



**School of  
Engineering**

ICP Institute of  
Computational Physics

## **Bachelorarbeit (Informatik)**

# Bildverarbeitung von Fluidbewegungen im Pharmabereich mittels künstlicher Intelligenz

---

**Autoren**

---

Sven Aebersold  
William Wong

---

**Hauptbetreuung**

---

Prof. Dr. Gernot Boiger

---

**Nebenbetreuung**

---

Marco Hostettler  
Dr. Daniel Brunner

---

**Datum**

---

10.06.2021

## Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

**Ort, Datum:**

Winterthur, 10.06.2021

Winterthur, 10.06.2021

**Name Studierende:**

Sven Aebersold

William Wong

# Zusammenfassung

Viele Medikamente werden in flüssiger (fluid) Form transportiert. Beim Transport gibt es Schwingungen, die sich auf die flüssigen Medikamente auswirken und diese in Bewegung versetzen können. Diese Instabilitäten tragen das Risiko, dass die in der Flüssigkeit enthaltenen Wirkstoffe (oftmals Proteinverbindungen) zerstört werden. Roche Pharma (Schweiz) AG versucht diesen Zerstörungseffekt in Experimenten zu untersuchen und die besten Transportbedingungen zu finden. Die Experimente werden mit Hochgeschwindigkeitskameras aufgenommen und danach manuell analysiert. In dieser Arbeit wird ein Ansatz untersucht, die Analyse der aufgenommenen Videos zu automatisieren. Hierfür wird ein Convolutional Neural Network, kurz CNN, mit selbst annotierten Daten trainiert. Die Daten werden im Laboratorium der ZHAW auf einem hauseigenen Shaker aufgenommen und danach manuell mit dem *MATLAB Video Labeler* annotiert. Um die Daten für das Netzwerk brauchbar zu machen, werden diese zuerst durch eine eigene Pipeline aufbereitet. Um Overfitting aufgrund eines kleinen und unausgeglichene Datensets zu überwinden, werden zudem Data Augmentation und Balancierung der Daten durchgeführt. Die Effekte dieser Verfahren auf das trainierte Modell und den Trainingsverlauf werden in definierten Experimenten untersucht. Ebenfalls werden mehrere Modelle mit verschiedenen Hyperparametern und Architekturen trainiert, um die besten Einstellungen für das vorliegende Klassifizierungsproblem zu finden. Es wird festgestellt, dass sich Data Augmentation und die Balancierung positiv auf das Ergebnis auswirken. Von den untersuchten Hyperparametern werden die besten Einstellungen für Batchsize, Kernel-Grösse, Anzahl Kernels, sowie Anzahl Dense Layer und Neuronen identifiziert. Mit dem besten trainierten Modell konnte ein FBeta-Score von 96 % bei der Erkennung der Instabilitäten erzielt werden. Jedoch ist das Modell auf den verwendeten Shaker sowie die vorhandenen Lichtverhältnisse spezifiziert. Ein Wechsel in der Experimentumgebung kann die Ergebnisse drastisch verschlechtern und zur Folge müssten neue Trainingsdaten gesammelt werden. Durch ein grösseres und vielfältigeres Dataset kann die Genauigkeit und die Verwendbarkeit des Modells weiter gesteigert werden.

# Abstract

Many drugs are transported in liquid (fluid) form. During transport, vibrations affect the liquid drugs and can set them in motion. These instabilities carry a risk of destroying the active ingredients (often protein compounds) contained in the fluid. Roche Pharma (Switzerland) AG is trying to investigate this destructive effect in experiments and to find the best transport conditions. The experiments are recorded with high-speed cameras and then analysed manually. In this thesis, an approach to automate the analysis of the recorded videos is investigated. For this purpose, a Convolutional Neural Network, or CNN, is trained with self-annotated data. The data is recorded in the ZHAW laboratory on an in-house shaker and then manually annotated using *MATLAB Video Labeler*. To make the data usable for the network, it is first processed through a custom pipeline. To overcome overfitting due to a small and unbalanced data set, augmentation and balancing are additionally performed on the data. The effects of these procedures on the trained model and the training process are investigated in defined experiments. Furthermore, several models are trained with different hyperparameters and architectures to find the best configuration for this classification problem. It is determined that data augmentation and balancing have a positive effect on the result. Of the hyperparameters studied, the best settings for batch size, kernel size, number of kernels, and number of dense layers and neurons are identified. The best-trained model achieves a FBeta-Score of 96 % in detecting the instabilities. However, the model is specific to the shaker and lighting conditions used. Changing the experiment environment can drastically degrade the results and consequently, new training data must be collected. A larger and more diverse dataset can further increase the accuracy and usability of the model.

# Vorwort

An dieser Stelle möchten wir uns bei Herrn Marco Hostettler, Herrn Dr. Daniel Brunner und Herrn Prof. Gernot Boiger für die Betreuung der Bachelorarbeit bedanken. Besonders möchten wir uns herzlich für die Unterstützung, Tipps und konstruktive Kritik bedanken.

Auch richtet sich unser Dank an Herrn Vincent Buff, der uns bei den Experimenten an der ZHAW stets ausgeholfen hat und für uns auch länger im Labor geblieben ist.

Ausserdem möchten wir uns bei Lukas Egli und Gianluca Vinzi für den Austausch von Fachwissen und die angenehme Zusammenarbeit während dem Filmen der Videos bedanken.

# Inhalt

1.	Einleitung .....	5
1.1	Ausgangslage .....	5
1.2	Zielsetzung .....	6
2.	Theoretische Grundlagen .....	7
2.1	Maschinelles Lernen (ML) .....	7
2.2	Artificial Neural Networks (ANN) .....	9
2.3	Computer Vision .....	11
2.4	Convolutional Neural Networks (CNN) .....	12
3.	Methoden .....	15
3.1	Datengewinnung und Aufbereitung .....	15
3.2	Data Augmentation und Balancierung .....	17
3.3	Aufbau und Training des Convolutional Neural Network (CNN) .....	20
3.4	Evaluierung der Hyperparameter und Modelle .....	25
4.	Resultate .....	30
4.1	Augmentation .....	30
4.2	Balancierung .....	32
4.3	Batchsize .....	34
4.4	Einfluss der Kernel-Grösse .....	35
4.5	Einfluss der Anzahl Kernel .....	36
4.6	Einfluss der Anzahl Dense Layer .....	38
4.7	Andere Architekturen .....	40
4.8	Analyse von neuen Videos .....	41
4.9	Zusammenfassung der Experimente .....	47
5.	Diskussion und Ausblick .....	48
6.	Verzeichnisse .....	49
6.1	Literaturverzeichnis .....	49
6.2	Glossar .....	53
6.3	Abbildungsverzeichnis .....	54
6.4	Tabellenverzeichnis .....	56
7.	Anhang .....	57
7.1	Kontaktinformation und Code .....	57
7.2	Offizielle Aufgabenstellung .....	57
7.3	Verwendete CNN Architekturen .....	58

# 1. Einleitung

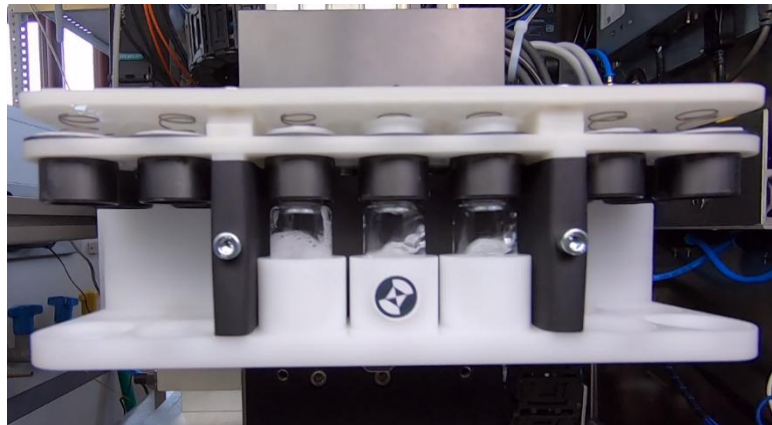


Abbildung 1: XZ-Shaker vom ZHAW ICP Lab

## 1.1 Ausgangslage

Im Pharmabereich müssen Medikamente bis zum Ende der Lieferkette sicher distribuiert werden. Beim Transport von Arzneimitteln gilt es hohe Anforderungen einzuhalten, damit die Substanzen nicht beschädigt werden. Impfstoffe müssen beispielsweise bei bestimmten Temperaturen gelagert werden, damit sie ihre Wirkung nicht verlieren [1]. Nebst Temperaturen können auch Vibrationen, die flüssige Medikamente beim Transport in Bewegung versetzen, eine Gefahr darstellen. Es besteht die Annahme, dass die Wirkstoffe, welche oftmals aus Proteinverbindungen bestehen, durch die Bewegung zerstört werden. Diese Annahme basiert darauf, dass mehr Bewegung zu mehr Scherspannung und Scherspannung wiederum zum Zerfall der Substanzen führt.

Im Labor von Roche Pharma (Schweiz) AG wird der Zerstörungseffekt von Medikamenten in Flüssigform untersucht. Medikamente werden in verschiedene Formen von Phiolen abgefüllt und mit einem von der ZHAW entwickelten Shaker (siehe Abbildung 1) in Bewegung versetzt. Der Shaker schüttelt die Phiolen in horizontaler und/oder vertikaler Richtung gemäss den eingestellten Frequenzen und Amplituden. Mit einer Hochgeschwindigkeitskamera wird der ganze Shaker aufgenommen, um die Dynamik der Fluide in den Phiolen danach manuell von Auge zu untersuchen. Hierbei gilt zu beobachten, ob Fluid-Instabilitäten wie Tröpfchen oder Schaumbildung entstanden sind. Die Experimente werden mehrfach mit anderen Formen von Phiolen sowie mit anderen Konfigurationen des Shakers wiederholt, um herauszufinden, welche Phiolen für welche Arten von Bewegungen besser geeignet sind. Das manuelle Analysieren der Hochgeschwindigkeitsaufnahmen ist allerdings aufwändig, wodurch die Idee entstand, die Untersuchung nach Fluid-Instabilitäten zu automatisieren.

Für Klassifikationsprobleme in Bildern haben sich in den letzten 10 Jahren Methoden aus dem Bereich des maschinellen Lernens etabliert [2], [3]. Eine Methode davon sind Convolutional Neural Networks (CNN), welche im Feld der Medizin schon weite Verwendung finden. CNNs konnten bereits erfolgreich für die Klassifizierung von Lungenanomalien aus Röntgenaufnahmen eingesetzt werden, um Radiologen bei Diagnosen zu unterstützen [4]. Des Weiteren wurde ein CNN mit einer Genauigkeit von 80 % entwickelt, um den Schweregrad von koronaren Herzkrankheiten anhand von Herz-CTs einzuschätzen [5].

## 1.2 Zielsetzung

In diesem Projekt wird ein Ansatz untersucht, die Auswertung der Hochgeschwindigkeitsaufnahmen mittels Methoden aus dem maschinellen Lernen (ML) zu automatisieren. Bei der Auswertung der Experimente ist entscheidend, welche Instabilitäten vorgekommen sind und mit welcher Intensität. Um im Rahmen eines Bilderkennungsproblems zu bleiben, wird in dieser Arbeit nur auf die Erkennung der Instabilitäten eingegangen. Der Zustand der Flüssigkeit lässt sich in vier grobe Kategorien aufteilen:

- Stille Oberfläche
- Wellenbildung
- Tröpfchenbildung
- Schaumbildung

Das Ziel dieser Arbeit ist es, Instabilitäten in den einzelnen Frames automatisiert mit einem Convolutional Neural Network zu erkennen und die Frames in eine dieser vier Kategorien einzuteilen. In einem ersten Schritt soll dafür ein Dataset erstellt werden, welches für das Training des CNNs verwendet werden kann. Um präzise Vorhersagen auf ungesehene Daten zu treffen, sollen verschiedene Hyperparameter und Architekturen untersucht werden. Im Verlaufe der Bachelorarbeit sollen folgende Fragestellungen beantwortet werden:

- Ist der Einsatz eines Convolutional Neural Networks geeignet, um die Aufnahmen der Experimente zu untersuchen?
- Was für einen Einfluss haben die Menge und Verteilung der Daten auf die Ergebnisse und wie verändert Data Augmentation die Ergebnisse?
- Welche Hyperparameter und Architektur eignen sich gut?
- Was muss beim Aufnehmen der Experimente mit dem Shaker beachtet werden?



## 2. Theoretische Grundlagen

Im folgenden Kapitel werden die theoretischen Grundlagen zu dieser Arbeit aufgezeigt. Dabei wird zuerst auf die Theorie des maschinellen Lernens und der oft verwendeten Artificial Neural Networks eingegangen. Ebenfalls wird auf die Verbindung von Computer Vision mit künstlichen neuronalen Netzwerken eingegangen. Zusätzlich werden Anwendungen dieser Theorien und deren Errungenschaften aufgezeigt.

### 2.1 Maschinelles Lernen (ML)

ML ist eine Unterkategorie der künstlichen Intelligenz mit dem Ziel, den menschlichen Lernprozess zu simulieren, indem aus Erfahrungen gelernt wird [6]. Als Erfahrungen sind hierbei Daten aus dem entsprechenden Umfeld zu verstehen, in dem ein sogenannter Agent erlernen soll, sich effizient und performant zu verhalten [7]. Damit Maschinen lernen können, wurden seit der Gründung des Forschungsfelds ML diverse Algorithmen und statistische Modelle für unterschiedliche Problemstellungen entwickelt. Obwohl dieses Forschungsfeld seit Ende 1950 existiert, hat es besonders seit Beginn des 21. Jahrhunderts einen enormen Aufschwung erfahren [8]. Die Gründe für den Aufschwung sind die Verfügbarkeit von Daten im heutigen Zeitalter von Big Data [9], effizientere Algorithmen und die kontinuierliche Steigerung an Rechenleistung. Seit Beginn des 21. Jahrhunderts hat sich ML bereits in zahlreichen Bereichen etabliert [7], [10]:

- Spracherkennung
- Natural Language Processing
- Computer Vision
- Robot Control
- Analyse von medizinischen Daten
- Effizientes Spielverhalten von Spielen
- Beschleunigung von empirischen Wissenschaften

Diese Anwendungen lassen sich in drei Hauptdisziplinen des induktiven maschinellen Lernens aufteilen [7], [11]:

- *Unsupervised Learning* (unbeaufsichtigtes Lernen): Hier liegt der Fokus darauf mittels mathematischen Formeln und geschickter Darstellung von Daten, Gemeinsamkeiten und Ähnlichkeiten zu entdecken. Eine Disziplin von Unsupervised Learning ist *Clustering*, wobei Datenobjekte aufgrund von Ähnlichkeiten in Gruppen zugeordnet werden.
- *Supervised Learning* (beaufsichtigtes Lernen): Hierbei werden Daten von Hand mit Informationen zu einer Klassenzugehörigkeit oder Eigenschaften versehen und dann einem Algorithmus übergeben. Der Algorithmus versucht dann eine Funktion zu erlernen, die eine Zuordnung von Input (Daten) zu Output (Labels) darstellt.
- *Reinforcement Learning*: Bei Reinforcement Learning wird ein naturgetreuer Ansatz des Lernens verfolgt. Ein Agent wird in eine Umgebung «geworfen», in der er durch eine Folge von Aktionen, welche eine Belohnung/Bestrafung auslösen, lernen soll, für eine gegebene Situation die Aktion zu wählen, die zur grössten Belohnung führt.

Da es sich in dieser Arbeit um ein Szenario aus der Disziplin des Supervised Learning handelt, wird folgend noch etwas genauer auf Supervised Learning eingegangen. Neben den induktiven Methoden gibt es Transduktive und Deduktive Ansätze [11], auf die hier aber nicht näher eingegangen wird.

### 2.1.1 Supervised Learning

Supervised Learning hat Erfolge in realen Applikationen erzielt und kann in Domänen wie Bildklassifikation verwendet werden [12]. Supervised Learning ist eine Art von induktivem Lernen, welches analog zum menschlichen Lernen ist, da die Fähigkeit, einen Task auszuführen, von geschehenen Ereignissen gelernt wird [12]. Im Fall von Supervised Learning sind das oft Daten, die manuell untersucht und, wie in Kapitel 2.1.2 beschrieben, annotiert werden. Die Aufgabe von Supervised Learning ist dann von einem gegebenen Trainingsset von  $N$  Input-Output-Paaren:

$$(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$$

eine Funktion  $h$  zu finden, welche die wahre Funktion  $y = f(x)$  approximiert. Dabei können  $x$  und  $y$  beliebige Werte darstellen. Die Funktion  $h$  wird eine Hypothese genannt und beim Lernen geht es darum, eine Hypothese aus einem gegebenen Hypothesenraum zu wählen, die auf den Trainingsdaten, aber auch auf neuen Daten gute Vorhersagen trifft. Beispielsweise bilden alle polynomialen Funktionen einen solchen Hypothesenraum und eine Hypothese ist eine spezifische polynomiale Funktion wie  $y = 2 + 1.5x + x^2$  [11].

Das Ziel ist also eine Funktion zu finden, die Input-Werte  $x$  so gut wie möglich auf bestimmte Output-Werte  $y$  abbildet. Gelingt es der Hypothese auch bei neuen Beispielen den  $y$ -Wert vorherzusagen, generalisiert diese gut auf den gegebenen Daten.

Ist der Output  $y$  ein finites Set an Werten (z. B. Hund, Katze, Esel) handelt es sich um ein sogenanntes Klassifikationsproblem. Ist  $y$  eine beliebige Zahl aus einem vorgegebenen Wertebereich, ist es wiederum ein Regressionsproblem [11]. Um diese Probleme zu adressieren, sind einige Methoden und Hypothesenräume erforscht worden. Einige häufig verwendete Beispiele sind Decision Trees, Support Vector Machines (SVM), Random Forests, aber auch künstliche neuronale Netzwerke [11]. In Abbildung 2 ist eine Übersicht verschiedener Supervised Learning Ansätze zu sehen.

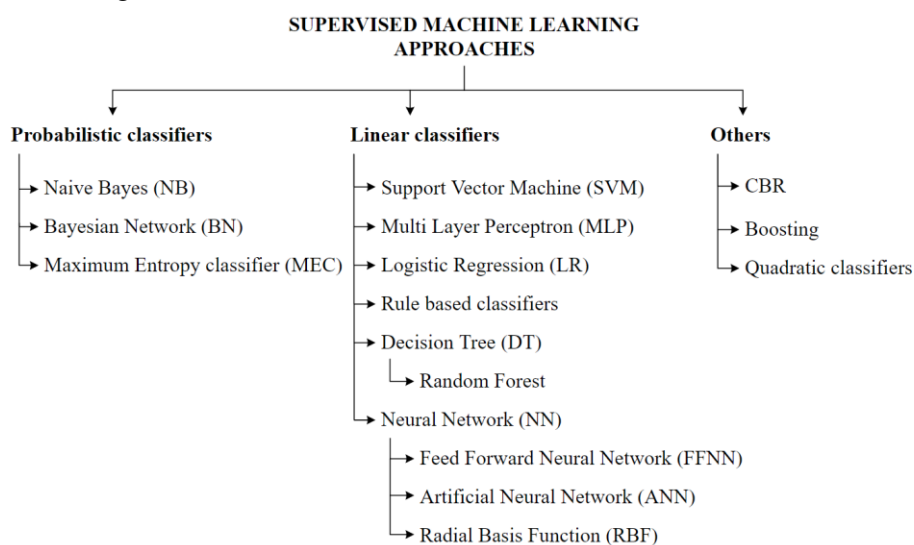


Abbildung 2: Supervised Learning Ansätze als Organigramm. Die Methoden sind in probabilistische, lineare und andere Klassifikatoren unterteilt [42].

Um die Hypothesen zu evaluieren und die Beste auswählen zu können, definiert man eine sogenannte Verlustfunktion (Loss Function). Während dem Training wird versucht, diese zu minimieren.

Die Verlustfunktion kann z. B. die mittlere quadratische Abweichung (MSE)

$$MSE = \frac{1}{N} \sum_{k=1}^N (y_k - \tilde{y}_k)^2 \quad (1)$$

von den geschätzten  $\tilde{y}$ -Werten zu den wahren  $y$ -Werten sein [11].

Die gewählte Hypothese, auch Modell genannt [13], wird auf ein Evaluationsdatenset angewendet, welches separate Daten vom Trainingsdatenset enthält. Basierend darauf, wie das Modell für diese Evaluationsdaten Vorhersagen trifft, lassen sich weitere Metriken berechnen, welche die Genauigkeit des Modells beschreiben. Einige standardisierte Metriken sind Accuracy, Recall, True Positive Rate, False Positive Rate, True Negative Rate, False Negative Rate sowie der FBeta-Score [14]. In Kapitel 3.4.2 wird näher auf diese Metriken eingegangen. Je nach Anwendung sind die Metriken unterschiedlich aussagekräftig.

#### 2.1.2 Annotation der Daten (Labeln)

Ein wichtiger Schritt, um im Supervised Learning erfolgreich ein Modell trainieren zu können, ist die Aufbereitung eines Datensets. Dabei werden die Daten, welche den Input bilden, mit dem richtigen Output-Wert versehen. Bei einem Klassifikationsproblem, bei dem auf einem Bild erkannt werden soll, ob es sich um einen Hund, eine Katze oder einen Esel handelt, werden die Daten mit der richtigen Klasse annotiert bzw. gelabelt. Das Bild und die Klasse bilden in diesem Fall ein Input-Output-Paar  $(x, y)$ . Dabei spielt die Richtigkeit dieser Annotation einen entscheidenden Faktor für die Genauigkeit des trainierten Modells. Das Erhalten von angemessenen Daten für das Training ist eine der grössten Herausforderungen im Gebiet von Supervised Learning [15].

## 2.2 Artificial Neural Networks (ANN)

Als einer der erfolgreichsten Algorithmen für Machine Learning haben sich Artificial Neural Networks herausgestellt. Gründe dafür sind die aussergewöhnlichen Informationsverarbeitungseigenschaften wie Nichtlinearität, Robustheit, Geschicklichkeit mit ungenauen/unscharfen Informationen umzugehen, und die Fähigkeit gut zu generalisieren [16].

Die erste Idee des Lernens, welches den Prozess von Neuronen im menschlichen Gehirn repliziert, stammt aus den 1940er Jahren von Donald Hebb [17], was zur Gründung der Forschung an künstlichen neuronalen Netzwerken führte. Allerdings stagnierte die Forschung im Jahr 1969, als Marvin Minsky und Seymour Papert die ungenügende Rechenleistung der damaligen Computer aufzeigten [18]. Im Jahr 2010 zeigte jedoch Dan Ciresan et al. [19], dass ein mithilfe von GPU trainiertes tiefes neuronales Netzwerk alle bisherigen Methoden auf dem Benchmark-Datensatz MNIST übertrifft. Als 2015 der Weltmeister im Brettspiel Go von AlphaGo [20] geschlagen wurde, kamen neuronale Netzwerke wieder in den Vordergrund und das «Deep Learning» nahm Fahrt auf. Heutzutage bilden Deep Learning und ANNs die Basis der meisten KI-Anwendungen [21].

