HTL - Perg
Höhere Abteilung für Informatik

Diplomarbeit

Secure Cam

Das bilderanalysierende Securitysystem

Projektteam: Michael Lengauer
Mario Lischka
Projektbetreuer: Prof. Dipl.-Ing. Michael Stumpfl

In Zusammenarbeit mit Institute of Bioinformatics, Johannes Kepler University Linz
Betreuer Herr Assoz.Univ.-Prof. Dr. Ulrich Bodenhofer

Bearbeitungszeitraum: 01.10.2016 – 5.04.2017

Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Perg, _____ Unterschrift _____

(Michael Lengauer)

Perg, _____ Unterschrift _____

(Mario Lischka)

Danksagung

Besonders danken möchten wir an diesem Punkt, dem Institut für Bioinformatik der JKU und hier im besonderen Dr. Ulrich Bodenhofer, der die Idee für die Arbeit hatte und uns mit Ressourcen, wie dem Jetson TX1 aber auch mit Quellen und praktischer Hilfe unterstützte. Sehr hilfreich waren auch die zwei Wochen, in denen wir im Institut an unserer Diplomarbeit arbeiten durften. Sehr herzlich möchten wir auch Dipl.-Ing. Michael Stumpfl für seine Betreuung und Unterstützung bei der Diplomarbeit danken.

Kurzfassung

Gegenstand der hier vorgestellten Diplomarbeit, war es, ein Security System zu realisieren, das mithilfe eines Neuronalen Netzes, Bilder aus beliebigen Videoquellen (Web-, IP-Kameras usw.) analysiert und Sicherheitsverstöße durch Menschen erkennt.

Zur Verwaltung, Steuerung und Visueller Präsentation des Security Systems, dient eine Web Applikation. Die Web Applikation zeigt Video Streams der Analysierten Bilder, schlüsselt verschiedene Daten aus geschehenen Vorfällen in Statistiken auf und ermöglicht es dem Benutzer Einstellungen am Security System vorzunehmen.

Die Diplomarbeit wurde mit Hilfe der Programmiersprache Python und darauf basierenden Frameworks realisiert.

Abstract

The object of our diploma thesis, was to realize a security system, which, with the help of neural network, can analyse pictures from any video source and detects security violations through humans.

A web application is used to manage, control and visualize the security system. The web application shows video streams of the analysed images, encapsulates various data from incidents in statistics, and allows the user to make settings on the security system.

The diploma project was programmed in Python, with the help of the underlying frameworks.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

n. N.	neuronales Netwerk
z. B.	zum Beispiel
GB	Gigabyte
CNN	Convolutional Neural Network
ZMQ	ZeroMQ
bzw.	beziehungsweise
Abb.	Abbildung

1 Einleitung

1.1 Ausgangssituation

Im März 2016 las Dr. Ulrich Bodenhofer, assoziierter Universitätsprofessor an der JKU am Institut für Bioinformatik, in der Tageszeitung Der Standard einen Artikel über zwei St. Pöltner Studenten, die ein Programm für den sehr verbreiteten Einplatinenrechner Raspberry Pi entwickelt haben. Mithilfe dieses Programms, Sec Pi genannt, lässt sich mit verschiedenen Sensoren, wie Bewegungsmeldern oder Temperatursensoren, die am Raspberry Pi angeschlossen werden ein Sicherheitssystem Marke Eigenbau realisieren.[1] Da am Institut für Bioinformatik auch viel mit neuronalen Netzen gearbeitet wird, kam Dr. Ulrich Bodenhofer die Idee eines Securitysystems, das nicht mithilfe von Sensoren, sondern durch die Analyse von Bildern mittels eines neuronalen Netzes Eindringlinge erkennt.

1.2 Problemstellung

Herkömmliche Sicherheitssysteme, die meistens mit Bewegungssensoren Eindringlinge erkennen, haben den großen Nachteil nicht zwischen Menschen und Haustieren unterscheiden zu können. Ein neuronales Netz kann dazu genutzt werden, auf Bildern, mithilfe der sogenannten Object Detection, Menschen zu erkennen.

1.3 Zielsetzung

Ziel der Diplomarbeit war es, zwei in Python geschrieben Programme zu erstellen. Ein Programm, in dem ein neuronales Netzwerk mit Bilddaten von verschiedenen Bildquellen gefüttert wird und die daraus gewonnen Daten ausgewertet und weiterverarbeitet werden.

Und ein Programm, das, mithilfe des Python Frameworks Django, eine Website für den Nutzer zur Verfügung stellt, auf der er sich einloggen, live Bilder und Bilder aus einem Archiv betrachten und generelle Einstellungen am Sicherheitssystem vornehmen kann.

2 Künstliche neuronale Netze Grundlagen

Obwohl an künstlichen neuronalen Netzen schon seit den 1940er Jahren geforscht und gearbeitet wird und sie somit schon ganz am Anfang der Informatik dabei waren, sind sie bis heute außerhalb von Informatikerkreisen eher unbekannt. Aber auch unter Informatikern wurde ihr Potential lange Zeit unterschätzt. Warum das so war und wie künstliche neuronale Netze grundsätzlich funktionieren wird in diesem Kapitel geklärt.

2.1 Geschichte

Die Geschichte der künstlichen neuronalen Netze beginnt für Informatische Techniken schon sehr früh, nämlich **1943**. Damals haben McCulloch und Walter Pitts das erste Mal die Funktion und den Aufbau eines künstlichen neuronalen Netzes beschrieben. [2] Um das in einen Kontext zu setzen, erst zwei Jahre zuvor hatte Konrad Zuse mit dem Z3 den ersten funktionsfähigen Digitalrechner gebaut. In den folgenden Jahren wurden von vielen Forschern die Grundlagen zu neuronalen Netzen geschaffen. **1947** stellten Warren McCulloch und Walter Pitts das erste theoretische Anwendungsbeispiel zur räumlichen Mustererkennung vor.[3] **1958** wurde von Frank Rosenblatt der erste Neurocomputer, der Mark I Perzeptron entwickelt, dieser konnte einfache Ziffern mit einem Bildsensor erkennen. **1959** stellte Rosenblatt mit dem Perzeptron eines der Grundmodelle der neuronalen Netze vor. Bis **1969** hatten neuronale Netze so ihre erste Blütezeit, in der viele Forscher an vielen Universitäten an ihnen arbeiteten, bis Marvin Minsky und Seymour Papert, ebenfalls Forscher auf dem Gebiet der Künstlichen Intelligenz, mit einer Arbeit, in der sie bewiesen, dass ein einlagiges Perzeptron den XOR-Operator nicht auflösen kann[4] und damit viele Probleme mit dem Model nicht gelöst werden können, fast die gesamten Forschungstätigkeiten auf dem Gebiet zum Erliegen brachten. Obwohl Rosenblatt schon gezeigt hatte das logische Operatoren zwar nicht mit einlagigen Perzeptrons, aber sehr wohl mit mehrlagigen Perzeptrons beschrieben werden können, konnte der Niedergang fast des gesamten Forschungsbereichs nicht mehr verhindert werden, da Rosenblatt verstarb und sich sonst keiner dafür einsetzte.

Erst Anfang bis Mitte der **1980er** Jahre wurde das Gebiet, federführend durch John Hopfield, wieder zum Leben erweckt. Es erfährt seither ungebrochene Popularität und ist aus der heutigen Künstlichen Intelligenz nicht mehr wegzudenken.

[5]

2.2 Theorie

2.2.1 Grundbegriffe

2.2.1.1 Neuronen

Neuronen sind die Knotenpunkte des neuronalen Netzes, sie erhalten von anderen Neuronen Eingabewerte und verarbeiten diese in Funktionen weiter um einen Ausgabewert zu generieren den sie weitergeben.

2.2.1.2 Input-Neuronen

Die Input-Neuronen oder zu Deutsch Eingabeneuronen sind spezielle Neuronen, die Werte, die in das neuronale Netz eingegeben werden sollen aufnehmen und weitergeben. Sie bilden die oberste Schicht des neuronalen Netzes.

2.2.1.3 Output-Neuronen

Die Output-Neuronen oder zu Deutsch Ausgabeneuronen sind spezielle Neuronen die Werte die aus dem neuronalen Netz ausgegeben werden ausgeben. Sie bilden die unterste Schicht eines neuronalen Netzes.

2.2.1.4 Gewichte

Um die Wichtigkeit und den Einfluss, den ein Neuron auf ein nachfolgendes Neuron hat abbilden zu können, gibt es immer zwischen zwei Neuronen ein Gewicht, das oft mit dem Ausgabewert des ausgebenden Neurons multipliziert wird und so den Eingabewert des anderen Neurons beeinflusst.

2.2.1.5 Schwellenwert

Der Schwellenwert ist jener Wert für ein Neuron, bei dem ein Neuron, je nach verwendeter Funktion, entweder überhaupt Ausgaben produziert oder die stärksten Ausgaben ausgibt. In der Praxis wird er meist mithilfe eines zusätzlichen Neurons des sogenannten Biasneuron realisiert.

2.2.1.6 Biasneuron

Das Biasneuron gibt immer 1 aus und wird mit dem negativen Schwellenwert gewichtet. Es bietet vor allem beim Training erhebliche Vorteile.

2.2.1.7 Neuronen Funktionen

Prinzipiell wird ein Neuron von seinen Funktionen bestimmt. Generell kann ein Neuron drei Funktionen besitzen.

Propagierungsfunktion

Die Propagierungsfunktion ist die Funktion, die die Eingaben für ein Neuron entgegennimmt und gewichtet. Sie gibt ihre Ausgaben an die Aktivierungsfunktion weiter.

Aktivierungsfunktion

Ist die zentrale Funktion eines Neurons und bestimmt seine Ausgabe am meisten. Für die Aktivierungsfunktion können viele verschiedene Funktionstypen verwendet werden. Sehr beliebt sind Schwellenwertfunktonen, Stückweise lineare Funktionen oder Sigmoide Funktionen mit Steigungsmaßen.

Ausgabefunktionen

Manchmal wird die tatsächliche Ausgabe eines Neurons noch einmal durch eine Ausgabefunktion beeinflusst, meistens ist jedoch das Ergebnis der Aktivierungsfunktion auch die Ausgabe eines Neurons.

2.2.2 Perzeptron

Wie schon bei der Geschichte der neuronalen Netze angemerkt ist eines der Standardmodelle für neuronale Netze das Perzeptron. Es wurde in den 1950er und 60er Jahren von Frank Rosenblatt entwickelt.

2.2.2.1 Einschichtiges Perzeptron

Das einschichtige Perzeptron hat wie sein Name schon sagt nur ein Neuron das Berechnungen durchführt. Man spricht dabei auch von nur einer Schicht versteckter Neuronen. Das einschichtige Perzeptron ist grundsätzlich aus denselben Bestandteilen wie ein Netz aus mehreren Neuronen aufgebaut. Es besitzt Eingabeneuronen, Gewichte, einen Schwellenwert, eine Aktivierungsfunktion und ein Ausgabeneuron.

Das einschichtige Perzeptron kann grundsätzlich nur sehr einfache Probleme lösen. Dazu gehören jedoch die Logikgatter AND, OR und NOT. Die Abbildung 1 zeigt ein einlagiges Perzeptron das ein Oder-Gatter realisiert.

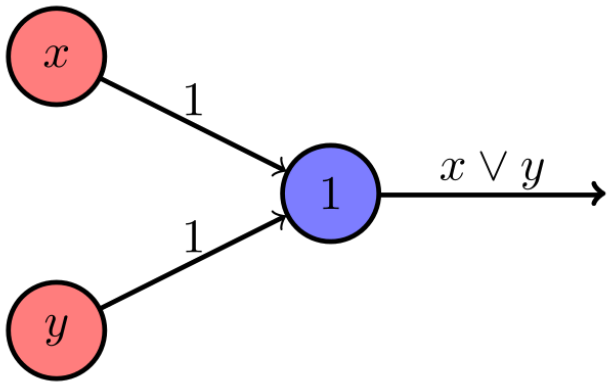

Abbildung 1: einlagiges Perzeptron[6]

2.2.3 Trainieren eines neuronalen Netzwerks

Dieser Punkt soll einen kleinen Einblick in das Trainieren von Neuronalen Netzen geben.

2.2.3.1 Allgemein

Grundsätzlich lernen neuronale Netze immer aus ihren Fehlern. Damit man auch weiß, wann ein neuronales Netzwerk einen Fehler begangen hat, muss man die gewünschte Ausgabe für seine Trainingsdaten kennen. Es gibt viele verschiedene sogenannte Lernregeln, die definieren, wie sich das neuronale Netz verändert, um die Daten besser analysieren zu können.

2.2.3.2 Perzeptron-Lernalgorithmus

Beim einschichtigen Perzeptron ist natürlich auch das Lernverfahren sehr einfach, es gibt aber einen guten Einblick, wie ein einzelnes Neuron trainiert wird. Der Prezeptron-Lernalgorithmus trainiert die Gewichte der Eingabeneuronen.

- Solange noch Trainingsdaten vorhanden sind und der Fehler noch zu groß ist, wird trainiert
- Ein Trainingsmuster wird in das Pezeptron eingegeben
- Danach wird die Ausgabe des Neurons überprüft
    - Ist die Ausgabe gleich der gewünschten Ausgabe geschieht nichts
    - Ist sie das nicht gibt es beim Pezeptron, das als Aktivierungsfunktion eine Schwellenwertfunktion nutzt, nur die Möglichkeiten
        - Ausgabe ist 0 sollte 1 sein
            - Werden die Gewichte der Eingaben erhöht
        - Ausgabe ist 1 sollte 0 sein
            - Werden die Gewichte der Eingaben verringert
- Danach beginnt der Kreislauf wieder von vorne

2.3 Eingesetzte neuronale Netzwerk Technologien

2.3.1 Caffe

Caffe ist ein Framework, mithilfe dessen man Neuronale Netze für die Objekterkennung trainieren und trainierte Netze nutzen kann. Es bietet Schnittstellen für C++, Python und MATHLAB. Entwickelt wurde das Framework von Yangqing Jia am Berkeley Berkeley Vision and Learning Center im Rahmen seiner Doktorarbeit(Phd).[7]

2.3.2 CNN Convolutional Neural Network

Convolutional Neural Networks oder zu Deutsch faltende neurale Netzwerke sind eine spezielle Form der n. N. die besonders für die Bildobjekterkennung extreme Vorteile mit sich bringt. Da es Informationen aus mehreren Pixeln zusammenfasst und unnütze Bildinformationen verwirft, kann es sehr viel effizienter arbeiten und braucht vor allem sehr viel weniger Speicherplatz wie ein vergleichbares n. N. ohne die speziellen Techniken des CNNs.

2.3.3 R-CNN Region-based Convolutional Networks

R-CNN zu Deutsch Regionen basiertes faltendes neurales Netzwerk ist ein an der Berkeley Universität von Ross Girshick, Jeff Donahue, Trevor Darrell und Jitendra Malik entwickeltes Objekterkennungssystem, das es ermöglicht nicht nur ganze Bilder zu analysieren sondern auch Objekte in Bildern zu lokalisieren und zu erkennen.

