Lineare Algebra, Datenanalyse und maschinelles Lernen
Prüfungsvorleistung

Aufgabe:

In Python soll mittels Tensorflow ein Convolutional Neural Network erstellt und auf dem FER2013-Datensatz trainiert werden, mit dem Ziel auf einer Testmenge des Datensatz eine Genauigkeit von mindestens 60% zu erreichen. Dabei sollen die Änderungen der Hyperparameter des CNN, sowie die Auswirkung von Data Augmentation dokumentiert werden.

Allgemeines:

Der FER2013-Datensatz enthält 35887 Graustufenbilder von Gesichtsausdrücken, welche nach 7 Emotionen klassifiziert werden. Das CNN wird dabei nur auf die Klassifizierung einer Teilmenge (80%) des gesamten Datensatz trainiert. Das Ziel des Trainings soll jedoch sein, dass das CNN das auf der Trainingsmenge erlernte Wissen möglichst weit generalisiert und somit in der Lage ist, auch die im Training nicht verwendeten Bilder ebenso gut zu klassifizieren.

Vermieden werden soll der Fall des Overfitting, erkenntlich an einer deutlich besseren Genauigkeit auf den Trainingsdaten als auf den Validierungs- bzw. Testdaten. Es tritt auf, wenn das Netzwerk beginnt sich die Bilder aus dem Trainingsdatensatz sinnbildlich einzuprägen und die Klassifizierung anhand von spezifischen, schlecht verallgemeinerbaren Merkmalen vornimmt. Das Overfitting kann durch einen größeren Trainingsdatensatz verzögert werden, da es somit schwieriger ist, sich einzelne Bilder einzuprägen. Data Augmentation bezeichnet die scheinbare Erweiterung eines Datensatz um individuelle Datenpunkte aus den bestehenden Daten heraus, ohne tatsächlich gänzlich neue Datenpunkte hinzuzufügen. Bei Bilddaten, wie z.B. im Fall von FER2013, bietet es sich an Bilder zu spiegeln, zu drehen, oder um eine geringe Anzahl an Pixeln zu verschieben. Data Augmentation kommt in dieser Arbeit erst bei den Schritten 6 und 7 zum Einsatz. Hier wird auf die Thematik noch einmal genauer eingegangen.

Die Modelle zu den einzelnen Entwicklungsschritten finden sich im Python-Script unter der jeweiligen create_model_x() -Funktion, sowie unter anderem die jeweils verwendete Lernrate, Batch-Größe und Testgenauigkeit. Zu jedem Entwicklungsschritt wird ein Plot abgebildet, der den Verlauf von Kostenfunktion und Genauigkeit auf der Trainings- und Validierungsmenge, über die Epochen des Trainings, angibt. Als Kostenfunktion wird die Log-loss (auch binary cross-entropy) Funktion verwendet, welche bei der Klassifizierung in 0 oder 1 benutzt werden kann. Da eine Zielaktivierung von 1, im Fall von FER2013, immer nur bei einer der 7 Klassen zutrifft, kann die Sparse- Variante der Kostenfunktion, sowie die zugehörige Genauigkeit, verwendet werden.

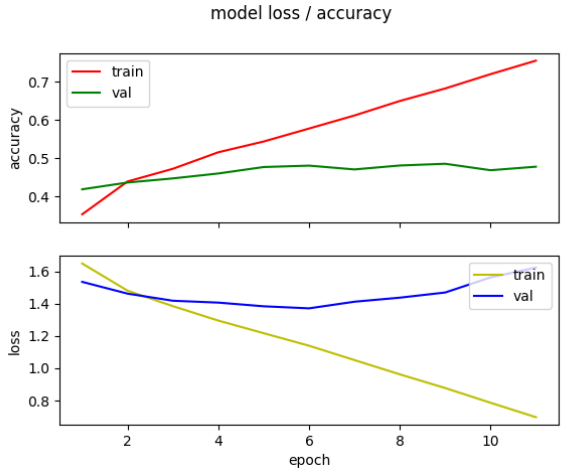
Die Lernrate und Batch-Größe werden lediglich in Schritt 4 leicht verändert und ansonsten bei üblichen Werten von 0.001, bzw. 32 belassen. Bei einer deutlich höheren Lernrate kommt kein sinnvolles Training zustande, eine geringere Lernrate verlangsamt das Training und bringt eher einen Vorteil beim Fine Tuning eines bereits trainierten Modells. Da in dieser Arbeit aber grundlegendere Änderungen mit größeren Auswirkungen betrachtet werden sollen, kommt diesen Hyperparametern hier weniger Beachtung zu.

