Entwicklung einer Funktion zur Überwachung der Bilderklassifizierung neuronaler Netze

Report Deep Learning

Studiengang Elektrotechnik

Studienrichtung Fahrzeugelektronik

Duale Hochschule Baden-Württemberg Ravensburg, Campus Friedrichshafen

von

Maximilian Mosbacher

Abgabedatum:	13.06.2022
Matrikelnummer:	9223075
Kurs:	TFE19-2
Gutachter der Dualen Hochschule:	Mark Schutera

1 Einleitung

1.1 Problemstellung

Im Rahmen der Validierung eines zuvor trainierten neuronalen Netzes werden Validierungsdaten klassifiziert. Hierzu wir mit jedem validierten Bild eine Matrix, bestehend aus Wahrscheinlichkeitswerten für die Übereinstimmung des Validierungsbildes mit jeder möglichen Klasse, generiert. Löst ein Bild nun nahezu identische Wahrscheinlichkeiten für mehrere Klassen aus kann dies zu fehlerhaften Klassifizierungen führen. Die mit dieser Dokumentation einhergehende Funktion soll dem Entwickler diese mögliche Fehlerquelle rückmelden und soll ihm die Möglichkeit geben die Wahrscheinlichkeitswerte visuell nachzuvollziehen.

1.2 Vorwissen

In diesem Kapitel werden zunächst Grundlagen und Fähigkeiten erläutert, welche vor der Umsetzung des Programmentwurfes angeeignet wurden. Ziel der ersten Vorlesung zum Thema Deep Learning war das Aufsetzen und Kennenlernen der zukünftig verwendeten Projektstruktur. Um das Projekt aus einem GitHub Repository laden und anschließend auch damit arbeiten zu können, musste zunächst Python inklusive einer passenden Entwicklungsumgebung (auch engl. IDE genannt) auf dem Computer installiert werden. In diesem Fall wurde die IDE Visual Studio Code, aufgrund der übersichtlichen Darstellung mithilfe zusätzlicher Programmierunterstützungen, gewählt. Zudem war hier der

Umgang mit dem Debugger bereits bekannt. Mit einer aktuellen Python Version konnten anschließend erforderliche Bibliotheken heruntergeladen und installiert werden. In diesem Projekt des Deep Learnings war die Bibliothek *tensorflow* besonders wichtig, da diese wichtige Funktionen zum Trainieren und Validieren neuronaler Netze beinhaltet. Zum Training des neuronalen Netzwerks benötigt es Daten, hier: Bilder unterschiedlicher Verkehrsschilder, welche bereits in Klassen definiert wurden. Mit heruntergeladenen Daten konnte das Netz nun trainiert werden. Ein wichtiger Hinweis hierbei war es, die Zahl der zu trainierenden Epochen zunächst von 50 auf eine zu reduzieren, da das Training des neuronalen Netzes auf dem verwendeten PC über 30 Minuten pro Epoche dauert. Abschließend mit der ersten Vorlesung wurde das Netz eine Epoche trainiert.
Zu Beginn der zweiten Vorlesung konnte direkt das trainierte Netz mit dem dafür vorgesehenen Programm *validation pipeline* validiert werden. Das Programm gibt die Genauigkeit, die vom trainierten Netz erkannten Klassen, und die tatsächlich verwendeten Klassen neuer, dem Netz bisher unbekannte Daten an. Hierbei fiel die deutlich zu niedrige Genauigkeit der Validierung von nur ca 5 Prozent auf. Wie sich später herausstellte wurden die Trainingsdaten in einer falschen Ordnerstruktur angelegt und dadurch das Netz gänzlich falsch trainiert. Nach Berichtigung der Struktur im Trainingsordner konnte das Netz erneut trainiert werden und die Genauigkeit der Validierung auf 85 Prozent gesteigert werden.
Die Klassifizierung eines Bildes durch das Netz erfolgt auf Basis der trainierten Daten. Hierbei werden für jedes Bild alle Klassen plausibilisiert und mit einer Wahrscheinlichkeit zur Übereinstimmung mit dem Netzwissen ausgegeben. Aufgrund des gering trainierten Netzes können bei dieser Klassifizierung Fehler auftreten. Nicht selten werden dabei mehrere Klassen mit ähnlich hohen Wahrscheinlichkeiten versehen. Ist dies der Fall könnte es durchaus möglich sein, dass sich hierbei für die falsche Klasse entschieden wurde. Dieses Projekt soll solche Situationen erkennen und den Entwickler davor warnen, dass andere Klassen für die jeweiligen Bilder ähnlich plausibel sind. Dies wird zusätzlich durch eine Visualisierung veranschaulicht, um knappe Entscheidungen zu verdeutlichen. Grenzwert sind hierfür 10 Prozent Differenz zwischen gewähltem und nächsthöchsten Werten. Da es sich hierbei um eine Warnung und keine Fehlermeldung handelt, wird der Handlungsspielraum relativ groß gewählt, um mögliche Fehlerquellen ausschließen zu können.

