



KANTONSSCHULE WOHLEN
PROJEKTUNTERRICHT

GestureCook: Rezepte-App mit Gestensteuerung durch KI

Silvan Metzker

geleitet von
Patric ROUSSELOT

28. Januar, 2020

1 Vorwort

Die Idee für ein solches Projekt kam mir während einer Diskussion mit meinem Vater in den Ferien. Mein Vater arbeitet bei einem Hausgeräte-Hersteller und erzählte mir über die Idee von Küchengeräten, welche durch eine Kochapp gesteuert werden. Auch erzählte er mir von einem Wegesystem im Kochherd, welche mithilfe des Gewichtes der Kochapp sagt, wie viele Gramm einer Zutat noch fehlen. Ich hatte damals schon immer die Absicht, etwas mithilfe von Künstlicher Intelligenz, kurz KI, zu analysieren. So kam mir damals die Idee einer App, welche mithilfe Gesten gesteuert wird. Dies soll innerhalb von Xcode geschehen, der Programmierumgebung für die Entwicklung von iOS-Apps.

2 Einleitung

Stellen Sie sich vor, sie stehen in der Küche. Auf der Küchenoberfläche liegt Ihr Smartphone, auf welchem eine Kochapp ein Rezept anzeigt. Ihre Hände sind noch schmutzig vom vorherigen Kochschritt. Doch nun wollen Sie gerne den nächsten Kochschritt anzeigen. Sie wollen das Smartphone nicht berühren, denn sonst wird es auch dreckig. Genau dieses Problem löst die App, welche in diesem Bericht thematisiert wird. Mit zwei Handgesten lässt sich so die App steuern.

Das Prinzip ist einfach, die Frontkamera, welche praktisch jedes Smartphone hat, nimmt konstant Bilder auf. Diese werden dann mithilfe einer Künstlichen Intelligenz analysiert. Doch ist so etwas überhaupt möglich? In den folgenden Kapiteln wird die Planung sowie Probleme und Produkt dokumentiert.

3 Hauptteil

3.1 Planung

Der erste Teil meiner Planung war die Evaluierung der Projektidee. Ist es möglich meine Idee zu verwirklichen? Ist sie zu schwer?

Diese Fragen liessen sich schnell mit gezielten Google-Suchanfragen lösen. Nachfolgend entwarf ich Skizzen (siehe Abb. 1) von der App. Wie sie aussehen soll und welche Arten von UI-Typen, also verschiedene Benutzeroberflächenelemente, verwendet werden sollen. So kam ich schliesslich auf die