### 2.2.1 Aufbau von Neuronen und Neuronalen Netzwerken

Im Gegensatz zu Support Vector Machines, Random Forest etc., deren Ideen ihren Ursprung in der Mathematik haben, kommt die Idee für künstliche neuronale Netze aus der Natur des menschlichen Gehirns. Grundsätzlich wird versucht, organische Neuronen in ihrer Funktion nachzubilden. Abbildung 3 zeigt den mathematischen Aufbau eines künstlichen Neurons. Die Output-Aktivierung der Einheit wird wie folgt berechnet:  $a_j = g(\sum_{i=0}^n w_{i,j} * a_i)$ . Dabei ist  $a_i$  die Output-Aktivierung der Einheit  $i$ , während  $w_{i,j}$  die Gewichtung der Verbindung von der Einheit  $i$  zu dieser Einheit ist.

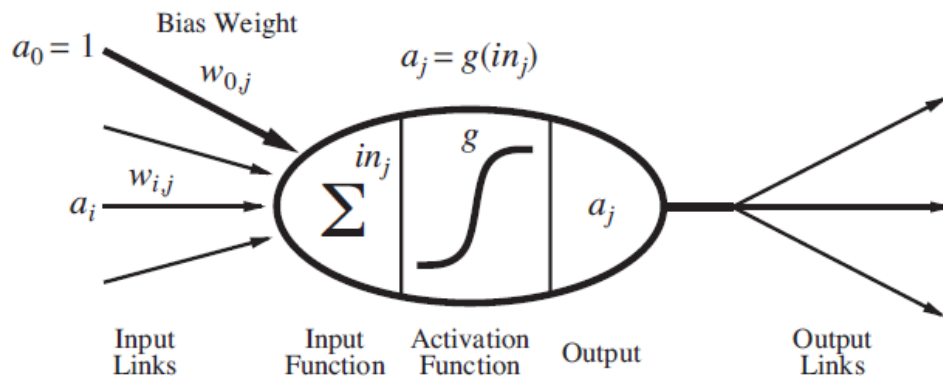


Abbildung 3: Aufbau eines künstlichen Neurons [11]. Die Input-Verbindungen sind gewichtet und werden in der Input Function aufsummiert. Die Aktivierungsfunktion bestimmt den Output, welcher mittels Output-Verbindungen an die nächsten Neuronen weitergegeben wird.

Das Neuron summiert über alle gewichteten Inputs und wendet dann die Aktivierungsfunktion an. Der von der Aktivierungsfunktion berechnete Wert wird dann als Output weiter an die nächsten Neuronen geschickt.

Neuronale Netzwerke bestehen aus mehreren Schichten mit solchen Neuronen, wobei es zwei Hauptarchitekturen gibt. In Feed-Forward Networks haben Neuronen nur Output-Verbindungen zu den Neuronen der nächsten Schicht. Hingegen haben Neuronen bei Recurrent/Feedback-Networks auch Output-Verbindungen zu Neuronen in derselben oder sogar vorhergehenden Schichten [16]. In diesem Projekt wird ein Feed-Forward Netzwerk zusammen mit einem Bildverarbeitenden Teil, welcher ein Convolutional Neural Network ausmacht (siehe Kapitel 2.4), verwendet. Recurrent Netzwerke konnten im Rahmen des Projekts nicht näher untersucht werden.

In Abbildung 4 ist ein einfacher Aufbau eines Feed-Forward Netzwerks zu sehen. Die erste Schicht wird als Input Layer bezeichnet, welche die Daten entgegennimmt. Die letzte Schicht ist der Output Layer, welche den Output  $y$  erzeugt. Die Schichten dazwischen werden als *hidden* (versteckte) Schichten bezeichnet. Hat ein Netzwerk viele solcher versteckten Schichten, wird es als ein *tiefes* Netzwerk bezeichnet, worin der Begriff *Deep Learning* seinen Ursprung hat. Oftmals spricht man bei den einzelnen Schichten von vollständig verbundenen Schichten, weil alle Neuronen miteinander verbunden sind. Jede Verbindung hat dabei eine Gewichtung und jedes Neuron ein Bias, welche beim Lernen durch Backpropagation angepasst werden und somit das Wissen des Netzwerks speichern [16].

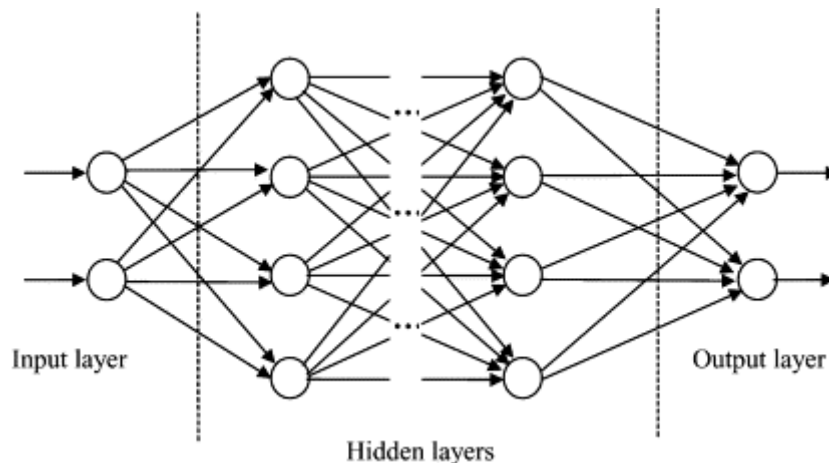


Abbildung 4: Architektur eines Feed-Forward Netzwerks [43] Zu sehen ist die Unterscheidung zwischen Input, Output sowie Hidden Layer und die Verbindungen gehen nur nach vorne.

Die gesamte Struktur des menschlichen Gehirns zu kopieren ist derzeit unmöglich aufgrund der immensen Komplexität. Allerdings konnten beispielsweise in der Arbeit von Cichy et al. [22] abstrakte Ähnlichkeiten, in der Art wie neuronale Netzwerke und das Gehirn arbeiten, gefunden werden.

## 2.3 Computer Vision

Computer Vision beschäftigt sich mit dem computerbasierten Analysieren von Bildern, um daraus Informationen zu gewinnen oder deren Inhalt zu verstehen [23]. Mithilfe von Techniken aus den Feldern Mathematik, Geometrie und ML wird versucht, aus Bildern numerische Informationen zu produzieren, um beispielsweise Entscheidungen zu treffen. Im Ingenieurwesen versucht man Computer Vision anzuwenden, um Aufgaben, die üblicherweise das menschliche visuelle System erfordert, zu automatisieren [24].

Bereits zu Beginn der 1970er Jahre wurden die ersten fundamentalen Computer Vision Algorithmen entwickelt, die bis heute Verwendung finden. Dazu gehören Optischer Fluss, Structure from Motion und erste Ansätze der Feature Detection wie Kanten- und Eckenerkennung [23].

### 2.3.1 Feature Detection

Features in Computer Vision sind Inhalte eines Bildes, die sich durch distinktive Formen, Linien oder Farben von den anderen Bereichen im Bild unterscheiden. Im Rahmen dieser Arbeit sind beispielsweise Features von Tröpfchen wie eine gebogene Oberfläche oder eine Ansammlung vieler heller Pixel bei Schaumbildung wichtig.

Feature Detection gehört zu den Kernbereichen von Computer Vision und beinhaltet Bildverarbeitungsoperationen, wobei jedes einzelne Pixel mit einer Faltungsmatrix (auch Filter oder Kernel genannt) untersucht wird. Bei der Kantendetektion werden beispielsweise die Farbwertgradienten durch die Faltung des Bildes mit einer definierten Matrix (Sobel, LaPlace etc.) an jedem Pixel berechnet [23]. Dadurch werden nicht relevante Informationen entfernt, und nur noch relevante Features, in diesem Falle die Kanten, bleiben übrig. Müssen aber komplexere Formen oder Muster detektiert werden, kommen mehrere Faltungsmatrizen zum Einsatz, die nacheinander angewendet und selbst definiert werden müssen.

### 2.3.2 Learning

Die Idee, geeignete Filter mit einem neuronalen Netz zu erlernen, wurde schon im Jahr 1979 von Kunihiko Fukushima vorgeschlagen und gilt als Inspiration der heutigen Convolutional Neural Networks [25]. Als Begründer der CNNs gilt Yann Lecun, der im Jahr 1998 erfolgreich ein Convolutional Neural Network entwickelt hat, um handgeschriebene Nummern aus Bildern in der Grösse von 32x32 Pixel zu identifizieren [26]. Für Bilder mit höherer Auflösung müssen allerdings grössere Netze trainiert werden, was aufgrund der Rechenleistung nicht möglich war. Aus diesem Grund wurden traditionelle Methoden (z. B. von Hand entworfene Filter) für Bilderkennungssysteme bevorzugt [23].

2012 gelang es Alex Krizhensky das LeNet zu erweitern und trainierte mithilfe von GPUs ein viel grösseres Netz, das sogenannte AlexNet [2]. AlexNet gewann den 2012 ImageNet Computer Vision Contest mit einer Genauigkeit von 85 %, knapp 11 % höher als die Zweitplatzierten [2]. Seitdem wurden immer mehr Erfolge mit CNNs erzielt. 2015 übertraf das von Microsoft entwickelte CNN mit einer Fehlerquote von 4.94 % die menschliche Performanz (5.1 %) auf dem ImageNet 2012 Dataset [3].

## 2.4 Convolutional Neural Networks (CNN)

Das Convolutional Neural Network ist ein Deep Learning Modell, welches von der Organisation des visuellen Kortex inspiriert [25] und so entworfen wurde, um die Hierarchie von relevanten Features eines Objektes (von low- zu high-level Features) aus Beispielbildern zu lernen.

Ein CNN besteht normalerweise aus drei Typen von Layer: Convolutional, Pooling und Fully-connected Layer. Convolutional und Pooling Layer sind für die Feature Extraktion zuständig. Die Fully-connected Layer gewichten die extrahierten Features und verbinden sie, beispielsweise bei einer Klassifikation, mit dem finalen Output [27]. Die extrahierten Features werden von einem Convolutional Layer zum Nächsten übergeben, wodurch, wie in der Abbildung 5 dargestellt, die Features progressiv und hierarchisch komplexer werden.

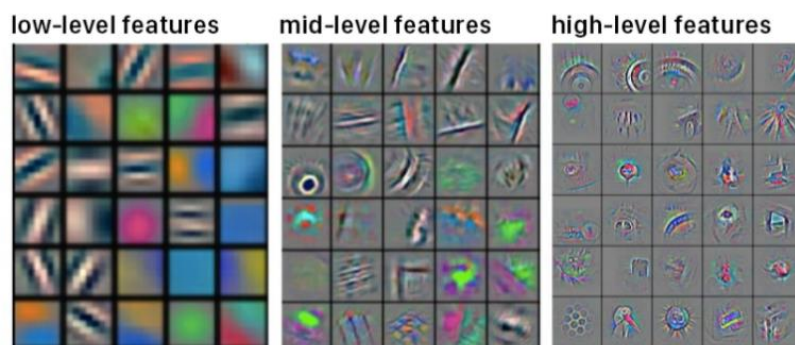


Abbildung 5: Low-, mid- und high-level Features [45]. Die Komplexität wird mit jedem Convolutional Layer erhöht.

Beim Training werden die Parameter des Netzwerks optimiert (bei den Convolutional Layer sind das die Werte in den Kernels), sodass der Output des Netzwerks sich möglichst wenig vom Ground Truth (die wahren Labels) unterscheidet.

### 2.4.1 Bausteine

#### Convolutional Layer

Im Convolutional Layer findet die Feature Extraktion statt, wobei ein Kernel, wie in Abbildung 6 zu sehen, über das gesamte Bild geschoben wird, und zuerst elementweise das Produkt berechnet und danach aufsummiert wird (diese Operation wird Faltung genannt). Die Summe der Faltung wird dann in einer sogenannten Feature Map abgespeichert [27].

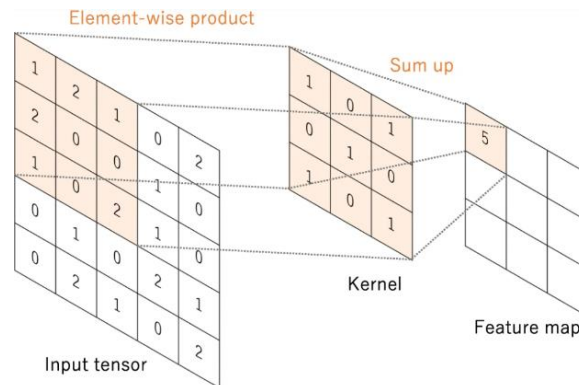


Abbildung 6: Faltung eines Bildes [27]. Der Kernel wird über den Input tensor geschoben und an jeder Stelle wird das Elementweise Produkt aufsummiert und in eine Feature Map geschrieben.

Dieser Schritt wird mit mehreren Kernels wiederholt, wodurch mehrere Feature Maps berechnet werden und somit die verschiedenen Charakteristiken des Input Bildes extrahiert werden. Die Feature Maps werden dann einer Aktivierungsfunktion übergeben. Hyperparameter sind im Convolutional Layer die Grösse (3x3, 5x5 etc.) und Anzahl der Kernels.

#### Pooling Layer

Im Pooling Layer werden die Dimensionen der Feature Maps reduziert, indem Bereiche der Features in kleineren Matrizen zusammengefasst werden. Damit müssen weniger Parameter beim nächsten Layer gelernt werden, was die Trainingszeit verkürzt [27]. Dies erfolgt mit einem Max-Pooling (nur der höchste Wert im Filter wird genommen) oder Average-Pooling (Durchschnitt im Filter). Ein Beispiel eines Max-Pooling wird in Abbildung 7 dargestellt. Im Pooling Layer gibt es keine lernbaren Parameter, sondern nur Hyperparameter wie Filtergrösse oder Stride (in welchen Abständen die Filter geschoben werden).

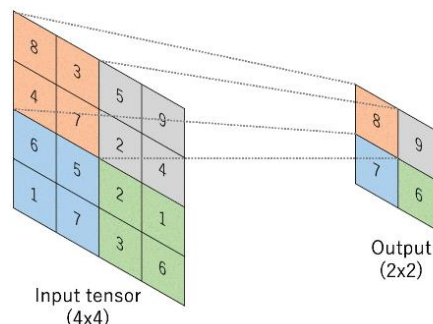


Abbildung 7: Max-Pooling mit einem 2x2 Filter und 2x2 Stride [27]. Der Maximale Wert aus dem 2x2 Bereich wird in den Output übernommen.

#### Dense / Fully-connected Layer

Die Feature Maps werden schlussendlich einem Dense Layer (auch Fully-connected Layer genannt) übergeben, wobei wie im traditionellen neuronalen Netz jeder Input mit jedem Output durch ein lernbares Gewicht verbunden ist. Die Outputs des letzten Dense Layer sind üblicherweise die Wahrscheinlichkeiten der verschiedenen Klassen. Hyperparameter sind bei Dense Layer nur die Anzahl Neuronen.



### 2.4.2 Anwendung in der Radiologie

Heutzutage werden Deep Learning Methoden in der Radiologie angewendet, um Bilddaten von Organen und Knochenstrukturen von Patienten zu analysieren [4], [27]. Diese Werke verfolgen ein ähnliches Ziel wie die vorliegende Arbeit: 2D-Bilder sollen automatisiert in verschiedene Klassen unterteilt werden. Eine erfolgreiche Anwendung wird von Setio et al. [28] vorgestellt, die ein CNN entwickelt haben, um aus CT-Scans zwischen Lungenknoten (nodules) und keine Lungenknoten (non-nodules), wie in Abbildung 8 ersichtlich, zu unterscheiden. In ihrer Arbeit werden die Features von Lungenknoten automatisch durch Trainingsdaten, welche aus den Datasets LIDC-IDRI [29], ANODE09 [30] und Danish Lung Cancer Screen Trial [31] stammen, und die im Rahmen ihrer Arbeit von Spezialisten annotiert wurden [28], gelernt.

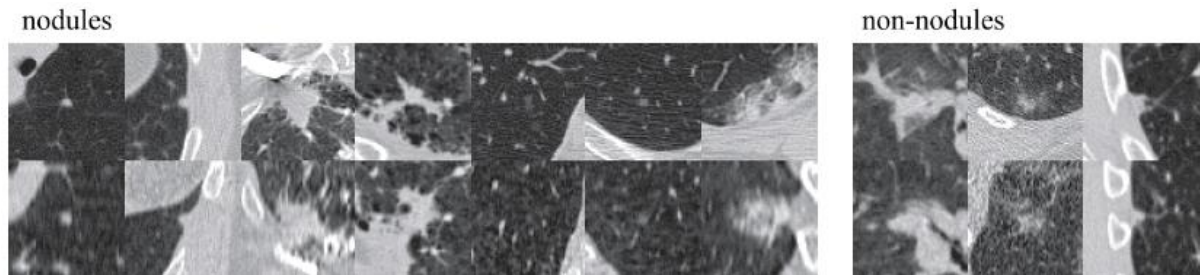


Abbildung 8: Beispiele von Lungenknoten (nodules) und Nicht-Lungenknoten (non-nodules) [28] aus den verwendeten Datasets.

Mit bestehenden CAD-Systemen werden aus den CT-Scans (3D-rekonstruierbar) die möglichen Kandidaten (mögliche Knoten) erkannt. Ziel ist es, gegenüber den bestehenden CAD-Systemen die False Positives von Knoten mit ihrem entwickelten CNN zu reduzieren. Die Kandidaten werden in kleinere 2-D (64x64) Bilder aus verschiedenen Orientierungen aufgeteilt (siehe Abbildung 9) und die Sequenz der 2-D Bilder dem Netzwerk übergeben, welches dann eine Klassifizierung vornimmt.

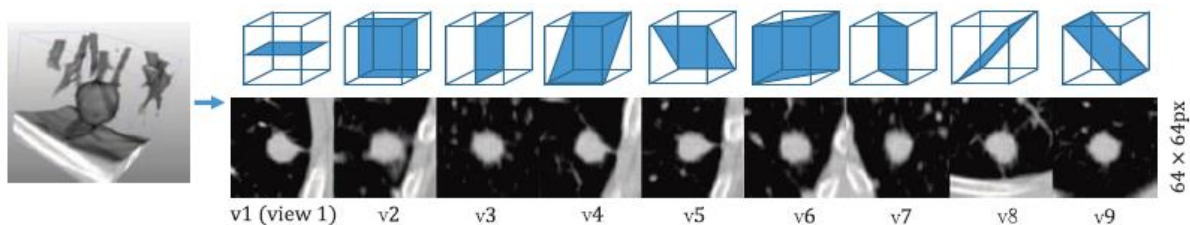


Abbildung 9: Sequenzieren des Kandidaten [28], der aus mehreren Perspektiven untersucht wird.

Da die Anzahl Bilder von Lungenknoten viel kleiner als die Anzahl Nicht-Lungenknoten sind, wurde Data Augmentation durchgeführt, um mehr Trainingsdaten zu generieren [28]. Mit einem Test-Set von 888 Scans erreichte ihr vorgestelltes System eine Detektionssensitivität (True Positive Rate) von 85.4 % [28] und zeigten daher, dass CNNs für die Erkennung von Lungenknoten geeignet sind.



## 3. Methoden

In den folgenden Abschnitten wird der Verlauf von Gewinnung der Daten bis zur Auswertung der trainierten Modelle beschrieben. Dabei werden die verwendeten Methoden genauer erklärt sowie die Massnahmen beschrieben, welche verwendet werden, um die Limitationen eines kleinen Datensets zu überwinden. Des Weiteren werden die gewählten Klassen vorgestellt und der Aufbau der Experimente erläutert. Für alle Module, die in dieser Arbeit geschrieben werden, wird die Programmiersprache Python verwendet, welches eine häufig verwendete Sprache in den Bereichen Datenwissenschaft und maschinelles Lernen ist.

### 3.1 Datengewinnung und Aufbereitung

Da Convolutional Neural Networks aus Beispielen lernen, müssen genug Daten vorhanden sein, sodass das Netzwerk die Features erlernen kann. Forschungen im Bereich Deep Learning nutzen üblicherweise ein bestehendes Dataset, wie es bei AlexNet [2] oder Setio et al. [28] der Fall ist.

Da im Rahmen dieser Arbeit ein System spezifisch für den bestehenden Shaker aus dem Labor angefertigt wird, kann kein bestehendes Dataset verwendet werden und es muss von Grund auf ein neues Dataset erstellt werden, mit welchem das CNN trainiert und getestet werden kann. Dazu gehört das Filmen und Bearbeiten der Videos, sowie das Annotieren der Frames. Während der ganzen Arbeit wird das Dataset gleichzeitig mit der Implementierung und der Evaluierung des CNN laufend ergänzt.

#### 3.1.1 Bearbeitung des Filmmaterials

Im Labor der ZHAW werden Videos mit der *GoPro Hero7* und zwei unterschiedlichen Beleuchtungen gefilmt. Als erste Beleuchtung wird die *Aputure Amaran AL-F7*, eine LED-Videoleuchte in der Grösse von 25 cm x 13 cm und eine Leistung von 15 W, verwendet. Als zweite Beleuchtung werden zwei Softboxen in der Grösse von 39 cm x 39 cm und Glühlampen mit der Leistung von 55 W eingesetzt. Durch unterschiedliche Beleuchtungen wird erhofft, dass das CNN robuster wird und besser auf verschiedene Lichtverhältnisse generalisiert. Bei den Aufnahmen wird der Shaker so konfiguriert, dass möglichst viele Tröpfchen entstehen. Auch wird die höchste Framerate der Kamera verwendet (120 FPS), sodass möglichst viele Bilder von den auftretenden Tröpfchen aufgenommen werden. Um Bilder von Schaumbildung zu erhalten, wird Seife in eine Phiole eingespritzt.

Beim Experimentieren mit dem Netzwerk wird festgestellt, dass einige Rahmenbedingungen beim Filmen eingehalten werden müssen, um ein möglichst gutes Ergebnis zu erzielen. Zum einen muss berücksichtigt werden, dass der Abstand zwischen der Kamera und dem Shaker ungefähr 20-25 cm beträgt, sodass das Motiv scharf abgebildet wird. Darüber hinaus muss die Beleuchtung so positioniert werden (idealerweise von oben), dass keine Reflexionen an der Glasoberfläche entstehen. Reflexionen können Fluid-Instabilitäten verdecken oder sind beim Betrachten eines einzelnen Frames mit kleinen Tröpfchen ununterscheidbar.

