Entwicklung zweier Setups für eine Augmented Reality Anwendung zum Erkennen von Obst- und Gemüsesorten mit neuronalen Netzen

Daniel Eggert

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG, Berliner Tor 7, 20099 Hamburg

Zusammenfassung. Diese Arbeit behandelt zwei Möglichkeiten zur Realisierung einer AR-Anwendung für das Erkennen von verschiedenen Obst und Gemüsesorten. Dafür werden die Frameworks Unity und Xcode im Umgang mit neuronalen Netzen getestet. Ziel dieser Arbeit ist es, ein grundlegendes Setup zu implementieren, anhand dessen eine AR-Anwendung mit komplexeren Funktionen entwickelt werden kann.

1 Einleitung

Nicht alle Menschen kennen beim Einkauf in einem Supermarkt oder auf Wochenmärkten alle der dort angebotenen Obst und Gemüsesorten. Auf Grund dessen müssen die Verkäufer und Mitarbeiter nach Informationen zu diesen gefragt werden, oder es wird im Internet nach Informationen gesucht. Schlechte Internetverbindung oder nicht ausgeschilderte Ware können die Informationsgewinnung über das Internet erschweren bis unmöglich machen. Auch haben Händler nicht immer die Zeit, z.B. bei großem Andrang, schnell weitere Informationen zu den angebotenen Sorten zur Verfügung zu stellen. Augmented Reality Anwendungen können die Informationsgewinnung erleichtern, da sie virtuelle Objekte und Informationen zu Produkten in Echtzeit, z.B. auf einem Smartphone, anzeigen können. Aus diesem Grund wird in dieser Arbeit beschrieben, wie sich eine Anwendung für das Lösen dieses Problems realisieren lassen kann. Die Anwendung, welche anhand des in dieser Arbeit implementierten Setups weiterentwickelt werden kann, soll sich dafür eignen, Nutzern schnell gewünschte Informationen zu einer bekannten oder unbekannten Obstsorte anzeigen zu können. Dafür werden zunächst im Grundlagenkapitel Begriffe und Bestandteile sowie verwendete Frameworks der Anwendung erläutert. Im Anschluss wird begründet wieso die bekannte Anwendung GoogleLens[10] nicht für die Lösung dieses Problems ausreicht. Im nachfolgendem Kapitel werden verwandte Arbeiten zu dem Thema der Obst- und Gemüseerkennung, der Einbindung eines neuronalen Netzes in Unity und Xcode vorgestellt. Danach werden die Anforderungen an die grundlegende Architektur näher spezifiziert und anhand dieser ein Konzept für die Umsetzung entwickelt. Anschließend erfolgt eine Beschreibung der Umsetzung und eine Begründung der Entscheidungen. Abschließend werden die wichtigen Erkenntnisse in einem Fazit zusammengefasst und es wird ein Ausblick für eine mögliche weitere Entwicklung des Projektes beschrieben.

2 Grundlagen

2.1 Augmented Reality

Augmented Reality(AR) ist eine Kombination der realen Welt mit virtuellen Informationen, welche mit Hilfe von AR-Displays angezeigt werden können. Zwei bekannte Beispiele für AR-Displays sind Head-Mounted-Displays und mobile Endgeräte. AR-Anwendungen haben viele Einsatzgebiete. Dies sind z.B. Einsatzbereiche in der Bildung, Medizin, Unterhaltung und Navigation. Die Entwicklung von AR-Anwendungen bringt einige Herausforderungen mit sich. Die Herausforderung für die vorliegende Arbeit ist das Tracking, also die Erkennung von realen Objekten sowie deren Position und Ausrichtung. Das Tracking kann dabei in ein sensorbasiertes, optisches, akustisches Tracking eingeteilt werden. Im Falle vom optischen Tracking erfolgt dieses mit einer oder mehreren Kameras.[14] Damit die Eigenschaften einer AR-Anwendung erfüllt sind, muss die Anwendung, nach der Definition von Azuma, interaktiv sein, virtuelle und reale Objekte beinhalten und in Echtzeit ablaufen.[3]

2.2 Unity

Unity ist ein Framework für das Entwickeln in einer 3-dimensionalen Echtzeitumgebung, welche sich neben der Entwicklung für Spiele, Simulationen für das Training von Deep-Reinforcement Algorithmen und weiteren Anwendungsfällen auch für die Entwicklung von AR- und VR-Anwendungen eignet. Die Intention hinter Unity ist es, die Entwicklung von Anwendungen für unterschiedliche Plattformen zu ermöglichen. Unity bietet unter anderem eine Rendering und Physik Engine, einen UI-Editor, sowie die Möglichkeit verschiedene von Unity bereitgestellte Komponenten, wie z.B. eine Kamera, in die Anwendung einzubauen. Des Weiteren können auch externe Plugins in Unity eingebaut werden. Die Programmiersprache der Unity Skripts ist C-Sharp. Ein Unityprojekt besteht aus unterschiedlichen Assets. Ein Beispiels für Assets wären Skripts, Szenen und 3D Objekte.[11]

AR-Foundation AR-Foundation ist eine Bibliothek von Unity, welche ermöglichen soll, eine plattformunabhängige AR-Anwendung in Unity zu entwickeln. Für jede Plattform die unterstützt werden soll muss ein zusätzliches Plugin installiert werden. Im Falle von iOS wäre es das ARKitXR-Plugin. Nicht jede von AR-Foundation bereitgestellte Funktion wird von jeder Plattform unterstützt, lediglich iOS unterstützt alle Funktionen. Für eine plattformunabhängige Entwicklung dürfen folglich nur die Funktionen genutzt werden, die auch von allen Plattformen unterstützt werden. Image-Tracking wird beispielsweise nicht von der Hololens unterstützt, jedoch wird für jede Plattform eine Funktion bereitgestellt, die Position und Rotation des Zielgeräts im physischen Raum festzustellen.[20]

Barracuda Barracuda ist eine Bibliothek für die Einbindung von neuronalen Netzen in Unity. Diese Bibliothek eignet sich für die plattformübergreifende Entwicklung, da alle Plattformen unterstützt werden, welche auch von Unity selbst unterstützt werden. Die Einbindung von selbst erstellten neuronalen Netzen erfordert die Umwandlung in ein ONNX Model, wofür ein Skript bereitgestellt wird. Es unterstützt jedoch nur sieben Modelarchitekturen. Hierbei handelt es sich um alle mit MLAgents erstellten Modelle, MobileNet v1 und v2, TinyYOLO, U-Net-Modelle, Fully connected Modelle, Fully Dense Modlle und Spade.[21] In einem Medium Artikel von Deren Lei wird jedoch erwähnt, dass die Unterstützung für TinyYOLOv3 ebenfalls in Unity integriert ist.[17]

2.3 ARKit

ARKit ist ein Framework von Apple für das Erstellen von Augmented Reality Anwendungen mit Xcode. Dafür stellt es viele Funktionen von AR dem Entwickler zur Verfügung, wie z.B. das Tracken von Bewegungen.[1]

2.4 Create ML

Create ML ist ein Entwicklungstool von Apple, um Modelle für Apple Anwendungen zu erstellen. Dafür liegen Templates mehrere Netztypen wie z.B. Objekterkennung und Texterkennung vor. Die mit Create ML erstellten Modelle können direkt auf einem Apple Gerät getestet werden, ohne diese zunächst in eine Anwendung integrieren zu müssen.[2]

