6 CNNs für Computer Vision

Systemtechnik BSc
FS 2025

Aufgaben

Applied Neural Networks im Modul ANN | WUCH

Lernziele

Nach dem Bearbeiten dieser Übungsserie ...

- verstehen Sie den Aufbau von ResNet, die Wirkungsweise von *residual connections* und von *Batch Normalization Layern*.
  Aufgaben: 1
- sind Sie in der Lage, grosse neuronale Netze auf *Google Colab* zu trainieren.
  Aufgaben: 1
- wissen Sie, wie durch das Einführen von *residual connections* und *Batch Normalization* Lagen das *vanishing gradient Problem* entschärft, und durch geschickte *Initialisierung* (He, Xavier) das Training beschleunigt werden konnte.
  Aufgaben: 1 2

Residual Nets

Das Vanishing Gradient-Problem

Das VGG16-Netz wurde von Simonyan und Zisserman für den Wettbewerb ILSVRC (Image Net Large Scale Visual Recognition Challenge) 2014 entwickelt. VGG steht für Visual Geometry Group an der Universität Oxford. Es besteht aus 16 Gewichtungsschichten (13 Faltungsschichten und 3 voll verknüpfte Schichten) mit nur 3x3 Merkmalsdetektoren oder Filtern. Die Klassifikationsfehlerrate für diese Architektur lag bei

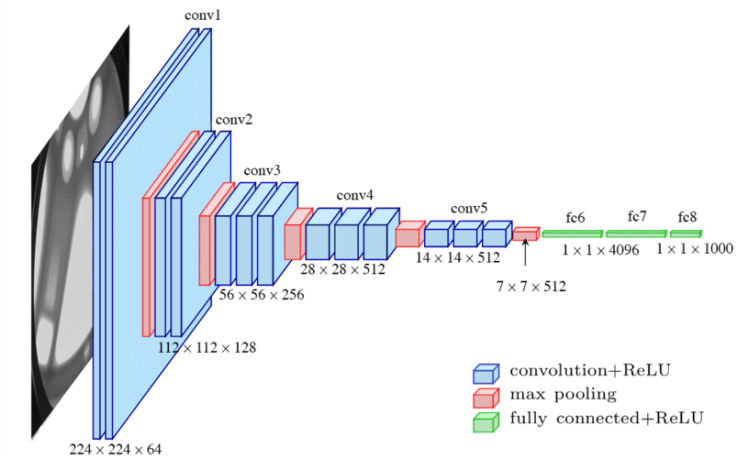

Abbildung 1: VGG16 Architektur

etwa 7 %. Das Konzept des VGG-Netzes ähnelt dem von Alexnet, d.h. mit zunehmender Tiefe des Netzes wird die Anzahl der Merkmalskarten oder der Faltungen erhöht. Je tiefer wir in das Netz eindringen, desto grösser wird die Anzahl der Merkmalskarten. Das Problem bei der VGG Architektur ist die grosse Anzahl hintereinander geschalteter Schichten und damit die grosse Anzahl von Parametern. Dies erhöht die Komplexität des Modells. Mit zunehmender Tiefe des neuronalen Netzes wird die Genauigkeit gesättigt und nimmt dann ab einem bestimmten Punkt rapide ab. Diese Verschlechterung ist überraschenderweise nicht auf eine Überanpassung zurückzuführen. Das Hinzufügen von immer mehr Schichten zu diesen tiefen Modellen führt zu höheren *Trainingsfehlern*. Ein weiteres Problem war das Problem der *vanishing gradients*, der verschwindenden Gradienten. Wenn wir unserem neuronalen Netz unter Verwendung einer Aktivierungsfunktion immer mehr Schichten hinzufügen, tendiert der Gradient der Verlustfunktion gegen Null, was verhindert, dass das Gewicht seinen Wert ändert und somit das Netz schwer trainierbar macht. In der Backpropagation-Phase wird der Fehler berechnet und die Gradientenwerte werden bestimmt. Die Gradienten werden an die versteckten Schichten zurückgesendet und die Gewichte entsprechend aktualisiert. Dieser Prozess wird fortgesetzt, bis die Eingabeschicht erreicht ist. Der Gradient wird immer kleiner, je weiter er die Eingabeschicht erreicht. Daher werden die Gewichte der Ausgangsschichten entweder sehr langsam aktualisiert oder bleiben gleich. Mit anderen Worten,

