Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelor Thesis

Neural Networks on Microsoft HoloLens 2

Léon Lazar

Course of Study:	Informatik (Computer Science)
Examiner:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Supervisor:	Johannes Kässinger, M.Sc.
Commenced:	November 5, 2020
Completed:	May 5, 2021

Abstract

The goal of the present Bachelor thesis is to enable comparing different approaches of integrating Neural Networks in HoloLens 2 applications in a quantitative and qualitative manner by defining highly diagnostic criteria. Moreover, multiple different approaches to accomplish the integration are proposed, implemented and evaluated using the aforementioned criteria. Finally, the work gives an expressive overview of all working approaches. The basic requirements are that Neural Networks trained by TensorFlow/Keras can be used and executed directly on the HoloLens 2 without requiring an internet connection. Furthermore, the Neural Networks have to be integrable in Mixed/Augmented Reality applications. In total four approaches are proposed: TensorFlow.js, Unity Barracuda, TensorFlow.NET, and Windows Machine Learning which is an already existing approach. For each working approach a benchmarking application is developed which runs a common reference model on a test datatset to measure inference time and accuracy. Moreover, a small proof of concept application is developed in order to show that the approach also works with real Augmented Reality applications. The application uses a MobileNetV2 model to classify image frames coming from the webcam and displays the results to the user. All the feasible approaches are evaluated using the aforementioned evaluation criteria which include ease of implementation, performance, accuracy, compatibility with Machine Learning frameworks and pre-trained models, and integrability with 3D frameworks. The Barracuda, TensorFlow.js and WinML approaches turned out to be feasible. Barracuda, which only can be integrated in Unity applications, is the most performant framework since it can make use of GPU inference. After that follows TensorFlow.js which can be integrated in JavaScript Augmented Reality frameworks such as A-Frame. Windows ML can currently only use CPU inference on the HoloLens 2 and is therefore the slowest one. It can be integrated in Unity projects with some difficulties as well as plain Win32 and UWP apps. Barracuda and Windows Machine Learning are also integrated in a biomechanical visualization application based on Unity for performing simulations. The results of this thesis make the different approaches for integrating Neural Networks on the HoloLens 2 comparable. Now an informed decision which approach is the best for a specific application can be made. Furthermore, the work shows that the use of Barracuda or TensorFlow.js on the HoloLens 2 is feasible and superior compared to the existing WinML approach.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence. 23

ANN Artificial Neural Network. 15

API Application Programming Interface. 21

AR Augmented Reality. 13

CNN Convolutional Neural Network. 17

CV Computer Vision. 25

DNN Deep Neural Network. 17

GPU Graphics Processing Unit. 15

HMD Head-Mounted Display. 18

HPU Holographic Processing Unit. 18

IDE Integrated development environment. 24

ML Machine Learning. 13

MR Mixed Reality. 13

NN Neural Network. 13

ONNX Open Neural Network Exchange. 22

PerSiVal Pervasive Simulation and Visualization. 13

PoC Proof of Concept. 32, 33

PWA Progressive Web App. 25

ReLU Rectified Linear Unit. 16

RNN Recurrent Neural Network. 17

UWP Universal Windows Platform. 18

VR Virtual Reality. 17

wasm WebAssembly. 22

WinML Windows Machine Learning. 22

1 Introduction

Machine Learning (ML) has become ubiquitous in computer science. Especially Neural Networks (NNs) are currently the most important ML method. Nearly every tech company makes use of this technology. Even modern smartphones contain specialized chips for executing NNs efficiently such as the Neural Core in the Google Pixel 4 [Rak]. NNs are used for image recognition, translation, speech detection, spelling correction, and many more applications.

However, one field where the use of NNs is still not that common are Augmented Reality (AR) or Mixed Reality (MR) headsets. AR/MR is about augmenting the field of view with additional information or overlaying virtual 3D objects onto the real world. There are some smartphone apps such as Google Lens which make use of AR. However, dedicated AR/MR headsets such as the Microsoft HoloLens are the way to go since the hands can stay free. Possible applications for AR/MR are virtual collaboration, games, education, or just displaying information such as navigation directions while driving a bicycle.

NNs would offer great opportunities on AR/MR devices. For example classifying plants, animals or other objects the user is looking at and displaying the name of it would be relatively easy to implement with NNs. A trivial approach would be to run the NN on an external server which receives the input data from the headset and sends the result back via Wi-Fi. However, this approach has several downsides. The wireless connection is likely to be unstable and introduces latency. Furthermore, having a computer or a stable connection nearby is not feasibly for all areas of use. Examples would be construction sites or military applications. This shows that it is beneficial to be able to run NNs directly on the AR device.

Pervasive Simulation and Visualization (PerSiVal) which is the superordinate project of this Bachelor thesis, is about running biomechanical simulations on a Microsoft HoloLens. A virtual arm is overlayed onto a real arm. Colours visualize the activation of the muscles. It is planned to use NNs to calculate the muscle activation values in a very fist approach, but aiming the simulation of the deformation of these muscles as well.

Goal of this Bachelor thesis is to make comparing different approaches of integrating NNs in HoloLens 2 applications in a qualitative and quantitative manner possible. As a prerequisite highly diagnostic criteria such as ease of implementation or performance have to be defined. Moreover, multiple different approaches shall be proposed, implemented and evaluated using the aforementioned criteria. In order to be able to evaluate the different approaches a basic benchmark application as well as a Proof of Concept application shall be developed for each one. Finally, an expressive overview is given which compares of approach and describes the individual pros and cons of each one. Based on that an informed decision which approach is suitable for a specific application can be made. Furthermore, one approach will be implemented in the existing muscle visualization application of the PerSiVal project.

The Bachelor Thesis is structured as follows: After the Introduction the Background chapter gives a rough overview over the topics Machine Learning, Neural Networks, Microsoft HoloLens and the used software. This is followed by the Related Work chapter which summarizes existing research work related to this topic. After that the Problem Statement describes the concrete problem which is to be solved by this thesis. Subsequently, the evaluation criteria, the experiment design, and the different approaches are described. Lastly, the results are explained and evaluated using the criteria. Furthermore, the different approaches are compared.

2 Background

This chapter gives an overview about the fundamental topics for this thesis such as ML, NNs, the Microsoft HoloLens and the used software components.

2.1 Machine Learning

In computer science the traditional approach is to design an algorithm respectively a mathematical model to solve a specific computational problem. The downside is that this requires the acquisition of domain knowledge for example understanding the physics of a problem under study. Another approach is ML. In contrast to the conventional approach, ML usually requires training instead of understanding the problem in detail. The example data can be used as a training set to train a ML system to solve the anticipated task. There are three different ML techniques: Supervised learning, unsupervised learning and reinforcement learning. For supervised learning the training set needs to have input-output pairs. An example are email texts as input and boolean values indicating whether an email is a spam message. This kind of training set is called labelled data. Opposed to supervised learning, unsupervised learning only requires unlabelled data meaning inputs without assigned output values. Generally speaking the goal of unsupervised learning is to identify properties of the data. An example application would be clustering of documents with similar topics. Reinforcement learning requires some kind of feedback from the environment which evaluates the output of the system. The general application are sequential decision-making problems where the system sequentially takes actions (outputs) based on observations (inputs) and receives feedback after each chosen action. [Sim18]

2.2 Neural Networks

Deep Learning is one of the most important methods of supervised learning which makes use of NNs, sometimes also called Artificial Neural Networks (ANNs). NNs are inspired by the human brain and use the concept of neurons. Early computational models for artificial neurons were proposed in 1943 by Warren S. McCulloch and Walter Pitts [MP43]. An early NN called the perceptron was developed by Frank Rosenblatt in 1957 [Ros57]. In 1969 Arthur Bryson and Yu-Chi described backpropagation as an optimization method which is nowadays used to train NNs [BH69; PI95]. However, at first the advent of distributed computing and powerful Graphics Processing Units (GPUs) which enables the usage of larger NNs caused the popularity of NNs and Deep Learning in recent years [GBC16].

Neurons have one or multiple inputs as well as an output value which is called activation. Each input has an assigned weight value. Each neuron has a bias value. These values and the input values are put in a weighed sum. Equation 2.1 shows an example of such a weighted sum. The input value

x of each input is weighted using the corresponding weight w. To this weighted sum, the bias b of the neuron is added. [Nie15]

$$w_0 \cdot x_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + ... + w_n \cdot x_n + b_0 \tag{2.1}$$

The weighted sum is put into an activation function in order to normalize the value. The sigmoid function (Equation 2.2) used to be the most common one. However, also other functions such as Rectified Linear Unit (ReLU) (Equation 2.3) or tanh are possible. ReLU is nowadays the most common one. [Nie15]

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

$$\sigma(z) = max(0, z) \tag{2.3}$$

Multiple neurons form a layer. The outputs of the neurons of a layer are connected to the inputs of all neurons of the following layer. A NN always has an input layer by which it receives external data and an output layer which produces the eventual result. Between the input and output layers are zero or more so-called hidden layers. Figure 2.1 shows an exemplary NN with $n + 1$ inputs, l hidden layers and $o + 1$ outputs. [Nie15]

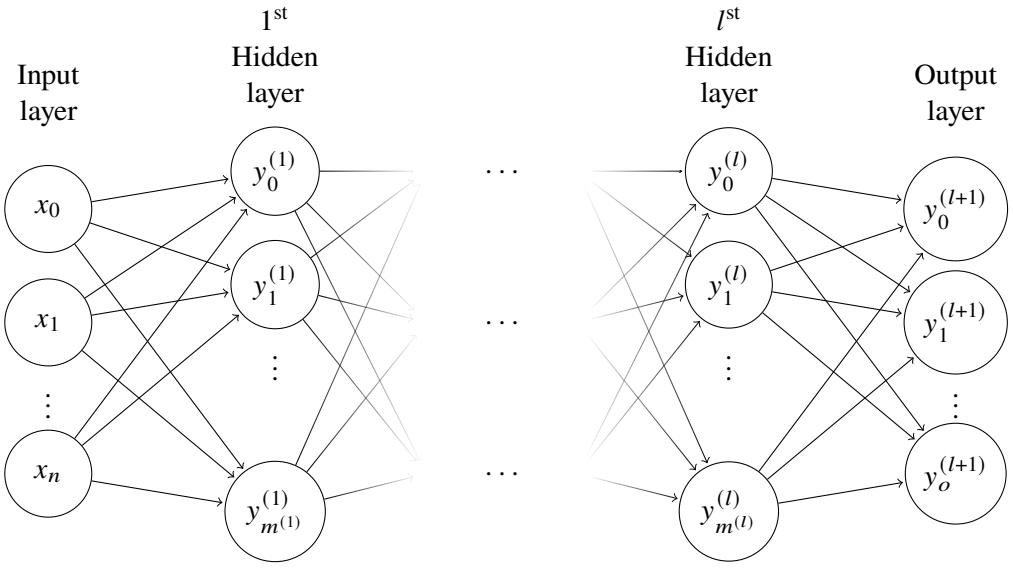

Figure 2.1: Neural Network with $n + 1$ inputs, l hidden layers and $o + 1$ outputs [Stu20]

Weights and activation values are the values which are actually learnt during the training of the NN. Goal of the training is to find values that the output of the NN approximates $y(x)$ (the desired output of a training example) for all training inputs x. To evaluate how well the NN accomplishes this task a cost function is used. Equation 2.2 is such a cost function. [Nie15]

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2 \tag{2.4}$$

b is a collection of the biases, w of weights, n is the number of training pairs, a is the vector of all output values for an input x and the sum is over all training inputs x. The goal is to find weights and biases to minimize the cost function. This can be achieved using a backpropagation algorithm which makes use of gradient descent. The algorithm iterates backwards layer-by-layer form the last layer while computing the gradient of the loss function respecting the weights for each layer. [Nie15]

The basic networks where the outputs from one layer are use as input for the subsequent layer are called feedforward neural networks. NNs with multiple hidden layer are often called Deep Neural Networks (DNNs). Information only flows forwards and never backwards since there are no loops. However, there are also NNs which allow to have feedback loops. Such NNs are called Recurrent Neural Network (RNN). [Nie15]

2.2.1 Convolutional Neural Networks

Another type of NNs, which are crucial for image classification and pattern recognition tasks, are Convolutional Neural Networks (CNNs). These NNs feature two additional types of layers: convolutional layers and pooling layers. The former calculates the output of neurons which are connected to regions of the input using convolution: the filter kernel glides step-by-step over the input. The output value is calculated using the scalar product of the kernel and the underlying section of the input. On that an activation function, usually ReLU (Equation 2.3) is applied to calculate the neurons final activation value. The values of the filters itself are determined during training. A convolutional layer can have multiple filters which extract different features. Each filter results in a different feature map. After the convolutional layers follow pooling layers which perform downsampling and reduce the number of parameters. Pooling layers are sometimes also called subsampling layers. The most common method is max-pooling which only keeps the maximum value of each 2x2 square. An alternative approach is average-pooling where the average value of each 2x2 square is used. Both approaches reduce the data by 75% which increases the performance and helps against overfitting. Lastly, CNNs also contain ordinary fully-connected layers like traditional NNs. For classification tasks the number of neurons usually corresponds to the number of classes. CNNs can contain multiple convolutional and pooling layers as well as multiple fully-connected layers at the end. [ON15]

Pixel values of images are usually represented as 8 bit integer, whereby the values range from 0-255. Many CNNs expect a range of 0-1, therefore the values have to be normalized accordingly. The reason for this is that the weight values are usually small values smaller than 1 and large integer inputs can slow down or rattle the training process. [Bro19]

2.3 Mixed Reality

In Virtual Reality (VR) users are immersed in an entirely synthetic world. VR and the real world are not antitheses, rather they are on opposite ends of a continuum. This concept, which was introduced by Milgram et al. in 1994, is called Reality-Virtuality continuum. Figure 2.2 visualizes the concept. The area between VR and the real world is called MR. It contains AR as well as augmented virtuality which means that real world objects are integrated in virtual worlds.

