HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Vladislav Förster

Evaluierung der Microsoft HoloLens für den Einsatz der KI-basierten forensischen Personenanalyse

2022

Fakultät **Biotechnik**

BACHELORARBEIT

Evaluierung der Microsoft HoloLens für den Einsatz der KI-basierten forensischen Personenanalyse

Autor:
Vladislav Förster

Studiengang:
Allgemeine und Digitale Forensik

Seminargruppe:
FO16w3

Erstprüfer:
Sven Becker

Zweitprüfer:
Dirk Labudde

Mittweida, 2022

Bibliografische Angaben

Förster, Vladislav: Evaluierung der Microsoft HoloLens für den Einsatz der KI-basierten forensischen Personenanalyse, 31 Seiten, 19 Abbildungen, Hochschule Mittweida, University of Applied Sciences, Fakultät Biotechnik

Bachelorarbeit, 2022

Referat

Diese Arbeit betrachtet verschiedene Methoden aus dem Machine-Learning, geht im genaueren auf Deep-Neural-Networks ein und versucht diese für die forensische Personenanlyse einzusetzen. Gestützt wurde sich auf auf das MediaPipe-Framework.
Die HoloLens 1 schränkte die Möglichkeiten darauf ein, Bildaufnahmen auf ihr zu tätigen und die Inferenz an einer externen Maschine auszuführen. Um die Möglichkeit der Inferenz auf AR-Geräten zu ermöglichen weiter zu untersuchen, wurden die aktuelle Forschung im Bereich von Edgie-AI betrachtet.
Die HoloLens 1 bietet keine Möglichkeit die Inferenz der aktuellen Frameworks auf dem Gerät durchzuführen. Es konnte eine Inferenz auf einer externen Maschine durchgeführt werden und an die HoloLens übertragen werden. Weiter wurde die aktuelle Forschung betrachtet, welche sich mit spezialisierten Hardware für die Inferenz von NN betrachtet.

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Tabellenverzeichnis

IV. Abkürzungsverzeichnis

AF	Aktivierungsfunktion
CNN	Convolutional-Neural-Network
Conv	Convolutional
DL	Deep-Learning
DNN	Deep-Neural-Network
FLOP	Floating-Point-Operation
IMU	Inertial-Measurement-Unit
ML	Machine-Learning
NN	Neuronales-Netz
RAM	Random-Access-Memory
RNN	Recurrent-Neural-Network

1 Einleitung

Immer mehr Aufgaben werden heutzutage automatisiert. Automatisierung meinte vor einigen Jahren noch, dass ein Programm geschrieben wird in dem genau beschrieben wird wie das Ergebnis erzielt werden soll. Erkentnisse in dem Bereich der Künstlichen-Intelligenz, dem Machine-Learing (ML), Neuronalen-Netzen (NN) und im speziellen dem Deep-Learning haben zu einem Paradigmenwechsel geführt. Eingabedaten werden an ein NN übergeben und die erwartete Ausgabe wird spezifiziert. Das NN lernt anschließend wie dieses Ergebnis erreicht werden kann [15]. Ein Grund für die steigende Beliebtheit von NN ist, dass Grafikkarten immer Leistungsfähiger werden und der Lernprozess dadurch schneller möglich ist.
NN sind besonders gut für Daten geeignet, in denen Eigenschaften nur schwer explizit beschrieben werden können und/oder sehr stark variieren. Besonders erwähnenswert sind hier Natural-Language-Processing (NLP) und Computer-Vision.
Smartphones haben zu einer Flut an Bild- und Videodaten geführt und diese bieten sich sehr gut als Trainings- und Testdaten für NN an.

Zuerst wird in 2.1 die HoloLens und HoloLens 2 betrachtet. In 2.2 wird kurz auf künstlichen Intelligenz im Allgemeinen eingegangen. Anschließend wird in 2.3 der Aufbau von Neuroanlen-Netzen erklärt und in 2.4 die Besonderheiten von Convolutional-Neural-Networks.
In 2.5 werden die ML-Lösungen von MediaPipe untersucht. Die Implementierung von Face-Mesh und Pose-Estimation wird in 3 beschrieben. Die Ergebnisse der Implementierung werden in 4 ausgewertet. In 5 werden die aktuellen Hardwareansätze betrachtet, welche die Arbeit mit DNNs beschleunigen sollen.

1.1 Motivation

Um eine Personenanalyse durchzuführen müssen bisher Bilder mit den Verdächtigen aufgenommen werden und anschließend an einem Rechner verarbeitet werden. Sollten die Verarbeitungsprogramme jedoch Schwierigkeiten mit den Bilder haben, können diese weitreichende Folgen mit sich bringen.

1.2 Zielstellung

Um die oben genannte Schwierigkeit zu verhindern, ist die Echtzeitverarbeitung praktisch. Verbunden mit einer AR-Brille kann die Arbeit stark erleichtert werden , direktes Feedback gegeben werden und auch ungeschultes Personal mit der Aufgabe beauftragt werden.

2 Theoretische Vorgehensweise

2.1 Hololens

Die Hololens 1 war Microsoft erste AR-Brille, wurde 2016 veröffentlicht und war nicht an Endkonsumenten gerichtet. Die Hololens 2 wurde 2019 veröffentlicht, wurde in allen Bereichen verbessert und richtete sich nun auch an Unternehmen.

	HoloLens 1	HoloLens 2
Kamera	1408x792 Foto, 1216x684 24fps Video, TOF-Kamera	3904x2196 Foto, 1920x1080 30fps Video, TOF-Kamera, 2 IR-Kameras
Sichtbereich	34°	52°
Architektur	x86 32-Bit	ARM 64-Bit
CPU-Kerne	4	8
RAM	2GB	4GB
Sensoren	IMU, 4 Mikofone, 1 Lichtsensor	IMU (Accelerometer, Gyroskop, Magnetometer), 5 Mikrofone

Tabelle 2.1: Vergleich relevanter Kenndaten HoloLens 1 und HoloLens 2 [18, 19]

Die Entwicklung für die HoloLens kann mit Hilfe der Unity-Engine, Unreal-Engine, OpenXR oder WebXR erfolgen [20]. Die Unreal Engine und OpenXR werden nur mit der HoloLens 2 unterstützt [7, 13].
Den Einstieg in die Mixed-Reality (MR) Entwicklung erleichtert Microsfot mit dem ”Mixed Reality Toolkit” (MRTK). Dabei handelt es sich um eine Sammlung von Werkzeugen wie z.B. C# Scripten, grafische Bauelemente oder Shadern [21]. Dieses Toolkit ist als Repository auf der microsoft Github-Seite für Unity und Unreal verfügbar [17].

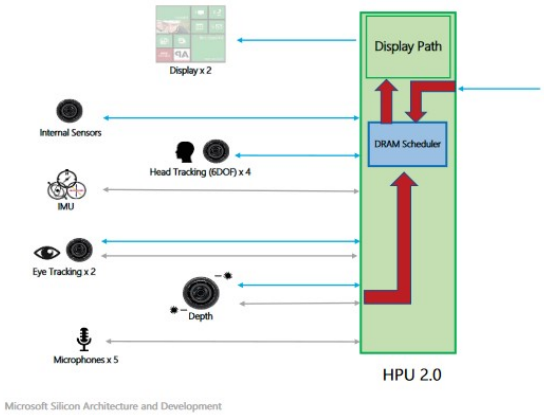


Abbildung 2.1: Verarbeitung der Sensordaten durch die HPU in der HoloLens 2 [29]

Eine Besonderheit in der Hardware der HoloLens ist die Holographic Processing Unit (HPU). Dabei handelt es sich um eine spezielle, von Microsoft entwickelte, Verarbeitungseinheit. Abb 2.1 zeigt wie alle Sensordaten an die HPU übermittelt und verarbeitet werden [29].

