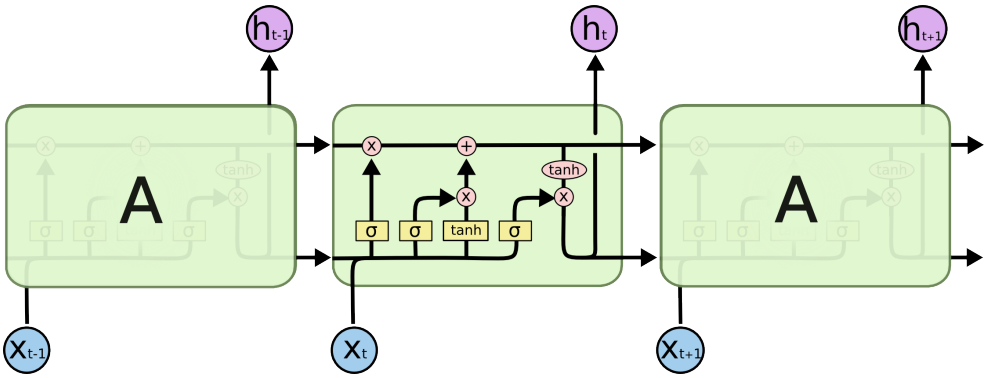


Figure 1.23: Long short term memory architecture [9]

This is why researchers proposed "The long short term memory" neural network which is designed to have a hidden vector inside in order to remember long-term information.

The core idea of LSTM is the cell state which tries to save the long term information. Then there are three gates to control the flow into this cell state. First one is the "forget gate layer" which decides if the old information is important to remember. The decision itself is made by the sigmoid layer which outputs a number between 0 and 1. 1 stands for completely keep this information and 0 is to completely drop this information.

After that is the input gate layer, which decides what information should be stored in the cell state. This layer creates the new candidate cell state and updates it. The last gate is the output gate. The output is created from combining the new input and updated cell state [9].

1.4.7 Activation functions

An important part of designing neural network is choosing the activation function. For the perceptron we used the sign as activation function because we were predicting only binary class. If we would want to predict the probability of certain value, it would make sense to use sigmoid as activation function because output of the sigmoid is value between 0 and 1. The mathematical representation of sigmoid function is:

$$\phi(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} \tag{1.6}$$

The convolution neural networks often use Rectified Linear Unit (ReLU) which only keeps the positive part of its argument. ReLU have replaced some activation functions (i.e. sigmoid, soft tahn) because of the ease in training multilayered neural networks. It can be seen in figure 1.24 and it is defined as:

$$\phi(x) = max\{0, x\} \tag{1.7}$$

1.4.8 Transfer learning

Transfer learning is the improvement of learning in a new task through the transfer knowledge from a related task that has already been learned. It is a popular approach in deep learning that researchers use pre-trained models as a tarting point for their task. The processing power to train new neural network could be high and the transfer learning is one of the solutions. This technique is used in computer vision tasks, natural language processing and etc. [45, 46].

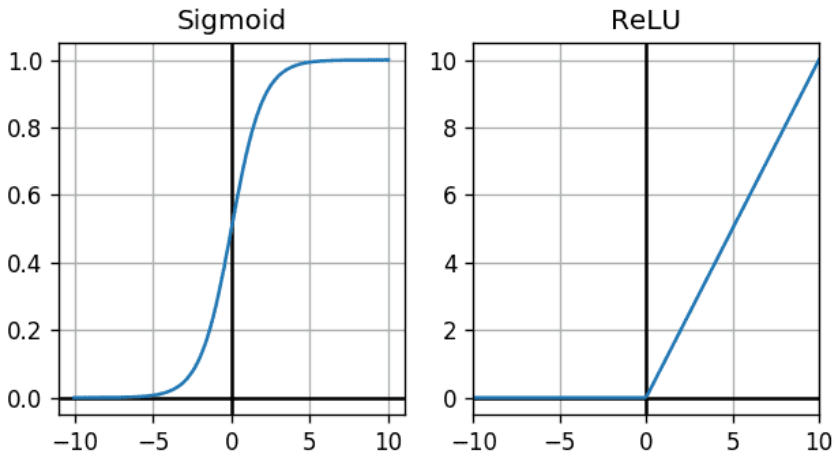


Figure 1.24: Sigmoid and ReLU [10]

1.4.9 Overfitting and Dropout

Deep neural networks are very powerful. However, the overfitting is a problem in these networks. Deep NNs could learn many combination of inputs and predict the output based on the exact input. But when this neural network sees a new data, it can not predict the output accurately and the testing accuracy is dropping. In machine learning, there are many methods to avoid overfitting such as L1, L2 regularization or early stopping.

For deep neural networks there is a technique called dropout. The key idea is to randomly drop neurons along with their connections from the neural network during the training. This causes that many neurons has to cooperate in order to predict better value and the neural network is not dependent only on certain neurons. This method gives major improvements over other regularization methods and provides better performance of neural networks in supervised learning [47].

Chapter 2

Datasets and designs

For my diploma thesis I created 2 datasets and both are from RGB images. One dataset contains fingers pressing the notes which are further used for the tone classification. The second dataset contains images of masked fingers to train U-net model in order to enhance classification.

2.1 RGB datasets

The images were captured with my laptops web camera with resolution 1280 x 720 pixels. For both datasets I used preprocessing with the use of OpenCV [48] Python library.

Preprocessing included function *getPerspectiveTransform*, which does the perspective transformation. Straight lines will remain straight even after the transformation. For this operation we need 4 points of the input image which were added manually. Resulting images contains only the cropped keyboard with two octaves of the piano.

All images were also rescaled down to 256 x 256 px in order to save disk space. I assumed that the task does not need higher resolution based on the resolution of famous datasets. For example MNIST database [49], containing handwritten digits, has only 28 x 28 pixel resolution. Another famous dataset called CIFAR-10 [50] contains 60 000 32x32 colour images.

2.1.1 Classification dataset

The problem I was solving is multi-label classification [51]. Result algorithm should be able to classify whether one or more key of the piano is pressed. To obtain balanced dataset, I would require all classes with all labels present and that would be 2^{24}, because every key on two octave piano could be pressed or not.

Therefore I created a dataset containing only pressed keys separately, one class was for every tone in the keyboard and one class with label *None*. I cap-

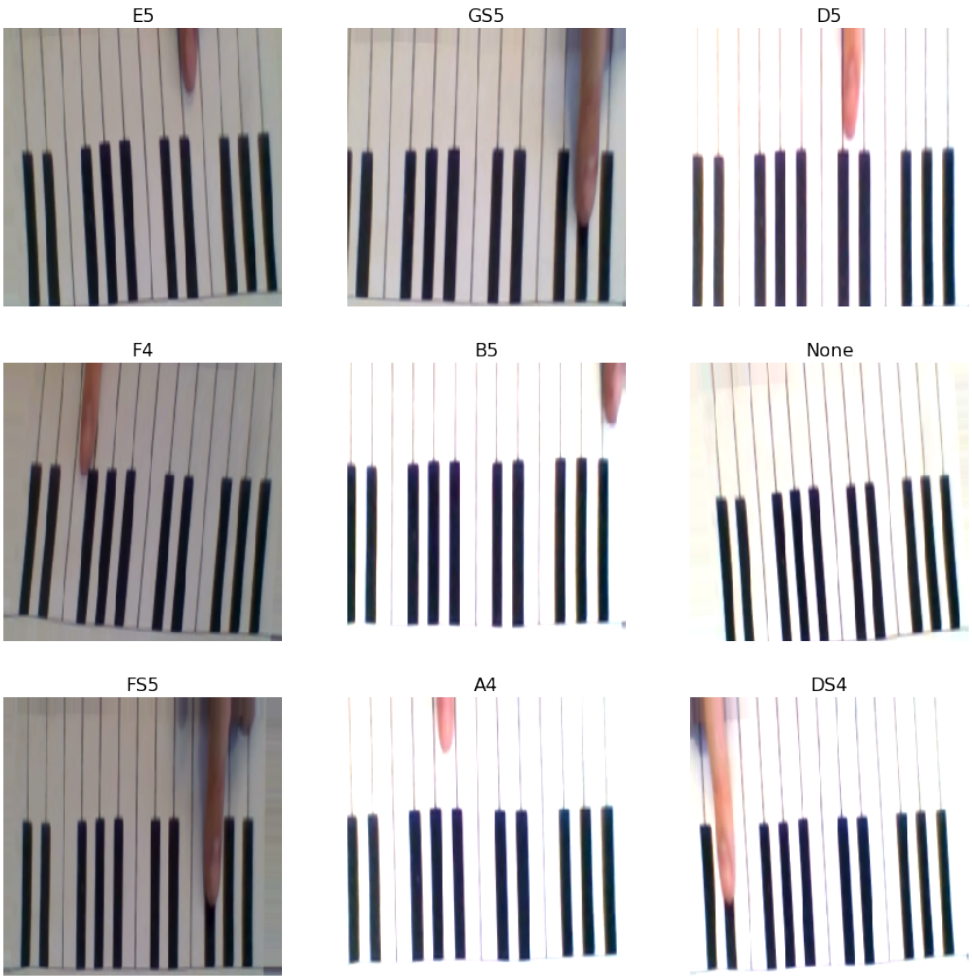


Figure 2.1: Classification dataset after image augmentation

tured 20 000 training data images (800 images per class) and 5 000 testing images (200 images per class).

Before I fed this data in the neural network I also used image augmentation in order to prevent overfitting of the neural network [7, chapter 8.3.4]. For the image augmentation I used *ImageDataGenerator* from tensorflow [52] library. You can see the augmented and labeled images in figure 2.1. Augmented images are generated in the runtime of training the neural network.