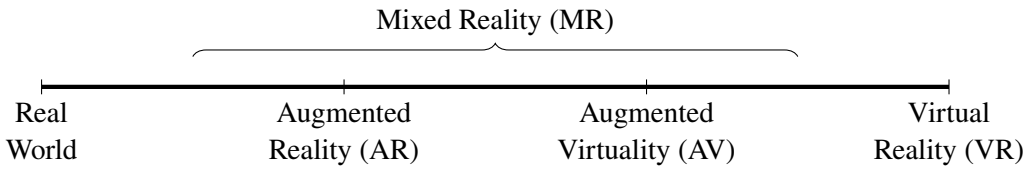

Figure 2.2: Reality-Virtuality continuum [MTUK95]

AR is about enriching the real world with virtual objects. This includes overlaying additional information over the real world using a smartphone or AR Head-Mounted Displays (HMDs) such as Google Glass. However, overlaying information on video streams showing the real world on a computer screen is also considered as AR. [MTUK95]

In practise, the differences between AR and MR are not always clear since both terms are sometimes used as synonyms. However, MR includes more than just AR according to the Reality-Virtuality continuum. Microsoft uses the term MR for experiences where virtual 3D holograms are inserted in the real world and coexist with physical objects. An example would be virtual models of furniture that can placed into real rooms in order to see how they fit. This requires sensors or cameras which detect the location of the MR device and nearby objects. [Bra20]

2.4 Microsoft HoloLens

The Microsoft HoloLens is a MR headset. MR is realized using see-through holographic lenses. The device can be controlled by gestures, voice and eye tracking. It can operate independently since it includes a computer. The operating system is Windows 10. There are two generations of the HoloLens. [ZMP19] Table 2.1 shows the specifications for both of them.

2.4.1 Microsoft HoloLens (1st gen)

The first generation of the HoloLens (shown in figure 2.3) was released in 2016. According to Microsoft it is the first untethered AR headset of the world. The internal computer is based on an Intel 32-bit architecture. Furthermore, it incorporates a Holographic Processing Unit (HPU) which is a custom-made coprocessor for processing sensor values and holograms [ZMP19]. The HoloLens (1st gen) is now in Long Term Servicing state. Therefore, it will not receive further updates other than bug and security fixes. The October 2018 update with build number 1809 is the latest available version for the HoloLens (1st gen). The most severe limitation of the old Windows version is that the HoloLens (1st gen) can only Universal Windows Platform (UWP) applications and no ordinary Win32 applications. Furthermore, up-to-date browsers such as the Chrome based Edge browser are not available. [Zel19]

Table 2.1: Comparision of the specifications of both HoloLens generations [CMP20; ZMP19]

	HoloLens (1st gen)	HoloLens 2
Release date	March 30, 2016	November 7, 2019
CPU	Intel 32-bit (1GHz)	Qualcomm Snapdragon 850 Compute Platform
Memory	2 GB RAM 1 GB HPU RAM	4 GB RAM
Storage	64 GB	64 GB
Display resolution (per eye)	1280×720	2048 × 1080
Field of view (FOV)	34°	52°
Camera	2.4 MP, HD video	8 MP, 1080p video
Microphones	Four channel array	Five channel array
Eye tracking	No	Yes
Biometric security	No	Yes (iris scan)
Hand tracking	One hand	Both hands
Connectivity	IEEE 802.11ac WiFi, Bluetooth LE 4.1, microUSB	IEEE 802.11 2x2 WiFI, Bluetooth LE 5.0, USB Type-C
Weight	579 g	566 g

Figure 2.3: HoloLens (1st gen)

2.4.2 Microsoft HoloLens 2

The HoloLens 2 (shown in picture 2.4) is the direct successor of the HoloLens (1st gen). It was released in 2019. Compared to the first generation it brings many improvements. It brings multiple new sensors to make eye tracking and tracking of both hands possible. The device is based on an ARM architecture in contrast to the Intel 32-bit architecture of the first generation [CMP20]. The HoloLens 2 still receives new Windows updates. The newer Windows versions enable the HoloLens 2 to run ordinary WIN32 applications next to UWPs. Furthermore, the new Chrome based Edge browser is available. [Zel19]

Figure 2.4: HoloLens 2

2.5 Software

This section covers the different software tools, frameworks, datasets, and models used in this thesis including multiple machine learning frameworks and graphics engines.

2.5.1 TensorFlow

TensorFlow is a framework for dataflow programming developed by Google. The main application is ML with emphasis on NNs. It was released in 2015 as an open-source package. TensorFlow is the successor of Google's DistBelief project. [AAB+16]

Directed graphs consisting of nodes describe TensorFlow computations. These directed graphs represent dataflow computations. Each node instantiates an operation and has zero or more inputs and zero or more outputs. Tensors are basically multi-dimensional arrays which are passed on edges from outputs to inputs. Operations represent abstract computations e.g. add or matrix multiply. Operations can have attributes which have to be provided at construction time. Kernels

are implementations of operations for specific types of devices such as CPUs or GPUs. Client programs interact with TensorFlow by creating a session. The session interface provides a run method which invokes the computations and an extend method which can be used to augment further nodes and edges to the current graph. One important feature of TensorFlow to point out is the automatic gradient computation which is required by many ML training algorithms. For example the minimization of the cost function during the training of NNs makes use of gradient computation. TensorFlow's low-level Application Programming Interfaces (APIs) is used to design computational graphs. The API is available in several programming languages including Python and C++. [AAB+16]

For designing NNs TensorFlow includes the high-level Keras API. In short, Keras provides an abstraction layer for defining NNs by its layer architecture. TensorFlow's computational graphs are still used under the hood. Section 2.5.3 describes Keras more detailed.

TensorFlow is available for several platforms including x86_64 systems running Linux or Windows, Android, Raspberry Pi and more. GPU acceleration is also supported on CUDA (mainly GPUs form Nvidia) enabled GPUs. Furthermore, community builds also support AMD ROCm capable GPUs. [Ten20a]

In 2019 TensorFlow 2.0 was released. The new version integrates Keras more tightly. Futhermore, the low-level API offers more possibilities to access the internal functions of TensorFlow. The SavedModel file format is more standardized and will replace the different formats of TensorFlow Lite and TensorFlow.js in the future. There are also several improvements in terms of training. TensorFlow 2.0 supports distributed training and Multi-GPU training. Now Python development with eager execution is the recommended way, even though the old Session-based model is still supported. Moreover, many APIs have been replaced or renamed. Code from TensorFlow 1 can be easily converted to TensorFlow 2.0 using an automatic conversion script. [Ten19]

Practical applications for TensorFlow are image object detection, classification, language translation, voice recognition, text analysis and many more. TensorFlow is used for in many Google products such as Search, GMail and Translate. However, many other companies such as Intel, CocaCola, Airbus, airbnb or China Mobile also make use of TensorFlow. [Tena]

2.5.2 TensorFlow.js

TensorFlow.js is a JavaScript implementation of TensorFlow intended to be used in web applications directly running in the browser. Like TensorFlow, it offers a high-level layers API which is similar to Keras as well as a low-level API.

TensorFlow.js can be used to create and train NNs. However, it is also possible to convert existing models from the TensorFlow SavedModel format or Keras HDF5 format into TensorFlow.js' JSON based format. TensorFlow.js can not only run in the browser but also be used with JavaScript based server-side systems such as React Native or Node.js. [Ten20b]

TensorFlow.js is able to make use of different backends for storage of the tensors and mathematical operations. There is a normal CPU backend which works on all browsers and just uses vanilla JavaScript for calculating and a GPU backend which uses WebGL. It stores the tensors as WebGL textures and WebGL shaders are used to implement the mathematical operations. TensorFlow.js chooses automatically the best available backend. So if WebGL is automatically chosen if it

is available on a system. The WebGL backend can be up to 100 times faster than the normal CPU backend. Furthermore, TensorFlow.js also has a WebAssembly (wasm) backend which can accelerate CPU calculations. [Tenb]

Since the usage of web technology TensorFlow.js runs on nearly every modern web capable device including mobile devices and ordinary computers. An interesting usage example of TensorFlow.js is the Teachable Machine from Google. It is a web application which allows even laypersons to train neuronal networks directly in the browser with only a few clicks. Currently, image classification, pose detection, and audio classification are supported. The user has to record or upload samples for each class and Teachable Machine trains a NN which can be downloaded in the TensorFlow.js, Keras HDF5, TensorFlow SavedModel or TensorFlow Lite format. [Goo]

2.5.3 Keras

Keras is a high-level API for NNs. It is written in Python and can use TensorFlow, Theano, and CNTK as backends. TensorFlow is the default one, but it can be changed to the other ones. Keras' ambition is to provide a modular, user-friendly and extensible API enabling users to easily design, train and use NNs. The modularity principle means that models consist of combinable stand-alone modules. Stand-alone modules are neural layers, cost functions, activation functions, optimizers, initialization schemes and regularizations schemes. Such modules can be combined to create new models. Moreover, it is possible to develop new custom modules. Keras provides two ways to define models: the functional API and sequential model. The former allows to define more complex models such as multi-output models, directed acyclic graphs, or models with shared layers whereas the sequential models are simply defined by a list of layer instances. Trained models can be exported to the HDF5 file format which is specific to Keras. Keras models can be deployed to several platforms including Android using the TensorFlow Android runtime, iOS using Apple's CoreML, Raspberry Pi, browsers using Keras.js or WebDNN, or Google Cloud. [Ker]

2.5.4 ONNX

Open Neural Network Exchange (ONNX) is an open format for ML models - especially NNs. ONNX specifies operators, a computation graph model, and standard data types. Moreover, the ONNX format contains metadata describing semantic information about the type denotations of the outputs and inputs. ONNX currently provides the type denotations `TENSOR`, `IMAGE`, `TEXT`, and `AUDIO`. The dimension denotations for `IMAGE` are `DATA_FEATURE`, `DATA_BATCH`, and `DATA_CHANNEL`. Further, metadata like the required image properties like `Image.BitmapPixelFormat = Bgr8` can be included. ONNX enables developers to share ML models between different frameworks and platforms. Frameworks such as the Microsoft Cognitive Toolkit (CNTK) or PyTorch are able to export ONNX models directly. Models from TensorFlow, Keras, and some other frameworks can be converted to ONNX using framework-specific converter tools such as Keras2onnx or tf2onnx. Furthermore, the winmltoolkit combines multiple converters in one toolkit. The ONNX project also provides a great selecting of pre-trained models in the ONNX format. Examples are MobileNet, VGG, ResNet for image classification or SSD, YOLO v2 for object detection. ONNX models can be directly deployed in supported frameworks such as Windows Machine Learning (WinML) or the

ONNX runtime developed by Microsoft which is available for many platforms including x86_64 computer and Android. The runtime provides APIs for Python, C#, C++, C, Java and Ruby. [Mic20; ONN; ONN20]

2.5.5 Windows Machine Learning

WinML is a runtime for NNs on Windows 10 developed by Microsoft. The runtime is part of the standard Windows 10 SDK and included in any Windows 10 installation. Furthermore, it is available as NuGet package which is usually a newer version WinML compared to the versions included in Windows. WinML is only intended to run models in the ONNX format. Training of NNs is not possible. Hence, models have to be trained by a supported ML framework and converted to ONNX afterwards.

Figure 2.5 outlines the architecture of WinML. The core element is the ONNX Model Inference Engine. The hardware acceleration is realized using DirectML which is a low-level ML inference API provided by DirectX. GPUs and Artificial Intelligence (AI) accelerators are supported. WinML can always use CPUs for inference, even if DirectML is not available. Moreover, WinML provides APIs for several programming languages including Python, C#, and C++. Besides the WinML runtime itself Microsoft provides the development tools winmltoolkit, mlgen and WinML Dashboard. [Ros]

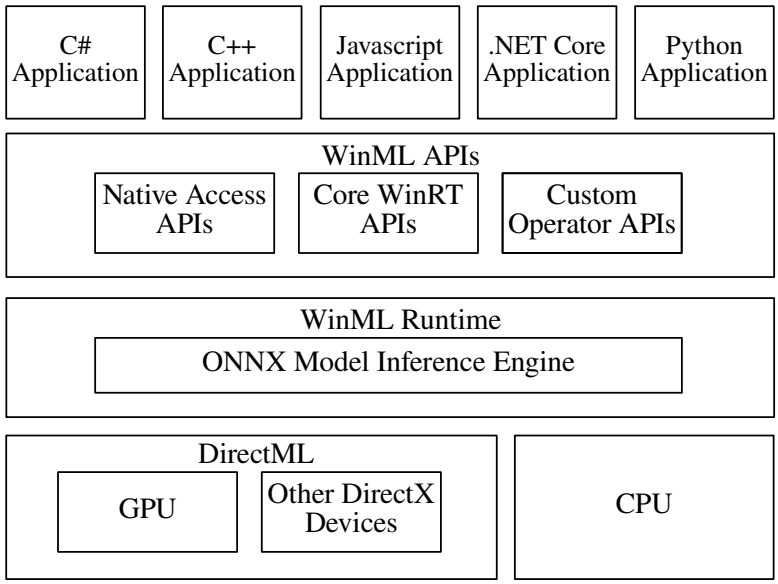

Figure 2.5: Architecture of WinML [Ros]

WinMLTools is a python library able to convert models trained by Keras, scikit-learn, Apple Core ML, lightgbm, xgboost, libSVM or TensorFlow into the ONNX format. The toolkit also supports float16 as well as 8 bit integer quantization. [CRC20] Quantization is supported since WinML Build 18362 [VRMC21].