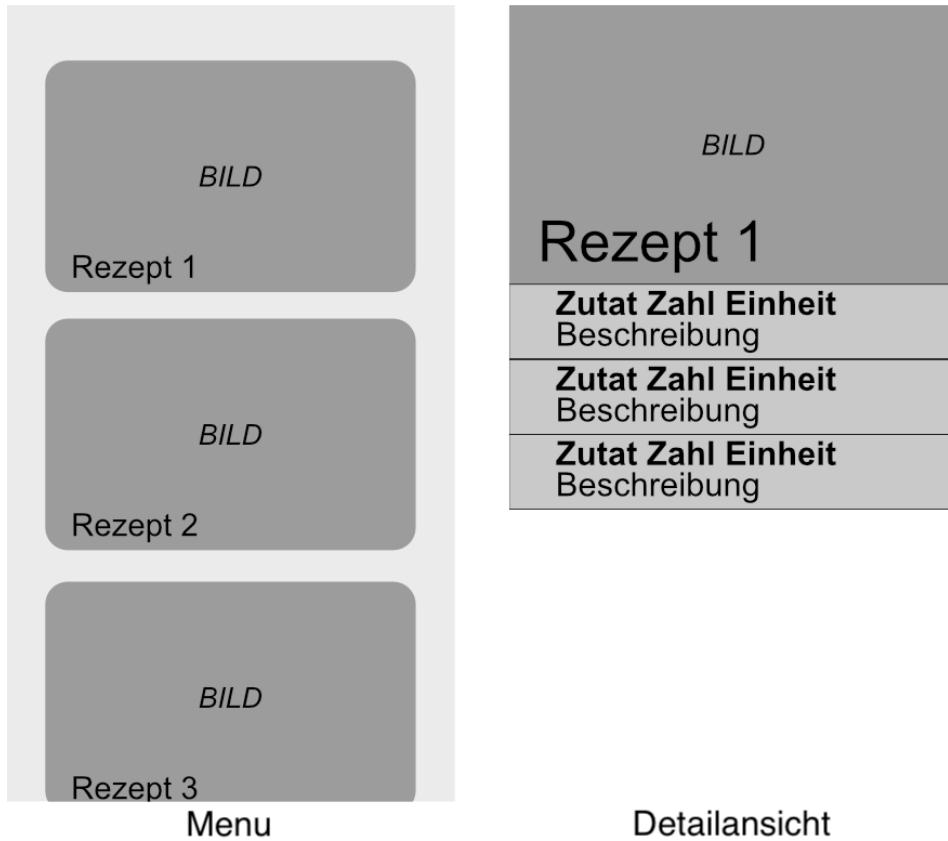


Abbildung 1: Frühe Skizze

Verwendung von UITableViews. Nach dieser Feststellung musste ich herausfinden woher ich Daten für die App, beziehungsweise die Kochrezepte, kriege und in welchem Format. So gelang ich mithilfe meines Vaters an Rezepte der V-ZUG AG in einem XML-Format. Deshalb musste ich mir überlegen, wie ich die XML-Daten in iterierbare Swift Datenstrukturen umwandeln kann. Dafür erstellte ich ebenfalls eine weitere Notiz für den Aufbau der XML-Datei (siehe Listing 1), welche ich von V-ZUG erhalten habe. Weiteres zum XML-Parser wird im nächsten Kapitel erklärt.

Listing 1: XML-Skizze

Legende:
Zeichen:

```

> Verschachteln
* Für jedes Item
/ Oder
Erstes Element: XML-Element(e)
Zweites Element: XML-Attribut(e)
Nach "": Drittes Element: Swift-Variable(n)

> declarations
>> short-name
>>> lang: recipeName
>> categories
>>> duration type: preparation: prepTime
>>> duration type: cooking: cookingTime
>> ingredient-categories
>>>* cat id: catIngredient_X: ingredientID
>>>> name
>>>>> lang: ingredientName
>>>> unit
>>>>> lang: ingredientUnit
>> tasks
>>> group type: equipment
>>>> message
>>>>> lang: equipment[x]
>>> group type: nil
>>>> title
>>>>> lang: taskName
>>>>* ingredient/max ref: ingredientID
>>>>> lang: message
>>>>> amount min/max: minMax[min,max]

```

3.2 XML-Parser

Der erste Schritt zur Kochapp fängt im sogenannten „Back-End“ an. Auf Englisch „back end“ bedeutet „hinteres Ende“, gemeint ist damit den Teil eines Programmcodes, mit welchem der Endnutzer nicht in Berührung kommt. In dem nachfolgenden Kapitel wird dann das „Front-End“ beschrieben, dabei handelt es sich um das Interface, mit welchem der Endnutzer konfrontiert wird und direkt zu Gesicht bekommt. Wie bereits erwähnt bemerkt der Benutzer das Back-End nicht direkt. In diesem Fall ein XML-Parser, dieser hat

den Auftrag eine sogenannte XML-Datei für die App lesbar zu machen. Jedes Rezept wird in einer XML-Datei gespeichert. Eine XML-Datei ist eine spezielle Art von Datenstruktur, in welcher Informationen mithilfe von sogenannten „Tags“ gespeichert werden. Tags sind Objekte wie z.B. `<lang>`, welche von einem schliessenden Tag geschlossen werden `</lang>` (siehe gekürztes XML-Beispiel in Listing 2). [2]

