Systemverifizierungsplan und Testbericht

Moritz Lechner
Konstantin Roßmann
Leon Sobotta

CE755 Advanced Computer Engineering
Computer Engineering

9. Februar 2023

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In diesem Dokument wird erläutert, wie das System zur Vegetationsbranderkennung mit Hilfe von Künstlichen Neuronalen Netzen (KNN), auf seine Funktionsfähigkeit getestet werden soll. Des Weiteren werden die Ergebnisse dieser Verifizierungen dargestellt.

Statt des ursprünglich geplanten Gesamtsystems wird jedoch nur das KNN getestet. Das liegt daran, dass zum Zeitpunkt der Tests nur das Neuronale Netz in Python implementiert ist. Grund dafür ist, dass wir auf halbem Weg gemerkt haben, dass unser ursprünglicher Ansatz, die Bildklassifizierung, nicht für dieses Projekt geeignet ist. Daher mussten wir auf ein Netz zur Objekterkennung umsteigen. Der Vorteil davon ist aber, dass die Erkennung der Brände genauer ist. Zusätzlich wird nicht nur erkannt ob sich ein Brand auf dem Bild befindet, sondern auch wo er auf dem Bild ist und wie viele es sind. Sämtliche erkannten Brände werden dann mit einer so genannten Bounding Box umrahmt und markiert.

Um von dem aktuellen Stand zum vollständigen System zu gelangen müssten wir die Gewichte, also Parameter des Netzes, in ein Rust Programm importieren, welches anhand dieser Gewichtungen die Bilder überprüft und danach gegebenenfalls labelt.

2 Verifzierungsobjekt

Überprüft wird also ein KNN zur Objekterkennung, welches auf dem YOLOv8 Modell von Ultralytics basiert. Das ist ein sehr komplexes aber auch optimiertes Netz, welches durch Trainieren mit einem bestimmten Datensatz auf den jeweiligen Anwendungsfall angepasst werden kann. Der Grund warum wir uns dafür entschieden haben dieses fertige Netz zu nutzen ist, dass Objekterkennungsmodelle sehr komplex sein müssen um präzies Vorhersagen treffen zu können. Ein solches Netz selber zu implementieren hätte einerseits unsere Fähigkeiten und andererseits die Rechenleistung unserer PCs überschritten.

3 Verifizierungsansatz

Als Ansatz zur Verifizierung werden die im *Systems Requirements Review* angegebenen Test genutzt, siehe Abbildung 1. Dabei beziehen wir uns jedoch nur auf die Test der Anforderungen die etwas mit dem KNN zu tun haben, da nur diese, mit dem aktuellen Fortschritt, getestet werden können. Wir werden also vor Allem die Funktionsfähigkeit des Netzes überprüfen. Dafür soll das trainierte Netz Testbilder, welche es nicht zum Lernen benutzt hat, auf Vegetionsbrände untersuchen. Um diese Anforderung als erfüllt zu betrachten, sollen mindestens 75% der Brände erkannt werden und jedes erkannte Feuer soll markiert werden.

Zusätzliche Anforderungen, welche getestet werden, sind, dass das KNN nach dem Trainieren minimiert werden soll um es einfacher in ein Rust Programm zu importiern und, dass die Ausgabe des Netzes die Parameter der Bounding Boxes sein soll.

Nachdem wir von der Bildklassifizierung zur Objekterkennung umgestiegen sind hat sich diese zweite Anforderung verändert. Zuvor sollte die Confidence des Netzes als Gleitkomma ausgegeben werden. Doch da wir nun mehrere Brände auf einem Bild erkennen wollen werden neben den Parametern der Boxen auch die Sicherheit für jeden erkannten Brand ausgegeben. Der Output kann durch eine einfache Sichtprobe überprüft werden, während das erfolgreiche Minimieren dadurch validiert wird, dass das minimierte Netz die gleichen Bilder überpüft wie das originale Netz. Wenn dann die gleichen Ergebnisse herauskommen war das Minimieren erfolgreich.

