

Maschinelles Sehen

Pytorch, SVMs, Neural Networks und CNNs (15 Punkte)

SoSe 2025

Ziel dieses Übungsblattes ist es sich mit zwei Methoden der Bildklassifikation auseinanderzusetzen.

Aufgabe 1: Support Vector Machines (5 Punkt)

Eine Support Vector Machine ist ein *supervised* Lernalgorithmus und dient häufig als Klassifikator zur Unterteilung von Objekten in unterschiedliche Klassen. Wie in der Vorlesung besprochen, kann das auch sehr effektiv für hochdimensionale Daten eingesetzt werden. D.h. in unserem Fall sind die hochdimensionalen Daten die Deskriptoren der Bilder (die Sammlung der Bildfeatures). Die SVM muss zunächst auf dem mitgelieferten Bilddatensatz trainiert werden. Dafür müssen Sie ähnlich wie in Hausaufgabe 2.3 Bildfeatures extrahieren und diese dann als Zeilenvektoren untereinander in eine Trainingsmatrix (X_{train}) schreiben (siehe Abbildung 1). Zu jedem Zeileneintrag in X_{train} gibt es dann auch einen korrespondierenden Zeileneintrag im Labelvektor y_{train} . Auf einigen Testbildern, sollen Sie dann testen, ob das funktioniert. Den Beispieldatensatz finden Sie im .zip.

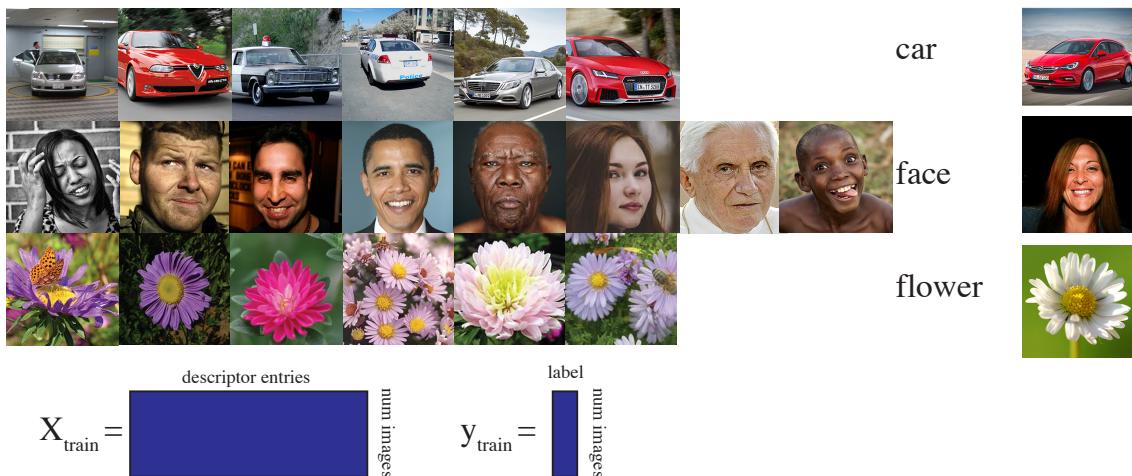


Abbildung 1: (Links) Trainingsbilder der drei Kategorien. (Rechts) Testbilder die nach dem Training in die korrekten Kategorien eingeordnet werden sollten.

Folgen Sie für die Implementierung den Hinweisen und Anweisungen im Source Code. In diesem Fall handelt es sich um ein sehr sehr kleines Beispiel. Sie können auch gern eine eigene Datenbank aufbauen bzw. erweitern, müssen dann aber wahrscheinlich die Parameter der SVM, Anzahl der Keypoints bzw. Größe der Features anpassen. Als Ergebnis sollten alle Trainingsbilder korrekt klassifiziert werden können.