2.5 Maschinelles Lernen und neuronale Netze

Maschinelles Lernen ist ein Teilgebiet der künstlichen Intelligenz. Dabei wird vom Computer anhand von Daten eine Grundlage für die Entscheidungsfindung bei spezifischen Aufgaben erlernt. Die Art des Trainings kann in drei verschiedene Kategorien eingeteilt werden, dem überwachten Lernen, dem unüberwachten Lernen und dem verstärkten Lernen. Es existieren auch Hybridformen zwischen dem überwachten und unüberwachten Lernen. In dieser Arbeit wird die Kategorie des überwachten Lernens genutzt. Das überwachte Lernen erfolgt durch annotierte Daten. Maschinelles Lernen kann für verschiedene Problemstellungen verwendet werden, so eignet es sich beispielsweise für die Bild- und Sprachverarbeitung.[5]

Neuronale Netze Ein Neuronales Netz ist der Funktionsweise des menschlichen Gehirns nachempfunden. Dabei sind verschiedene Neuronen über mehrere Schichten (engl. Layer) miteinander verknüpft. Anhand einer Eingabe, welche z.B. aus einem Bild bestehen kann, liefert das Netz eine Ausgabe, beispielsweise was auf diesem Bild erkannt wurde.[5]

Convolutional Neural Networks CNN sind für das Verarbeiten von Bildern geeignete neuronale Netze. Sie nutzen Filter um die Merkmale von Bildern aus einem Bild zu sondieren. Die Funktionsweise besteht darin, eine eingegebene Feature Map zu falten und das Ergebnis als neue Feature-Map weiterzugeben. In der Regel besteht ein CNN aus Convolutional Layern, Pooling Layern und einem Fully Connected Layer. Der Convolutional Layer extrahiert die Merkmale aus dem Bild. Die Aufgabe des Pooling Layers besteht darin, die Merkmalspunkte zu verkleinern, damit weniger Eingabeparameter benötigt werden. Zusätzlich extrahiert er die Rotation und Translation der Merkmale der übergebenen Feature-Map. In einem Fully-Connected Layer werden alle Neuronen mit den vorherigen Layern verknüpft um damit die Ausgaben zu kategorisieren.[7] CNN's eignen sich so z.B. für Klassifizierungs- und Segmentierungsaufgaben und übernehmen damit vorwiegend die Aufgaben des maschinellen Sehens.[13]

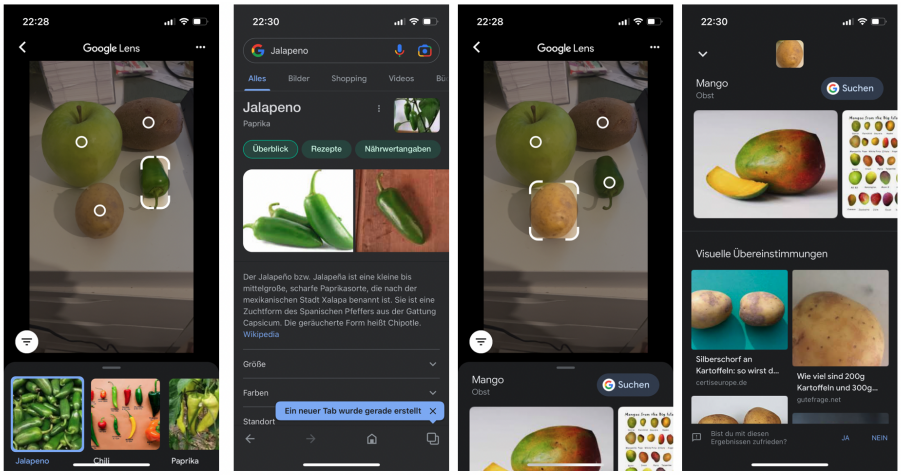
2.6 Transferlernen

Bei einer großen Anzahl an unterschiedlichen Klassifizierungsmöglichkeiten ist eine hohe Anzahl an Trainingsdaten erforderlich. Die Beschaffung der Trainingsdaten ist häufig zeitaufwendig, kostenintensiv oder unrealistisch. Transferlernen ist eine Möglichkeit, mit dieser Problematik umzugehen. Das Prinzip des Transferlernens ist der menschlichen Psychologie bei Lernprozessen nachempfunden. Wenn eine Person eine Tätigkeit beherrscht, die einer neu zu erlernenden Tätigkeit sehr ähnlich ist, so lässt sich diese in der Regel leichter erlernen, da sich das Wissen der bereits gelernten auf die neue übertragen lässt. Transferlernen nutzt ein bereits auf eine spezifische Domäne trainiertes Netz, um dieses in einer neuen, aber ähnlichen Domäne zu verwenden. Dieses muss unter Umständen weiter angepasst werden, aber es werden weniger Trainingsdaten benötigt, da die bereits zuverlässig erkannten Klassen nichtmehr trainiert werden müssen. Um durch Transferlernen ein gutes Ergebnis zu erzielen darf die ursprüngliche Domäne keine zu großen Abweichungen von der neuen Zieldomäne haben. Aber auch wenn sich die ursprüngliche und Zieldomäne sehr ähneln, können Probleme auftreten, falls diese fehlinterpretiert werden. Dies wäre z.B. bei der Klassifikation von Emotionen der Fall, da Emotionen von verschiedenen Personen unterschiedlich ausgedrückt werden können. Liegen nur sehr leichte Unterschiede zwischen den Domänen vor, spricht man bei der Anpassung von homogenem Transferlernen. Unterscheiden sie sich stärker wie z.B. bei den Fall von Emotionen, muss der Merkmalsraum gegebenenfalls angepasst werden und man spricht von heterogenem Transferlernen.[24]

3 Abgrenzung zu GoogleLens

GoogleLens[10] ist ein bekannte Möglichkeit, um nach unbekannten Objekten im Internet zu suchen. GoogleLens bietet auch noch weitere Funktionen wie z.B. das Erkennen von Texten.

Bilyk et al. haben GoogleLens bezüglich der Erkennung von Pflanzen mit anderen Pflanzenerkennungsapps verglichen. Dabei stand die Nutzung für die Bildung im Vordergrund. Sie beschrieben, dass spezifizierte Pflanzenerkennungsapps möglicherweise aufgrund ihres auf Pflanzen angepassten Trainingsdatensatzes eine höhere Erkennung aufweisen könnten, da die Möglichkeit für falsche Erkennungen dadurch gesenkt werden kann. Die Autoren gingen aber davon aus, dass GoogleLens aufgrund seines großen Datensatzes den anderen Pflanzenerkennung Apps überlegen ist. In ihrer Arbeit haben sie auch festgestellt, dass GoogleLens mit einer Genauigkeit von 92,6% die höchste Genauigkeit der von ihnen getesteten Pflanzenapps aufweist und auch am intuitivsten war.[4]



1.1 Jalapeno erkannt 1.2 Jalapeno Suche 1.3 Flasche Erkennung 1.4 Richtige Übereinstimmungen

Abb. 1. Ergebnisse einer Anwendung von GoogleLens, mit einem Apfel(Granny Smith), einer Kiwi, einer Jalapeno und einer Kartoffel