2 Umsetzung

Die in diesem Projekt entwickelte Funktion dient der Überwachung der Wahrscheinlichkeitsmatrizen, welche für jedes validierte Bild angelegt werden. Besitzen zwei oder mehrere Werte ähnliche Wahrscheinlichkeiten, so wir der Entwickler vor möglichen Fehlinterpretationen gewarnt.
Die Funktion bekommt vom Validierungsprogramm alle Wahrscheinlichkeitsmatrizen übergeben. Im ersten Schritt werden aus der Gesamtzahl der Matrizen mithilfe einer for-Schleife die einzelnen Matrizen extrahiert. Anschließend wird jede einzelne Matrix mithilfe einer *tensorflow*-Funktion namens *softmax* mit realistischen Wahrscheinlichkeiten ersetzt, da die Werte zuvor nicht die Summe 1 pro Matrix hatten. Daraufhin wird die maximale Wahrscheinlichkeit in der einzelnen Matrix bestimmt. Dieser Maximalwert wird beim darauffolgenden durchlaufen der einzelnen Matrix mit den restlichen Wahrscheinlichkeiten verglichen. Ergibt sich eine Differenz, die unter den Grenzwert 0.1 fällt, so wird dies dem Entwickler als Warnung ausgegeben. Zusätzlich wir das Bild in eine Liste gespeichert.

Zur Visualisierung der problematischen Bilder wird auf eine weitere Python-Bibliothek *matplotlib* zurückgegriffen. Diese ermöglicht die Darstellung aller Wahrscheinlichkeitswerte eines Bildes in Form von Histogrammen. Dafür wird die Liste der problematischen Bilder durchlaufen und die jeweiligen Wahrscheinlichkeitsmatrizen ausgegeben. Abschließend werden alle Histogramme in einen dafür angelegten Ordner abgespeichert. Beim erneuten Ausführen der Funktion werden zunächst alte Bilder gelöscht bevor neue abgespeichert werden.

Abbildung 2.1 zeigt ein solches Histogramm, bei dem zwei Wahrscheinlichkeiten annähernd gleich groß bzw gleich plausibel sind. Durch die Funktion wird der Entwickler vor

möglichen Fehlinterpretationen des Netzes gewarnt und kann darauf manuell oder durch erweitertes Training des Netzes reagieren. Die Anhänge A.1, A.2 und A.3 zeigen weitere Beispiele für mögliche Fehler der Klassifizierung.