Listing 2: XML-Beispiel gekürzt, Quelle: VZUG AG

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:recipe xmlns:tns="http://vzug.ch/zughome/recipe-definition">
  <declarations>
    <device id="CS-SL-15">
      <recipe-id>
        <short-name>
          <lang ref="de">Confierter Lachs</lang>
        </short-name>
      </recipe-id>
    </device>
  </declarations>
</tns:recipe>
```

Die Information wird dann entweder innerhalb eines Tags gespeichert als *Attribut* `<device id="CS-SL-15">`. Oder *zwischen* dem öffnenden Tag und dem schliessendem Tag `<lang>Confierter Lachs</lang>`.

Mit diesem Wissen wendete ich mich an den XML-Parser von Xcode, bzw. der Programmierungsumgebung für iOS-Apps. Das Problem besteht darin, dass es keinen XML-Parser gibt für Xcode, welcher eine XML-Datei einliest und eine Tabelle ausgibt, so wie es dies in z.B. Python gibt. Denn den einfachsten Programmcode für einen XML-Parser ist dieser hier:

Listing 3: XML-Parser in Swift

```
// Beispiel für eine Variable welche anzeigt ob ein Element
// geöffnet wurde oder nicht.
var shortNameElement = false

// Beispiel für eine Variable welche den ausgelesenen Wert
// temporär trägt
var elementName: String = ""
```

```

// Beispiel für eine Variable welche den ausgelesenen Wert
// temporär trägt und später in eine Struktur übertragen wird
var shortName: String = ""

// 1 - Element wurde geöffnet
func parser(_ parser: XMLParser, didStartElement elementName:
    String, namespaceURI: String?, qualifiedName qName: String?,
    attributes attributeDict: [String : String] = [:]) {
    if elementName == "short-name" {
        shortNameElement = true
    }
    self.elementName = elementName
}

// 2 - Element wurde geschlossen
func parser(_ parser: XMLParser, didEndElement elementName:
    String, namespaceURI: String?, qualifiedName qName: String?) {
    if elementName == "short-name" {
        shortNameElement = false
    }
}

// 3 - Interpretiere die Daten im Element
func parser(_ parser: XMLParser, foundCharacters string: String) {
    let data = string.trimmingCharacters(in:
        CharacterSet.whitespacesAndNewlines)
    if (!data.isEmpty) {
        if self.elementName == "lang" && shortNameElement {
            shortName = data
        }
    }
}

```

Das Grundlegende Prinzip ist einfach, der Parser hat drei Funktionen, je nachdem wo der Parser sich befindet wird eine davon aufgerufen. Die erste der drei wird aufgerufen, wenn ein Element geöffnet wurde und ändert so eine Variable auf den Wert „true“, sodass die dritte Funktion weiß, welches Objekt geöffnet wurde.

Die zweite Funktion ändert die gleiche Variable wieder auf „false“ sobald das

Element geschlossen wird.

Die dritte Funktion interpretiert schlussendlich den Wert zwischen zwei Elementen und speichert diesen in eine temporäre Variable. Diese Funktion weiss nur mithilfe der ersten und der zweiten Funktion welches Element gerade interpretiert wird. Attribute können in der ersten Funktion geprüft werden. [3] Nun haben wir eine Durcheinander von Variablen, welche vom Parser geschrieben wurden. Um dies zu ordnen werden all ausgelesenen Variablen in einer Struktur gespeichert. Eine Struktur ist eine Anordnung von Variablen in Swift (siehe Listing 4).

Listing 4: Struktur für ein Rezept