2.2 KI-Methoden im allgemeinen

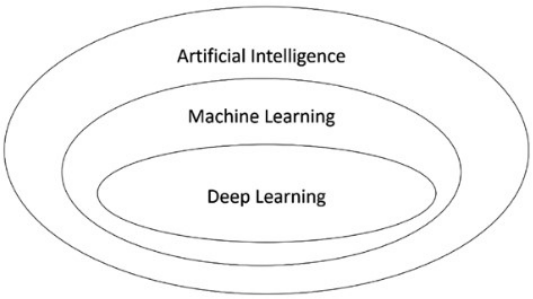


Abbildung 2.2: Gebiete im Bereich KI [28]

Mit Künstlicher Intelligenz wird die Idee beschrieben, bei welcher Maschinen menschliches Denken und Lernen nachahmen und somit auch sehr komplexe Aufgaben bewältigen können. [30]

In dem Teilgebiet ML wird auf Methoden eingegangen die es Maschinen ermöglichen zu lernen und die Genauigkeit von Voraussagen zu verbessern. Wichtige Gebiete aus der Mathemtik, die hier verwendet werden, sind daher die Statistik, um die Voraussagen zu machen, und die Optimierung, um die Voraussagen zu verbessern. Das eigentliche lernen erfolgt während des Trainings, in welchem die Parameterwerte angepasst werden.
Es wird zwischen überwachtem Lernen und unüberwachtem Lernen unterschieden.
Überwachtes Lernen ist die am häufigsten verwendete Methode und verwendet Daten bei denen ein Label hinzugefügt wurde und somit das gewünschte Ergebnis bekannt ist. Den Daten muss diese Markierung jedoch von einenm Mensch unter hohem Zeitaufwand hinzufügt werden.
Unüberwachtes Lernen nutzt Daten ohne Label und versucht die gegebenen Daten selbstständig in Gruppen einzuordnen.
Wird während des lernens ein Abbruchskriterium erreicht, kann das Modell mit den aktuellen Parameterwerten gespeichert werden und zur eigentlich Anwendung auf einem Gerät genutzt werden.
Je nach Komplexität muss die Hardware bestimmte Voraussetzungen erfüllen. Viele Methoden können parallel ausgeführt werden, weshalb sich eine leistungsstarke GPU besonders gut eignet. Einige Anwendungen, wie das Recommendation System von Facebook, benötigt dagegen eine große Menge an dynamischem Speicher und viele CPUs

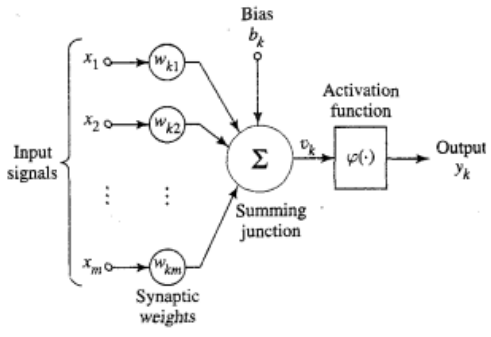
[3] [28, Kapitel 2][15].

Das eigentliche finden der Feature ist als Feature-Learning oder Representation-Learning bekannt. Die Rohdaten können in ihrer Ausgangsform an die Maschine übergeben werden und aus ihnen werden zu beginn grobe Feature gefunden, mit jedem weiteren Verarbeitungsschritt werden anschließend detailliertere Feature gefunden und sind eine Kombination aus den vorherigen Schritten. In Bildern können im ersten Schritt beispielsweise Kanten gefunden werden, anschließend werden möglicherweise die gemeinsam auftretenden Kanten gefunden welche komplexere Formen darstellen und in weiteren Schritten dann Kombinationen aus diesen Formen die ein gesamtes Objekt darstellen. Dieses Vorgehen ist ein Merkmal des Deep-Learning und erfordert viele Hidden-Layer in einem Neuronalen Netz [15].

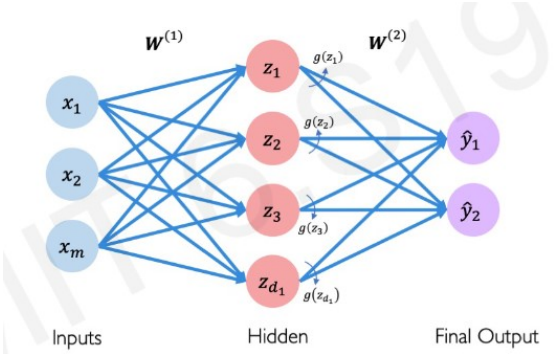
2.3 Neuronale Netze

Neuronale-Netze sind die state-of-the-art Methode in vielen Bereichen des ML. Sie basieren auf dem selben Mechanismus wie das menschliche Gehirn. Der Grundbaustein ist das Perzeptron welches Eingabedaten x_m erhält. Jeder Datenpunkt wird mit einem Gewicht w_{km} multipliziert und am Addierer $\sum$ anschließend mit einem Bias b_k addiert. Eine nichtlineare-Aktivierungsfunktion $\varphi(\cdot)$ nimmt diesen Wert als Eingabe und liefert die Ausgabe y_k (Abb. 2.3a) [11]. Die Ausgabe lässt sich zusammengefasst wie folgt berechnen

$$y_k = \varphi(b_k + \sum_{i}^{m} x_i w_i)$$



(a) Aufbau eines Perzeptrons in NN [11, Seite 33]



(b) Mehrere Neuronen bilden einen Hidde-Layer [2]

Es werden mehrere Neuronen in einem sogennanten Layer-Hidden zusammengefasst. Die gesamte Architektur besteht somit aus einem Input-Layer, mehreren Hidden-Layern und einem Output-Layer (Abb. 2.3b). Um das Netzwerk zu beschreiben werden die Begriffe depth und width verwendet. Wird von depth gesprochen, ist die Anzahl der Layer in einem Netzwerk gemeint. Wird von width gesprochen, ist die Anzahl an Neuronen in

den Layern gemeint.
Die Vorhersage, also das Durchlaufen der Daten vom Input-Layer durch die Hidden-Layer bis zum Output-Layer, wird Forward-Pass bezeichnet.
Um die Gewichte anzupassen wird eine Optimierung durchgeführt bei der, der Fehler des Ouput-Layers mit einer Loss-Function berechnet wird. Anschließend wird mit Hilfe der Kettenregel der Gradient berechnet und die Gewichte auf Grundlage dieser angepasst [31].

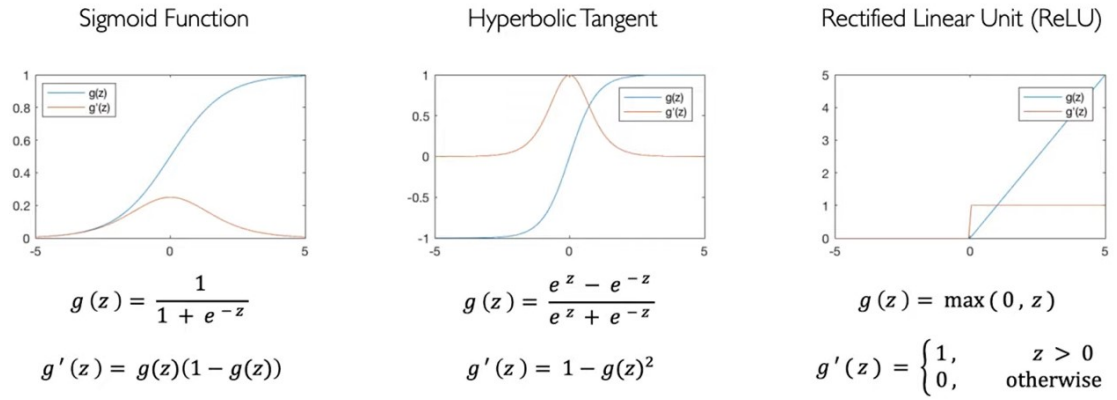


Abbildung 2.4: Die Graphen der Aktivierungsfunktionen: Sigmoidfunktion, Hyperbeltangensfunktion und Rectified-Linear-Unit (ReLU) [2]

Es existieren verschiedene Aktivierungsfunktionen welche unterschiedliche Werte für den Gradient liefern. Sie daher einen direkten Einfluss darauf wie schnell das NN konvergiert.

Die Sigmoidfunktion ist eine leicht verständliche AF, liefert überall einen positive Gradienten, wird für binäre Klassifikation oder logistische Regression verwendet und eignet sich somit für flache NN.
Der relativ kleine Gradient führt es einer langsame Konvergenz und kann im schlimmsten Fall zum Problem des verschwindenden Gradienten führen. Ein weiterer Nachteil ist, dass sie nicht Nullzentriert ist.
Verschiedene Variationen der Sigmoidfunktion wie SiLU oder dSiLU versuchen die Nachteile zu beseitigen.

Die Hyperbeltangensfunktion (tanh) konvergiert , im Vergleich zur Sigmoidfunktion, durch den größeren Gradienten schneller und ist Nullzentriert. Die Berechnung des Gradienten ist jedoch komplizierter und das Problem des verschwindenden Gradienten besteht hier auch.

Die Reactified-Linear-Unit hat einen konstanten Gradienten der 1 beträgt, beseitigt somit das Problem des verschwindenden Gradienten und konvergiert schnell. Ein weiterer

Vorteil ist, dass keine rechenintensiven Exponentialfunktionen berechnet werden müssen wie in den vorherigen AFs. Die ReLU führt auch dazu, dass schwach besetzte Matrizen erzeugt werden und die notwendige Rechenleistung verringern.
Es gibt jedoch das ”sterbende ReLU Problem”. Ist der Gradient für jeden Eingang an einem Neuron 0 wird keine Änderung mehr an den Gewichten vorgenommen, somit gibt es auch keine Verbesserung und das Neuron wird ”totes Neuron” genannt.
Variationen wie Leaky-ReLU, Parametric-ReLU oder Exponential-LU sind Variationen um dieses Problem zu lösen.

Die Softmaxfunktion wird im Output-Layer für Multiclass-Classification verwendet und ist gegeben durch

$$g(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)}$$

Abb. 2.5 listet noch weitere AFs auf, die im Bereich Deep-Learning verwendet werden, hier aber nicht weiter betrachtet werden.