Da der Shaker immer in Bewegung ist, können die Phiolen nicht aus einer fixen Position ausgeschnitten werden. Dafür wird ein Landmark auf den Shaker geklebt und mit einem CSRT-Tracker verfolgt, um die horizontale und vertikale Bewegung des Shakers pro Frame zu erhalten. Mit der Bewegung kann der Shaker anschliessend zentriert und als neues Video abgespeichert werden. Danach werden die Phiolen als einzelne Videos ausgeschnitten. Hierbei muss beachtet werden, dass die Frames 180 Pixel breit und 140 Pixel hoch sein müssen, da das CNN eine

einheitliche Grösse verlangt. In Abbildung 10 sind die Bearbeitungsschritte des Rohmaterials zu sehen.



Abbildung 10: Schritte vom rohen Filmmaterial zu einer einzelnen Phiole. Links ist das rohe Video zusehen, in der Mitte der zentrierte Shaker und rechts eine ausgeschnittene Phiole.

### 3.1.2 Annotation

Die ausgeschnittenen Phiolen werden mit dem *MATLAB Video Labeler* manuell gemäss den folgenden Klassen annotiert:

1. *Still*: Ruhiger Zustand
2. *Wave*: Leichte Bewegung, noch keine starken Effekte
3. *NearDrops*: Flüssigkeit zieht sich in die Länge, Tropfen beginnen sich fast abzulösen
4. *SmallDrops*: Kleinere Tröpfchen
5. *Drops*: Klar sichtbare Tropfenbildung
6. *Foam*: Schaumbildung
7. *Useless*: Frames, die nicht eindeutig zu einem Label zugeordnet werden können, sowie starke Reflektionen

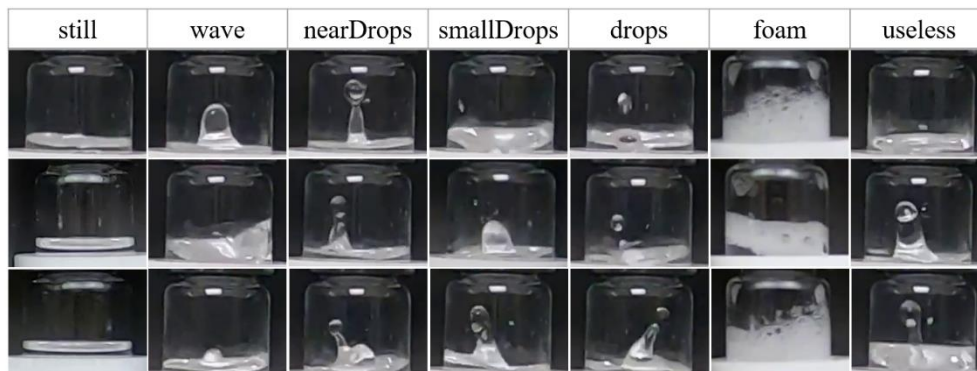


Abbildung 11: Beispielbilder aller Klassen

Die für das Labor relevanten Klassen sind *Still*, *Wave*, *Drops* und *Foam*. Zu Beginn wurde in Betracht gezogen, nur diese 4 Klassen zu definieren. Allerdings werden die anderen Klassen zusätzlich definiert, um die Möglichkeit zu haben, weitere Informationen zu gewinnen: *NearDrops* sind nichts anderes als *Waves* mit der zusätzlichen Information, dass bald Tröpfchen entstehen könnten. Die zusätzliche Unterscheidung erfordert bei der manuellen Annotation zwar minimal mehr Aufwand, liefert dafür aber wertvolle Information. Somit könnte beispielsweise im Video überprüft werden, ob bei klassifizierten *Drops* vorher *NearDrops* erkannt worden sind, was dem System eine höhere Zuversicht gibt, dass es sich bei erkannten Tröpfchen auch um True Positives handelt. *SmallDrops* und *Drops* sind prinzipiell beide als *Drops* zu sehen, nur werden die kleinen weniger gut erkennbaren Tröpfchen von den gut erkennbaren Tröpfchen getrennt. Der Grundgedanke ist hier, dass das Netzwerk für die Klasse *Drops* ausschliesslich aus klar erkennbaren Beispielen lernen soll, damit die gut erkennbaren Tröpfchen beim Klassifizieren auch mit einer höheren Sicherheit erkannt werden. Als weiteres

Argument für die Trennung gilt, dass die Klassen jederzeit zusammengefasst werden können und nur Informationen gewonnen werden.

Es gilt, dass für alle Frames, die nicht als *Useless* annotiert sind, nur eine Klasse vergeben wird. Die Klassen sind gemäss der Abbildung 11 von links nach rechts ihrer Stärke nach geordnet. Beispielsweise impliziert die Klasse *Drops*, dass auch schwächere Klassen wie kleine Tröpfchen und Wellen im Bild vorhanden sein können. Die Klasse *Useless* wird in Kombination mit einer anderen Klasse verwendet. Beispielsweise wird eine Spiegelung, die auf den ersten Blick wie *SmallDrops* aussieht, als *Useless* und *SmallDrops* annotiert. Die *Useless* Klasse erlaubt somit eine konditionelle Filterung der Daten, je nachdem wie sich solche Fälle auf das Training auswirken.

### 3.1.3 Dataset

Im Verlauf der Arbeit wurden insgesamt 19 Phiolen annotiert. Das Dataset beinhaltet per 25.05.2021 66'679 annotierte Frames mit der in Tabelle 1 ersichtlichen Aufteilung.

*Tabelle 1: Verteilung der annotierten Daten im erstellten Dataset*

<b>Still</b>	<b>Wave</b>	<b>NearDrops</b>	<b>SmallDrops</b>	<b>Drops</b>	<b>Foam</b>	<b>Useless</b>
5963	32504	1425	2614	1538	18811	3824

Zuerst wurden aus den Videos die einzelnen Frames extrahiert und lokal abgespeichert. Anhand des Index vom Frame und der Phiolen-Nummer wird das Frame mit dem korrekten Label aus der Annotation (als CSV-File) versehen.

## 3.2 Data Augmentation und Balancierung

In diesem Kapitel werden die Gründe für die Data Augmentation und die Arbeitsweise der Balancierung erläutert.

### 3.2.1 Augmentation

Klassifikatoren, die von unausgebalancierten Datasets lernen, können in der Praxis zu Problemen führen [32]. In solchen Fällen befindet sich die Mehrheit der Instanzen des Datasets in einer Klasse, während wenige Instanzen, üblicherweise die wichtigeren, in der anderen Klasse sind. Lernende Klassifikatoren tendieren dazu, alle Daten in die Mehrheitsklasse zu klassifizieren [32]. Die wichtige Klasse wird also häufig verpasst, was in Anbetracht der vorliegenden Arbeit ungeeignet ist. Beim vorgestellten CNN für Lungenknotendetektion von Setio et al. [28] wurden aufgrund dieses Problems die Lungenknotenbilder augmentiert. Data Augmentation ist eine Methode, mit der Datasets durch leicht veränderte Kopien von bestehenden Daten erweitert werden.

Wie in der Aufteilung des Datasets in Tabelle 1 ersichtlich, sind die Instanzen in dieser Arbeit ebenfalls ungleich verteilt, wobei es am meisten Frames mit dem Label *Wave* gibt. Grundsätzlich ist die Produktion von Bildern mit Tröpfchen schwierig. Bei Shaker Konfigurationen, wo versucht wird, möglichst viele Tröpfchen zu produzieren, ist die Flüssigkeit während der meisten Zeit nur in Bewegung, ohne dass sich Tröpfchen bilden. Dies führt dazu, dass die meisten Frames davon als *Wave* annotiert werden können und nur ein kleiner Bruchteil als *Drops*. Darüber hinaus werden nur grosse und gut erkennbare Tröpfchen (ohne Spiegelung) als *Drops* annotiert. Schaumbildung hingegen lässt sich leicht reproduzieren, da Seife direkt zu kontinuierlicher Schaumbildung im gesamten Video führt.

Um die Bilder zu augmentieren, werden diese einmal horizontal und einmal vertikal gespiegelt. Des Weiteren wird ein 3x3 Average Blur auf das Original-Bild und die gespiegelten Bilder angewendet. Der Effekt des 3x3 Average Filters wird in Abbildung 13 veranschaulicht. Ein Bild wird somit, wie in Abbildung 12 zu sehen, um fünf Bilder erweitert.

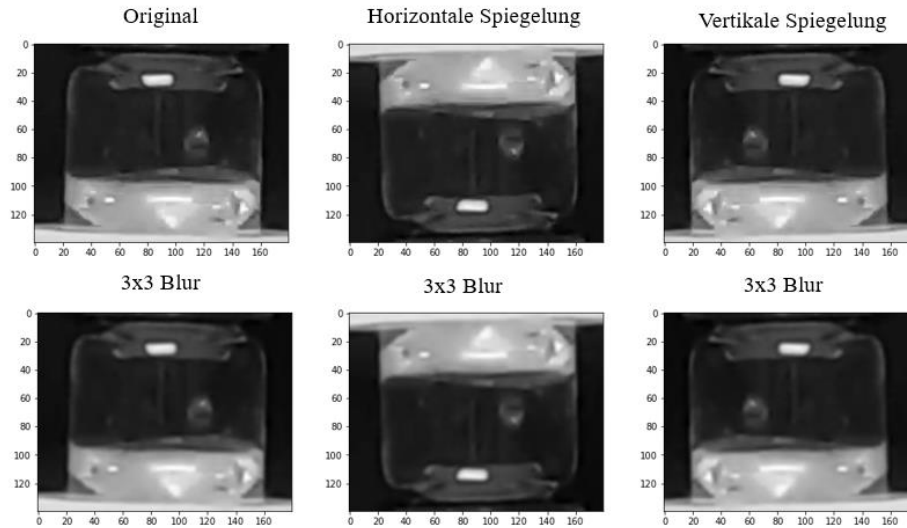


Abbildung 12: Beispiel eines vollständig augmentierten Bildes. Ein Originalbild wird durch Spiegelung und einen Average Filter um fünf Bilder erweitert.

Obwohl die Phiole in realen Experimenten nie verkehrt wie in der horizontalen Spiegelung (siehe Abbildung 12) zu sehen sind, wird hier die Annahme getroffen, dass das CNN die Features der relevanten Bereiche, in diesem Fall die Tröpfchen in der Mitte, erlernt und die

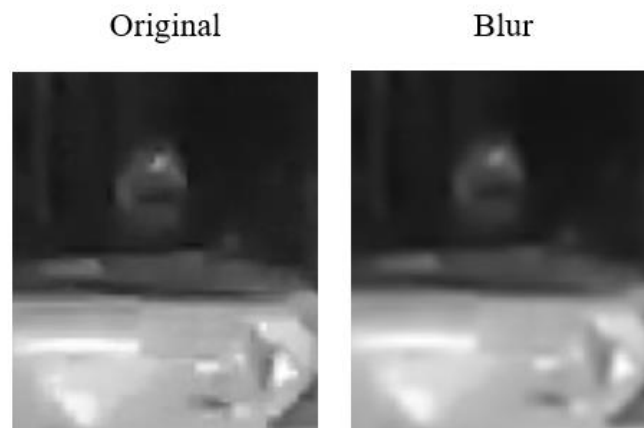


Abbildung 13: Vergrößerung eines Tröpfchens zur Veranschaulichung des 3x3 Average Filter

restlichen Bereiche vernachlässigt. Unter dieser Annahme generalisiert das CNN die Features der Tröpfchen und erkennt diese auch, wenn die Phiole wieder normal stehen.

### 3.2.2 Balancierung der Daten

Als weitere Massnahme um eine geeignete Verteilung der Instanzen für die Trainingsdaten zu finden, wird eine Funktion eingeführt, womit man die Verhältnisse der Instanzen der Klassen zueinander regulieren kann. Im Rahmen dieser Arbeit sollen verschiedene Verteilungen wie Gleichverteilung oder eine möglichst realitätstreue (vorwiegend *Waves*) Verteilung untersucht werden.

Die Balancierungsfunktion benötigt als Input für jede Klasse eine Gewichtung, wobei das Gewicht 1 die Anzahl Instanzen von der Klasse mit der tiefsten Anzahl bedeutet. Wird ein höheres Gewicht gesetzt als die Klasse Instanzen verfügt, werden einfach alle vorhandenen Instanzen genommen.

#### **Beispiel:**

Es existiert das in Tabelle 2 erfundene Dataset, wobei *NearDrops* die Klasse mit der niedrigsten Anzahl Instanzen ist:

*Tabelle 2: Beispielverteilung eines fiktiven Datasets*

Still	Wave	NearDrops	SmallDrops	Drops	Foam
50	100	<b>15</b>	25	20	90

Für die Balancierung wird nun die in Tabelle 3 definierte Gewichtung verwendet:

*Tabelle 3: Beispielgewichtung für das fiktive Dataset*

Still	Wave	NearDrops	SmallDrops	Drops	Foam
1	4	1	2	2	5

Nach der Balancierung besteht das in Tabelle 4 ersichtliche Dataset:

*Tabelle 4: Erhaltenes Dataset nach Balancierung*

Still	Wave	NearDrops	SmallDrops	Drops	Foam
15	60	15	25	20	75

### 3.3 Aufbau und Training des Convolutional Neural Network (CNN)

Um ein Convolutional Neural Network zu bauen, gibt es mehrere Frameworks für verschiedene Sprachen. Für Python sind *PyTorch*, *Keras* und *Tensorflow* sehr häufig verwendete Frameworks, wobei *Keras* auf *Tensorflow* aufbaut und die Verwendung vereinfacht. Aufgrund der Einfachheit, einzelne Schichten anzupassen und neue hinzuzufügen bzw. zu entfernen, wurde entschieden, *Keras* in dieser Arbeit zu verwenden.

#### 3.3.1 Erste Schritte

Das erste Modell, welches in Abbildung 14 zu sehen ist, wird anhand der LeNet Architektur [26] gebaut:

- Drei Convolutional Layer (Conv2D) mit jeweils 3x3 Filter. Im ersten Layer werden 32 Filter, im zweiten 64 und im dritten Layer 128 eingesetzt.
- Nach jedem Convolutional Layer wird ein Max-Pooling Layer (MaxPooling2D) mit einem 2x2 Kernel verwendet, um die Feature Map in den Dimensionen jeweils zu halbieren.
- Für die Klassifizierung wird ein Dense Layer mit 128 Neuronen und ein Output Layer mit Softmax-Aktivierungsfunktion verwendet. Beim Output Layer hat es so viele Neuronen, wie es zu unterscheidende Klassen gibt. Softmax sorgt dafür, dass für jede Klasse eine Wahrscheinlichkeit ausgegeben wird und die Summe der Wahrscheinlichkeiten genau eins ergibt.
- Nach jedem Convolutional Layer und dem Dense Layer wird zusätzlich die LeakyReLU-Aktivierungsfunktion verwendet, um das Problem von verschwindenden Gradienten zu mildern und dem Netzwerk zu helfen, nicht-lineare Grenzen zu lernen [33].
- Um Overfitting zu vermeiden, wird zudem Dropout in das Netzwerk eingebaut. Dropout führt dazu, dass bei jedem Lernschritt ein prozentualer Anteil zufälliger Neuronen in einer Schicht ausgeschaltet werden, wodurch die übrigen Neuronen jeweils besser generalisieren müssen.
- Als Verlustfunktion wurde Categorical Crossentropy (CCE). CCE ist die bevorzugte Verlustfunktion bei Klassifikationsproblemen mit mehreren Klassen (mehr als zwei) [34].

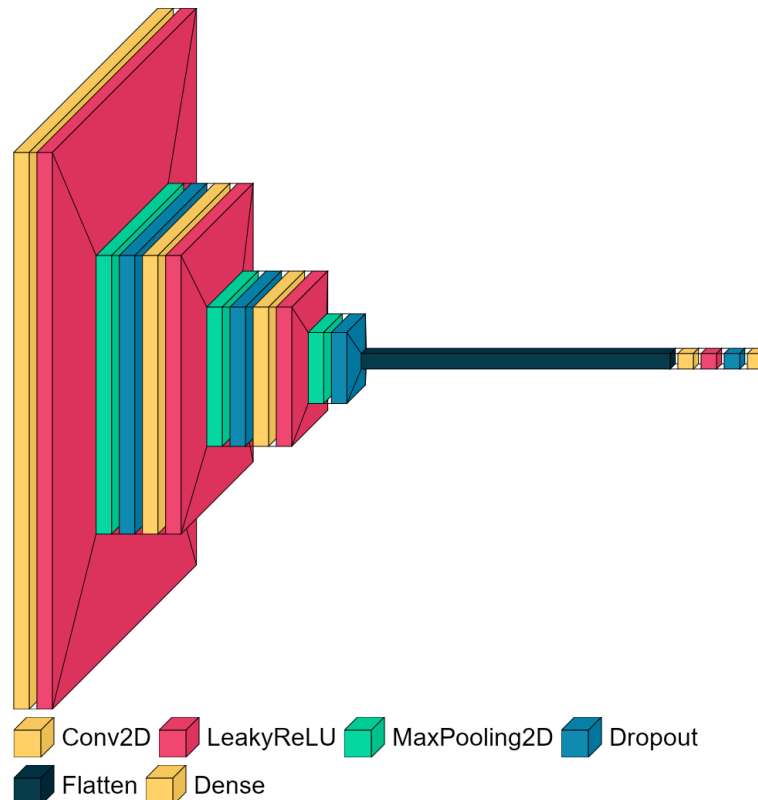


Abbildung 14: Architektur des ersten CNN. Die ersten drei gelben Layer sind Convolutional Layer und die letzten beiden sind Dense Layer. Der letzte Layer ist der Output Layer.

### 3.3.2 Verbesserungen

Von dem ersten Netzwerk aus können nun iterativ Änderungen gemacht und jeweils ein neues Modell trainiert werden. Dabei werden die Hyperparameter einzeln verändert und die Auswirkung auf den Trainingsverlauf und die Genauigkeit untersucht.

Das Netzwerk hat viele Hyperparameter, weshalb es praktisch unendlich viele Kombinationen gibt, die ausprobiert werden könnten. Aus Zeitgründen wird nur eine beschränkte Anzahl Variationen der Hyperparameter untersucht. Zu den Hyperparametern und Algorithmen, die bei den Versuchen geändert werden, zählen:

- Filtergröße von den Convolutional Layern
- Anzahl Filter pro Convolutional Layer
- Batchsize
- Anzahl Neuronen pro Dense Layer

Zusätzlich werden Änderungen an der Architektur wie mehr Dense und/oder mehr Convolutional Layer untersucht, sowie die Auswirkung von Data Augmentation und Balancierung. Der Output Layer wird nicht verändert, weshalb er fortan nicht mehr erwähnt wird.

### 3.3.3 Training

Um das CNN zu trainieren, werden mehrere Schritte durchgeführt. Zuerst werden die annotierten Daten geladen und dann, wie in Kapitel 3.2 beschrieben, balanciert und augmentiert. Damit die Daten dem CNN übergeben werden können, müssen sie noch vorverarbeitet werden. Dieser Schritt wird als Preprocessing bezeichnet.

## Preprocessing

Beim Preprocessing werden die Daten in ein Format gebracht, welches das CNN verarbeiten kann. Die Bilder bestehen zu Beginn aus einem dreidimensionalen Array: [Höhe, Breite, Farbkanäle]. Da Graustufenbilder verwendet werden, müssen die Bilder vom RGB Farbraum in ein Graustufenbild konvertiert und die Farbwerte normalisiert werden. Danach werden alle Bilder in eine Liste gespeichert, welche dann in ein vierdimensionales Array mit folgendem Aufbau umgewandelt wird: [Bildindex, Höhe, Breite, 1]. Die Labels müssen ebenfalls in eine für das Netzwerk verständliche Form gebracht werden. Dafür werden die Namen der Labels zuerst in Integer umgewandelt und dann in eine One-Hot-Kodierung konvertiert. Die One-Hot-Kodierung hat anstelle von einem Integer-Wert als Label einen Zeilenvektor mit lauter Nullen ausser dem Index, welcher dem ursprünglichen Label entspricht. Beispielsweise entspricht das Label «Drops» dem Integer-Wert 4, welcher wiederum in der One-Hot-Kodierung einem Zeilenvektor  $[0, 0, 0, 0, 1, 0]$  entspricht. Abbildung 15 zeigt die einzelnen Schritte in einem konzeptionellen Diagramm auf.

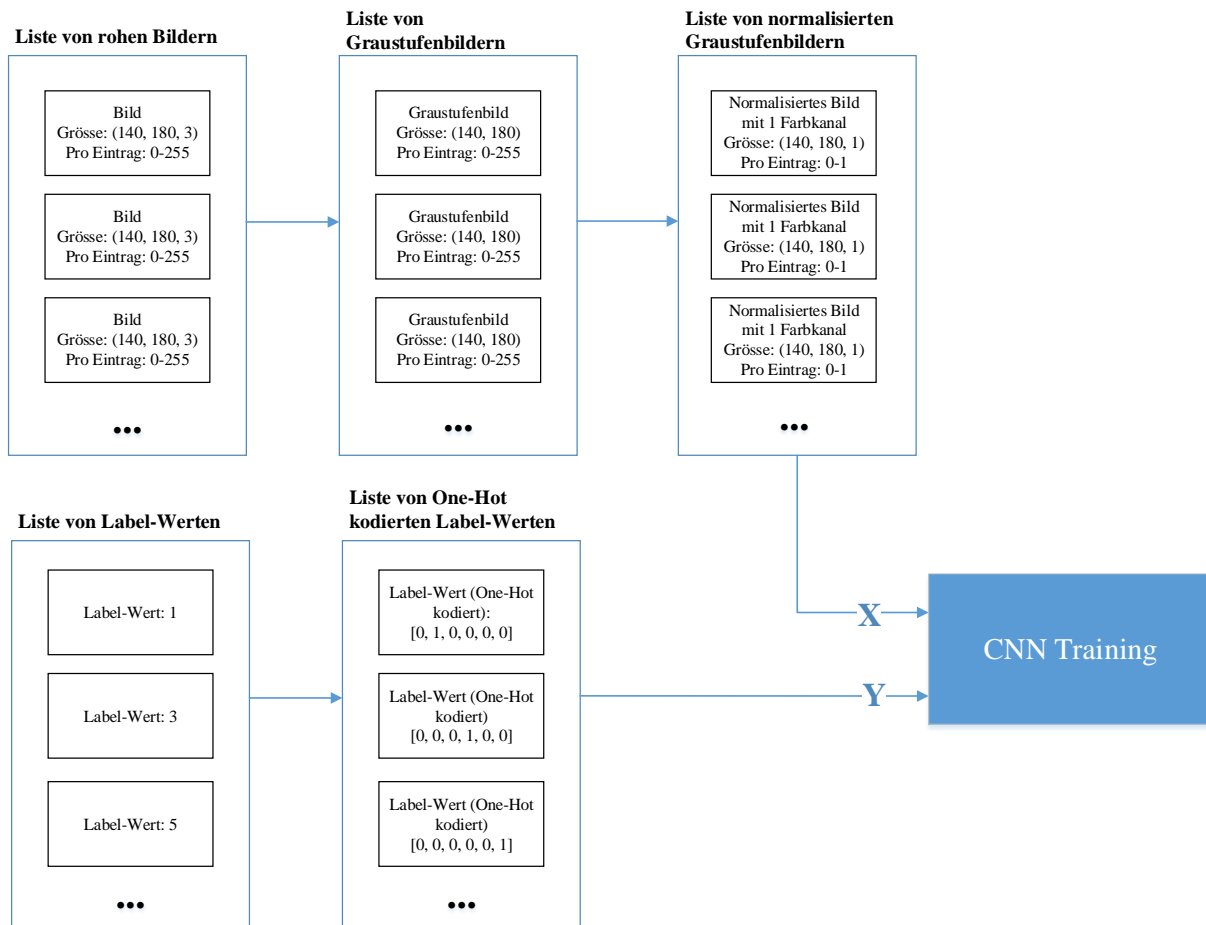


Abbildung 15: Konzeptionelles Diagramm des Preprocessing. Es wird die Verarbeitung der Daten von den rohen Bildern zum verwendbaren Input X sowie die Verarbeitung der Annotationen zur One-Hot-Kodierung beschrieben.