```
struct recipeStruct {
    var shortName: String = ""
    var prepTime: String = ""
    var cookingTime: String = ""
    var equipment: [String] = []
    var ingredientCategories: [(ingredientID: String, name: String,
        unit: String)] = [] //X: ingredientX[0: ID, 1: Name, 2:
    Unit]
    var tasks: [(groupTitle: String, tasks: [(message: String,
        ingredientID: String, minMax: [String])])] = [] //X:
    groupX[0: groupTitle, X: taskX[0: message, 1: ingredientID,
    2: minMax[min,max]]]
}
```

Im XML-Parser wurden mir bereits die drei Variablen und die drei Arrays (ein Array ist etwas ähnliches wie eine Liste), bzw. geschachtelte Arrays, erstellt. Das heisst die können jetzt direkt in dieser Struktur gespeichert werden. So ist eine dieser Strukturen ein Rezept, also kann man alle Strukturen zu einem Array hinzufügen und man erhält eine Liste von Rezepten. Diese Liste kann nun direkt vom Front-End iteriert werden und angezeigt werden. Das Bild ist gleichnamig wie das Rezept, also kann man die Variable „shortName“ benutzen um das zugehörige Bild anzuzeigen. Da die Struktur der XML-Datei viel komplexer ist als das Beispiel (siehe Listing 3), werden auch viel mehr spezifische Variablen erstellt werden. Im Endeffekt muss ein XML-Parser in Swift genau auf die zu erwartende XML-Dateistruktur angepasst werden. Denn je nachdem was man auslesen will, müssen andere Variablen gewählt werden. Im nachfolgenden Kapitel wird das Front-End erklärt.



Abbildung 2: Komponente der TableView

3.3 UITableView

Im vorherigen Kapitel wurden die Daten bereit gemacht um dargestellt zu werden. Die Rezepte sind nun in einem Array von Strukturen. Was ist eine „UITableView“? Jeder von uns hat eine solche bereits verwendet, es gibt sie auf Android sowie iOS. UITableViews sind einfache Listen. Sie bestehen aus zwei Komponenten. Der eine Teil ist die UITableView selbst und der zweite eine UITableViewCell (siehe Abb. 2), also eine „Zelle“ analog aus Tabellen. Solche UITableViews können auch in Sections eingeteilt werden. Praktisch ist, dass das Design von Xcode erstellt wird. Sogar die Steuerung, bzw. das Scrollen und die Navigation übernimmt Xcode. Denn wenn man eine UITableView importiert, kommt diese automatisch mit einem Navigation Controller. Dieser übernimmt einem die Arbeit von Knöpfen die einen Bildschirm zurückgehen, oder einen Titel auf der Oberseite des Bildschirms.

Allerdings kann nicht alles automatisch erstellt werden, denn eine normale