2.4 Convolutional Neural Networks

Für Computer-Vision haben sich die sogennanten Convolutional-Neural-Networks (CNN) als bestes Verfahren etabliert. CNNs sind besonders für Daten geeignet welche Arrays als Eingabe entgegennehmen und Farbbilder sind eben dies. Die Besonderheit in diesem NN sind der Convolutional-Layer und der Pooling-Layer

2.4.1 Convolutional-Layer

Um zu verstehen was in diesen Layern passiert, soll die Funktionsweise von Filtern in der Bildbearbeitung betrachtet werden. Als Eingangsbild wird ein Grauwertbild I_{In} angenommen, auf welchem mathematische Operationen ausgeführt werden und das Ausgangsbild I_{Out} erzeugen. Bei der mathematischen Operation handelt es um eine diskrete Faltung im Ortsbereich bzw. digitale Filterung im Ortsbereich. Meist wird eine quadratische Maske (engl. Kernel) mit der Größe m genutzt. Jedem Pixel in der Filtermaske wird ein Gewicht h(s,t) zugeordnet. Um nun den neuen Wert für einen Pixel zu berechnet wird für jeden Pixel P(x,y) die Faltung wie folgt berechnet

$$P_{neu}(x,y) = \sum_{u=-a}^{a} \sum_{v=-b}^{b} P_{alt}(x+1-u, y+1-v) * h(u,v)$$

Im Randbereich des Bildes besitzen die Pixel jedoch weniger Nachbarn und die fehlenden Pixel können beispielsweise durch Spiegelung, dem arithmetischem Mittel oder

Function	Computation Equation
Sigmoid	$f(x) = \left(\frac{1}{(1+ exp^{-x})} \right)$
HardSigmoid	$f(x) = max\left(0, min\left(1, \frac{(x+1)}{2}\right)\right)$
SiLU	$a_k(s) = z_k \alpha(z_k)$
dSiLU	$a_k(s) = \alpha(z_k)(1 + z_k(1 - \alpha(z_k)))$
Tanh	$f(x) = \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right)$
Hardtanh	$f(x) = \begin{pmatrix} -1 & if\ x < -1 \\ x\ if & -1 = x \leqslant 1 \\ 1 & if\ x > 1 \end{pmatrix}$
Softmax	$f(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)}$
Softplus	$f(x) = \log(1 + exp^x)$
Softsign	$f(x) = \left(\frac{x}{\lvert x \rvert + 1}\right)$
ReLU	$f(x) = max(0, x) = \begin{cases} x_i, & if\ x_i \geq 0 \\ 0, & if\ x_i < 0 \end{cases}$
LReLU	$f(x) = \alpha x + x = \begin{cases} x & if\ x > 0 \\ \alpha x & if\ x \leq 0 \end{cases}$
PReLU	$f(x_i) = \begin{pmatrix} x_i\ , & if\ x_i > 0 \\ a_i x_i\ , & if\ x_i \leq 0 \end{pmatrix}$
RReLU	$f(x_i) = \begin{pmatrix} x_{ji}\ , & if\ x_{ji} \geq 0 \\ a_{ji} x_{ji}\ , & if\ x_{ji} < 0 \end{pmatrix}$
SReLU	$f(x) = \begin{pmatrix} t_i^r + a^r{}_i (x_i - t^r{}_i), & x_i \geq t^r{}_i \\ x_i\ , & t_i^r > x_i > t_i^l \\ t^l{}_i + a^l{}_i \left(x_i - t^l{}_i\right), & x_i \leq t^l{}_i \end{pmatrix}$
ELU	$f(x) = \begin{pmatrix} x, & if\ x > 0 \\ \alpha\ exp(x) - 1, & if\ x \leq 0 \end{pmatrix}$
PELU	$f(x) = \begin{pmatrix} cx, & if\ x > 0 \\ \alpha\ exp^{\frac{x}{b}} - 1, & if\ x \leq 0 \end{pmatrix}$
SELU	$f(x) = \tau \begin{pmatrix} x, & if\ x > 0 \\ \alpha\ exp(x) - \alpha, & if\ x \leq 0 \end{pmatrix}$
Maxout	$f(x) = max(w_1^T x + b_1\ , w^T{}_2 x + b_2)$
Swish	$f(x) = x \cdot sigmoid(x) = \frac{x}{1+e^{-x}}$
ELiSH	$f(x) = \begin{cases} \left(\frac{x}{1+e^{-x}}\right), & x \geq 0 \\ \left(\frac{e^x - 1}{1+e^{-x}}\right), & x < 0 \end{cases}$
HardELiSH	$f(x) = \begin{cases} x \times max\left(0, min\left(1, \left(\frac{x+1}{2}\right)\right)\right), & x \geq 0 \\ (e^x - 1) \times max\left(0, min\left(1, \left(\frac{x+1}{2}\right)\right)\right), & x < 0 \end{cases}$

Abbildung 2.5: Zusammenfassung von AFs [24]

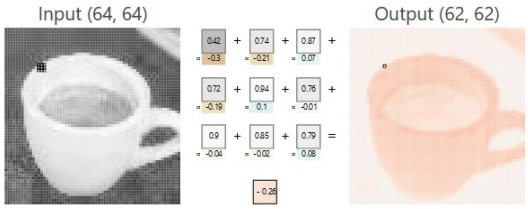
Zero-Padding aufgefüllt werden [22, Kaptiel 5].

Diese Berechnungen erfolgen auch in dem Convolutional(Conv)-Layer des NN. Die Gewichte in h werden im Training jedoch nach jedem Durchlauf aktualisiert. Soll die Höhe und Breite von Ausgabe und Eingabe gleich bleiben, wird der Kernel üblicherweise ungerade und quadratisch gewählt und zusätzlich wird ein Zero-Padding verwendet bei dem die Anzahl an angefügten Zeilen $Padding_x = Kernel_x - 1$ beträgt und Spalten $Padding_y = Kernel_y - 1$ beträgt. Die wahl eines solchen Kernels und Paddings ermöglicht eine leichte Zuordnung in dem der Ausgabewert an Position (x,y) durch die Faltung berechnet wurde, bei der das Zentrum des Kernels ebenfalls an Postion (x,y) ist. In den gegebenen Beispielen hat sich der Filter jeweils um einen Pixel in der Breite bzw. Höhe bewegt aber es handelt sich um einen Parameter der frei gewählt werden kann und Stride genannt wird.

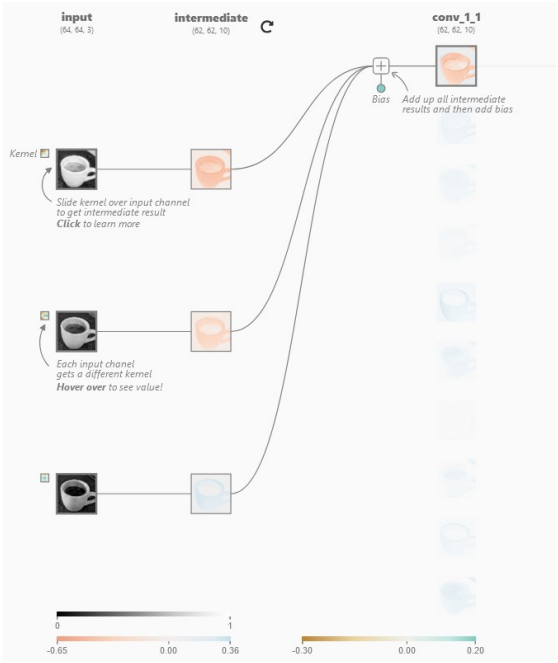
Wichtige Parameter die in diesem Layer vom Netzwerkarchitekten verändert werden können sind somit Kernelgröße, Padding und Stride. Um solche Paramter von anderen zu unterscheiden werden sie Hyperparameter genannt.

Abb. 2.5 veranschaulicht die Berechnung für einen Pixel eines Farbkanals. Werte innerhalb der Quadrates zeigen normalisierten Grauwert und Werte darunter das Gewicht des Kernels. Die Ausgabe eines Neurons in diesem Layer wird Feature-Map genannt. Die Eingabedaten sind für Bilder durch Höhe x Breite x Tiefe gegeben.

Bisher wurden Grauwertbilder angenommen, werden RGB-Bilder verwendet, erhält jeder Farbkanal einen Kernel mit dem die Faltung berechnet wird und anschließend werden die Ergbnisse mit einem Bias summiert (Abb. 2.6).