I performed the augmentation of the resulting images with the following operation:

- Rescaling pixels in the image between 0 and 1 (to feed this values into NN),
- random rotation in range of 5 degrees,

- width and eight shift range 0.05,
- brightness range of 0.6 to 1.4 of original brightness,
- zoom range of 0.1.

2.1.2 Masking dataset

Second dataset I created was the masking dataset. This one contained 5 000 original images and 5 000 masked images. The masked images were created by using following masking techniques.

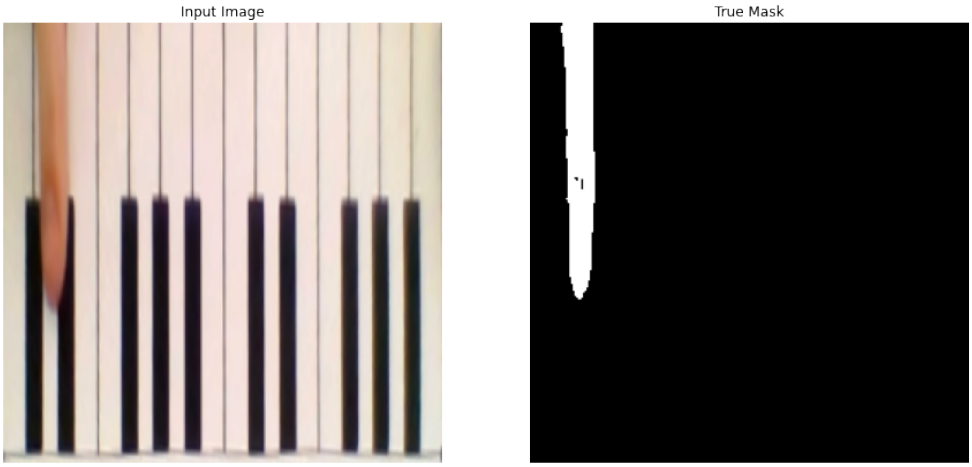


Figure 2.2: Masked dataset

First, I programmed an OpenCV [48] program to switch the RGB image into HSV one. Then I programmed 6 sliders to pick the correct range of thresholds for hue, saturation and value separately. With the correct values, I was able to mask out the finger from the original image. You can see an example of one image in figure 2.2.

This dataset is used in the U-net neural network. Image augmentation is also used here but only on the original RGB image. I used only the brightness range between 0.6 to 1.4, also in order to prevent overfitting of the U-net.

2.2 Designs

To achieve best results I proposed several neural networks architecture designs. This section is divided into RGB camera designs with neural networks, Leap Motion design and MediaPipe design which is again RGB approach.

2.2.1 RGB camera neural network designs

Proposed architecture is similar to VGG Net[41]. Sequential model by tensorflow library – six convolutional layers with ReLU as activation function and 3 x 3 kernel size, max pooling layers with pool size 2 x 2, dropout and two batch normalizations. Then flattening the neural network and four hidden layers of fully-connected NN. The output layer has 25 classing and is using softmax as activation function.

2.2.1.1 RGB images only

First neural network architecture used only RGB images without the mask. Resulting design is:

1. Convolution layer, 16 filters, input shape of 256 x 256 x 3
2. Batch normalization + Max pooling
3. Convolution layer, 32 filters + Max pooling
4. Dropout 0.2
5. Convolution layer, 64 filters + Max pooling
6. Convolution layer, 128 filters + Batch normalization + Max pooling
7. Convolution layer, 256 filters + Dropout 0.3 + Max pooling
8. Convolution layer, 512 filters + Dropnout 0.3 + Max pooling
9. Flattening
10. Fully-connected NN, 2048 nodes
11. Fully-connected NN, 1024 nodes
12. Fully-connected NN, 256 nodes
13. Fully-connected NN, 64 nodes
14. Fully-connected output layer with 25 nodes

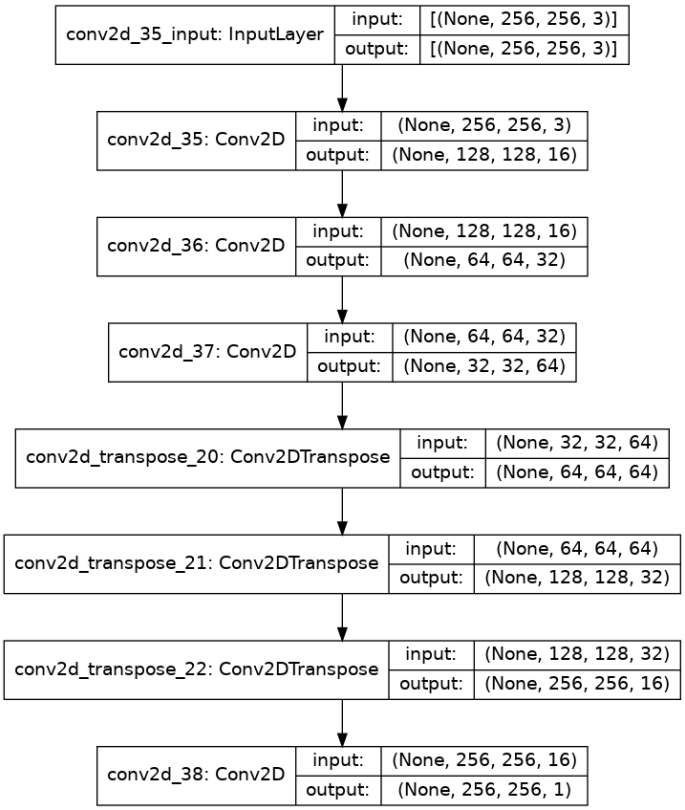


Figure 2.3: U-net model

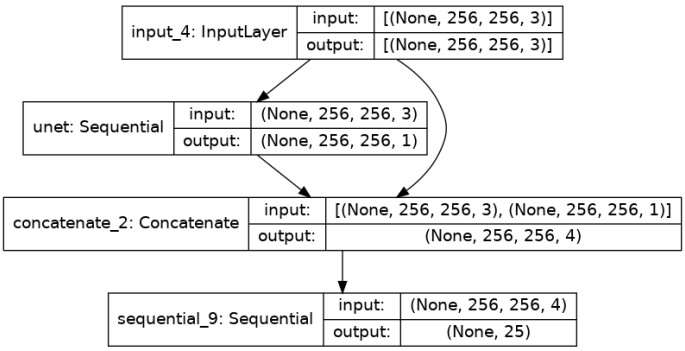


Figure 2.4: Merged model

2.2.1.2 RGB images + mask images

To mask finger every time in order to enhance classification accuracy, I could have used simple traditional techniques as when I created the dataset. But I could not ensure the same light conditions or for example same shape or

size of the finger playing the piano. To get the mask, I used an U-net neural network architecture.

I created the U-net NN that you can see in figure 2.3. After training, this neural network should be able to distinguish finger in each frame and enhance finger localization and overall accuracy. The main goal of this U-net is to encode important data into smaller feature space and then decode it into the preprocessed mask. In my case, I expected it to encode the color variable of the finger, because it is different than the black and white piano notes. This is the reason I also used the image augmentation on the mask dataset with brightness range.

I concatenated the output of the U-net with the previous model but with input shape of 256 x 256 x 4 (3 channels for RGB values and 1 channel for the mask of the finger). Another difference is that the fully-connected layer with 2 048 is missing. This is only done to not have such a deep model. The dropout rate 0.2 was also added to every Convolution layer with exception of last three layers, where the dropout rate was 0.5.

2.2.1.3 LSTM architecture

The key idea of using long short term memory architecture is that the input images are coming as a time series. During last years these networks have become the state-of-the-art in fields such as speech recognition, handwriting recognition [9, 53, 54]. For classification purposes I used combination of LSTM architecture and Convolution neural networks from Keras library [55] (a neural network library on top of TensorFlow) called *ConvLSTM2D*.

The input for this type of NN is a 5D tensor with shape *(samples, time, channels, rows, cols)*. That is why I created a sequence generator derived from tensorflows *ImageDataGenerator*. I had to switch change the *ImageDataGenerators* parameter *shuffle* to *False* in order to get the resulting sequence and not mixed sequence with different keys.

I tried several architectures, with one convolution LSTM layers (which combines the functionality of LSTM and convolution) and then several normal convolution layers. An example of one used LSTM architecture is:

1. ConvLSTM2D layer, 16 filters
2. Batch normalization + Max pooling 3D
3. Convolution layer, 20 filters + Max pooling
4. Convolution layer, 20 filters + Max pooling
5. Convolution layer, 20 filters + Batch normalization + Max pooling
6. Flattening

7. Fully-connected NN, 1024 nodes
8. Fully-connected NN, 256 nodes
9. Fully-connected NN, 64 nodes
10. Fully-connected output layer with 25 nodes

2.2.1.4 Multi-model architecture

Previous architectures with softmax activation function in the last layer could only predict one class, therefore only one tone could be played at the time. There are several possibilities to predict more than one class. First one is by using sigmoid activation function in the last layer to get the probability of each class. Then we would probably need also to predict how many fingers are there in the picture.

Another possible solution is to split the last fully-connected layers into branches in order to predict each class separately. You can see an example architecture for three tones in figure 2.5.

This type of architecture needs to have specified loss function for every output. It had 24 loss functions in total, one for every tone. The outputs had sigmoid as activation function to predict if the probability whether the key is pressed or not based on the filters passed from the last convolutional layer.

2.2.2 Leap Motion design

For the Leap Motion part, I will use Stereo IR 170 Camera Module provided by my supervisor. The teacher also provided a C++ library to communicate with the sensor called MultiLeap. However, all implementation of my thesis is in language Python, so using package *ctypes* I am able to connect the C++ library with the main part.