Um Objekte in einem Bild zu erkennen, kann entweder ein gespeichertes Bild übergeben, oder in Echtzeit ein Foto aufgenommen werden. In Abbildung 1.1 ist das Ergebnis nach einer Live Aufnahme zu sehen. Die Kreise zeigen dabei an, dass ein Ergebnis zu diesem Objekt vorliegt. Hier wurde die Jalapeno ausgewählt und es ist zu sehen, dass das Objekt richtigerweise auch als Jalapeno identifiziert wurde. Zusätzlich werden im unteren Abschnitt die Bilder aufgezeigt, deren Objekte Ähnlichkeiten zu dem identifizierten Objekt aufweisen. Nun kann mit einem Klick auf das Google Symbol der Name des identifizierten Objekts an die Googlesuche übergeben werden, damit der Nutzer nach Informationen zu diesem suchen kann, das ist in Abbildung 1.2 zu betrachten. GoogleLens eignet sich folglich für den Fall, dass ein Objekt, im Falle dieser Arbeit eine Obst oder Gemüsesorte, unbekannt ist, um dieses zu identifizieren. Die Informationen zu diesem Objekt müssen jedoch selbst gesucht werden. Dies ist der erste

Unterschied zu der geplanten Anwendung, denn diese soll dem Nutzer generierte Informationen zu der erkannten Obst oder Gemüsesorte anzeigen, ohne dass er selbst nach diesen suchen muss. Ein weiterer Unterschied ist die Notwendigkeit einer Verbindung zum Internet. Die geplante Anwendung soll die Objekte auch ohne Zugang zum Internet erkennen können, damit diese auch an Orten mit schlechtem Internet schnell Informationen liefern kann. Auch lässt sich an Abbildung 1.3 erkennen, dass beispielsweise die Kartoffel fälschlicherweise als Mango identifiziert wurde und dies obwohl die ersten visuellen Übereinstimmungen Kartoffeln sind, wie sich in Abbildung 1.4 zeigt. Auch hier soll die geplante Anwendung falsche Klassifizierungen vermeiden, wobei es eine Herausforderung darstellen wird, die Genauigkeit von GoogleLens zu übertreffen. Hier wurde jedoch keine Studie zu Obst und Gemüsesorten gefunden, die Genauigkeit sollte sich jedoch nicht prägnant von der Erkennung von Pflanzen unterscheiden, da diese die gleichen Eigenschaften wie Obst aufweisen und es sich bei Obst und Gemüse auch um Pflanzen handelt. Des Weiteren soll die Anwendung die Erkennung in Echtzeit ausführen, sodass die Namen der erkannten Obstsorten sofort angezeigt werden, sobald diese erkannt wurden.

4 Verwandte Arbeiten

Die Erkennung von Obst und Gemüse wurde bereits von einigen Autoren erforscht.

Ein Beispiel bieten Duong et al.[7]. Die Autoren wollen mit ihrer Anwendung ermöglichen, Obst automatisch und aus jeder Position heraus zu erkennen. Dafür verwenden sie MixedNet und EfficientNet als Grundlagen für ihre neuronalen Netze. Diese vortrainierten Netze wollen sie mit eigenen Anpassungen verbessern. Bei Efficient Net handelt es sich um ein Netz, welches 3-Dimensionen skalieren und durch eine Balance der Netzdimensionen genau und effizient arbeiten soll. MixedNet basiert auf dem Aufbau von Mobile Net und soll, mit Hilfe einer Reduktion von Parametern, eine Balance zwischen Effizienz und Genauigkeit bieten.[7]

Femling et al.[9] haben mit einem RasperryPi, ausgestattet mit einem Display, einer Kamera und einer Waage ein System entwickelt, welche dem Kunden dabei helfen soll, das von ihm ausgewählte Obst schneller zu erkennen. Damit soll er dieses leichter abwiegen können. Das Ziel war dabei, ein möglichst wenig komplexes und wenig zeitaufwendiges System mit einer benutzerfreundlichen UI zu entwickeln. Als Problem identifizierten die Autoren das variable Aussehen von Obst und Gemüse. Um zehn verschiedene Obstsorten unverpackt zu erkennen, verwendeten die Autoren die Methode des Transferlernens, indem sie bereits vortrainierte Netze weiter anpassten. Die verwendeten Netze sind Inceptionv3 von Google und Mobile-Net. Bei beiden Netzen, in ihrer nicht modifizierten Form, traten bei unterschiedlichen Obstsorten Probleme bei der korrekten Erkennung auf, z.B. bei der von Kiwis. Ein großer Unterschied zwischen beiden Netzen ist die Zeit, welche für die Erkennung benötigt wird. So benötigt Inception durchschnittlich 3,3 Sekunden, während Mobile-Net nur 0.43 Sekunden braucht.[9]

Lawal stellt in seiner Arbeit ein Modell für das Erkennen von Tomaten vor. Dieses soll Ernterobotern ermöglichen die Ernte der Tomaten autonom zu bewerkstelligen. Dafür hat er die Architektur von YOLOv3 leicht abgewandelt und diese mit einem Trainingsdatensatz für Tomaten trainiert.[15]

Martinez et al. haben ein von ihnen trainiertes TinyYOLOv2 Modell anhand der Unity Bibliothek Barracuda in Unity integriert. Dieses haben sie mit Hilfe von Nahaufnahmen von Autos auf diese trainiert, um damit den Verkehrsfluss zu messen. Dafür haben sie einige Vorverarbeitungsschritte genutzt, um die Autos in den an das neuronale Netz übergebenen Bildern schärfer darzustellen. Dies führte zu einer Verbesserung der Leistung des Modells.[18]

Ein Beispiel für den Nutzen von maschinellem Lernen in Xcode mit ARKit zeigen Le et al. auf. Darin haben sie ein YOLOv3 Modell trainiert und in ein Modell für Create ML konvertiert. Sie haben YOLOv3 und nicht YOLOv2 verwendet, da YOLOv2 Schwierigkeiten dabei hat, kleinere Objekte zu erkennen.[16]

5 Anforderungen und Konzept

Wie bereits in der Einleitung erwähnt ist das Ziel dieser Arbeit, ein Setup zu entwickeln, womit eine Anwendung realisiert werden kann, welche dem Nutzer eine schnelle Informationsgewinnung über eine ihm bekannte oder unbekannte Obst oder Gemüsesorte ermöglicht.

Die Informationsgewinnung kann durch eine AR-Anwendung realisiert werden, schließlich ist es das Konzept von AR-Anwendungen, virtuelle Objekte in der realen Welt in Echtzeit zu integrieren.[3]

Zunächst stellt sich hier die Frage nach einem Framework, in welchem die AR-Anwendung realisiert werden kann. Dabei kommen Unity, Unreal Engine und Xcode in Frage, da sich alle drei dazu eignen AR-Anwendungen zu entwickeln. Aufgrund des Umfangs des Projektes werden hier nur zwei der drei Möglichkeiten näher untersucht. Die Wahl fällt hierbei auf Unity und Xcode, da bei diesen Frameworks bereits Vorerfahrung existiert, weshalb nicht das komplette Framework neu erlernt werden muss. Unity besitzt gegenüber Xcode einen Vorteil, da Unity plattformunabhängige Entwicklung ermöglicht und die Anwendung somit auf mehreren AR-Endgeräten getestet und genutzt werden kann. Da alle Funktionen von AR Foundation jedoch nur von iOS genutzt werden können, existiert hier bereits eine Einschränkung, sodass nicht alle Funktionen genutzt werden können, wenn Plattformunabhängigkeit gewährleistet sein soll. Da Xcode mit ARKit dieselben Funktionen wie auch AR Foundation bietet, hätte die Verwendung von Xcode in dem Fall keinen Nachteil.