### Test-Validierungs-Split und Epochen

Sind die Daten vorverarbeitet worden, werden diese in Trainings- und Validierungsset, sowie noch in ein Testset, wie in Abbildung 16 dargestellt, aufgeteilt. Für die Experimente werden die Daten wie folgt aufgeteilt: 70 % als Trainings-, 20 % als Validierungs- und 10 % als Testdaten. Die Test- und Validierungsdaten werden dann dem Netzwerk zum Training übergeben. Das Modell wird folgend in mehreren Epochen (in dieser Arbeit maximal 30) trainiert. In jeder Epoche werden alle Trainingsdaten einmal in mehreren Trainingsschritten abgearbeitet. In jedem Trainingsschritt werden zufällige Samples in der mit der Batchsize spezifizierten Menge aus dem Trainingsset genommen und damit das CNN trainiert.

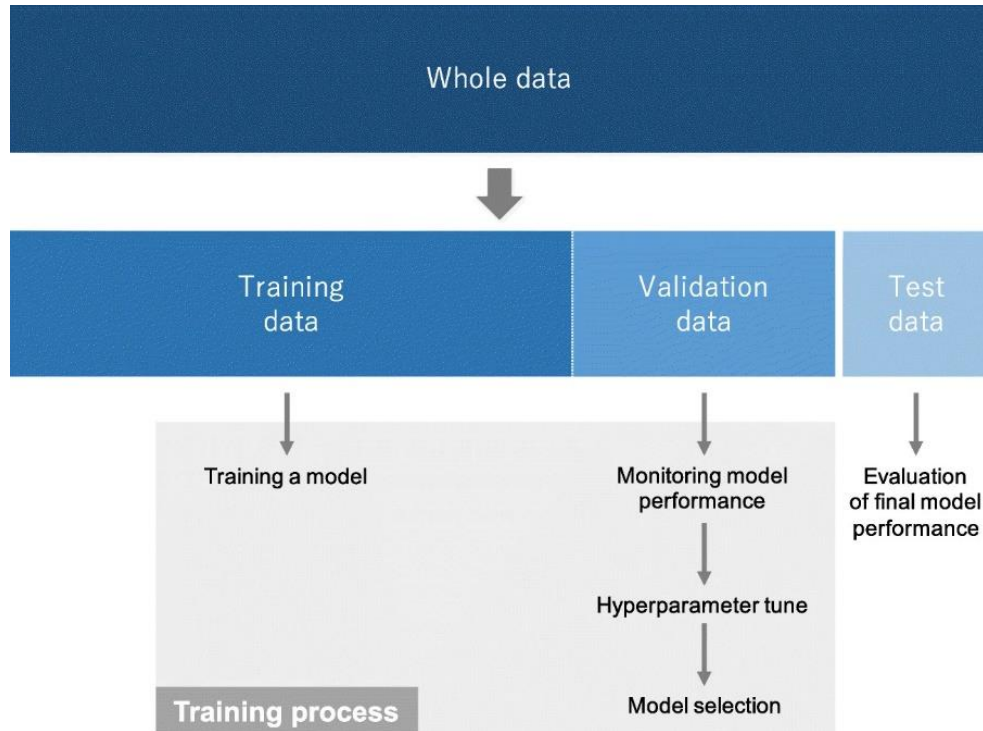


Abbildung 16: Split des Dataset in Trainings-, Validations- und Testdaten [27]

### Frühzeitiges Abbrechen, um Overfitting zu vermeiden

Während des Trainings ist aufgefallen, dass der Validierungs-Loss gegen Ende des Trainings wieder ansteigt. Dies kann darauf zurückgeführt werden, dass das Modell nach dem Erreichen des besten Loss (bzw. der besten Gewichtungen) beginnt, sich zu sehr auf die Trainingsdaten zu spezialisieren [35]. Um dies zu vermeiden, wurde ein Callback erstellt, welcher das Training abbricht, sobald der Validation-Loss beginnt zu inkrementieren und in den nächsten 5 Epochen nicht wieder unter den Bestwert fällt. Der Callback stellt nach dem Abbruch die beste Gewichtung des Modells wieder her. Das Phänomen von Overfitting wird in Abbildung 17 anhand des Verlaufs der Loss-Funktion bei einem Training dargestellt. Overfitting ist dadurch erkennbar, dass die Loss-Funktion bei den Trainingsdaten sehr klein wird, für die Validierungsdaten allerdings nicht.

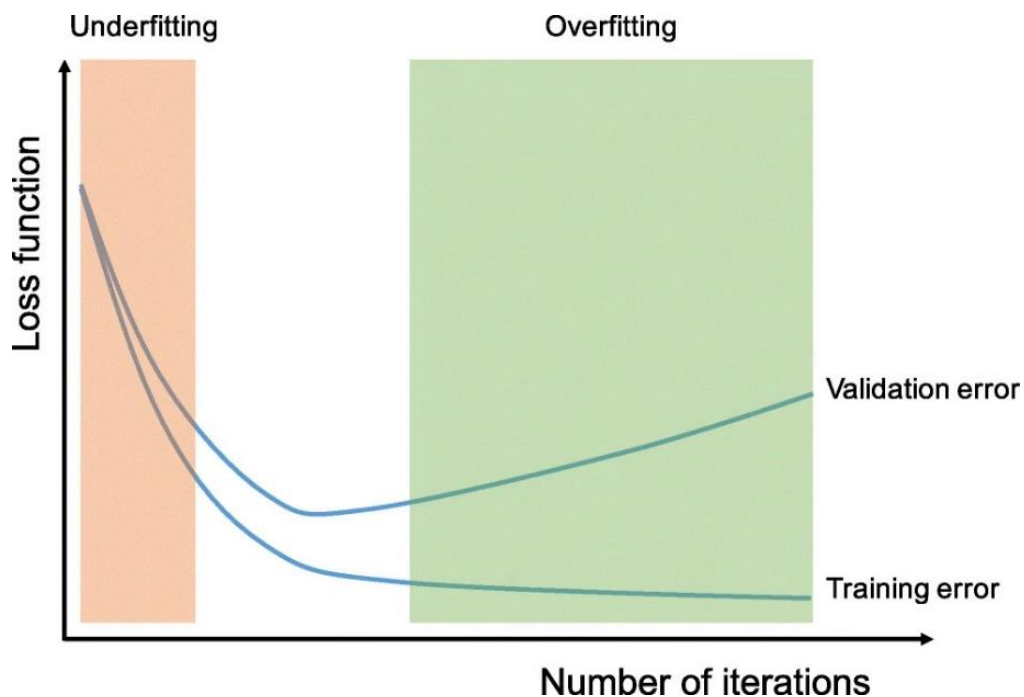


Abbildung 17: Beispiel der Loss Function bei einem Training, bei dem es zu Overfitting kommt. Zu sehen ist dies am Validation Error, der wieder ansteigt und vom Training Error divergiert [27].

### 3.4 Evaluierung der Hyperparameter und Modelle

Im Rahmen dieser Arbeit werden mehrere Modelle trainiert mit dem Ziel, eine passende Architektur und Hyperparameter zu finden. Damit die Zeit für das Training möglichst gering bleibt, wird das Training auf GPUs durchgeführt.

#### 3.4.1 Versuchsaufbau

Beim Trainieren des CNN kommen viele zufällige Prozesse vor. So werden die Gewichtungen zu Beginn des Trainings mit zufälligen Werten initialisiert und die Samples für die Trainingsschritte in einer zufälligen Reihenfolge gewählt. Damit eine gute Bewertung der erhaltenen Modelle vorgenommen werden kann, sollten die Ergebnisse bzw. der Trainingsverlauf reproduzierbar sein. Um dies zu erreichen, werden sämtliche verwendete Random Number Generators (RNG) mit einem Seed initialisiert. Somit wird sichergestellt, dass alle zufälligen Prozesse immer dieselben Sequenzen von zufälligen Nummern erhalten.

Das Einzige, was den Ausgang des Trainings nun noch beeinflusst, sind die gewählte Architektur und die gewählten Hyperparameter. Die trainierten Modelle können somit anhand verschiedener Aspekte bewertet werden:

- Während des Trainings ist der Verlauf von Loss, Accuracy und Recall interessant, da versucht wird, diese Werte zu optimieren. Dabei ist auch zu sehen, ob es zu Overfitting kommt.
- Nach dem Training wird das Modell auf die Testdaten, welche nicht für das Training verwendet wurden, angewendet. Somit lässt sich die Genauigkeit des Modells auf ungesesehenen Daten bestimmen. Die dazu verwendeten Metriken werden im Kapitel 3.4.2 genauer erläutert.
- Es wird ein neues Video analysiert, welches nicht annotiert wurde und somit nicht zum Datensatz gehört. Damit kann ein reales Szenario approximiert werden. Von dem analysierten Video werden Stichproben nach einem definierten Protokoll (siehe Kapitel 3.4.3) genommen.

Um die erhaltenen Modelle gut unterscheiden zu können, werden die beim Training verwendeten Parameter in den Dateinamen geschrieben. Der Standarddateiname ist wie folgt aufgebaut:

```
model_{kernelSize}f_{c1Kernels}k_{c2Kernels}k_{c3Kernels}k_{d1Nodes}d_{epochs}ep_{batchSize}bs_{augmentationType}_{stillWeight}_{wavesWeight}_{nearDropsWeight}_{smallDropsWeight}_{dropsWeight}_{foamWeights}.h5
```

Als Beispiel ist der Dateiname des ersten Modells, welches in Kapitel 3.3.1 beschrieben ist, folgender:

```
model_3x3f_32k_64k_128k_128d_10ep_64bs_fullaug_10s_10w_10nd_10sd_10d_10f.h5
```

Gibt es weitere geänderte Einstellungen wie eine andere Architektur oder verringerte Data Augmentation, werden diese Infos ebenfalls in den Dateinamen geschrieben.

### 3.4.2 Definition der Metriken für Testdaten

Das fertig trainierte Modell wird auf die vor dem Training definierten Testdaten angewendet. Dabei wird durch das Modell für jeden Frame im Testdatensatz eine Vorhersage  $\tilde{y}$  getroffen. Diese Vorhersage kann dann mit dem Ground Truth Label  $y$  verglichen werden. Es wird zwischen vier Fällen unterschieden:

- True Positive (TP)
- False Positive (FP)
- True Negative (TN)
- False Negative (FN)

Tabelle 5 zeigt eine Konfusionsmatrix, die beschreibt, wie Ground Truth  $y$  und Vorhersage  $\tilde{y}$  im jeweiligen Fall in Beziehung stehen. Dabei ist folgendes zu beachten:

- «Ground Truth  $y$ » sagt für einen bestimmten  $y$  Wert z. B. *Drops*, ob es sich bei dem gegebenen Bild tatsächlich um ein Bild mit dem Label *Drops* handelt.
- «Vorhergesagtes  $\tilde{y}$ » gibt an, ob das Netzwerk für das gegebene Bild die Vorhersage *Drops* getroffen hat.

Tabelle 5: Beziehung von vorhergesagtem Output zu wahren Output

Confusion-Matrix		Vorhersage $\tilde{y}$	
		Negative	Positive
Ground Truth $y$	Negative	TN	FP
	Positive	FN	TP

In Abbildung 18 wird noch eine Konfusionsmatrix mit allen definierten Klassen aufgezeigt. Farblich markiert sind die True Negatives, True Positives, False Negatives und False Positives für die Klasse *Drops*.

		Prediction					
		still	wave	nearDrops	smallDrops	drops	foam
Ground truth	still	231	0	0	0	0	0
	wave	0	322	1	3	1	0
	nearDrops	0	4	174	3	3	0
	smallDrops	0	4	2	117	5	0
	drops	0	2	4	8	110	0
	foam	0	0	0	0	0	184

TN	true negative
TP	true positive
FN	false negative
FP	false positive

Abbildung 18: Beispiel einer Konfusionsmatrix für ein Multi-Label Klassifizierungsproblem. Das Beispiel zeigt die Bereiche, die für die Klasse Drop als TP, FP, TN und FN gelten.

Mithilfe der Konfusionsmatrix lassen sich Metriken berechnen, welche die Genauigkeit des Modells auf den Testdaten beschreibt. Verwendete Metriken sind Accuracy, Precision, Recall (True Positive Rate), False Positive Rate, und False Negative Rate [14]:

- $Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$
- $Precision = \frac{TP}{TP+FP}$
- $Recall = \frac{TP}{TP+FN}$
- $False\ Positive\ Rate = \frac{FP}{FP+TN}$
- $False\ Negative\ Rate = \frac{FN}{TP+FN}$

Um zusätzlich einen Messwert für die Gesamtleistung zu bekommen, wird noch der FBeta-Score berechnet:

$$FBeta - Score = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \quad (2)$$

Das Beta ( $\beta$ ) ist eine Gewichtung, um zu bestimmen, ob Recall oder Precision mehr Einfluss auf den Score hat. Bei  $\beta = 1$  ist die Gewichtung ausgeglichen. Bei  $\beta > 1$  wird Recall mehr gewichtet als Precision und bei  $\beta < 1$  umgekehrt. Im Falle dieses Projekts sind Recall und False Negative Rate zwei sehr wichtige Metriken. Nicht erkannte Tröpfchen sind ungünstiger als ein paar Wellen, die fälschlicherweise als Tröpfchen vorhergesagt werden. In einem Video nach False Negatives zu suchen, dauert länger als ein paar False Positives zu überprüfen. Der Grund dafür ist, dass die informationsreichere Klasse (in diesem Fall Tröpfchen) seltener vorkommt. Folgend wird ein Beta von 2.0 verwendet, um Recall stärker zu gewichten.

Die Metriken werden jeweils für jede Klasse berechnet und danach der Durchschnitt berechnet. Somit kann die durchschnittliche Performanz auf allen Klassen dargestellt und verglichen werden.

### 3.4.3 Neue Videos analysieren und protokollieren

Das CNN soll zusätzlich in ein User-Interface integriert werden, sodass frei wählbare Videos klassifiziert werden können. Nebst einer guten Performanz auf dem Dataset soll das Modell auch in der realen Anwendung gut abschneiden. Das Interface erlaubt es, trainierte Modelle mit beliebigen Videos, die das Netzwerk noch nie zuvor gesehen hat, zu testen und zu evaluieren.

Im Interface (siehe Abbildung 19) werden folgende Funktionen eingebaut:

1. Laden eines gewünschten Videos und Start der automatischen Analyse
2. Speichern und Wiederladen von Analysen als CSV-File
3. Navigierung im Video
4. Übersicht der Verteilung aller Klassen aus der Analyse
5. Für jedes Frame wird die vorhergesagte Klasse angezeigt, womit man auch navigieren kann

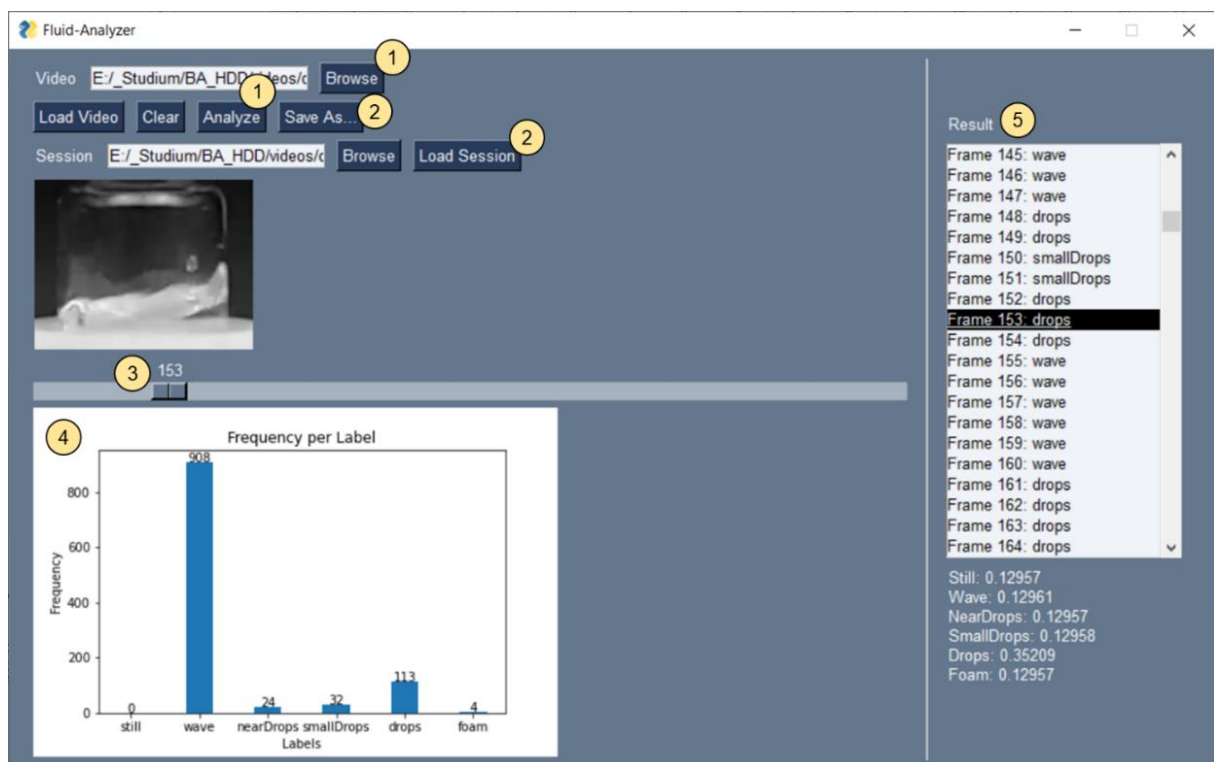


Abbildung 19: GUI zur Auswertung von beliebigen Videos.

Um verschiedene Modelle zu evaluieren und zu vergleichen, wird folgendes Protokoll durchgeführt:

**Protokoll für Tröpfchen (*Drops*) und Wellen (*Waves*):**

1. Zu evaluierendes Modell einstellen.
2. Testvideo laden und Analyse starten.
3. Verteilung überprüfen (stimmt diese annähernd überein mit dem Video).
4. Zufällig 50 *Drops* aus den *Results* auswählen und im Bild kontrollieren, ob es sich um einen True-Positive Fall handelt. Anzahl True-Positives zählen.
5. Zufällig 50 *Waves* aus den *Results* auswählen und im Bild kontrollieren, ob es sich um einen True-Positive Fall handelt. Anzahl True-Positives zählen.
6. Speichern der Analyse als CSV.
7. Wiederholen mit anderen Modellen.

**Protokoll für Schaumbildung (*Foam*) und Ruhezustand (*Still*):**

Bei Schaumbildung werden idealerweise Test-Videos verwendet, die während des ganzen Videos Schaum haben. Daher muss nur die Übersicht der Verteilung verglichen werden, um zu schauen, ob das Modell richtig klassifiziert hat. Für die Klasse *Still* werden am besten Videos verwendet, wo die Oberfläche während der meisten Zeit flach bleibt.

## 4. Resultate

Um die besten Hyperparameter und die beste Architektur zu bestimmen, wurden von der Grundarchitektur aus (siehe Kapitel 3.3.1) iterativ Änderungen vorgenommen und jeweils ein neues Modell trainiert. Die trainierten Modelle werden basierend auf den in Kapitel 3.4.1 beschriebenen Aspekten bewertet. Ebenfalls wird der Einfluss von Augmentation und Balancierung untersucht. Die berechneten Metriken (siehe Kapitel 3.4.2) werden über alle Klassen gemittelt und auf vier Stellen gerundet, um einen quantitativen Vergleich vorzunehmen.

### 4.1 Augmentation

Es wurde bereits in einigen Arbeiten [4], [5], [19], [28] gezeigt, dass durch die Vervielfältigung des Dataset mittels Augmentation eine Verbesserung in der Performanz des Modells erreicht werden kann. Um herauszufinden, wie sich die implementierte Data Augmentation (siehe Kapitel 3.2.1) auf die Performanz für die Klassifizierung von Fluid-Instabilitäten auswirkt, wurden Modelle mit und ohne Data Augmentation trainiert und verglichen:

- **No Augmentation:** Das erste Modell wird ohne Data Augmentation trainiert.
- **Flip + Original Blur:** Beim zweiten Modell wird eine leichte Data Augmentation eingesetzt, wobei das Originalbild horizontal und vertikal gespiegelt wird und ein Blur auf das Originalbild eingesetzt wird. Damit wird ein Bild um 3 Bilder erweitert.
- **Full Augmentation:** Das letzte Modell verwendet die volle Data Augmentation und wendet den Blur noch auf die gespiegelten Bilder an. Damit wird ein Bild um 5 Bilder erweitert.

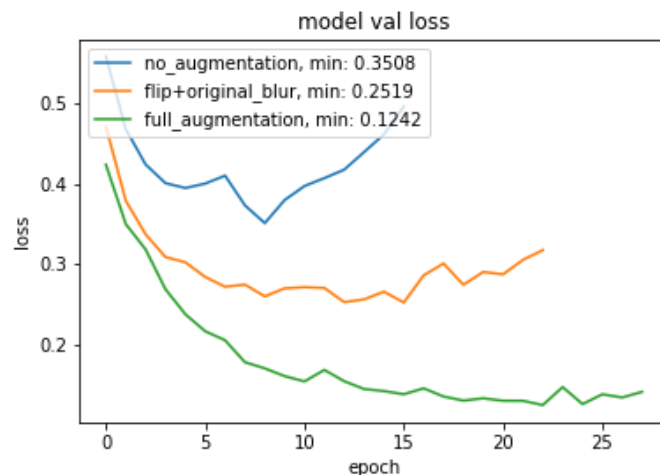


Abbildung 20: Verlauf des Validation Loss beim Training von Modellen mit und ohne Data Augmentation. Die volle Data Augmentation erreicht den tiefsten Loss.