(a)



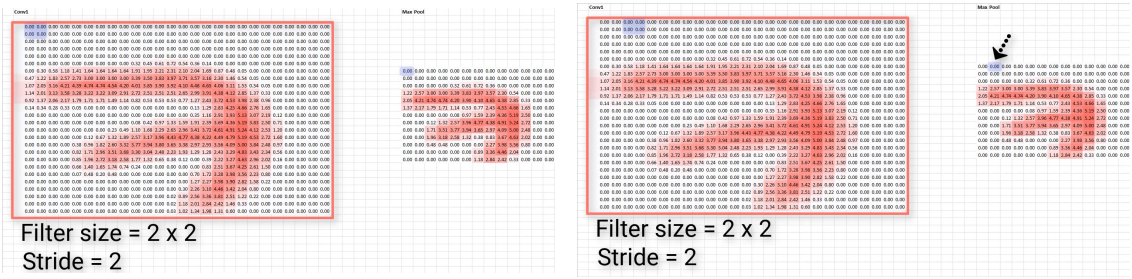
(b)

Abbildung 2.6: Beispiel der Berechnung in einem Conv-Layer für Grauwert- und RGB-Bild [32]

2.4.2 Pooling-Layer

Der Pooling-Layer, auch Subsampling-Layer genannt, reduziert die Höhe und Breite und führt zu einer schnelleren Berechnung. Es kann zwischen Average-Pooling, Min-Pooling und Max-Pooling gewählt werden.
Das Grundlegende vorgehen ist für alle Varianten identisch. Die Pixelwerte in einer Nachbarschaft um das Zielpixel werden betrachtet, um die gesamte Nachbarschaft durch einen einzigen Pixelwert zu ersetzen. Ein Beispiel für Max-Pooling ist in Abb. 2.7 zu sehen.

Eine besondere Art ist das Global-Pooling, dabei wird die Höhe und Breite auf 1 reduziert.
Die Hyperparameter in diesem Layer sind Kernel-Size, Stride und Padding.



(a) Max-Pooling für den Pixel (0,0) und dessen Nachbarschaft

(b) Max-Pooling nach einem Schritt für den Pixel (2,0) und dessen Nachbarschaft

Abbildung 2.7: Beispiel von Max-Pooling [10]

2.4.3 1x1-Conv-Layer

Der Pooling-Layer kann die Höhe und Breite der Feature-Maps reduzieren, doch die Tiefe würde immer weiter ansteigen und eine weitere Berechnung schnell unmöglich machen. Daher wird ein 1x1-Conv-Layer verwendet.
Die Ausgabe eines Layers $hxbxt$ wird an diesen Layer übergeben, die Kernelgröße ist $1x1xt$. Der Filter betrachtet daher zwar nur einen Pixel in der Höhe und Breite, erstreckt sich jedoch über die gesamte Tiefe. Die

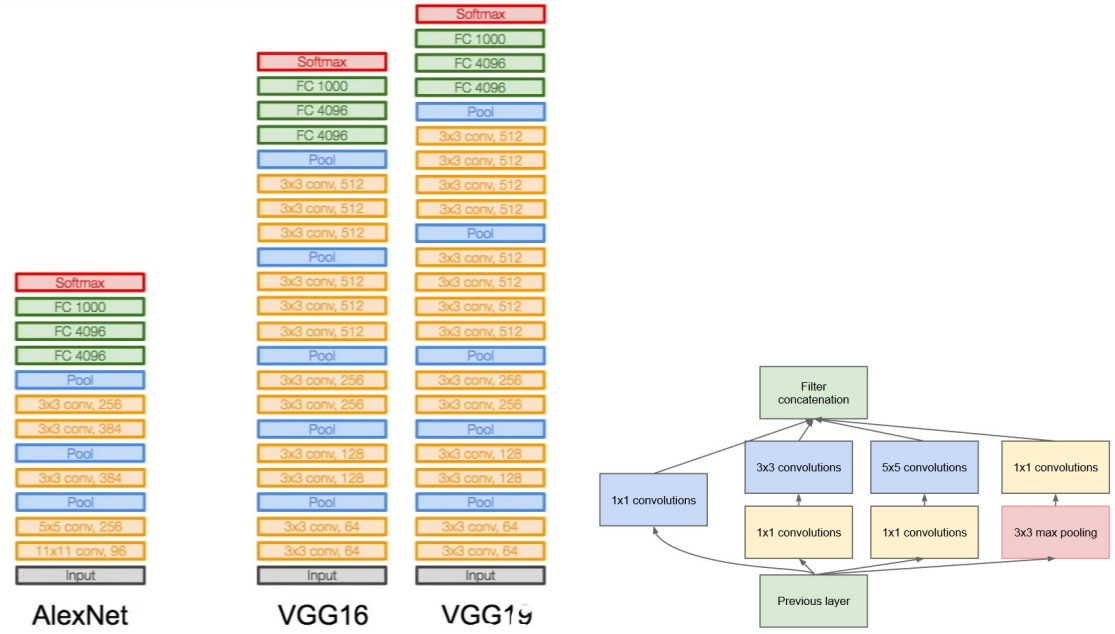
2.4.4 Besonders relevante CNN-Architekturen

Die Architektur von CNNs ändert sich stetig und einige Architekturen haben die Entwicklung besonders stark beeinflusst und sollen deshalb kurz erwähnt werden.

AlexNet war 2012 das erste CNN welche die ImageNet-Large-Scale-Visual-Recognition-Challenge (ILSVRC) gewann. Das NN besitzt 8 Layer und hat als AF die ReLU genutzt [14].

VGGNet hat 2014 hat auf dem Erfolg von AlexNet aufgebaut und noch mehr Layer hinzugefügt (Abb. 2.8a). Dabei wurde eine Kernelgröße von 3 verwendet, andere CNN haben noch oft eine Kernelgröße von 11, 7 oder 5 verwendet. Das receptive field [33, Kapitel 6.2.6.] von Drei 3x3-Conv-Layer ist das selbe wie in einem 7x7-Conv-Layer. Die Anzahl der Paramter ist für Drei 3x3-Conv-Layer 3 * (3^2T^2) = $27T^2$ und für einen 7x7-Conv-Layer dagegen 7^2T^2 = $49T^2$ [26].

GoogLeNet hat 2014 ebenfalls rel. viele 3x3-Conv-Layer verwendet, zusätzlich werden aber sogennante Inception-Modules verwendet, deshalb auch den Namen Inception-



(a) VGG Architektur im Vergleich zu AlexNet [1] (b) GoogLeNet Architektur [27]

Abbildung 2.8: VGG und GoogLeNet

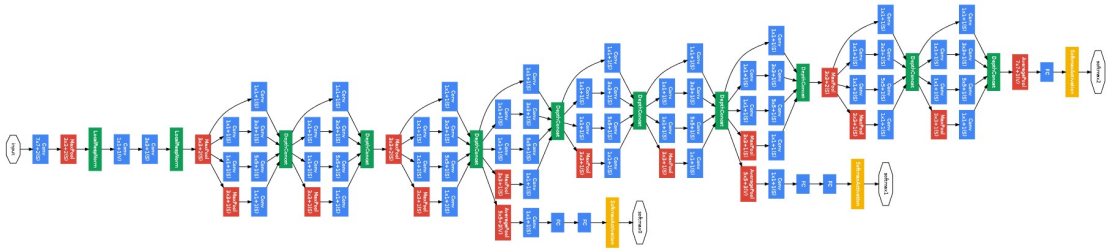


Abbildung 2.9: Inception Block in der GoogLeNet Architektur [27]

Architecture erhalten hat. Dabei werden 1x1-, 3x3 und 5x5-Convolutions sowie ein Max-Pooling ausgeführt (Abb. 2.8b). Die 3x3- und 5x5-Conv-Layer sind Rechenintesiv und werden deshalb auch als bottlenecks bezeichnet. Um die Anzahl der Berechnungen zu verringern werden zusätzlich 1x1-Conv-Layer davor gesetzt. Diese Struktur, in der die Dimension vor der rechenintensiven Berechnung verringert wird, wird Bottleneck-Layer genannt. Wurden alle Conv-Layer ausgeführt, werden sie am Ende zusammengeführt. Das Netzwerk besitzt 22-Layer weshalb der Gradient sehr klein wird und deshalb , während des Trainings, zusätzliche Hilfsklassifikatoren (engl. auxiliary classifiers) innerhalb der Hidden-Layer eingefügt werden. Außerdem helfen sie Overfitting zu vermeiden. [27].

Seit AlexNet hat sich die Forschung stark dahin beweget mehr Layer hinzuzufügen. MobileNets haben dagegen das Ziel die Effizienz von CNNs zu erhöhen. Die Basis bilden Depthwise-Separable-Convolutions, welche die standardmäßige Convolution in

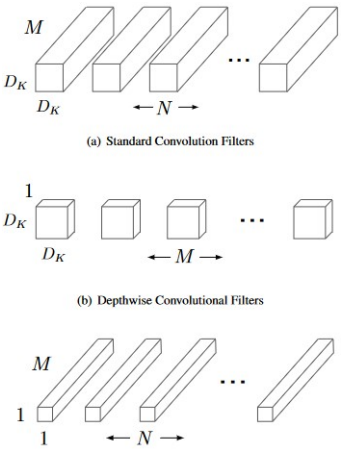


Abbildung 2.10: Standardmäßige Convolution im Vergleich zur Depthwise-Separable-Convolutions

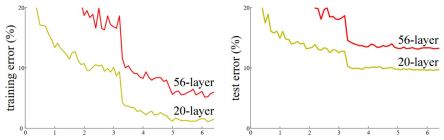


Abbildung 2.11: Traniing-Error für 20- und 56-Layer NN

Depthwise-Convolution und Pointwise-Convolution faktorisiert und auch im Inception-Block des GoogLeNet verwendet wurde [9]. Die standardmäßige Convolution benötigt

$$K_{Hoehe} * K_{Breite} * I_{Tiefe} * O_{Hoehe} * O_{Breite} * O_{Tiefe}$$

FLOPs, die Depthwise-Separable-Convolution dagegen nur

$$I_{Tiefe} * O_{Hoehe} * O_{Breite} * (K_{Hoehe} * K_{Breite} + O_{Tiefe})$$

FLOPs [33, Kapitel 3.4.2.].