Nr.	Anforderungsbeschreibung	Verifikation	Bestanden
SW-0001	Wenn ein Vegetationsbrand fotografiert wird, muss dieser erkannt werden.	Testdurchlauf des Programms mithilfe des Datensatzes, sowie darauffolgende Stichproben.	-
SW-0007	Nach dem Trainieren soll das Netz minimiert werden.	Testdurchlauf vor und nach dem minimieren, sowie vergleich der Daten, mithilfe von Test-Datensatz.	-
SW-0015	Die Ausgabe des Programmes sollen die Parameter der Bounding Boxes sein	Sichtprobe	-

Abbildung 1: Die Anforderung an das KNN und die Entsprechenden Verifikationen

4 Verifizierungsprogramme

Als Vorbereitung auf die Tests wurde das Netz trainiert. Danach waren die ersten Tests zur Überprüfung des Netzes manuelle Aufrufe des Programms, bei denen die zu testenden Bilder als einzeln Argument übergeben wurden. So wurden 20 Bilder mit Bränden getestet, von denen 15 erfolgreich erkannt wurden. Hierbei kam es zu einigen False Positives wie zum Beispiel in Abbildung 7 unten rechts zu sehen ist. Dabei ist die Confidence des Netzes aber sehr gering und die Markierung kann über einen Grenzwert von 0,25 heraus gefiltert werden. Das bedeutet, dass alle Stellen, bei denen die Sicherheit unter diesem Grenzwert liegen nicht als Feuer markiert werden, dadurch kann ein Großteil der False Positives eliminiert werden. Auf den Bildern in diesem Dokument sind noch einige Markierungen, die unter diesen Grenzwert fallen, zu sehen, um zu verdeutlichen was für das Netz alles die Eigenschaften eines Brandes aufweist.

Aussagekräftig war dieser Test jedoch nicht, da nur eine kleine Anzahl an Bildern überprüft wurde. Darum haben wir als nächstes unser Netz dem ausführlichen Test mit dem integriertem Validation Task des YOLOv8 Modells unterzogen, welcher eine Genauigkeit von etwa 51 % ausgibt, wie in Abbildung 2 unter dem Punkt *P (Precision)* mit dem Wert *0,509* zu erkennen ist. Dieser Test vergleicht die Ausgaben des eigenen Netzes mit gelabelten Bildern aus einem anderen Datensatz, dabei wird nur mit Bildern getestet, auf denen sich Brände befinden.

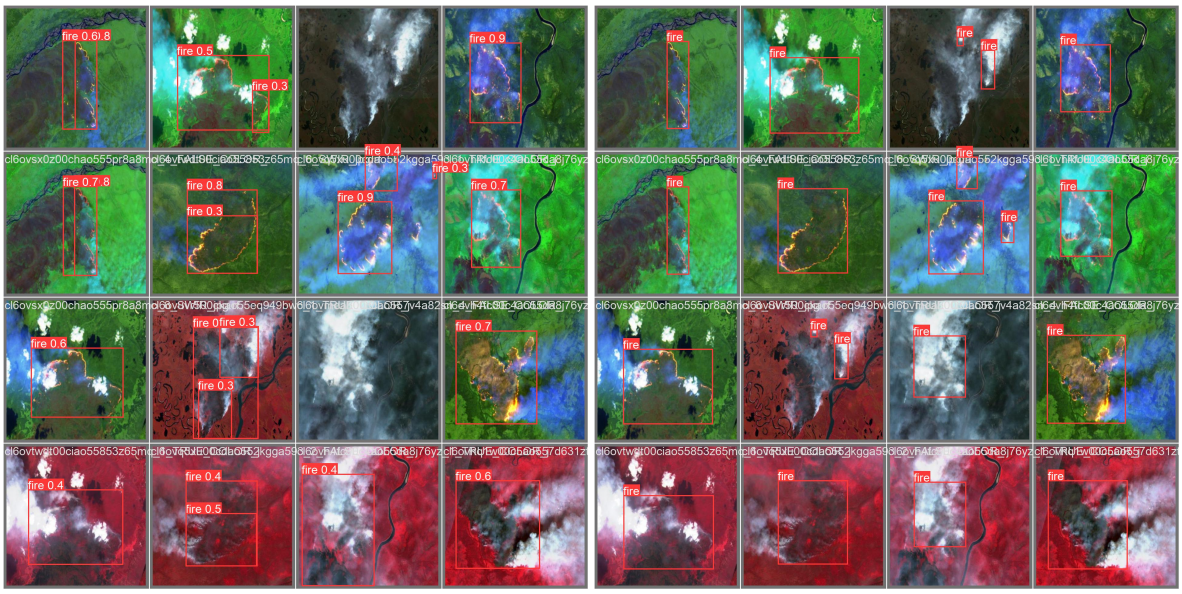
In Abbildung 3 ist jedoch deutlich zu sehen, dass der Großteil der Feuer erkannt wird, wobei es zwei False Negatives bei den beiden grauen Bildern gibt. Warum die Genauigkeit trotzdem nur bei so einem geringen Wert liegt, ist darauf zurückzuführen, dass die meisten Bounding Boxes nicht exakt mit den Label Boxen übereinstimmen, bei der Überprüfung aber nur Bilder als korrekt gezählt werden, bei denen die Übereinstimmung bei 100% liegt.