Anhand des Validation Loss in Abbildung 20 ist erkennbar, dass ein Training mit Data Augmentation den tiefsten Validation Loss erreicht und die anderen Varianten im Training bereits früher abbrechen. In Abbildung 21 ist ersichtlich, dass die volle Data Augmentation ebenfalls die höchsten Werte bei Recall und Accuracy erreicht.



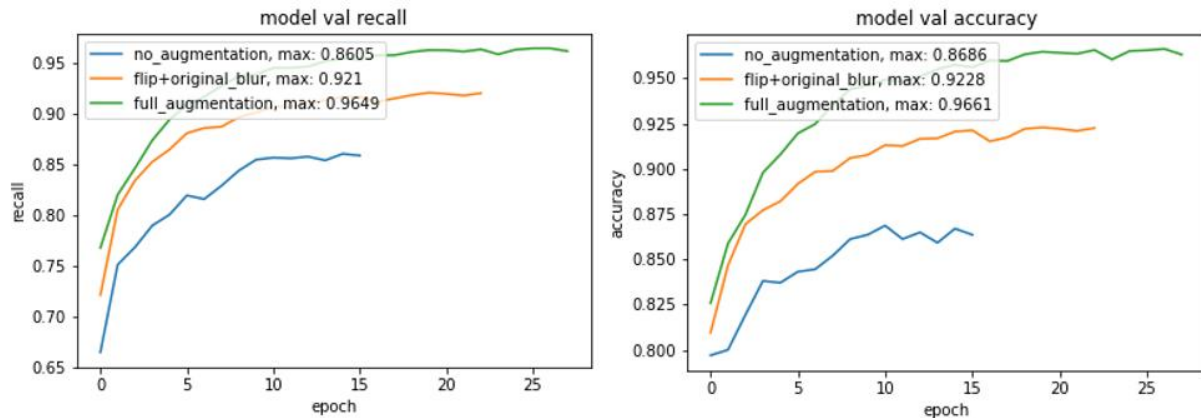


Abbildung 21: Verlauf von Recall und Accuracy beim Training von Modellen mit und ohne Data Augmentation. Das Modell mit voller Data Augmentation erreicht die besten Werte für Recall und Accuracy.

In Tabelle 6 ist zu sehen, dass die volle Data Augmentation den höchsten FBeta-Score erreicht und deutlich besser abschneidet als keine Data Augmentation. Wichtig zu erwähnen ist, dass der Score nicht viel höher ist als bei der leichten Data Augmentation. Es wird angenommen, dass dies daran liegt, dass das CNN aus den gespiegelten Bildern schon die richtigen Kernels lernt und zusätzlicher Blur auf den gespiegelten Bildern nicht dazu führt, dass die Kernels besser werden.

Tabelle 6: Vergleich der Metriken für die trainierten Modelle mit und ohne Data Augmentation.

Modell	Precision	Recall	False Pos Rate	False Neg Rate	Accuracy	FBeta-Score
No_augmentation	0.6047	0.6193	0.0767	0.3807	0.8659	0.5957
Flip+original_blur	0.9582	0.9480	0.0076	0.0520	0.9880	0.9499
Full_augmentation	0.9563	0.9502	0.0073	0.0498	0.9883	<b>0.9514</b>

## 4.2 Balancierung

In dieser Arbeit ist ein Dataset mit insgesamt 66'679 annotierten Frames erstellt worden, jedoch ist dieses unausgeglichen (siehe Kapitel 3.1.3). Unausgeglichene Datasets können zu Problemen führen [32], indem die Klasse mit mehr Instanzen beim Training einen stärkeren Einfluss auf Loss und Accuracy hat. Dadurch werden die Gewichtungen während dem Training zugunsten der am meisten vorhandenen Klasse angepasst. Um dies zu verhindern, wurde in Kapitel 3.2.2 eine Funktion vorgestellt, um mittels Gewichtungen die Verteilung beim Training anzupassen. Um zu erfassen, wie sich die Balancierung auf die Ergebnisse auswirkt, wird mit drei Verteilungen experimentiert:

- **Gleichverteilt**, daher überall eine Gewichtung von eins.
- **Balanciert**, indem eine geschätzte realitätsgetreue Verteilung leicht approximiert wird. Die Gewichtungen sind wie folgt:

Tabelle 7: Gewichtung für die Balancierung des Dataset

Gewichtung					
Still	Wave	NearDrops	SmallDrops	Drops	Foam
3.0	4.0	1.0	1.5	1.2	1.0

- **Unausgeglichen** wie im Dataset.

Auf den ersten Blick scheint es in Abbildung 22 und Abbildung 23 so, als würde die unausgeglichene Verteilung die besten Ergebnisse erzielen.

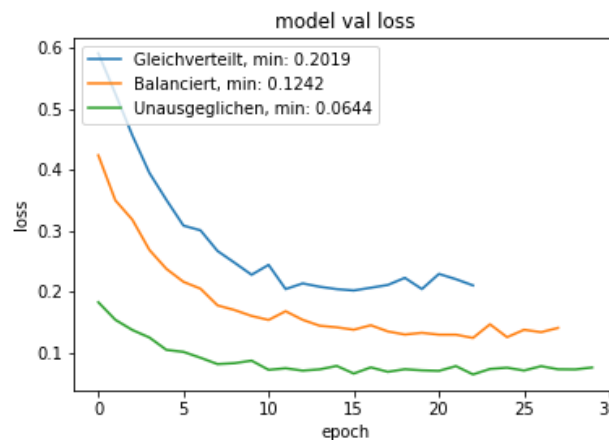


Abbildung 22: Verlauf von Validation Loss beim Training von Modellen mit unterschiedlichen Verteilungen.

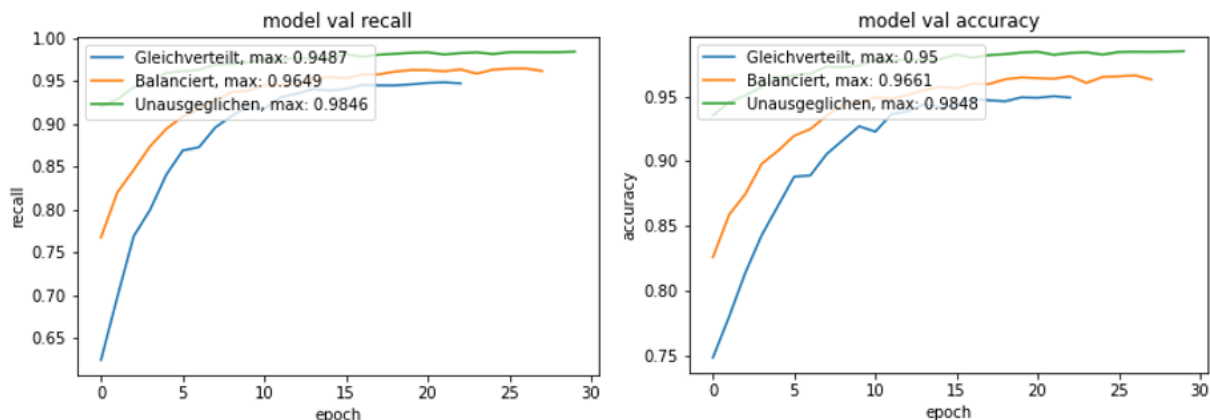


Abbildung 23: Verlauf von Recall und Accuracy beim Training von Modellen mit unterschiedlichen Verteilungen.

Jedoch ist aus den Ergebnissen auf dem Testset (siehe Tabelle 8) ersichtlich, dass die balancierte Verteilung am besten abschneidet. Es ist hier gut zu sehen, dass beim unausgeglichene Fall Validation Loss, Recall und Accuracy stark durch die primär auftretenden Klassen beeinflusst werden. In der erhaltenen Konfusionsmatrix (siehe Tabelle 9) ist erkennbar, dass das Modell dazu tendiert, einen grossen Teil der Daten als *Wave* zu klassifizieren, welches die meist vorhandene Klasse ist.

*Tabelle 8: Vergleich der Metriken für die trainierten Modellen mit unterschiedlichen Verteilungen.*

Modell	Precision	Recall	False Pos Rate	False Neg Rate	Accuracy	FBeta-Score
Gleichverteilt	0.9056	0.9332	0.0145	0.0668	0.9754	0.9264
Balanciert	0.9563	0.9502	0.0073	0.0498	0.9883	<b>0.9514</b>
Unausgeglichene	0.8871	0.8308	0.0242	0.1692	0.9643	0.8386

*Tabelle 9: Konfusionsmatrix des Modells mit einem unausgeglichene Dataset.*

	Still	Wave	NearDrops	SmallDrops	Drops	Foam
Still	<b>2518</b>	0	0	0	0	0
Wave	0	<b>3356</b>	0	2	0	0
NearDrops	0	145	<b>557</b>	73	64	0
SmallDrops	0	294	63	<b>841</b>	61	0
Drops	0	176	58	115	<b>658</b>	0
Foam	0	0	0	0	0	<b>840</b>

### 4.3 Batchsize

Die Batchsize bestimmt, wie viele Samples aus dem Trainingsset pro Trainingsschritt verwendet werden, um die Gewichtungen mittels Backpropagation aufgrund der Differenz von richtigem Label und der Vorhersage anzupassen. Experimentiert wird mit den Batchgrößen 32, 64 und 128. Der Verlauf von Loss, Recall und Accuracy zeigen in Abbildung 24 und Abbildung 25, dass eine Batchsize von 64 und 128 die besseren Werte erzielen. Im Test auf ungesehenen Daten zeigt sich in Tabelle 10 zudem, dass das Netzwerk mit einer Batchsize von 64 den höchsten FBeta-Score erreicht.

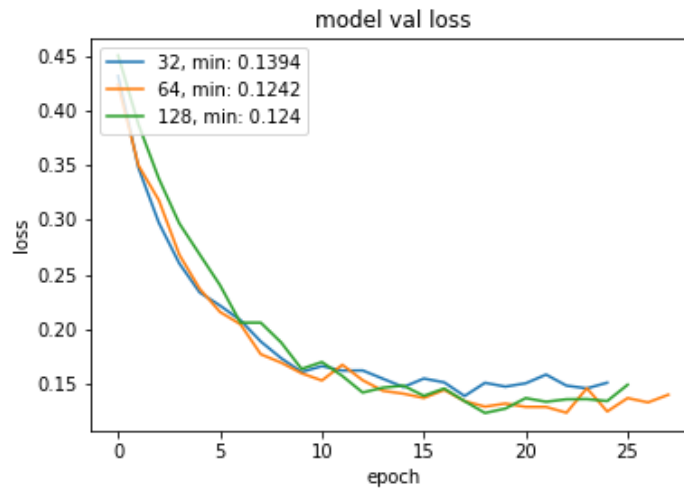


Abbildung 24: Verlauf des Validation Loss während dem Training mit unterschiedlichen Batchsizes. 64 und 128 schneiden besser ab als 32.

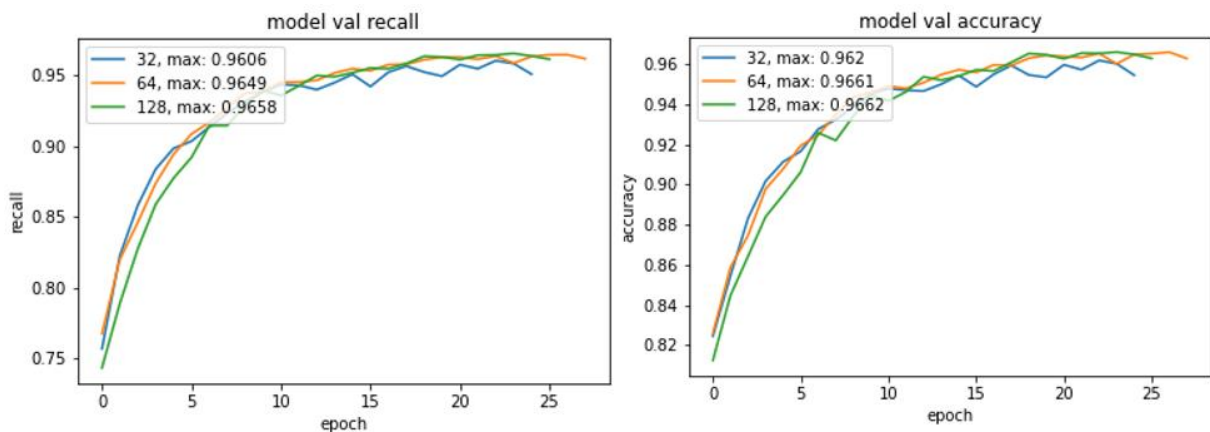


Abbildung 25: Verlauf von Recall und Accuracy während dem Training mit unterschiedlichen Batchsizes. Auch hier schneiden 64 und 128 besser ab als 32.

Tabelle 10: Vergleich der Metriken für die trainierten Modelle mit unterschiedlicher Batchsize.

Modell	Precision	Recall	False Pos Rate	False Neg Rate	Accuracy	FBeta-Score
32	0.9502	0.9394	0.0091	0.0606	0.9858	0.9415
64	0.9563	0.9502	0.0073	0.0498	0.9883	<b>0.9514</b>
128	0.9509	0.9486	0.0079	0.0514	0.9872	0.9490

#### 4.4 Einfluss der Kernel-Grösse

Die Kernel-Grösse ist ein wichtiger Hyperparameter, da er die Grösse der verwendeten Kernels bei der Faltung bestimmt. Die meisten state-of-the-art CNNs benutzen Kernels der Grösse 3x3, 5x5 oder 7x7 [36]. Durch die Experimente wird klar, dass Kernels der Grösse 3x3 oder 5x5 die besten Ergebnisse für das vorliegende Klassifizierungsproblem liefern.

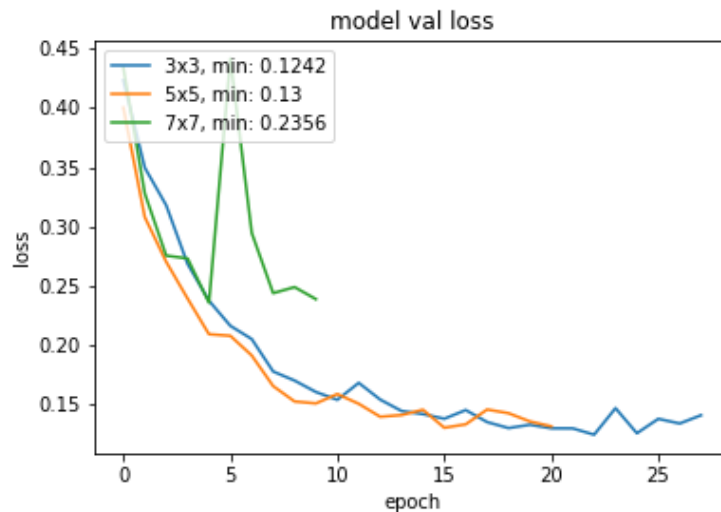


Abbildung 26: Der Verlauf des Validation Loss bei verschiedenen Kernel Grössen. 3x3 und 5x5 ergeben weitaus bessere Ergebnisse als 7x7 Kernels.

Wie in Abbildung 26 zu sehen ist, haben Kernels der Grösse 3x3 und 5x5 einen niedrigeren Loss als 7x7. Eine Kernel-Grösse von 3x3 ist dabei noch etwas besser als 5x5, wenn ein Dense Layer mit 128 Neuronen verwendet wird. Dieses Ergebnis zeigt sich ebenfalls im Verlauf von Recall und Accuracy während dem Training, wie in Abbildung 27 zu sehen ist. Die Auswertung auf dem Testset zeigt in Tabelle 11 zudem, dass 3x3 auch auf ungesehenen Daten mit einem FBeta-Score von 0.9514 am besten abschneidet.

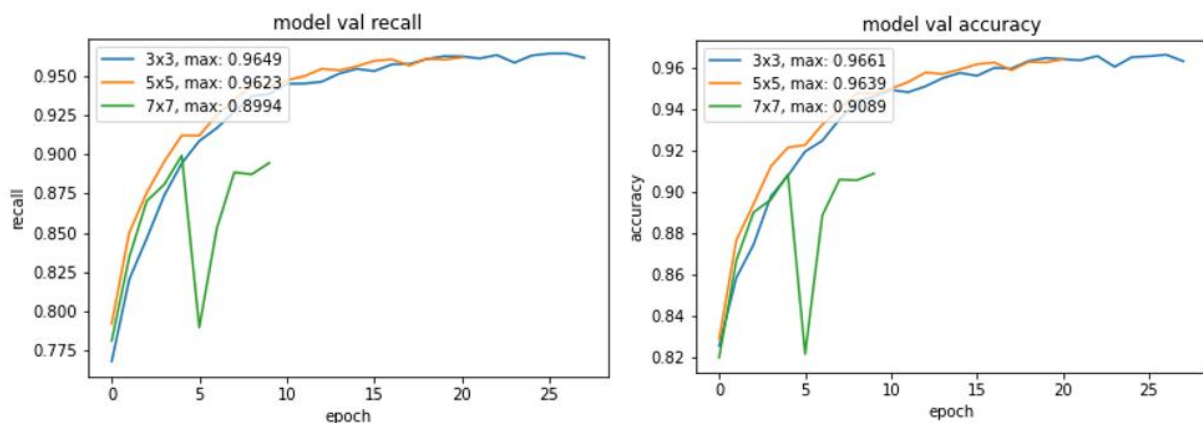


Abbildung 27: Verlauf von Recall und Accuracy bei verschiedenen Kernel Grössen mit einem Dense Layer und 128 Neuronen. Auch hier ist das beste Ergebnis bei 3x3 zu sehen.

Tabelle 11: Vergleich der Metriken für die trainierten Modelle mit unterschiedlicher Kernel-Grösse und einem Dense Layer und 128 Neuronen.

Modell	Precision	Recall	False Pos Rate	False Neg Rate	Accuracy	FBeta-Score
3x3_128d	0.9563	0.9502	0.0073	0.0498	0.9883	<b>0.9514</b>
5x5_128d	0.9509	0.9477	0.0082	0.0523	0.9869	0.9482
7x7_128d	0.8972	0.867	0.0191	0.133	0.9703	0.8718

Nun wird die Anzahl Dense Layer und Knoten auf 2 Dense Layer mit 128 respektive 256 Neuronen erhöht, um zu sehen, ob sich das Ergebnis verändert. 5x5 schneidet bei einem grösseren Netzwerk besser ab als 3x3 Kernel und hat sich, verglichen mit dem kleineren Netzwerk, leicht verbessert, wie in Abbildung 28 und Tabelle 12 zu sehen ist. 3x3 mit einem Dense Layer hat jedoch immer noch den besten Score.

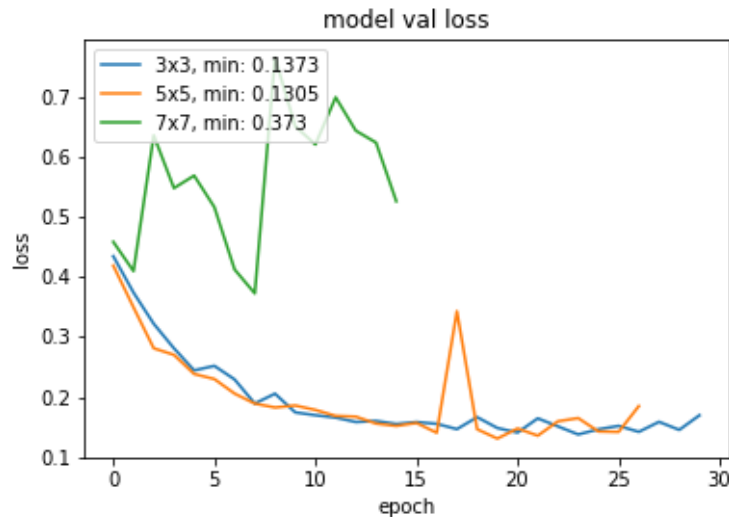


Abbildung 28: Vergleich von verschiedenen Kernel-Grössen, diesmal mit einem etwas grösseren Netzwerk aus 2 Dense Layer und 128 respektive 256 Neuronen. 5x5 ist diesmal besser als 3x3.

Tabelle 12: Vergleich der Metriken für die trainierten Modelle mit unterschiedlicher Kernel-Grösse und zwei Dense Layer mit 128 und 256 Neuronen.

Modell	Precision	Recall	False Pos Rate	False Neg Rate	Accuracy	FBeta-Score
3x3_128d_256d	0.9511	0.9428	0.0084	0.0572	0.9866	0.9442
5x5_128d_256d	0.9479	0.9489	0.0080	0.0510	0.9867	<b>0.9487</b>
7x7_128d_256d	0.8124	0.7866	0.0325	0.2134	0.9485	0.7792

#### 4.5 Einfluss der Anzahl Kernel

Mit der Anzahl Kernels kann bestimmt werden, wie viele verschiedene Features ein Netzwerk erlernen soll. Grundsätzlich wird die Anzahl der Kernels grösser mit jedem weiteren Convolutional Layer, da die Features mit jeder Faltung komplexer werden [36], [30]. Um die Auswirkung der Anzahl Kernel in den verschiedenen Convolutional Layer zu messen, werden vier Modelle mit steigender Anzahl Kernels trainiert und verglichen. Obwohl die Loss Funktion, welche in Abbildung 29 zu sehen ist, bei 16-32-64 (16 im ersten, 32 im zweiten und 64 im dritten Convolutional Layer) den minimalen Wert erreicht, zeigen Recall und Accuracy in Abbildung 28, dass 64-96-128 Kernels bessere Werte erreichen.

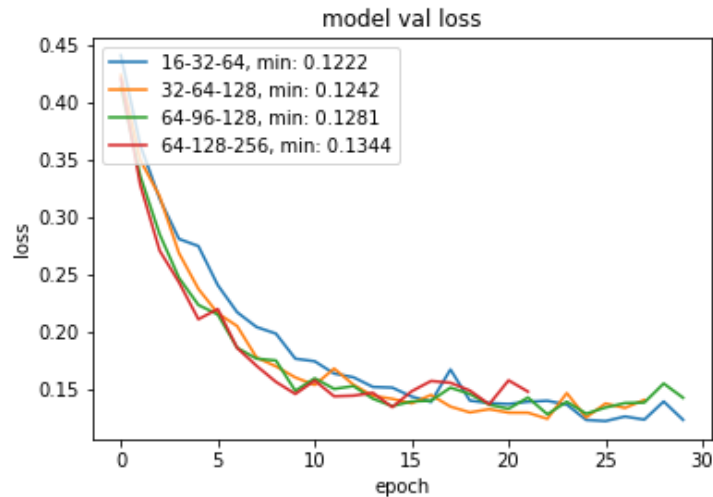


Abbildung 29: Verlauf der Loss Funktion bei Modellen mit verschiedener Anzahl Kernels pro Layer. Die Legende gibt die Anzahl Kernel in aufsteigender Reihenfolge an. 16-32-64 erreicht den minimalen Wert.