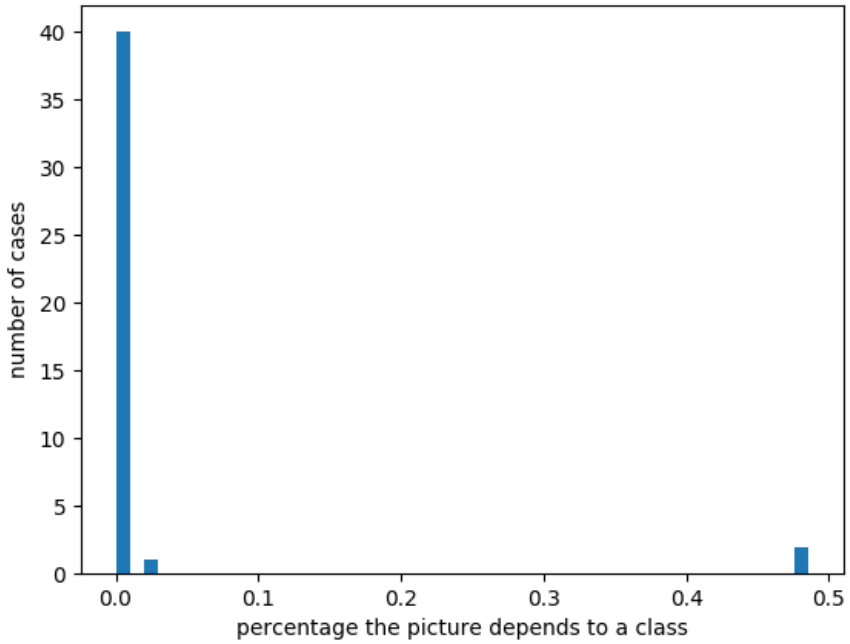

Abbildung 2.1: Nahezu gleiche Wahrscheinlichkeiten für Bild 42 von Batch 5

Zusätzlich wurde eine weitere Funktion hinzugefügt, welche dem Entwickler Rückmeldung über die Häufigkeit der Warnungen pro Klasse gibt. Hierzu werden wie schon zuvor bei Unterschreiten des Schwellwerts von 10 Prozent die Klassen in eine Liste gespeichert, welche dann mit *matplotlib* in Form eines Histogramms ausgegeben wird. In der Beispielabbildung 2.2 ist ersichtlich, dass Klasse 21 die meisten Warnungen auslöst. Hierdurch kann der Entwickler nun besonderes Augenmerk auf diese und andere häufig warnende Klassen richten, sowie deren Ursachen herausfinden.

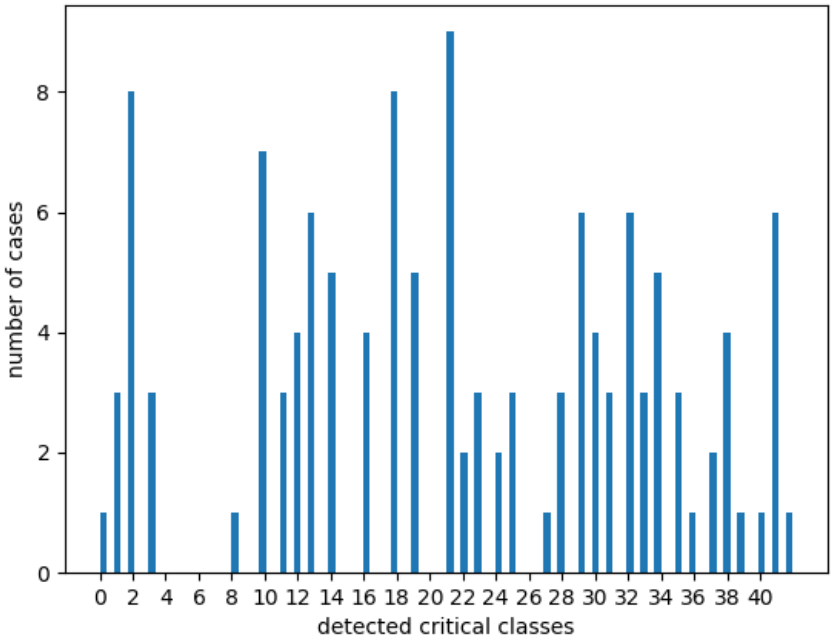

Abbildung 2.2: Histogramm zur Veranschaulichung der je Klasse aufgetretenen Warnungen (Batch 5)