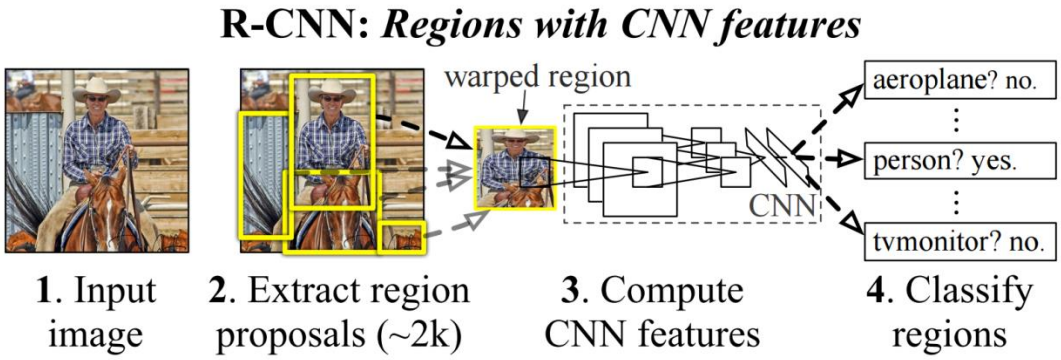

Abbildung 2: R-CNN [8]

2.3.4 ZF Net

ZF ist ein von Matthew D Zeiler und Rob Fergus entwickeltes Convolutional Neural Network, das besonders durch seine Kompaktheit brilliert und damit auch auf System mit wenig Arbeitsspeicher ausführbar ist. [11][11][11]

2.3.5 Faster R-CNN

Faster R-CNN ist einer der Nachfolger des R-CNN Systems und wurde im Rahmen des Microsoft Research Projekts von Shaoqing Ren, Kaiming He, Ross Girshick und Jian Sun entwickelt. Es verbessert die Zusammenarbeit der Objektsregionserkennung und der Objekterkennung extrem und ermöglicht so, mit vergleichbar geringen Hardwareanforderungen, eine Echtzeitobjekterkennung und Visualisierung.[9]

Abbildung 3: Faster R-CNN Beispielerkennung[10]

2.3.6 CUDA

Ist eine Technologie von NVIDIA die es Programmen erlaubt, Berechnungen hochgradig paralleler Anwendungen auf den Kernen eines NVIDIA Grafikprozessors auszuführen Das beschleunigt diese Anwendungen um ein Vielfaches aufgrund der hohen Anzahl an Kernen moderner Grafikprozessoren.[12]

2.3.7 cuDNN

Die NVIDIA CUDA® Deep Neural Network library (cuDNN) ist eine neuronale Netzwerk Bibliothek, die das Ausführen dieser auf NVIDIA Grafikprozessoren noch einmal um ein Vielfaches beschleunigt.[13]

3 Technische Grundlagen

3.1 Generelle Technische Grundlagen

3.1.1 Open Source

Open Source ist eine Methode der Softwareveröffentlichung, bei der nicht nur die binär kodierten Dateien einer Anwendung bereitgestellt werden, sondern der gesamte Programmcode. Open Source Programme sind nicht zwingend kostenlos, jedoch sind sie das meistens, da ein Kopierschutz so natürlich nicht realisierbar ist.

Die wichtigsten Open Source Lizenzen sind die MIT Lizenz, die GNU General Public License (GPL) und die Apache License. [14]

3.2 Arbeitsumgebungen

Dieser Punkt behandelt die wichtigsten Arbeitswerkzeuge, mit denen der Praktische Teil der Arbeit realisiert wurde.

3.2.1 Ubuntu

Ubuntu ist ein auf dem Linux Kernel basierendes Betriebssystem, das besonders wegen seiner Benutzerfreundlichkeit geschätzt wird. Es ist laut der Website Linux Counter die meistgenutzte Linux-Distribution.[17] Ubuntu ist ein Open Source Projekt und wird von einer großen Entwickler Community unter Leitung der Canonical Ltd. weiterentwickelt wird.[18] Es unterstützt mit einer großen Anzahl an Treibern eine breite Masse an Hardware darunter auch

3.2.2 Git

Git ist ein Versionsverwaltungsprogramm mithilfe dessen der Fortschritt eines Softwareprojekts gesichert werden kann, wobei man jedoch auch immer wieder auf einen älteren Stand zurückgehen kann, ohne das Projekt wirklich mehrmals speichern zu müssen. Git ermöglicht einerseits eine lokale Versionsverwaltung, außerdem jedoch auch noch durch die Pull und Push Funktionen das Zusammenarbeiten mehrere Entwickler auf einem gemeinsamen Git Server. Git ist eine opensource Software und wurde von Linus Torvalds initiiert.

3.2.3 Jetson TX1 Developer KIT

Das Jetson TX1 Developer Kit ist eine Developer Kit der Firma NVIDA. Herzstück des KITs ist das namensgebende Jetson TX1 Module, ein circa scheckkartengroßer Rechner. Im Modul verbaut ist eine Quad-Core ARM Cortex-A57 CPU sowie 4GB LPDDR4 RAM und einer Maxwell GPU mit 256 Shader-Kernen. Außerdem ist das Modul noch mit 16GB eMMC Speicher ausgestattet.

Das Modul ist, in der TX1 Dev KIT Version, auf einem Anschlussbord im mini-ITX Formfaktor befestigt. Durch das Bord wird das System um eine große Anzahl an Anschlüssen erweitert.

Die wichtigsten davon sind:

USB 3.0 Type A

USB 2.0 Micro AB

HDMI

M.2 Key E

PCI-E x4

Gigabit Ethernet

SD-Kartenslott

Wi-Fi Modul mit Antennen

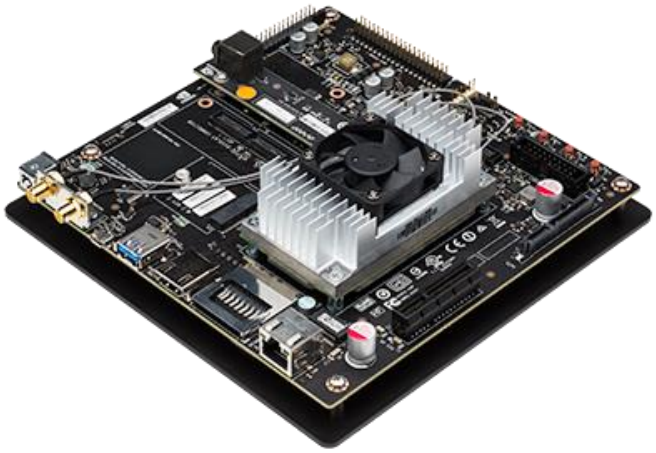

Abbildung 4: Jetson TX1

Auf dem System kann mit Hilfe der von NVIDIA zur Verfügung gestellten Software JetPack, Ubuntu in den Versionn v14.04 (32Bit) und 16.04(64Bit) installiert werden.

[15][16]

3.2.4 PyCharm

PyCharm ist eine Entwicklungsumgebung von Jetbrains, die zur Entwicklung von Python Programmen gedacht ist. Die Entwicklungsumgebung erleichtert die Entwicklung mit Python durch die Zurverfügungstellung zahlreicher Komfortfunktionen, wie einem intelligenten Editor, einem eingebauten Debugger, sowie Tools für die erleichterte Navigation und Umbenennung von Klassen, Methoden, Variablen usw. Die IDE wird in zwei Versionen angeboten der Community Edition und der Professional Edition, die Community Edition ist kostenlos, unterstützt einen dafür aber nicht bei der Web Entwicklung mit Python, auch für die Verwendung von Datenbanken werden einem keine Tools zur Verfügung gestellt.

[19]

3.3 Python

3.3.1 Threads

Python Threads funktionieren nicht wie Threads in den meisten anderen Programmiersprachen, in denen diese häufig zur parallelen Ausführung von Aufgaben genutzt werden. In Python verhindert eine solche parallele Ausführung von Code der GIL, der Global Interpreter Lock. Der GIL regelt, dass keine zwei Python Statements, eines Python Prozesses, gleichzeitig ausgeführt werden können. Daraus ergibt sich der große Nachteil, dass Python-Code selbst so nicht effizient auf mehreren CPU Kernen ausgeführt werden kann. Andererseits ergibt sich jedoch auch der Vorteil, dass man beim Programmieren von Python-Code nicht auf den gleichzeitigen Zugriff seiner gemeinsam genutzten Ressourcen achten muss.

Python Threads sind dennoch nicht nutzlos. Sie können sehr hilfreich sein, wenn man mehrere Code Teile hat, deren Hauptaufgabe es ist zu warten, zum Beispiel auf Eingaben von Kommunikationspartnern oder Subprozessen, die in anderen Programmiersprachen geschrieben sind. [20]

3.4 Python Frameworks

3.4.1 ZMQ

OpenCV ist eine open source Bildverarbeitungsbibliothek die über 500 Funktionen zur Verfügung stellt. Von Funktionen die einem den Zugriff auf WebCams ermöglichen über Funktionen zur Bildmanipulation bis hin zu Funktionen die mit neuronalen Netzen Objekte, wie Gesichter, erkennen können stellt einem die in C und C++ geschriebene Bibliothek alle Werkzeuge zur Verfügung um, ohne selbst das Rad neu erfinden zu müssen, komplexe Programme zu schreiben die Bilder analysieren können. Besonders durch die sehr effiziente Nutzung von Mehrkernprozessoren, und die damit einhergehende Geschwindigkeit, ist es auch für Echtzeitanwendungen nutzbar. Um OpenCV nicht nur mit C und C++ verwenden zu können gibt es für einige Programmiersprachen Schnittstellen für die Bibliothek, zu den unterstützten Sprachen zählen Python, Ruby, Matlab, Java und noch einige mehr. [22]

3.4.2 OpenCV

OpenCV ist eine open source Bildverarbeitungsbibliothek, die über 500 Funktionen zur Verfügung stellt. Von Funktionen, die einem den Zugriff auf WebCams ermöglichen über Funktionen zur Bildmanipulation bis hin zu Funktionen, die mit neuronalen Netzen Objekte, wie Gesichter, erkennen können, stellt einem die in C und C++ geschriebene Bibliothek alle Werkzeuge zur Verfügung um, ohne selbst das Rad neu erfinden zu müssen, komplexe Programme zu schreiben, die Bilder analysieren können. Besonders durch die sehr effiziente Nutzung von Mehrkernprozessoren, und die damit ein-

hergehende Geschwindigkeit, ist es auch für Echtzeitanwendungen nutzbar. Um OpenCV nicht nur mit C und C++ verwenden zu können, gibt es für einige Programmiersprachen Schnittstellen für die Bibliothek. Zu den unterstützten Sprachen zählen Python, Ruby, Matlab, Java und noch einige mehr. [22]

3.4.3 Django

Django ist ein open source Webframework, welches in Python geschrieben ist. Es verwendet das Model-View-Presenter-Schema. Django setzt nicht, wie andrere Web-Frameworks, auf implizite URL-Abbildung über Verzeichnisse und Funktionsnamen, sondern auf die explizite Konfiguration einer Anwendung. Das heißt, dass eine URL bei einer Anfrage nicht die Verzeichnisstruktur der gewollten Datei, sondern sie wird durch die „urls.py“-Datei der Anwendung zum richtigen Antwortgeben geleitet. Das Hauptziel ist, die Erleichterung bei der Erstellung von komplexen, datenbankgetriebenen Webseiten. Django betont die Wiederverwendbarkeit und Einbindungsmöglichkeit von Komponenten, schnelle Entwicklung und das Prinzip des sich nicht Wiederholens. Python wird hier überall verwendet, auch für Einstellungsdateien und Datenmodellen. Django bietet auch eine optionale administrative CRUD-Schnittstelle für den Verwalter an.

Im Kernframework implementiert:

- Ein eigenständiger Webserver, welcher komplett in Python geschrieben ist.
- Ein Template-System, das das Konzept der Vererbung aus der objektorientierten Programmierung übernimmt.
- Ein Caching-Framework, das eine von mehreren Cache-Methoden verwenden kann.
- Unterstützung von Middleware-Klassen, die in verschiedenen Stufen der Anforderungsbearbeitung eingreifen und benutzerdefinierte Funktionen ausführen können.
- Ein internes Dispatchersystem, das es Komponenten einer Applikation ermöglicht, Ereignisse über vordefinierte Signale miteinander zu kommunizieren.
- Ein Internationalisierungssystem, inklusive Übersetzung von Djangos eigenen Komponenten in eine Vielzahl von Sprachen.
- Ein System zur Erweiterung der Fähigkeit der Template Engine.
- Eine Schnittstelle zu Pythons eingebautem Unit Test Framework. [23]

4 Stand der Technik

Wenn man im Internet nach Alarmanlagen sucht, findet man eine breite Palette von preisgünstig bis zum Highendmodel. Doch die meisten davon sind nur mit Bewegungsmeldern ausgestattet. Also weiß die Anlage nur, ob sich eine Regung im abgesicherten Bereich tut. Etwas speziellere Sicherheitsvorrichtungen sind schon mit Kameras ausgestattet. Dies kann von Vorteil sein, um bei einem Alarm zu sehen ob es sich um einen Einbrecher oder einen Fehlalarm handelt. Wenn man nun z.B. Wandern ist und auf dem mobilen Gerät keinen Empfang hat um auf den Fehlalarm zu reagieren, bevor die Polizei oder Sicherheitsfirma anrückt, kann das ziemlich auf die Geldbörse gehen. Das Problem ist nun, dass nicht nur bei Haustieren, die sich im Haus aufhalten, sondern auch bei Pflanzen die Blätter verlieren, dieses Phänomen des Fehlalarms auftreten kann.

Um Fehlalarme zu umgehen, kann unser System Personen erkennen. Nur wenn eine Person von einer scharfgestellten Kamera erfasst wird, wird der Alarm ausgelöst.

5 Evaluierung

Da uns vom Projektinitiator Dr. Ulrich Bodenhofer zu Beginn des Projekts zwar eine klare Idee vorgestellt wurde, was die Anwendung nach Ihrer Fertigstellung können soll, aber nur grobe Vorschläge mit welchen Mitteln sie das erreichen soll, mussten wir zu Beginn des Projekts viele verschiedene Technologien gegeneinander abwägen und testen.

5.1 Neuronales Netzwerk

Gerade bei den Technologien rund um die Erkennung der Bilder durch ein neuronales Netzwerk, musste aufgrund der bei den Diplomarbeitsmitgliedern noch mangelnde Expertise, viel Zeit darauf verwendet werden, sich einen groben Überblick über das sehr komplexe Fachgebiet zu verschaffen. Durch die vorgegebene Hardware und die allgemeinen Anforderungen an das finale System waren folgende Punkte für die Auswahl der richtigen Technologien zu beachten.

- Geringer Ressourcenbedarf
    - Besonders der nur 4GB große RAM des Zielsystems Jetson TX1 stellt hier ein Limit dar.
- Hohe Geschwindigkeit der Erkennung
    - Da die Anwendung ein Sicherheitssystem darstellen soll, ist eine hohe Erkennungsfrequenz wichtig.
- Erkennung von Personen möglich
    - Die Fähigkeit des neuronalen Netzes eine eigene Klasse für die Erkennung von Personen zu besitzen ist sehr wichtig, da die Erkennung von Kleidung oder ähnlichen Objekten, die mit Personen in Verbindung zu bringen sind, nicht ausreichen würde.

5.1.1 Erste Versuche

Die ersten Versuche fanden mit dem neuronalen Netzwerkframework Caffe, dem neuronalen Netzwerk AlexNet und der in Caffe standardmäßig genutzten Objekt-Regionserkennung R-CNN statt.

Nachteile

AlexNet hatte den großen Nachteil, dass es keine eigene Klasse für die Erkennung von Personen besitzt. Außerdem benötigt es weit über 4 GB an Arbeitsspeicher, weshalb es nicht für den Jetson TX1 geeignet ist. Auch die Geschwindigkeit der Erkennung, durch dieses Setup ließ zu wünschen übrig.

5.1.2 Finale Lösung

Die Technologien, die final eingesetzt werden, sind zum einen das Faster R-CNN System, das auf das Caffe Framework aufbaut, mithilfe dessen vor allem die Objektpositionserkennung sehr viel effizienter gelöst wird. Für die Objekterkennung wird das neuronale Netzwerk ZF Net genutzt, das mit einem Arbeitsspeicherbedarf unter drei GB perfekt für den Jetson TX1 passt. Außerdem kommt es trotz seiner Kompaktheit auf ähnlich gute Erkennungsraten wie AlexNet.