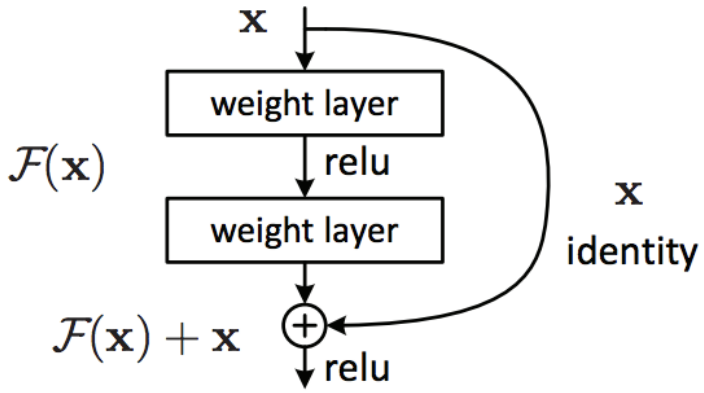

Abbildung 2: Residual Block mit Identity Mapping

die ersten Schichten des Netzes lernen nicht effektiv. Daher hat das tiefe neuronale Netz Schwierigkeiten, zu konvergieren, was die Genauigkeit des Modells beeinträchtigt und somit den Trainingsfehler erhöht. Dieses Problem wurde weitgehend durch normalisierte Initialisierung, d. h. Normalisierung der Anfangsgewichte der Netze und *Batch-Normalisierung*, gelöst.

ResNet

Residual Networks oder ResNet sind gleich aufgebaut wie die herkömmlichen tiefen neuronalen Netze. Der einzige Unterschied besteht darin, dass zusätzlich Identitäts-Abbildungen (identity mappings) zwischen den Schichten hinzugefügt werden.
Wie in 2 zu sehen, ist $F(x)$ das Residuum oder die Restabbildung, die sich zwischen zwei Faltungsschichten (Gewichtungsschichten) befindet und als Differenz zwischen der Eingabe x und der Ausgabe $H(x)$ des Residual-Blocks bezeichnet werden kann. Die Residualfunktion $F(x)$ kann also wie folgt geschrieben werden:

$$F(x) = H(x) - x$$

Das bedeutet, dass unser Hauptziel beim Training des tiefen Residualnetzwerks darin besteht, das Residuum $F(x)$ anstelle der Funktion $H(x)$ zu lernen, was die Gesamtgenauigkeit des Netzwerks erhöht.
Vorteile von ResNet:

(a) **Leicht zu trainieren**

(b) **Der Trainingsfehler nimmt mit zunehmender Tiefe des neuronalen Netzes nicht zu**, wie dies bei einfachen neuronalen Netzen der Fall ist, bei denen wir einfach Schichten übereinander legen.

(c) Das Hinzufügen der Identitätszuordnung führt **keine zusätzlichen Parameter** ein. Daher erhöht sich auch die Rechenkomplexität nicht.

(d) Die **Genauigkeitsgewinne sind mit zunehmender Tiefe höher**, so dass die Ergebnisse wesentlich besser sind als bei anderen Netzen wie dem VGG-Netz.

Diese Serie zeigt auch, wie Sie auf Colab grosse neuronale Netze mit der richtigen PyTorch-Version und PyTorch Lightning kostenlos trainieren können. Binden Sie Ihr Google Drive in Colab ein, damit Sie direkt auf Ihren Verzeichnissen arbeiten können.