Anhang A

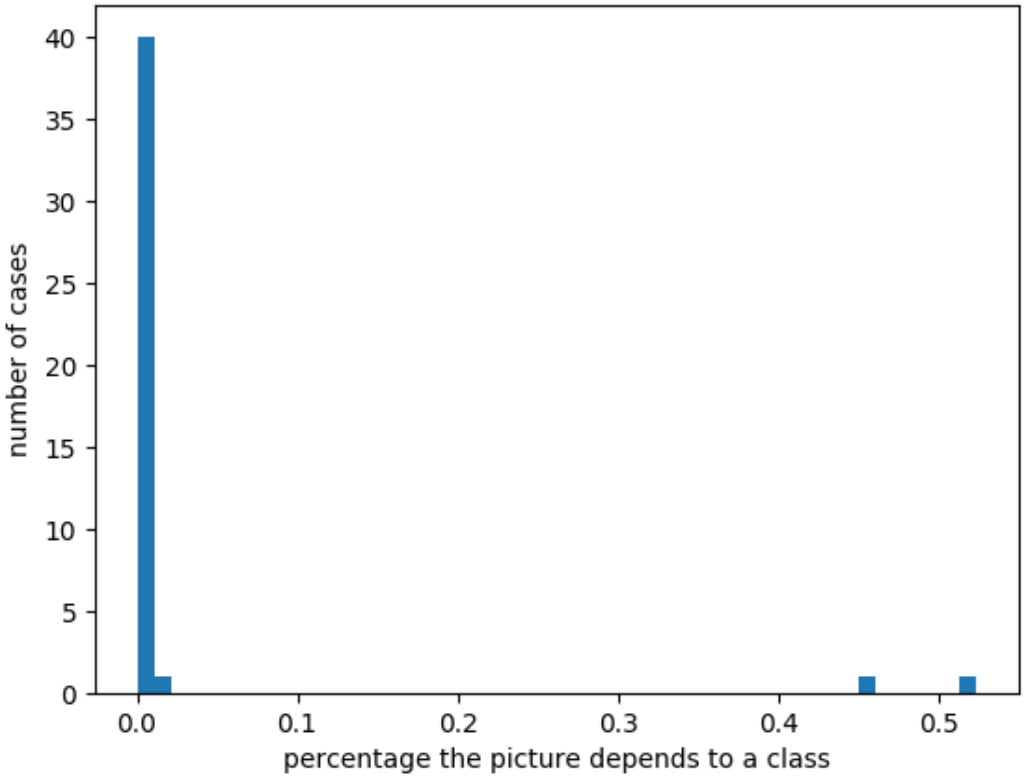

Abbildung A.1: Nahe beieinander liegende Wahrscheinlichkeiten für Bild 41 von Batch 5