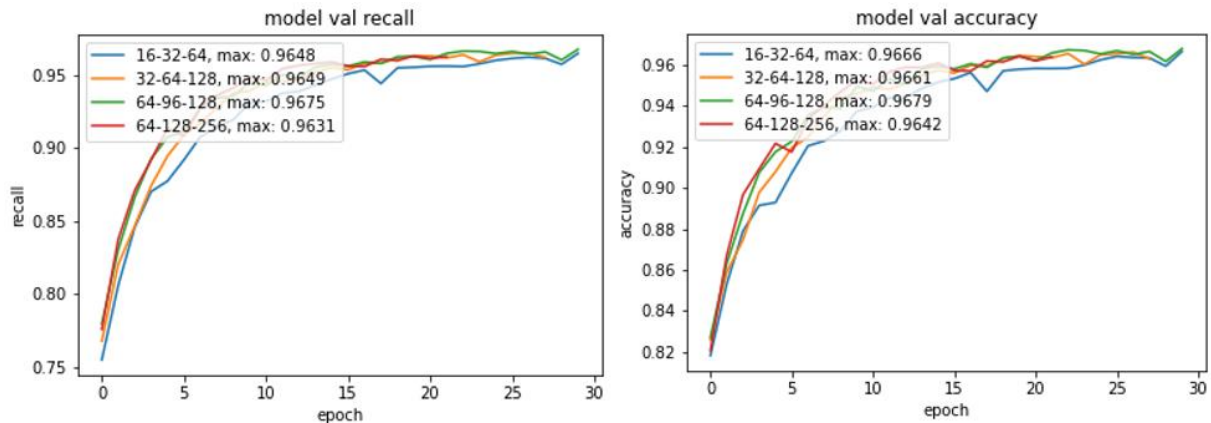


Abbildung 30: Verlauf von Recall und Accuracy. Gegenüber der Loss-Funktion erreicht das Netzwerk mit 64-96-128 Kernels hier die besten Werte.

Durch die Auswertung der Modelle auf dem Testset wird klar, dass 64-96-128 die besseren Ergebnisse liefert. Die Ergebnisse sind in Tabelle 13 aufgelistet.

Tabelle 13: Vergleich der Metriken für die trainierten Modelle mit einer unterschiedlichen Anzahl Neuronen in einem Dense Layer.

Modell	Precision	Recall	False Pos Rate	False Neg Rate	Accuracy	FBeta-Score
16-32-64	0.9577	0.9466	0.0076	0.0534	0.988	0.9486
32-64-128	0.9563	0.9502	0.0073	0.0498	0.9883	0.9514
64-96-128	0.9589	0.9534	0.0069	0.0466	0.9889	<b>0.9544</b>
64-128-256	0.9505	0.9405	0.0090	0.0595	0.9861	0.9422

Aufgrund der Ergebnisse wird angenommen, dass mit mehr Kernels auch mehr Features erlernt werden können. Zu viele Kernels führen wiederum zu Overfitting, weil Features erlernt werden, die unwichtige Informationen erfassen.



## 4.6 Einfluss der Anzahl Dense Layer

Die Dense Layer eines CNN stellen den klassifizierenden Teil des Netzwerks dar. Das Netzwerk lernt mittels den Gewichtungen, welche Feature Maps wichtig sind. Mehr Dense Layer vertiefen das Netzwerk, wodurch komplexere Funktionen erlernt werden können [37]. Tiefere Netzwerke sind bevorzugt gegenüber breiten Netzwerken, da gezeigt wurde, dass tiefere Netze oft besser abschneiden als breite [38]. Um den Einfluss der Anzahl Dense Layer zu messen, werden Modelle mit ein bis drei Layer und 64 bzw. 128 Neuronen pro Layer trainiert. Die Ergebnisse in Abbildung 31 und Abbildung 32 zeigen, dass ein Layer mit 128 Neuronen (1x128) die besten Ergebnisse erzielt.

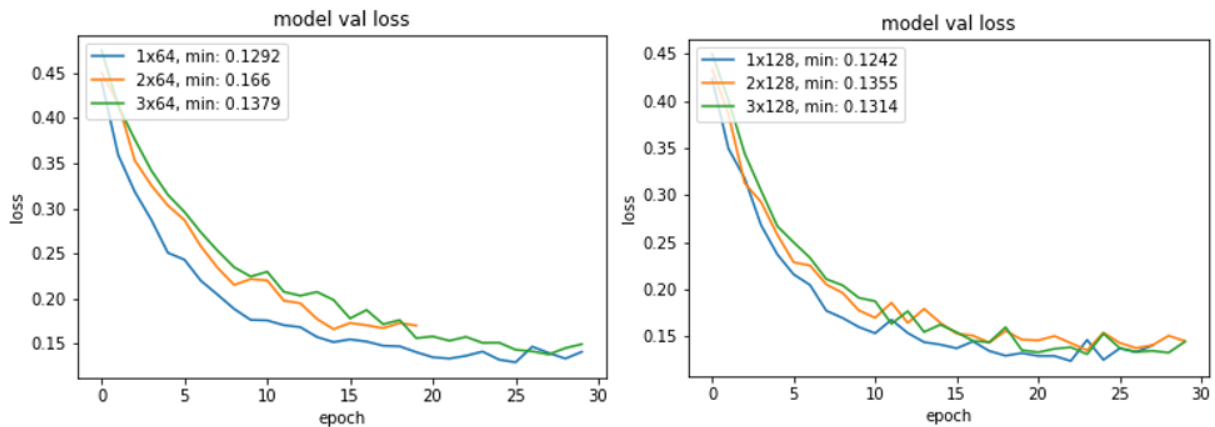


Abbildung 31: Verlauf des Validation Loss bei Netzwerken mit 1-3 Dense Layer. Die Modelle auf der linken Seite haben 64 Neuronen pro Layer und die Modelle auf der rechten Seite 128 Neuronen.

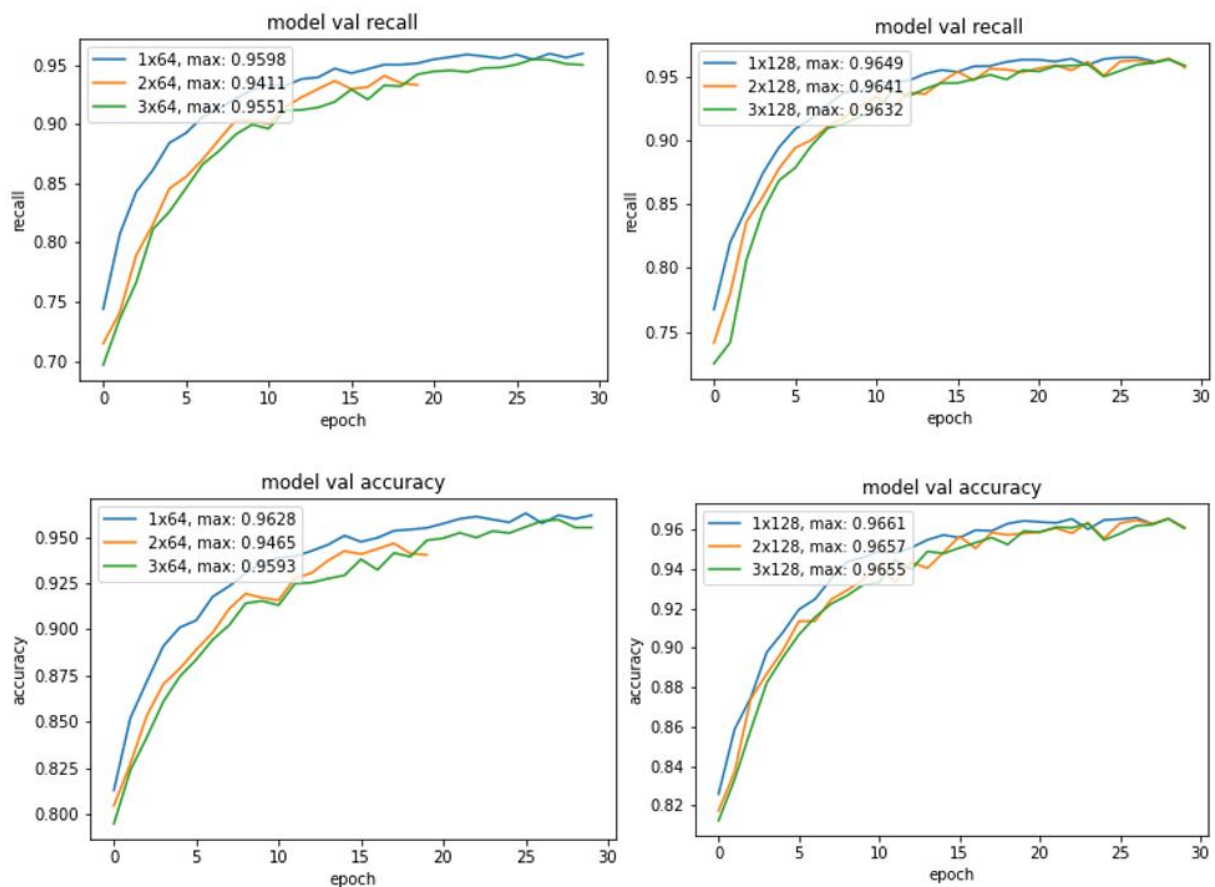


Abbildung 32: Verlauf von Recall und Accuracy bei Netzwerken mit 1-3 Dense Layer. Es ist zu sehen, dass das Netzwerk mit 1x128 (ein Dense Layer an 128 Neuronen) die besten Werte erzielt.



Tabelle 14: Vergleich der Metriken für die trainierten Modelle mit einer unterschiedlichen Anzahl Dense Layer mit jeweils 64 respektive 128 Neuronen pro Layer.

Modell	Precision	Recall	False Pos Rate	False Neg Rate	Accuracy	FBeta-Score
1x64	0.9473	0.9416	0.0085	0.0584	0.9862	0.9424
2x64	0.9243	0.9174	0.0122	0.0826	0.9802	0.9185
3x64	0.9452	0.9369	0.0093	0.0631	0.9850	0.9381
1x128	0.9563	0.9502	0.0073	0.0498	0.9883	<b>0.9514</b>
2x128	0.9520	0.9419	0.0085	0.0581	0.9865	0.9437
3x128	0.9504	0.9420	0.0088	0.0580	0.9863	0.9431

Die Tabelle 14 zeigt, dass das Modell mit einem Dense Layer an 128 Neuronen auch auf dem Testset am besten abschneidet. Obwohl ein tieferes Netzwerk oft bevorzugt wird, scheint in dem vorliegenden Fall ein einzelner Layer besser abzuschneiden.

#### 4.6.1 Variation der Anzahl Neuronen in den Dense Layer

Um zu sehen, ob das beste Ergebnis von einem Layer mit 128 Neuronen durch eine andere Anzahl Layer und Neuronen übertroffen werden kann, werden noch einige Modelle mit anderen Variationen trainiert:

- 256 im ersten Dense Layer und 128 im zweiten (256-128)
- 256 im ersten Dense Layer und 256 im zweiten (256-256)
- 128 im ersten Dense Layer und 64 im zweiten (128-64)
- 128 im ersten Layer, 128 im zweiten und 64 im dritten (128-128-64)
- 128 im ersten Layer, 256 im zweiten und 128 im dritten (128-256-128)

Zum Vergleich werden zusätzlich das beste Modell (mit einem Layer an 128 Neuronen) und ein Modell aus Abschnitt 4.4 (mit einer Kernel-Grösse von 3x3 und 128 Neuronen im ersten Layer bzw. 256 Neuronen im zweiten Layer) hinzugezogen. Der Verlauf der Loss Funktion in Abbildung 33 und der Vergleich der Metriken in Tabelle 15 zeigen jedoch, dass ein Layer mit 128 Neuronen immer noch am besten abschneidet. Allerdings kommt das Netzwerk mit 2 Layer und jeweils 256 Neuronen knapp auf einen FBeta-Score von 0.95.

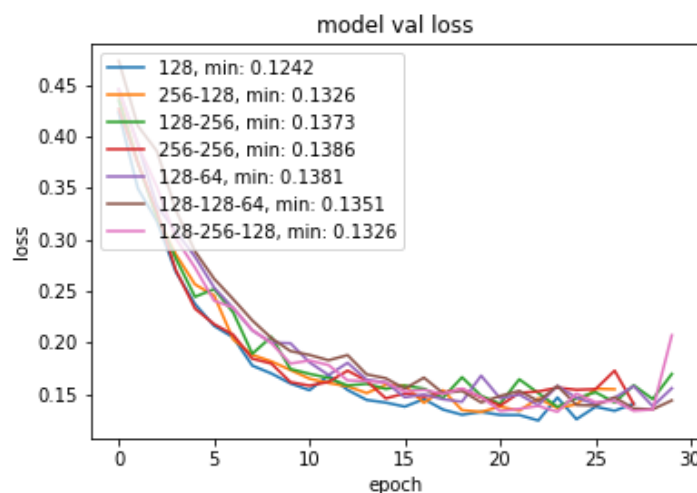


Abbildung 33: Verlauf des Validation Loss bei sieben verschiedenen Variationen von Anzahl Layer und Neuronen. Ein Layer mit 128 Neuronen schneidet immer noch am besten ab.

Tabelle 15: Vergleich der Metriken für trainierte Modelle mit unterschiedlichen Variationen von Anzahl Layer und Neuronen.

Modell	Precision	Recall	False Pos Rate	False Neg Rate	Accuracy	FBeta-Score
128	0.9563	0.9502	0.0073	0.0498	0.9883	<b>0.9514</b>
256-128	0.9566	0.9479	0.0079	0.0521	0.9876	0.9495
128-256	0.9511	0.9428	0.0084	0.0572	0.9866	0.9442
256-256	0.9514	0.9496	0.0076	0.0504	0.9876	0.9499
128-64	0.9516	0.9465	0.0082	0.0535	0.9869	0.9474
128-128-64	0.9496	0.9465	0.0082	0.0535	0.9867	0.9469
128-256-128	0.9331	0.9173	0.0127	0.0827	0.9804	0.9201

## 4.7 Andere Architekturen

Zusätzlich zu den Hyperparametern für die LeNet Architektur wurde entschieden, noch drei weitere Architekturen zu untersuchen. Die verwendeten Architekturen sind:

- **Multi-Convolutional** (Multi-Conv), welches mehrere Convolutional Layer vor einem Pooling Layer einsetzt. Die Architektur orientiert sich am VGG-16 Netz [39].
- **Bottleneck** [40] schliesst jeweils einen Convolutional Layer mit einer Kernel-Grösse von 3x3 oder 5x5 zwischen zwei Convolutional Layer mit einer Kernel-Grösse von 1x1 ein.
- **Residual Net** (ResNet) [40] verwendet Bottlenecks sowie Skip-connections und Shortcuts. Der Vorteil dieser Architektur ist die Tiefe des Netzwerks, ohne dass der Verbrauch von Arbeitsspeicher und Rechenleistung zu hoch wird.

In dieser Arbeit werden aufgrund der vorhandenen Ressourcen kleinere Versionen der Architekturen eingesetzt. Zudem wird aus Zeitgründen jeweils nur ein Modell trainiert. Wie die Ergebnisse in der Tabelle 16 zeigen, erreicht das Residual Net den besten FBeta-Score von 0.9604. Das eingesetzte ResNet verfügt über 31 Convolutional und zwei Dense Layer mit 256 respektive 128 Neuronen.

Tabelle 16: Vergleich der Metriken von trainierten Modellen mit unterschiedlichen Architekturen. Das ResNet hat den besten FBeta-Score.

Modell	Precision	Recall	False Pos Rate	False Neg Rate	Accuracy	FBeta-Score
LeNet	0.9589	0.9534	0.0069	0.0466	0.9889	0.9544
Multi-Conv	0.9637	0.9575	0.0062	0.0425	0.9901	0.9587
Bottleneck	0.9468	0.9437	0.0083	0.0563	0.9864	0.9442
ResNet	0.9660	0.959	0.0062	0.0410	0.9903	<b>0.9604</b>

## 4.8 Analyse von neuen Videos

Um die trainierten Modelle mit neuen Videos zu testen, wird das in Kapitel 3.4.3 definierte Protokoll für 4 Videos durchgeführt. Es wird sichergestellt, dass Videos analysiert werden, die entweder neu gefilmt wurden oder nicht im Dataset vorhanden sind, um ein möglichst reales Szenario zu approximieren.

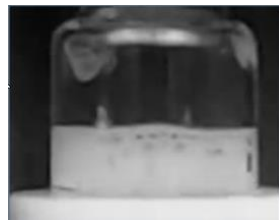
Aus Zeitgründen werden die drei besten Modelle ausgewählt und mit neuen Videos getestet. Folgende Modelle wurden ausgewählt:

*Tabelle 17: Ausgewählte Modelle zum Testen mit neuen Videos*

Modell-Nummer	Modell-Name
1	LeNet
2	Multi-Conv
3	ResNet

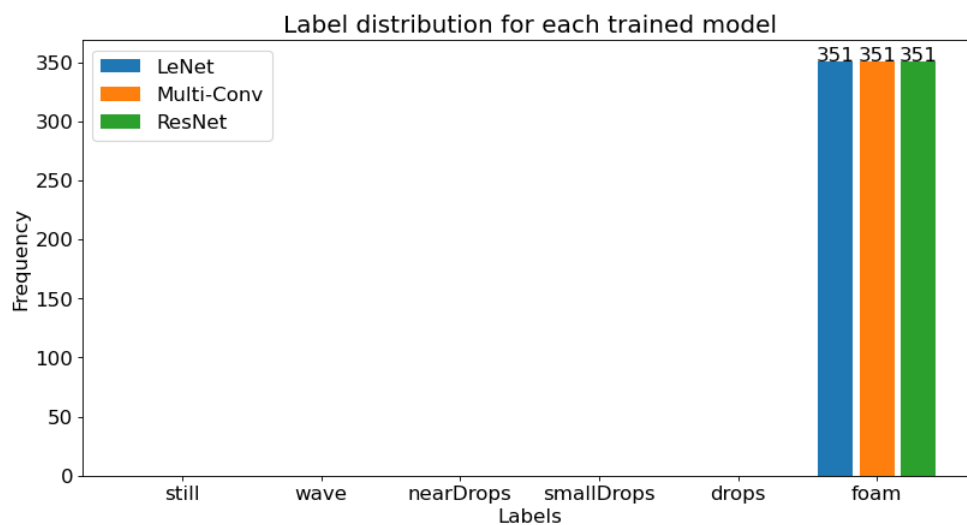
### 4.8.1 Video 1 (Foam)

Bei Video 1 handelt es sich um eine Phiole mit Schaumbildung und leichter Vibration. Das Video besteht aus 351 *Foam* Frames, wobei ein Frame als Illustration in Abbildung 34 dargestellt wird.



*Abbildung 34: Beispiel eines Foam Frame in Video 1*

Wie in Abbildung 35 zu sehen, wird das Video von allen Modellen korrekt klassifiziert:



*Abbildung 35: Verteilung der klassifizierten Frames in Video 1. Alle Modelle erkennen nur Foam Frames.*

#### 4.8.2 Video 2 (Still)

Das Video 2 besteht aus 38 Frames. Im Video bleibt die Oberfläche meistens flach, allerdings bilden sich auch leichte Wellen. In Abbildung 36 ist die Auswertung der Modelle zu sehen.

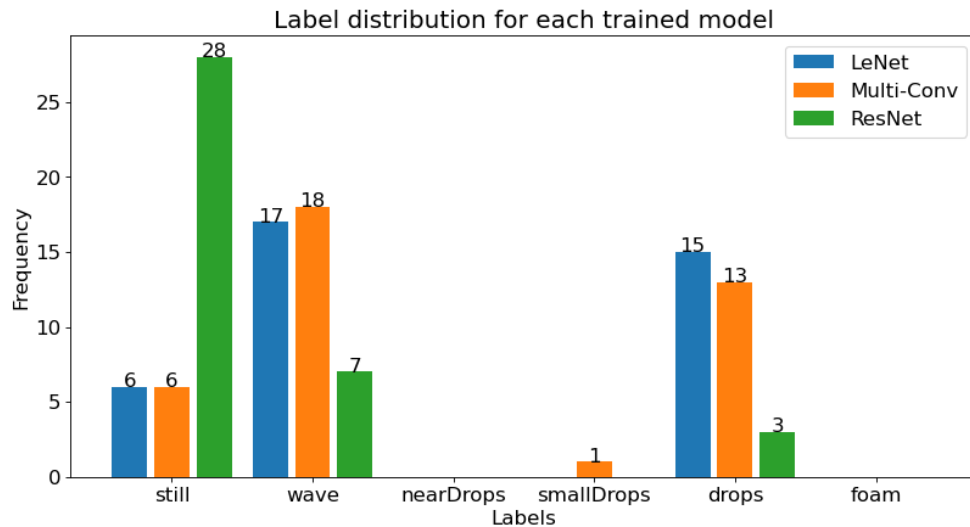
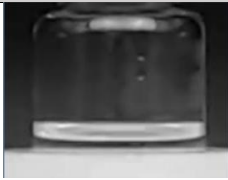

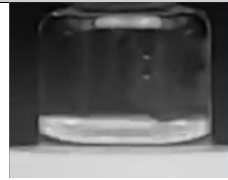


Abbildung 36: Verteilung der klassifizierten Frames in Video 2 von den drei Modellen

#### LeNet:

*Still* Frames werden vom LeNet fälschlicherweise als *Wave* klassifiziert, einige sogar als *Drops*. Dies liegt vermutlich an der Lichtreflektion in der Mitte der Phiole. Beispiele davon werden in Tabelle 18 dargestellt. Obwohl sich die Frames kaum unterscheiden, werden zwei davon falsch klassifiziert.




Tabelle 18: Beispiele der Klassifizierung von Video 2 mit dem LeNet

Still	Wave	Drops
		
TP	FP	FP

#### Multi-Conv:

Es werden wie auch schon vom LeNet *Still* Frames als *Wave* klassifiziert und einige als *Drops*. Die Auswertung unterscheidet sich kaum vom LeNet. Beispiele davon werden in Tabelle 19 dargestellt.

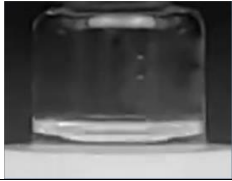
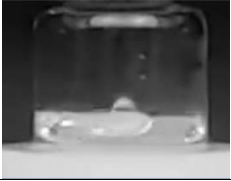
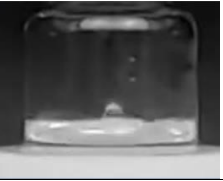
Tabelle 19: Beispiele der Klassifizierung von Video 2 mit dem Multi-Conv

Still	Wave	Drops
		
TP	FP	FP

### ResNet:

Die Auswertung vom ResNet stimmt mit den Erwartungen überein. Wie in Tabelle 20 zu sehen ist, klassifiziert das ResNet korrekterweise leichte Bewegungen als *Waves* und ruhige Oberflächen als *Still*. Auch False Positives von Tröpfchen werden, wie in Abbildung 36 ersichtlich, reduziert.