ResNet hat 2015 die ILSVRC gewonnen. Es wurde angenommen, dass die Accuracy mit zunehmender Anzahl an Layer zunehmen würde, in der Praxis nahm dieser aber ab einem Punkt wieder ab. Es könnte angenommen werden dies lag an Overfitting, doch der Training-Error widerlegte diese Hypothese (Abb. refresNetError). [8] Die Lösung ist, anstatt die Funktion $\mathscr{H}(x)$ zu lernen, wird das Residuum $\mathscr{F}(x) := \mathscr{H}(x) - xx$ gelernt. Umformuliert ergibt sich die Gleichung

$$\mathscr{F}(x) + x$$

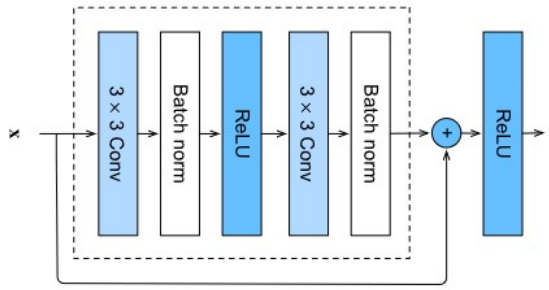


Abbildung 2.12: Der ResNet-Block

2.5 MediaPipe

2.5.1 Was ist MediaPipe

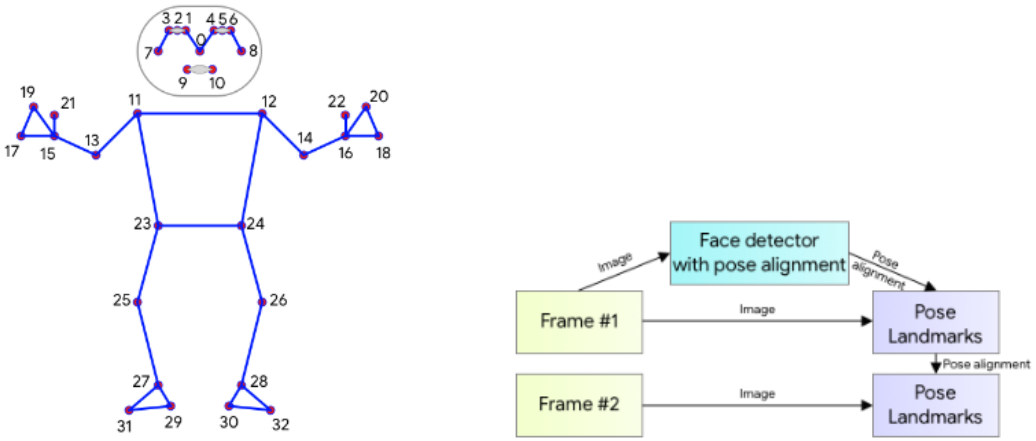
Um KI-basierte Methoden zu verwenden bietet sich MediaPipe als ein sehr gutes Framework an um eine Echtzeitanalyse durchzuführen. Desweiteren ist es OpenSource, liefert bereits trainierte NN und ermöglicht eine leichte Erweiterung durch die interne Verwendung von TensorFlow.

MediaPipe ist ein Framework das es ermöglicht eine Pipeline für die Inferenz von KI-Modellen zu erstellen.
Bei der Verwendung wird ein Graph erstellt, dessen Aufbau in der GraphConfig Datei beschrieben wird. Dort enthalten sind in dem Nodes bzw. Calculators platziert werden. Nodes können Eingangs- und Ausgangsströme besitzen. Um eigene Nodes zu schreiben muss die C++ API verwendet werden. Sollten sich die Anforderung im Laufe eines Projekts ändern, können Nodes sehr leicht ausgetauscht werden, zum Beispiel ein anderes NN ausprobiert werden oder zusätzliche Schritte in der Datentransformation hinzugefügt werden [16]. Es werden jedoch viele weitere Programmiersprachen unterstützt bei denen sich die bereitgestellten Lösungen unterscheiden.

2.5.2 MediaPipe's Pose-Estimation

Die bevorzugte Methode um eine Pose-Estimation durchzuführen beruht auf Heatmaps, dabei wird für jeden Keypoint eine Heatmap erzeugt. Der Vorteil bei diesem Vorgehen ist die Möglichkeit die Pose-Estimation für mehrere Personen in einem Bild durchzuführen was aber zu einem größeren Modell führt und es für Echtzeitanwendungen ungeeignet macht.

MediaPipe ist jedoch nicht darauf ausgelegt die Post-Estimation für mehrere Personen



(a) Blazepose Keypoints (b) BlazePose Inferenz

Abbildung 2.13: BlazePose

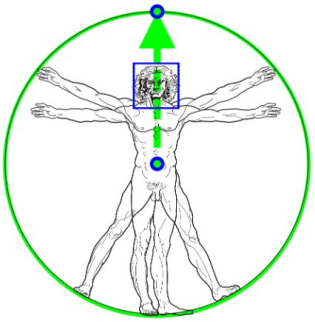


Abbildung 2.14: Punkte nach verarbeitung durch BlazePose [5]

in einem Bild/Video durchzuführen und verwendet daher ein performantes CNN. Bei der Inferenz erzeugt das Netzwerk 33-Keypoints für eine Person.
Um eine besonders schnelle Berechnung zu erreichen benutzt BlazePose einen Detector und einen Tracker. Eine Person wird zu Beginn in dem Bild mit dem Detector gesucht und der Tracker kann in den folgenden Bildern die Keypoints, Präsenz einer Person und die Region-of-Interest (ROI) vorhersagen. Kann der Tracker keine Person im Bild finden, wird der Detector erneut ausgeführt.
Um eine Person zu finden wird BlazeFace verwendet, weshalb die Voraussetzung ist, dass das Gesicht und die Hüfte der Person sichtbar sein muss oder zumindestens voraussgesagbar ist. Zusätzlich zur Bounding-Box des Gesichtes werden Mittelpunkt der Hüfte, ein Kreis der die Gesamte Person enthält und der Winkel zwischen Mittelpunkt der Hüfte und Schulter markiert(Abb. 2.14). [5] Das NN nutzt Heatmaps und offset-loss im Trainingsprozess um das Ergebnis zu verbessern und wird für die Inferenz entfernt. Um Low-Level-Feature und High-Level-Feature gleichmäßig zu benutzen werden Skip-Connections, wie aus ResNet bekannt, benutzt. Die Gradienten vom Regression-Encoder werden nicht an die Heatmap-Feature übertragen (Abb. 2.15).
Um das Netzwerk effizienter zu machen werden die Werte für den Winkel, Skalierung

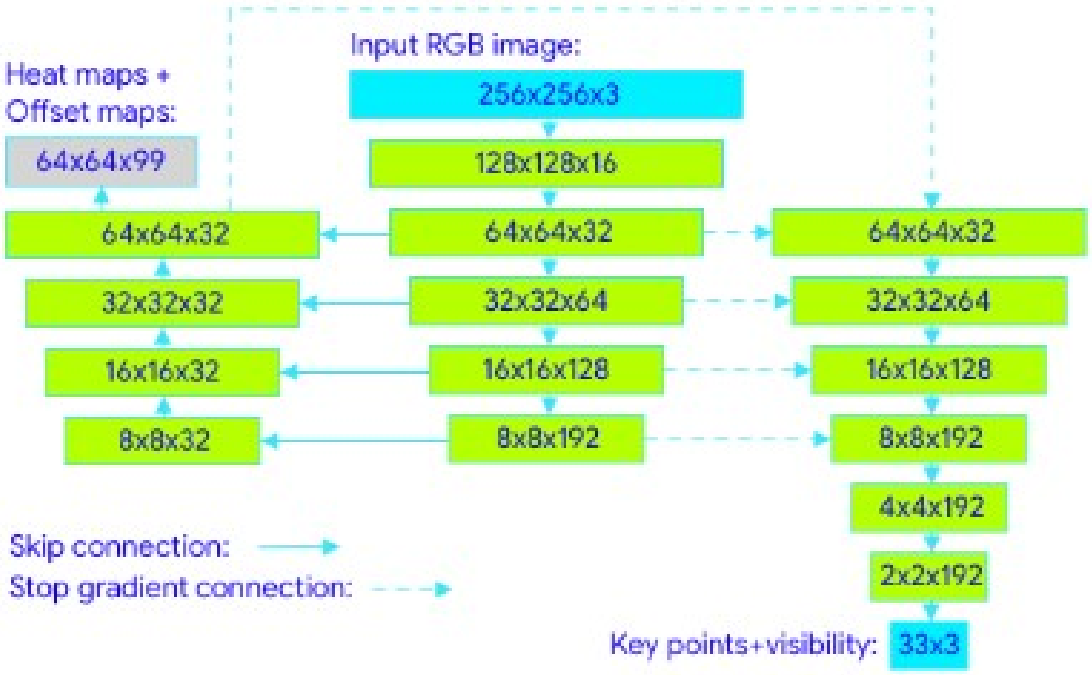


Abbildung 2.15: Architektur des NN von BlazePose [5]

und Translation für das Vorbereiten der Daten und Training auf ein Intervall eingegrenzt [5].