UITableViewCell ist eine Zelle mit einem Label (ein Text) im Innern. Um Bilder oder mehrere Labels, bei welchen die eine z.B. fett geschrieben ist, in einer Zelle zu erstellen (siehe Abb. 1), braucht es CustomUITableViewCells. Diese kann man einmal erstellen und dann tausendfach mit verschiedenem Inhalt anzeigen lassen. Diese Möglichkeit erlaubt es einem nun die vorher kreierte Liste zu iterieren und so für jeden Schritt des Rezeptes, sowie des Titels mit Bild, anzuzeigen. [1], [4]

3.4 Künstliche Intelligenz trainieren

Nun haben wir eine voll funktionale Rezepte-App erstellt. Allerdings fehlt noch der wichtigste Teil der App, die Gestensteuerung. Das Ziel ist es, die Library „Core ML“ von Apple zu verwenden um die Künstliche Intelligenz zu erstellen und damit den Output der Frontkamera auszuwerten. Das „ML“ steht für „Machine Learning“ also Maschinelles Lernen. Diese Bezeichnung ist sehr passend, denn tatsächlich muss die KI zuerst „lernen“ um akkurate Ergebnisse zu erzielen. Gerade mithilfe Core ML ist dies ein sehr einfacher Prozess, denn es gibt sogar eine Applikation von Apple namens „Create ML“, welche einem beim Trainieren hilft. In diesem Fall soll die Gestensteuerung wie folgt funktionieren: Sobald der Benutzer die Hand über die Kamera hält und eine Faust formt, soll die App nach unten scrollen. Wenn der Nutzer die Hand wieder wegnimmt soll es aufhören zu scrollen. Mit weit gespreizter Hand, kann er nun wieder nach oben scrollen.

So soll die KI zwischen drei Fällen unterscheiden, erstens einer Faust, zweitens einer gespreizten Hand und drittens „Nichts“. „Nichts“ für den Fall, dass der Nutzer die Hand nicht über der Kamera hält, also keine Aktion gestartet werden soll. Das Training ist nun nichts weiter als Bilder von Fäusten, gespreizten Händen und leeren Hintergründen bzw. eine Decke, ein Gesicht oder unklare Handzeichen. Innerhalb der Create ML Applikation kann man nun drei Ordner mit dem Namen der drei Fälle und den zugehörigen Bildern importieren, diese werden dann von der Künstlichen Intelligenz analysiert. Doch nur das Training ohne Auswertung hat keine Bedeutung, man weiss nicht, ob die Künstliche Intelligenz funktioniert oder nicht. Deshalb macht man weitere Fotos von diesen drei Fällen, welche die KI noch nie gesehen hat und lässt diese von der KI evaluieren. So bekam ich mit insgesamt 707 Bildern eine Genauigkeit von 91% (siehe Abb. 3) was weitaus genug genau für Gestensteuerung ist, mehr dazu im nächsten Kapitel. [5] Doch diese 90% erhielt ich erst nach vielen Fehlversuchen. Anfangs war die Idee, als Gesten für

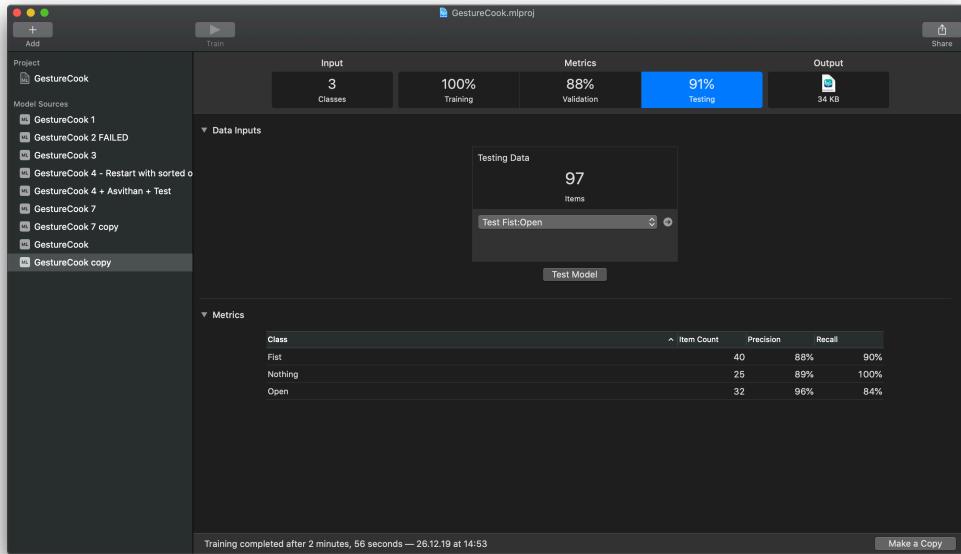


Abbildung 3: Create ML Testresultat nach ungefähr 700 Bildern