Bei Sichtproben über 3 Testbatches mit jeweils 16 Bildern zeigt sich, dass auf alle 48 Bilder nur 2 False Negatives kommen. False Positives werden bei diesem Test leider nicht überprüft. Manuelle Tests haben aber gezeigt, dass diese selten auftreten, und davon die meisten Fälle auf Fotos, auf denen andere, richtige Brände erkannt werden.

```
!yolo task=detect mode=val model=/content/runs/detect/train/weights/best.pt data=/content/datasets/Wildfire-Satellite-2/data.yaml

Ultralytics YOLOv8.0.25 🚀 Python-3.8.10 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15110MiB)
Model summary (fused): 218 layers, 25840339 parameters, 0 gradients, 78.7 GFLOPs
val: Scanning /content/datasets/Wildfire-Satellite-2/valid/labels... 200 images, 1 backgrounds, 0 corrupt: 100% 200/200 [00:00<00:00, 2522.62it/s]
val: New cache created: /content/datasets/Wildfire-Satellite-2/valid/labels.cache
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 13/13 [00:07<00:00,  1.78it/s]
                   all        200        317      0.509      0.495      0.474      0.291
Speed: 0.9ms pre-process, 23.4ms inference, 0.0ms loss, 2.3ms post-process per image
```

Abbildung 2: Die Ausgabe des Validation Task des YOLOv8 Netzes



(a) Die Prediction des Netzes (b) Die Labels

Abbildung 3: Vergleich zwischen Prediction und Label

5 Testbeschreibung

Wie in Abschnitt 4 bereits erwähnt wird das Netz also zuerst trainiert. Dafür wurde ein Datensatz mit etwa 8000 Bildern von der Website Roboflow genutzt. Als erste Testbilder wurden danach die genommen, die der Dozent zur Verfügung gestellt hat. Dabei wurden 4 der 6 Brände erkannt. Für den zweiten, ausführlichen Test mit dem Validation Task wurde ein zweiter Datensatz mit 100 Bildern genutzt. Auf Basis des ersten Tests haben wir eine Genauigkeit von etwa 70% erwartet.

6 Anomalien

Es ist anzumerken, dass das Netz beim Trainieren nur 552 Bilder aus dem 8000 Bilder großen Datensatz genutzt hat. Es hat alle Bilder erkannt und konnte auch alle richtig Lesen, trotzdem hat es nur dieses kleinen Teil genutzt. Bei der Onlinerecherche konnten keine ähnlichen Erfahrungen gefunden werden.

Die geringe Anzahl an Trainingsbildern kann auch der Grund für die stellenweise schlechte Erkennung sein. So kann man in Abbildung 4 deutlich sehen, dass der große Hauptbrand nicht erkannt wird, während die kleineren Brände mit hoher Confidence markiert werden.

7 Zusammenfassung

Basierend auf den Ergebnissen des Validation Task des YOLOv8 Modells kann gesagt werden, dass die 75% Erfolgsquote erreicht wurden, das KNN kann also Vegetionsbrände erkenne. Damit ist sowohl unsere wichtigste Anforderung an das Netz, als auch unser Missionsziel, welches in Abbildung 5 unter *MG-01* zu sehen ist, erfolgreich eingehalten worden.