5.2 Prozess Kommunikation

Die Kommunikation zwischen den Prozessen erfolgt über oben erwähnte ZMQ-Sockets. Diese haben den Vorteil, da sie keinen Speicherbereich zugeschrieben bekommen, wie bei Shared Memory. ZMQ muss nach lesen der Nachrichten diese nicht aus dem Speicherbereich entfernen, oder hat bei parallel Zugriff keine Kollisionen, da es über einen Buffer verfügt, indem alle Nachrichten der Reihe nach ausgelesen werden können. Wenn der Sender eine Nachricht sendet, und mehrere diese gleichzeitig Konsumieren wollen ist das kein Problem, da beide die Nachricht in ihrem Buffer haben. Dasselbe Problem würde auftreten, wenn man die Nachrichten im RAM-Verzeichnis „/dev/shm“ abspeichern würde.

5.3 Anzeige Technologie

Als Anzeige wird unsere Django Anwendung genutzt. Wir entschieden uns für Django, da wir sehr viel Input durch Bücher und im Netz dazu fanden und unser Projektbetreuer uns mit seinem Wissen über Django/Python tatkräftig unterstützen konnte. Diese schon angesprochene explizite Konfiguration einer Anwendung stellt sich für den Programmierer als hilfreiches Design heraus. Django lässt den Programmieren in die Webanwendung eintauchen, ohne viele Entscheidungen über die Infrastruktur der Anwendung zu tätigen. Die Django-Seite hat ebenfalls Tutorials für Python-Einsteiger, welche sehr gut dokumentiert sind.

6 Beschreibung der Anwendung

In diesem Kapitel wird die Anwendung, die den Kern dieser Arbeit darstellt ausführlich beschrieben, wobei versucht wurde auf die wichtigsten Codeteile besonderes Augenmerk zu werfen.

6.1 Gesamtsystem

Die Anwendung ist prinzipiell in zwei Python Projekte unterteilt. Eines um die Bilderkennung mittels neuronalem Netz abzuwickeln und eines das sich um die Anzeige auf einer, durch das Python Framework Django realisierten, Webapplikation kümmert.

Der Aufbau wird sehr gut durch das in Abbildung 5 und Abbildung 6 dargestellte Datenflussdiagramm ersichtlich, bei der aus Gründen der Übersicht der Steuerungsdatenfluss vernachlässigt wird.

Beide Prozesse arbeiten voneinander getrennt. Der Datenaustausch zwischen den beiden Prozessen findet einerseits über ZMQ Sockets und dem IPC Protokoll, wie zum Beispiel bei der Übertragung des Livebildes statt, andererseits jedoch auch über das Dateisystem. Über das Dateisystem werden jedoch nur die Bilder für das Archiv ausgetauscht, indem die Neuronale Netzwerk Anwendung die Bilder im Dateisystem speichert und dann der Django Webapplikation darüber Bescheid gibt. Die Webapplikation lädt sich die Bilder dann bei Bedarf wieder aus dem Dateisystem.

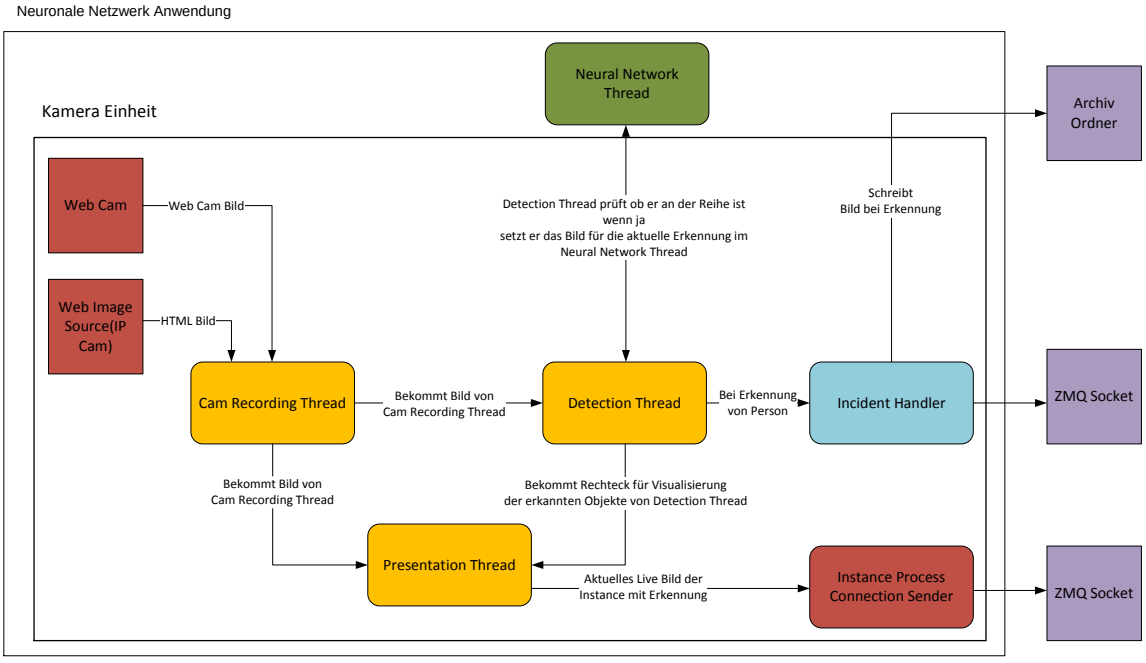

Abbildung 5: Neuronale Netzwerk Anwendung Datenflussdiagramm

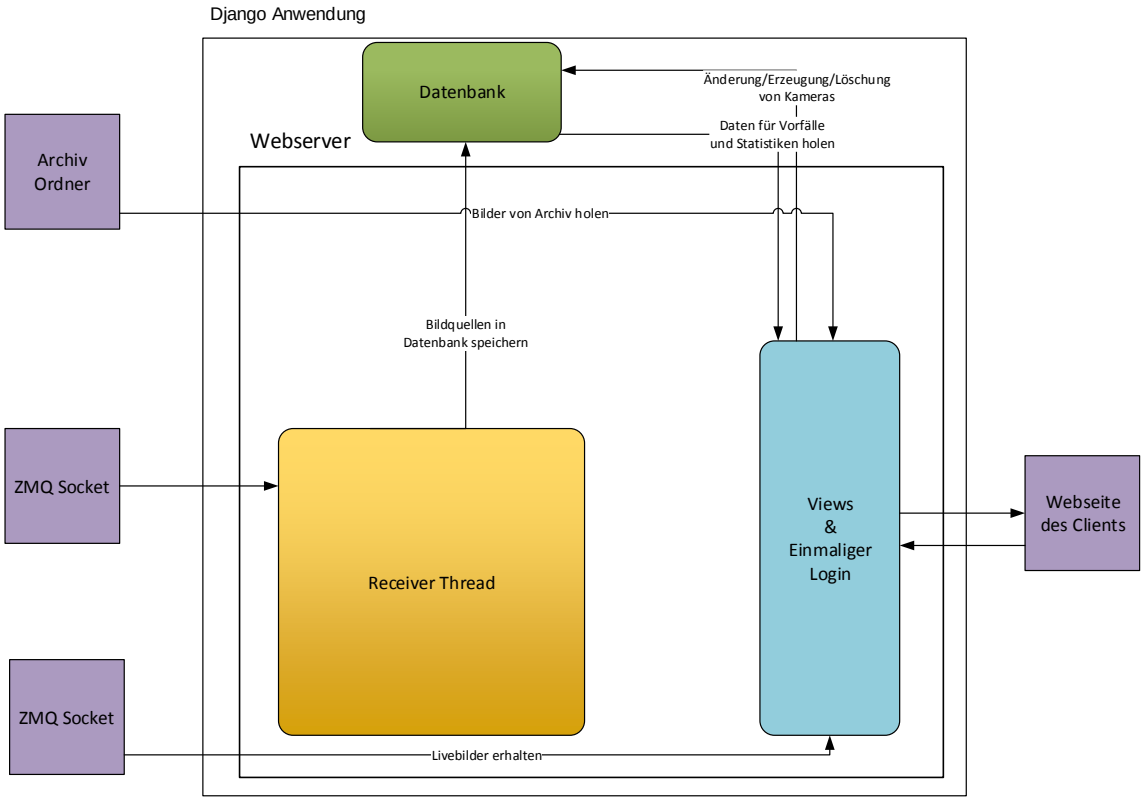

Abbildung 6: Django Webanwendung Datenflussdiagramm

6.2 Neuronale Netzwerk Anwendung

Dieser Punkt beschreibt die neuronale Netzwerk Anwendung.

6.2.1 Abstrakte Beschreibung

Die Aufgabe der neuronalen Netzwerk Anwendung besteht darin, die Verbindung zwischen Neuronalem Netzwerk, den Kameras und der Django Webapplikation herzustellen.

Beim Start der Anwendung wird einen Thread gestartet, der mithilfe des Caffe Frameworks das Neuronale Netz ZF Net in der Faster R-CNN Umgebung startet. Daraufhin startet die Anwendung einen weiteren Thread, der für die Kommunikation mit der Webapplikation zuständig ist. Dieser wartet nun zunächst auf den Befehl eine neue Kamera ins System einzubinden. Wenn dieser eintrifft, wird ein neues „Instance“ Objekt angelegt, das bei seiner Initialisierung wiederum drei Threads startet. Einen der für die Verbindung mit der Kamera verantwortlich ist, einen der für die Erkennung von Personen mithilfe des neuronalen Netzes zuständig ist und einen der die Weitergabe der Live-Bilddaten der Kameras, mit grafischer Visualisierung von erkannten Personen, an die Webapplikation abwickelt.

6.2.2 Genereller Aufbau

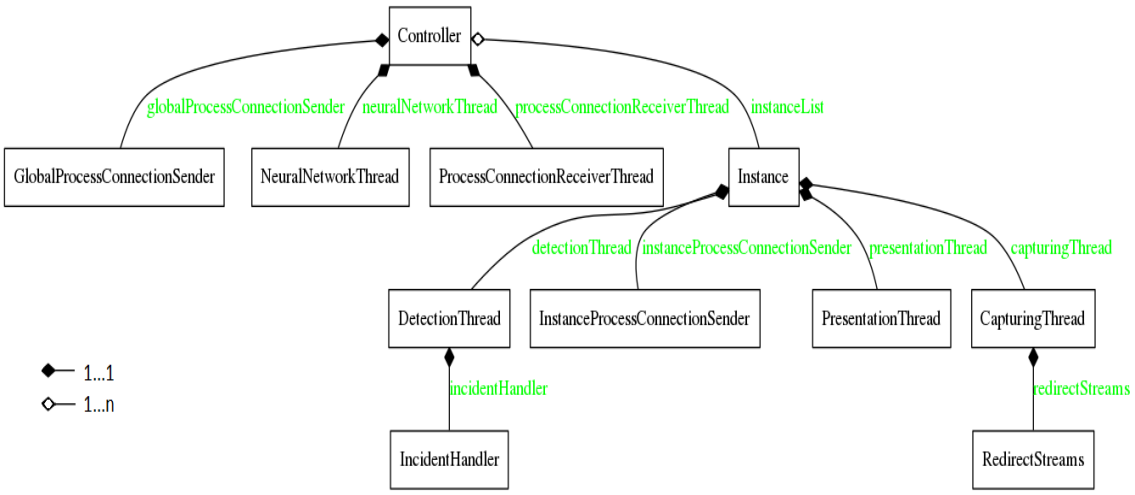

Abbildung 7: Neuronale Netzwerk Anwendung Klassendiagramm

Wie im Klassendiagram der neuronalen Netz Anwendung Abbildung 7 erkennbar ist, ist die Funktionalität der Anwendung in elf Klassen unterteilt.

6.2.2.1 Controller

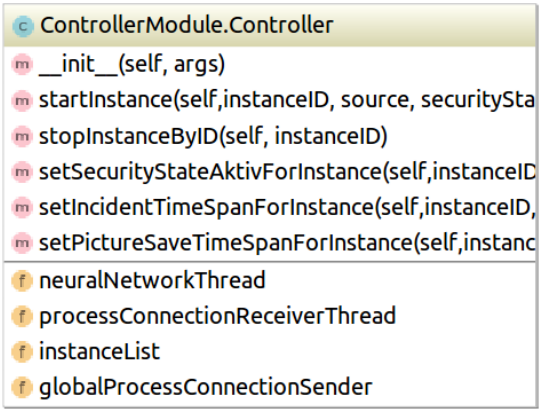

Abbildung 8: Controller Klasse

Die Klasse „Controller“ (Abbildung 8) ist die zentrale Verwaltungsklasse der neuronalen Netzwerk Anwendung. Sie verwaltet verschiedene Objekte, darunter den „neuralNetworkThread“, der für die Analyse der Kamerabilder genutzt wird, sowie zwei Objekte, die für die Kommunikation mit der Webapplikation genutzt werden, den „processConnectionReceiverThread“ und den „globalProcessConnectionSender“. Hauptaufgabe der im „Controller“ beheimateten Methoden liegt darin, die einzelnen Kameraeinheiten („Instances“) zu verwalten. Dafür wird die Liste „instanceList“ verwendet.

6.2.2.2 GlobalProcessConnectionSender

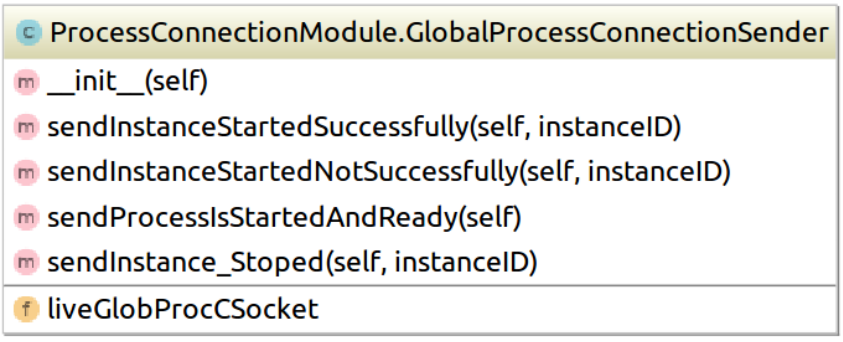

Abbildung 9: GlobalProcessConnectionSender Klasse

Die Methoden der „GlobalProcessConnectionSender“ Klasse (Abbildung 9) nutzen den ZeroMQ Socket „liveGlobProcCSocket“ und das IPC Protokoll, um der Webapplikation verschiedene Statusmeldungen in Form von Statuscodes zu schicken. Dabei wird das Zero MQ Pattern Pub-Sub (Publisher – Subscriber) verwendet.

6.2.2.3 ProcessConnectionReceiverThread

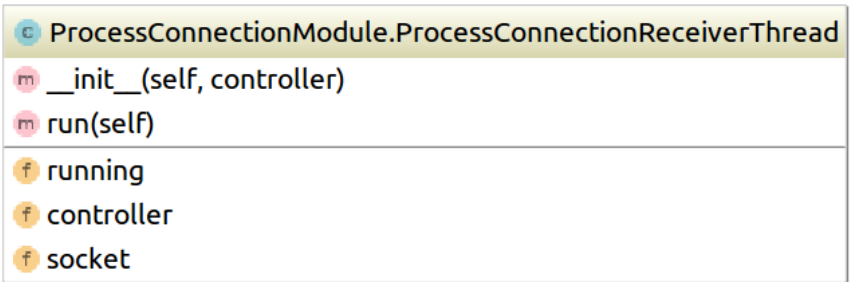

Abbildung 10: ProcessConnectionReceiverThread Klasse

Die „ProcessConnectionReceiverThread“ Klasse (Abbildung 10) erbt von der Thread Klasse. Die Aufgabe der „ProcessConnectionReceiverThread“ Klasse ist es, Nachrichten, die die Webapplikation über einen ZeroMQ Socket an die neuronale Netzwerk Anwendung sendet, entgegenzunehmen.