Des Weiteren ist die zuverlässige Erkennung von Obst- oder Gemüsesorten essentiell. Sie ist die Grundfunktion, um Informationen zu diesen anzeigen zu können. Um eine zuverlässige Erkennung der Obst oder Gemüsesorten gewährleisten zu können, wird ein neuronales Netz verwendet. Zwar ließe sich durch AR Foundation in Unity und deren Äquivalenz in ARKit in Xcode auch eine Erkennung anhand von Referenzbildern ermöglichen, jedoch ist dies aufgrund

der verschiedenen Formen, welche eine Obst oder Gemüsesorte annehmen kann nicht praktikabel, da somit jede Form, die ein Obst oder Gemüse annehmen kann, als Referenzbild vorhanden sein müsste. neuronale Netze bieten wie im Grundlagenkapitel beschrieben den Vorteil, anhand des Trainings auch ähnliche Formen erkennen zu können. Des Weiteren, wäre auch der Speicherbedarf der Anwendung zu groß, wenn alle existierenden Obst- und Gemüsesorten als Referenzbild lokal in der Anwendung vorliegen müssten. Die oben genannten Frameworks müssen folglich die Möglichkeit besitzen, neuronale Netze für das Erkennen der Objekte zu integrieren. Mit Barracuda für Unity und den Apple eigenen mlModels bieten beide Frameworks diese Möglichkeit. Barracuda von Unity besitzt jedoch erhebliche Einschränkungen, da nur bestimmte Modelltypen und Architekturen verwendet werden können. Des Weiteren müssen die Modelle zunächst in einem anderen Framework trainiert und anschließend in das ONNX Format konvertiert werden. ARKit bietet hier den Vorteil, dass man mit Create ML selbst Modelle in dem benötigten Format erstellen kann und die Einbindung eines Modelles damit auf jedenfall möglich ist.

Da es mit erheblichen Aufwand verbunden wäre, Trainingsdaten zu jeder Obst- und Gemüsesorte zu sammeln, wird zunächst der Ansatz des Transferlernens untersucht. Hierbei werden wie auch bei [9] und [7] mehrere Netze getestet und deren Schwächen und Stärken bei einer Verwendung mit einem iPhone analysiert. Die Wahl der Netze fällt hierbei auf MobileNet, welches auch schon von [9] verwendet wurde und Tiny-YOLO3, welches bereits von [17] und in der großen Version von [15] und [16] verwendet wurde. Als erstes wird hier MobileNet getestet, da dieses mehr Objekte erkennen kann als Tiny-YOLOv3, welches mit keiner Obstsorte vortrainiert wurde. Tiny-YOLOv3 bietet jedoch den Vorteil, die Objekte auch lokalisieren zu können und mehrere Objekte gleichzeitig zu erkennen. Auch wenn sich Tiny-YOLOv2 nach Angaben der Dokumentation von Barracuda verwenden lassen soll, wird aufgrund dessen geringer durchschnittlichen Genauigkeit von ca. 22% darauf verzichtet. Hierbei wäre der optimale Fall, wenn sich MobileNet und dessen Ergebnis verwenden ließe, da sich hier der Ansatz von Transferlernen lohnen würde, weshalb hier weniger Trainingsdaten gesammelt werden müssten.

Zusammenfassend sind sowohl Unity als auch Xcode theoretisch geeignet, eine Grundlage für eine AR Anwendung zu bilden, welche die genannten Anforderungen erfüllt. Da Unity jedoch Plattform unabhängige Entwicklung ermöglicht wird Unity gegenüber Xcode zunächst bevorzugt. Für den Fall, dass sich die getesteten neuronalen Netze nicht in Unity integrieren lassen, oder sich die Netze als nicht praktikabel herausstellen, wird alternativ eine Anwendung in Xcode entwickelt, da Xcode in Bezug auf neuronale Netze mehr Möglichkeiten bietet. Die zwei Setups werden in Abbildung 2 gegenübergestellt.

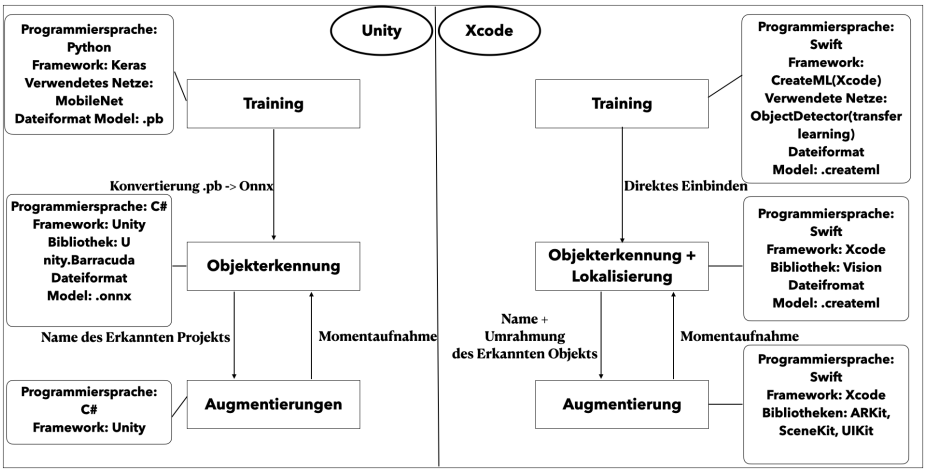


Abb. 2. Die zwei Setups gegenübergestellt

6 Implementierung und Ergebnisse

6.1 Umsetzung in Unity

Implementierung Die Entwicklung eines Setups in Unity erfolgte in zwei Schritten. Der erste Schritt war das Umwandeln des vortrainierten Netzes MobileNet in ein ONNX Modell. Dafür wurde MobileNetv2 aus der Keras Bibliothek heruntergeladen und als .pb Datei lokal abgespeichert. Danach wurde das Modell mit der Python Bibliothek tf2onnx[8] in ein ONNX Modell umgewandelt. Dieses Modell wurde nun in einem neu erstellten Modellordner in Unity eingefügt. Dass die Umwandlung erfolgreich war, lässt sich in Abbildung 3 erkennen. So wird das MobileNet Modell(genannt fruit) mit dem Modell Symbol von Unity angezeigt. Bei Auswahl des Modells können im rechten Fenster noch zusätzliche Informationen wie Eingabe und Ausgabe und die einzelnen Schichten angezeigt werden. Der zweite Schritt war das AR-Setup in Unity. Dafür wurde zunächst das Unity Paket ARFoundation und ARKitXRPlugin installiert. Um neuronale Netze verwenden zu können, wurde das Paket Barracuda mit der Version 2.0.0 gewählt, da dieses die neuste für den produktiven Einsatz bereite Version ist und die neueren noch im experimentellen Modus befindlichen Versionen gemäß der Dokumentation keine sichtbaren Vorteile gegenüber dieser bieten. Anschließend wurde das standardmäßige Setup von Unity gelöscht und ein fertiges AR-Session-Origin mit Hilfe der UI von Unity hinzugefügt. Dieses enthält eine AR Camera. Als nächstes wurde ein eigenes Skript angelegt und als Komponente der AR-Session Origin übergeben. Dieses Skript wird verwendet, um Anhand von MobileNet eine Erkennung durchzuführen und das Klassifizierte Objekt als TextObjekt auszugeben. Dafür wird zunächst ein Kamerabild von der AR Camera über den in dieser integrierten AR Camera Manager entnommen. Dieses muss nun noch von einem Plane Bild in ein RGB Bild gemäß der Anleitung in [19]