Schritt 0:

Ein Convolutional Neural Network ist eine Form Artificial Neural Network mit mindestens einem Convolutional Layer. Es unterscheidet sich von einer dichten Schicht insofern, dass jedes Neuron nur mit einer kleinen lokalen Gruppe (definiert durch Kernel) von Neuronen aus der vorhergehenden Schicht verbunden ist. Zudem teilen sich die Verbindungen Gewichte zu relativ gleichen Positionen im Kernel. So könnten mit dem passenden Kernel zum Beispiel vergleichsweise effizient Kanten innerhalb des Bildes identifiziert werden. Um in einer Schicht mehrere solcher Features zu markieren, besitzen Convolutional Layer meist mehrere Ebenen um mehrere Feature Maps darzustellen.

Convolutional Layer werden üblicherweise gefolgt von einem Pooling Layer (typ. MaxPooling), welches eine lokale Gruppe von Neuronen auf eine bestimmte Weise zusammenfasst und somit den Datenumfang und die Größe des Netzwerks reduziert.

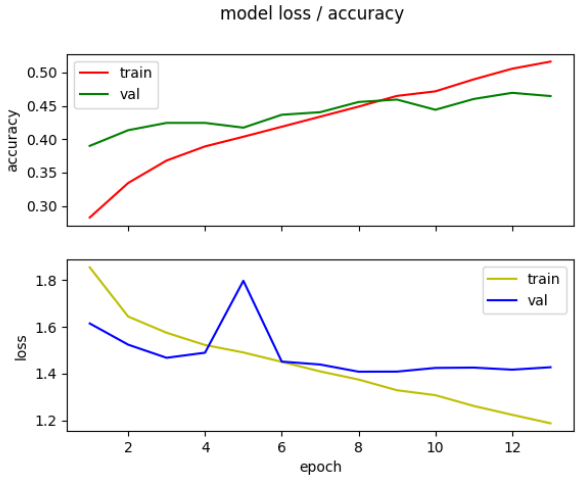
Zuletzt werden die extrahierten Feature Maps noch einmal von einer dichten Schicht verarbeitet und schließlich die Klassifizierung vorgenommen.



Das Modell aus Schritt 0 stellt also ein CNN mit minimaler Tiefe dar, welches die Definition erfüllt. Hierbei tritt bereits ab der 3. Epoche ein deutliches Overfitting auf: Die Genauigkeit auf den Trainingsdaten nimmt nahezu linear weiter zu, während die Validierungsgenauigkeit ca. 48% nicht mehr überschreitet. Schlussendlich steigt der Loss auf der Validierungsmenge wieder an. (Von hier an Spekulationen!) Grund dafür ist vermutlich, dass durch die wenigen Schichten sehr große Gradientenschritte realisiert werden können, worauf womöglich auch der anfangs sehr schnelle Genauigkeitszuwachs hindeutet, welche dann aber auch eine schnellere Überanpassung an die Trainingsdaten zur Folge haben. Bei weniger tiefen neuronalen Netzen sind für die Bildung des Gradienten weniger Backpropagation-Schritte notwendig, sodass eventuell der Bezug zu dem an der Ausgabeschicht berechneten Fehler noch stärker ist. Hinzu kommt, dass bei diesem Modell besonders in der ersten dichten Schicht sehr viele freie Parameter zur Verfügung stehen, was das Einprägen spezifischer Merkmale begünstigt.