hoch und runter den Daumen nach oben, bzw. nach unten zu definieren. Dies stellte sich allerdings als beinahe unerkennbar für die KI heraus. Mit ganzen 746 Bildern erhielt ich eine Genauigkeit von nur 45%, dies ist zu ungenau für eine Unterscheidung von drei Fällen. Ich versuchte die Bilder in möglichst vielen verschiedenen Lichtsituationen aufzunehmen, denn die KI sollte in vielen verschiedenen Umgebungen funktionieren. Einer meiner Fehler war auch mit den neuen Gesten, die fehlende Qualität der Bilder. Auch wenn unklare Bilder die KI verbessern kann, können diese auch zu unklar sein. Entweder war die Handgeste nicht genug eindeutig (siehe Abb. 4a) oder die Lichtverhältnisse sind zu schlecht, damit die KI das Bild erkennt (siehe Abb. 4b). Ich stellte sicher, dass die Gesten auch für einen Menschen klar erkennbar sind (siehe Beispiele Abb. 4c und 4d), denn falls dies nicht der Fall ist, verwirrt dieses Bild die KI. Wobei eine grosse Datenmenge aber unqualitative Bilder auch eine Taktik sein kann, Fehlentscheidungen vorzubeugen, bin ich zum Schluss gekommen, dass der Fokus auf Qualität des Materials effektiver ist. Deshalb Qualität vor Quantität. Erstaunlicherweise funktionierte die KI sehr gut bei meinem dunkelhäutigen Kollegen, ohne dass ein Bild einer dunkelhäutigen Hand in den Trainingsdaten vorhanden war. In der Finalen Version

allerdings wurden auch dunkelhäutige Hände als Trainingsdaten verwendet um eine möglichst akkurate Schätzung zu erzeugen.
Als Resultat der fertig trainierten KI erhält man eine Datei mit dem Suffix „.mlmodel“, welche erstaunlicherweise nur um die 34 Kilobyte gross ist.



(a) Bild scharf, aber Geste unklar



(b) Lichtverhältnisse schlecht, unklar



(c) Klar erkennbare Faust



(d) Klar erkennbare gespreizte Hand

Abbildung 4: Qualität der Trainingsdaten beeinflusst das Ergebnis der KI.

3.5 Künstliche Intelligenz implementieren

Die KI zu trainieren ist ein Punkt, aber Sie muss noch implementiert werden. Die Implementation ist ein langer Prozess mit vielen Details, welche ich hier nur grob dokumentiere. Die Anwendung des erhaltenen ML Model geschieht mithilfe von Vision, ein Framework konzipiert einzig und allein für die Anwendung von Künstlicher Intelligenz. Dies ist ein Prozess aus drei Schritten. Zuerst braucht man einen Request, das ist der erste Teil. Im Request spezifiziert man was man schlussendlich erhalten will und was erkennt werden soll. Der zweite Teil ist der Request Handler, in diesem wird der KI die zu analysierten Daten überbracht. Mit anderen Worten schiesst die Kamera konstant Bilder, welche dann der Künstlichen Intelligenz übertragen und von dieser analysiert werden. Im dritten Schritt werden die Observierungen der Künstlichen Intelligenz ausgewertet und demnach auch Aktionen aufgrund von diesen Beobachtungen ausgeführt. [5] Im letzten Schritt wird auch klar, wieso eine Genauigkeit von 92% ausreichend ist für eine Gestensteuerung. Verantwortlich ist dieser Stück des Codes:

Listing 5: KI Buffer