6.2.2.4 NeuralNetworkThread

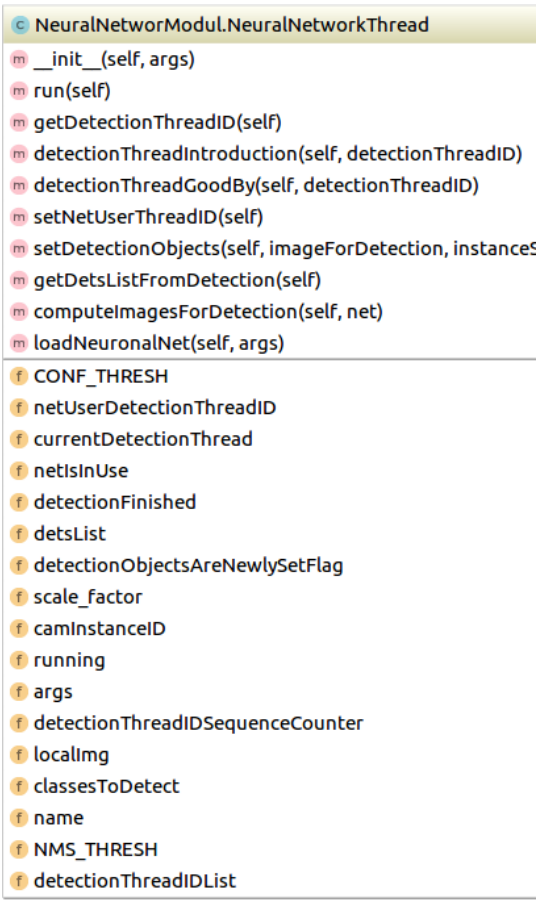

Abbildung 11: NeuralNetworkThread Klasse

Die „NeuralNetworkThread“ Klasse (Abbildung 11) erbt ebenfalls von der Thread Klasse. Ihre Hauptaufgabe ist es, das neuronale Netzwerk zu laden und dieses für die Analyse von Bildern der einzelnen „Instance“ Objekte, bereitzustellen.

6.2.2.5 Instance

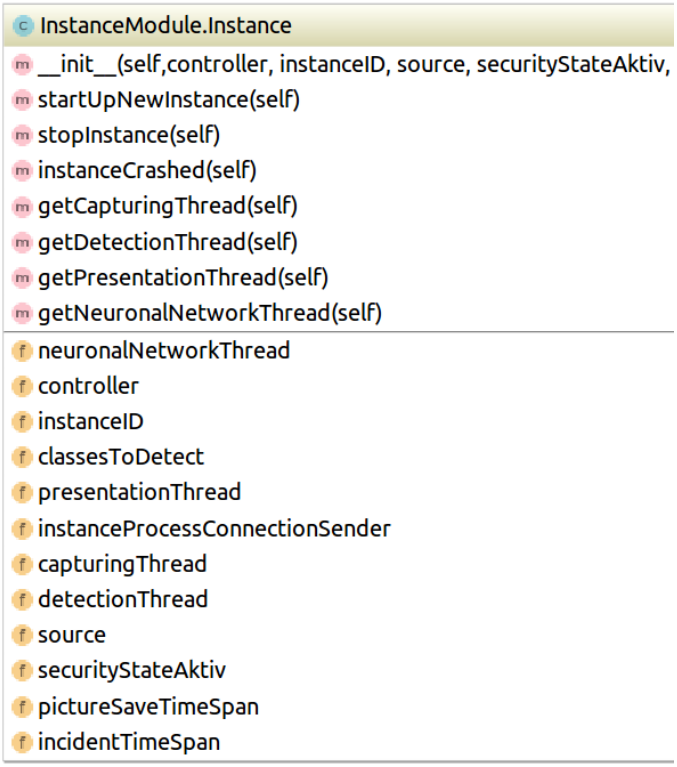

Abbildung 12: Instance Klasse

Die „Instance“ Klasse (Abbildung 12) beschreibt eine Kameraeinheit. Die drei wichtigsten Objekte einer „Instance“ sind der „capturingThread“, der „detectionThread“ und der „presentationThread“. Durch die Methoden „startUpNewInstance()“, „stopInstance()“ und „instanceCrashed()“ kann ein „Instance“ Objekt verwaltet werden.

Für „Instance“ spezifische Kommunikation mit der Webapplikation, stellt die „Instance“ Klasse das „instanceProcessConnectionSender“ Objekt zur Verfügung.

6.2.2.6 CapturingThread

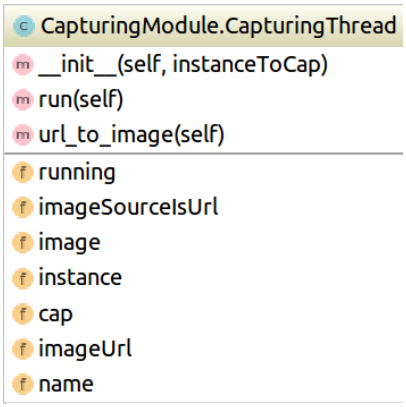

Abbildung 13: CapturingThread Klasse

Die „CapturingThread“ Klasse (Abbildung 13) realisiert die Verbindung zu einer Kamera. Sie unterstützt Web Kameras und alle IP Kameras, die ihr Livebild über eine Webplattform zur Verfügung stellen.

6.2.2.7 DetectionThread

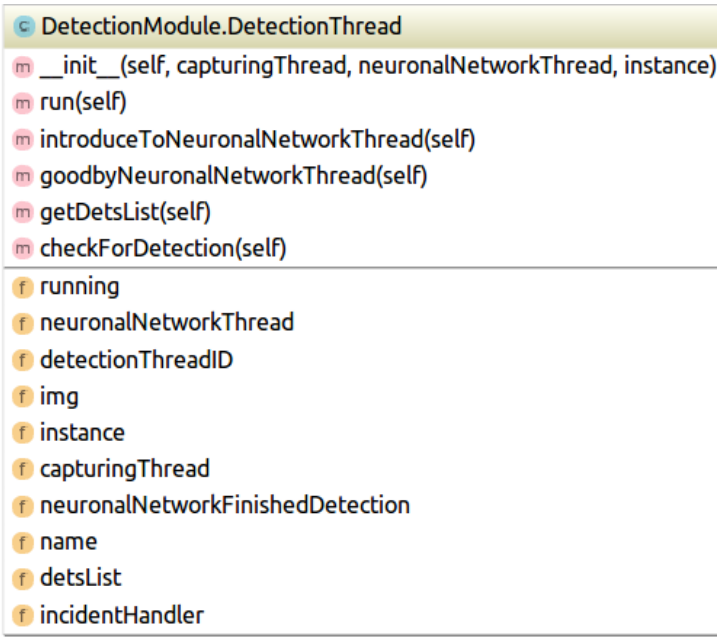

Abbildung 14: DetectionThread Klasse

Die „Detection Thread“ Klasse (Abbildung 14) ist eine Schnittstelle für jede einzelne „Instance“, zum „NeuralNetworkThread“ und damit zum neuronalem Netzwerk. Sie holt sich die für sie bestimmte Erkennung vom „NeuralNetworkThread“ und speichert sie im „detsList“ Objekt. Eine genauere Erklärung des Prozesses ist unter Punkt 6.3 Interessante Lösungen zu finden.

6.2.2.8 IncidentHandler

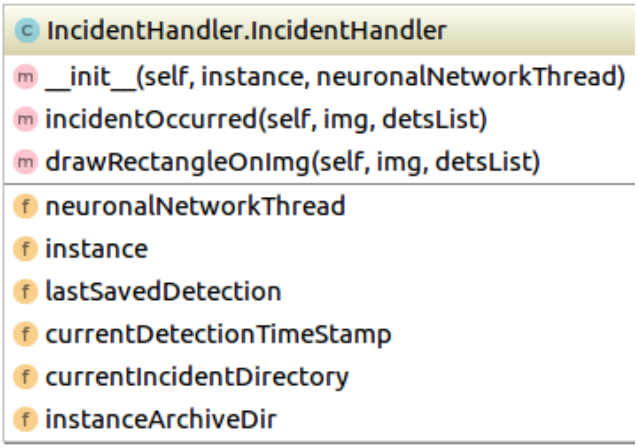

Abbildung 15: IncidentHandler Klasse

Die „IncidentHandler“ Klasse (Abbildung 15) definiert in der „incidentOccurred()“ Methode die Vorgehensweise wie auf einen Vorfall, also die Erkennung einer Person, einer „Instance“ reagiert wird. Werden die Bedingungen der „incidentOccurred“ Methode erfüllt, speichert sie das aktuelle Bild ab und benachrichtigt die Webapplikation, mithilfe des „Instance“ spezifischen „instanceProcessConnectionSender“ Objekts. Die genaue Vorgehensweise wird in Punkt 6.3 Interessante Lösungen näher erklärt.

6.2.2.9 PresentationThread

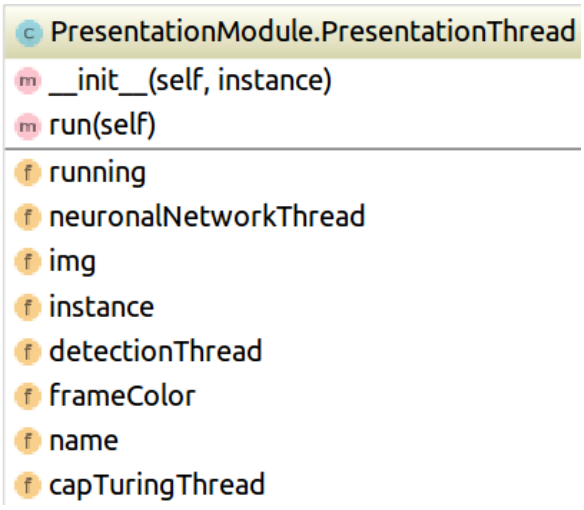

Abbildung 16: PresentationThread Klasse

Die „PresentationThread“ Klasse (Abbildung 16) erfüllt mit ihrer „run()“ Methode die Aufgabe, die aktuellen Bilddaten, des „capturingThread“s mit den aktuellen Erkennungsdaten des „detectionThread“s zusammenzuführen. Das daraus gewonnene Bild wird dann mithilfe des „Instance“ spezifischen „instanceProcessConnectionSender“ Objekts an die Webapplikation gesendet.

6.2.2.10 InstanceProcessConnectionSender

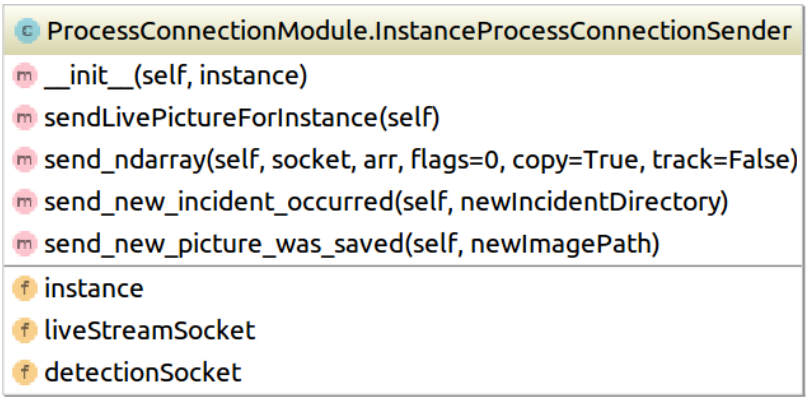

Abbildung 17: InstanceProcessConnectionSender Klasse

Die „InstanceProcessConnectionSender“ Klasse (Abbildung 17) beinhalltet alle Methoden die zum Senden „Instance“ spezifischer Daten an die Webapplikation genutzt werden.

6.3 Interessante Lösungen

6.3.1 Instance Kernsystem

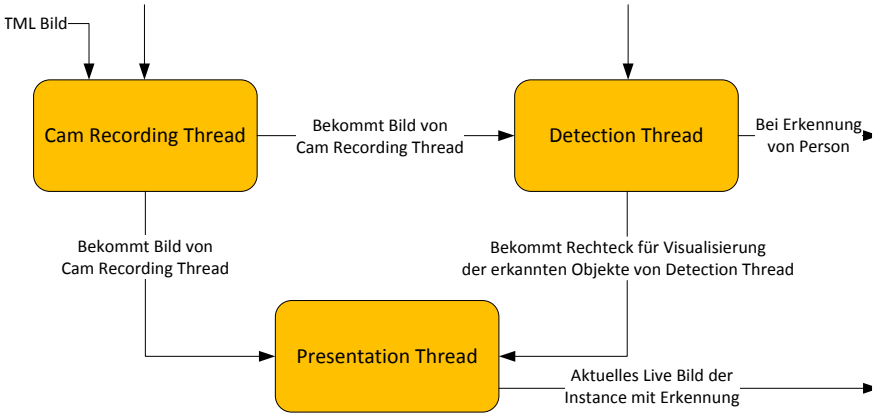

Abbildung 18: Instance Kernsystem

Der Kern einer jeden „Instance“ besteht aus den drei Threads „CapturingThread“, „DetectionThread“ und „PresentationThread“ das bietet einige große Vorteile gegenüber einem System mit nur einem Thread. Zum einen können damit trotz der, in Punkt 3.3 Python beschriebenen, Parallelitätsschwäche von Python, durch die Verwendung von Python fremden Code, das Beziehen des Kamerabildes, die Analyse der Bilder sowie das Senden von Bildern an die Webapplikation parallel ausgeführt werden. Außerdem ist es mit Hilfe dieser Architektur auch sehr leicht, logisch zueinander gehörende Code-Teile als eine Einheit zu verwalten und zu starten.

CapturingThread

Der „CapturingThread“ übernimmt die Aufgabe, die Bilder von der Bildquelle, die beim Erstellen der „Instance“ angegeben wurde, zu beziehen und sie dem „DetectionThread“ und dem „PresentationThread“ bereitzustellen.

DetectionThread

Der „DetectionThread“ ist zum einen für die Kommunikation mit dem „NeuralNetworkThread“ zuständig, dieser Vorgang wird in Punkt 6.3.2 Neuronales Netzwerk Nutzungssystem näher beleuchtet und zum anderen ist er für die Erkennung von tatsächlichen Personenerkennungen und das darauf folgende Auslösen der „incidentOccurred“ Routine zuständig. Diese wird in Punkt 6.3.3 Vorfalls Bearbeitung näher behandelt.

PresentationThread

Der „PresentationThread“ übernimmt das Zusammenführen der Bilddaten von der Kamera mit den Analysedaten des neuronalen Netzes. Dabei wird, um die Bildanzeigerate hochzuhalten, nicht auf die Analyse eines jeden Bildes gewartet, sondern das aktuelle Bild mit der aktuellen Analyse zusammengeführt. Das so generierte Bild wird dann mit Hilfe der „InstanceProcessConnectionSender“ Klasse an die Webapplikation weitergeleitet.

6.3.2 Neuronales Netzwerk Nutzungssystem

Da das in der Diplomarbeit verwendete neuronale Netzwerksystem Faster R-CNN mit dem neuronalen Netzwerk ZF Net ca. zwei GB an Arbeitsspeicher benötigt und der Jetson TX1, das Zielsystem der Diplomarbeit, nur vier GB besitzt, musste ein System gefunden werden, das mehrere „Instance“ Objekte das gleiche neuronale Netz zur Analyse ihrer Bilder nutzen lässt.

Die erste Idee um das zu realisieren war es, eine eigene Klasse zu schaffen, in der ein neuronales Netz geladen werden würde und die mithilfe einer Methode die Verwendung des neuronalen Netzes ermöglicht hätte. Von dieser Klasse sollte dann ein Objekt erzeugt werden das mit allen „Instancen“ geteilt würde. So hätten die einzelnen „Instancen“ ihre Bilder mit einem neuronalen Netz analysieren können.

Aus in der Diplomarbeit ungeklärt gebliebenen Gründen, hat diese Methode, bei der das neuronale Netzwerk schon beim Erstellen des Objekts in der „__init__“ Methode erstellt worden wäre, eine Analysezeit von über 20 Sekunden mit sich gezogen. Das war gegenüber den 0,3 Sekunden Analysezeit, bei der direkten Verwendung des neuronalen Netzes mit nur einer „Instance“ jedoch natürlich nicht akzeptabel.

Da aus den ersten Versuchen mit dem neuronalen Netz bekannt war, dass es sehr gut funktioniert, wenn man es direkt in einem Thread nutzt es sehr gut funktioniert, lag die Entscheidung einen eigenen Thread für das neuronale Netzwerk zu realisieren sehr nah.