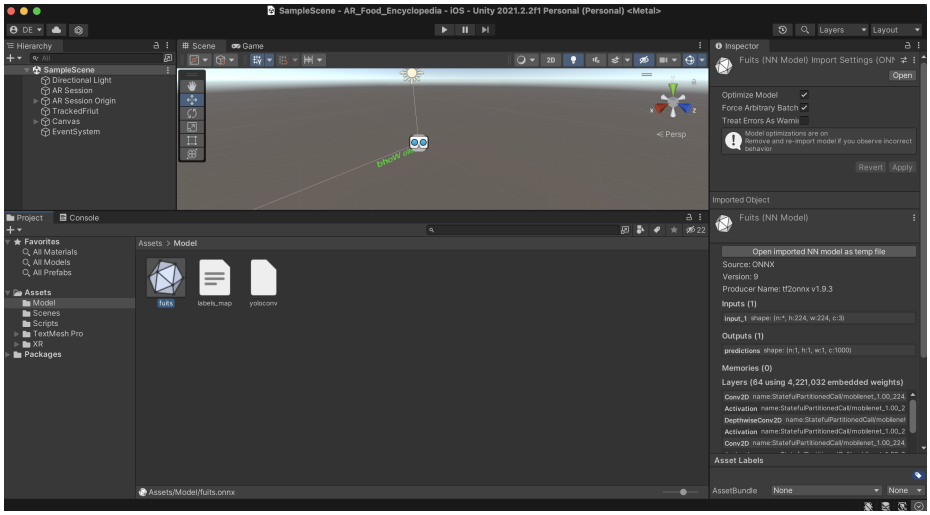


Abb. 3. Erfolgreiche Umwandlung von MobileNet in ein für Unity nutzbares ONNX Modell

umgewandelt werden. Dieses gemäß der Anleitung in einem Buffer gespeicherte Bild wird nun in ein Texture2D Objekt umgewandelt und anschließend in einen Tensorobjekt transformiert. Dieser Tensor kann nun als Input an das Modell übergeben werden. Mit PeekOutput wird dem Modell die Vorhersage entnommen. Davon wird das Ergebnis mit der höchsten Wahrscheinlichkeit genutzt. Die Stelle des höchsten Ergebnisses wird nun genutzt, um das entsprechende Label zu generieren. Die Schritte nach der Erstellung des RGB-Bildes sind in Abbildung 4 zu sehen. Wichtig ist hierbei, die Ein- und Ausgabe des Modells mit .Dispose() freizugeben, da das Programm ansonsten nach einer Erkennung abstürzt. Tiny-YOLOv3 ließ sich nicht in ein Unity nutzbares Modell umwandeln. Für den Versuch wurden zwei verschiedene Github Repositories[6][23] für das Konvertieren der Tiny-YOLO Modelle in ein Tensorflow Modell verwendet. Besonders das Repository von [23] ist dabei von Interesse, da es bereits eine Klasse für das Transferlernen integriert, zusätzlich kann die Funktion eines in DarkNet trainierten YOLOv3 oder Tiny-YOLOv3 Modell nach der Umwandlung mit der bereitgestellten detect.py Datei, getestet werden. Das Importieren und Umwandeln der Tiny-YOLOv3 Version in ein .pb Modell klappte mit beiden Reposetories. Zusätzlich wurde die detect.py Datei von[23] für das mit diesem Repository importierten Modells getestet. Der Test zeigte, dass das Modell noch funktioniert. Die beiden Tiny-YOLOv3 Modelle wurden nun mit der Bibliothek tf2onnx, welche auch bei MobileNet verwendet wurde, in ein onnx Modell umgewandelt. Hierbei traten ein paar Warnungen auf, dass bestimmte Pythonklassen nicht gefunden wurden, die Umwandlung selbst wurde aber als erfolgreich beschrieben und es wurde eine onnx Datei erstellt. Diese Modelle wurde nun wie auch MobileNet in den Modelordner im Unity Projekt gelegt. Dabei kam es bei

```
m_Texture = new Texture2D(
    conversionParams.outputDimensions.x,
    conversionParams.outputDimensions.y,
    conversionParams.outputFormat,
    false);

m_Texture.LoadRawTextureData(buffer);
m_Texture.Apply();
buffer.Dispose();
var model= ModelLoader.Load(fruitModel);
var engine = WorkerFactory.CreateWorker(WorkerFactory.Type.ComputePrecompiled, model);
var input = new Tensor(m_Texture, 3);
engine.Execute(input);
Tensor output = engine.PeekOutput();
Debug.Log(output);
var arrayOutputs= output.ToReadOnlyArray();
int index = output.ArgMax()[0];
var fruit = getMobileNetLabels(index);
Debug.Log(fruit);
input.Dispose();
output.Dispose();
engine.Dispose();
fruitName = fruit;
```

Abb. 4. Implementierungsausschnitt der Erkennung mit Barracuda mit MobileNet

beiden Modellen zu unterschiedlichen Fehlern, welche jeweils in Abbildung 5 und 6 zu sehen sind. Beide Modelle zeigen auch nicht das Unity Symbol für ein Modell an, wie sich an yoloconv, ein mit [6] und tf2onnx umgewandeltes Tiny-YOLOv3 Modell, in Abbildung 3 sehen lässt. Eine Lösung um diesen Fehler zu beheben, wurde zunächst nicht gefunden.

Abb. 5. Errors des mit [6] + tf2onnx[8] konvertierten Modells

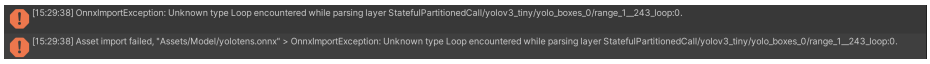


Abb. 6. Errors des mit [23] + tf2onnx[8] konvertierten Modells

Ergebnis Auch wenn sich MobileNet mit Unity einbinden ließ, wurde die Weiterentwicklung des Setups mit Unity abgebrochen. Dies lag daran, dass sich