2.5.3 MediaPipe's Face-Mesh

Face-Mesh ist eine Lösung um dem Gesicht einer Person, in einem Video, in Echtzeit, 468 3D-Landmarken einzuzeichnen. Zusätzlich kann mit Hilfe der Landmarken ein 3D-Mesh erzeugt werden. Face-Mesh benutzt den BlazeFace-Detector für die Gesichtserkennung und wird auf der GPU ausgeführt.
BlazeFace ist ein Framework zur Objekterkennung und kann somit auch für andere Objekte verwendeten werden. Für die Gesichtserkennung werden 2 Modelle benutzt, da die Front- und Rückkamera eines Smartphones eine unterschiedliche Brennweite besitzen. Es werden 6 Landmarken für das Gesicht genutzt um eine Translations- und Rotationsinvarianz zu erreichen.

Die Architektur des Netzwerks basiert auf dem des MobileNetV2 für BlazeFace sollten Verbesserungen in Hinsicht des receptive-field, dem Feature-Extractor, Anchor-Anordnung und dem Postprocessing vorgenommen werden und folgende Änderungen hervorbrachte.

Es wurde ein sogenannten BlazeBlock eingeführt (Abb. BlazeBlock). Die Berechnung der 1x1-Conv benötigt im Vergleich zur Depthwise-Convolution 4.3x so lange. Daher werden in der Architektur 5x5 anstatt von 3x3 Kernel für die Depthwise-Convolution

Model	FPS	AR Dataset, PCK@0.2	Yoga Dataset, PCK@0.2
OpenPose (body only)	0.4[1]	**87.8**	83.4
BlazePose Full	10[2]	84.1	**84.5**
BlazePose Lite	**31**[2]	79.6	77.6

Abbildung 2.16: BlazePose und OpenPose im Vergleich
[1]Desktop CPU with 20 cores (Intel i9-7900X)
[2]Pixel 2 Single Core via XNNPACK backend[5]

verwendet und im Gegenzug wird die Anzahl der Bottleneck-Layer reduziert.

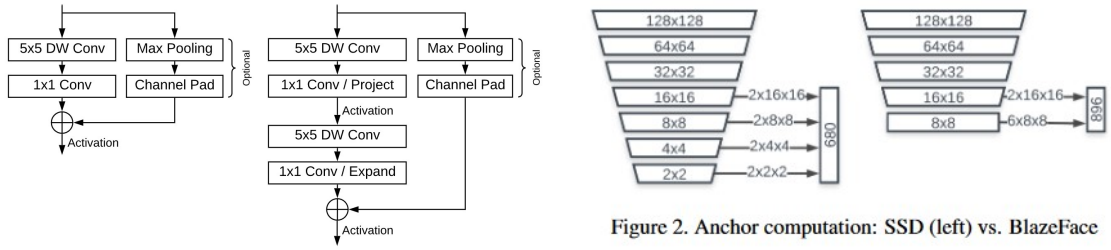


Figure 2. Anchor computation: SSD (left) vs. BlazeFace

(a) (b)

Abbildung 2.17: (a)BlazeBlock, (b) BlazeFace Anchor-Berechnungen [4]

In einem Single-Shot-Detector (SSD) wird ein Anchor für verschieden Auflösungen definiert. BlazeBlock definiert keine Anchor für 1x1 und 4x4 Feature-Maps da das Pooling-Pyramid-Network zeigt, dass Berechnungen ab einer bestimmt Auflösung redundant sind. Dafür werden in der 8x8 Feature-Map 6 Anchor genutzt (Abb. 2.18b).

Welcher Anchor gewählt wird, wird standardmäßige durch die Non-Maximum-Suppression bestimmt. In einem Video wechselt der bevorzugte Anchor aber sehr oft und führt zu einem unerwünschtem Ergbenis. Deshalb wird Non-Maximum-Suppression durch eine blending Strategie ersetz. Das Ergebnis erzielt eine 3x schnellere Inferenz und sogar eine leichte Verbesserung in der Precision [4]. In der Ausgabe durch BlazeFace besitzt jeweils eine Landmarke für das Augenzentrum, der Anchor wird mit seiner horizontalen Achse an der Verbindungslinie dieser Landmarken ausgerichtet und das Bild, für die Weiterverarbeitung durch das Mesh-Prediction NN, zugeschnitten. Das zugeschnittene Bild wird für das NN auf eine Größe von 256x256 oder 128x128 gebracht. Das NN hat eine ResNet-Architektur mit starkem Subsampling um sehr früh große Bereiche des Bildes zu untersuchen [12].

Device	MobileNetV2-SSD, ms	Ours, ms
Apple iPhone 7	4.2	**1.8**
Apple iPhone XS	2.1	**0.6**
Google Pixel 3	7.2	**3.4**
Huawei P20	21.3	**5.8**
Samsung Galaxy S9+ (SM-G965U1)	7.2	**3.7**

(a)

Model	Average Precision	Inference Time, ms (iPhone XS)
MobileNetV2-SSD	97.95%	2.1
Ours	**98.61%**	**0.6**

(b)

Abbildung 2.18: (a)BlazeFace Inferenzzeit, (b) BlazeFace Precision [4]

3 Methodische Vorgehensweise

Die HoloLens 1 unterstützt die Unity-Version 2019.4.25.f1 welche dem Mixed-Reality-Toolkit (MRTK) 2.6.1 verwendet wurde.
Es existiert ein Unity-Plugin welches MediaPipe integriert, dieses setzt aber die Unity-Version 2020.3.30f1 voraus [23] und konnte in den Versuchen mit der HoloLens 1 daher nicht verwendet werden.

3.1 Netzwerkverbindung

Um die korrekte Übertragung der Daten zwischen der HoloLens und der externen Maschine zu gewährleisten werden TCP-Sockets verwendet. Auf der externen Maschine wird ein TCP-Socket gestartet der eine Verbindung von jeder IP-Adresse auf dem Port 51512 zulässt.
Die HoloLens verbindet sich mit der externen Maschine und sendet einen 4-Byte langen Header um die Bildgröße anzukündigen und sendet im Anschluss das Bild.
Auf dem externen Gerät wird der Header gelesen, um zu bestimmen wie viele Bytes anschließend für das Bild gelesen werden müssen.

3.2 Bildaufnahme

Bilder können auf der HoloLens aufgenommen werden und über das Netzwerk an einen externe Maschine versendet werden. Um eine Maschine auszuwählen mit dem man sich verbinden möchte, wurde in Unity ein Script geschrieben. Die Eingabe einer IP wird durch ein Eingabefeld ermöglicht und anschließend kann mit der Tap-Geste ein Bild aufgenommen und versendet werden. Dieses Bild ist als BRGA32-Array verfügbar und mehrere MB groß. Um die Dateigröße zu verringern wurde der Alpha-Channel entfernt und mit JPEG komprimiert.

Als externe Maschine wurde ein Computer mit einem AMD Ryzen 7 5900x und einer NVidia GTX 1070 verwendet. Der Server wartet auf eine eingehende Verbindung von der HoloLens. Für Kommunikation wird ein Vier Byte langer Header versendet um die Anzahl der einzulesnden Bytes zu ermitteln und anschließend die eigentlichen Bytes für das Bild gelesen.

3.3 Verarbeitung der Bilder

Das übertragene Originalbild wird auf dem Computer gespeichert und jeweils eine Variante nach Verarbeitung mit MediaPipes- Pose-Estimation und Face-Mesh gespeichert und das Bild aus der Pose-Estimation an die HoloLens geschickt.

4 Fazit

Eine Anwendung von CNNs auf der HoloLens 1 war in dieser Arbeit nicht möglich, da die Hardware bereits 6 Jahre alt ist und nicht für KI ausgelegt war. Die aktuelle HoloLens 2 besitzt zwar eine HPU und einen AI Coprocessor, ermöglicht jedoch keine Inferenz von eigenen DNNs. Es konnte eine Anwendung gefunden werden welche WinML und das SqueezeNet für eine object-classifictaion nutzt [25]. Und eine weitere Anwendung nutzt die HoloLens 2 mit dem EfficientNetB0 um ebenfalls eine object-classification, hier aber mit 1000 Klassen, durchzuführen [6].