Mit einem eigenen Thread ging allerdings das Problem einher wie die einzelnen „Instance“ Objekte diesen nutzen sollten.

6.3.2.1 System Code

In Abbildung 19 ist die „run“ Methode der „NeuralNetworkThread“ Klasse abgebildet. Die „run()“ Methode ist bei Python Threads immer jene Methode die beim Starten eines Threads aufgerufen wird.

Wie schon erwähnt war die Lösung für das Analysezeitproblem das neuronale Netz nicht schon in der „__init__“ Methode zu laden, sondern erst in der „run()“ Methode. Das geschieht im Statement „net = self.loadNeuronalNet(self.args)“.

So lange das Boolean Objekt „running“ auf True gesetzt ist, läuft der „NeuralNetworkThread“. Am Beginn dieser Schleife wird der Thread mit Hilfe des threading.Event „detectionObjectsAreNewlySetFlag.wait“ angehalten. Der Thread wartet so lange, bis an anderer Stelle mit demselben threading.Event die „clear“ Methode aufgerufen wird.

```
def run(self):
    self.running = False
    net =  self.loadNeuralNet(self.args)
    self.running = True
    while self.running == True:
        self.detectionObjectsAreNewlySetFlag.wait()

        self.computeImagesForDetection(net)
```

Abbildung 19: NeuralNetworkThread - run Methode

Abbildung 20 zeigt die „run“ Methode der „DetectionThread“ Klasse. Beim Starten eines „DetectionThreads“ meldet sich dieser zuerst beim „NeuralNetworkThread“ an (Abbildung 21). Dabei erhält der „DetectionThread“ eine ID, mit der er auch im „NeuralNetworkThread“ in die „detectionThreadIDList“ Liste eingetragen wird (Abbildung 22).

In der „netUserDetectionThreadID“ wird die ID des „DetectionThreads“, der aktuell berechtigt ist das neuronale Netzwerk zu nutzen, gespeichert.

Ist kein „DetectionThread“ beim „NeuralNetworkThread“ angemeldet, ist die „netUserDetectionThreadID“ auf -255 gesetzt. Wenn das der Fall ist, löst die Anmeldung eines neuen „DetectionThreads“ die Methode „setNetUserThreadID“ aus, die die „netUserDetectionThreadID“ auf die nächste ID in der „detectionThreadIDList“ setzt. Ist schon eine „DetectionThread“ registriert, geschieht das an anderer Stelle.

Der „DetectionThread“ ist jetzt registriert und läuft in seiner Routine weiter. Hier gelangt er in einer Schleife, die so lange läuft, bis das „running“ Attribut auf False gesetzt wird. Das geschieht, wen die gesamte „Instance“ gestoppt wird.

Jede Schleifeniteration wird überprüft, ob das neuronale Netzwerk verwendet wird und wenn nicht, ob der prüfende „DetectionThread“ als Benutzer eingetragen ist.

Ist das der Fall, wird zunächst der „netIsInUse“ Parameter auf True gesetzt und dann die Objekte, die der „NeuralNetworkThread“ zur Analyse eines Bildes benötigt gesetzt. Dazu gehört das eigentliche Bild jedoch auch welche Objekte erkannt werden sollen und welcher „DetectionThread“ das Netz aktuell nutzt. Das Setzen dieser Werte löst im Anschluss die Analyse durch den „NeuralNetworkThread“ aus, indem für das threading.Event „detectionObjectsAreNewlySetFlag“ die „clear“ Methode ausgeführt wird (Abbildung 19). Ist wiederum die Analyse des Bildes durch das neuronale Netzwerk fertig, löst das die „clear“ Methode für „neuralNetworkFinishedDetection“ aus. Und der „DetectionThread“ holt sich die Daten der Analyse und speichert sie. Im Anschluss löst der „DetectionThread“ die „setNetUserThreadID“ Methode aus, die den nächsten „DetectionThread“ bestimmt, der berechtigt ist das neuronale Netz zu nutzen und setzt den „netIsInUse“ Parameter wieder auf False.

```
def run(self):
    # Introduce detectionThread to neuralNetworkThread
    self.introduceToNeuralNetworkThread()

    while self.running:
        if (self.neuralNetworkThread.netIsInUse == False) and \
                (self.neuralNetworkThread.netUserDetectionThreadID == self.detectionThreadID):
            self.neuralNetworkThread.netIsInUse = True
            self.img = deepcopy(self.capturingThread.image)
            self.neuralNetworkThread.setDetectionObjects(self.img, self.instance.classesToDetect,
                                                         self.instance.instanceID, self)
            self.neuralNetworkFinishedDetection.wait()
            self.detsList = self.neuralNetworkThread.getDetsListFromDetection()
            self.neuralNetworkThread.setNetUserThreadID()
            self.neuralNetworkThread.netIsInUse = False
            if self.checkForDetection() and self.instance.securityStateAktiv == True:
                self.incidentHandler.incidentOccurred(self.img, self.detsList)

    #When Instance Stops
    self.goodbyNeuralNetworkThread()
```

Abbildung 20: DetectionThread - run Methode

```
def introduceToNeuralNetworkThread(self):
    self.detectionThreadID = \
        self.neuralNetworkThread.detectionThreadIntroduction(self.detectionThreadID)
    print('Instance with ID:{} introduced itself as {}'.format(
        self.instance.instanceID, self.detectionThreadID))
```

Abbildung 21 DetectionThread - introduceToNeuronalNetworkThread Methode

```
def detectionThreadIntroduction(self, detectionThreadID):
    #Generate new detectionThreadID
    self.detectionThreadIDSequenceCounter = \
        self.detectionThreadIDSequenceCounter + 1
    #Add detectionThreadID to detectionThreadIDList
    self.detectionThreadIDList.append(self.detectionThreadIDSequenceCounter)
    if self.netUserDetectionThreadID == -255:
        self.setNetUserThreadID()
    #return detectionThreadID
    return self.detectionThreadIDSequenceCounter
```

Abbildung 22: NeuralNetworkThread – detectionThreadIntroduction Methode

6.3.3 Vorfalls Bearbeitung

Ein Vorfall wird vom „DetectionThread“ erkannt, wenn bei einer „Instance“ das „securityStateAktive“ Attribut auf True gesetzt ist und eine Person vom neuronalen Netz erkannt wurde. Daraufhin wird in der „IncidentHandler“ Klasse die „incidentOccurred“ Methode aufgerufen (**Fehler! Verweisquelle konnte nicht gefunden werden.**).

Die Bilder von Vorfällen werden im Dateisystem in einem Archiv gespeichert, da der Nutzer des Systems jedoch einstellen kann, wie lange ein Vorfall, während dem die Bilder gruppiert im Archiv gespeichert werden dauert und wie oft während eines solchen Vorfalls Bilder gespeichert werden sollen. Dafür ist eine Logik die das realisiert vonnöten.

Der „IncidentHandler“ besitzt die Parameter „lastSavedDetection“, „currentDetectionTimeStamp“ um abprüfen zu können, ob das aktuelle Bild gespeichert werden soll.

Die Logik hinter der „incidentOccurred“ Methode ist jetzt wie folgt aufgebaut.

- Wenn der „lastSavedDetection“ Parameter noch nicht gesetzt ist, also noch kein Vorfall vorgefallen ist
    - wird ein neuer Ordner für die Bilder des Vorfalls angelegt,
    - der „lastSavedDetection“ Parameter auf „currentDetectionTimeStamp“ gesetzt
    - die Analysedaten visualisiert,
    - das Bild gespeichert
    - und die Webapplikation darüber benachrichtigt
- Ist die Zeit, die seit dem Zeitpunkt des letzten gespeicherten Bildes vergangen ist kleiner als die Vorfalls Zeitspanne, die der Benutzer festgelegt hat,
    - Wird überprüft ob die Zeitspanne größer ist als die Zeitspanne die der Benutzer für die Zeit zwischen dem Speichern zweier Bilder festgelegt hat ist.
        - Ist das so
            - werden die Analysedaten visualisiert,
            - wird das Bild gespeichert
            - und der „lastSavedDetection“ Parameter auf „currentDetectionTimeStamp“ gesetzt
            - und die Webapplikation darüber benachrichtigt
        - Ist das nicht so
            - Wird nichts gemacht
- Ist die Zeit, die seit dem Zeitpunkt des letzten gespeicherten Bildes vergangen ist größer als die Vorfalls Zeitspanne, die der Benutzer festgelegt hat,
    - wird ein neuer Ordner für die Bilder des Vorfalls angelegt,
    - „lastSavedDetection“ wird auf „currentDetectionTimeStamp“ gesetzt
    - die Analysedaten visualisiert,
    - das Bild gespeichert
    - und die Webapplikation darüber benachrichtigt

```
def incidentOccurred(self, img, detsList):
    #IncidentOccures Timstamp gets taken
    self.currentDetectionTimeStamp = datetime.datetime.now()
    if self.lastSavedDetection is None:
        #First Icident of the Session
        self.currentIncidentDirectory = self.instanceArchiveDir+\
                                        '/{:%Y-%m-%d %H:%M:%S}'.format(self.currentDetectionTimeStamp)
        if not os.path.exists(self.currentIncidentDirectory):
            os.makedirs(self.currentIncidentDirectory)
        imagePath = self.currentIncidentDirectory +\
                    '/{:%Y-%m-%d %H:%M:%S}'.format(self.currentDetectionTimeStamp) + '.png'
        cv2.imwrite(imagePath, self.drawRectangleOnImg(img, detsList))
        self.lastSavedDetection = self.currentDetectionTimeStamp
        self.instance.instanceProcessConnectionSender.send_new_incident_occurred(self.currentIncidentDirectory)
        self.instance.instanceProcessConnectionSender.send_new_picture_was_saved(imagePath)
    else:
        if (self.currentDetectionTimeStamp - self.lastSavedDetection).seconds <= self.instance.incidentTimeSpan:
            if (self.currentDetectionTimeStamp - self.lastSavedDetection).seconds >=\
                    self.instance.pictureSaveTimeSpan:
                # Picture for Incident
                imagePath = self.currentIncidentDirectory + \
                            '/{:%Y-%m-%d %H:%M:%S}'.format(self.currentDetectionTimeStamp) + '.png'
                cv2.imwrite(imagePath, self.drawRectangleOnImg(img, detsList))
                self.lastSavedDetection = self.currentDetectionTimeStamp
                self.instance.instanceProcessConnectionSender.send_new_picture_was_saved(imagePath)
            else:
                # Picture in Incident which was taken to close to another one
                None
        else:
            # New Incident but not the First one of the Session
            self.currentIncidentDirectory = self.instanceArchiveDir + \
                                            '/{:%Y-%m-%d %H:%M:%S}'.format(
                self.currentDetectionTimeStamp)
            if not os.path.exists(self.currentIncidentDirectory):
                os.makedirs(self.currentIncidentDirectory)
            imagePath= self.currentIncidentDirectory + \
                       '/{:%Y-%m-%d %H:%M:%S}'.format(self.currentDetectionTimeStamp) + '.png'
            cv2.imwrite(imagePath, self.drawRectangleOnImg(img, detsList))
            self.lastSavedDetection = self.currentDetectionTimeStamp
            self.instance.instanceProcessConnectionSender.send_new_incident_occurred(self.currentIncidentDirectory)
            self.instance.instanceProcessConnectionSender.send_new_picture_was_saved(imagePath)
```

Abbildung 23: IncidentHandler - incidentOccurred Methode

6.3.4 Prozessschnittstelle

Da das Gesamtsystem aus zwei Anwendungen besteht, müssen beide Anwendungen für die jeweils andere, eine Kommunikationsschnittstelle zur Verfügung stellen. Dies wird von beiden Anwendungen mittels der ZMQ Bibliothek realisiert. In der neuronalen Netz Anwendung ist der gesamte Code, der dies ermöglicht in einem Modul, dem „ProcessConnectionModule“ beheimatet. Dieses beinhaltet drei Klassen die „InstanceProcessConnectionSender“ Klasse, die „GlobalProcessConnectionSender“ Klasse und die „ProcessConnectionReceiverThread“ Klasse.

```
PROCESS_IS_STARTED_AND_READY_CODE='1'

SEND_LIVE_PICTURE_FOR_INSTANCE_CODE = '10'
START_INSTANCE_CODE = '11'
STOP_INSTANCE_CODE = '12'
SET_SECURITY_STATE_CODE = '13'
SET_INCIDENT_TIME_SPAN_CODE = '14'
SET_PICTURE_SAVE_TIME_SPAN_CODE = '15'

INSTANCE_WAS_STARTED_CODE='20'
INSTANCE_WAS_NOT_STARTED_CODE='21'
INSTANCE_STOPED_CODE='25'

DETECTION_NEW_INCIDENT_OCCURRED_CODE = '30'
DETECTION_NEW_IMAGE_WAS_SAVED_CODE = '31'
```

Abbildung 24: SecCamProcessComunicationCodes.py

Die beiden Anwendungen kommunizieren mit Hilfe von Codes, die im „SecCamProcessComunicationCodes.py“ File definiert sind(Abbildung 24).

6.3.4.1 Empfangs-Schnittstelle

Die Empfangs-Schnittstelle wird durch einen Thread realisiert. Der „ProcessConnectionReceiverThread“ definiert in seiner „run“ Methode eine Routine, die alle 0,1 Sekunden versucht Daten vom Socket zu lesen und zu interpretieren. In der Abbildung 25 ist die 1.1.1.1 Empfangs-Schnittstelle grafisch dargestellt.

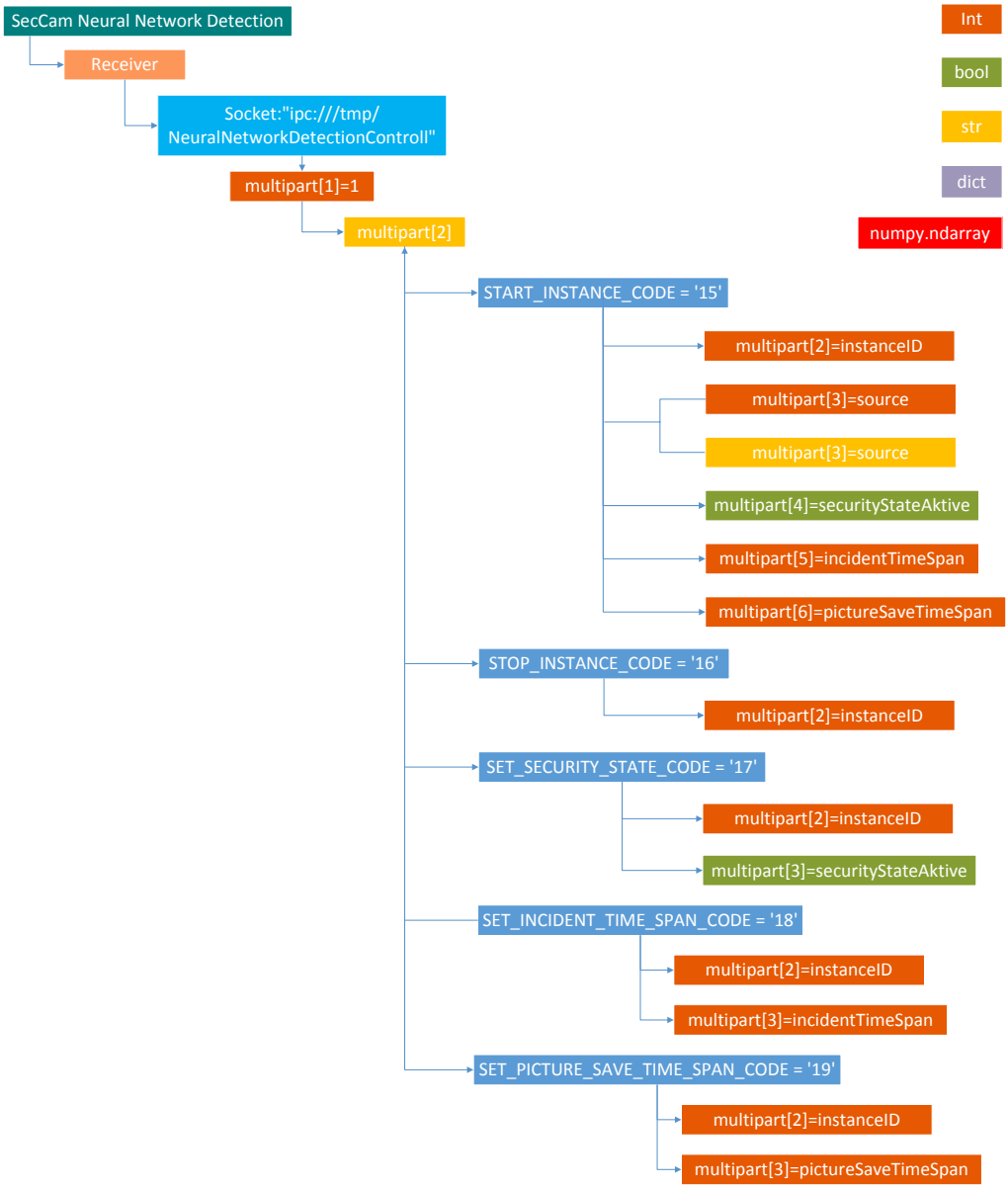

Abbildung 25:Neuronale Netzwerk Anwendung Empfangs-Schnittstelle

6.3.4.2 Ausgabe-Schnittstelle

Die Ausgabeschnittstelle in Form der Klassen „InstanceProcessConnectionSender“ und „GlobalProcessConnectionSender“ werden von vielen unterschiedlichen Programmteilen genutzt, um der Webapplikation Daten zu senden. Sie ist in Abbildung 26 abgebildet.