The MultiLeap library from my supervisor provides an array of points of each fingertips (one hand or both hands). If the point is not visible by the Leap Motion sensor, it returns point [0,0,0], which is the origin point in the center of the sensor so the finger should never be there.

Then I collected the four points on the piano with proposed calibration like in previous designs. But this time, corners of the piano need to be calibrated by the fingers to get precise position of the finger with consideration of location of the Controller.

After that, I computed for each fingertip the distance from the calibrated corner points to get the location of the finger. With this approach, I should be able to get the relative position of the fingers with respect to the keyboard no matter how the Stereo IR 170 Camera Module is placed (as long it stays static).

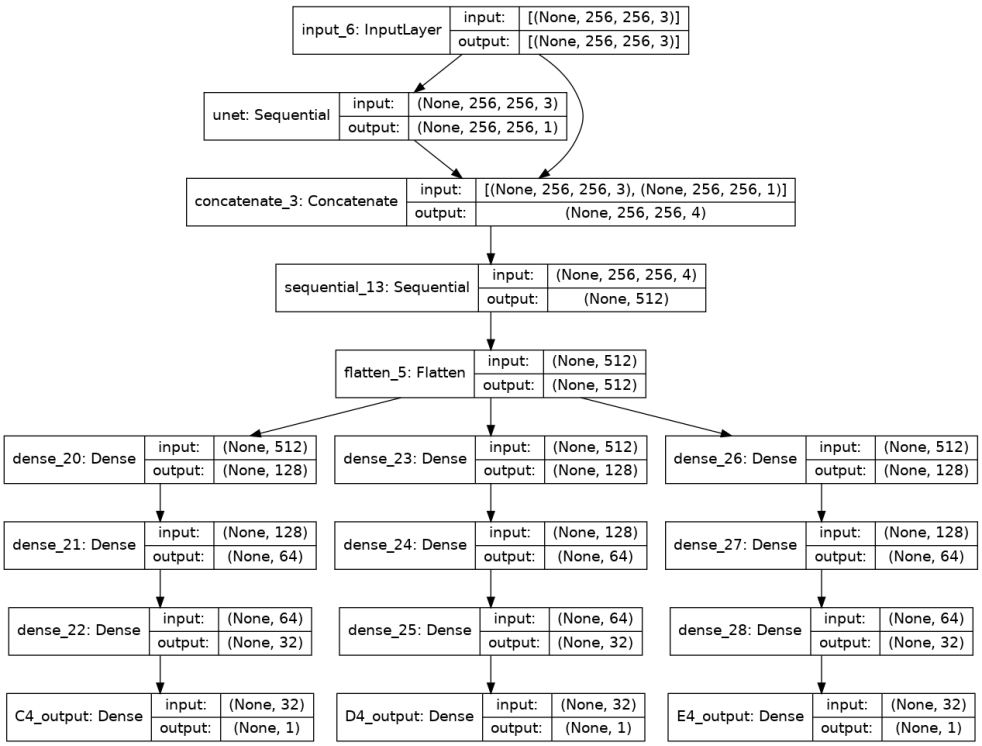


Figure 2.5: Multi-model architecture

Then I defined regions of tones. If the users fingertips are in these regions, then specified tone should play based. I also used The sliding window algorithm in order to improve the accuracy of the design.

This designs should work well for any angle of the controller, because user can calibrate, where the printed piano is placed. However, if the controller is facing down, the piano should work the best, because the Leap Motion controller recognizes have the best angle to get all information about hands. However, the Stereo IR 170 Camera Module is optimized to face up in the desktop mode or face front with headset mode.

2.2.3 MediaPipe design

For this design I used the MediaPipe library for Python. The API is really good quality with simple documentation and examples how to use the API in Python language with the help of the OpenCV library.

The MediaPipe works on the CPU so it does not need any special computation equipment or expensive GPU.

My design is to predefine regions of the played notes in the image of the piano. The same image is also used to be printed out. Each note has its own region. So I created an calibration script, where user can define his own regions.

These regions are then stored as list of points defining polygons in a JSON file.

The crutial part of the desing consists of calulating two perspective transformation matrices. One matrix is used for presentation purposes where user can see if he correctly chose the four points on the printed out piano. The second matrix is there to compute new coordinates of the fingertips in the output image with predefined regions of tones.

After the captured fingertip point is transformed into the piano image with the regions, I check if any of the fingertips belong to any predefined polygon using the Python *shapely* library.

The tones to play are then calculated from the previous information and are used for the music playing part of the software.

This designs should work well for any angle of the camera because user can calibrate, where the printed piano is placed. However, if the camera is facing down, the piano should work the best because the MediaPipe library will recognizes the position of the fingers more accurately.

Chapter 3

Implementation and results

For the implementation I used Python programming language and following libraries and packages:

- TensorFlow – Open source machine learning library [52]
- Keras – Deep learning framework [55]
- NumPy – Library for scientific computing in Python [56]
- OpenCV – Library for real-time Computer Vision tasks [29, 48]
- LibROSA – Python package for music and audio analysis [57]
- PyGame – Set of Python modules designed for writing videogames. In the work it is used for music playing [58]
- Matplotlib – Library for creating visualizations in Python
- MultiLeap library – Windows C++ Library for Stereo IR 170 provided by my supervisor [23, 22]
- Mediapipe Hand tracking module – Open source cross-platform library used CV tasks.

The main workflow for all types of models is similar. First, the camera (or IR Stereo 170 Camera Module) is opened and the capture window is opened. The user has to click with the cursor on four corners of the printed piano paper sheet in the input image. With this four points, using OpenCV library I computed the transformation matrix M. With this matrix I transform the input image and show the transformed image in the window. Then the transformation matrix N is computed and this matrix is used to find coordinates of transformed point on the keyboard,

After that, the prediction of the tone or tones are made based on the approach. I processed the prediction note with the Sliding window technique.

The basic idea in the sliding window algorithm is to form a window over some part of data, and this window can slide over the data to capture different portions of it and perform some computations on it. It is one of the approaches how to handle time series [59].

Algorithm 1 Sliding window algorithm example

```
arr ← [...] {An empty array}
thr ← 5
while CONDITION do
    new_value ← get_new_value_function()
    APPEND new_value TO arr
    window ← LAST thr VALUES OF arr
    compute_mean(window) {Use window in computations}
end while
```

These sub-arrays could be used in other computations as searching for maximum in the sub-array, summing of these values, counting and more. The counting can be used to enhance the accuracy of predicting certain tone of the piano based on the input images represented by time series.

This approach causes that the tones to play are stored in a queue and if the same tone repeats more than specified threshold, then the tone is played. With each new frame it is possible to get prediction of the tones to play, but if the same tone repeats in the sliding window more than specified threshold, then it is played.

The threshold can be set high, the piano will play more slowly and there can be even a delay but the tone is more accurate. Or it can be set as a low number, then the slightest movement over the piano keyboard will be registered as a tone to play.

3.1 Deep Learning approach

The prediction of all deep learning approaches is similar. The image is fed into the trained neural network. The prediction output is an array of probabilities whether the tone should play.

Some of the deep learning approaches did not work as I expected. That is why I tested four different architectures in order to discover whether I am able to enhance the overall performance. The architectures are CNN with RGB images, CNN with RGB + mask images, recurrent neural network – the LSTM and multi-model architecture. All models were trained on GPU NVIDIA GeForce RTX 3060.

3.1.1 RGB images

First model that I tried is deep CNN described in 2.2.1.1. With this approach, I was obtaining high accuracy (around 97 %), but in some results one or two tones were predicted wrong in almost all cases. I assumed that it was caused by the pattern repetition, as there are two octaves.

This is why I designed the approach with RGB mask and the autoencoder.

3.1.2 RGB images with mask

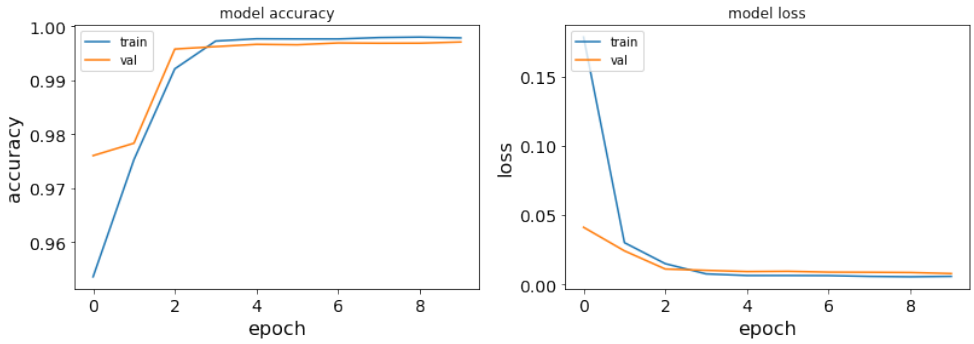


Figure 3.1: Masked dataset

Second model that I tested is the one described in 2.4.

First I trained the U-net model to get the mask of the finger from the original image. With this mask, the CNN should be able to better localize the finger on the image.

In order to achieve it, I created a python generator object to load both training RGB image and the masked image from the dataset. The training dataset was split into training data and validation data. The validation data were used check how the model is learning and to tune the hyperparameters.

As optimizer for the training I chose Adam [60], which is an algorithm for first-order gradient-based optimalization of stochastic objective function. It is now widely used for many deep learning models because it achieves decent results fast.