Aufgabe 1. ResNet mit PyTorch Lightning

(a) Öffnen sie das Jupyter-Notebook `ANN06_resnet_TEMPLATE_pl.ipynb`, welches Sie auf Moodle finden und führen Sie die ersten Zellen aus. Sie haben zwei Möglichkeiten.

- Entweder Sie trainieren das Modell in einer Google Colab-Umgebung.
- Oder Sie führen das Notebook lokal auf Ihrem Rechner aus, falls eine passende PyTorch-Version und ausreichend Rechenleistung vorhanden ist.

(b) Visualisieren Sie das Modell mit `torchviz.make_dot`, um eine graphische Darstellung der Architektur zu erhalten.

(c) Trainieren Sie das ResNet-Modell mit PyTorch Lightning für mindestens 5 Epochen.

(d) Erstellen Sie eine Konfusionsmatrix der Testdaten und analysieren Sie, welche 4 Objekte am häufigsten verwechselt werden.

(e) Drucken Sie einen `classification_report` der Testdaten aus und interpretieren Sie die Ergebnisse bezüglich Präzision, Recall und F1-Score.

(f) Stellen Sie die ersten 25 Testbilder dar. Beachten Sie, dass die Bilder vorher normalisiert wurden, und wenden Sie eine Umkehrung der Normalisierung an, um sie korrekt darzustellen.

(g) Visualisieren Sie die Softmax-Ausgaben für die ersten 6 Testbilder als Balkendiagramm. Verwenden Sie eine logarithmische Skalierung für die y-Achse und beschriften Sie die x-Achse mit den realen Labels.

Aufgabe 2. Weitere Aufgaben (optional)
Dies sind ein paar Vorschläge für Übungen, die Ihnen helfen können, Ihre Fähigkeiten mit PyTorch Lightning zu verbessern. Es ist wichtig, praktische Erfahrungen mit PyTorch Lightning zu machen, um zu lernen, wie man es richtig benutzt.

(a) Trainieren Sie für mehr Epochen. Verbessert es die Klassifizierungsgenauigkeit?

(b) Ändern Sie die *Aktivierungsfunktion* für einige der Schichten in eine Sigmoid-Funktion.

(c) Können Sie eine einfache Möglichkeit finden, die Aktivierungsfunktion für alle Schichten zu ändern?

(d) Zeichnen Sie die Ausgabe der Max-Pooling-Schichten anstelle der Conv-Schichten.

(e) Ersetzen Sie die 2x2 Max-Pooling-Schichten durch `stride=2` in den Faltungsschichten. Ist eine Unterschied in der Klassifizierungsgenauigkeit erkennbar? Was passiert, wenn Sie das Ergebnis immer wieder optimieren? Der Unterschied ist zufällig, wie würden Sie also messen, ob es wirklich einen Unterschied gibt?

(f) Was sind die Vor- und Nachteile der Verwendung von Max-Pooling gegenüber Stride in der Faltungsschicht?

(g) Ändern Sie die Parameter für die Schichten, z. B. den Kernel, die Tiefe, die Grösse usw. Wie gross ist der Unterschied im Zeitaufwand und in der Klassifizierungsgenauigkeit?

(h) Fügen sie einige Faltungsschichten und vollständig verbundene Schichten hinzu oder entfernen Sie diese.

(i) Welches ist das einfachste Netz, das Sie entwerfen können und das immer noch gute Leistungen erbringt?

(j) Ändern Sie das Funktionsmodell so, dass es eine weitere Faltungsschicht enthält, die parallel zu den vorhandenen Faltungsschichten geschaltet ist, bevor sie in die dichten Schichten übergeht.

(k) Ändern Sie das funktionale Modell so, dass es die vorhergesagte Klasse sowohl als One-Hot-kodiertes Array als auch als Ganzzahl ausgibt, so dass wir anschliessend nicht `torch.argmax()` verwenden müssen.

(l) Erklären Sie einem Freund, wie ein CNN funktioniert.

Lösungen

Lösung 1.
Sie finden die Lösung auf Moodle unter `CNN_MNIST_SOLUTION_pl.ipynb`.

Lösung 2.