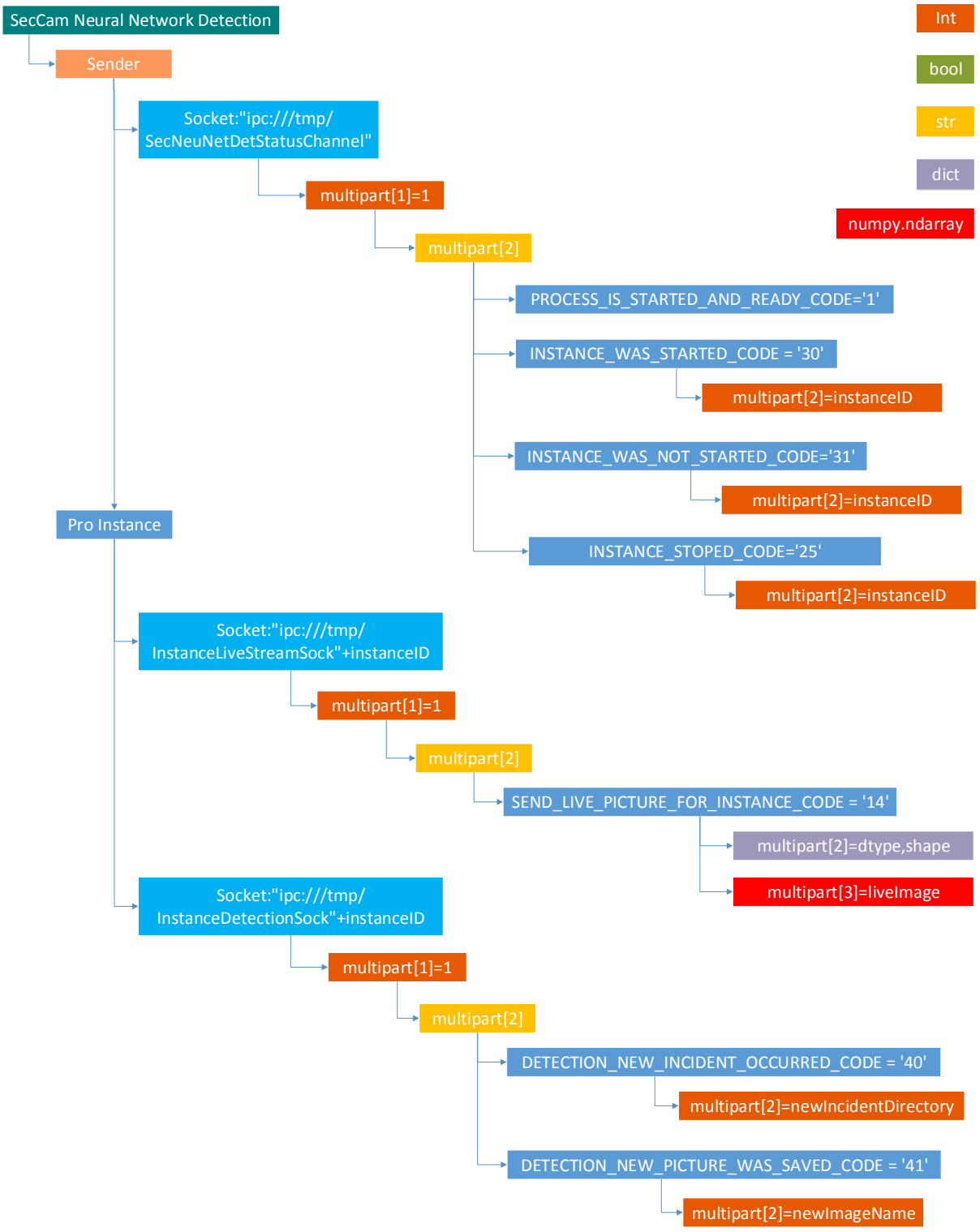

Abbildung 26: Neuronale Netzwerk Anwendung Ausgabe-Schnittstelle

6.4 Django Webanwendung

6.4.1 Abstrakte Beschreibung

Die Django Anwendung ist für die visuelle Darstellung der Daten und Bilder, die sie von der Neuronalen Netzwerk Anwendung bekommt, zuständig.

Mit dem Start des Django Webservers, wird ebenfalls der Receiver Thread gestartet, über diesen erhält die parallel laufende Anwendung die zu startenden Kamerainstanzen. Nachdem die Instanz initialisiert wurde, erhält der Receiver Thread einen Code, durch den er erkennt, dass die Kamera gestartet wurde und online ist. Der User kann nun auf die bereits in der Datenbank eingefügten Kamerainstanzen zugreifen und die Bilder und Informationen über die Vorfälle bzw. das System anzeigen lassen. Wenn neue Kameras hinzugefügt, geändert oder gelöscht werden, wird dies auch in der Datenbank gespeichert und über den Receiver-Thread der Neuronalen Netzwerk Anwendung mitgeteilt. Im Archiv gespeicherte Bilder können ebenfalls angesehen werden. Bei dem ersten Benutzerzugriff muss der Benutzer seinen Anmeldenamen und das dazugehörige Passwort eingeben um Zugriff auf Seiten der Webanwendung zu haben.

6.4.2 Webseiten

6.4.2.1 Login

Wenn der User das erste Mal auf die Webseite zugreift, wird er auf die Login-Seite befördert. Falls der Benutzer den Benutzernamen und das Passwort richtig eingibt, wird beim Server ein Attribut in der Session gesetzt, um die Seite vor unbefugtem Zugriff zu schützen. Das Attribut speichert den Loginnamen des Benutzers serverseitig, sodass der Benutzer nicht weiß, ob es in der Session vorhanden ist, oder nicht. Durch Django können Anfragen auf andere Seiten der Webseite abgefangen werden, und direkt auf die Anmeldeseite (siehe Abb. 27) weitergeleitet werden.

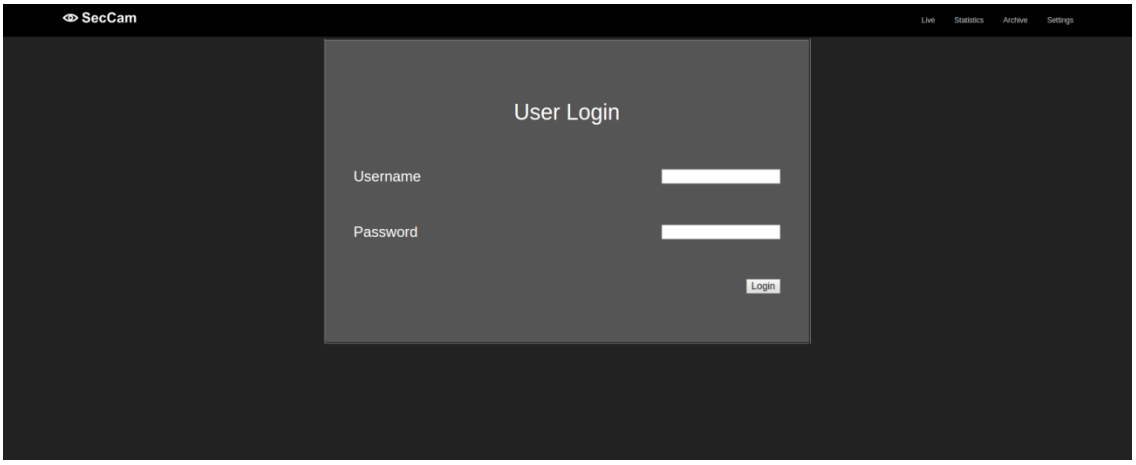

Abbildung 27: Anmeldeseite

6.4.2.2 Liveanzeige

Die Liveanzeige (siehe Abb. 28) ist das Herzstück der Webseite, hier können Livebilder der Kameras angesehen werden. Der Sicherheitsmodus (ob erkannte Menschen im Archiv abgespeichert werden, oder nicht) kann hier auch umgestellt werden. Wenn der Button auf grün ist, ist diese Kamera im Sicherheitsmodus. Mit einem Mausklick auf das Bild wird ein neues Fenster geöffnet indem die Übertragung besser zu sehen ist.

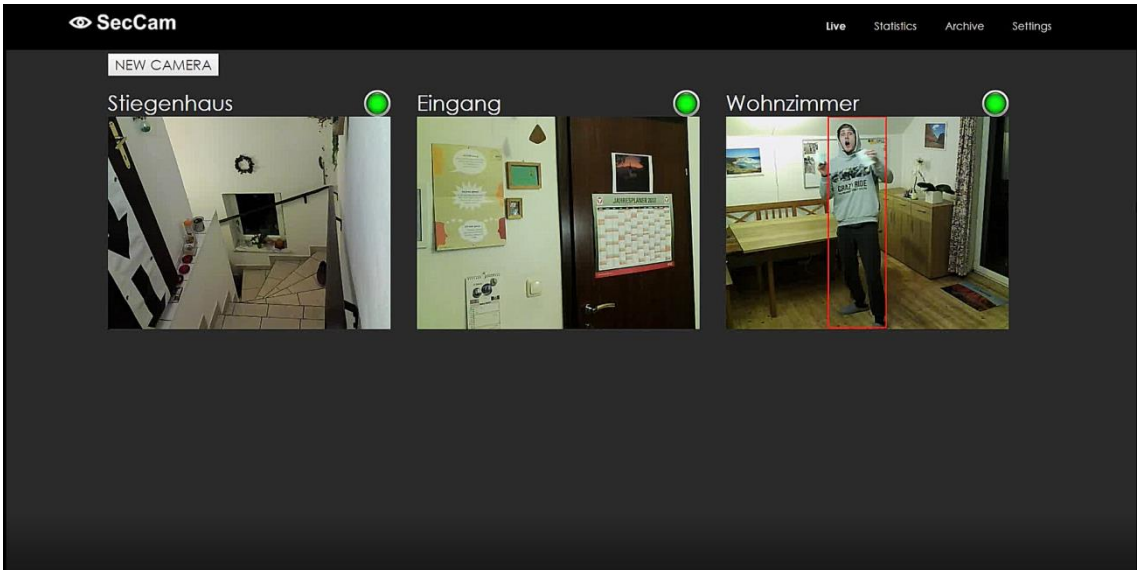

Abbildung 28: Livebilder-Anzeige

Wenn eine neue Kamera angelegt werden möchte, wird der Benutzer auf die „newInstance“-Seite (siehe Abb. 29) weitergeleitet. Hier kann er die benötigten Daten für das neue Livebild eintragen (Kameranamen, Bildquelle, Sicherheitsmodus, Vorfallzeitspanne und Bilderzeitspanne).

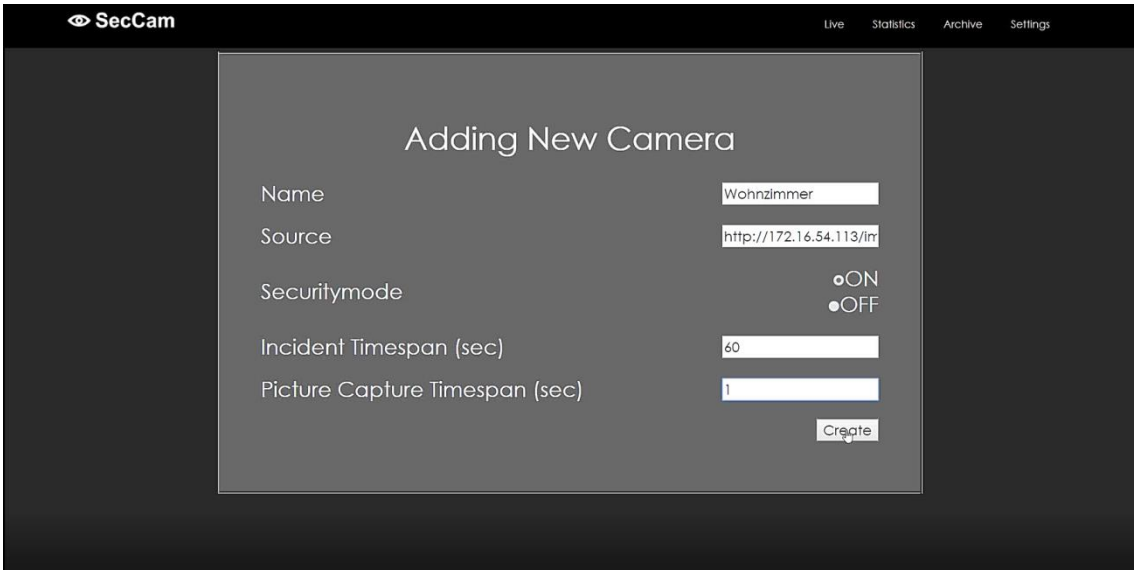

Abbildung 29: Neue Kamera hinzufügen

6.4.2.3 Statistiken

In den Statistiken, werden vorgefertigte Statistiken über das System aufgezeigt (siehe Abb. 30). Die Statistiken berechnen sich aus den in der Datenbank befindlichen Informationen über Kameras, Vorfälle und Bilder.

Abbildung 30: Statistik-Anzeige

6.4.2.4 Archiv

Das Archiv zeigt alle eingefügten Kameras an und die im Sicherheitsmodus erkannten Vorfälle.

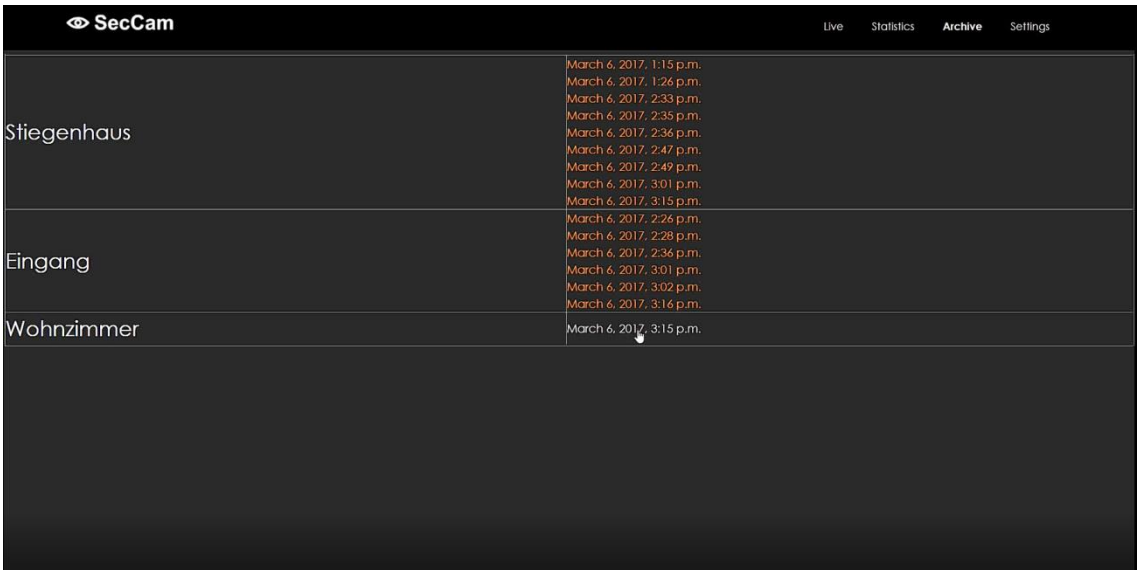

Abbildung 31: Archiv-Anzeige

Über die Verknüpfungen kommt man auf den spezifischen Vorfall und kann sich die Bilder ansehen.