It combines the best properties of the AdaGrad [61] and RMSProp [62] algorithms to provide an optimalization that can handle sparse gradients on noisy problems. The core idea is that you define the learning rate as usual and then define a decay rate which shifts the learning rate after each epoch.

Output activation function for the U-net model is a sigmoid in order to get the probability if each pixel is bright or not. As a loss function I chose *binary_crossentropy*, which computes the cross-entropy loss between true labels and predicted labels [52].

I then trained this U-net for 30 epochs but after around 5 epochs the model was not improving and the results were really high. You can see the loss and

accuracy of the training process in figure 3.1. On the testing data, I have got **99.6 %** accuracy. For the training I used slight brightness range in the RGB image to prevent overfitting.

The result images of the U-net model are in the figure 3.2. As you can see, the Convolutional U-net model was able to predict the mask very well. The images are smoother than the original mask created by traditional image processing and even some holes in the mask are filled.

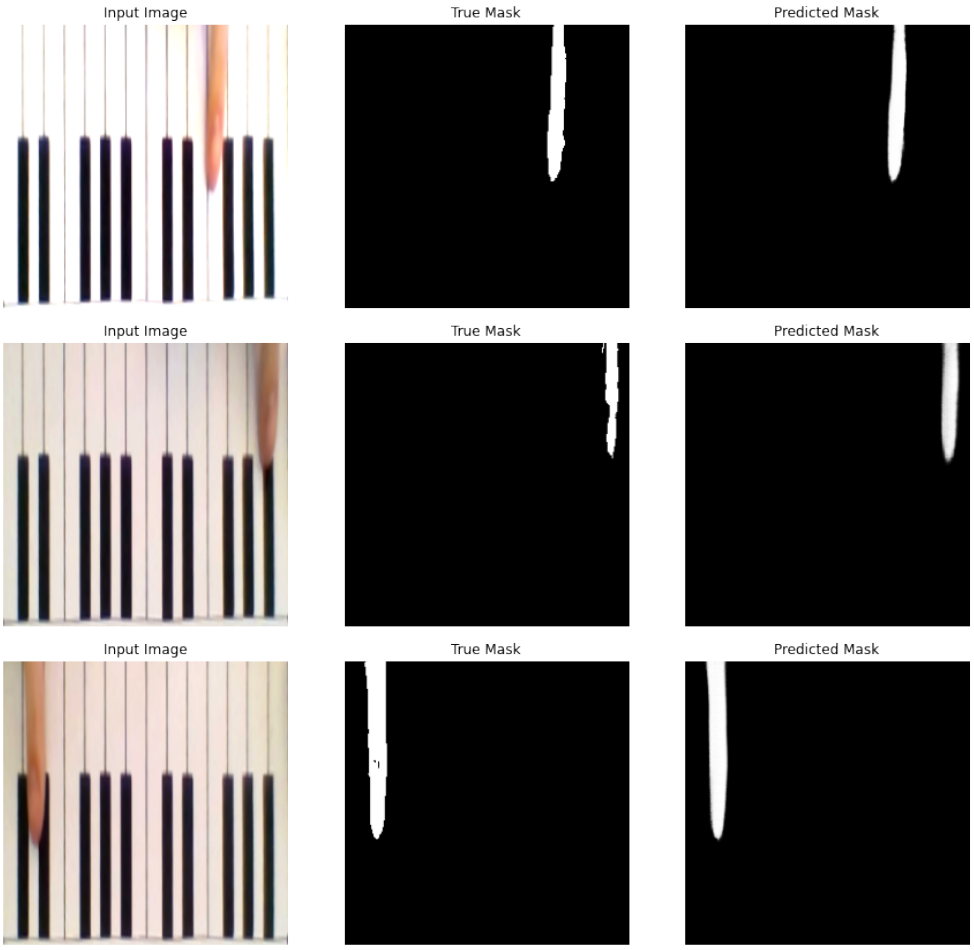


Figure 3.2: U-net finger masking result

With the U-net trained, I was able to use the architecture of CNN with mask described in 2.4. First, I wanted the U-net use it as it is and not train the weights further. That is why I used the transfer learning technique [45] where I set the layers in the U-net model to not train anymore and lock the weights.

Then, I trained this architecture with the proposed dataset containing 25 classes. With softmax activation function I could get one tone played at

the time from the prediction.

First, I trained the model for 30 epochs, 32 images in one batch, 80 steps in one epoch. These 30 epochs lasted around 170 seconds each. Then I trained the model even more for 30 epochs with 563 steps. Each epoch lasted around 14 minutes of training with my GPU NVIDIA GeForce RTX 3060.

As the accuracy metrics I chose *categorical_accurary* and as loss function I selected *categorical_crossentropy*. The training accuracy and loss are in figures 3.3 and 3.4.

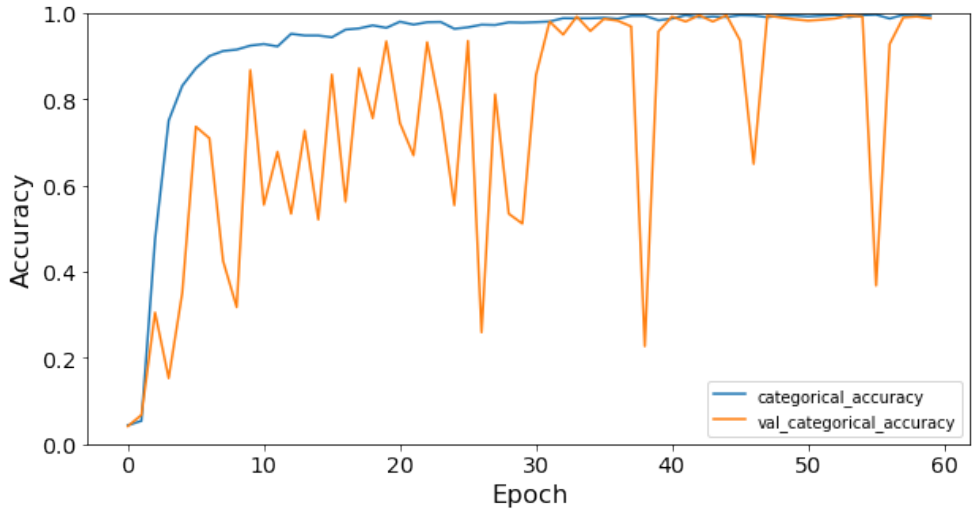


Figure 3.3: CNN accuracy

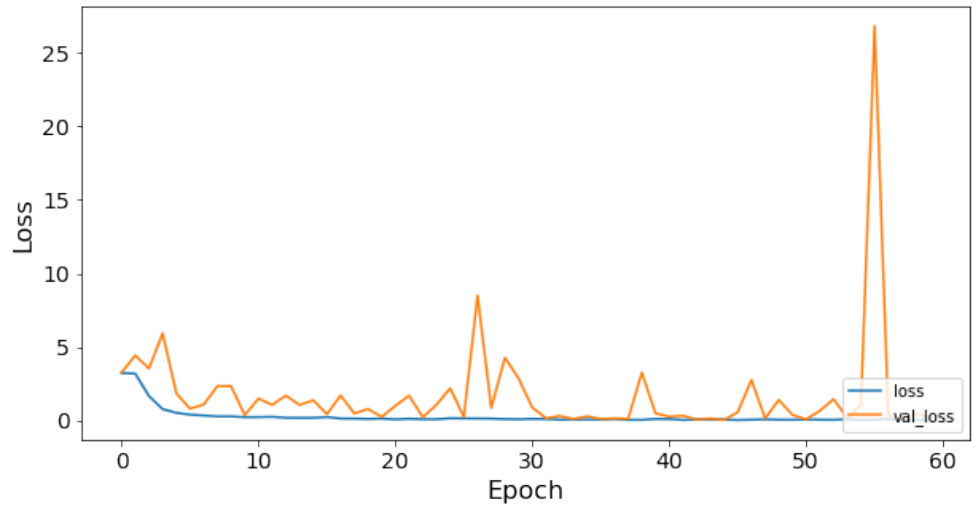


Figure 3.4: CNN loss

I was little bit concerned with the spiking validation loss, but it is a common unavoidable consequence of mini-batch gradient descent in Adam optimizer with batch size of 32 [63]. I also used only 30 (of around 100 possible ones) validation steps to save some time in the training process. After 60 epochs, the training loss converged and the model was not improving anymore.

In the end, I evaluated the resulting model on the test data and got categorical accuracy of **99.93 %**.

I tested this model and with the sliding window technique I managed to create a decent piano, where the user can play one note at the time.

3.1.3 LSTM architecture

For the proposed LSTM architecture I had to create a generator of the sequences derived from Tensorflows *ImageDataGenerator*. I set the argument *shuffle* to False to get the images in the same order as they were captured.

From the sequence of consequent images I labeled the images with the last class. I tried this approach before testing the sliding windows algorithm. My idea was to put a sequence of images in the LSTM to predict the class.

The training of the recurrent neural network took around 24 hours, then the accuracy was not improving anymore. For the LSTM model I got around 95% accuracy, but for the piano application, this model was not suitable.

There was a lot of preprocessing, such as concatenating all images into one big NumPy array. First, I tried sliding window of 32 images but the results were slow too much. Then I created a sliding window of 8 input images concatenated together. With every new image, I dropped the last image, shifted the array and append the new image. With NumPy library, I assumed it would be fast enough, but it was not.

Even with the input of eight images, all of the preprocessing of the images and computation of the LSTM architecture made the resulting piano very slow and the FPS dropped to around 1 frame per second or lower.

3.1.4 Multi-model architecture

Next neural network approach is multi-model architecture, where each tone is predicted separately. The multi-model design for three notes is described in figure 2.5. The main idea behind this architecture is that I am predicting every tone separately. The network should be able to predict more tones if they are played simultaneously.