Mlgen is an extension for the Visual Studio Integrated development environment (IDE) which is able to automatically create C# or C++ wrapper classes for ONNX models according the model's metadata like the types of output and input tensors and mapping of the channels. Mlgen uses the metadata to select the corresponding C# data types in the wrapper class accordingly. For example the data type of image inputs is `ImageFeatureValue` and of basic tensor inputs is TensorFloat. If the `ImageFeatureValue` data type is used, the input images are automatically converted into the proper format and tensorized. One important thing to point out is that after conversion with WinMLTools none of the metadata is already set wherefore they have to be set manually. Changing metadata can be accomplished with the WinML Dashboard. The Windows application can load any ONNX model and visualizes the layer structure of the model. It allows to set the input and output type denotations and the types of the input and output dimensions. [Mic21]

WinML can also be used inside of UWP apps which can run on every Windows 10 devices including the XBox and the HoloLens. Since the Unity game engine can also export UWP apps and supports programming in C#, WinML can also be integrated in applications developed with Unity. [Ros]

2.5.6 Unity

Unity is an engine for 3D and 2D game development by Unity Technologies. Games can be exported to many platforms including desktop computers running Windows, Linux, or Mac OS, Android and iOS smart devices, gaming consoles including Nintendo Switch, XBox and Playstation [Unia]. Moreover, Unity also has support for VR and MR. The HoloLens is also officially supported through UWP apps. The game logic is programmed in C#. Therefore, many C# libraries can be used inside of Unity games. Countless successful games such as Heartstone, Monument Valley 2, or Cities: Skylines were developed using Unity. However, the engine can also be used for professional applications like visualization of technical models. [Unib]

2.5.7 Barracuda

Barracuda is a NN inference library for the Unity game engine. It is also developed by Unity Technologies. Similar to WinML it is only meant for executing NNs in the ONNX format—training is not possible. For this reason, models have to be trained with another ML framework [Uni20b]. Currently, it officially supports models trained by PyTorch, TensorFlow, and Keras. Keras and TensorFlow models have to be converted into ONNX format whereas PyTorch directly supports exporting to the format [Uni20a]. Barracuda currently does not support every model architecture and all of the ONNX operations. For example single-shot detector models do not work. Fully dense or convolutional NNs are generally working. Moreover, MobileNet v1/v2 and Tiny YOLO v2 are officially supported [Uni20c]. The library can run on every platform which is supported by the Unity engine. However, it cannot be used outside of Unity projects. GPU inference is also possible unless OpenGL ES, OpenGL Core or WebGL are used [Uni20d]. There are multiple implementations of CPU and GPU workers which different stability and efficiency characteristics. Next to plain tensors Barracuda also supports textures as input for the NNs. The texture tensorization currently does not allow further parameters for scaling etc. and always normalizes the pixel values for each colour channel in a range of 0-1. Barracuda uses the channel-last format internally whereas the ONNX models should have the channel-first layout. The models are automatically converted to channel-last by Barracuda. [Uni20e]

2.5.8 A-Frame

A-Frame is an open-source VR and AR web framework. It was originally developed by Mozilla but is now maintained by Supermedium. The framework is based on the three.js 3D library. 3D scenes can be entirely described in declarative HTML whereas the logic has to be implemented in JavaScript. Owing to A-Frame is entirely based on platform-independent web technologies it supports most AR, MR and VR headsets including Microsoft HoloLens, Oculus Rift, and Samsung GearVR as well as smartphone based solutions such as Google Cardboard. A-Frame does not only allow creating basic 360° content but also interactive applications which make use of controllers and positional tracking. The framework interfaces with the devices using the WebXR API. For this reason A-Frame surpasses the browser's layout engine and does 3D object updates directly in the memory making it very performant. Complex and large A-Frame applications are able to run at a framerate of 90fps. The finished applications are basically just websites which can be opened with the browser of the devices. When the application is implemented as Progressive Web App (PWA) it can also be installed locally and is available offline afterwards. Multiple companies including Google, Samsung, Disney, Toyota, Ford and the NASA have been using A-Frame for games, car configurators, virtual tours etc. [A-F20]

2.5.9 ARToolKit

ARToolKit is a library for developing AR applications. AR is about overlaying virtual objects onto objects in the real world. For instance a virtual model can be placed on a physical paper card containing a square marker pattern similar to a QR code. ARToolKit uses Computer Vision (CV) algorithms to calculate the orientation of the markers and the camera position in real time. [Kat] It supports iOS, Windows, Android, Mac OS X, and Linux platforms. Moreover, the ARToolKit can also be integrated in Unity applications. The development started at the Human Interface Technology Laboratory at the University of Washington by Dr Hirokazu Kato. Meanwhile it is developed as an open source project under the name ARToolKitX. [QDK18] Furthermore, Qian developed a version of ARToolKitX specifically for Unity on the HoloLens called HoloLensARToolKit [Qia20].

2.5.10 LeNet-5

LeNet-5 is a CNN architecture. Its original purpose is recognizing handwritten characters. The MNIST dataset, which contains handwritten digits, was used to train and benchmark the CNN. The architecture was published 1998 in the paper "Gradient-based learning applied to document recognition" by Lecun et al. Figure 2.6 shows the architecture of LeNet-5. It has in total seven layers (input layer not counted). The input layer has a size of 32x32. The first layer after the input is a convolutional layer with six 5x5 filters resulting in six 28x28 feature maps. The second layer is a subsampling or pooling layer with a filter size of 2x2 resulting in 14x14 feature maps. It uses the average-pooling method. That is followed by another convolutional layer with 16 5x5 filters which reduces the size of the feature maps to 10x10. Here only 10 of 16 are connected to the 6 feature maps of the previous layer in order to break symmetry. The fourth layer is another average-pooling layer with 2x2 filters resulting in 5x5 filter maps. The last subsampling layer is followed by two fully-connected layers. The first has 120 neurons which are connected to all of the $5 \cdot 5 \cdot 16 = 400$ nodes of the precious subsampling layer. The second fully connected layer has 84 neurons. Lastly

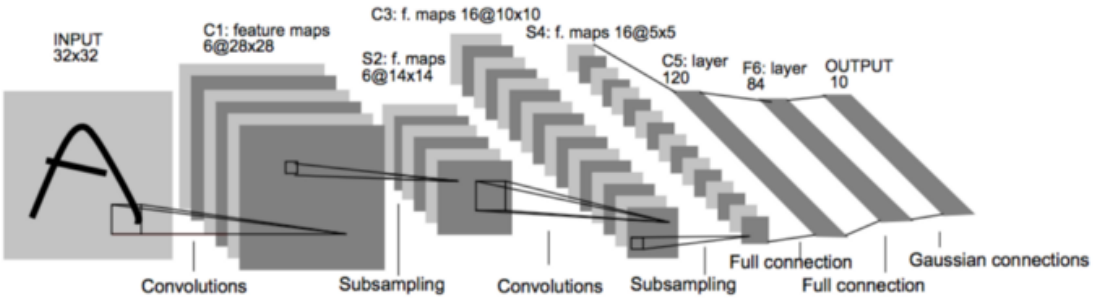

Figure 2.6: LeNet-5 Architecture [LBBH98]

there is the fully connected output layers with ten neurons corresponding to the digits zero to nine. The output layer has a softmax activation function whereas the other layers use tanh activation functions. LeNet-5 reaches an accuracy of 95% on the MNIST test dataset. [LBBH98]

2.5.11 MobileNet

MobileNet is a Convolutional Neural Network architecture developed by the Google employees Howard et al. and was published in the paper "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications" in 2017. As the name suggest it is optimized for mobile devices. For making the model smaller and more efficient MobileNet makes use of depthwise separable convolutions. Such convolutions factorize ordinary convolutions into two parts: a depthwise convolution and a pointwise convolution which is a 1x1 convolution. Filters are separately applied to each channel by the depthwise convolution. Afterwards the outputs of the depthwise convolution are combined using the pointwise convolution. This way filtering and combination is separated in two layers, whereas ordinary convolutions do this in one layer. The convolution is factorized which reduces model size and number of computations. MobileNet consists in total of 28 layers. The first layer is an ordinary convolutional layer. After that follow 13 pairs of depthwise and pointwise convolutional layers. At the end are one average polling, one fully-connected, and one softmax layer. Batch normalization and ReLU follow after all layers, except the fully-connected layer. The computational costs as well as the size of the NN can be further influenced using the parameters α and ρ. α is called width multiplier and can be used to make the NN thinner. ρ is called resolution multiplier with corresponds to the image input resolution. MobileNet is intended for computer vision tasks such as image recognition or object detection. On the ImageNet data a MobileNet with $\alpha = 1.0$ and $\rho = 224$ reaches an accuracy of 70.6%. [HZC+17]

In 2018 Sandler et al. published a new version in the paper "MobileNetV2: Inverted Residuals and Linear Bottlenecks". The MobileNetV2 architecture also makes use of depthwise convolutional layers. However, it introduces new building blocks called bottleneck residual blocks. These blocks consist of one 1x1 pointwise convolutional layer followed by a 3x3 depthwise-covolution and a 1x1 pointwise convolutional layer at the end. After the first two layers ReLU6 is applied whereas the last layer has no non-linearity. Removing the non-linearity further improves the performance according to the authors. The architecture has at the beginning a normal convolutional layer. After that follow 19 residual bottleneck blocks. Some bottlenecks are grouped together. The first block per group has a stride of 1 whereas the other have a stride of 2. After the bottleneck blocks follow

one 1x1 pointwise convolutional layer, one 7x7 average-pooling layer, and another 1x1 pointwise convolutional layer. MobileNetV2 reaches an accuracy of 72.0% on ImageNet which is higher than the accuracy of MobileNetV1. In the same time it is smaller with 3.4M parameters compared to 4.2M of the first generation ($\alpha = 1.0$ and $\rho = 224$ for both models). [SHZ+18]

2.5.12 Fashion-MNIST

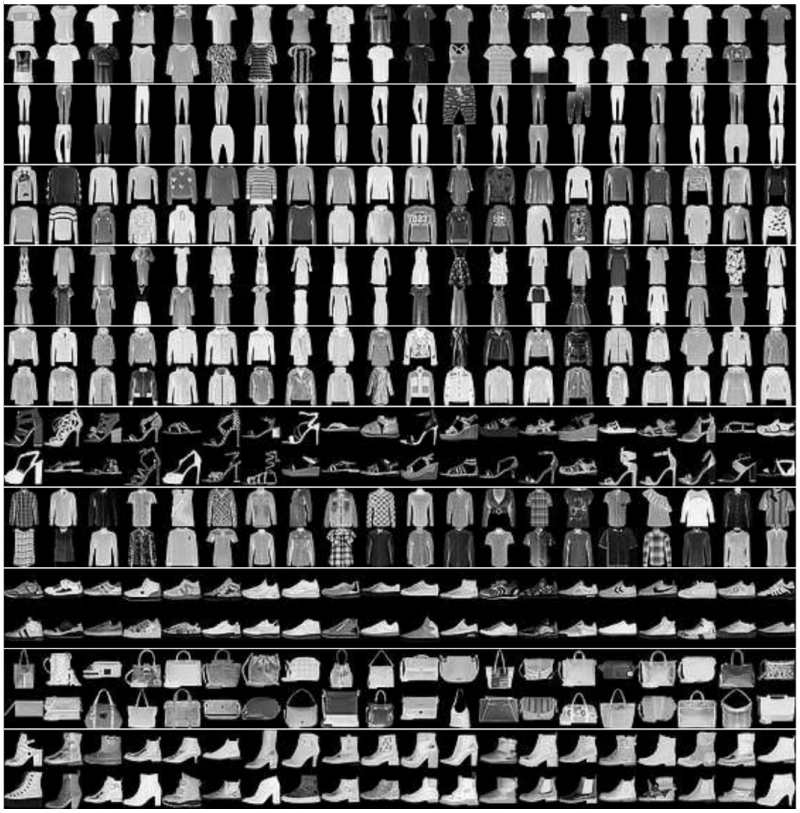

Figure 2.7: Some examples of the pictures in the Fashion-MNIST dataset [XRV17]

Fashion-MNIST is a dataset for benchmarking ML algorithms developed by Zalando Research. It aims to replace the classic MNIST dataset from 1998. MNIST is very popular since it is very small and many ML frameworks include it out of the box. However, it is nowadays not challenging enough for modern ML algorithms. Because of that Fashion-MNIST is a drop-in replacement for MNIST since it uses the exact same format and also has ten different classes. While being compatible to MNIST the data is considerably more complex since it contains pictures of clothing items instead of just digits. The pictures are extracted from Zalandos's catalogue and converted into a 28x28 pixel 8-bit greyscale images. There are the ten classes T-Shirt/top, trouser, pullover, dress, coat, sandals, shirt, sneaker, bag, and ankle boots. Figure 2.7 shows some examples of the pictures in the Fashion-MNIST dataset. The horizontal white lines separate the classes which are in the same order as mentioned above. The dataset contains 60,000 image-label pairs for training as well as 10,000 for testing. The authors of the dataset evaluated Fashion-MNIST on multiple common ML algorithms and showed that the accuracy is always indeed lower compared to MNIST. [XRV17]

3 Related Work

This chapter gives an overview of already existing work related to the topic of this thesis. There are already several approaches of using NNs in conjunction with the Microsoft HoloLens.

Kowalski et al. developed a framework for alignment of faces, estimation of head poses and retrieval of facial attributes. It is called HoloFace and is developed for the HoloLens (1st gen) using Unity and C++ plugins. Possible use cases include facial or emotion recognition. The framework offers two different implementations of face alignment methods. One implementation uses regression trees and can run directly on the HoloLens. The other implementation uses a CNN and runs on an external computer which is connected to the HoloLens via Wi-Fi. According to the authors CNNs are too expensive to run directly on the HoloLens (1st gen). Both methods are capable of processing 30fps which the maximum frame rate of the camera. On a computer equipped with a NVIDIA GeForce 1070GTX GPU the regression tree method can run at over 1000fps whereas the CNN method can run at 160fps. The authors managed to reduce the bandwidth of the network commutation to 3.1 Mbit/s by sending a small image only containing the face. As a result both methods are able to process the maximum framerate of 30fps on the HoloLens. However, the CNN method is considerably more precise but depends on an external computer. The article demonstrates that running NN on an external server connected to a HoloLens application is a feasible approach. [KNGG18]

Naritomi et al. developed a HoloLens (1st gen) application which can overlay real food with virtual fake food of a different choosable category. Image transformation and segmentation are handled by CNNs. The CNNs run on an external server which is connected to the HoloLens via Wi-Fi. The HoloLens itself only does the rendering and spacial mapping where no NNs are involved. The work shows that it is possible to combine NN powered image generation on a external device with the HoloLens (1st gen). [NTEY18]

Both papers make use of an external server for executing NN. The HoloLens is connected to the server via Wi-Fi and does not execute NNs itself. There are several other papers which use the same approach. For instance [KCG20] which uses NNs for scene classification. Another more recent example is [ZHP20] which implement an escape room system based on AR. The used server coordinated the game and uses a CNN for image recognition. It is important to point out, that all the aforementioned papers use the first generation of the HoloLens.