Tabelle 20: Beispiele der Klassifizierung von Video 2 mit dem ResNet

Still	Wave	Drops
		
TP	TP	FP

#### 4.8.3 Video 3 (Wave)

Video 3 ist ein Ausschnitt aus einem gefilmten Truck-Profil mit starker Vibration, wobei nur Wellen und keine Tröpfchen entstehen. Das Video besteht insgesamt aus 801 Frames. In Abbildung 37 ist die Auswertung der Modelle zu sehen.

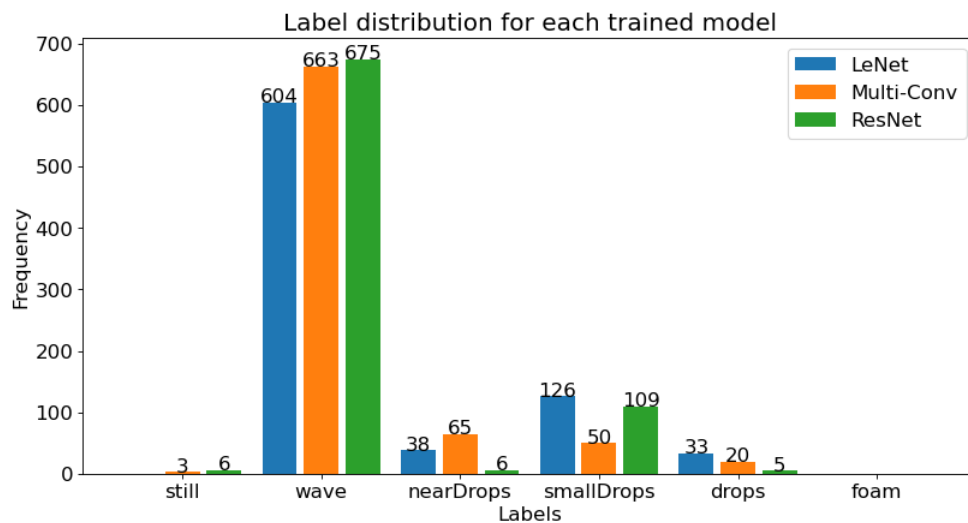





Abbildung 37: Verteilung der klassifizierten Frames in Video 3 von den drei Modellen

**LeNet:**

Von den 50 zufällig gewählten *Wave* Frames wurden alle korrekt klassifiziert. Alle 33 erkannten *Drops* waren hier False Positives. Bei *Drops* und *SmallDrops* Frames ist zu sehen, dass eine leichte Reflektion vom Licht in der Mitte besteht, was das Modell als *Drops* oder *SmallDrops* erkennt. Beispiele werden in Tabelle 21 dargestellt.




Tabelle 21: Beispiele der Klassifizierung von Video 3 mit dem LeNet

Wave	SmallDrops	Drops
		
TP	FP	FP

**Multi-Conv:**

Von 50 *Waves* waren alle korrekt. Alle 20 erkannten *Drops* sind False Positives. Auch bei *SmallDrops* handelt es sich nur um eine Lichtreflexion und werden daher falsch klassifiziert. Beispiele werden in Tabelle 22 dargestellt.




Tabelle 22: Beispiele der Klassifizierung von Video 3 mit dem Multi-Conv

Wave	SmallDrops	Drops
		
TP	FP	FP

**ResNet:**

Die Verteilung stimmt mit der Realität besser überein. Es werden mehr Frames als *Wave* klassifiziert als bei den anderen. Aber auch hier ist bemerkbar, dass die Lichtreflektionen dazu führen, dass Frames als *SmallDrops* klassifiziert werden. Jedoch hat es weniger False Positives als bei den anderen Modellen. Beispiele werden in Tabelle 23 dargestellt.

Tabelle 23: Beispiele der Klassifizierung von Video 3 mit dem ResNet

Wave	SmallDrops	Drops
		
TP	FP	FP

#### 4.8.4 Video 4 (Drops)

Bei Video 4 handelt es sich um ein Video mit starker Vibration und vielen klaren Tröpfchen. Insgesamt werden 1081 Frames mit den Modellen analysiert. In Abbildung 38 ist die Auswertung der drei Modelle zu sehen.

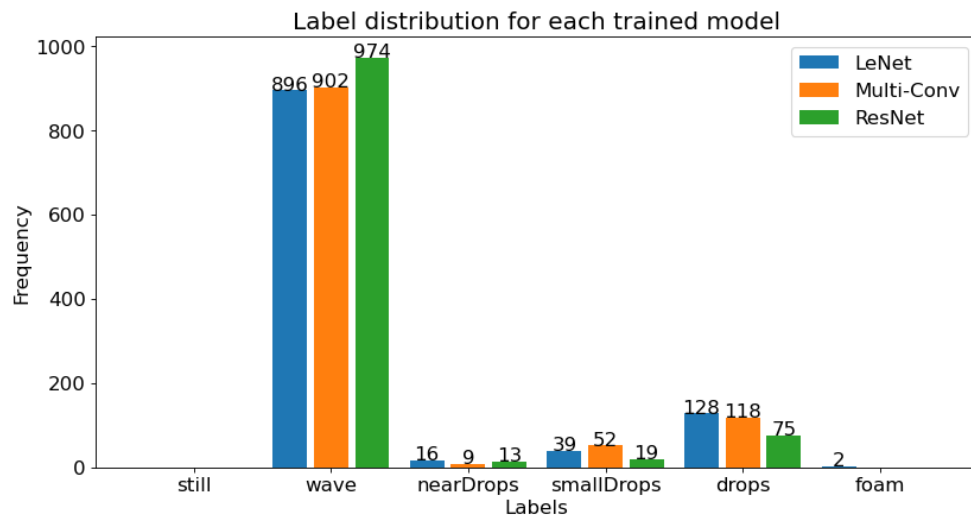


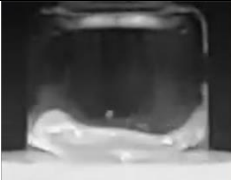







Abbildung 38: Verteilung der klassifizierten Frames in Video 4 von den drei Modellen

#### LeNet:

Von den 128 Frames, die als *Drops* klassifiziert wurden, wurden 50 stichprobenmässig überprüft. Dabei waren 34 von 50 korrekt. Von den 50 überprüften *Wave* Frames waren alle korrekt. Es wurden hier zusätzlich 10 *NearDrops* und 10 *SmallDrops* stichprobenmässig überprüft, von denen 9 *NearDrops* und alle *SmallDrops* korrekt klassifiziert wurden. Es ist das einzige Modell, dass 2 Frames fälschlicherweise als *Foam* klassifiziert. Beispiele sind in Tabelle 24 dargestellt.





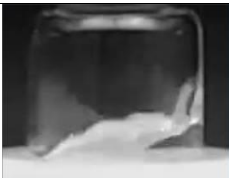



Tabelle 24: Beispiele der Klassifizierung von Video 4 mit dem LeNet

Wave	NearDrops	SmallDrops	Drops
			
TP	TP	TP	FP
			
TP	FP	TP	TP

**Multi-Conv:**

Von den 118 Frames, die als *Drops* klassifiziert wurden, wurden 50 stichprobenmässig überprüft. Dabei waren 32 von 50 korrekt. Von den 50 überprüften *Waves* waren alle korrekt. Es wurden hier zusätzlich alle 9 *NearDrops* und 10 *SmallDrops* überprüft, von denen 8 *NearDrops* und 7 *SmallDrops* korrekt waren. Beispiele sind in Tabelle 25 dargestellt.








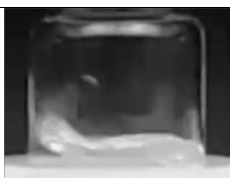
Tabelle 25: Beispiele der Klassifizierung von Video 4 mit dem Multi-Conv

Wave	NearDrops	SmallDrops	Drops
			
TP	TP	FP	TP
			
TP	TP	TP	TP

**ResNet:**

Von den 75 Frames, die als *Drops* klassifiziert wurden, wurden 50 stichprobenmässig überprüft. Dabei waren 48 von 50 korrekt. Von den 50 überprüften *Waves* waren alle korrekt. Es wurden zusätzlich 10 *NearDrops* und 10 *SmallDrops* stichprobenmässig überprüft, von denen alle korrekt klassifiziert wurden. Beispiele sind in Tabelle 26 dargestellt.

Tabelle 26: Beispiele der Klassifizierung von Video 4 mit dem ResNet

Wave	NearDrops	SmallDrops	Drops
			
TP	TP	TP	TP
			
TP	TP	TP	FP



## 4.9 Zusammenfassung der Experimente

Das optimierte LeNet sowie die Multi-Convolutional und ResNet Architekturen erreichen einen FBeta-Score von über 95 %. Für das LeNet wurden erfolgreich die besten Hyperparameter identifiziert. Das optimierte LeNet mit einem FBeta-Score von 0.9544 hat folgende Hyperparameter:

- Batchsize 64
- Kernel-Grösse 3x3
- Anzahl Kernels:
  - 64 im ersten Convolutional Layer
  - 96 im zweiten Convolutional Layer
  - 128 im dritten Convolutional Layer
- Ein Dense Layer mit 128 Neuronen
- Eine Verteilung mit der in Tabelle 7 definierten Gewichtung

Durch die tiefe und spezielle Architektur des ResNet erreicht dies eine leicht bessere Performanz wie das LeNet auf dem Testset. Besonders bei der Analyse von neuen Videos hat das ResNet bessere Ergebnisse erzielt.

## 5. Diskussion und Ausblick

In der vorliegenden Arbeit wurde ein Supervised Learning Ansatz mit Convolutional Neural Networks für die Auswertung von Hochgeschwindigkeitsaufnahmen von Fluid-Instabilitäten präsentiert. Für das Training des Netzwerks wurde ein eigenes Dataset aufbereitet. Es wurde gezeigt, dass sich Data Augmentation sowie die Balancierung positiv auf das Ergebnis auswirken. Anhand der durchgeführten Experimente konnten die besten Hyperparameter für die LeNet Architektur festgelegt werden. Es wurde festgestellt, dass ein ResNet das optimierte LeNet jedoch übertrifft. Das optimierte LeNet erreicht einen FBeta-Score von 95.44 %, während das ResNet einen FBeta-Score von 96.04 % erreicht und somit um 0.6 % besser abschneidet. Die erreichten Scores zeigen, dass sich CNNs eignen, um die Aufnahmen der Experimente von Fluid-Instabilitäten zu untersuchen. Im Verlauf der Arbeit wurde jedoch klar, dass die trainierten Modelle in einigen Aspekten limitiert sind. Es wurde festgestellt, dass Lichtreflexionen zu Fehlklassifizierungen führen, was die Auswertung einer automatischen Analyse negativ beeinflusst. Die Genauigkeit der Klassifizierung ist daher von der Einhaltung der Rahmenbedingungen beim Filmen abhängig. Auch gibt es noch Potenzial für Verbesserungen des Modells. Für die Vorhersage der Klasse könnten Optimierungen wie Ensembles und der Einbezug von temporalen Informationen ein besseres Ergebnis liefern. Durch ein grösseres und diverseres Dataset können tiefere Architekturen verwendet werden, wodurch ein noch stabileres und besser verwendbares Modell trainiert werden kann. Die Erweiterung des Dataset kann von Hand aber auch mit realistischen Renderings erfolgen [41]. Des Weiteren ist das entwickelte Modell eher als Unterstützung anzusehen und kann das manuelle Analysieren von Videos nicht komplett ersetzen. Das System ist wie erwartet nicht fähig, jedes einzelne Bild in einem Video korrekt zu klassifizieren, charakterisiert aber einen grossen Bestandteil des Videos korrekt und gibt daher ein gutes Gesamtbild der auftretenden Instabilitäten. Es besteht die Meinung, dass die drei besten Modelle bereits heute zur Unterstützung bei der Analyse der Videos verwendet werden können, um den Aufwand zu verringern. In einem nächsten Schritt kann eine Applikation um das Modell und den erstellten Datensatz gebaut werden, um es in einer produktiven Umgebung einzusetzen. Vorstellbare Features wären:

- Extraktion der vorhergesagten Frames mit der grössten Konfidenz von jeder vorhandenen Klasse, um die Klassifizierung manuell zu validieren.
- Ein In-App Annotations-Tool, um das Modell iterativ auf neue Daten (z. B. bei neuen Kameraeinstellungen oder Veränderungen im Experimentumfeld) anzupassen und das Modell somit weiter zu generalisieren und das Dataset zu erweitern.
- Eine automatisierte Videobearbeitungspipeline, um die Schritte vom rohen Video zu den analysierbaren Phiole-Videos zu vereinfachen.
- Die analysierten Frames nach der Klassifizierung weiter auf die Intensität der Instabilität analysieren.

## 6. Verzeichnisse

### 6.1 Literaturverzeichnis

- [1] D. Cl  net, „Accurate prediction of vaccine stability under real storage conditions and during temperature excursions,“ *European Journal of Pharmaceutics and Biopharmaceutics*, pp. 76-84, April 2018.
- [2] A. Krizhevsky, I. Sutskever und G. E. Hinton, „ImageNet Classification with Deep Convolutional Neural Networks,“ *Communications of the ACM*, p. 84–90, 24. Mai 2017.
- [3] K. He, X. Zhang, S. Ren und J. Sun, „Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,“ ICCV, 2015.
- [4] S. Kido, Y. Hirano and N. Hashimoto, "Detection and classification of lung abnormalities by use of convolutional neural network (CNN) and regions with CNN features (R-CNN)," 2018 International Workshop on Advanced Image Technology (IWAIT), 2018, pp. 1-4.
- [5] M. Dobko, B. Petryshak und O. Dobosevych, CNN-CASS: CNN for Classification of Coronary Artery Stenosis Score in MPR Images, Lviv, Ukraine: The Machine Learning Lab, Ukrainian Catholic University, 2020.
- [6] H. Wang, C. Ma und L. Zhou, „A Brief Review of Machine Learning and Its Application,“ in *2009 International Conference on Information Engineering and Computer Science*, Wuhan, China, 2009.
- [7] M. Mohri, A. Rostamizadeh und A. Talwalkar, Foundations of Machine Learning, second edition, Cambridge, Massachusetts: MIT Press, 2018.
- [8] A. L. Fradkov, „Early History of Machine Learning,“ *IFAC-PapersOnLine*, pp. 1385-1390, Januar 2020.
- [9] L. Zhou, S. Pan, J. Wang und A. V. Vasilakos, „Machine learning on big data: Opportunities and challenges,“ *Neurocomputing, Volume 237*, pp. 350-361, 26. Januar 2017.
- [10] T. M. Mitchell, „The Discipline of Machine Learning,“ Machine Learning Department, Carnegie Mellon University, Pittsburgh, PA, 2006.
- [11] S. Russel und P. Norvig, „Learning from examples,“ in *Artificial Intelligence A Modern Approach Third Edition*, Upper Saddle River, New Jersey, Pearson Education, 2010, pp. 693-757.
- [12] L. Bing, „Supervised Learning,“ in *Web Data Mining. Data-Centric Systems and Applications*, Berlin, Springer, 2011, p. 63.

- [13] S. Raschka, *Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning*, Madison, Wisconsin: arXiv, 2020.
- [14] J. D. Novaković, A. Veljović, S. S. Ilić, Ž. Papić und T. Milica, „Evaluation of Classification Models in Machine Learning,“ *Theory and Applications of Mathematics & Computer Science* 7, pp. 39-46, 1. April 2017.
- [15] B. Planche, Z. Wu, K. Ma, S. Sun, S. Kluckner, T. Chen, A. Hutter, S. Zakharov, H. Kosch und J. Ernst, „DepthSynth: Real-Time Realistic Synthetic Data Generation from CAD Models for 2.5D Recognition,“ arXiv:1702.08558v2 [cs.CV], 2017.
- [16] A. Jain, J. Mao und K. Mohiuddin, „Artificial neural networks: a tutorial,“ *Computer*, vol. 29, no. 3, pp. 32-44, März 1996.
- [17] D. Hebb, „The Organization of Behavior: A Neuropsychological Theory,“ *The American Journal of Psychology*, pp. 633-642, Oktober 1949.
- [18] M. Minsky und S. A. Papert, *Perceptrons: An Introduction to Computational Geometry*, Cambridge: MIT Press, 1969.
- [19] D. C. Ciresan, U. Meier, L. M. Gambardella und J. Schmidhuber, „Deep Big Simple Neural Nets Excel on Hand-written Digit Recognition,“ *Neural Computation*, Volume 22, p. 3207–3220, Dezember 2010.
- [20] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, L. Timothy, L. Madeleine, K. Koray, G. Thore und H. Demis, „Mastering the game of Go with deep neural networks and tree search,“ *Nature*, Nr. 529, p. 484–489, 2016.
- [21] M. Haenlein und A. Kaplan, „A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence,“ *California Management Review*, Volume 61, pp. 5-14, 17. Juli 2019.
- [22] M. C. Radoslaw, K. Aditya, P. Dimitrios, T. Antonio und O. Aude, „Comparison of deep neural networks to spatio-temporal cortical dynamics of human visual object recognition reveals hierarchical correspondence,“ *scientific reports*, 2016.
- [23] R. Szeliski, *Computer Vision: Algorithms and Applications*, Springer Science & Business Media, 2010.
- [24] T. S. Huang, „Computer Vision: Evolution and Promise,“ 19th CERN School of Computing, Genf, 1996.
- [25] K. Fukushima, *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*, Tokyo: Springer-Verlag, 1980.
- [26] Y. LeCun, L. Bottou, Y. Bengio und P. Haffner, „Gradient-based learning applied to document recognition,“ *Proceedings of the IEEE*, pp. 2278-2324, 1998.

- [27] R. Yamashita, M. Nishio, R. K. G. Do und K. Togashi, „Convolutional neural networks: an overview and application in radiology,“ *Insights into Imaging*, p. 611–629, August 2018.
- [28] A. A. A. Setio, F. Ciompi, G. Litjens, P. Gerke, C. Jacobs, S. J. van Riel, M. M. W. Wille, M. Naqibullah, C. I. Sánchez und B. van Ginneken, „Pulmonary nodule detection in CT images: false positive reduction using multi-view convolutional network,“ *IEEE Transactions on Medical Imaging*, pp. 1160-1169, Mai 2016.
- [29] S. Armato III, G. McLennan, L. Bidaut, M. McNitt-Gray, C. Meyer, A. Reeves, B. Zhao, D. Aberle, C. Henschke, E. Hoffman, E. Kazerooni, H. Macmahon, E. Beek, D. Yankelevitz, A. Biancardi, P. Bland, M. Brown, R. Engelmann, G. Laderach und L. Clarke, „The Lung Image Database Consortium (LIDC) and Image Database Resource Initiative (IDRI): A Completed Reference Database of Lung Nodules on CT Scans,“ *Medical Physics*, p. 915–931, 24. Januar 2011.
- [30] B. van Ginneken, S. G. Armato, B. de Hoop, S. van Amelsvoort-van de Vorst, T. Duindam, M. Niemeijer, K. Murphy, A. Schilham, A. Retico, M. E. Fantacci, N. Camarlinghi, F. Bagagli, I. Gori, T. Hara, H. Fujita, G. Gargano, R. Bellotti, S. Tangaro, L. Bolaños, F. De Carlo, P. Cerello, S. Cristian Cheran, E. Lopez Torres und M. Prokop, „Comparing and combining algorithms for computer-aided detection of pulmonary nodules in computed tomography scans: The ANODE09 study,“ *Medical Image Analysis*, pp. 1361-8415, Dezember 2010.
- [31] J. H. Pedersen, H. Ashraf, A. Dirksen, K. Bach, H. Hansen, P. Toennesen, H. Thorsen, J. Brodersen, B. G. Skov, M. Døssing, J. Mortensen, K. Richter, P. Clementsen und N. Seersholm, „The Danish Randomized Lung Cancer CT Screening Trial—Overall Design and Results of the Prevalence Round,“ *Journal of Thoracic Oncology*, pp. 608-614, Mai 2009.
- [32] S. Kotsiantis, D. Kanellopoulos und P. Pintelas, „Handling imbalanced datasets: A review,“ *GESTS International Transactions on Computer Science and Engineering*, pp. 25-36, 30. November 2005.
- [33] A. K. Dubey und V. Jain, „Comparative Study of Convolution Neural Network’s Relu and Leaky-Relu Activation Functions,“ in *Applications of Computing, Automation and Wireless Systems in Electrical Engineering*, Singapore, Springer Nature, 2019, pp. 873-880.
- [34] A. Demirkaya, J. Chen und S. Oymak, „Exploring the Role of Loss Functions in Multiclass Classification,“ in *2020 54th Annual Conference on Information Sciences and Systems (CISS)*, Princeton, NJ, USA, 2020.
- [35] H. Li, J. Li, X. Guan, B. Liang, Y. Lai und X. Luo, „Research on Overfitting of Deep Learning,“ in *15th International Conference on Computational Intelligence and Security (CIS)*, Macao, China, 2019.
- [36] H. H. Aghdam und E. J. Heravi, *Guide to Convolutional Neural Networks*, New York: Springer, 2017.

- [37] S. Liang und R. Srikant, „Why Deep Neural Networks for Function Approximation?“, in *ICLR 2017*, Illinois, 2017.
- [38] C.-C. J. Kuo, „Understanding convolutional neural networks with a mathematical model“, *Journal of Visual Communication and Image Representation*, pp. 406-413, 5. November 2016.
- [39] K. Simonyan und A. Zisserman, „Very Deep Convolutional Networks for Large-Scale Image Recognition“, arXiv:1409.1556v1 [cs.CV], 2014.
- [40] K. He, X. Zhang, S. Ren und J. Sun, „Deep Residual Learning for Image Recognition“, arXiv:1512.03385v1 [cs.CV], 2015.
- [41] I. B. Barbosa, M. Cristani, B. Caputo, A. Rognhaugen und T. Theoharis, „Looking beyond appearances: Synthetic training data for deep CNNs in re-identification“, *Computer Vision and Image Understanding*, pp. 50-62, 12 Dezember 2017.
- [42] S. Renuka und S. Pothula, „A State of Art Techniques on Machine Learning Algorithms: A Perspective of Supervised Learning Approaches in Data Classification“, in *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, 2018.
- [43] S. A. Kalogirou, „Artificial neural networks in renewable energy systems applications: a review“, in *Renewable and Sustainable Energy Reviews, Volume 5, Issue 4*, Cyprus, ScienceDirect, 2001, pp. 373-401.
- [44] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever und R. Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting“, *Journal of Machine Learning Research*, pp. 1929-1958, 6. Januar 2014.
- [45] J. Guibas und V. Tejpai, „Convolutional Neural Networks Explained“, Twopointseven, 29. Oktober 2017. [Online]. Available: <https://twopointseven.github.io/2017-10-29/cnn/>. [Zugriff am 22. Mai 2021].