Schritt 1:

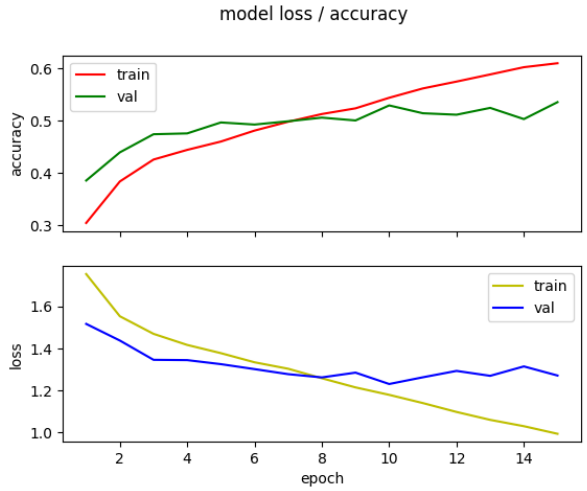
Hinzugefügt wurde ein zweites Convolutional Layer nach dem ersten, um eine ausführlichere Verarbeitung der Eingabe und Extraktion von Features zu ermöglichen. Direkt nach jedem einzelnen Convolutional Layer wurde jedoch zunächst ein BatchNormalization Layer eingefügt, welches Eingabe und Gradient über jeden Batch normalisiert und so das Training stabilisieren und beschleunigen soll [1]. Zudem soll das nach der ersten dichten Schicht eingefügte Dropout Layer ein Overfitting verzögern, indem Neuronen für jeden Gradientenschritt mit einer Wahrscheinlichkeit von 50% nicht beachtet werden.



In diesem Schritt tritt das Overfitting erst nach 9 Epochen auf, der längere Anstieg der Validierungsgenauigkeit bringt aber noch keinen Vorteil gegenüber Schritt 0.

Schritt 2:

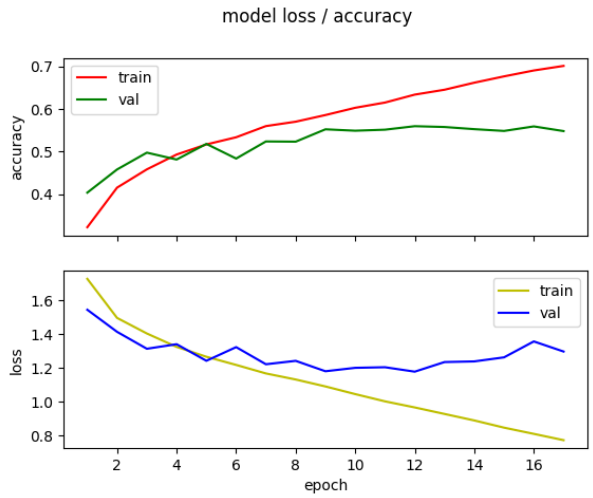
Für Schritt 2 wurde ein zweiter Block von Convolutional Layern eingefügt. Die immer weitere Schmälerung des Eingangsbildes in späteren Convolutional Layern ermöglicht wiederum auch die Erhöhung der Anzahl an Feature-Maps, sodass dennoch ein sinnvoller Umfang von Daten in den dichten Schichten zur Klassifizierung zur Verfügung steht. Somit werden in tieferen Convolutional Layern idealerweise immer komplexere Features des Eingangsbildes abgebildet.



Der Verlauf der Kostenfunktion auf der Validierungsmenge ist nun stabiler und der längere Anstieg der Genauigkeit führt letztlich zu einer deutlich besseren Testgenauigkeit.

Schritt 3:

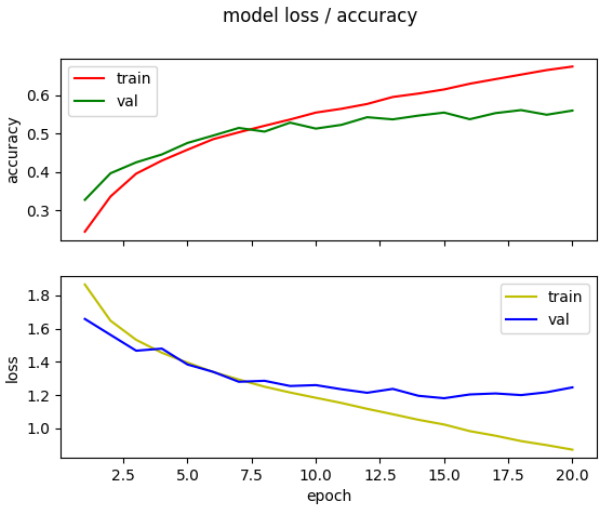
Hier wird noch einmal der letzte Schritt wiederholt und erneut steigt die Testgenauigkeit an.