```
let bufferSize = 5
var commandBuffer = [RemoteCommand]()
var currentCommand:RemoteCommand = .none{
    didSet {
        commandBuffer.append(currentCommand)
        if commandBuffer.count == bufferSize {
            if commandBuffer.filter({$0 == currentCommand}).count == bufferSize{
                showAndSendCommand(currentCommand)
            }
            commandBuffer.removeAll()
        }
    }
}
```

Hier wird ein Buffer erstellt. Die Variable „currentCommand“ wird mit jedem Bild und der neuen Schätzung der KI aktualisiert. Wenn man diese Variable mit jedem Bild ausgibt. Dann die Faust macht sie wieder öffnet und dann die Hand wegnimmt, bekommt man eine solche Stringfolge:

Listing 6: Output KI-Resultate

```
Fist,  
Fist, Fist, Fist, Open, Open, Open, Open, Open, Open, Open,  
Open, Open, Open, Open, Open, Open, Open, Open, Open,  
Open, Open, Open, Fist, Fist, Nothing, Nothing, Nothing,  
Nothing, Nothing, Nothing, Nothing, Nothing, Nothing,  
Nothing
```

Der Buffer geht jedes Element durch und schaut nun wo fünf gleiche Elemente hintereinander folgen. Dies schliesst Fehlentscheidungen aus. Dadurch wird die Wahrscheinlichkeit für eine Fehlentscheidung von 8% auf 0.00032768% reduziert. Diese Berechnung basiert allerdings auf den Testresultaten und ist stark abhängig von den Testdaten, sollte also nicht zu ernst genommen werden. Allerdings wissen wir, dass es sehr selten zu einer Fehlentscheidung im getesteten Situationen kommen soll. Ein Nebeneffekt des Buffers ist allerdings eine kleine Verzögerung. Die Gestensteuerung funktioniert besser als erwartet, sogar in dunklen Räumen oder mit verschiedenen Handtypen funktioniert sie sehr gut. Einzig mit besonders grossen Händen hatte die KI Schwierigkeiten und bei falschem Abstand.

3.6 Publikation auf dem App Store

Sobald die App fertig ist muss sie nur noch publiziert werden. Da meine App in der Programmierumgebung Xcode mit Swift programmiert wurde ist nur eine Veröffentlichung auf dem App Store möglich. Die Publikation setzt mehrere Punkte voraus. Allererstes benötigt man einen Apple Developer Account, welcher 100 CHF im Jahr kostet. Diese muss man bereitstellen um Apps zu auf dem App Store zu veröffentlichen. Auch sollte die App natürlich gegen keine Richtlinien von Apple verstossen, sowie deren Designgrundlagen folgen. Ebenfalls braucht man eine Website mit einer „Privacy Policy“, dies ist eine Website mit Bestimmungen für Behandlung von Nutzerdaten. Eine solche Website lässt sich über Dritte kostenlos generieren mit allen wichtigen rechtlichen Bestimmungen die man braucht. Ein App-Icon wird genauso benötigt wie mindestens ein Screenshot und eine Beschreibung. Als Icon (siehe Titelblatt) verwendete ich den Buchstabe G mit einem darin versteckten C, welches die Initialen von GestureCook darstellt. Schlussendlich schaut das Personal von Apple die App an und man wird entweder angenommen oder abgelehnt und muss die App überarbeiten. Bei mir ging dies allerdings ohne

Probleme und wurde beim ersten Versuch angenommen. Die App ist nun im App Store verfügbar und momentan kostenlos.

4 Fazit

Wie mit jedem Projekt, lernt man enorm viel dazu. Auch bei diesem Projekt musste ich die Grenzen meines Wissens stark übertreten, allerdings ist genau dieser Prozess, der den einem antreibt. Ein fehlerloses Programm gibt es erst nach tausenden Fehlern, aber aus jedem davon lernt man denselben wieder zu vermeiden.

Das Projekt hatte eine breite Themenspanne, ich lernte KI anzuwenden, zu trainieren, UITableViews mit veränderten UITableViewCells und natürlich den XML-Parser. Insgesamt hatte ich wenig Probleme, mit der Ausnahme von falsch ausgewählten Gesten und tiefer Qualität des Datenmaterials. Also müssen bei einer Gestensteuerung die Gesten klar erkennbar sein. Auch müssen die Trainingsdaten von hoher Qualität sein, denn auch grosse Bildermengen können schlechte Daten schwer ausgleichen.

Das Projekt zeigt, dass es nicht so schwer ist eine Künstliche Intelligenz zu verwenden. Weil bestehende Libraries einem helfen mit wenig Aufwand, gute Resultate zu erreichen. Es ist fast schwerer einen XML-Parser in Xcode zu erstellen, denn je nach XML-Datei sieht dieser wieder anders aus und muss auf den Anwendungsbereich angepasst werden. Die Gestensteuerung funktioniert einiges besser als erwartet und erkennt die Gesten in fast allen Umgebungen. Sie lässt einem auch mit dreckigen Händen die App bedienen. Mit einem Buffer lässt sich die Wahrscheinlichkeit für eine Fehlentscheidung noch um ein Vielfaches verkleinern, hinterlässt allerdings eine kleine Verzögerung. Im Ganzen war dieses Projekt ein Erfolg und ich bin positiv Überrascht vom Ergebnis.

Bibliografie

- [1] Sean Allen. Swift UITableView tutorial with custom cells—Beginner series. YouTube, 2017. https://www.youtube.com/watch?v=Ft05QT2D_H8&t.
- [2] Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, Franois Yergeau, et al. Extensible markup language (xml) 1.0, 2000.
- [3] Arthur Knopper. Parse xml ios tutorial. *IOSCREATOR*, <https://www.ioscreator.com/tutorials/parse-xml-ios-tutorial>, 2019.
- [4] Apple. Uitableview. *Apple Developer Documentation*, <https://developer.apple.com/documentation/uikit/uitableview>, 2020.
- [5] Brian Advent. Build a cool touchless gesture ui with core ml and vision. ios machine learning tutorial, 2019. <https://www.youtube.com/watch?v=G7DYmhZMz6k>.