S. Bovo, a Microsoft employee, published the article [Bov19a] about using NNs on the HoloLens using WinML. The model is trained using the Azure Custom Vision cloud service. It is capable of distinguishing two different faces. The model is exported in the ONNX format and has a size of roughly 3Mb. On the HoloLens a small Unity UWP application captures frames from the webcam, passes them to the NN trough WinML and displays the detected class in the users field of view. The evaluation time of the model ranges between 250 and 400ms which results in at least 2 frames by second. This evaluation was done on the first generation on the HoloLens. However, the author also published the source code for the HoloLens 2 which works the same way [Bov19b]. The code

is slightly different since the model is trained for recognizing up/down thumbs. Furthermore, the Unity configuration files are different as another architecture is targeted. There are no published time measurements of the HoloLens 2 application. A downside when using WinML in Unity is, that WinML related code has to be wrapped in `#if UNITY_WSA && !UNITY_EDITOR` compiler directives which could be inconvenient for the programmer since Visual Studio threats the code like it is commented out. Moreover, the application cannot be tested directly inside of Unity. It has to be exported as UWP and tested directly on the HoloLens or the HoloLens emulator. The article demonstrates that WinML is a feasible way to use NN on the HoloLens in UWP/Unity applications. [Bov19a]

We already showed in a previous research project that NNs trained by TensorFlow/Keras can be converted into ONNX and executed in WinML on the HoloLens (1st gen). Furthermore, the work describes that a NN trained by Keras can be integrated in the Unity-based muscle visualization project of the PerSiVal project using WinML. That work shows that WinML is a feasible way to integrate NNs in Unity applications as well as plain UWP applications on the HoloLens (1st gen). [HKL20]

To conclude there are several works which use NNs on an external server and connect via WiFi to an application on the HoloLens which can be feasible in situations where having a server and a reliable connection is possible. This is not ideal for every situation. There can be situations where carrying an external computer is not possible and untethered operation is required. For example when the HoloLens is intended to be used outside on a construction site or for military applications. Furthermore, most of them only deal with the first generation of the HoloLens. The HoloLens 2 could make new approaches possible due to the different architecture, more powerful hardware, and newer software including the new Chrome based Edge browser. This Bachelor thesis differs from these works since it specifically targets the HoloLens 2 and aims to find and compare multiple ways to execute NNs directly on the HoloLens without requiring an external device and a network connection. Furthermore, the main goal is to make the different approaches comparable by specifying multiple criteria in order to be able to make an informed decision which of the approaches is the best for a specific application.

Moreover, [Bov19a] shows that it is possible to use WinMLs in order to execute NN trained by the Azure Vision Cloud service directly on both generation of the HoloLens. However, this approach has a few limitations: In practice more common machine learning frameworks such as TensorFlow or PyTorch are usually used to train NNs. The article does not show if it is possible to use NNs trained by frameworks other than Azure Custom Vision. Furthermore, WinML can only be used in UWP and WIN32 applications. Lastly, integrating WinML in Unity applications can be a hassle for developers. This thesis differs from the work by Bovo since it aims to overcome these limitations. Beeing able to use models trained by TensorFlow is a minimal requirement in this work. Another goal is to find different ways which are easier to use and platform independent. Our previous work [HKL20] already shows that it is possible to us NNs with WinML on the HoloLens (1st gen). However, it also only targets the first generation of the HoloLens and has the same downsides as the work by Bovo except for the possibility to use models trained by TensorFlow/Keras.

4 Problem Statement

AR/MR headsets offer great opportunities for enterprise, educational as well as private applications. Enterprise applications could be training of new assembly workers, visualizing new design ideas, remote collaboration, or displaying virtual plans of buildings on the intended building site. In education for example 3D objects and simulations can be used to support teaching of complex biological systems. Private users could use MR headsets not only for games but also for displaying information such as navigation directions.

Another emerging technology in computer science is ML—especially NNs. Many task such as image classification, object detection, speech recognition or text translation can be solved easier than before, under the premise that a sufficient labelled training dataset is available. Moreover, also simulations, which are in the focus of the Pervasive Simulation and Visualization (PerSiVal) procect of the University of Stuttgart, benefit from NNs.

Combing the topics NNs and MR/AR opens up totally new possibilities. An example would be classifying objects the user is looking at. This could be used as a more interactive replacement for audio guides. Imagine walking through a botanical garden and receiving names as well as additional information of exotic plants just by looking at them.

Part of this work is to propose and implement multiple approaches to execute NNs on the Microsoft HoloLens 2. A requirement for these approaches is that the NN runs directly on the device and do neither require a Wi-Fi connection nor an external device. The reason is that a Wi-Fi connection is likely to be unstable and to introduce latency. Especially when moving around the connection could be interrupted from time to time which would harm the user experience. If the service which executes the NN is only reachable over the internet—for example a cloud service—using the HoloLens 2 outside would be even more complicated since it has no cellular connection. An external mobile router with cellular connectivity would be required. Using cloud services could also be problematic in terms of data privacy. Furthermore, carrying a computer for model execution close to the HoloLens 2 is also not feasible since it complicates the setup and introduces further costs. Executing NNs directly on the HoloLens 2 would overcome all of these issues.

Another requirement is to enable the usage of NNs trained by TensorFlow/Keras. The reason is that TensorFlow is the predominant ML framework according a survey among 1388 enterprises conducted in 2020 by O'Reilly. Over 50% of the interviewed enterprises use TensorFlow [MS20].

So in summary, all approaches for executing NNs on the HoloLens 2 have to fulfil the following requirements:

- Run offline and independently of external devices.
- Support integration into MR/AR applications which means it can be integrated in ARMR 3D frameworks.
- Execute models trained by TensorFlow using the Keras layers API.

The goal of this Bachelor thesis is to make all approaches qualitatively and quantitatively comparable. Therefore, a set of diagnostic criteria such as performance or ease of implementation has to be defined. In order to evaluate the different approaches a benchmark as well as a Proof of Concept (PoC) application shall be developed for each one. Finally, all feasible approaches are compared in an extensive overview describing the specific pros and cons of each approach. This overview enables developers to select the best approach for a specific application.

5 Experiment Design

This chapter describes the design of the experiments and evaluation criteria. In total four different possible ways to bring NN on the HoloLens are described. For each way it is planned to develop a basic bechmarking application which can load a NN, runs a test dataset on it and measures the error rate and the inference time. In order to compare the different ways the same model will be used for each one and trained by TensorFlow using Python on a computer. For each working approach a Proof of Concept (PoC) application shall be developed. Eventually, at least one way will be used to integrate a NN in the existing muscle visualization application.

5.1 Evaluation Criteria

The different approaches will be evaluated on the following criteria:

1. **Ease of implementation:** Evaluation of the approach in terms of easiness for the programmer and investigation if there are any inconveniences or hurdles. An example for an inconvenience would be that code cannot be debugged with an IDE. A further factor is the availability of third-party resources and example projects.
2. **Performance:** Measurement of the models inference time and comparison between the different approaches. It will be measured using the benchmarcking applications for each approach with the LeNet-5 model and the Fashion-MNIST test dataset. The time points before and after executing the prediction command are recorded and the difference is calculated afterwards. The entire test dataset is fed into the NN at once in one batch.
3. **Accuracy/Error rate:** Measurement of the accuracy and error rates and comparison between the different approaches and the original model. For the error rates the Top-1-Error and the Top-5-Errors will be measured, also using the benchmarking applications with the LeNet-5 model and the Fashion-MNIST test dataset. The former represents the percentage of the example dataset's data pairs where the by the NN predicted class did not correspond with the correct class. The Top-5-Error represents the percentage of data pairs where the correct class is not among the five classes with the highest probabilities predicted by the NN.
4. **Compatibility with common machine learning frameworks and pre-trained models:** List of which models of common ML frameworks can be used. As mentioned in the problem statement in chapter 4, TensorFlow is the most used ML framework. It will be checked if models trained by TensorFlow can be used with this approach and if they have to be changed or retrained in order to be compatible. This is especially relevant for using pre-trained NNs since there are many pre-trained NNs available in the TensorFlow format. Alternatively, the ONNX model zoo also provides pre-trained models which are also be verified if they are compatible.

5. **Integrability with frameworks or game engines:** List which game engines and frameworks can be used with this approach. HoloLens applications are usually developed with the Unity game engine. Another way is using JavaScript frameworks such as A-Frame or three.js. It will be investigated which of these frameworks can be used with the approach.
6. **Platform independency:** List of AR platforms apart from the HoloLens 2 which are compatible with the approach. It will be investigated if the applications can only run the HoloLens or also on other devices such as smartphones without severe changes.

5.2 Neural Network for Benchmarking Applications

The NN will be trained using TensorFlow and Keras with a small Python script and converted to the TensorFlow.js and the ONNX formats. The architecture will be LeNet-5 which is already described in 2.5.10 in detail. Unlike the original model an input format of 28x28 pixels is used here. Fashion-MNIST by Zalando Research (described in Section 2.5.12) will be used as training dataset. The dataset with 10,000 examples which will be used for the evaluation of the NNs implementations of the different approaches.

5.3 Proof of Concept Applications

The Proof of Concept applications proof that the approaches cannot only be used in simple 2D applications, but also in AR/MR application. The intended applications classifies the object the user is looking at and display the detected class directly into the user's field of view. The classification is realized using a pre-trained MobileNetV2 NN ($\alpha = 1.0$ and $\rho = 224$) which was trained on the ImageNet dataset. The apps also measure the average inference time. Every 100 frames the average time of the last 100 frames is calculated and displayed.

5.4 Approach 1: TensorFlow.js

The first approach uses only web technologies - especially JavaScript. It is planned to use TensorFlow.js (described in section 2.5.2) to integrate NNs. For the AR part there are multiple JavaScript 3D libraries such as Babylon.js, Three.js and A-Frame. The last one mainly focuses on VR and AR whereas the other ones are more general 3d libraries but support AR as well. Web applications can be accessed with the Edge browser or Mozillas' Firefox Reality on the HoloLens 2.

5.5 Approach 2: Unity Barracuda

Approach 2 uses the Barracuda library (described in 2.5.7) for the Unity game engine. Since the library can only be used in applications developed using Unity the benchmarking application has also to be implemented as Unity application. Similar to approach 2 the NN will be converted from the TensorFlow format into the ONNX format.

5.6 Approach 3: Windows Machine Learning

The third approach makes use of WinML (described in 2.5.5). WinML will be integrated in a UWP app. In UWP apps the HolographicSpace API or OpenXR API can be used to develop AR apps. Furthermore, it should be possible to integrate it in applications developed with the Unity game engine since Unity also exports UWP apps for the HoloLens. Keras2onnx will be used to convert the NN trained by TensorFlow into the ONNX format.

5.7 Approach 4: TensorFlow.NET

In the last approach it is planned to use TensorFlow directly on the HoloLens 2. Since the HoloLens 2 has a 64 bit architecture and also supports Win32 applications it should be easier than on the HoloLens (1st gen). TenserFlows official C binding does not offer all the functions of the Python API. Therefore, it is planned to use the C# library TensorFlow.NET which is a wrapper around TensorFlow and offers an API which is similar to the Python API. TensorFlow.NET would be a convenient option since it provides a convenient C# API which is oriented on the Python API.

6 Experiments and Evaluation

The following chapter describes setup and execution of the experiments and evaluates their results according to the evaluation criteria. In the end there is a discussion and comparison of the different approaches.

6.1 Setup

The following two sections describe the hard and software which will be used for the experiments. In the software section the specific software versions of programs used on the development computer and directly on the HoloLens 2 are listed.

6.1.1 Hardware

For software development a Lenovo Thinkpad T480s notebook is used which has a Intel Core i5-8250U quad-core processor and 16 GB DDR4 RAM. The test device is the Microsoft HoloLens 2 MR headset. It is already described in detail in section 2.4.2.

6.1.2 Software

The operating system of the notebook is Manjaro Linux. The training scripts will be written using the PyCharm Professional 2020.3 IDE in Python 3. The Python interpreter runs in a Miniconda environment and has the version number 3.7.9. For training TensorFlow 2.4.1 is used. Keras2onnx which is used for conversion into the ONNX format has version number 1.7.0.

The development of UWP and Win32 applications can only be accomplished under Windows. For that reason Windows 10 is used in a virtual machine using VMware Workstation 16.1.0. The exact version is Windows 10 Education version 21H1 with build number 19043.906. Visual Studio Community 2019 in version 16.8.6 will be used as IDE. Unity has version 2019.4.23f1.

On the HoloLens 2 the insider preview of Windows Holographic with build number 89.0.76.5.0 is installed. Furthermore, the browsers Firefox Reality in version 12.1 rc1 and the Chrome based Microsoft Edge with build number 89.0.76.5.0 are installed. The WinML applications use the WinML version included in Windows 10 SDK 18262. TensorFlow.js is used in version 3.0.0, A-Frame in version 1.2.0.

6.2 Approach 1: TensorFlow.js

Approach 1 makes use of TensorFlow.js which can be used in web applications and runs directly in the browser. First of all, the NN, which is trained by TensorFlow/Keras using a Python script, has to be converted into the proper format for TensorFlow.js. The tensorflowjs python package can accomplish this. Only the code shown in listing 6.1 has to be inserted in the training script to export the model in the JSON based TensorFlow.js format. Alternatively, an already saved model can be converted using the `tensorflowjs_converter` command line tool.

```
import tensorflowjs as tfjs
tfjs.converters.save_keras_model(keras_model, 'Filename.json')
```

Listing 6.1: Python code to convert Keras model into the TensorFlow.js format

The base of web applications are HTML files which contain the layout and content of a web page. Client side logic is programmed using JavaScript. Optionally, CSS can be used to modify the look of a web page. Just as any other JavaScript library TensorFlow.js can be imported using a `<script>` tag in the HTML file or installed using the Node package manager and a JavaScript build tool. Here the tag method is used. Since the benchmarking application uses the Fashion-MNIST dataset there has to be a method to import the datatset into the application. Google already provides a wrapper class for the MNIST dataset in the TensorFlow.js examples repository [Tenc]. This class can also be used for Fashion-MNIST since it is a drop-in replacement for MNIST. Only the URL of image and label files have to be changed accordingly.