Lediglich im letzten Convolutional Layer wurde die Größe des Kernels verringert. Dadurch soll verhindert werden, dass durch das anschließende Pooling Layer Daten verloren gehen. Da ein Pooling Layer mit Kernel und Stride von (2, 2) gerade die Halbierung von Eingangsbreite und -Höhe bewirkt, bei Teilungsresten aber abgerundet wird, würde die äußerste Reihe und Spalte von Werten aus dem vorherigen Layer für die weitere Verarbeitung entfallen. Besonders in tieferen Schichten macht ein solcher Verlust einen großen Anteil der Feature Maps aus. Es soll daher auch in den folgenden Schritten auf diesen Aspekt geachtet werden.

Schritt 4:

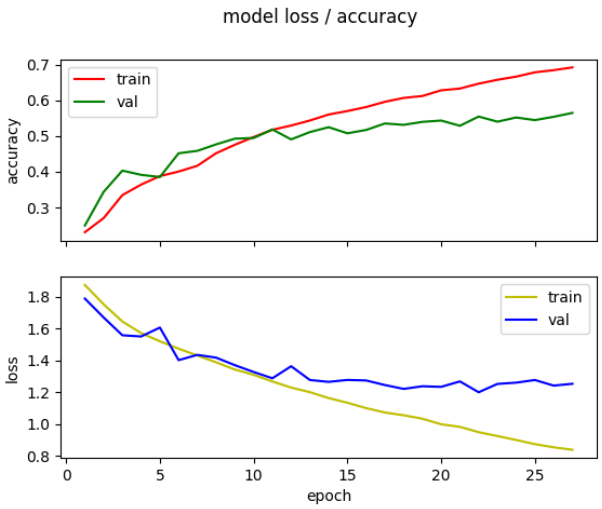
Dieser Schritt nutzt nahezu die gleiche Anordnung von Schichten, aber am Anfang eine größere Zahl von Feature Maps, sowie eine zusätzliche dichte Schicht.



Wichtig ist, dass bei diesem Schritt nur die halbe Batchgröße und Lernrate eingestellt wurden, wovon ein stabilerer Verlauf des Trainings erwartet wird. Dies spiegelt sich auch im Plot wider. Allerdings verzögert der langsamere Anstieg deutlich die gesamte Trainingszeit und erzielt gleichzeitig keine höhere Testgenauigkeit als der vorherige Schritt. Für die weiteren Schritte werden daher wieder die ursprüngliche Batchgröße und Lernrate genutzt.

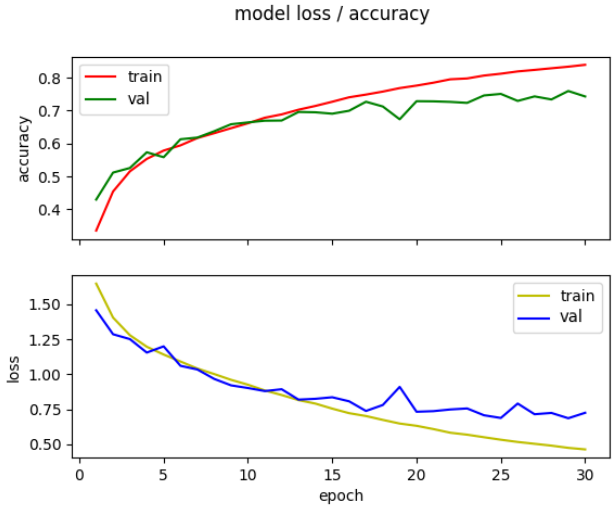
Schritt 5:

Hier wird nochmals ein weiterer Block von Convolutional Layern angefügt. Der Aufbau des Modells ist somit ähnlich dem im Paper [2] dargestellten Modell. Zusätzlich mussten mehrere ZeroPadding Layer eingefügt werden, um zu gewährleisten, dass die Breite und Höhe der Feature Maps nach dem letzten Pooling Layer nicht auf null reduziert wird. Durch das ZeroPadding werden die Feature Maps an den Rändern um Neuronen mit konstanter Aktivierung von 0 erweitert. Dadurch werden tiefere CNN ermöglicht, gleichzeitig verringern sie aber auch den Anteil der tatsächlich aus dem Eingabebild abstrahierten Daten innerhalb der Feature Maps, weshalb sie, wenn möglich, eher vermieden werden sollten.