MobileNet als reiner Klassifizierer für diesen Anwendungsfall nicht eignen würde. Gerade wenn mehrere Objekte im Bild sind, was z.B. auch im Supermarkt der Fall sein würde, fällt es schwer nachzuvollziehen, welches Objekt erkannt wurde. Des Weiteren ist die Erkennung von nur einem Objekt eine große Einschränkung. Die Einbindung eines Objekterkenners in der Form von YOLOv3, welcher die Schwächen von MobileNet nicht besitzt, funktionierte in dem Fall zunächst nicht. Der Fehler bei der Konvertierung ist aufgrund fehlender Erfahrung in diesem Bereich und der vielen nötigen Schritte nicht nachvollziehbar gewesen. Des Weiteren wird Tiny-YOLOv3 nach der offiziellen Seite von Barracuda nicht unterstützt, zwar konnte Deren Lei[17] mit der Barracuda Version 1.2.3 es einbauen, es könnte jedoch sein, dass dies mit der aktuellen Unity Version(2021.2.f1) nicht mehr funktioniert. Tiny-YOLOv2, welches auch nach der offiziellen Seite von Barracuda unterstützt wird, hat jedoch eine, verglichen mit den anderen YOLO Modellen, geringe durchschnittliche Erkennung. Des Weiteren ist die fehlende offizielle Unterstützung für weitere Objekterkenner damit ein großer Nachteil, falls sich Tiny-YOLOv2 auch allgemein nicht dazu eignen sollte, viele Obst- oder Gemüsesorten zu unterscheiden. Dies ist jedoch möglich, da nach Le et al.[16] schon die größere YOLOv2 Version Probleme mit dem Erkennen von kleinen Objekten hat und Obst und Gemüse sind kleine Objekte. Mit ARKit sollte die Entwicklung einer auf neuronalen Netzen basierenden Anwendung, aufgrund der Möglichkeit selbst neuronale Netze zu erstellen möglich sein. Zusätzlich haben auch Le et al. [16] bereits ein mit DarkNet trainiertes Modell in ARKit einbinden können, wodurch es auch möglich ist, gegebenenfalls in einem anderen Framework erstellte Modelle einbauen zu können. Die Entwicklung der Anwendung ist folglich in Xcode weniger riskant als in Unity. Deshalb wird in der nächsten Phase nun die Entwicklung eines Setups mit Xcode durchgeführt.

Es hat sich im späteren Projektverlauf herausgestellt, dass in einer neueren, aber experimentellen Version 2.4.0 von Barracuda das yoloconv Modell als dieses von Unity identifizieren ließ. Dies geschah jedoch erst nach der Vollendung des nächsten Schritts, weshalb diese die Entscheidung nicht beeinflusst hat. Die Version 2.4.0 wurde zunächst nicht getestet, da diese laut Dokumentation[22] keinen Unterschied in den unterstützten Architekturen aufweist.

6.2 Umsetzung der Alternative in Xcode

Ziel dieser alternativen Phase ist es sicherzustellen, dass ein Modell in Xcode mit einem für ARKit bereitgestelltem Setup eingebunden werden kann, damit anhand dessen die Entwicklung einer vollständigen Anwendung ermöglicht wird. Dafür soll wie auch im Setup von Unity ein Textobjekt das erkannte Obst oder Gemüse anzeigen. In diesem Fall wird die Transferlearning Vorlage von Create ML verwendet, dabei handelt es sich um ein Modell für einen Object Detector. Das Modell wird vorerst nur mit zwei Apfel- und zwei Birnensorten trainiert. Im Ergebnis wird bewertet, ob sich das Setup eignet, um die Entwicklung der Anwendung mit Xcode fortzuführen. Die Pipeline des Projekts wird in den folgenden Abschnitten vorgestellt.

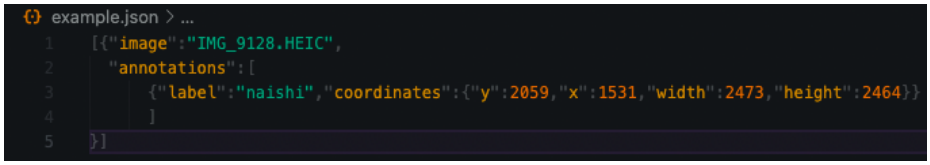


Abb. 7. Das benötigte Format für die Label der Trainingsbilder zum Trainieren eines Objekterkenners

Erstellung der Trainingsdaten Zunächst müssen die Trainingsdaten erstellt werden. Dabei erfordert Create ML ein bestimmtes Format, welches in Abbildung 7 exemplarisch dargestellt wird. Hierfür wurde sich für die kostenpflichtige Software RectLabelPro[12] entschieden. RectLabelPro unterstützt den Export der Trainingsdaten in verschiedene Formate, so z.B. für Create ML und Tensorflow. Dies ermöglicht im Falle des Nutzens einer anderen Netzarchitektur die bereits vorhandenen Trainingsdaten weiter zu verwenden. Die Funktion ist dabei nutzerfreundlich, so zieht man mit der Maus ein Rechteck, um das Objekt und weist ihm einen Namen zu. Dieser Schritt ist in Abbildung 8 zu sehen.

Training des Modells Die im vorherigen Abschnitt generierten Trainingsdaten werden nun in das entsprechende Feld für die Trainingsdaten eingefügt. In Abbildung 9 ist erkenntlich, dass dieser Schritt funktioniert hat, da die Klassen sowie die vorliegenden Trainingsbilder erkannt wurden. Bis auf die Auswahlmöglichkeit bei der Batchsize und Trainingsdaten, kann das Modell jedoch nicht weiter beeinflusst werden. Nachdem das Training gestartet wurde, kann man nun den Verlauf des Trainings betrachten. Hierbei ist in Abbildung 10 die schnelle Verbesserung des Loss-Wertes deutlich erkennbar. Nachdem alle Trainingsdurchläufe fertig sind, kann das Modell mit weiteren Bildern oder mit einem iPhone getestet werden. Das Testen des Trainingerfolges mit dem iPhone erfolgt anhand von Liveaufnahmen.

Einbinden des Modells Die .createML Datei, welche nach dem Training vorliegt muss nun in das Projekt eingebunden werden. Dafür wird das Modell zunächst dem Projekt hinzugefügt. Mit der Vision Bibliothek von Xcode wird das Modell in der Anwendung nutzbar gemacht.

Die Erkennung Um Objekte in einem Bild zu erkennen, wird alle 2 Sekunden das aktuelle Bild der iPhone Kamera an das Modell übergeben. Hierbei muss das Bild nicht herunter skaliert werden, dies macht Vision automatisch. Die Ergebnisse des Modells liegen nach der Bearbeitung mit dem Namen der erkannten Objekte, einem Rechteck, um diese zu lokalisieren und einem Übereinstimmungswert vor, welcher die Wahrscheinlichkeit angibt, dass es sich wirklich um das erkannte Objekt handelt.