Die Anforderung, dass das Netz minimiert werden soll haben wir indirekt umgesetzt. Wir haben zwar nicht aktiv eine minimierte Version implementiert, aber wir haben darauf geachtet, dass das genutzte Modell kompakt ist. Das YOLOv8 Modell ist zwar sehr komplex und leistungsstark, aber trotzdem optimiert genug als dass man es ohne weitere Minimierung exportieren kann. Diese Anforderung kann also als erfüllt betrachtet werden. Genauso ist es mit der letzten Anforderung *SW-0015* auch. Wie in Abbildung 6 zu sehen ist, werden für alle 3 Feuer, die in Abbildung 4 erkannt werden, die Confidence und Koordinaten ausgegeben.

Die Mission Objectives aus Abbildung 5 wurden nur teilweise erreicht. Um *MO-01* eindeutig bestätigen zu können hätten die Ergebisse ausführlich mit denen des Farbfilter Projektes verglichen werden müssen. Auf den ersten Blick erschienen beide Algorithmen aber ungefähr gleich genau, wobei der Farbfilter mehr False Positives, und das KNN mehr False Negatives hatte. Wie gesagt bezieht sich diese Einschätzung aber nur auf eine kleine Stichprobe von 5 Bildern.

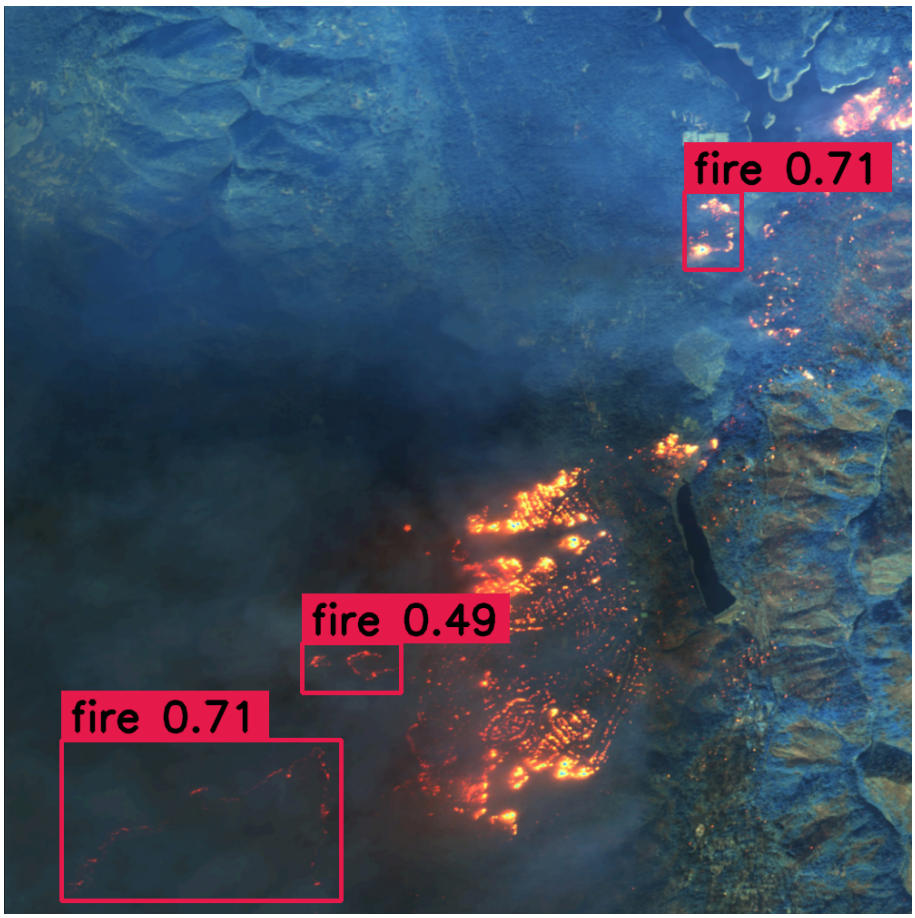


Abbildung 4: Ausgabe des Netzes

Bezüglich *MO-02* kann gesagt werden, dass die Rechenintensität dank des optimierten Modells von Ultralytics deutlich geringer ausfällt als erwartet. Sogar während der Live Erkennung über die Kamera hatten unsere Laptops nur eine Auslastung von 50%. Wenn die einzelnen Bilder überprüft werden sollen, fällt die Auslastung noch einmal deutlich geringer aus, doch die Laufzeit des Programms ist so gering, dass dabei kein genauer Wert ermittelt werden konnte. Das Verhältnis zwischen Rechenaufwand und Genauigkeit ist also sehr gut, da bei hoher Genuigkeit nur wenig Rechenleistung beansprucht wird. Das wird besonders im Vergleich zum Farbfilter deutlich, der für jedes Bild eine Laufzeit von fast 10 Sekunden hatte. Dazu muss aber gesagt werden, dass das Programm dabei auf dem Zielgerät Raspberry Pi ausgeführt wurde. Unserer Meinung nach hätte das KNN jedoch selbst auf dem Raspberry Pi eine kürzere Laufzeit.