First the benchmarking application loads the Fashion-MNIST dataset and the LeNet-5 model. Then a button press starts the evaluation. The entire test dataset is fed in one batch into the NN. The script saves the time points before and after the prediction command using `performance.now()` and calculates the elapsed time afterwards. This is done 100 times and in the end the average values are calculated in order to get consistent results, since it could be possible that the inference time slows down over time when the device gets warmer. After that the array with the predicted classes is compared to the label array from the test dataset to compare the accuracy and the error rates. The application has to be hosted on a local web server in order to be accessible by the HoloLens. This requires a WiFi connection. However, the execution of the NN is done directly on the HoloLens 2. It is possible to make web applications usable offline by creating a PWA. This is done with the Proof of Concept application in the following section. The application is tested on the Firefox Reality Browser and the Chrome based Edge browser. In Firefox Reality the application did not work at all since it froze just after the script starts to load the NN. So no further debugging was possible. Official TensorFlow.js examples also did not work. On the other hand there were no problems with the Chrome based Edge browser. TensorFlow.js makes use of GPU inference through the WebGL API.

6.2.1 Proof of Concept Application

In order to proof that the TensorFlow.js approach cannot only be used in simple 2D web applications but also in AR/MR the aforementioned Proof of Concept application is developed. The application uses the A-Frame AR/VR JavaScript framework for realizing the AR part of the application. For executing the NN TensorFlow.js is used again. The main HTML file specifies the layout of the A-Frame scene. The scene basically only contains a text field and a point acting as a crosshair. Both

elements are fixed to the camera view. This is only a very simple AR scene. However, A-Frame is also capable of rendering 3D objects which makes it suitable for games and other more complex MR/AR applications. After opening the web application it asks the user to permit webcam access and waits a few moments until the NN is fully loaded. After that the user has to click on the AR button in the bottom right corner in order to enable the immersive mode. In the immersive mode the browser is completely hidden and only the A-Frame scene is visible. In this case only the text and the point are visible to the user. The main JavaScript file initializes the webcam access and creates a video element which contains the webcam stream. The video element could also be created using HTML. However, the element should be invisible and hiding the element using CSS did not work because it interfered with A-Frame. It is not possible to enable the immersive mode while the video element is created in the HTML file. TensorFlow.js directly provides the `tf.data.webcam` API for accessing the webcam which would be easier that the aforementioned approach. However, this API currently seems to be incompatible with the Chrome based Edge browser on the HoloLens 2 since the API is not able to activate and access the webcam. Even official TensorFlow.js demonstration applications which use this API do not work. They always tell that there is no webcam available. The main script of the PoC application grabs the current frame of the webcam element and feeds it into the NN within the main loop. Similar to the benchmarking application the inference time is recorded. After that the predicted ImageNet class, the probability as well as the inference time are written into the text element fixed to the camera.

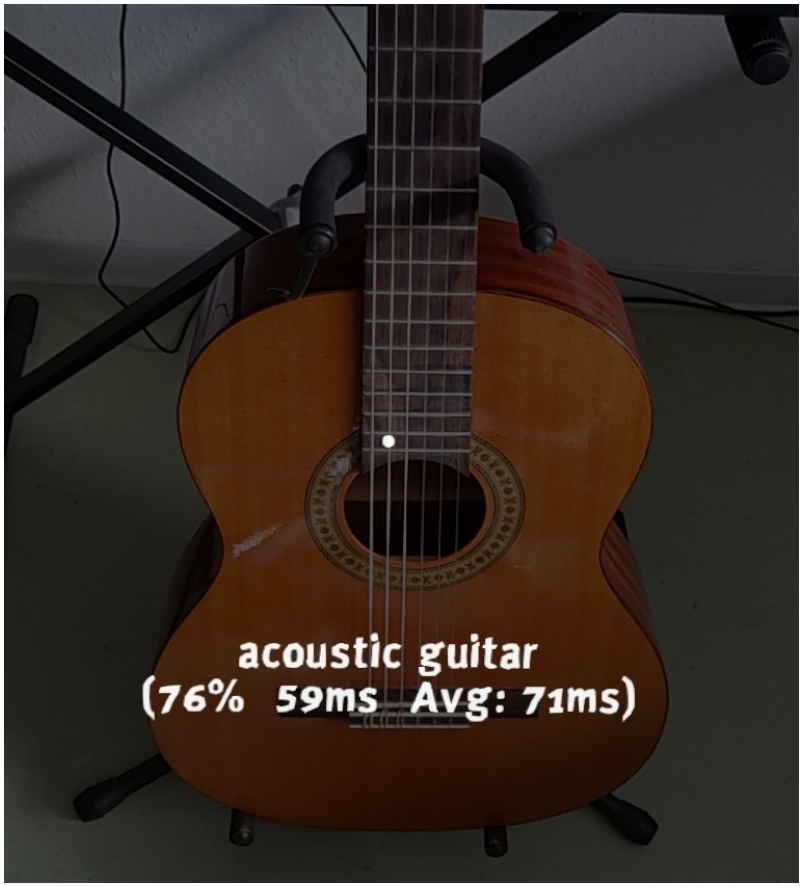

Figure 6.1: TensorFlow.js PoC application running on the HoloLens 2

The application has to be hosted on a web server, too. However, it is implemented as a PWA and can therefore work offline as well. After visiting the website the browser offers to install the PWA locally. When the application is installed it shows app in the app menu of Windows just as any other native app. Furthermore, it works completely offline since all necessary files are cached. Installable PWAs require a webmanifest file which contains information such as name, description, theme colour, start URL, and a reference to the compulsory icon. Furthermore, it is required that the website is served via HTTPS [Moz21a]. In order to be able to cache the necessary files a service worker is required. The service worker caches the file during installation of the app and returns the cached files if they are requested by the application. Thus, the files which have to be cached have to be specified in the service worker. The service worker code is adapted from the Mozilla PWA documentation [Moz21b].

Figure 6.1 shows a screenshot of the finished application running in the Chrome based Edge browser on the HoloLens 2. It only looks dark on the screenshot. In person the dark grey layer is not perceptible. This is probably a flaw in A-Frame since official A-Frame example applications such as [A-F] have the same issue. The application runs smoothly and the inference time for one picture is 71ms in average.

6.2.2 Evaluation

This section evaluates the TensorFlow.js approach using the criteria described in 5.1. The evaluation application is also executed on the ThinkPad T480s notebook in a Chromium browser for comparison. Table 6.1 shows the results.

Table 6.1: Measurement results of approach 1

	Inference time	Accuracy	Top-1 error	Top-5 error
Python script on notebook	3803ms	89.03%	10.97%	0.17%
TF.js on notebook	257ms	90.70%	9.29%	0.10%
TF.js on HoloLens 2	1035ms	90.70%	9.29%	0.10%

1. **Ease of implementation:** Using TensorFlow.js does not have any major drawbacks which could make development difficult. Since JavaScript is a very common programming language there are several IDEs such as WebStorm which also support autocompletion for TensorFlow.js. Furthermore, there are many example applications, tutorials and other resources since TensorFlow.js is already applied very broadly. Another thing to point out is, that the TensorFlow.js API is very extensive and covers nearly all functions of the TensorFlow Python API. This means that common tasks such as normalization or cropping of images are directly supported as Tensor operations and do not have to be implemented by hand. Furthermore, Tensor operations are usually more performant that routines written in vanilla JavaScript since they can be executed using WebGL.
2. **Performance:** As the table shows the inference time for the entire test dataset is 1035ms. This is slower than the TensorFlow.js application on the notebook which was to be expected since the notebook has a more powerful CPU and on-board GPU. The inference time of

the Python script is slower since the Python version of TensorFlow requires a dedicated GPU which the notebook does not have. Therefore, the Python script can only use CPU inference. The PoC application on the HoloLens 2 also runs very smoothly with an average inference time of 71ms per frame. This proofs that the performance is sufficient for practical applications.

3. **Accuracy/Error rate:** The accuracy is 90.7%, the Top 1 error 9.29%, and the Top 5 error 0.1%. These are the same values as these of the TensorFlow.js app. The accuracy is even slightly higher than the accuracy measured after training with the Python script.

4. **Compatibility with common machine learning frameworks and pre-trained models:** All models trained with TensorFlow or Keras (also with other backends) can be converted into the TensorFlow.js format. That means that all the pre-trained TensorFlow models and Keras applications can be used. Furthermore, there are also pre-trained TensorFlow.js models such as MobileNet, PoseNet or Blaze Face [Ten21].

5. **Integrability with frameworks or game engines:** TensorFlow.js can be integrated in any JavaScript web application. The PoC application combines TensorFlow.js with the A-Frame AR framework. However, TensorFlow.js also works with Babylon.js and Three.js which are 3D JavaScript libraries also supporting WebXR for developing AR applications since there are already projects which combine TensorFlow.js with these frameworks [Dav20; Rui21]. Moreover, integrating TensorFlow.js plain JavaScript applications which directly use the WebXR APIs are possible, too. On the other hand the Unity game engine, which is frequently used for developing HoloLens apps, cannot make use of TensorFlow.js since the logic code has to be written in C#.

6. **Platform independency:** TensorFlow.js can in theory run on every device with a JavaScript capable web browser. Nonetheless, there is no guarantee since the processing power of the device can be a limiting factor. Furthermore, compatibility issues with TensorFlow.js could happen which is case with Firefox Reality. The browser should support WebGL since it is considerably more performant than vanilla JavaScript. However, TensorFlow.js can fall back on the CPU backend which only requires JavaScript. AR/MR applications developed with A-Frame or the other AR JavaScript libraries should work on each headset which is supported by the library. As mentioned in the background A-Frame supports among others Microsoft HoloLens, Oculus Rift, and Samsung GearVR as well as smartphone based solutions such as Google Cardboard [A-F20].

To sum up the experiment shows that using TensorFlow.js in the Chrome based Edge browser on the HoloLens 2 is a feasible approach. It is easy to implement, sufficiently performant and accurate, platform independent and can be integrated in AR/MR applications developed with JavaScript AR libraries.

6.3 Approach 2: Unity Barracuda

The following section covers the second approach which makes use of Unity Barracuda. As mentioned in section 2.5.7 Barracuda can only be used in Unity applications. In order to use Barracuda, it has to be installed in a project using the Unity package manager. Therefore, the benchmarking application as well as the proof of concept applications have to be developed as Unity application. Similar to the last approach the NN has to be converted into the proper format—in this case ONNX. Here the keras2onnx Python package is used since the NN is trained with TensorFlow using the Keras sequential model API. Listing 6.2 shows how keras2onnx can be used to convert an existing Keras model (in variable `keras_model`) into the ONNX format. The parameter `target_opset` specifies the opset version of the ONNX file. Opset 9 is recommended for Barracuda [Uni20a]. The name of the input layer has to be entered in the `channel_first_inputs` parameter if the model uses the channel-last format which is the default for TensorFlow and Keras. The converter then transposes the input layer into the channel-first format which is the native data layout for ONNX. However, it is important to point out that Barracuda works internally with the channel-last format and automatically converts models back to channel-last. This means that the input tensors have to be in the channel-last layout whereas the ONNX models have to be in the channel-first layout [Uni20a]. Moreover, the tf2onnx package can be used if the TensorFlow saved mode format is used instead of Keras .h5 format.

```
import keras2onnx
onnx_model = keras2onnx.convert_keras(keras_model, target_opset=9,
        channel_first_inputs=["input_layer_name"])
keras2onnx.save_model(onnx_model, "Filename.onnx")
```

Listing 6.2: Python code to convert Keras model into the ONNX format for Barracuda

Unity applications consist of scenes which contain all the 3D elements and describe how the 3D environment is structured. Such scenes are created and edited with the graphical Unity editor whereas the game logic has to be implemented using the Scripting API in C#. For this the Unity editor opens the Visual Studio IDE for editing the code and debugging. This PoC application works very similar to the application in approach 1. Since the benchmarking application for Barracuda also uses the Fashion-MNIST dataset the application also requires a wrapper class. The wrapper class loads the test dataset in the original ubyte format, normalizes the values and generates a float array which can be tensorized afterwards. The class which needs access to the NN model has to have a public property of the type `NNModel` where the model is injected by the Unity engine. The model file has to be imported in the Unity project, placed in a folder called Models inside of the Assets folder and dropped in the according slot of the component containing the script file using the Editor. Firstly, the main function loads the NN and the dataset. Furthermore, the inference engine called Worker is created using `WorkerFactory.CreateWorker(WorkerFactory.Type.ComputePrecompiled, model)`. The worker type `ComputePrecompiled` is used to enable GPU execution. Then the dataset is tensorized and fed into the model. The inference time is measured using the C# `System.Diagnostics.Stopwatch` class which can be used to accurately measure execution time. The result is again compared with the labels of the test dataset to compute the accuracy and the error rates. That is also done 100 times and the average values are calculated in the end. The results are saved in a log file in order to be accessible afterwards.

6.3.1 Proof of Concept Application

To show that this approach is feasible the Proof of Concept application is also developed with Untiy and Barracuda. The webcam is accessed using the `WebCamTexture` class provided by Unity. Barracuda provides a method for converting tensors into textures. However, it does not accept parameters for bias and scaling. Therefore, the image preprocessing which includes cropping, scaling and normalization has to be implemented by hand in C#. Ashikhmin already implemented this for a Unity Barracuda applications which run on Android [Ash20]. His conversion methods are also used in this PoC application.

The application runs not as smoothly as the JavaScript PoC application since the text displayed to the user streaks a bit if the user moves his head. This is probably due to the image preprocessing which is implemented in C# and not as more efficient tensor operations. However, the inference time per frame is 7ms in average which is considerably faster than the TensorFlow.js application which runs completely fluently. More efficient image preprocessing routines and improved concurrent execution of them could further improve the fluency of the entire application since the complex computations block the main thread. Barracuda supports asynchronous executing which was implemented firstly. It was switched to synchronous execution in order to measure the inference time accurately which worsens the aforementioned lag since it blocks the main thread. However, if the time measurement is not required any more it can be switch back to asynchronous execution.