Aufgabe 2: Neuronale Netze (5 Punkt)

Für diese Aufgabe müssen Sie ein zweischichtiges Neuronales Netz von Grund auf neu implementieren. Sie können das mitgelieferte Jupyter-Notebook als Basis nehmen und erweitern. Die Datei *two-layer-nn.py* gibt Ihnen einige detailliertere Anweisungen zu Eingabe, Architektur, Aktivierungsfunktionen und Trainingsparametern. Für das Training des Netzwerks müssen Sie zwei verschiedene Verlustfunktionen über die einfache Bilddatenbank aus Aufgabe 2.1 (Bildabruf) vergleichen:

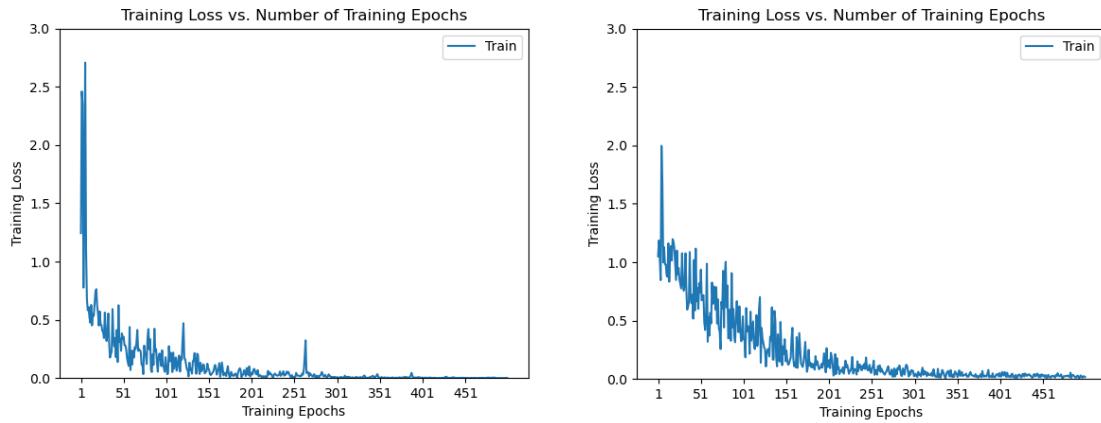


Abbildung 2: Plots von zwei Kostenfunktionen eines Two-layer Neural Networks. (Links) MSE Loss (Rechts) Cross-entropy loss.

- a) Mean-squared error loss Funktion
- b) Cross-entropy loss Funktion

Daraus ergeben sich die folgenden Trainingsplots und Testergebnisse. **Anmerkung:** Ich bin mir der Tatsache bewusst, dass dies kein realistischer Anwendungsfall ist und sich Ihre Trainingsdaten überanpassen (Overfitting). Er ist nur für diese Hausaufgabe gedacht.

Bitte geben Sie Ihren Code ab, so dass ich den Code ausführen und die Darstellungen und Testergebnisse wie angegeben ohne Änderungen meinerseits betrachten kann.

Aufgabe 3: Pytorch / Deep Learning / Convolutional Neural Networks (5 Punkt)

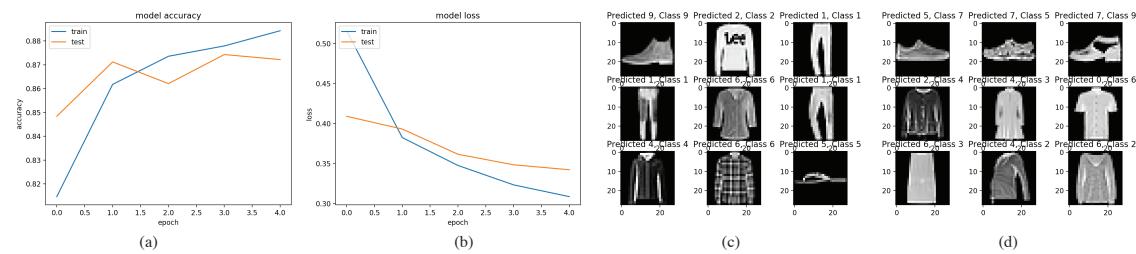


Abbildung 3: (a) Genauigkeit der Klassifikationsergebnisse mit steigender Anzahl von Epochen. (b) Fehler der Klassifikationsergebnisse mit steigender Anzahl von Epochen. (c) Beispiele richtiger Klassifikation. (d) Beispiele falscher Klassifikation.

Ein weiterer Ansatz zur Bildklassifikation basiert auf der besprochenen erweiterten Art von tiefen Neuronalen Netzen, den sog. Convolutional Neural Networks (CNN). D.h. Featurevorgaben wie SIFT o.ä. gibt es hier nicht, sondern diese werden gelernt.