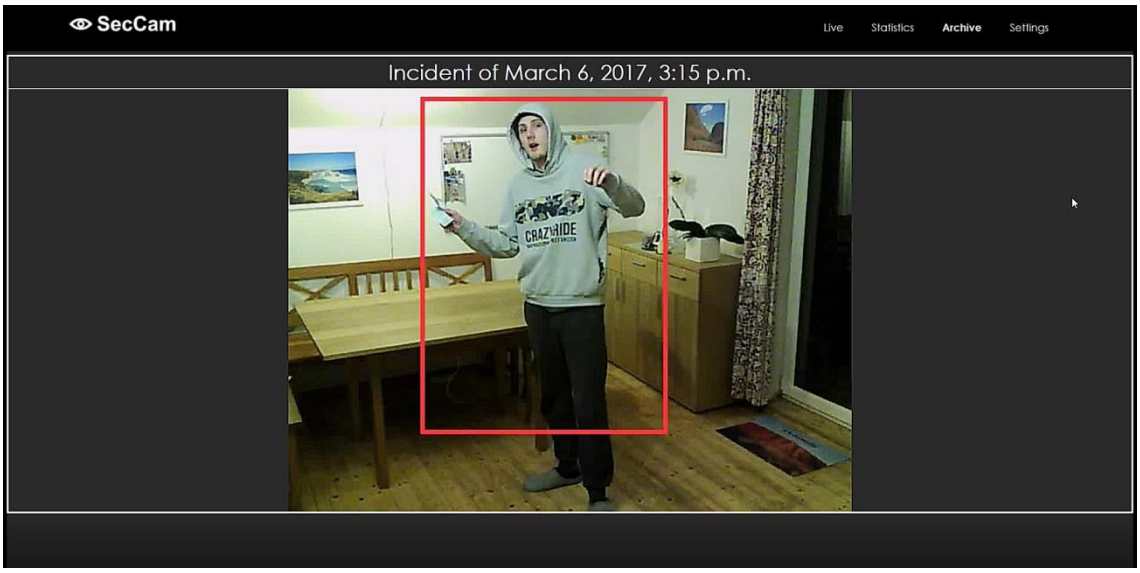

Abbildung 32: Spezifischer Vorfall

Auf gespeicherte Bilder kann auch über das Dateisystem zugegriffen werden.

6.4.2.5 Einstellungen

In den Einstellungen können Kameras gelöscht werden und die Daten der Kamera (z.B. die Bildquelle) geändert werden.

Abbildung 33: Anzeige aller Kameras

Mit einem Klick auf den Kameranamen, gelangt man auf die Kameraänderungsseite.

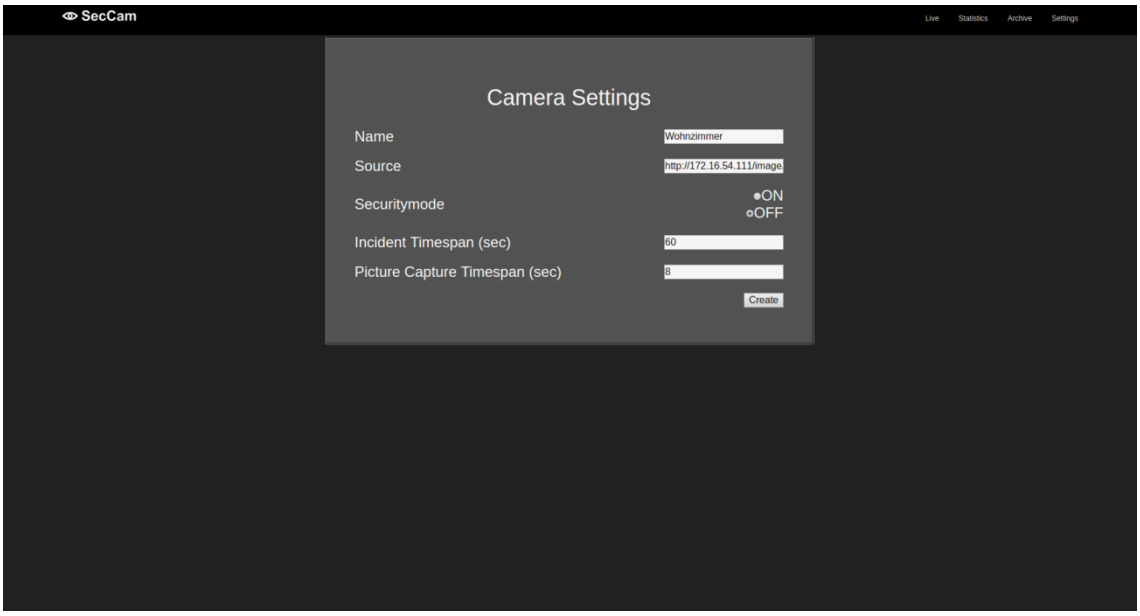

Abbildung 34: Kameraeinstellungen

6.4.3 Benachrichtigung per Email

Das System weiß nicht nur den Benutzernamen und das Passwort des Benutzers, sondern auch seine E-Mail, auf die eine Benachrichtigung (siehe Abb. 34) gesendet wird, wenn ein neuer Vorfall entstanden ist. Im Betreff steht, dass ein neuer Vorfall aufgetreten ist und von welcher Kamera dieser ausgelöst worden ist. Im Hauptteil befinden sich die ID und die Zeit des Vorfalls, und wieder von welcher Kamera dieser erfasst wurde. So kann der Benutzer im Archiv sofort die richtigen Bilder finden.

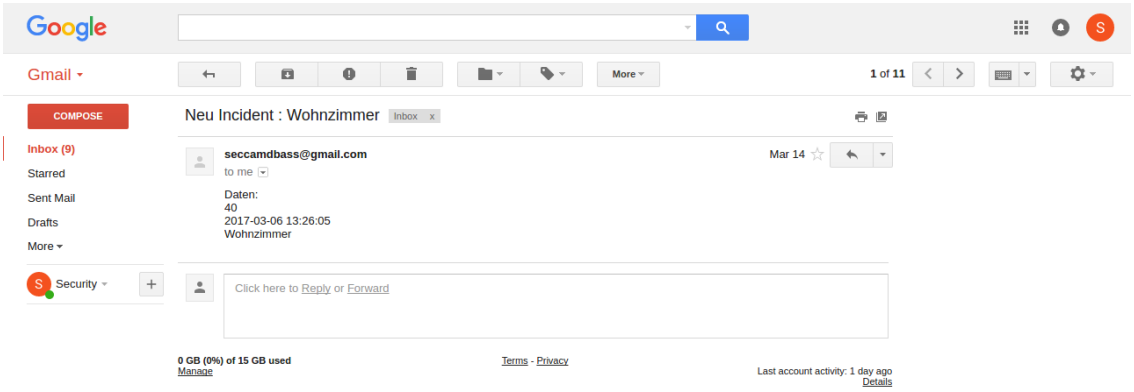

Abbildung 35: Vom Webserver gesendete E-Mail

6.4.4 Datenmodel

Das Datenmodel der Webseite besteht aus vier Tabellen (siehe Abb. 36).

Die Tabelle „User“ wird vom Djangoframework zur Verfügung gestellt und ist in der Bibliothek „django.contrib.auth“ zu finden. Wir benutzten nur die Felder „id“, „username“, „email“ und „password“. Es wurde keine eigene Klasse erstellt, da das Feld „password“ bereits einen Ausleseschutz besitzt. Das Passwort wird nicht als Klartext, sondern wird mittels des, in der Bibliothek „django.contrib.auth.hashers“ befindlichen, „PBKDF2PasswordHashers“ verschlüsselt und so gespeichert.

Die anderen in Relation stehenden Tabellen sind für die Speicherung von Kamera-, Vorfalls- und Bilddaten verantwortlich.

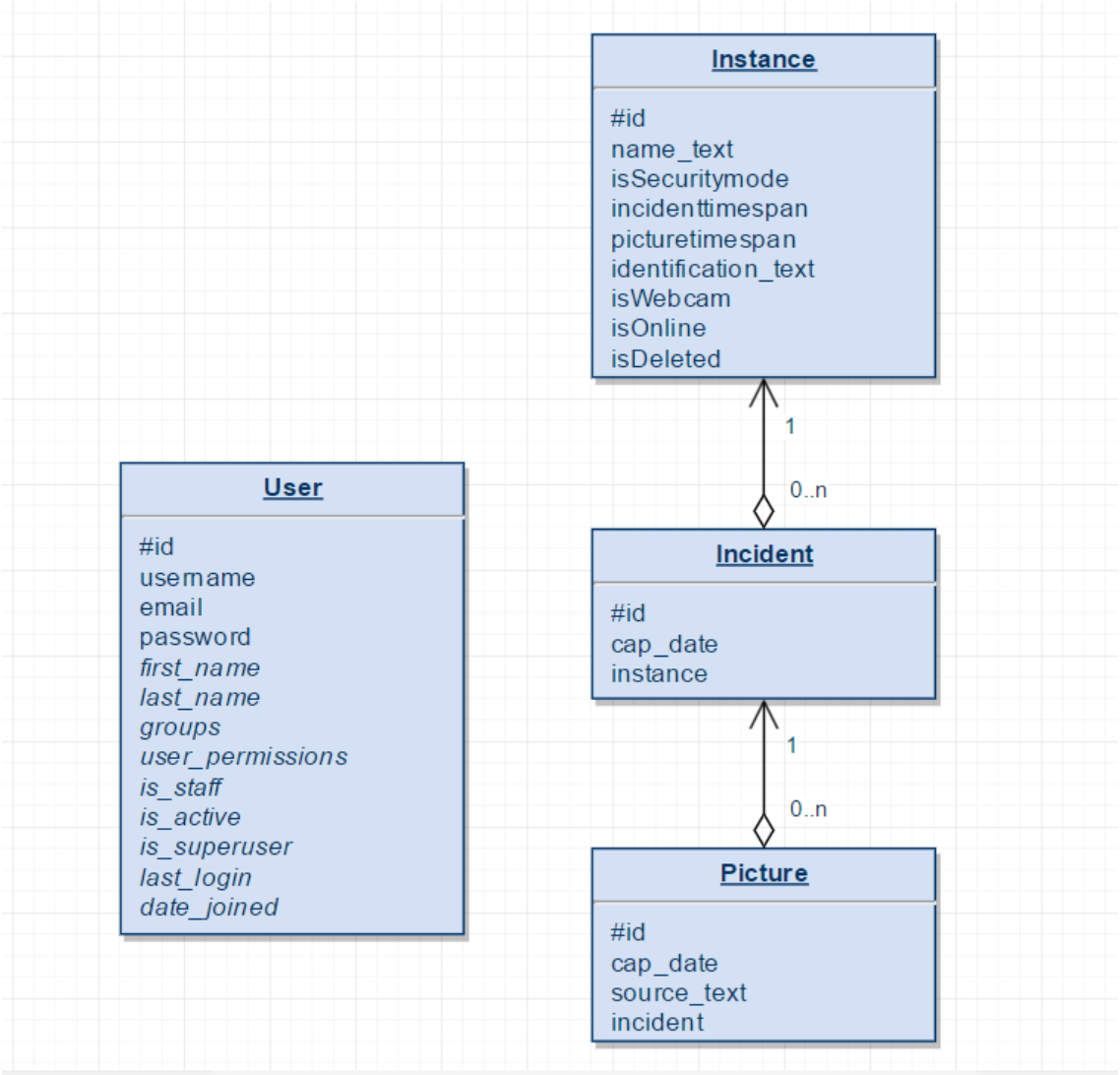

Abbildung 36: Datenmodel

Die „Instance“-Tabelle, repräsentiert eine Kamera mit allen für sie wichtigen Daten.

Instance	
Id	Die id ist der Primärschlüssel (Einzigartigkeit einer Ausprägung einer Tabelle). Durch Sie können Daten einer bestimmten Kamera zugeordnet werden.
name_text	Der Name den der Benutzer auf der Webseite für diese Kamera festlegt.
isSecuritymode	Dieses Attribut bestimmt ob eine Kamera im Sicherheitsmodus ist, oder nicht. (Dieser Datentyp kann entweder „wahr“ oder „falsch“ sein)
incidenttimespan	Gibt die Zeitspanne an, in der mehrere Bilder als ein Vorfall gelten.
picturetimespan	Gibt die Zeitspanne zwischen zwei Bildern an.
identification_text	Hier wird die Bildquelle angegeben.
isWebcam	Bestimmt, ob die Kamera eine Webcam ist.
isOnline	Bestimmt, ob die Kamera Online ist, bzw. Bilder liefert.
isDeleted	Bestimmt, ob eine Kamera deinstalliert ist. Die Kamera wird beim Löschen nicht aus der Datenbank gelöscht, da sonst die Vorfälle ebenfalls aus dem Archiv entfernt werden müsste.

Die Tabelle „Incident“, steht für einen Vorfall. Ein Vorfall muss zu einer Kamera zugeordnet sein.

Incident	
id	Primärschlüssel
cap_date	Dieses Feld speichert die Uhrzeit und das dazu passende Datum, an dem ein Vorfall ausgelöst worden ist.
instance	Die „instance“-Feld speichert ein Object der Klasse „Instance“, von dessen Kamera der Vorfall entstanden ist.

Die „Picture“-Tabelle, speichert das Speicherungsdatum und die Quelle eines Bildes. Hier muss jedes Bild mit einem Vorfall verbunden sein

Picture	
id	Primärschlüssel
cap_date	Hier wird wie bei „Incident“ die Uhrzeit und das Datum wenn das Bild erkannt worden ist, gespeichert
source_text	Es repräsentiert den Dateipfad, wo ein Bild abgelegt worden ist.
incident	Dieses Feld speichert ein Object der Klasse „Incident“. So wird das Bild zu einem Vorfall hinzugefügt.

6.4.5 Receiver-Thread

Der Receiver Thread ist für die Annahme und das Versenden von Daten über den ZMQ Socket verantwortlich. Er wird zur Startzeit des Webservers gestartet. Sofort wartet er auf den Start der Partneranwendung. Sobald dieser erfolgt ist, wird der Thread die Übermittlung der in der Datenbank befindlichen, aber noch nicht gelöschten, Kameradaten beginnen. Für jede gestartete Instanz erhält er eine Nachricht zurück. Falls diese initialisiert worden ist und Bilder übertragen kann, wird die Instanz als online markiert. Nun können diese Kameras Vorfälle und Livebilder übermitteln. Wenn ein Vorfall eintritt werden die Bilddaten vom Receiver-Thread in die Datenbank gespeichert, nicht aber das Bild selbst, welches im Archiv gespeichert wird. Wenn die Verbindung zu einer Kamera abbricht, wird es dem Thread mitgeteilt, welcher diese dann wieder auf offline setzt. Änderungen, Erstellungen oder Löschungen von Kameras werden der Neuronalen Netzwerk Anwendung übermittelt.