5 Ausblick

NVidia war in der Vergangenheit führend in der Hardwareherstellung für KI, durch das Wachstums in diesem Sektor finden sich immer mehr Unternehmen die gegen das Closed-Source Vorgehen von NVidia vorgehen. Grafikkarten sind durch ihre parallele Verarbeitung besonders für Neuronale-Netze geeignet aber der räumliche Abstand von Speicher und der Grafikkarte auf dem Mainboard stellen sich als große Schwachstelle in den Berechnungen heraus und neue Ansätze die nicht auf der Von-Neumann-Architekur aufbauen werden von vielen Unternehmen untersucht.

Es existieren unterschiedliche Ansätze für KI-spezialisierte Hardware wobei noch keine klare Aussage darüber gemacht werden kann welche Technologie letztendlich verwendet wird. Durch die Dominanz von DNNs spezialisieren sich die meisten darauf die Trainings- und Inferenzzeit von ihnen zu verringern. Einen Überblick gibt Abb. 5.1 [34]

SUMMARY OF STATE OF THE ART ML PROCESSORS

Architecture	Algorithms for Intelligence	Tasks/Applications	Power/Energy Efficiency
Boltzmann machine (RBM) processor [30]	NN	Energy-efficient restricted RBM processor(MNIST)	310mW,1.45TOPS/W
Neuromorphic [31]	TDNN(BNN/CNN)	Apply TDNN technique to BNN(MNIST)	48.2 TSOP/S/W
DNN accelerator [32]	DNN	DNN classifier(MNIST)	33.7mW, 0.36 J/inference
DNN accelerator [33]	DNN	Fully-variable weight bit-precision(Alexnet/VGG-16)	3.2mW@0.63V/297mW@1.1V, 345.6GOPS@16b weights
DCNN processor [34]	CNN	CNN for intelligent embedded system (AlexNet)	39mW, 676GOPS(CAs) /76GOPS(DSP)
FC-DNN accelerator [35]	FC-DNN	Accelerator (MNIST) with fault tolerance	0.56μW/Decision
CNN processor (Eyeriss) [36]	DNN/CNN	Accelerator (Imagenet with Row Stationary)	278mW, 7.94mJ/Decision
CNN FR processor [37]	CNN	Face detection and recognition	620μW
CNN-RNN processor [38]	CNN, RNN, general purpose DNN	General purpose DNN (quantization-table-based multiplier)	63mW
SoC with a CNN accelerator [39]	CNN	IoT Edge Mote, image and enviromental data processing	sub-mW
Mixed-signal binary CNN processor [40]	Binary CNN	Image classification (CIFAR-10), near memory computing	899μW
Hybrid-NN processor (Thinker) [41]	CNN/FCN	Hybrid-NN (AlexNet/LRCN)	4-447mW, 409.6GOPS
Neuromorphic [42]	Embedded Reinforcement Learning: CNN/DNN/SVM	Mobile self-driving micro-robot at the edge of the cloud	690μW
Computing-in-memory [43]	CNN	Real-time 3D hand-gesture recognition processor	6.57mW, 11.8GOPS
Computing-in-memory [44]	CNN-based Machine Learning	LeNet-5 CNN	28.1 TOPS/W
Computing-in-memory [45]	SVM	In-memory machine learning classifier (SVM)	0.042nJ/Decision
Computing-in-memory [46]	binary DNN	Binary-based CIM-SRAM macro(MNIST)	55.8TOPS/W
Computing-in-memory [47]	Versatile DNN	Binary/Ternary recongurable in-memory DNN Accelerator	0.6W, 1.4TOPS
VAD system [48]	DNN	Voice activity detector (AFE and a digital BNN classifier)	1μW(voice detector)/ 0.38μW(feature extraction)
AI SoC [49]	DNN	3D-Stacked log-quantization for DNN inference (CNN/MLP/RNN)	3.3W, 1.96TOPS@4b weights
ConvNet processor [50]	CNN	Face recognition (VGG-16/AlexNet)	7.5-300mW, 10TOPS/W
DLA(deep learning accelerator) [51]	DNN	Keyword spotting and face detection	288μW

Abbildung 5.1: Überblick über KI-Prozessoren [34]

Literatur

[1] Alexander Amini und Ava Soleimany. Lecture 9: CNN Architectures. URL: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf (besucht am 28. 02. 2022).

[2] Alexander Amini und Ava Soleimany. MIT 6.S191: Introduction to Deep Learning. URL: http://introtodeeplearning.com/2021/index.html (besucht am 28. 02. 2022).

[3] Michael Anderson u. a. First-Generation Inference Accelerator Deployment at Facebook. 2021. arXiv: 2107.04140 [cs.AR].

[4] Valentin Bazarevsky u. a. BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs. 2019. arXiv: 1907.05047 [cs.CV].

[5] Valentin Bazarevsky u. a. BlazePose: On-device Real-time Body Pose tracking. 2020. arXiv: 2006.10204 [cs.CV].

[6] Mitchell Doughty. HoloLens-2-Machine-Learning. URL: https://github.com/doughtmw/HoloLens2-Machine-Learning (besucht am 10. 03. 2022).

[7] Epic. Unreal Engine 4.25 released! URL: https://www.unrealengine.com/en-US/blog/unreal-engine-4-25-released (besucht am 02. 02. 2022).

[8] Kaiming He u. a. "Deep Residual Learning for Image Recognition". In: CoRR abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[9] Andrew G. Howard u. a. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: CoRR abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[10] Jeremy Howard. Lesson 4: Practical Deep Learning for Coders. URL: https://www.youtube.com/watch?v=V2h3IOBDvrA (besucht am 28. 02. 2022).

[11] S. Hyakin. "Neural Networks: A Comprehensive Foundation". In: 1994.

[12] Yury Kartynnik u. a. Real-time Facial Surface Geometry from Monocular Video on Mobile GPUs. 2019. arXiv: 1907.06724 [cs.CV].

[13] Khronos. Multiple Conformant OpenXR Implementations Ship Bringing to Life the Dream of Porta URL: https://www.khronos.org/news/press/multiple-conformant-openxr-implementations-ship-bringing-to-life-the-dream-of-portable-xr-applications (besucht am 02. 02. 2022).

[14] Alex Krizhevsky, Ilya Sutskever und Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: Communications of the ACM 60 (2012), S. 84–90.

[15] Yann LeCun, Y. Bengio und Geoffrey Hinton. “Deep Learning”. In: Nature 521 (Mai 2015), S. 436–44. DOI: 10.1038/nature14539.

[16] Google LLC. Framework Concepts. URL: https://google.github.io/mediapipe/framework_concepts/framework_concepts.html (besucht am 28. 02. 2022).

[17] Microsoft. Open source projects and samples from Microsoft. URL: https://github.com/microsoft (besucht am 20. 02. 2022).

[18] Microsoft. HoloLens-Hardware. URL: https://docs.microsoft.com/de-de/hololens/hololens1-hardware (besucht am 27. 02. 2022).

[19] Microsoft. HoloLens2-Hardware. URL: https://docs.microsoft.com/en-us/hololens/hololens2-hardware (besucht am 27. 02. 2022).

[20] Microsoft. Introduction to Mixed Reality development. URL: https://docs.microsoft.com/en-us/windows/mixed-reality/develop/development?tabs=unity (besucht am 27. 02. 2022).

[21] Microsoft. What is the Mixed Reality Toolkit. URL: https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/?view=mrtkunity-2021-05 (besucht am 27. 02. 2022).

[22] Alfred Nischwitz u. a. “Bildverarbeitung”. In: Band II des Standardwerks Computergrafik und Bildv Springer Vieweg, 2020. ISBN: 978-3-658-28704-7. DOI: 10.1007/978-3-658-28705-4. URL: https://doi.org/10.1007/978-3-658-28705-4.

[23] Junrou Nishida. MediaPipe Unity Plugin. URL: https://github.com/homuler/MediapipeUnityPlugin (besucht am 14. 02. 2022).

[24] Chigozie Nwankpa u. a. “Activation Functions: Comparison of trends in Practice and Research for Deep Learning”. In: CoRR abs/1811.03378 (2018). arXiv: 1811.03378. URL: http://arxiv.org/abs/1811.03378.

[25] Rene Schulte. Deep Learning inference running on the HoloLens. URL: https://github.com/reneschulte/WinMLExperiments (besucht am 10. 03. 2022).

[26] Karen Simonyan und Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: CoRR abs/1409.1556 (2015).

[27] Christian Szegedy u. a. “Going deeper with convolutions”. In: 2015 IEEE Conference on Compute 2015, S. 1–9. DOI: 10.1109/CVPR.2015.7298594.