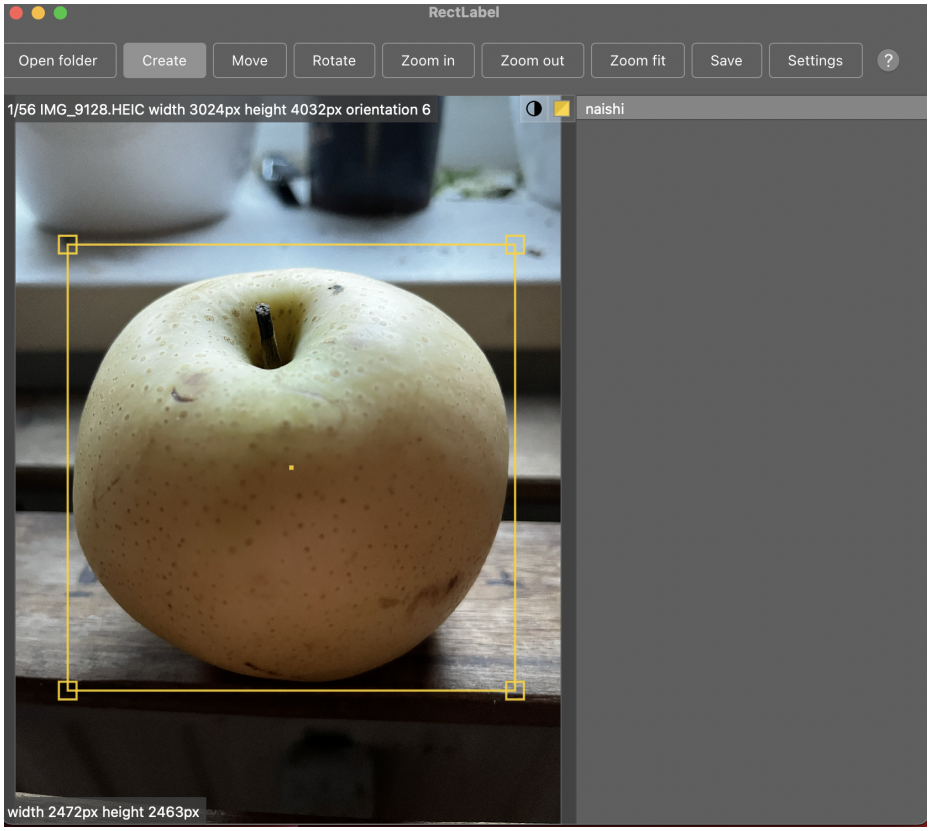


Abb. 8. Der Prozess des Annotierens mit RectLabelPro

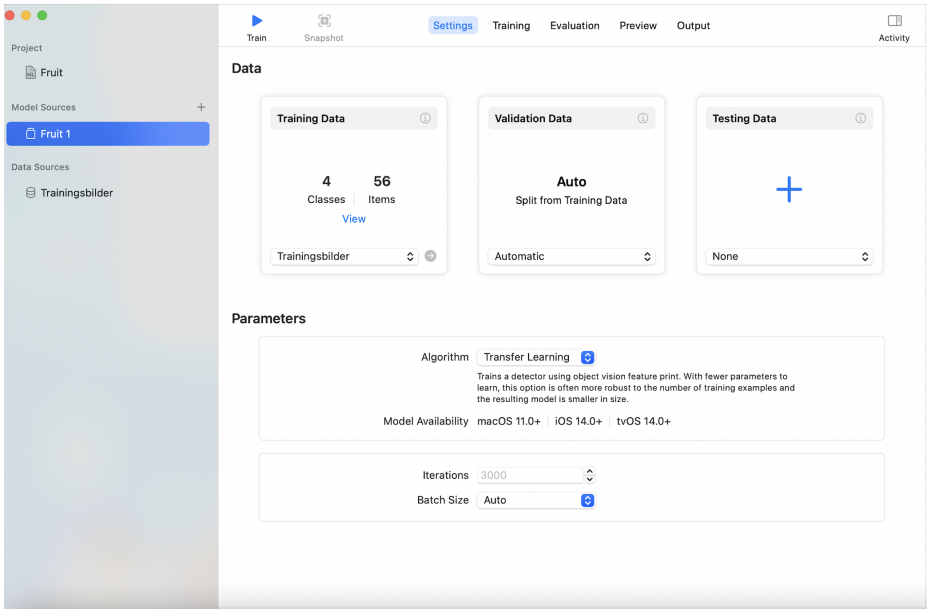


Abb. 9. Der Import des Trainingdatensatzes hat funktioniert

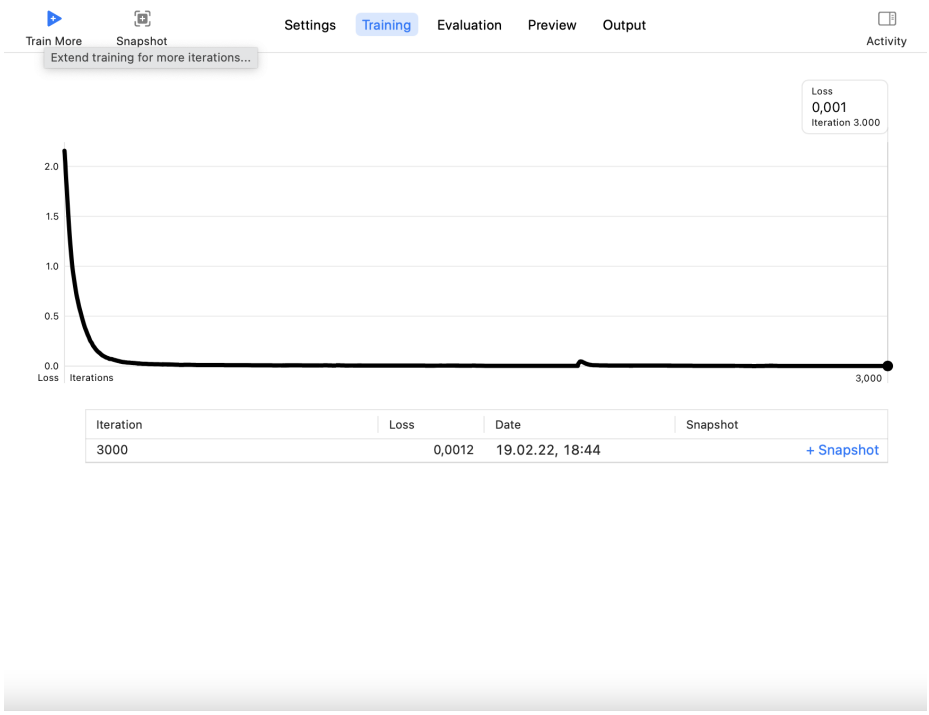


Abb. 10. Darstellung des Ergebnisses in Xcode mit ARKit

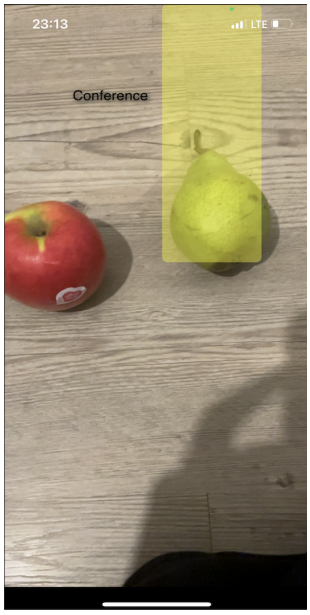


Abb. 11. Das Frontend des Setups

Die Darstellung der Ergebnisse Die bisher einzigen Augmentierungen sind die Rechtecke um das Objekt herum und deren Namen. Dargestellt werden nur Objekte, welche mit einer Wahrscheinlichkeit von 0,9 richtig erkannt wurden. Um die Ergebnisse des Modells verwenden zu können, wird das Rechteck mit der aktuellen Bildschirmgröße normalisiert. Die Darstellung ist in Abbildung 11 ersichtlich.