However, as I described in Design section, I have only data for predicting one tone at the time. For multi-label classification I would be necessary to make dataset containing all combination of tones pressed or not to have balanced dataset and that is $2\hat{2}5$.

Every output had sigmoid as activation function to get the probability, whether the tone is played or not. The metrics used was *BinaryAccuracy* and loss function for each output was *BinaryCrossentropy*

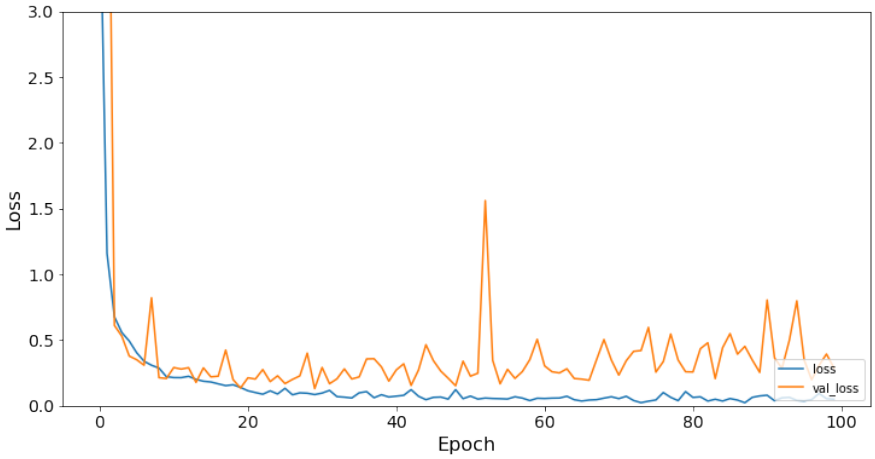


Figure 3.5: Multi-model loss

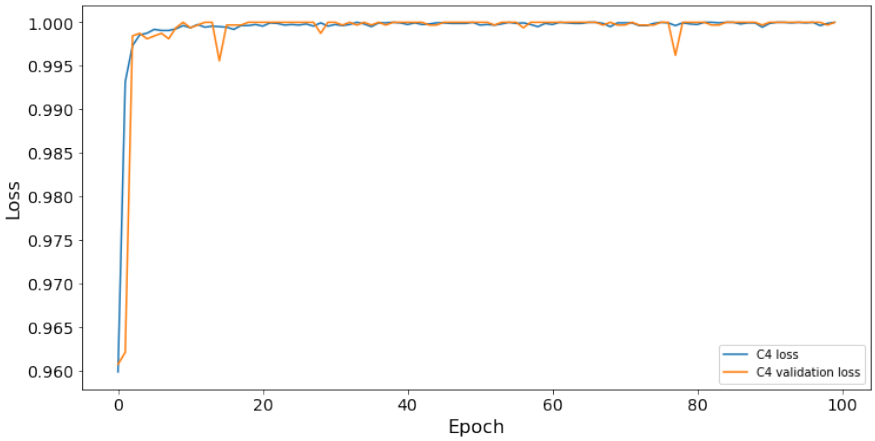


Figure 3.6: Multi-model C4 output accuracy

3.2 Leap Motion approach

In this section, I will describe my tests of the piano with the Stereo IR 170 Camera Module by Leap Motion. I used a C++ library provided by my supervisor. The library enables to connect to the controller and also to get fingertip location of each frame. However, the main part of the algorithm is written

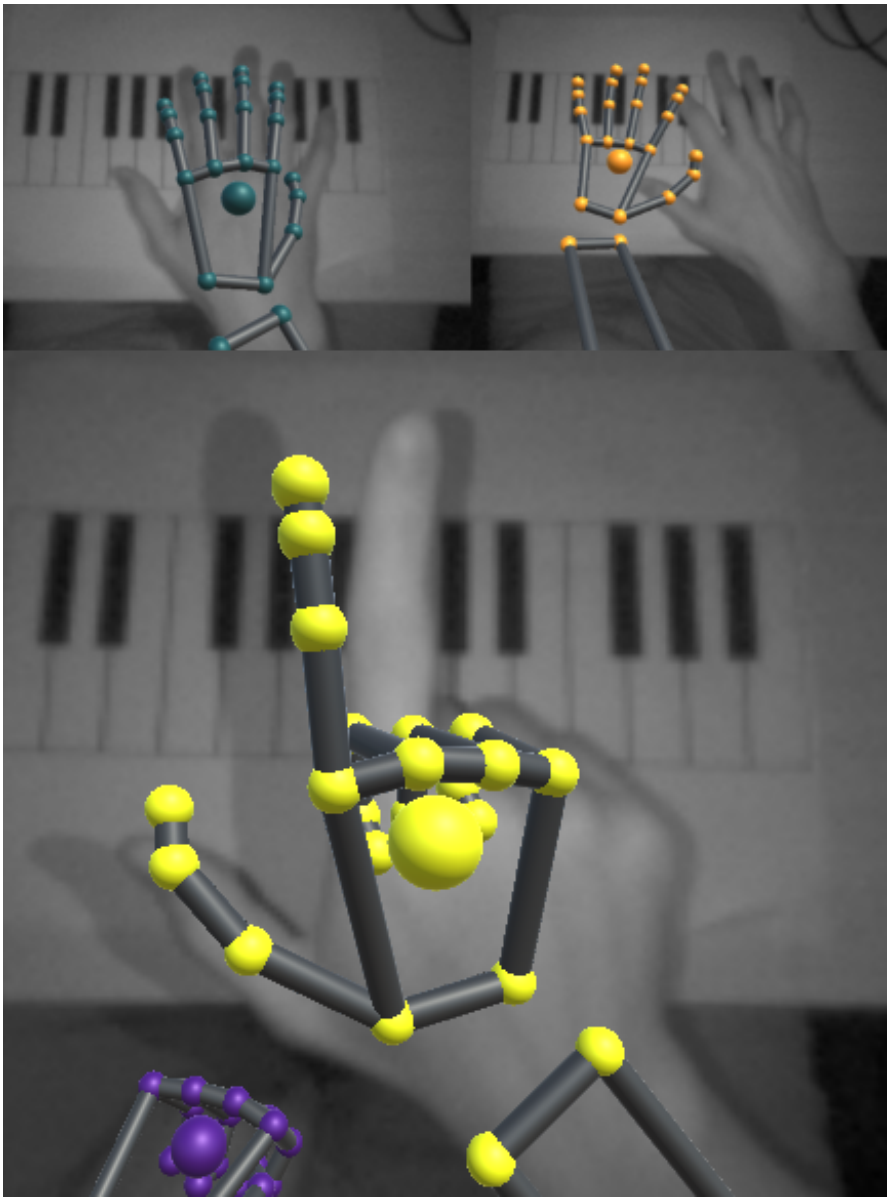


Figure 3.7: Error in the Leap motion approach, switching hands (top left), Classifying piano as hand (top right), Bad fingertip position (bottom)

in language Python, so I used Python package *ctypes* in order to use the C++ API.

The connection itself works very well when the Stereo IR 170 Camera module was placed on my desk facing up. The data about fingertips were correct – this was using the *Desktop Mode*. I also tried the Headset mode which is optimized for VR headsets.

First setup that I tested was to attach the controller above my web camera of my laptop. The results were sometimes glitchy as the camera in desktop mode is optimized to face up from the desk. These glitches were for example switching left and right hand or inaccurate fingertip location.

Then I tilted the laptop, so it can see the piano like in the previous approaches and then the glitches were even stronger. With the help of my supervisor, we tested both modes (Desktop and Headset) with and without the printed out keyboard and came to conclusion, that the printed piano is causing the glitches. You can see an example glitch in figure 3.8 from Leap Motion Visualizer where you can see that the finger looks like pressed even though it is not. Also in both of the images is visible that the parts of the hand are slightly shifted. The glitches are probably caused by the black and white pattern which could resemble fingers.

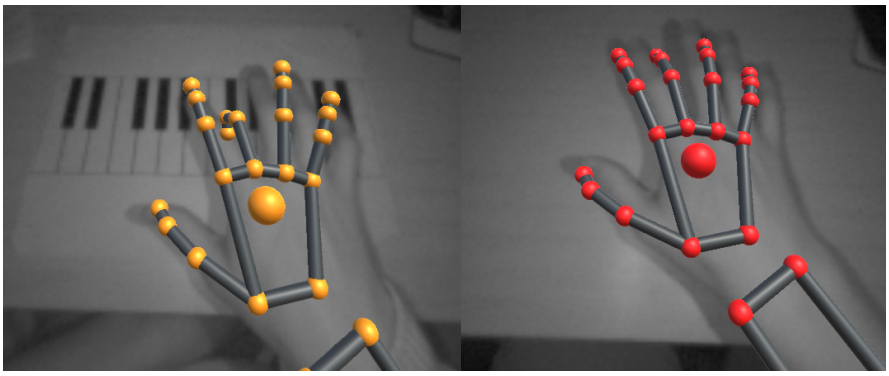


Figure 3.8: Leap Motion with and without hand over the keyboard

Another test consisted of attaching the Stereo IR 170 Camera module above the keyboard with slight tilt. The setup consisted of 25cm high holder and the tilt was around 30 degrees. Idea behind this setup is to resemble controller attached to a headset.

This test led to better results, however some glitches remained mainly with the printed out paper piano under the hands.

Despite the problem, I implemented the calibration process, where 100 000 points are collected for each corner, when the finger is pointed there during the calibration. Then the average between these points is calculated get the corners of the piano.