[28] Tom Taulli. “AI Foundations”. In: Artificial Intelligence Basics: A Non-Technical Introduction. Berkeley, CA: Apress, 2019, S. 1–17. ISBN: 978-1-4842-5028-0. DOI: 10.1007/978-1-4842-5028-0_1. URL: https://doi.org/10.1007/978-1-4842-5028-0_1.

[29] Elene Terry. “Silicon at the Heart of HoloLens 2”. In: 2019 IEEE Hot Chips 31 Symposium (HCS). 2019, S. 1–26. DOI: 10.1109/HOTCHIPS.2019.8875669.

[30] Columbia University. Artificial Intelligence (AI) vs. Machine Learning. URL: https://ai.engineering.columbia.edu/ai-vs-machine-learning/ (besucht am 25. 02. 2022).

[31] Stanford University. CS231n Convolutional Neural Networks for Visual Recognition. Optimization URL: https://cs231n.github.io/optimization-2/ (besucht am 28. 02. 2022).

[32] Jay Wang u. a. CNN Explainer. Learn Convolutional Neural Network (CNN) in your browser! URL: https://poloclub.github.io/cnn-explainer/#article-pooling (besucht am 25. 02. 2022).

[33] Aston Zhang u. a. “Dive into Deep Learning”. In: arXiv preprint arXiv:2106.11342 (2021).

[34] Zhuo Zou u. a. “Edge and Fog Computing Enabled AI for IoT-An Overview”. In: 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS). 2019, S. 51–56. DOI: 10.1109/AICAS.2019.8771621.

Anhang A: Pyhton-Code

MediaPipe Face-Mesh

```
import cv2
import mediapipe as mp
import os

def drawFaceMesh(filename):
    mp_drawing = mp.solutions.drawing_utils
    mp_drawing_styles = mp.solutions.drawing_styles
    mp_face_mesh = mp.solutions.face_mesh

    drawing_spec = mp_drawing.DrawingSpec(thickness=1,
        ↪ circle_radius=1)
    with mp_face_mesh.FaceMesh(
        static_image_mode=True,
        max_num_faces=1,
        refine_landmarks=True,
        min_detection_confidence=0.6) as face_mesh:
            inputImage = os.getcwd() + "\\" + filename
            image = cv2.imread(inputImage)
            # Convert the BGR image to RGB before processing.
            results = face_mesh.process(cv2.cvtColor(image, cv2.
                ↪ COLOR_BGR2RGB))
            print(results==True)
            # Print and draw face mesh landmarks on the image.
            if not results.multi_face_landmarks:
                return 1
            annotated_image = image.copy()
            for face_landmarks in results.multi_face_landmarks:
                print('face_landmarks:', face_landmarks)
                mp_drawing.draw_landmarks(
                    image=annotated_image,
                    landmark_list=face_landmarks,
                    connections=mp_face_mesh.FACEMESH_TESSELATION,
                    landmark_drawing_spec=None,
                    connection_drawing_spec=mp_drawing_styles
                    .get_default_face_mesh_tesselation_style())
                mp_drawing.draw_landmarks(
                    image=annotated_image,
                    landmark_list=face_landmarks,
```

```
                    connections=mp_face_mesh.FACEMESH_CONTOURS,
                    landmark_drawing_spec=None,
                    connection_drawing_spec=mp_drawing_styles
                    .get_default_face_mesh_contours_style())
                mp_drawing.draw_landmarks(
                    image=annotated_image,
                    landmark_list=face_landmarks,
                    connections=mp_face_mesh.FACEMESH_IRISES,
                    landmark_drawing_spec=None,
                    connection_drawing_spec=mp_drawing_styles
                    .get_default_face_mesh_iris_connections_style()
                        ↪ )

            savePath = 'Images_Annotated\\' + 'faceMesh_' + os.
                ↪ path.basename(filename)
            isWritten = cv2.imwrite(savePath, annotated_image)
            if isWritten:
                print("File written successfully: " + savePath )
            else:
                print("File could not be written")

            return savePath
```

MediaPipe Pose-Estimation

```
import cv2
import mediapipe as mp
import numpy as np
import os
import sys

mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_pose = mp.solutions.pose

def drawKeypoints(filename):

    BG_COLOR = (192, 192, 192) # gray
    with mp_pose.Pose(
        static_image_mode=True,
        model_complexity=2,
        enable_segmentation=True,
        min_detection_confidence=0.5) as pose:
```

```
        inputImage = os.getcwd() + "\\" + filename
        image = cv2.imread(inputImage)
        image_height, image_width, _ = image.shape
        # Convert the BGR image to RGB before processing.
        results = pose.process(cv2.cvtColor(image, cv2.
            ↪ COLOR_BGR2RGB))
        if not results.pose_landmarks:
            return 1

        annotated_image = image.copy()
        # Draw segmentation on the image.
        # To improve segmentation around boundaries, consider
            ↪ applying a joint
        # bilateral filter to "results.segmentation_mask" with "
            ↪ image".
        condition = np.stack((results.segmentation_mask,) * 3,
            ↪ axis=-1) > 0.1
        bg_image = np.zeros(image.shape, dtype=np.uint8)
        bg_image[:] = BG_COLOR
        annotated_image = np.where(condition, annotated_image,
            ↪ bg_image)
        # Draw pose landmarks on the image.
        mp_drawing.draw_landmarks(
            annotated_image,
            results.pose_landmarks,
            mp_pose.POSE_CONNECTIONS,
            landmark_drawing_spec=mp_drawing_styles.
                ↪ get_default_pose_landmarks_style())
        savePath = 'Images_Annotated\\' + 'annotated_' + os.path.
            ↪ basename(filename)
        isWritten = cv2.imwrite(savePath, annotated_image)
        if isWritten:
            print("File written successfully: " + savePath )
        else:
            print("File could not be written")

        return savePath
```

Server

```
from ctypes import sizeof
import socket
from time import sleep
```

```
import numpy
import sys
import mediapipe as mp
import mpPose
import mpFace
import os


tcp_port = 51512
tcp_ip = '0.0.0.0'
buf_size = 1024



while True:
    # define header
    header = numpy.empty(4,dtype=bytes)
    bytesLeft = 4
    #setup socket
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((tcp_ip,tcp_port))
    s.listen(1)
    c,addr = s.accept()
    #read header
    bytesRead = c.recv_into(header, bytesLeft)
    bytesLeft -= bytesRead

    #convert from bytes to integer
    bytesLeft = numpy.frombuffer(header, dtype=int)[0]
    print(bytesLeft)

    data = bytearray()
    i = 0
    fileName = "Image"
    fileEnding = ".jpg"
    fileWritten = False
    gfullFileName = ""
    c.settimeout(1.5)
    while(fileWritten == False):
        fullFileName = "Images\\"+ fileName + str(i) + fileEnding
        try:
            with open (fullFileName, "xb") as f:
                while len(data) != bytesLeft:
```

```
                readData = c.recv(1024)
                data.extend(readData)
            print(hex(data[-2]))
            print(hex(data[-1]))

            f.write(data)
            fileWritten = True


    except Exception as e:
        print(e)
        i += 1
    gfullFileName = fullFileName


writtenPosePath = mpPose.drawKeypoints(gfullFileName)
if writtenPosePath == 1:
    print("No pose found")
    continue
filesize = os.path.getsize(writtenPosePath)

writtenFacePath = mpFace

header = filesize
print(f"file size is {filesize}")
print(f"and the header is {header}")
print(type(header))
print(f"header as bytes {header.to_bytes(4, 'little')}")
print(type(header.to_bytes(4, 'little')))

c.send(header.to_bytes(4, 'little'))
print(writtenPosePath)
with open(writtenPosePath, "rb") as file:
    c.sendfile(file)
    print(file.tell())


c.close()
s.close()
```

Anhang B: Tabellen

Layer/block	Input size	Conv. kernel sizes
Convolution	128×128×3	5×5×3×24 (stride 2)
Single BlazeBlock	64×64×24	5×5×24×1 1×1×24×24
Single BlazeBlock	64×64×24	5×5×24×1 1×1×24×24
Single BlazeBlock	64×64×24	5×5×24×1 (stride 2) 1×1×24×48
Single BlazeBlock	32×32×48	5×5×48×1 1×1×48×48
Single BlazeBlock	32×32×48	5×5×48×1 1×1×48×48
Double BlazeBlock	32×32×48	5×5×48×1 (stride 2) 1×1×48×24 5×5×24×1 1×1×24×96
Double BlazeBlock	16×16×96	5×5×96×1 1×1×96×24 5×5×24×1 1×1×24×96
Double BlazeBlock	16×16×96	5×5×96×1 1×1×96×24 5×5×24×1 1×1×24×96
Double BlazeBlock	16×16×96	5×5×96×1 (stride 2) 1×1×96×24 5×5×24×1 1×1×24×96
Double BlazeBlock	8×8×96	5×5×96×1 1×1×96×24 5×5×24×1 1×1×24×96
Double BlazeBlock	8×8×96	5×5×96×1 1×1×96×24 5×5×24×1 1×1×24×96

Tabelle B.1: feature-extraction network architecture [12]

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, 2022