Ergebnis Die Einbindung des Modells erfolgte ohne Probleme. Eine Herausforderung stellt die richtige Skalierung der Ergebnisse dar. Da das Modell ein Bild der Größe 412x412 entgegen nimmt, wird ein nicht quadratischen Bild in ein quadratisches umgewandelt. Dies führt noch zu einer leichten Verfälschung der Ergebnisse bei der Lokalisierung. Für das Training wurden jedoch Bilder der selben Bildgröße verwendet. Das Training selbst bietet jedoch wenig Einflussmöglichkeiten aus Entwicklersicht. Ob die Vorlagen für Objekterkenner auch für Obstsorten geeignet sind, muss noch festgestellt werden. In der aktuellen Verwendung kommt es häufig zu Verwechslungen zwischen Obstorten oder auch einer falschen Position. Da es sich aber bisher nur um experimentelle Trainingsdaten ohne Optimierung handelt, muss der Trainingsprozess mit Create ML noch weiter getestet werden.

7 Fazit

In dieser Arbeit wurden zwei mögliche Setups für eine AR-Anwendung mit der Verwendung eines neuronalen Netzes für das Tracking verwendet. Hierbei hat sich Xcode als das geeignetere Setup präsentiert. Dies liegt vor allem an der lückenhaften Dokumentation von Unity Barracuda, da die fehlende Dokumentation der unterstützten Modelle ein Problem darstellt. Auch die benötigten Konvertierungen von Modellen, gerade im Fall von YOLO, stellen einen Mehraufwand dar. Des Weiteren führt nicht jedes Git Projekt zu einem Erfolg bezüglich der Integration in Unity. Hierbei ist es wegen der fehlenden Dokumentation schwer nachzuvollziehen, ob das Modell generell nicht in Unity genutzt werden kann, ob selbst ein Fehler bei der Konvertierung gemacht wurde, oder ob das Git Projekt für die Konvertierung für Unity nicht funktioniert. Xcode bietet hierbei den Vorteil, dass dieses selbst ein Tool bereitstellt, um verschiedene Arten von neuronalen Netzen, unter anderem einen Objekterkenner zu erstellen. Diese lassen sich auch ohne größere Probleme in eine AR-Anwendung integrieren. Lediglich die Normalisierung der Positionen der erkannten Objekte stellt hierbei ein Problem dar. Außerdem könnten auch noch andere Modelle, welche nicht mit Create ML erstellt wurden, z.B. YOLO, verwendet werden. Mit RectLabel Pro kann dies ohne größeren Mehraufwand erfolgen, da die Bilder beispielsweise auch in das für YOLO benötigte Format exportiert werden können. Falls die Integration in Xcode scheitert, können die annotierten Trainingsbilder somit trotzdem noch in Create ML verwendet werden.

8 Ausblick

Um eine Anwendung zu realisieren, welche dem Nutzer weiterhilft, muss das Setup weiter ausgebaut werden. Zunächst muss das Training weiter optimiert werden, weil das Tracking und Erkennen der Obst- und Gemüsesorten den Grundbaustein des Projektes darstellt. Da mit Create ML selbst nicht viel an den Modellen verändert werden kann, sollte dies zunächst mit der Optimierung der Trainingsdaten getestet werden. Dafür könnten z.B. künstliche Trainingsbilder zu den aufgenommenen Obstsorten generiert werden. Dies wäre auch nötig, da Obstsorten an vielen Orten angetroffen werden können. Beispielsweise wäre das an Bäumen, im Obstregal, in Obstkörben, oder auf den verschiedensten Arten von Tischen und Tellern, auf denen sie liegen können. Auch spielen die Lichtverhältnisse und die Qualität der Kamera eine Rolle. Die verschiedenen Umgebungen mit jeder Obst und Gemüsesorte manuell aufzunehmen, ist dabei zu zeitaufwendig. Sobald die Erkennung der Obst- und Gemüsesorten zuverlässig wirkt, muss festgestellt werden, welche Informationen für die Nutzer relevant sind und diese müssen anschließend in einem geeigneten Format dargestellt werden.

Literatur

1. Apple: Arkit (2021), `https://developer.apple.com/augmented-reality/`

2. Apple: Create ml (2021), https://developer.apple.com/documentation/createml
3. Azuma, R.T.: A survey of augmented reality. Presence: teleoperators & virtual environments **6**(4), 355–385 (1997)
4. Bilyk, Z.I., Shapovalov, Y.B., Shapovalov, V.B., Megalinska, A.P., Andruszkiewicz, F., Dołhańczuk-Śródka, A.: Assessment of mobile phone applications feasibility on plant recognition: comparison with google lens ar-app. CEUR Workshop Proceedings (2020)
5. Deru, M., Ndiaye, A.: Deep learning mit tensorflow, keras und tensorflow.js (2020)
6. Do, B.: tensorflow-lite-yolo-v3 (2021), https://github.com/peace195/tensorflow-lite-YOLOv3
7. Duong, L.T., Nguyen, P.T., Di Sipio, C., Di Ruscio, D.: Automated fruit recognition using efficientnet and mixnet. Computers and Electronics in Agriculture **171**, 105326 (2020)
8. Exchange, O.N.N.: tf2onnx - convert tensorflow, keras, tensorflow.js and tflite models to onnx (2021), https://github.com/onnx/tensorflow-onnx
9. Femling, F., Olsson, A., Alonso-Fernandez, F.: Fruit and vegetable identification using machine learning for retail applications. In: 2018 14th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS). pp. 9–15. IEEE (2018)
10. Google: Google lens (2021), https://lens.google
11. Juliani, A., Berges, V.P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., et al.: Unity: A general platform for intelligent agents. arXiv preprint arXiv:1809.02627 (2018)
12. Kawamura, R.: Rectlabel pro (2022), https://rectlabel.com
13. Khan, A., Sohail, A., Zahoora, U., Qureshi, A.S.: A survey of the recent architectures of deep convolutional neural networks. Artificial Intelligence Review **53**(8), 5455–5516 (2020)
14. Kim, S.K., Kang, S.J., Choi, Y.J., Choi, M.H., Hong, M.: Augmented-reality survey: from concept to application. KSII Transactions on Internet and Information Systems (TIIS) **11**(2), 982–1004 (2017)
15. Lawal, M.O.: Tomato detection based on modified yolov3 framework. Scientific Reports **11**(1), 1–11 (2021)
16. Le, H., Nguyen, M., Yan, W.Q., Nguyen, H.: Augmented reality and machine learning incorporation using yolov3 and arkit. Applied Sciences **11**(13), 6006 (2021)
17. Lei, D.: Object detection with localization using unity barracuda and arfoundation (2020), https://derenlei.medium.com/object-detection-with-localization-using-unity-barracuda-and-arfoundation-794b4eff02f8
18. Martinez, L.M., Salisbury, J.P., Scott, N.V.: Application of tiny yolov2 car detection for traffic flow quantification during the covid health pandemic: Preliminary results
19. Unity: Accessing the device camera image on the cpu (2021), https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.0/manual/cpu-camera-image.html
20. Unity: Ar foundation manual (2021), https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.2/manual/index.html
21. Unity: Baracuda manual (2021), https://docs.unity3d.com/Packages/com.unity.barracuda@2.2/manual/index.html
22. Unity: Baracuda manual (2021), https://docs.unity3d.com/Packages/com.unity.barracuda@2.4/manual/SupportedArchitectures.html

23. Zhang, Z.: Yolov3 implemented in tensorflow 2.0 (2021), `https://github.com/zzh8829/yolov3-tf2`
24. Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., He, Q.: A comprehensive survey on transfer learning. Proceedings of the IEEE **109**(1), 43–76 (2020)