Zum letzten Missionsziel kann keine Aussage gemacht werden, da wir mit der Implementierung nicht bis zur Einbindung des Netzes in Rust gekommen sind.

Abschließend kann also gesagt werden, dass die Vegetationsbranderkennung mit Künstlichen Neuronalen Netzen trotz einem kleinen Trainingsdatensatz gut funktioniert. Die Genauigkeit kann noch verbessert werden und ist nicht wie geplant ins ScOSA integriert worden, doch das Hauptziel wurde erreicht und Brände können zuverlässig erkannt werden.

Nr.	Ziel
MG-01	Die Mission wird zeigen, ob sich KNN eignen, um Satellitenbilder nach Vegetationsbränden zu untersuchen.
MO-01	Die Mission wird zeigen, dass ein kompaktes KNN besser geeignet ist, als andere Algorithmen um Vegetationsbrände zu erkennen.
MO-02	Die Mission wird zeigen, ob KNN zu rechenintensiv sind im Vergleich zur Genauigkeit.
MO-03	Die Mission wird zeigen, dass Rust auch für Neuronale Netze verwendbar ist.

Abbildung 5: Die Missionsziele

```
(base) [moritzlechner@fedora Abgabe]$ python test-single-image.py ../fire005.bmp -cp
Using Device:  cpu
Model summary (fused): 218 layers, 25840339 parameters, 0 gradients, 78.7 GFLOPs
Ultralytics YOLOv8.0.25 🚀 Python-3.9.12 torch-1.13.1+cu117 CPU

0: 640x640 3 fires, 324.4ms
Speed: 1.5ms pre-process, 324.4ms inference, 1.1ms postprocess per image at shape (1, 3, 640, 640)
Detected Fires: 3

Fire 1
Confidence:  0.71
Coordinates: TL:(x=749.0, y=204.0)
             BR:(x=812.0, y=290.0)

Fire 2
Confidence:  0.71
Coordinates: TL:(x=63.0, y=808.0)
             BR:(x=371.0, y=985.0)

Fire 3
Confidence:  0.49
Coordinates: TL:(x=328.0, y=701.0)
             BR:(x=437.0, y=756.0)
```