However I have run into many problems. Sometimes during the calibration process, the Stereo IR 170 Camera Module stopped tracking my hand on the printed piano, so I had to repeat the process. I repeated this process twice for Desktop mode where I collected 10x4 corner points with the proposed setup above the hands. But the results were very inaccurate. As you can see in figure 3.9, there are four colors representing the four corners of the piano. However, in some calibrations, which should be very accurate with

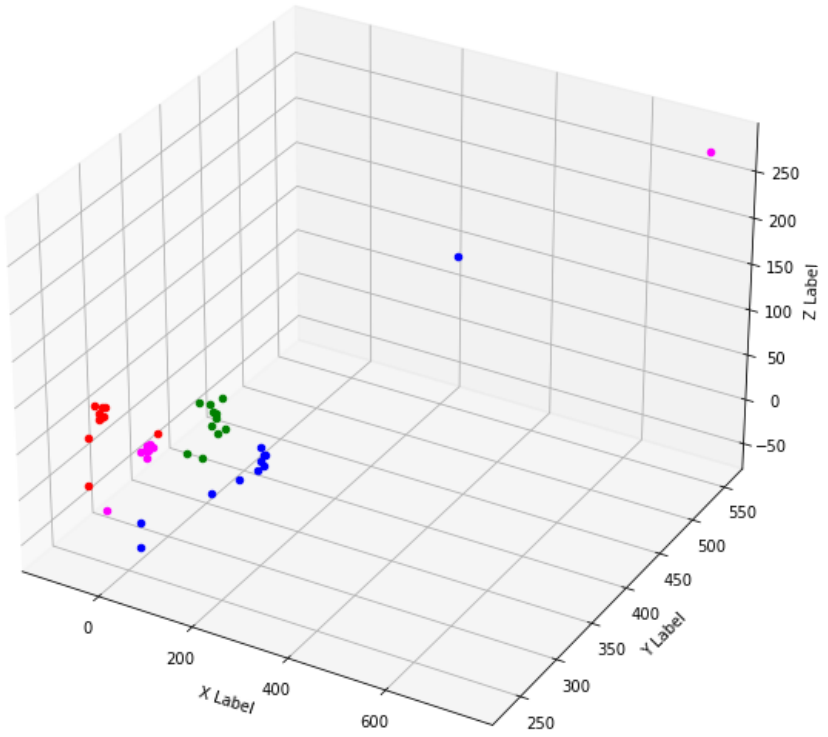


Figure 3.9: Leap Motion piano calibration process with Desktop Mode

the controller I got some outliers laying 10 or 20 centimeters further. The points resemble something similar polygon (which is the expected output), but these points are sometimes around 2 centimeter apart which is crutial for the piano application.

The Headset mode brought better results, but still not good enough for the purposes of the piano. As you can see in figure 3.10, the points are representing a polygon, but also the with average error around one centimeter. The piano calibration process was done with the right index finger and the controller had most of the issues with top left corner of the piano.

With all encountered errors, I propose that Leap Motion Stereo IR 170 Camera Module is not suitable for my piano application as it is. Mostly because the software is optimized for hand gesture tracking and not for fingertip tracking. Also it is now mostly used for virtual reality application where the user can adjust the position of the fingers based on perception what he sees. On the other part the desktop mode is optimized to be placed on surface and track hands gestures from below. Also the printed out paper piano significantly corrupted the accuracy of the sensor as you can see in figure 3.8 and

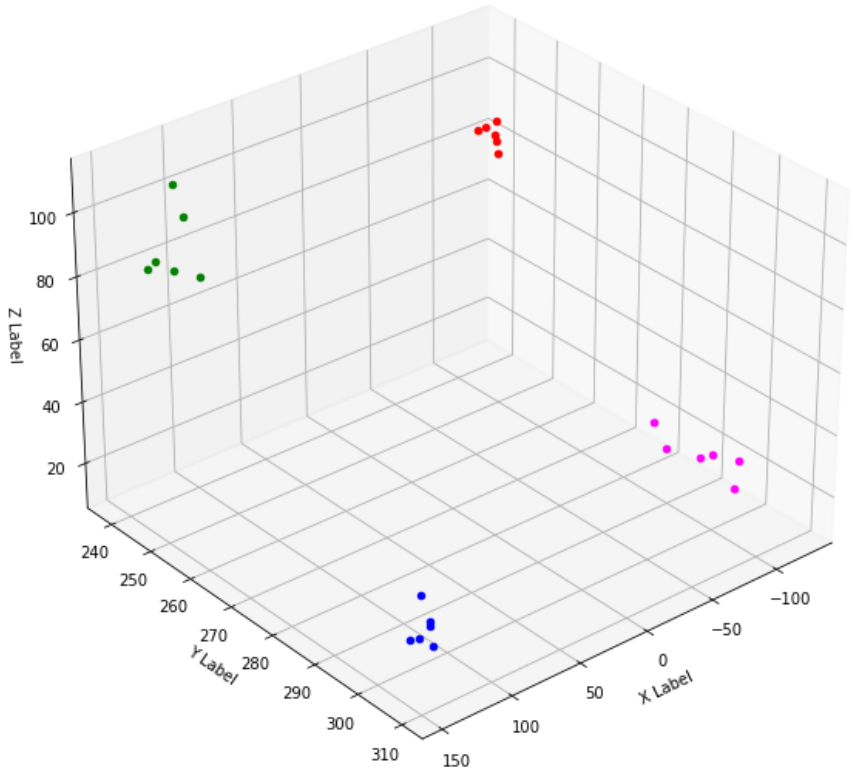


Figure 3.10: Leap Motion piano calibration process with Headset Mode

also later in accuracy comparison section 3.4. The readers may ask themselves whether the printed piano is necessary and I say it is really helpful to interact with something simple such as printed piano sheet. It is simpler than swinging finger in the air and picturing the piano in head or see the piano only on the screen. However, if the user has a direct feedback of what he is doing with the virtual fingers for example in virtual reality, then the user could adapt to the finger movement.

3.3 Mediapipe approach

The MediaPipe approach works the best. Its pretrained models worked very precisely. According to MediaPipe documentation, they achieve an average precision of 95.7 % in palm detection. Using a regular cross entropy loss and no decoder in their model gives a baseline of just 86.22 %.

The researchers experimented with different model sizes to achieve the best trade-off between quality and speed. Increasing model complexity achieves

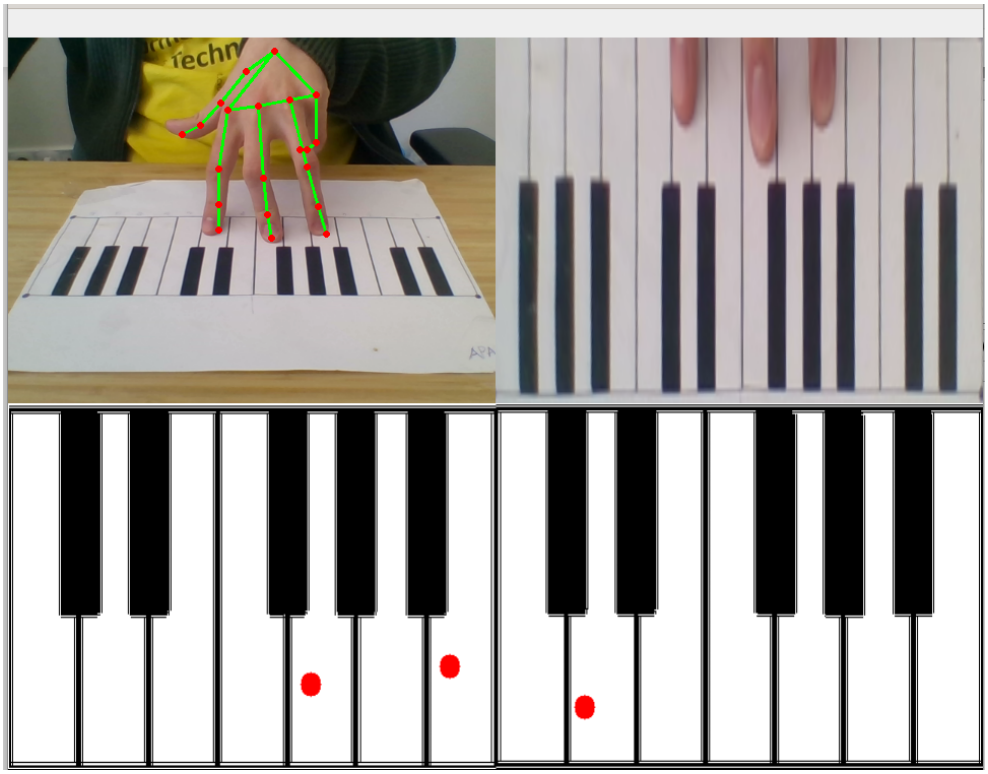


Figure 3.11: Resulting piano

only minor improvements in quality but decreases significantly speed [28]. For example, Light model with 1 million of parameters got 11.83 MSE and 6.6ms in Pixel 3 smartphone. Full model with 1.98 million parameters got 10.05 MSE and 16.1ms. The Heavy model with 4.02 million parameters managed to achieve 9.817 MSE but the time complexity was 36.9ms. According to the article, "Full" model provides a decent trade-off between quality and speed.

For the piano application the default value from the API was chosen. I implemented the designed pipeline, where two transformation matrices M and N are used. Transformation matrix M is used in OpenCV perspective transformation for visualization purposes that user can see if he clicked correctly with cursor on the four corners of the piano. Second matrix N is used to transform all fingertip points to new coordinates on the piano.

These points are then evaluated if they are in any defined polygon. Then a list of indexes is passed to a function which transform the indexes into the list of tones to play. These tones are then included into the *SlidingWindow* class.