6.4.6 Django-Projekt

Das Django-Teil bezieht sich hauptsächlich auf die Speicherung und Anzeige der Informationen. Im Projekt wurde eine Anwendung hinzugefügt. In dieser Anwendung sind die oben beschriebenen Tabellen festgelegt. Nach dem Hochfahren des Servers und des Ladens des Models wird in der Anwendung der Receiver-Thread aufgerufen. Die gültigen Anfragen, werden durch „urls.py“ auf die richtigen Anzeigeantworten in „views.py“ weitergeleitet.

7 Verbesserungsvorschläge und Resümee

7.1 Verbesserungsvorschläge

Django Webanwendung

- Im Internet hosten
  Da unsere Arbeit sich zurzeit nur auf das lokale Netzwerk bezieht, könnte man dem entgegenwirken, indem man die Applikation online hostet. Zurzeit erhält der Besitzer der Anlage die E-Mail-Benachrichtigung, kann aber, wenn er außerhalb des Netzwerkes ist, nicht auf diese zugreifen.
- Mail Server
  Die Benachrichtigungen laufen zurzeit eine konfigurierte Gmail-Adresse. Der Webserver kann über ein sogenanntes Applikationspasswort auf die Adresse zugreifen. Das Problem ist, dass damit man die Funktion der Applikationspasswörter verwenden kann, die Zwei-Schritt-Überprüfung eingeschaltet sein muss. Hierbei handelt es sich um eine Anmeldeüberprüfung bei der der Besitzer der E-Mail-Adresse eine SMS an sein Handy bekommt, um seine Identität zu beweisen. Wenn nun eine Änderung von Seiten Googles kommen würde, benötigt man das Handy des Besitzers um die Änderungen vom G-Mail-Server zu holen und in die Django Webanwendung einzutragen. Dies kann behoben werden, indem die Applikation nicht G-Mail benutzen müsste, sondern einen eigenen Mailserver besitzen würde, von dem die E-Mails ausgesendet werden.

Neuronale Netzwerk Anwendung

- Trainieren des neuronalen Netzes mit eigenen Bildern
  Die Erkennung von Personen durch das in der Anwendung verwendete neuronale Netz, funktioniert bei guten Lichtbedingungen und gutem Blickwinkel der Kameras sehr zuverlässig. Werden die Lichtbedingungen jedoch schwieriger oder die Kameras so positioniert, dass Bilder aus eher ungewöhnlichen Winkeln aufgenommen werden, kommt das neuronale Netz schnell an seine Grenzen.
  Die Lösung für das Lichtproblem könnte der in vielen IP Kameras eingebaute Nachtsichtmodus sein. Da dieser jedoch nur schwarzweißen Bilder ausgibt hat das neuronale Netz im aktuellen Stand große Probleme darauf etwas zuerkennen. Um die Erkennung von schwarzweißen Bildern zu verbessern wäre eine Möglichkeit das neuronale Netz mit einer sehr großen Anzahl dieser weiter zu trainieren.

- Performance der Hardware
  Da von unserem Auftraggeber als Zielsystem der Jetson TX1 vorgegeben wurde war es für die Diplomarbeit keine Option, die Performance der Software

durch die Verwendung potenterer Hardware zu verbessern. Dennoch wäre es eine Überlegung wert, die Anwendung zukünftig auf einem leistungsstärkeren NVIDIA Grafikprozessor auszuführen und so die Analyse der Bilder durch das neuronale Netz erheblich zu beschleunigen.

7.2 Resümee

Im Rahmen dieser Diplomarbeit konnten wir unser Wissen in den verwendeten Technologien immens erweitern. Gerade im Bereich neuronale Netze, der für uns noch gänzlich unbekannt war, konnten wir die Diplomarbeit als hervorragenden Anlass nehmen uns in diese sehr interessante Technologie einzuarbeiten.

Auch die zwei Wochen, an denen wir am Institut für Bioinformatik arbeiten durften, waren eine hervorragende neue Erfahrung, die uns einen guten Einblick in die Universitätswelt gewährte.

Abschließend sind wir mit unserer Arbeit durchwegs zufrieden, wir haben eine Anwendung geschaffen die den Anforderungen unseres Auftraggebers entspricht und haben dabei den Umgang mit für uns noch unbekannten Technologien gemeistert.

Quellenverzeichnis

[1] DerStandard: St. Pöltner Studenten entwickeln Alarmanlage mit Raspberry Pi.

http://derstandard.at/2000032928133/St-Poeltner-Studenten-entwickeln-Alarmanlage-mit-Raspberry-Pi

Version: 15.März.2016 – Zuletzt besucht:24.2.2017

[2] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biology, 5(4):115–133, 1943.

[3] W. Pitts and W.S. McCulloch. How we know universals the perception of auditoryandvisualforms. Bulletin of Mathematical Biology,9(3):127–147, 1947.

[4] M. Minsky and S. Papert. Perzeptrons. MIT Press, Cambridge, Mass, 1969.

[5] David Kriesel, 2007, Ein kleiner Überblick über Neuronale Netze / Geschichte Seite : 9, erhältlich auf
http://www.dkriesel.com

[6] Einlagiges Perzeptron
Von MartinThoma - Eigenes Werk, CC BY 3.0,
https://commons.wikimedia.org/w/index.php?curid=28211506

[7] Caffe Deep Learning Framework
Jia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, Trevor
http://caffe.berkeleyvision.org/

[8] R-CNN Grafik
Von: rbg,jdonahue,trevor,malik Rich feature hierarchies for accurate object detection and semantic segmentation
https://people.eecs.berkeley.edu/~rbg/papers/r-cnn-cvpr.pdf

[9] Faster R-CNN
Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, Microsoft Research
http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf

[10] Faster R-CNN Beispielerkennung

Quelle: https://github.com/rbgirshick/py-faster-rcnn/blob/master/data/demo/004545.jpg

Objekt Erkennung Faster R-CNN

[11] ZF
Matthew D Zeiler, Rob Fergus,
Visualizing and Understanding Convolutional Networks
https://arxiv.org/abs/1311.2901

[12] CUDA
https://developer.nvidia.com/cuda-zone

[13] cuDNN
https://developer.nvidia.com/cudnn

[14] Open Source Lizenz Nutzungsverteilung

https://www.blackducksoftware.com/top-open-source-licenses

[15] Jetson TX1 Stats
http://elinux.org/Jetson_TX1

[16] Nvidia Jetson TX1 Page
http://www.nvidia.com/object/jetson-tx1-dev-kit.html

[17] Linux Counter
https://www.linuxcounter.net/statistics/distributions

[18] Canonical Ltd Ubuntu Entwicklung
https://www.ubuntu.com/about/about-ubuntu/our-philosophy

[19] Pycharm Features
https://www.jetbrains.com/pycharm/features/

[20] Illustrating Python multithreading vs multiprocessing

Nathan Grigg

https://nathangrigg.com/2015/04/python-threading-vs-processes

[21] ZMQ

https://en.wikipedia.org/wiki/ZeroMQ

http://learning-0mq-with-pyzmq.readthedocs.io/en/latest/pyzmq/patterns/pubsub.html

http://api.zeromq.org/2-1:zmq-ipc

[22] OpenCV
Gary Bradski und Adrian Kaehler : Learning OpenCV 2008
What is OpenCV Seite 1
(English)

[23] Django

https://en.wikipedia.org/wiki/Django_(web_framework)

https://de.wikipedia.org/wiki/Django_(Framework)

[24] Faster R-CNN Installation

Requirements: software

Basic installation

https://github.com/rbgirshick/py-faster-rcnn

[25] Caffe Benötigte Ubuntu Softwarepakete

General dependencies

http://caffe.berkeleyvision.org/install_apt.html

[26] Cython ohne CUDA

SrCMpink commented on 14 Jun 2016

https://github.com/rbgirshick/py-faster-rcnn/issues/123

8 Anhang

8.1 Installation

Dieses Kapitel beschreibt alle Arbeiten, die nötig sind um ein Secure Cam Softwaresystem auf einem Ubuntu LTS 16.4 zum Laufen zu bringen. Hierbei wird nicht auf den Vorgang der Installation eines Ubuntu LTS 16.4 System eingegangen, sondern nur auf die darauf aufbauenden Technologien.

Daher hier ein Verweis auf die Offizielle Installationsanleitung von Ubuntu.

https://www.ubuntu.com/download/desktop/install-ubuntu-desktop

8.1.1 Faster R-CNN

Dieser Punkt beschreibt die Installation des Faster R-CNN Systems und darin integriert die Installation von Caffe. [24]

8.1.1.1 Benötigte Ubuntu Softwarepakete

```
sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev libopencv-dev libhdf5-serial-dev protobuf-compiler
```

Hier ist es wichtig beide Befehle separat einzugeben.

```
sudo apt-get install --no-install-recommends libboost-all-dev
```

```
After this operation, 272 MB of additional disk space will be used.
Do you want to continue? [Y/n] Yes
```

Abbildung 37: Achtung beim ersten Statement Yes nicht Y

```
sudo apt-get install libgflags-dev libgoogle-glog-dev liblmdb-dev
```

[25]

CUDA

Download der aktuellen CUDA Version für Ubuntu 16.4, die heruntergeladene Datei mithilfe der Anleitung aus Abbildung 38 installieren. CUDA wird nicht benötigt, wenn das System nur auf einer CPU laufen soll.

https://developer.nvidia.com/cuda-downloads

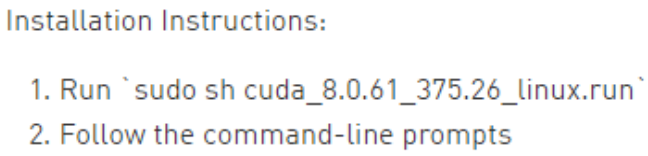

Abbildung 38: CUDA Installationsanleitung

cuDNN

Für eine bessere Performance der Objekterkennung wird auf NVIDIA Systemen auch noch die Installation von cuDNN empfohlen.

Download Link:

https://developer.nvidia.com/cudnn

```
PREREQUISITES

    CUDA 7.0 and a GPU of compute capability 3.0 or higher are required.

ALL PLATFORMS

    Extract the cuDNN archive to a directory of your choice, referred to below as <installpath>.
    Then follow the platform-specific instructions as follows.

LINUX

    cd <installpath>
    export LD_LIBRARY_PATH=`pwd`:$LD_LIBRARY_PATH

    Add <installpath> to your build and link process by adding -I<installpath> to your compile
    line and -L<installpath> -lcudnn to your link line.
```

Abbildung 39:cuDNN Installationsanleitung

BLAS

```
sudo apt-get install libatlas-base-dev
```

Python

Da das Caffe Framework nur Python der Version 2.x unterstützt ist darauf zu achten, diese Version auch zu verwenden.

```
sudo apt-get install python-dev

sudo apt-get -y install python-pip
```

Cython

```
pip2 install cython
```

OpenCV

```
pip2 install opencv-python
```

easydict

```
pip2 install easydict
```

Numpy

```
sudo pip2 install numpy
```

8.1.1.2 Klonen des Faster R-CNN Repository

Installation von Git

```
sudo apt-get install git
```

Klonen von Faster R-CNN

```
cd ~
git clone --recursive https://github.com/rbgirshick/py-faster-rcnn.git
```

8.1.1.3 Build der Cython Module

```
cd py-faster-rcnn
cd lib
make
```

Sollen die Cython Module ohne CUDA Unterstützung erstellt werden, muss die setup.py Datei folgendermaßen angepasst werden.

```
#CUDA = locate cuda()

…

#self.set_executable('compiler_so', CUDA['nvcc'])

…

#Extension('nms.gpu_nms',
    #['nms/nms_kernel.cu', 'nms/gpu_nms.pyx'],
    #library_dirs=[CUDA['lib64']],
    #libraries=['cudart'],
    #language='c++',
    #runtime_library_dirs=[CUDA['lib64']],
    # this syntax is specific to this build system
    # we're only going to use certain compiler args with nvcc and not with
    # gcc the implementation of this trick is in customize_compiler() below
    #extra_compile_args={'gcc': ["-Wno-unused-function"],
                        #'nvcc': ['-arch=sm_35',
                                 #'--ptxas-options=-v',
                                 #'-c',
                                 #'--compiler-options',
                                 #"'-fPIC'"]},
    #include_dirs = [numpy_include, CUDA['include']]
#),
```

8.1.1.4 Kompilierung von Caffe und pycaffe

Zunächst müssen alle für Caffe benötigten Python Module installieren. Dabei hilft einem die requirements.txt, die sich unter folgendem Pfad finden lässt:

~py-faster-rcnn/caffe-fast-rcnn/python/ requirements.txt

```
cd ~/py-faster-rcnn/caffe-fast-rcnn/python
```

Um die darin angeführten Module zu installieren, nutzt man den folgenden Befehl.

```
for req in $(cat requirements.txt); do pip install $req; done
```

Konfiguration der Makefile.config Datei

```
cd ..
cp Makefile.config.example Makefile.config
gedit Makefile.config
```

CPU&GPU

Wenn Caffe mit CPU&GPU genutzt werden soll, müssen keine Änderungen an der Makefile.config Datei vorgenommen werden.

cuDNN

Wenn cuDNN verwendet werden soll, muss die Zeile

USE_CUDNN := 1

vom Kommentarzeichen befreit werden.

Nur CPU

Wenn nur die CPU verwendet werden soll, muss die Zeile

#CPU_ONLY := 1

vom Kommentarzeichen # befreit werden.

Kompilieren von Caffe und pycaffe

Dann wird zum kompilieren von Caffe und pycaffe wieder das make Programm ausgeführt.

```
cd ~/py-faster-rcnn/caffe-fast-rcnn
make -j8 && make pycaffe
```

8.1.1.5 Download der vortrainierten Netze

```
cd ~/py-faster-rcnn
./data/scripts/fetch_faster_rcnn_models.sh
```

8.1.1.6 Testen der Installation via Demo.py

CPU&GPU und cuDNN

```
cd ~/py-faster-rcnn
./tools/demo.py --net zf
```

Nur CPU

Zuerst muss die Datei nms_wrapper.py angepasst werden.

```
cd ~/py-faster-rcnn/lib/fast_rcnn
gedit nms_wrapper.py
```

```
from fast_rcnn.config import cfg
#from nms.gpu_nms import gpu_nms
from nms.cpu_nms import cpu_nms

def nms(dets, thresh, force_cpu=False):
    """Dispatch to either CPU or GPU NMS implementations."""

    if dets.shape[0] == 0:
        return []
    #if cfg.USE_GPU_NMS and not force_cpu:
        #return gpu_nms(dets, thresh, device_id=cfg.GPU_ID)
    else:
        return cpu_nms(dets, thresh)
```

Abbildung 40: nms_wrapper.py nur CPU

```
cd ~/py-faster-rcnn
./tools/demo.py --cpu --net zf
```

[24]

8.1.2 ZMQ

```
sudo apt-get install python-zmq
```

8.1.3 Django

```
sudo apt-get install python-django
```

8.2 Ausführung der Prozesse

Django Webanwendung

Der Django Webserver muss zuerst gestartet werden. Wenn er nur auf dem eigenen Rechner erkennbar sein soll, wird er so aufgerufen:

```
python manage.py runserver --noreload
```

Dieser Terminalbefehl muss im Ordner des Webservers, in dem sich manage.py befindet, ausgeführt werden.

Wenn der Webserver von Anderen im Netzwerk gefunden werden soll, muss er mit einer passenden IP-Adresse und Port aufgerufen werden, auf die sich der Server binden kann (in diesem Beispiel „127.0.0.1:8000“).

```
python manage.py runserver 127.0.0.1:8000 –noreload
```

Nun kann der Django-Webserver über seine IP-Adresse im Browser angefragt werden.

Neuronale Netzwerk Anwendung

Auch hier muss man sich im Projektordner befinden, in dem sich „Main.py“ befindet.

```
cd SecCamNeuralNetworkDet
```

Wenn die Analyse der Bilddaten nur mit der CPU ausgeführt werden sollen, muss das „—cpu“ Argument mit übergeben werden.

```
python Main.py --cpu
```

Wenn die Analyse der Bilddaten mit der GPU ausgeführt werden sollen, muss kein spezielles Argument übergeben werden.

```
python Main.py
```