## 6.2 Glossar

<b>Begriff</b>	<b>Definition / Erklärung</b>
Convolutional Neural Networks (CNN)	Ein künstliches neuronales Netz, das in der Bilderkennung verwendet wird.
Data Augmentation	Eine Methode, um die Menge der Daten durch Hinzufügen von leicht modifizierten Kopien des Datasets zu erhöhen.
Dropout	Methode, um Overfitting zu verringern. Bei jedem Trainingsschritt wird ein im Vorfeld festgelegter prozentualer Anteil der Neuronen in einem Layer ausgeschaltet.
Faltung	Eine mathematische Operation in der Bildverarbeitung, wobei eine Matrix über ein Bild geschoben wird, und zuerst elementweise das Produkt berechnet und danach aufsummiert wird.
Feature	In Computer Vision ist ein Feature ein gewisses Merkmal in einem Bild, welches sich von anderen Bereichen unterscheidet. Features können spezielle Farben oder Strukturen wie Ecken, Punkte oder Objekte sein.
Fluid-Instabilitäten	Das Auftreten von Bewegung in der Flüssigkeit, wobei sich z. B. Wellen, Tröpfchen oder Schaum bilden.
Ground Truth	Information von der klar ist, dass sie stimmt. In diesem Zusammenhang die richtige Klassifizierung, die von Hand gegeben wurde.
Hyperparameter	Konfigurierbare Parameter bei der Definition des Netzwerks. Diese werden nicht durch das Training erlernt.
Kernel / Filter	Eine Matrix, die bei der Faltung verwendet wird. Abhängig davon, welche Werte im Kernel sind, können Bildverarbeitungsoperationen wie Blurring eines Bildes oder Feature Detection durchgeführt werden.
Loss Function	Englischer Begriff für Verlustfunktion. Misst den Unterschied von Vorhersage zum wahren Wert.
Maschinelles Lernen (ML)	Unterkategorie der künstlichen Intelligenz mit dem Ziel, Maschinen zu trainieren, um aus Daten zu lernen und sich zu verbessern.
Modell	Eine durch das Training erlernte Funktion, welche die Daten abbildet und Vorhersagen treffen kann.
Overfitting	Ein Phänomen im maschinellen Lernen, bei dem ein Modell sich zu stark an die Trainingsdaten anpasst und auf unbekannte Eingaben schlechte Vorhersagen trifft.

## 6.3 Abbildungsverzeichnis

Abbildung 1: XZ-Shaker vom ZHAW ICP Lab .....	5
Abbildung 2: Supervised Learning Ansätze als Organigramm. Die Methoden sind in probabilistische, lineare und andere Klassifikatoren unterteilt [42]. .....	8
Abbildung 3: Aufbau eines künstlichen Neurons [11]. Die Input-Verbindungen sind gewichtet und werden in der Input Function aufsummiert. Die Aktivierungsfunktion bestimmt den Output, welcher mittels Output-Verbindungen an die nächsten Neuronen weitergegeben wird. ....	10
Abbildung 4: Architektur eines Feed-Forward Netzwerks [43] Zu sehen ist die Unterscheidung zwischen Input, Output sowie Hidden Layer und die Verbindungen gehen nur nach vorne.....	11
Abbildung 5: Low-, mid- und high-level Features [45]. Die Komplexität wird mit jedem Convolutional Layer erhöht.....	12
Abbildung 6: Faltung eines Bildes [27]. Der Kernel wird über den Input tensor geschoben und an jeder Stelle wird das Elementweise Produkt aufsummiert und in eine Feature Map geschrieben.....	13
Abbildung 7: Max-Pooling mit einem 2x2 Filter und 2x2 Stride [27]. Der Maximale Wert aus dem 2x2 Bereich wird in den Output übernommen.....	13
Abbildung 8: Beispiele von Lungeknoten (nodules) und Nicht-Lungenknoten (non-nodules) [28] aus den verwendeten Datasets.....	14
Abbildung 9: Sequenzieren des Kandidaten [28], der aus mehreren Perspektiven untersucht wird. ....	14
Abbildung 10: Schritte vom rohen Filmmaterial zu einer einzelnen Phiole. Links ist das rohe Video zusehen, in der Mitte der zentrierte Shaker und rechts eine ausgeschnittene Phiole. ...	16
Abbildung 11: Beispielbilder aller Klassen.....	16
Abbildung 12: Beispiel eines vollständig augmentierten Bildes. Ein Originalbild wird durch Spiegelung und einen Average Filter um fünf Bilder erweitert. ....	18
Abbildung 13: Vergrößerung eines Tröpfchens zur Veranschaulichung des 3x3 Average Filter .....	18
Abbildung 14: Architektur des ersten CNN. Die ersten drei gelben Layer sind Convolutional Layer und die letzten beiden sind Dense Layer. Der letzte Layer ist der Output Layer. ....	21
Abbildung 15: Konzeptionelles Diagramm des Preprocessing. Es wird die Verarbeitung der Daten von den rohen Bildern zum verwendbaren Input X sowie die Verarbeitung der Annotationen zur One-Hot-Kodierung beschrieben. ....	22
Abbildung 16: Split des Dataset in Trainings-, Validations- und Testdaten [27].....	23
Abbildung 17: Beispiel der Loss Function bei einem Training, bei dem es zu Overfitting kommt. Zu sehen ist dies am Validation Error, der wieder ansteigt und vom Training Error divergiert [27].....	24
Abbildung 18: Beispiel einer Konfusionsmatrix für ein Multi-Label Klassifizierungsproblem. Das Beispiel zeigt die Bereiche, die für die Klasse Drop als TP, FP, TN und FN gelten. ....	26
Abbildung 19: GUI zur Auswertung von beliebigen Videos. ....	28
Abbildung 20: Verlauf des Validation Loss beim Training von Modellen mit und ohne Data Augmentation. Die volle Data Augmentation erreicht den tiefsten Loss. ....	30
Abbildung 21: Verlauf von Recall und Accuracy beim Training von Modellen mit und ohne Data Augmentation. Das Modell mit voller Data Augmentation erreicht die besten Werte für Recall und Accuracy.....	31



Abbildung 22: Verlauf von Validation Loss beim Training von Modellen mit unterschiedlichen Verteilungen.....	32
Abbildung 23: Verlauf von Recall und Accuracy beim Training von Modellen mit unterschiedlichen Verteilungen.....	32
Abbildung 24: Verlauf des Validation Loss während dem Training mit unterschiedlichen Batchsizes. 64 und 128 schneiden besser ab als 32. ....	34
Abbildung 25: Verlauf von Recall und Accuracy während dem Training mit unterschiedlichen Batchsizes. Auch hier schneiden 64 und 128 besser ab als 32.....	34
Abbildung 26: Der Verlauf des Validation Loss bei verschiedenen Kernel Grössen. 3x3 und 5x5 ergeben weitaus bessere Ergebnisse als 7x7 Kernels. ....	35
Abbildung 27: Verlauf von Recall und Accuracy bei verschiedenen Kernel Grössen mit einem Dense Layer und 128 Neuronen. Auch hier ist das beste Ergebnis bei 3x3 zu sehen. ....	35
Abbildung 28: Vergleich von verschiedenen Kernel-Grössen, diesmal mit einem etwas grösseren Netzwerk aus 2 Dense Layer und 128 respektive 256 Neuronen. 5x5 ist diesmal besser als 3x3.....	36
Abbildung 29: Verlauf der Loss Funktion bei Modellen mit verschiedener Anzahl Kernels pro Layer. Die Legende gibt die Anzahl Kernel in aufsteigender Reihenfolge an. 16-32-64 erreicht den minimalen Wert. ....	37
Abbildung 30: Verlauf von Recall und Accuracy. Gegenüber der Loss-Funktion erreicht das Netzwerk mit 64-96-128 Kernels hier die besten Werte. ....	37
Abbildung 31: Verlauf des Validation Loss bei Netzwerken mit 1-3 Dense Layer. Die Modelle auf der linken Seite haben 64 Neuronen pro Layer und die Modelle auf der rechten Seite 128 Neuronen.....	38
Abbildung 32: Verlauf von Recall und Accuracy bei Netzwerken mit 1-3 Dense Layer. Es ist zu sehen, dass das Netzwerk mit 1x128 (ein Dense Layer an 128 Neuronen) die besten Werte erzielt. ....	38
Abbildung 33: Verlauf des Validation Loss bei sieben verschiedenen Variationen von Anzahl Layer und Neuronen. Ein Layer mit 128 Neuronen schneidet immer noch am besten ab. ....	39
Abbildung 34: Beispiel eines Foam Frame in Video 1.....	41
Abbildung 35: Verteilung der Klassifizierten Frames in Video 1. Alle Modelle erkennen nur Foam Frames. ....	41
Abbildung 36: Verteilung der klassifizierten Frames in Video 2 von den drei Modellen. ....	42
Abbildung 37: Verteilung der klassifizierten Frames in Video 3 von den drei Modellen. ....	43
Abbildung 38: Verteilung der klassifizierten Frames in Video 4 von den drei Modellen .....	45

## 6.4 Tabellenverzeichnis

Tabelle 1: Verteilung der annotierten Daten im erstellten Dataset .....	17
Tabelle 2: Beispielfverteilung eins fiktiven Datasets.....	19
Tabelle 3: Beispieltgewichtung für das fiktive Dataset .....	19
Tabelle 4: Erhaltenes Dataset nach Balancierung.....	19
Tabelle 5: Beziehung von vorhergesagtem Output zu wahrem Output.....	26
Tabelle 6: Vergleich der Metriken für die trainierten Modelle mit und ohne Data Augmentation.....	31
Tabelle 7: Gewichtung für die Balancierung des Dataset .....	32
Tabelle 8: Vergleich der Metriken für die trainierten Modellen mit unterschiedlichen Verteilungen.....	33
Tabelle 9: Konfusionsmatrix des Modells mit einem unausgeglichenen Dataset.....	33
Tabelle 10: Vergleich der Metriken für die trainierten Modelle mit unterschiedlicher Batchsize.....	34
Tabelle 11: Vergleich der Metriken für die trainierten Modelle mit unterschiedlicher Kernel- Grösse und einem Dense Layer und 128 Neuronen.....	35
Tabelle 12: Vergleich der Metriken für die trainierten Modelle mit unterschiedlicher Kernel- Grösse und zwei Dense Layer mit 128 und 256 Neuronen.....	36
Tabelle 13: Vergleich der Metriken für die trainierten Modelle mit einer unterschiedlichen Anzahl Neuronen in einem Dense Layer.....	37
Tabelle 14: Vergleich der Metriken für die trainierten Modelle mit einer unterschiedlichen Anzahl Dense Layer mit jeweils 64 respektive 128 Neuronen pro Layer.....	39
Tabelle 15: Vergleich der Metriken für trainierte Modelle mit unterschiedlichen V ariationen von Anzahl Layer und Neuronen.....	40
Tabelle 16: Vergleich der Metriken von trainierten Modellen mit unterschiedlichen Architekturen. Das ResNet hat den besten FBeta-Score.....	40
Tabelle 17: Ausgewählte Modelle zum Testen mit neuen Videos.....	41
Tabelle 18: Beispiele der Klassifizierung von Video 2 mit dem LeNet.....	42
Tabelle 19: Beispiele der Klassifizierung von Video 2 mit dem Multi-Conv.....	42
Tabelle 20: Beispiele der Klassifizierung von Video 2 mit dem ResNet.....	43
Tabelle 21: Beispiele der Klassifizierung von Video 3 mit dem LeNet.....	44
Tabelle 22: Beispiele der Klassifizierung von Video 3 mit dem Multi-Conv.....	44
Tabelle 23: Beispiele der Klassifizierung von Video 3 mit dem ResNet.....	44
Tabelle 24: Beispiele der Klassifizierung von Video 4 mit dem LeNet.....	45
Tabelle 25: Beispiele der Klassifizierung von Video 4 mit dem Multi-Conv.....	46
Tabelle 26: Beispiele der Klassifizierung von Video 4 mit dem ResNet.....	46

# 7. Anhang

## 7.1 Kontaktinformation und Code


Für allfällige Fragen bezüglich dieser Arbeit kontaktieren Sie uns via:

- [wongw@outlook.com](mailto:wongw@outlook.com)
- [sven.a@hotmail.com](mailto:sven.a@hotmail.com)

Sämtlicher Softwarecode ist unter folgendem Link verfügbar:

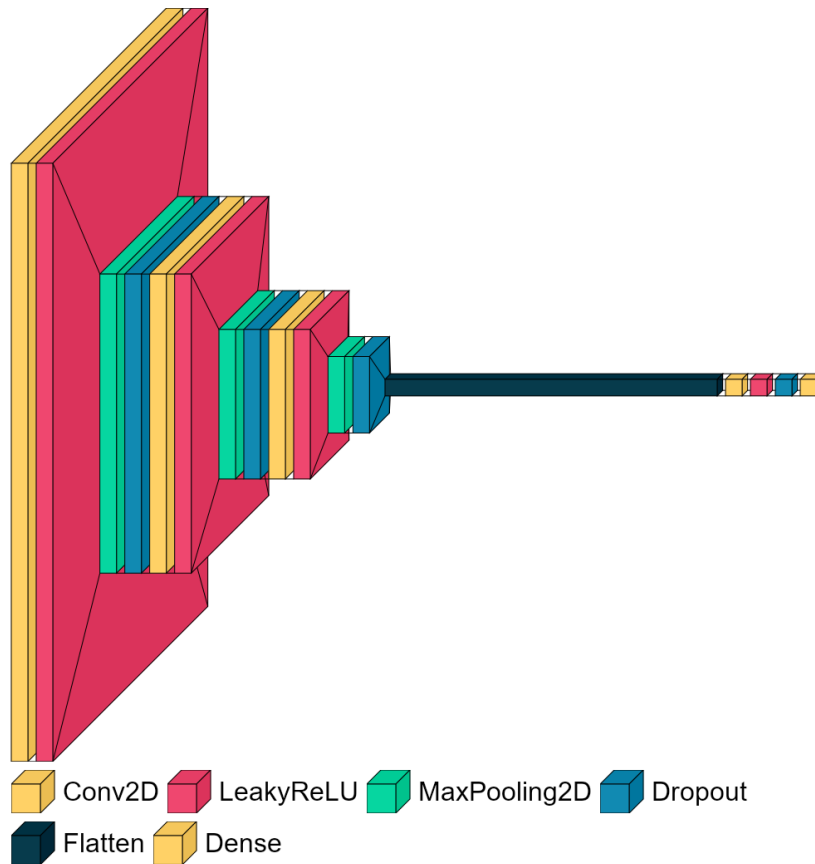
- [https://github.zhaw.ch/wongwil1/BA\\_fluid](https://github.zhaw.ch/wongwil1/BA_fluid)

## 7.2 Offizielle Aufgabenstellung

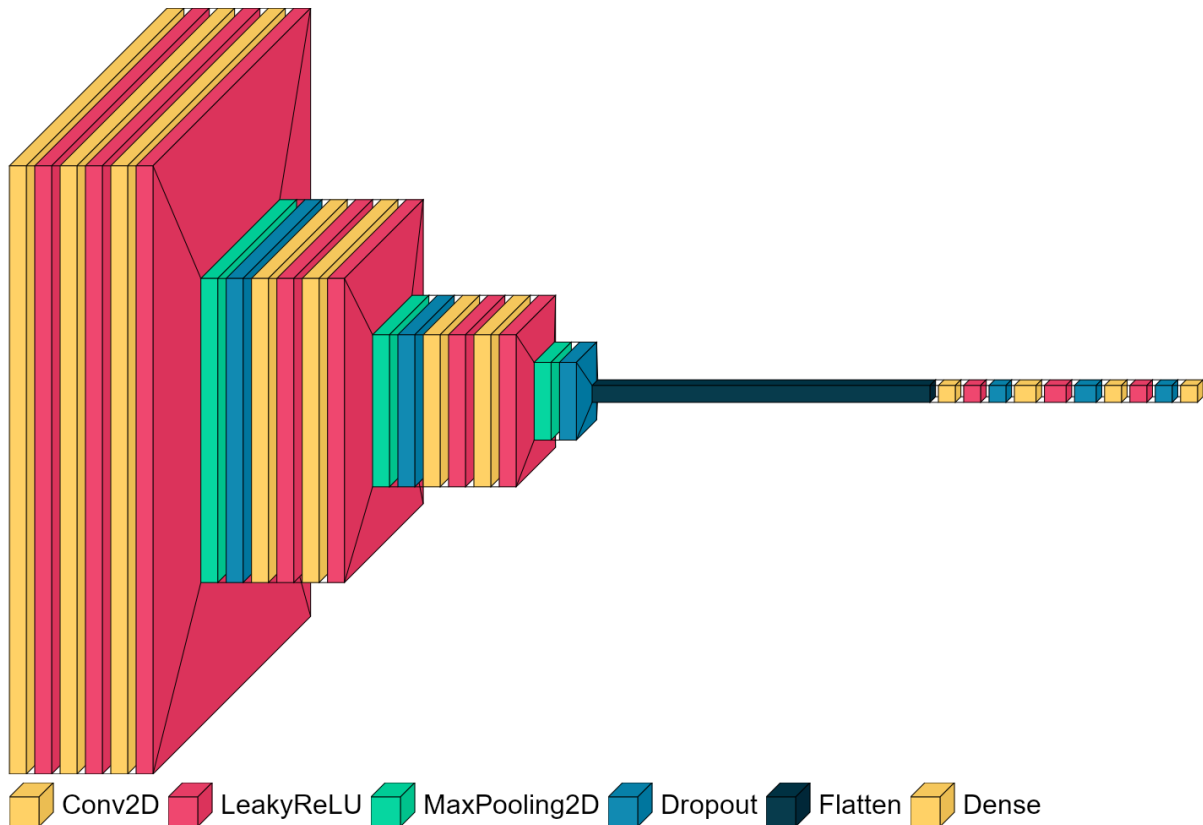
Bachelorarbeit 2021 - FS: BA21_boig_08	
<b>Allgemeines:</b>	
<b>Titel:</b>	Bildverarbeitung von Fluidbewegungen im Pharmabereich mittels KI
<b>Anzahl Studierende:</b>	2
<b>Durchführung in Englisch möglich:</b>	Nein
<b>Betreuer:</b>	<b>Zugeteilte Studenten:</b>
<b>HauptbetreuerIn:</b> Gernot Kurt Boiger, boig 	Diese Arbeit ist zugeteilt an: - Sven Aebersold, aebersve (IT) - William Wong, wongwil1 (IT)
<b>Fachgebiet:</b>	<b>Studiengänge:</b>
BV      Bildverarbeitung CP      Computational Physics DA      Datenanalyse	ET      Elektrotechnik IT      Informatik MT      Maschinentechnik ST      Systemtechnik
<b>Zuordnung der Arbeit :</b>	<b>Infrastruktur:</b>
ICP      Institute of Computational Physics	benötigt keinen zugeteilten Arbeitsplatz an der ZHAW
<b>Interne Partner :</b>	<b>Industriepartner:</b>
Es wurde kein interner Partner definiert!	Es wurden keine Industriepartner definiert!
<b>Beschreibung:</b>	
Beim Transport von Medikamenten in Flüssigform wird das Fluid in Bewegung versetzt, wodurch die aktive Substanz im Medikament zerstört werden kann. Dieser Effekt wird im Vorfeld im Labor untersucht. Im Rahmen der vorgestellten Bachelorarbeit soll ein Programm entwickelt werden, um die Auswertung von Hochgeschwindigkeitsaufnahmen dieser Experimente zu automatisieren. Dabei sollen Effekte wie Oberflächenmuster, Tröpfchen- oder Schaumbildung mittels KI und Model Training detektiert und wenn möglich charakterisiert werden.	

## 7.3 Verwendete CNN Architekturen

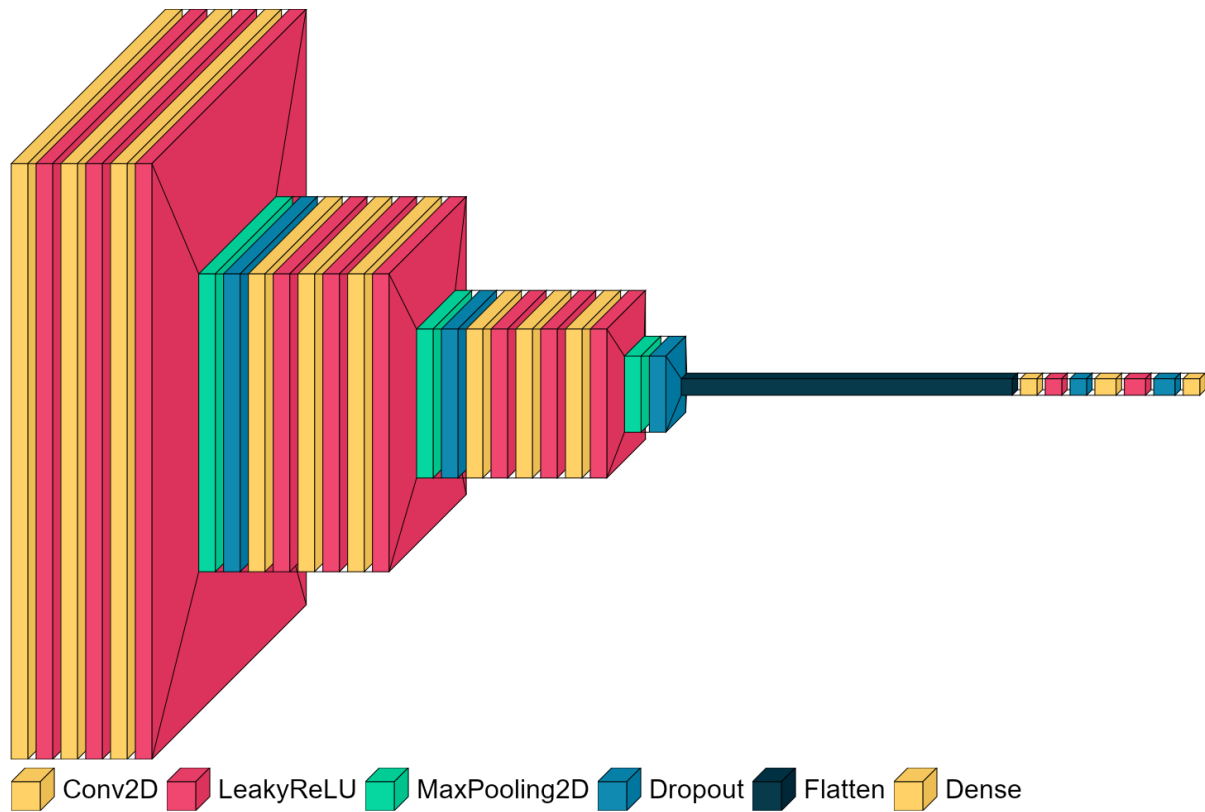
### LeNet



### Multi-Convolutional



## Bottleneck



## ResNet