This class decides which tones will start play and which ones have to stop. If a number of same tones that are passed in the following frames is

higher than specified threshold, then the tone is to play. If the number of the tones drops below the threshold, this tone is now labeled to stop playing. For my implementation I chose default threshold as 3 frames.

There could cause some problems. If the camera cannot see full hand or the hand is crooked, there are sometimes glitches. However, it can be fixed by placing camera to see all fingers and hands.

In the end, the user can choose if the output is stored as a MIDI file. In that case, it is necessary to specify beats per minute (BPM) because the software itself cannot recognize in which key the user plays and also the tempo of user playing. However, the MIDI file is stored in the *.mid* format that user can take and upload for example to MuseScore [64] application, where you can edit your output further. For example the user can change tempo of the notation, transpose the notes, switch to different instrument etc.

3.4 Accuracy comparison between Leap Motion and MediaPipe

To compare accuracy between the MediaPipe approach and Leap Motion approach. I used the same piano calibration as I used with the Stereo IR 170 Camera Module. For Leap Motion approach I collected ten times 100 000 points for each corner and then computed the mean.

I have collected two sets for the Leap Motion – with the printed out paper piano and without it (only with same locations of the corner points with blank paper sheet). This is done to compare the accuracies with and without the printed piano whether the assumption was right or not.

For MediaPipe I collected ten times 100 points to calculate mean for each corner of the piano. The smaller number of the points collected is because Leap Motion work with higher framerate and even with this number difference the MediaPipe collection took longer. Because the coordinates for MediaPipe is relative, the points were stretched to the shapes of the real piano in millimeters in order to compare the results from MediaPipe and Leap Motion.

As a measurement I calculated the euclidean distances between all ten points in all approaches. The distances were calculated for both approaches 2D and 3D. An ideal result would be that the distances between these points are zero, because the points each represents one corner of the piano.

3D calibration – mean distance	0	1	2	3	mean
Leap Motion without printed piano	10.02	7.67	8.05	7.46	8.30
Leap Motion with printed piano	18.01	8.32	25.37	18.93	17.66
MediaPipe with printed piano	17.95	8.40	8.89	16.27	12.88

Table 3.1: Accuracy comparison for Leap Motion and MediaPipe in 3D

2D calibration – mean distance	0	1	2	3	mean
Leap Motion without printed piano	4.87	5.23	5.86	6.06	5.51
Leap Motion with printed piano	17.52	6.64	15.76	14.20	13.53
MediaPipe with printed piano	1.74	1.14	1.11	0.90	1.22

Table 3.2: Accuracy comparison for Leap Motion and MediaPipe in 3D

As you can see in these tables, the euclidean distance between the calibration points (points 0 to 4) are similar in 3D, where MediaPipe library in only calculating the Z axis based on the 2D image. Leap Motion without printed piano got an error of mean 8.3 millimeters. The MediaPipe approach with 12.88mm was still more accurate than Leap Motion with printed out piano paper sheet with 17.66mm. The mean error are for 2 all of these three is bigger than one centimeter which can be crutial to recognize played note.

However, if we only calculate the error distance in 2D, the MediaPipe outperform the Leap Motion with only 1.22 mm mean error. For the MediaPipe it was used the x and y coordinates and for the Leap Motion x and z as they represents width and depth. The error for Leap Motion approach without printed piano got in 2D 5.51 mm mean error and with the piano 13.53 mm.

The maximum error measured was Leap Motion with printed piano and the error was 6.36 centimeters.

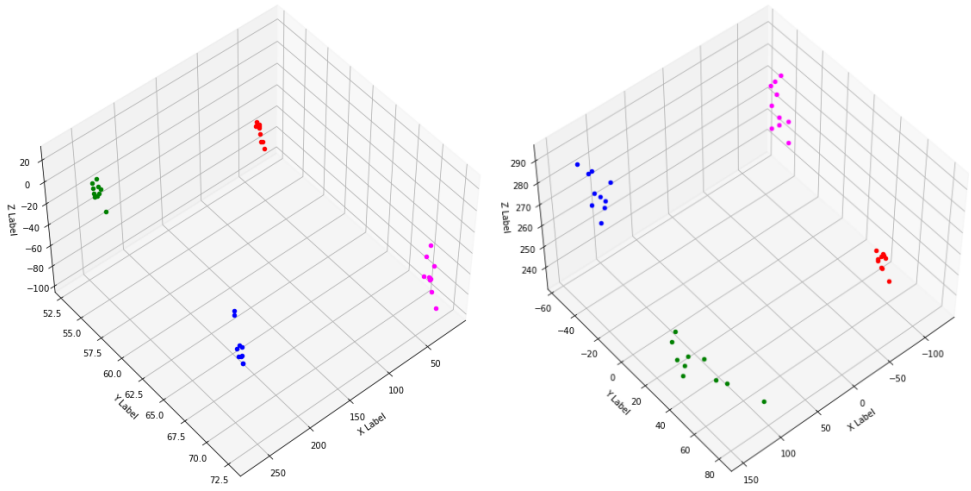


Figure 3.12: Collected calibration points, MediaPipe (left) and Leap Motion (right)

Conclusion

In my thesis, I analysed virtual piano keyboards, controllers, hand gesture tracking methods, hand pose recognition and concluded that Neural Networks are now the state-of-the-art method in these types of recognition. Until now all virtual piano keyboards are mostly for VR or using approaches where the user cannot see or interact with anything but the piano on the screen.

I proposed semi-virtual approach where the user can type on piano printed out on paper sheet. Using computer vision techniques in combination with state-of-the-art deep learning approaches the fingers are detected and the position of the fingertips are predicted. With the fingertip detection a simple software to play tone is used to imitate real piano.

For my work I created dataset consisted of 25 000 256 x 256 images of fingers playing notes. I also created dataset of 5 000 similar images where the finger are masked out to enhance performance of the neural networks.

In this work, I tested several approaches. Four deep convolutional neural networks approaches, one with only RGB images, second using RGB images with pretrained U-net model similar to autoencoder architecture, third one using recurrent neural network (LSTM) and fourth model with more than one outputs.

The neural network with pretrained U-net model performed very well with average accuracy of 99.6 % on test data. This model can be used for the final piano, but it could only play one note at the time. The LSTM model did not performed sufficiently because the model complexity was high and the frame preprocessing was also too complex and the frame rate drop for the piano was significant.

Next approach was by using Stereo IR 170 Camera Module by Leap Motion, which is projecting a pattern and computing finger location based on the distortion of that pattern. The test for this approach did not went well. The sensor is highly optimized for gesture recognition but not precise fingertip detection. With tested both modes (desktop and headset). I came to conclusion that Leap Motion is not suitable for this type of proposed semi-virtual piano.

Also this approach could be only used in Windows platform where the Leap Library is supported. The accuracy of the Leap Motion approach was also highly disrupted by the printed out paper piano and the comparison of these accuracies is described in the work.

The best results were achieved by using MediaPipe library by Google, where pretrained models are used to detect palm and then detect 21 3D hand regions including fingertips. This library, in combination with OpenCV perspective transformation and sliding window algorithm was able to play more tones at the time and performed very well. The result of the piano can be stored as MIDI file which can be played in other software (e.g. MuseScore). The result of my thesis is the research about possibilities of virtual piano that is easy to install and play. I need to add one last thing in the end that first song ever played on this virtual piano is "Jak se loví gorila" by Petr Skoumal also known as the Apakrychle [65] (APA) song.

Future work with my thesis is possible. It is possible to enhance accuracy wherher the tone is pressed using the height dimension with the MediaPipe or with other 3D sensor. The piano could be also used to interpret not only the piano tones but also different instruments or even define own region and use it as a typing keyboard.

Bibliography

[1] Ruffle, J. K. M. B.; Farmer, ., Adam D. PhD FRCP1; et al. Artificial Intelligence-Assisted Gastroenterology— Promises and Pitfalls. *The American Journal of Gastroenterology*, volume 114, 2019.

[2] Smits, T.; Wevers, M. The visual digital turn: Using neural networks to study historical images. *Digital Scholarship in the Humanities*, volume 35, 01 2018, doi:10.1093/llc/fqy085.

[3] Weichert, F.; Bachmann, D.; et al. Analysis of the accuracy and robustness of the leap motion controller. *Sensors*, volume 13, no. 5, 2013: pp. 6380–6393.

[4] Lugaresi, C.; Tang, J.; et al. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172*, 2019.

[5] Camuñas-Mesa, L.; Linares-Barranco, B.; et al. Neuromorphic Spiking Neural Networks and Their Memristor-CMOS Hardware Implementations. *Materials*, volume 12, 08 2019: p. 2745, doi:10.3390/ma12172745.

[6] Bre, F.; Gimenez, J.; et al. Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks. *Energy and Buildings*, volume 158, 11 2017, doi:10.1016/j.enbuild.2017.11.045.

[7] Aggarwal, C. C. *Neural Networks and Deep Learning*. Cham: Springer, 2018, ISBN 978-3-319-94462-3, 497 pp., doi:10.1007/978-3-319-94463-0.