6.3.2 Evaluation

This section evaluates the Barracuda approach using the criteria described in 5.1. Table 6.2 shows the results of the benchmarking application.

Table 6.2: Measurement results of approach 2

	Inference time	Accuracy	Top-1 error	Top-5 error
Python script on notebook	3803ms	89.03%	10.97%	0.17%
Barracuda test app on HoloLens 2	183ms	89.03%	10.97%	0.17%

1. **Ease of implementation:** Since Barracuda is an official Unity component it is tightly integrated in Unity and can be accessed like any other Unity API. Furthermore, the Unity editor can show some information of the ONNX model such as the layer list. There is an official API documentation by Unity [Uni20b]. However, there are only very few third party example projects and resources since Barracuda is not used very often. The Barracuda API is not as extensive as the TensorFlow API. Therefore, tasks like image preprocessing have to be implemented by hand if the provided functions are not sufficient.
2. **Performance:** Barracuda runs very performant since is able to make use of GPU inference. The benchmarking application needs in average 183ms for evaluation the entire Fashion-MNIST test dataset in one batch. In the PoC application Barracuda has an average inference time of 7ms per frame. However, the image preprocessing routines slow down the entire application a bit.

3. **Accuracy/Error rate:** The accuracy is 89.3%, the Top 1 error 10.97%, and the Top 5 error 0.17% measured by the Barracuda test application and the Fashion-MNIST test datatset. As Table 6.2 shows, Barracuda is in this case as accurate as the Python version of TensorFlow.
4. **Compatibility with common machine learning frameworks and pre-trained models:** Barracuda uses the ONNX format. It officially supports models from Pytorch, Tensorflow, and Keras. Pytorch can directly export ONNX whereas the models for the other frameworks have to be converted. Pre-trained models from the ONNX model zoo can be used. Furthermore, pre-trained models for TensorFlow or Keras can be converted into ONNX and used with Barracuda.
5. **Integrability with frameworks or game engines:** Barracuda is a library which is specifically developed for Unity. Therefore, Barracuda can only be used in Unity-based projects.
6. **Platform independency:** Barracuda can be used on all platforms supported by Unity including Windows, Linux, Mac OS, Android, iOS, Nintendo Switch, XBox and Playstation [Unia]. GPU inference is also possible unless OpenGL ES, OpenGL Core or WebGL are used [Uni20d].

This section showed that Barracuda is a feasible way to use NN on the HoloLens 2. Based on the GPU inference it runs very performant. Barracauda can only be used in Unity projects. Applications with Barracuda can run on every platform supported by Unity.

6.4 Approach 3: Windows Machine Learning

This section covers the third approach which makes use of WinML. WinML can be used in UWP apps which include Unity projects as well as Win32 apps. Here two benchmarking apps are developed. One benchmarking app is based on Unity whereas the second app is a plain UWP 2D app. The reason is that it is possible that Unity slows down the NN execution of WinML. The second app can measure the inference time of WinML without the possible overhead introduced by Unity. Similar to Barracuda WinML uses the ONNX file format. Therefore, the same code as Listing 6.2 can be used to convert Keras models into ONNX. Furthermore, the WinMLTools Python package can be used. It is based on ONNXMLTools and offers some WinML specific features such as weight quantization which can reduce the file size [CRC20]. However, the ONNX version depends on the specific version of WinML. The applications in this section uses the WinML version included in Windows 10 Build 18362. With the newer Build 19041 the compilation of the Unity project failed. Therefore, the applications use the previous version. The WinML version in Build 18362 supports ONNX opset 8 [VRMC21]. WinML uses the channel-first format also internally. Therefore, the models as well as the input tensors have to be in the channel-first format. WinML uses the DirectML API for GPU inference. Unfortunately, DirectML is currently not available for ARM(64). However, it is planned to support ARM in the future according to a Microsoft developer on GitHub [Cha20]. Therefore, only CPU inference can be used here.

The mlgen Visual Studio extension automatically generates wrapper classes for the model when an ONNX file is imported into the project. The wrapper class provides methods for loading the NN and evaluating inputs. The benchmarking applications use the same Fashion-MNIST wrapper class

as approach 2. The 2D UWP benchmarking app only consists of a start button and an output text field. The functionality is the same as the previous benchmarking apps meaning it also runs the Fashion-MNIST dataset and measures accuracy, Top-1 error, Top-5 error and inference time.

Furthermore, WinML is integrated in the same benchmarking app as Barracuda. Both frameworks can activated using checkboxes in the Unity editor. All code lines which use the WinML APIs have to be wrapped in `#if WINDOWS_UWP` compiler directives. The reason is that these APIs are in the `Windows.Ai.MachineLearning` namespace which is not accessible inside the Unity editor. Therefore, Unity displays an error message and the project cannot be exported as UWP. The compiler directives make the code invisible to Unity. As soon as the project is exported as UWP application the code is enabled again. This makes the development relatively uncomfortable since the code inside the compiler directives cannot be debugged and IDE support such as autocompletion is not available in this area. Furthermore, the entire app cannot be executed and tested directly inside of Unity. It has to be exported and installed on the HoloLens or the HoloLens emulator every time. In order to easily integrate WinML the model class is wrapped in another class which creates the tensor object and calls the inference method. All lines which use the WinML APIs are wrapped in the aforementioned compiler directives. Thanks to this way the WinML code is abstracted and the other parts of the Unity project do not have to use the compiler directives.

6.4.1 Proof of Concept Application

The Proof of Concept application for the WinML approach uses the same Unity base project as the Barracuda approach and the framworks can be switched using a checkbox in the editor. Similar to the benchmarking application the WinML related code is abstracted in a wrapper class so that WinML can easily be integrated in Unity. Furthermore, the image tensorization and normalization algorithm is modified since WinML expects tensors to be in the channel-first format whereas Barracuda expects channel-last. The inference time for one frame ranges is around 700ms in average. The application feels also really slow since the text streaks but is still usable.

6.4.2 Evaluation

This section evaluates the WinML approach using the criteria described in 5.1. Table 6.3 shows the results of the benchmarking application.

Table 6.3: Measurement results of approach 3

	Inference time	Accuracy	Top-1 error	Top-5 error
Python script on notebook	3803ms	89.03%	10.97%	0.17%
WinML Unity test app on HoloLens 2	4045ms	89.03%	10.97%	0.17%
WinML 2D UWP test app on HoloLens 2	3428ms	89.03%	10.97%	0.17%

1. **Ease of implementation:** Integrating WinML in Unity projects is really inconvenient since code which uses the WinML API has to be wrapped in `#if WINDOWS_UWP` compiler directives. Because of that debugging and autocompletion is not possible for WinML related code.

Furthermore, the app cannot be tested inside of Unity or with the holographic remoting player. It has to be exported and compiles as UWP app and installed on the HoloLens or the emulator every time in order to test the entire application. There is an official WinML documentation by Microsoft which contains a few example projects. However, there are only very few third party example projects and resources since WinML is used as often as e.g. TensorFlow.js.

2. **Performance:** The Unity WinML benchmarking application needs 4045ms for the entire Fashion-MNIST test dataset on the LeNet-5 model. The 2D UWP application is slightly faster with 3248ms. The inference time of the PoC application which uses the MobileNetV2 model is 700ms in average for one frame.
3. **Accuracy/Error rate:** The accuracy is 89.03%, the Top-1 error 10.97%, and the Top-5 error 0.17% measured by both WinML benchmarking applications and the Fashion-MNIST test datatset. These values are the same as the values reached by Barracuda and the Python version TensorFlow.
4. **Compatibility with common machine learning frameworks and pre-trained models:** WinML supports all ONNX NN models. CNTK, PyTorch and ML.Net can directly export to ONNX. Models trained by TensorFlow, Keras, Apple Core ML, and a few others can be converted using ONNXMLTools or WinMLTools which supports additional features such as quantization [CRC20].
5. **Integrability with frameworks or game engines:** WinML can be integrated in Unity projects and also used in plain Win32 or UWP apps. 3D MR apps can also be developed as UWP or Win32 app using the low-level HolographicSpace API or OpenXR API [TC21].
6. **Platform independency:** Applications developed with WinML can run on all Windows 10 platforms including desktop computers, Windows Mixed Realty (includes HoloLens), and XBox (only UWP).

This section showed that WinML is also a feasible way to use NNs on the HoloLens 2. WinML has the worst performance of all approaches since it currently cannot make use of GPU inference.

6.5 Approach 4: TensorFlow.NET

The last approach was supposed to TensorFlow.NET, which is a C# binding for TensorFlow. The API aims to be as similar as possible to the original TensorFlow Python API. Therefore, code can be easily translated from Python to C#. However, it turns out that TensorFlow.NET still relies on the original TensorFlow binaries which are officially only available for the x64 architecture but not for the ARM64 architecture [Sci21a; Sci21b]. Since the HoloLens 2 uses the ARM64 architecture, TensorFlow.NET cannot be used on the HoloLens 2. Therefore, it is not feasible to realize this approach. Anyway, this approach would have some severe drawbacks compared to the others. For example GPU inference would be not possible since the GPU version of the TensorFlow binaries used by TensorFlow.NET requires CUDA which is only available on NVIDIA GPUs and not on the HoloLens 2 [Sci21b].

6.6 Muscle visualization application

The superordinate project Pervasive Simulation and Visualization (PerSiVal) is about simulating and visualizing muscle activation on a HoloLens. A muscle model of an arm is overlayed on a real human arm. Different colours visualize the muscle activation. There is already an application for the HoloLens (1st gen) which implements this function. It is developed using the Unity game engine. The application uses HoloLensARToolKit to track the human arm using markers on shoulder, elbow and wrist. The calculation of the muscle activation values is currently realized using a static lookup table. Only four input values are used to find the output values which describe the activations of five different muscles. However, in the future it is planned to also simulate the deformation of the muscles which requires to increase the number of input as well as output values. This would make static lookup tables unfeasible. Therefore, the table should be replaced by an NN. There is already a small NN trained by Keras which can replace the lookup table.

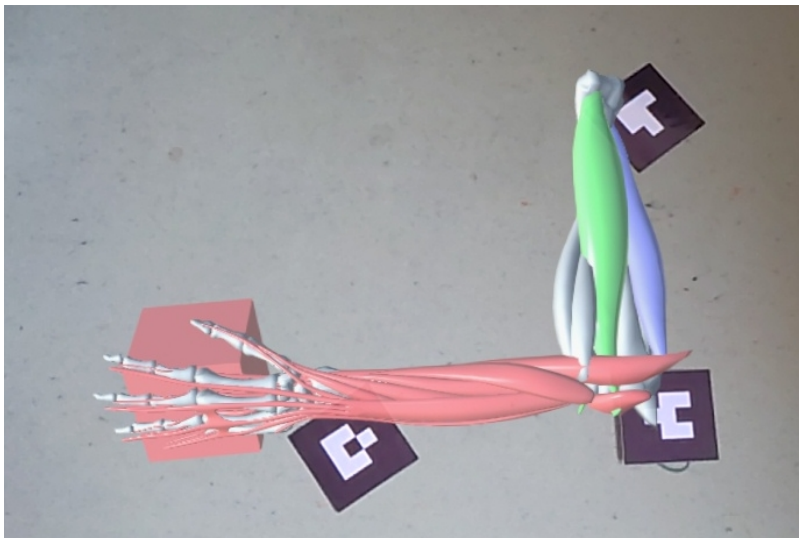

Figure 6.2: Muscle activation visualization application running on the HoloLens 2

The first step was to migrate the Unity project that it runs on the HoloLens 2. The Unity version has to be switched to Unity 2019 which supports the ARM64 architecture of the HoloLens 2. Therefore, the scripting backend has to be switched to IL2CPP since the .NET backend is deprecated. Furthermore, the project has to be configured to target the ARM64 architecture. Consistently, it has to be ensured that all dependencies are compatible with the ARM64 architecture. [Fer20] In this case the HoloLensARToolKit has to be replaced with a newer version which supports the HoloLens 2 and the new architecture. The C# code itself did also require some changes since the API of HoloLensARToolKit changed slightly. [Qia20]

The static table lookup is replaced by the NN which is also converted into the ONNX format. In order to compare the two approaches WinML and Barracuda are implemented and can be easily switched in the code. Similar to the benchmarking apps the WinML related code is abstracted using wrapper classes that they can easily be replaced. Since the NN only has numerical values as input no complex preprocessing and normalization is required. The input values only have to be put in array which is then tensorized using the according function of the used framework.

The final applications runs smoothly with both NN inference engines. However, Barracuda has an average evaluation time of under 1ms whereas WinML is considerably slower with values ranging between 15 and 20ms. Figure 6.2 shows the application running on the HoloLens 2.

6.7 Discussion and Comparison

To sum up three of the four approaches are feasible. Only approach four did not work since TensorFlow.NET relies on the TensorFlow binaries which are not available for Windows on the ARM architecture.

Approach one uses TensorFlow.js which can easily be integrated in JavaScript AR frameworks such as A-Frame. The accuracy results of the benchmarking application are similar to the values of the reference model in the Python script. The inference time for the test dataset was 1035ms. The proof of concept applications was easy to develop and runs very stable and smoothly. Inference for one frame takes around 80ms. One main advantage of TensorFlow.js is that it offers a powerful API which covers most of the functions of the normal version of TensorFlow and makes task such as preprocessing and normalization easy and efficient. The second main advantage is the platform independency. Applications developed with TensorFlow.js and a JavaScript AR framework can run on multiple other devices next to the HoloLens 2 without modification.

The second approach uses the Barracuda library for Unity. Accuracy and error rates are the same as the values of the reference model in the Python training script. The average inference time of 168ms is very fast thanks to the GPU backend. The implementation of the Proof of Concept application was less easy than witch the first approach since the image preprocessing and normalization has to be implemented by hand in C#. For that reason the applications also runs less smoothly although the inference time of 5-10ms is still very fast. The usage of Barracuda is limited to Unity projects. However, Barruda can run on each device which is supported by Unity and can use GPU inference in the most cases which makes it very fast.