Abbildung 6: Die Ausgabe des Programms in Form der Parameter für das Bild 4

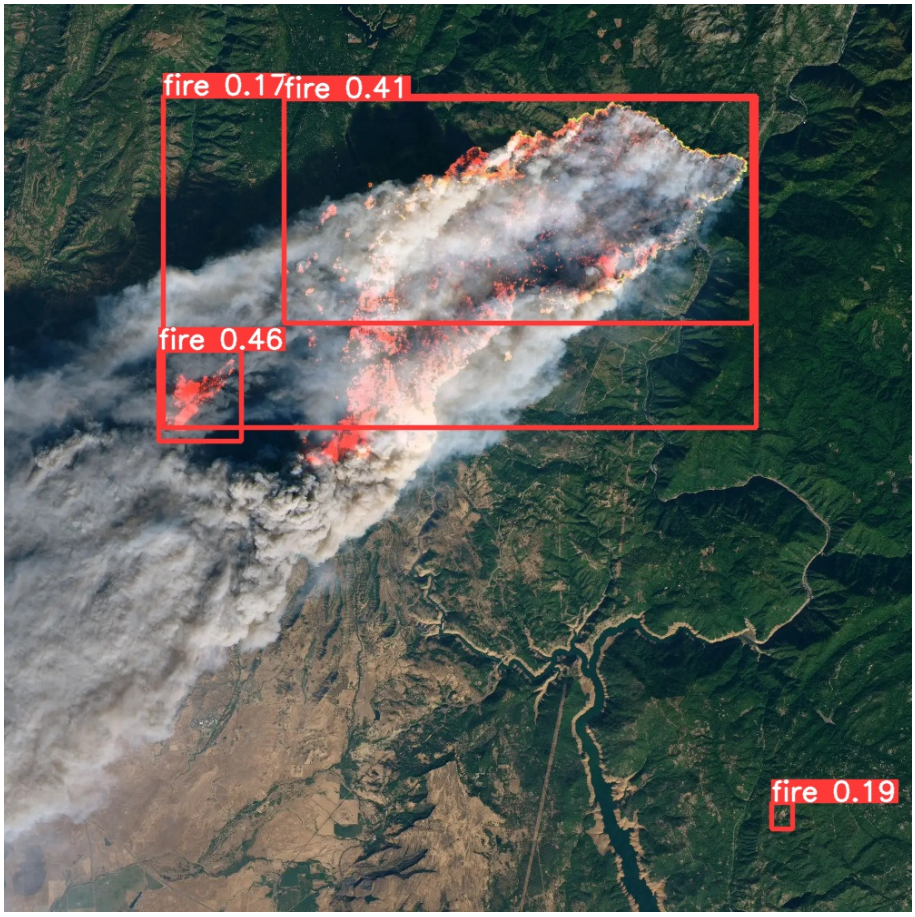


Abbildung 7: Beispiel Ergebnis des Netzes

Glossar

Bildklassifizierung Ein Verfahren, bei dem die Art des Bildinhalts in unterschiedliche Kategorien unterschieden wird (klassisches Beispiel: Hund oder Katze). 3

Bounding Box Aus dem Englischen und bedeutet in etwa Grenz Box. In unserem Anwendungsfall werden damit die Feuer markiert. 3

Brand Mit sämtlichen erwähnten Bränden sind Vegetationsbrände, z.B. Waldbrände gemeint. 3

Confidence Die Sicherheit, mit welcher das Netz seine Aussage trifft. 4

Datensatz Eine Sammlung an vielen Daten der gleichen Art, in unserem Fall Satellitenbilder. 3

False Negatives Tritt auf, wenn die Einschätzung fälschlicher Weise Negativ war. Das richtige Ergebnis wäre also Positiv gewesen. 5

False Positives Tritt auf, wenn die Einschätzung fälschlicher Weise Positiv war. Das richtige Ergebnis wäre also Negativ gewesen. 4

Gewichte Parameter des KNN, welche beim Trainieren angepasst werden, und mit denen das Netz letzendlich seine Entscheidungen trifft. 3

Künstlichen Neuronalen Netzen (KNN) Ein Algorithmus der mithilfe eines vorher erstellten Datensatzes "trainiert" wird. Nach dem Training kann das KNN auf das gelernte zugreifen und mit neuen Daten vergleichen. 3

Label Ein Label ist in unserem Fall ein Wert oder auch ein Bild, von dem bekannt ist ob es sich um einen Brand handelt oder nicht. 2, 5

Modell In unserem Fall bezeichnet es die innere Struktur des KNN. 3

Objekterkennung Ein Verfahren, bei dem unterschiedliche Objekte auf einem Bild erkannt und markiert werden können). 3

Prediction Englisch für Vorhersage. 2, 5

Raspberry Pi Der Raspberry Pi ist ein Einplatinencomputer. 7

ScOSA Scaleable On-Board System for Avionics. 8

Testbatches Batch ist englisch für Stapel. Testbatches sind für uns also Stapel an Testbildern. 5

Validation Task Ein Test, der im Modell integriert ist und die Funktionsfähigkeit überprüft. 4