[8] Tabian, I.; Fu, H.; et al. A Convolutional Neural Network for Impact Detection and Characterization of Complex Composite Structures. *Sensors*, volume 19, no. 22, 2019, ISSN 1424-8220, doi:10.3390/s19224933. Available from: https://www.mdpi.com/1424-8220/19/22/4933

[9] Olah, C. Aug 2015. Available from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[10] Thakur, D. ML Basics #4: Replace Negatives with Zeros! Sep 2019. Available from: https://dhruvs.space/posts/ml-basics-issue-4/

[11] Liang, H.; Wang, J.; et al. Barehanded Music: Real-time Hand Interaction for Virtual Piano. 2016. Available from: https://dl.acm.org/doi/abs/10.1145/2856400.2856411

[12] Matsui, N.; Yamamoto, Y. A New Input Method of Computers with One CCD Camera: Virtual Keyboard. January 2001. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.8059&rep=rep1&type=pdf

[13] Du, Y.; Liu, S.; et al. Hand Gesture Recognition with Leap Motion. November 2017. Available from: https://arxiv.org/abs/1711.04293

[14] Marin, G.; Dominio, F.; et al. Hand gesture recognition with leap motion and kinect devices. In *2014 IEEE International Conference on Image Processing (ICIP)*, 2014, pp. 1565–1569, doi:10.1109/ICIP.2014.7025313.

[15] Aristidou, A.; Lasenby, J. Motion capture with constrained inverse kinematics for real-time hand tracking. In *2010 4th International Symposium on Communications, Control and Signal Processing (ISCCSP)*, 2010, pp. 1–5, doi:10.1109/ISCCSP.2010.5463419.

[16] Kölsch, M.; Turk, M. Keyboards without keyboards: A survey of virtual keyboards. In *Workshop on Sensing and Input for Media-centric Systems, Santa Barbara, CA*, 2002.

[17] Tran, D.-S.; Ho, N.-H.; et al. Real-time hand gesture spotting and recognition using RGB-D camera and 3D convolutional neural network. *Applied Sciences*, volume 10, no. 2, 2020: p. 722.

[18] Biswas, S.; Hazra, R. Robust edge detection based on Modified Moore-Neighbor. *Optik*, volume 168, 2018: pp. 931–943.

[19] Qiao, W.; Wei, R.; et al. A real-time virtual piano based on gesture capture data. In *2017 12th International Conference on Computer Science and Education (ICCSE)*, IEEE, 2017, pp. 740–743.

[20] Obermaier, B.; Muller, G. R.; et al. " Virtual keyboard" controlled by spontaneous EEG activity. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, volume 11, no. 4, 2003: pp. 422–426.

[21] Du, H.; Oggier, T.; et al. A virtual keyboard based on true-3D optical ranging. In *Proceedings of the British Machine Vision Conference*, volume 1, 2005, pp. 220–229.

[22] Novacek, T.; Jirina, M. Overview of controllers of user interface for virtual reality. 2020. Available from: Tobepublished

[23] Potter, L. E.; Araullo, J.; et al. The Leap Motion Controller: A View on Sign Language. In *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, OzCHI '13, New York, NY, USA: Association for Computing Machinery, 2013, ISBN 9781450325257, p. 175–178, doi: 10.1145/2541016.2541072. Available from: https://doi.org/10.1145/2541016.2541072

[24] Szeliski, R. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[25] Sebe, N.; Cohen, I.; et al. *Machine learning in computer vision*, volume 29. Springer Science & Business Media, 2005.

[26] Raj, H. RGB image representation. 2019. Available from: https://www.geeksforgeeks.org/matlab-rgb-image-representation/

[27] G., R. Integrated Feature Extraction for Image Retrieval. 11 2017.

[28] Zhang, F.; Bazarevsky, V.; et al. MediaPipe Hands: On-device Real-time Hand Tracking. *arXiv preprint arXiv:2006.10214*, 2020.

[29] Bradski, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[30] Loy, G. Musicians make a standard: the MIDI phenomenon. *Computer Music Journal*, volume 9, no. 4, 1985: pp. 8–26.

[31] Moore, F. R. The dysfunctions of MIDI. *Computer music journal*, volume 12, no. 1, 1988: pp. 19–28.

[32] Rothstein, J. *MIDI: A comprehensive introduction*, volume 7. AR Editions, Inc., 1992.

[33] Rizo, D.; De León, P. J. P.; et al. A Pattern Recognition Approach for Melody Track Selection in MIDI Files. In *ISMIR*, 2006, pp. 61–66.

[34] Li, X.; Chen, H.; et al. H-DenseUNet: hybrid densely connected UNet for liver and tumor segmentation from CT volumes. *IEEE transactions on medical imaging*, volume 37, no. 12, 2018: pp. 2663–2674.

[35] Weng, Y.; Zhou, T.; et al. NAS-Unet: Neural architecture search for medical image segmentation. *IEEE Access*, volume 7, 2019: pp. 44247–44257.

[36] Chen, W.; Liu, B.; et al. S3D-UNet: separable 3D U-Net for brain tumor segmentation. In *International MICCAI Brainlesion Workshop*, Springer, 2018, pp. 358–368.

[37] Zeng, Z.; Xie, W.; et al. RIC-Unet: An improved neural network based on Unet for nuclei segmentation in histology images. *Ieee Access*, volume 7, 2019: pp. 21420–21428.

[38] Hubel, D. H.; Wiesel, T. N. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, volume 148, no. 3, 1959: pp. 574–591.

[39] LeCun, Y.; et al. LeNet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, volume 20, no. 5, 2015: p. 14.

[40] Das, S. Nov 2017. Available from: https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

[41] He, K.; Zhang, X.; et al. Deep Residual Learning for Image Recognition. *CoRR*, volume abs/1512.03385, 2015, 1512.03385. Available from: http://arxiv.org/abs/1512.03385

[42] Szegedy, C.; Liu, W.; et al. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[43] Krizhevsky, A.; Sutskever, I.; et al. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, volume 25, 2012: pp. 1097–1105.

[44] Informatik, F.; Bengio, Y.; et al. Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies. *A Field Guide to Dynamical Recurrent Neural Networks*, 03 2003.

[45] Torrey, L.; Shavlik, J. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, IGI global, 2010, pp. 242–264.

[46] Pan, S. J.; Yang, Q. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, volume 22, no. 10, 2009: pp. 1345–1359.

[47] Srivastava, N.; Hinton, G.; et al. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, volume 15, no. 1, 2014: pp. 1929–1958.

[48] Itseez. *The OpenCV Reference Manual*. Second edition, April 2014.

[49] LeCun, Y.; Cortes, C. MNIST handwritten digit database. 2010. Available from: http://yann.lecun.com/exdb/mnist/

[50] Krizhevsky, A.; Nair, V.; et al. CIFAR-10 (Canadian Institute for Advanced Research). 2009. Available from: http://www.cs.toronto.edu/~kriz/cifar.html

[51] Tsoumakas, G.; Katakis, I. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, volume 3, no. 3, 2007: pp. 1–13.

[52] Abadi, M.; Barham, P.; et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[53] Gers, F. A.; Schmidhuber, J.; et al. Learning to forget: Continual prediction with LSTM. 1999.

[54] Greff, K.; Srivastava, R. K.; et al. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, volume 28, no. 10, 2016: pp. 2222–2232.

[55] Chollet, F.; et al. Keras. 2015. Available from: https://github.com/fchollet/keras

[56] Harris, C. R.; Millman, K. J.; et al. Array programming with NumPy. *Nature*, volume 585, no. 7825, Sept. 2020: pp. 357–362, doi:10.1038/s41586-020-2649-2. Available from: https://doi.org/10.1038/s41586-020-2649-2

[57] McFee, B.; Raffel, C.; et al. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, 2015.

[58] Shinners, P. PyGame. http://pygame.org/, 2011.

[59] Chu, C.-S. J. Time series segmentation: A sliding window approach. *Information Sciences*, volume 85, no. 1, 1995: pp. 147–173, ISSN 0020-0255, doi:https://doi.org/10.1016/0020-0255(95)00021-G. Available from: https://www.sciencedirect.com/science/article/pii/002002559500021G

[60] Kingma, D. P.; Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[61] Duchi, J.; Hazan, E.; et al. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, volume 12, no. 7, 2011.

[62] Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[63] xboard (https://stats.stackexchange.com/users/10220/xboard). Explanation of Spikes in training loss vs. iterations with Adam Optimizer. Cross Validated, uRL:https://stats.stackexchange.com/q/304150 (version: 2018-12-04), https://stats.stackexchange.com/q/304150. Available from: https://stats.stackexchange.com/q/304150

[64] Shinn, M. *Instant MuseScore*. Packt Publishing Ltd, 2013.

[65] Valenta, M.; Novacek, T.; et al. Apakrychle. year n+1. Available from: https://apakrychle.ninja/

Appendix A

Acronyms

AI Artificial intelligence

APA Apakrychle

API Application programming interface

BPM Beats per minute

CCD Charge-coupled device

CNN Convolutioanl neural network

CPU Central processing unit

CV Computer vision

DoF Degrees of freedom

DL Deep learning

EEG Electroencephalography

FPS Frames per second

GPU Graphics processing unit

HCI Human-computer interaction

HOG Histogram of oriented gradient

HSV Hue, saturation, value

JSON JavaScript Object Notation

LED Light-emitting

LSTM Long short term memory

MIDI Musical Instrument Digital Interface

ML Machine learning

MSE Mean Squared Error

NN Neural network

PCA Principal component analysis

px pixel/pixels

RGB Red, green, blue

RoI Region of interest

RNN Recurrent neural network

SMF The Standard MIDI format

SVM Support vector machine

TCP Transmission Control Protocol

VR Virtual reality

Appendix B

Contents of enclosed CD

- `README.md` ..... the file with the description and installatio
- `play_piano.py` ..... Python script with the piano
- `utils.py` ..... Python file with used python functions
- `NN_jupyter_notebooks` ..... Jupyter notebooks with with experiments
- `tests` ..... the directory with tests
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `Leap Motion assets` ..... Leap Motion assets
- `environments` ..... environments.yml with used Python packages
- `audio_files` ..... the directory with audio assets