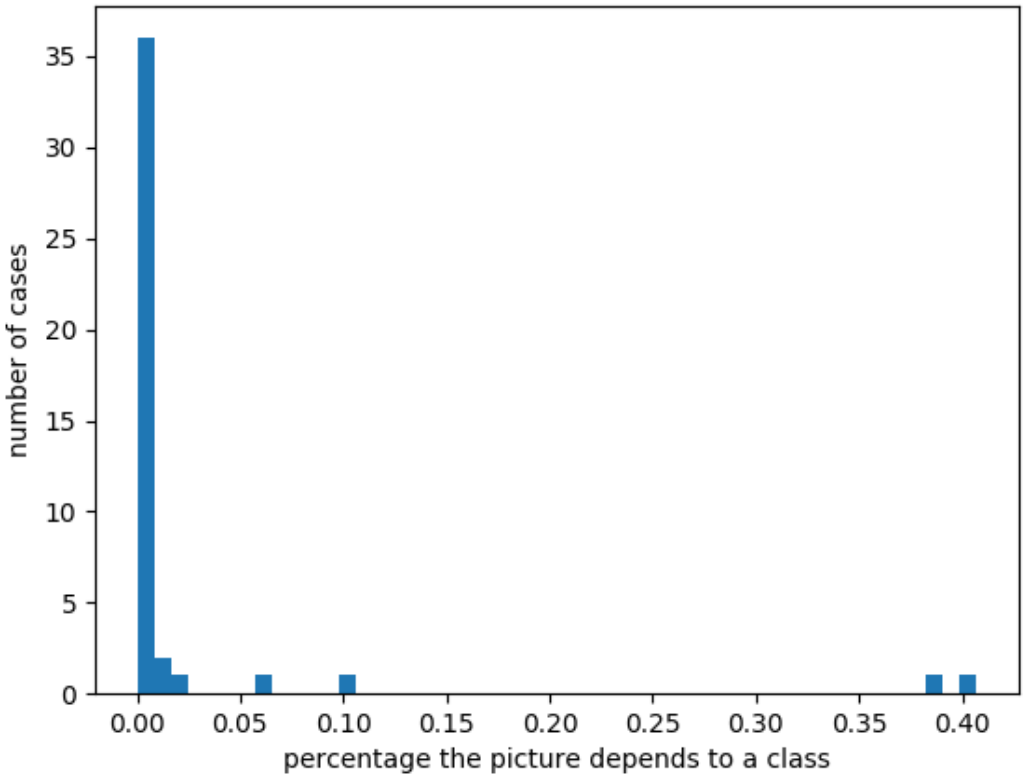

Abbildung A.2: Nahe beieinander liegende Wahrscheinlichkeiten für Bild 47 von Batch 5

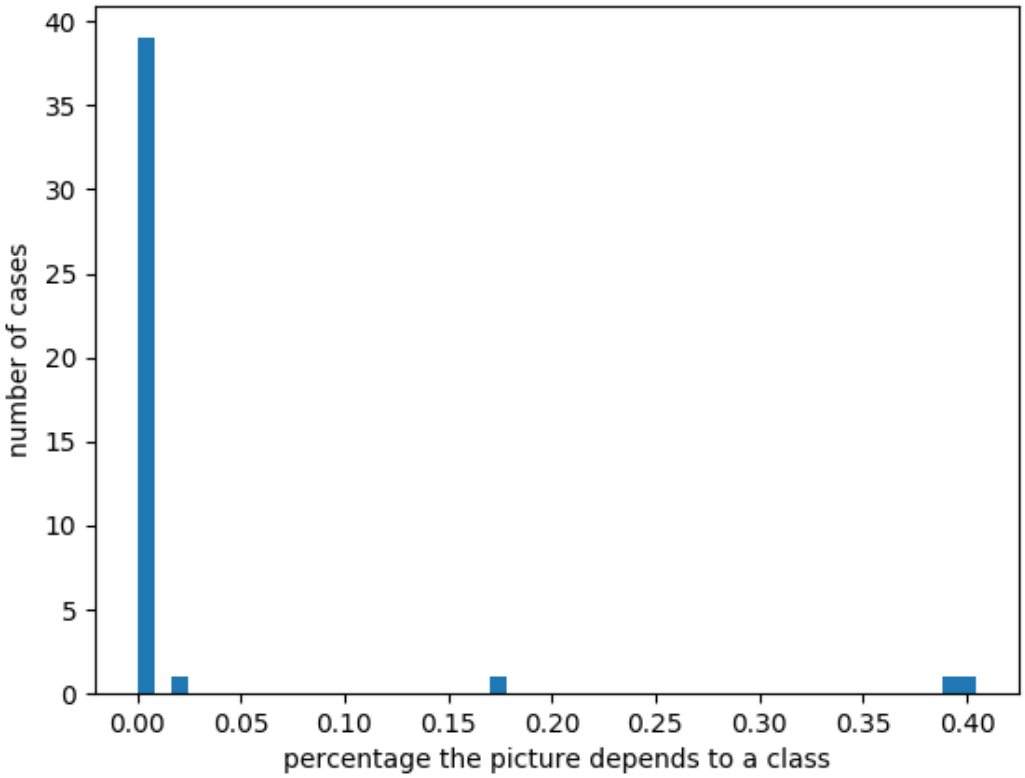

Abbildung A.3: Nahe beieinander liegende Wahrscheinlichkeiten für Bild 112 von Batch 5