Schritt 6:

Da sich die Genauigkeit in Schritt 5 nicht sonderlich gegenüber Schritt 3 und 4 verbessert hat, aber pro Epoche deutlich mehr Zeit beansprucht wurde und wie zuvor erklärt Padding Layer eher vermieden werden sollen, wird hier die vorherige Architektur weiter genutzt.

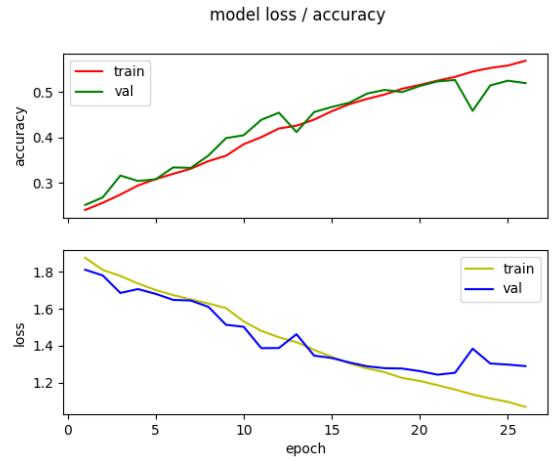


In Schritt 6 sollte nun schließlich die 60%ige Genauigkeit erreicht werden. Dazu wurde noch einmal die Zahl der Feature Maps deutlich erhöht. Zudem wurde nun Data Augmentation eingesetzt, genauer die Spiegelung der Bilder, sowie die zufällige Verschiebung entlang der x-Achse um jeweils einen Pixel.

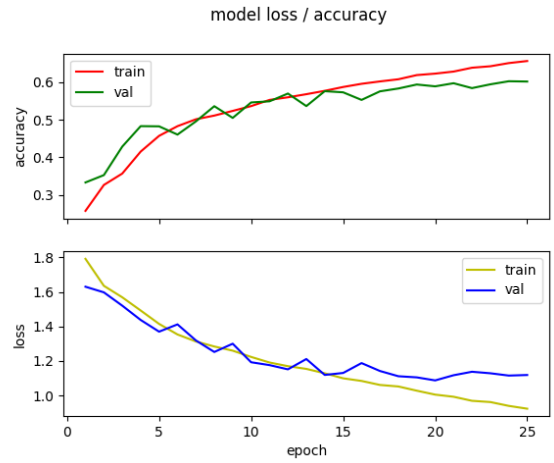
Das Resultat von diesem Durchlauf ist überraschend gut mit einer Verbesserung der Genauigkeit von fast 20%. Das Overfitting tritt erst nach ca. 16 Epochen auf und auch danach steigt die Validierungsgenauigkeit weiter an.

Um den plötzlichen Anstieg der Genauigkeit besser nachvollziehen zu können betrachtet Schritt 7 an einem kleineren Modell nochmals genauer die Auswirkung von Data Augmentation.

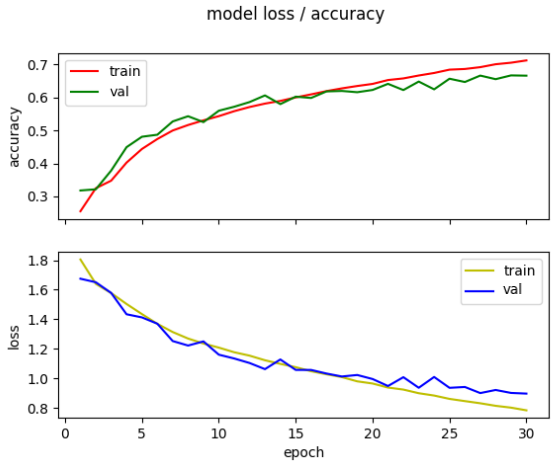
Schritt 7:



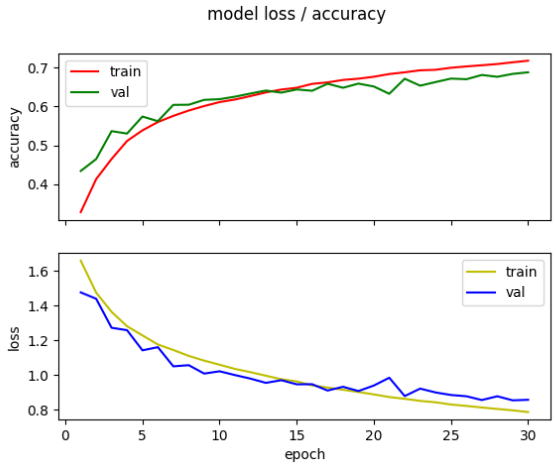
Zunächst wird als Vergleichswert das Modell noch einmal ohne Data Augmentation trainiert. Die Testgenauigkeit liegt hier bei 50.13%. Bei diesem Durchlauf steigt auch die Trainingsgenauigkeit nur sehr langsam an, vermutlich da das Modell aus allen Versuchen die geringste Anzahl von Parametern besitzt.



Für den nächsten Versuch wurden die Bilder an der y-Achse gespiegelt. Da pro Epoche der doppelte Umfang von Bildern genutzt wird, verdoppelt sich einerseits die Trainingsdauer, andererseits steigen aber auch beide Genauigkeiten schneller an. Letztlich kann die Testgenauigkeit auf 59.87% gesteigert werden.



Der dritte Versuch nutzt nur die Verschiebung der Bilder entlang der x-Achse. Die Testgenauigkeit erreicht hierbei sogar 66.64%. Zudem zeigt der Plot das Potential das auch nach der 30.ten Epoche noch die Validierungsgenauigkeit ansteigen könnte, hier wurde das Training also womöglich zu früh abgebrochen.



Im letzten Versuch kommen beide Formen der Data Augmentation zum Einsatz. Dadurch ergibt sich nochmals eine Steigerung um 2% gegenüber dem vorherigen Versuch auf 68.77%. Da der Datensatz nun allerdings den vierfachen ursprünglichen Umfang hat, nimmt auch das Training erheblich mehr Zeit in Anspruch.

Fazit:

Die aus dieser Arbeit gewonnen Erkenntnisse lassen sich zusammenfassen in solche zum Aufbau des Netzwerkes und solche zu Data Augmentation.

Bezüglich des Aufbaus eines Convolutional Neural Network kann beobachtet werden, dass tiefere Netzwerke mit mehr Convolutional Layern bessere Genauigkeiten erzielen als weniger tiefe Netzwerke. Gleichzeitig wird die Zahl der benötigten Parameter des Netzwerkes drastisch reduziert. Auf der anderen Seite beanspruchen CNNs mit vielen Schichten jedoch beim Training deutlich mehr Zeit (mehr Zeit pro Epoche und insgesamt mehr Epochen) als ein überwiegend aus dichten Schichten aufgebautes Netzwerk, trotz der geringeren Anzahl an zu trainierenden Parametern.

Bezüglich Data Augmentation lässt sich feststellen, dass beide Formen den Datensatz zu erweitern eine erhebliche Verbesserung der Genauigkeit bewirkt haben. Der FER2013 Datensatz scheint verglichen mit anderen Bilddatensätzen, wie z.B. MNIST, deutlich mehr zum Overfitting zu verleiten. Wie eingangs beschrieben trägt Data Augmentation daher intuitiv einen erheblichen Teil dazu bei, die Genauigkeit auf diesem Datensatz zu verbessern. Dabei scheint besonders die leichte Verschiebung der Bilder einen großen Vorteil zu bringen.

Schlussendlich ist es somit möglich auf dem FER2013 Datensatz ein Modell mit ca. 100‘000 Parametern zu trainieren, welches die geschätzte menschliche Genauigkeit auf dem Datensatz [3] erreicht und somit auch die als Ziel gestellte Genauigkeit von 60% übertreffen kann.

Quellen:

[1] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in 32nd International Conference on Machine Learning, ICML 2015, vol. 1

[2] Yousif Khaireddin and Zhuofa Chen, “Facial Emotion Recognition: State of the Art Performance on FER2013”, https://arxiv.org/ftp/arxiv/papers/2105/2105.03588.pdf

[3] https://www.kaggle.com/competitions/challenges-in-representation-learning-facial-expression-recognition-challenge/discussion/4402#23304