The third approach makes use of the WinML framework. Both benchmarking applications produce the same accuracy results as the Barracuda app and the reference model in the Python script. The WinML Unity benchmarking application reached an average evaluation time of 4045ms whereas the 2D UWP application reached 3428ms. WinML is by far the slowest framework compared to the first to approaches since it currently cannot make use of GPU inference and only uses the CPU. Furthermore, integrating WinML in Unity projects is a hassle since the WinML related has to be wrapped in compiler directives and cannot be debugged. The advantage of WinML is that it can be used in any Win32 and UWP apps which can be useful if an application is developed using the low-level OpenXR API.

Which of the above ways is the best one mainly depends on which 3D framework or technologies in general are used for developing an application. If the application is developed using a JavaScript framework TensorFlow.js is the only way. If Unity is used the best way is using Barracuda. Barracuda is superior to WinML in most points. It is by far more performant since it can use the GPU and considerably easier to integrate in Unity projects. The only drawback of Barracuda compared to WinML is that Barracuda currently does not support all of the ONNX operations. If the application should be directly developed as UWP or Win32 app without Unity e.g. using

the OpenXR API WinML is the only way. Another point to consider is the platform dependency. The TensorFlow.js and Barracuda approaches are the most flexible since they both support a great variety of platforms whereas WinML is limited to Windows 10 devices.

This Bachelor thesis defined the evaluation criteria ease of implementation, accuracy, compability with ML frameworks and pre-trained models, integrability with game engines, and platform independency which make the different approaches comparable. Furthermore, it describes two new feasible approaches to use NNs on the HoloLens 2 applications (TensorFlow.js and Barracuda). The WinML approach was already used by others on the HoloLens 2 as mentioned in the Related Work section. However, this work shows that models trained by TensorFlow/Keras can also be used with WinML and the HoloLens 2. A limitation of this Bachelor thesis is that there are possibly further ways which are not covered. For example, it could by possible to compile the TensorFlow binaries by hand and integrate them directly in a C++ application. Furthermore, there are other web based ML frameworks such as Keras.js or ONNX.js which are not covered in this thesis.

Table 6.4 on page 51 lists the evaluation results of all approaches in a condensed form. The numbers in the first column correspond to the numbers of the evaluation criteria in section 5.1. In some cells such as supported platforms only the most important ones are listed. $\oplus$ and $\ominus$ are used to describe positive/negative aspects. The performance and error values refer to the benchmarking applications with the Fashion-MNIST dataset and LeNet-5 NN. Figure 6.3 depicts the general workflows of the approaches.

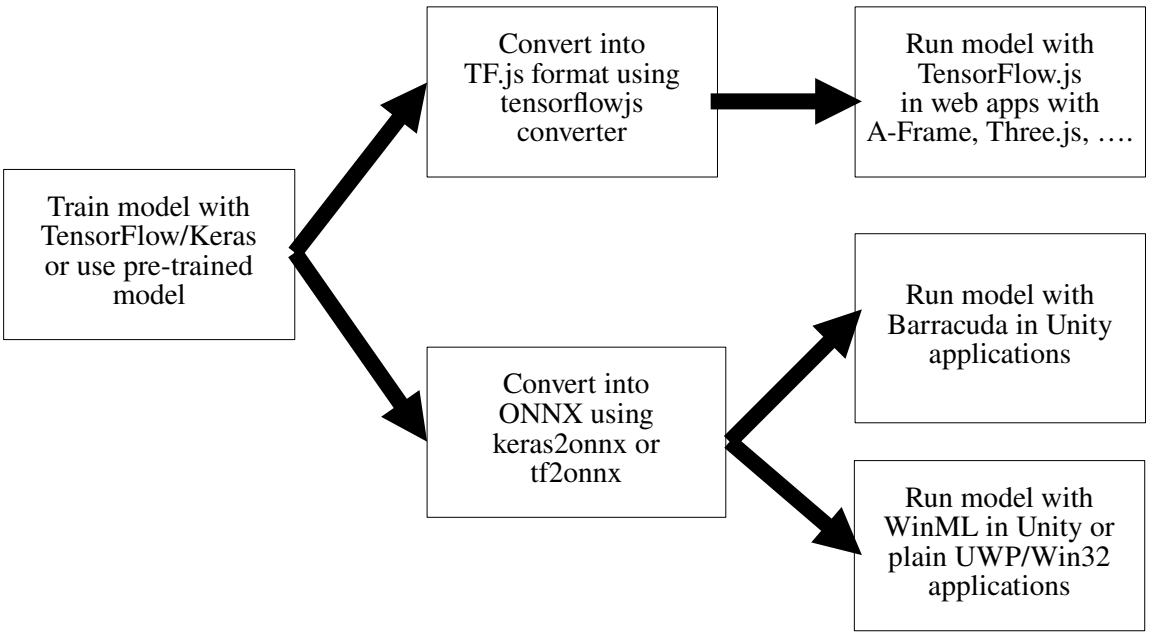

Figure 6.3: Workflows of the TensorFlow.js, Barracuda, and WinML approaches

Table 6.4: Comparison of the evaluation results of all approaches

		Approach 1: TensorFlow.js	Approach 2: Unity Barracuda	Approach 3: Windows ML
1	Ease of implementation	⊕: Many example projects and other third-party sources, Comprehensive API supporting many operations for preprocessing	⊖: Preprocessing sometimes needs to be implemented by hand	⊖: WinML related code in Unity cannot be debugged, Preprocessing sometimes needs to be implemented by hand
2	Performance (Inference time)	1035ms	183ms	4045ms (Unity), 3428ms (2D UWP)
3	Accuracy	90.7%	89.03%	89.03%
	Top 1 Error	9.29%	10.97%	10.97%
	Top 5 Error	0.1%	0.17%	0.17%
4	Compatible frameworks	TensorFlow, Keras	PyTorch After convertion: TensorFlow, Keras (Currently not all ONNX operators supported)	ONNX compatible frameworks: CNTK, PyTorch, ML.Net After convertion: TensorFlow, Keras, Apple Core ML, ...
	Sources of pre-trained models	Pre-trained TF.js models After convertion: TensorFlowF Model Garden, Keras applications	ONNX Model Zoo After conversion: TensorFlow Model Garden, Keras applications	ONNX Model Zoo After conversion: TensorFlow Model Garden, Keras applications
5	Compatible game engines and 3D frameworks	A-Frame, Three.js, Babylon.js	Unity	Unity
6	Compatible platforms	All WebXR compatible platforms: Windows Mixed Reality (HoloLens), HTC Vive, Oculus Rift, Samsung GearVR, ...	All platforms supported by Unity: Win/Lin/Max desktops, iOS/Android smartphones, Windows Mixed Reality (HoloLens), Oculus Rift, HTC Vive, Nintendo Switch, Playstation, ...	All Windows 10 devices: Desktops, Windows Mixed Reality (HoloLens), XBox (only UWP), ...

7 Conclusion and Outlook

The goal of this Bachelor thesis is to make different approaches of integrating NNs in HoloLens 2 application comparable by defining a set of diagnostic evaluation criteria. Furthermore, multiple approaches were poroposed, implemented, and evaluated using the aforementioned criteria. In the end all approaches were compared in an extensive overview which helps choosing which approach is the best for a specific applications. The basic requirements for all approaches are that the NN can be executed directly on the HoloLens 2 without needing neither an external device nor a WiFi connection. Furthermore, the approach has to be integratable in MR/AR applications and can use NNs trained by Keras/TensorFlow. All feasible ways which were found were evaluated using the defined criteria ease of implementation, accuracy, compability with ML frameworks and pre-trained models, integrability with game engines, and platform independency. The procedure for each approach was to implement a benchmark application which evaluates the Fashion-MNIST test dataset on the LeNet-5 reference model trained by a TensorFlow/Keras Python script. The benchmarking applications measure the inference time of the test dataset, the accuracy, the Top-1 and Top-5 error rates. Furthermore, for all working approaches a Proof of Concept application was developed in order to show that this approach cannot only be used with simple 2D applications but also with full AR/MR applications. The PoC applications use a MobileNetV2 to classify objects the user is looking at and display the result in the user's field of view.

In total four different possible approaches were proposed. The first one uses TensorFlow.js which is integrated in web applications, the second approach uses Barracuda in a Unity project, the third approach uses WinML in Unity and plain 2D UWP, and the last approach makes use of TensorFlow.NET. Three of the four approaches are feasible. Approach four which uses TensorFlow.NET was not possible to realize easily since it relies on the TensorFlow binaries which are not officially available for ARM64 on the Windows platform. All the other three approaches are working. Barracuda is by far the most performant since it can make use of GPU inference. TensorFlow.js follows after that and has still a sufficient performance since it also supports GPU inference through WebGL. The slowest one was WinML which currently only supports CPU inference on the HoloLens 2 due to its ARM64 architecture. The accuracy measurements of the LeNet-5 model on the Fashion-MNIST test dataset showed that TensorFlow.js has a slightly higher accuracy as the reference model in the TensorFlow Python training script. Barracuda and WinML both reach exactly the same accuracy as the reference model. TensorFlow.js can be easily integrated in JavaScript AR frameworks such as A-Frame. Furthermore, such apps are platform independent and can run also on other devices which support JavaScript web applications and WebXR. Barracuda can only be integrated in Unity projects. Such projects can run on any device which is supported by Unity. WinML can be used in Unity projects as well as plain UWP and Win32 apps and can run on every Windows 10 device. Integrating WinML in Unity is inconvenient sice the use of compiler directives around the WinML is required whereby this code cannot be debugged in the Unity editor. To conclude the evaluation shows that Barracuda is the best option since it is the most performant and can be easily integrated in Unity projects and can also run on platforms other than

the HoloLens 2. TensorFlow.js is also a good option if JavaScript AR frameworks used in the project since it has a powerful API and is platform independent. WinML should be only considered if plain UWP or Win32 have to be developed in the project e.g. to directly use the low-level OpenXR APIs. In Unity projects Barracuda is superior since it easier to integrate and remarkably more performant. The only possible disadvantage of Barracuda is that it currently does not support all the ONNX operations.

This Bachelor thesis makes approaches for integrating NN in HoloLens 2 applications comparable by specifying multiple evaluation criteria. Furthermore, in total three different feasible approaches are implemented and described in detail. All of these allow to use models trained by TensorFlow/Keras, work completely locally meaning they do neither require a WiFi connection nor an external device. These results can now be used to make an informed decision which approach is the best for a specific application. Moreover, integrating NNs in any HoloLens 2 project is now possible. The PerSiVal project can now not only replace the static lookup table of the muscle activations with a NN but also make use of more heavyweight simulations in the muscle visualization application on the HoloLens 2.

7.1 Outlook

In the future further developments of the used ML frameworks could overcome some of the limitations of the aforementioned approaches. One example is the release of DirectML for the ARM64 architecture. Then WinML could also make use of GPU inference on the Microsoft HoloLens 2 which would be more performant. Furthermore, Barracuda could support more of the ONNX operations and more model architectures in the future. This would enable to use further model architectures such as single-shot detector models.

Future research could investigate if there are more ways apart from these described in this thesis. For example there could be a way to compile the TensorFlow binaries by hand and integrate them in applications developed in C++ or modify TensorFlow.NET in order to use the compiled binaries. Furthermore, it could be investigated if distributed approaches where the results of a locally running NN are combined with values of a more complex external NN are advantageous in terms of accuracy and performance. The PerSiVal project could now also investigate if there are further applications of NNs for the muscle activation visualization application apart form simulating. For example could a NN track the arm instead of the ArToolkit which would make the paper markers redundant.

Bibliography

[A-F] A-Frame. *HelloWorld*. URL: https://aframe.io/examples/showcase/helloworld/ (cit. on p. 40).

[A-F20] A-Frame. *Introduction*. Apr. 7, 2020. URL: https://aframe.io/docs/1.2.0/introduction/ (cit. on pp. 25, 41).

[AAB+16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. 2016. arXiv: 1603.04467 [cs.DC] (cit. on pp. 20, 21).

[Ash20] A. Ashikhmin. *TFClassify-Unity-Barracuda*. Feb. 2, 2020. URL: https://github.com/Syn-McJ/TFClassify-Unity-Barracuda (cit. on p. 43).

[BH69] A. E. Bryson, Y.-C. Ho. *Applied optimal control : optimization, estimation, and control*. Waltham, Mass. : Blaisdell Pub. Co., 1969 (cit. on p. 15).

[Bov19a] S. Bovo. *Back to the future now: Execute your Azure trained Machine Learning models on HoloLens!* Microsoft. Jan. 15, 2019. URL: https://techcommunity.microsoft.com/t5/windows-dev-appconsult/back-to-the-future-now-execute-your-azure-trained-machine/ba-p/318077 (cit. on pp. 29, 30).

[Bov19b] S. Bovo. *HoloLens-Windows-MachineLearning*. May 1, 2019. URL: https://github.com/sbovo/HoloLens-Windows-MachineLearning (cit. on p. 29).

[Bra20] B. Bray. *What is Mixed Reality?* Microsoft. Aug. 26, 2020. URL: https://docs.microsoft.com/de-de/windows/mixed-reality/discover/mixed-reality (cit. on p. 18).

[Bro19] J. Brownlee. *How to Manually Scale Image Pixel Data for Deep Learning*. July 5, 2019. URL: https://machinelearningmastery.com/how-to-manually-scale-image-pixel-data-for-deep-learning/ (cit. on p. 17).

[Cha20] C. Chaoweeraprasit. *DirectML v1.4 ARM64 version?* Dec. 17, 2020. URL: https://github.com/microsoft/DirectML/issues/63 (cit. on p. 44).

[CMP20] S. Cooley, E. Miller, S. Paniagua. *HoloLens 2 hardware*. Microsoft. Oct. 20, 2020. URL: https://docs.microsoft.com/en-us/hololens/hololens2-hardware (cit. on pp. 19, 20).

[CRC20] W.-S. Chin, Q. Radich, E. Cowley. *Convert ML models to ONNX with WinMLTools | Microsoft Docs*. Microsoft Inc. May 13, 2020 (cit. on pp. 23, 44, 46).