Für diese Aufgabe müssen Sie sich mit Pytorch (<https://pytorch.org/tutorials/>) vertraut machen. Die notwendigen Pakete dafür sollten Sie in der 0. Übung schon installiert haben.

Für diese Aufgabe greifen wir auf die sogenannten *Fashion MNIST* Datensatz von Zalando zurück (<https://github.com/zalandoresearch/fashion-mnist#get-the-data>). Dieser besteht aus 60k Trainingsbildern und 10k Testbildern aus 10 unterschiedlichen Kategorien. Die Bilder haben jeweils eine Größe von 28x28px. Ihre Aufgabe ist es drei unterschiedliche Neuronale Netzwerkarchitekturen zu erstellen, die in ihrer Genauigkeit mindestens über 85% Klassifikationsgenauigkeit kommen.

Implementieren Sie dazu bitte verschiedene Klassen des Neuronalen Netzwerkmodells in *cnn.py* entsprechend der Standardarchitektur in Abbildung 4 und erstellen Sie mindestens zwei zusätzliche Modelle (neuronale Netze) mit einer anderen Architektur, die von den in der Vorlesung besprochenen

Architekturen (z.B. LeNet, AlexNet, VGG-16) inspiriert sind. **Anmerkung:** Das Training von CNNs kann sehr lange dauern. Beginnen Sie daher mit dem Training des Netzes mit nur wenigen Epochen, um abschätzen zu können, ob das Training einwandfrei funktionieren wird. Später sollten sie zwischen 10-50 Epochen trainieren. **Anmerkung:** Bitte verwenden Sie die *Cross-Entropie* Kostenfunktion für das Modell (softmax).

Sie können die Standardimplementierung nach Belieben ändern, wenn Ihnen das leichter fällt. Ein guter Ausgangspunkt ist https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html#model-training-and-validation-code, das ich für den mitgelieferten Quellcode verwendet habe.

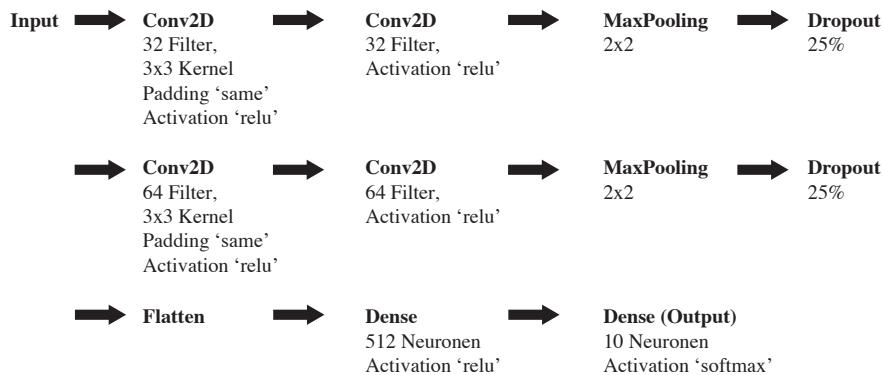


Abbildung 4: Zu implementierende Architektur eines Convolutional Neural Networks. **Hinweis: Um das korrekte Padding zu implementieren, nutzen Sie bitte diesen Hinweis:** <https://discuss.pytorch.org/t/converting-tensorflow-model-to-pytorch-issue-with-padding/84224>, ggf. müssen Sie auch die Bildgrößen anpassen.

Bitte geben Sie ihre Ergebnisplots und Ausgaben mit ab, sonst werde ich Punkte abziehen.

Abgabe: Die Abgabe soll via Moodle zum angegebenen Zeitpunkt erfolgen. Geben Sie bitte jeweils nur eine einzige .zip-Datei mit den Quellen Ihrer Lösung ab. Verspätete Abgaben werden mit einem Abschlag von 3 Punkten je angefangener Woche Verspätung belegt. Bitte fügen Sie alle notwendigen Bilder ein, sodass jede Aufgabe direkt ausführbar ist. Die Bilddatenbank brauchen Sie nicht mit abzugeben. Fügen Sie zusätzlich .png-Dateien für Ihre erzeugten Ergebnisplots bei. Verspätete Abgaben werden mit einem Abschlag von 3 Punkten je angefangener Woche Verspätung belegt.