[Dav20] M. Davaadorj. *3D-Posenet*. Aug. 10, 2020. URL: https://github.com/mishig25/3d-posenet (cit. on p. 41).

[Fer20] H. Ferrone. *Porting von HoloLens-Apps (1. Generation) zu HoloLens 2 - Mixed Reality | Microsoft Docs*. Microsoft. Dec. 9, 2020. URL: https://docs.microsoft.com/de-de/windows/mixed-reality/develop/porting-apps/porting-hl1-hl2 (cit. on p. 47).

[GBC16] I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on p. 15).

[Goo] Google. *Teachable Machine*. URL: https://teachablemachine.withgoogle.com/ (cit. on p. 22).

[HKL20] T. Hubatscheck, J. Kurzweg, L. Lazar. "Reducing Neural Networks for MobileDevices". June 11, 2020 (cit. on p. 30).

[HZC+17] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. Apr. 17, 2017. arXiv: 1704.04861 [cs.CV] (cit. on p. 26).

[Kat] H. Kato. *ARToolKit*. HIT Lab, University of Washington. URL: http://www.hitl.washington.edu/artoolkit/ (cit. on p. 25).

[KCG20] A. Khurshid, S. Cleger, R. Grunitzki. "A Scene Classification Approach for Augmented Reality Devices". In: *HCI International 2020 – Late Breaking Papers: Virtual and Augmented Reality*. Springer International Publishing, 2020, pp. 164–177. DOI: 10.1007/978-3-030-59990-4_14 (cit. on p. 29).

[Ker] Keras. *Keras: The Python Deep Learning library*. URL: https://keras.io/ (cit. on p. 22).

[KNGG18] M. Kowalski, Z. Nasarzewski, G. Galinski, P. Garbat. *HoloFace: Augmenting Human-to-Human Interactions on HoloLens*. 2018. arXiv: 1802.00278 [cs.CV] (cit. on p. 29).

[LBBH98] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86 (11 1998), pp. 2278–2324. ISSN: 1558-2256. DOI: 10.1109/5.726791 (cit. on pp. 25, 26).

[Mic20] Microsoft. *Onnxruntime*. Mar. 2020. URL: https://github.com/microsoft/onnxruntime (cit. on p. 23).

[Mic21] Microsoft. *Windows ML*. Mar. 3, 2021. URL: https://github.com/Microsoft/Windows-Machine-Learning (cit. on p. 24).

[Moz21a] Mozilla. *How to make PWAs installable - Progressive web apps (PWAs)*. Feb. 24, 2021. URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Installable_PWAs (cit. on p. 40).

[Moz21b] Mozilla. *Making PWAs work offline with Service workers - Progressive web apps (PWAs)*. Apr. 1, 2021. URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Offline_Service_workers (cit. on p. 40).

[MP43] W. S. McCulloch, W. Pitts. *A logical calculus of the ideas immanent in nervous activity*. 1943. DOI: 10.1007/bf02478259 (cit. on p. 15).

[MS20] R. Magoulas, S. Swoyer. *AI Adoption in the Enterprise 2020*. O'Reilly Media, Inc., 2020. URL: https://get.oreilly.com/ind_ai-adoption-in-the-enterprise-2020.html (cit. on p. 31).

[MTUK95] P. Milgram, H. Takemura, A. Utsumi, F. Kishino. "Augmented reality: a class of displays on the reality-virtuality continuum". In: *Telemanipulator and Telepresence Technologies*. Ed. by H. Das. Vol. 2351. International Society for Optics and Photonics. SPIE, 1995, pp. 282–292. DOI: 10.1117/12.197321. URL: https://doi.org/10.1117/12.197321 (cit. on p. 18).

[Nie15] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL: http://neuralnetworksanddeeplearning.com/ (cit. on pp. 16, 17).

[NTEY18] S. Naritomi, R. Tanno, T. Ege, K. Yanai. "FoodChangeLens: CNN-Based Food Transformation on HoloLens". In: *2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*. 2018, pp. 197–199. DOI: 10.1109/AIVR.2018.00046 (cit. on p. 29).

[ON15] K. O'Shea, R. Nash. *An Introduction to Convolutional Neural Networks*. Nov. 26, 2015. arXiv: 1511.08458 [cs.NE] (cit. on p. 17).

[ONN] ONNX. *SUPPORTED TOOLS*. URL: https://onnx.ai/supported-tools.html (cit. on p. 23).

[ONN20] ONNX. *ONNX*. Mar. 2020. URL: https://github.com/onnx/onnx (cit. on p. 23).

[PI95] N. Peter, R. S. A. Intelligence. *A Modern Approach*. 1995 (cit. on p. 15).

[QDK18] L. Qian, A. Deguet, P. Kazanzides. "ARssist: augmented reality on a head-mounted display for the first assistant in robotic surgery". In: *Healthcare technology letters* 5.5 (2018), pp. 194–200 (cit. on p. 25).

[Qia20] L. Qian. *HoloLensARToolKit*. Apr. 19, 2020. URL: https://github.com/qian256/HoloLensARToolKit (cit. on pp. 25, 47).

[Rak] B. Rakowski. *Pixel 4 is here to help*. Google. URL: https://blog.google/products/pixel/pixel-4/ (cit. on p. 13).

[Ros] V. Rosane. *Introduction to Windows Machine Learning*. Microsoft. URL: https://docs.microsoft.com/en-us/windows/ai/windows-ml/ (cit. on pp. 23, 24).

[Ros57] F. Rosenblatt. "The perceptron: A probabilistic model for Visual Perception". In: *Proceedings of the 15th International Congress of Psychology, North Holland*. 1957, pp. 290–297 (cit. on p. 15).

[Rui21] H. Ruiz. *RiggingJs*. Mar. 23, 2021. URL: https://github.com/haruiz/RiggingJs (cit. on p. 41).

[Sci21a] SciSharp. *SciSharp.TensorFlow.Redist*. Feb. 13, 2021. URL: https://www.nuget.org/packages/SciSharp.TensorFlow.Redist/ (cit. on p. 46).

[Sci21b] SciSharp. *TensorFlow.NET*. Feb. 20, 2021. URL: https://github.com/SciSharp/TensorFlow.NET (cit. on p. 46).

[SHZ+18] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 4510-4520* (Jan. 13, 2018). arXiv: 1801.04381 [cs.CV] (cit. on pp. 26, 27).

[Sim18] O. Simeone. *A Very Brief Introduction to Machine Learning With Applications to Communication Systems*. Aug. 7, 2018. arXiv: 1808.02342 [cs.IT] (cit. on p. 15).

[Stu20] D. Stutz. *Illustrating (Convolutional) Neural Networks in LaTeX with TikZ*. June 2, 2020. URL: https://davidstutz.de/illustrating-convolutional-neural-networks-in-latex-with-tikz/ (cit. on p. 16).

[TC21] A. Turner, D. Coulter. *Native development overview - Mixed Reality | Microsoft Docs*. Aug. 4, 2021. URL: https://docs.microsoft.com/en-us/windows/mixed-reality/develop/native/directx-development-overview (cit. on p. 46).

[Tena] TensorFlow. *Case Studies and Mentions*. URL: https://www.tensorflow.org/about/case-studies (cit. on p. 21).

[Tenb] TensorFlow. *Platform and environment*. URL: https://www.tensorflow.org/js/guide/platform_environment (cit. on p. 22).

[Tenc] TensorFlow. *TensorFlow.js Examples*. URL: https://github.com/tensorflow/tfjs-examples/blob/master/mnist-core/data.js (cit. on p. 38).

[Ten19] TensorFlow. *TensorFlow 2.0 is now available!* Sept. 30, 2019. URL: https://blog.tensorflow.org/2019/09/tensorflow-20-is-now-available.html (cit. on p. 21).

[Ten20a] TensorFlow. *TensorFlow*. Mar. 2020. URL: https://github.com/tensorflow/tensorflow (cit. on p. 21).

[Ten20b] TensorFlow. *TensorFlow.js*. Mar. 2020. URL: https://github.com/tensorflow/tfjs (cit. on p. 21).

[Ten21] TensorFlow. *Pre-trained TensorFlow.js models*. Feb. 18, 2021. URL: https://github.com/tensorflow/tfjs-models/blob/master/README.md (cit. on p. 41).

[Unia] Unity. *Multiplatform | Unity*. URL: https://unity.com/features/multiplatform (cit. on pp. 24, 44).

[Unib] Unity. *Unity*. URL: https://unity.com/ (cit. on p. 24).

[Uni20a] Unity. *Exporting your model to ONNX format | Barracuda | 1.0.4*. Dec. 11, 2020. URL: https://docs.unity3d.com/Packages/com.unity.barracuda@1.0/manual/Exporting.html (cit. on pp. 24, 42).

[Uni20b] Unity. *Introduction to Barracuda | Barracuda | 1.0.4*. Unity. Dec. 11, 2020. URL: https://docs.unity3d.com/Packages/com.unity.barracuda@1.0/manual/index.html (cit. on pp. 24, 43).

[Uni20c] Unity. *Supported neural architectures and models | Barracuda | 1.0.4*. Dec. 11, 2020. URL: https://docs.unity3d.com/Packages/com.unity.barracuda@1.0/manual/Exporting.html (cit. on p. 24).

[Uni20d] Unity. *Supported platforms | Barracuda | 1.0.4*. Dec. 11, 2020. URL: https://docs.unity3d.com/Packages/com.unity.barracuda@1.0/manual/SupportedPlatforms.html (cit. on pp. 24, 44).

[Uni20e] Unity. *Working with data | Barracuda | 1.0.4*. Dec. 11, 2020. URL: https://docs.unity3d.com/Packages/com.unity.barracuda@1.0/manual/AdvancedTopics.html (cit. on p. 24).

[VRMC21] R. Vallim, Q. Radich, J. Martinez, E. Cowley. *Release notes | Microsoft Docs*. Microsoft. Aug. 4, 2021. URL: https://docs.microsoft.com/en-us/windows/ai/windows-ml/release-notes (cit. on pp. 23, 44).

[XRV17] H. Xiao, K. Rasul, R. Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Zalando Research. Aug. 28, 2017. arXiv: cs.LG/1708.07747 [cs.LG] (cit. on p. 27).

[Zel19] M. Zeller. *Release notes - May 2019 - Mixed Reality*. Microsoft. July 11, 2019. URL: https://docs.microsoft.com/de-de/windows/mixed-reality/enthusiast-guide/release-notes-may-2019 (cit. on pp. 18, 20).

[ZHP20] H. Zeng, X. He, H. Pan. "Implementation of escape room system based on augmented reality involving deep convolutional neural network". In: *Virtual Reality* (Oct. 2020). DOI: 10.1007/s10055-020-00476-0 (cit. on p. 29).

[ZMP19] M. Zeller, E. Miller, S. Paniagua. *HoloLens (1st gen) hardware*. Microsoft. Sept. 16, 2019. URL: https://docs.microsoft.com/en-us/hololens/hololens1-hardware (cit. on pp. 18, 19).

All links were last followed on May 2, 2021.

A German Abstract

Ziel der vorliegenden Bachelorarbeit ist es, verschiedene Ansätze, Neuronale Netze auf der HoloLens 2 auszuführen, vergleichbar zu machen. Um dies zu ermöglichen, werden qualitative und quantitative Kriterien festgelegt. Am Ende werden alle Ansätze in einer ausführlichen Übersicht verglichen. Darüber hinaus werden mehrere verschiedene Ansätze, die Integration zu realisieren, vorgeschlagen, implementiert und mit den eben genannten Kriterien evaluiert. Die grundlegenden Anforderungen sind, dass Neuronale Netze die mit TensorFlow/Keras trainiert wurden verwendet und direkt auf der HoloLens 2 ohne eine aktive WLAN-Verbindung ausgeführt werden können. Außerdem sollen die Neronalen Netze in Mixed/Augumented Reality Anwendungen integrierbar sein. Insgesamt werden vier Ansätze vorgestellt: TensorFlow.js, Unity Barracuda, TensorFlow.NET. und Windows Machine Learning, welches bereits in Verwendung auf der HoloLens 2 ist. Für jeden lauffähigen Ansatz werden Testanwendungen entwickelt, die einen Testdatensatz auf einem gemeinsamen Referenzmodel ausführen und die Ausführungszeit sowie die Genauigkeit messen. Darüber hinaus werden kleine Proof of Concept Anwendungen entwickelt um zu zeigen, dass der Ansatz auch in Augmented bzw. Mixed Reality Anwendungen integrierbar ist. Die PoC Applikationen nutzen ein MobileNetV2 Modell, um Bilder von der Webcam zu klassifizieren und das Ergebnis dem Nutzer anzuzeigen. Alle realisierbaren Ansätze werden angesichts verschiedenerer Kriterien wie Einfachheit der Implementierung, Performance, Genauigkeit, Kompatibilität mit Machine Learning Frameworks und vor-trainieren Modellen und Integrierbarkeit in 3D Frameworks evaluiert. Die Barracuda, TensorFlow.js und Windows ML Ansätze stellten sich als realisierbar heraus. Barracuda, das nur in Unity Anwendungen verwendet werden kann, ist das performanteste Framework, da es die GPU zur Ausführung nutzen kann. Danach folgt TensorFlow.js das in JavaScript 3D Frameworks wie A-Frame integriert werden kann. Windows ML kann derzeit nur die CPU auf der HoloLens 2 nutzen und ist daher das langsamste. Es kann in UWP und Win32 und mit einigen Schwierigkeiten in Unity Anwendungen integriert werden. Barracuda und WinML werden auch in eine biomechanische Visualisierungs-Anwendung auf Unity-Basis integriert, um darin Simulationen zu berechnen. Die Ergebnisse dieser Bachelorarbeit ermöglichen es verschiedene Ansätze, Neuronale Netze auf der HoloLens 2 auszuführen, zu vergleichen. Nun kann eine fundierte Entscheidung getroffen werden, welcher Ansatz für eine bestimmte Anwendung am besten geeignet ist. Darüber hinaus wurde gezeigt, dass die Barracuda und TensorFlow.js Ansätze praxistauglich sind und dem bestehenden Windows ML Ansatz in den meisten Punkten überlegen sind.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, May 5, 2021 

place, date, signature