Concolutional Neural Network for classification of handwritten digits

Zusammenfassung—In diesem Projekt haben wir ein *convolutional neural network* zur Klassifikation von handgeschriebenen Ziffern implementiert. Wir haben zwei *convolutional* Schichten, gefolgt von zwei *fully-connected* Schichten in unserer Architektur umgesetzt. Nach jeder Faltung addieren wir ein *bias* und wenden dann als Aktivierung die *ReLU* an. Dann wird jeweils *max-pooling* dahinter geschaltet um die Anzahl der Parameter zu reduzieren, ohne den Verlust der wesentlichen Merkmale. Auch bei den *fully-connected* Schichten kommt *ReLU* zum Einsatz und zum Schluss benutzen wir *softmax* um auf die zehn Klassen ein *scope* auszugeben. Trainiert und getestet wurde mit dem MNIST Datensatz, dabei diente TensorFlow als Framework zur Modellierung des Netzes der in Python implementiert wird. Wir haben uns auf einige wesentliche Fragestellungen bezüglich der Architektur konzentriert, um am Ende der Analyse dieser Fragen, eine möglichst gute CNN Architektur zur Klassifikation von handgeschriebenen Ziffern zu erhalten.

Index Terms—Machine Learning, Deep Learning, CNN, Neural Network, MNIST, handwritten digits

1 Einleitung

SCHREIBE die Einleiung am Ende, wenn alles andere Festgelegt und niedergeschrieben ist...

Aufgaben:

1. Implementiere ein CNN. Erläutere die Wahl der Architektur, z.B. die Anzahl der *Convolution-* und *Pooling* Schichten, die Größe der *fully-connected* Schichten, die Aktivierungsfunktion usw.
2. Trainiere den CNN. Erkläre alle implementierten Schritte, mögliche Performance Messungen und die benutzen Trainingsalgorithmen.
3. Teste dein CNN und erkläre die Ergebnisse der Auswertung.
4. Bekommt man die gleichen Resultate, wenn man das CNN mehrmals mit den gleichen Parametern ausführt? Was sind die Quellen der Zufälligkeit?
5. Führe mehr Optimierungs-Iterationen aus. Werden die ERgebnisse dadurch besser? Wie lange dauert die berechnung (das Training)?
6. Welchen Effekt hat eine Änderung der Lernrate für den Optimierer?
7. Wie lernt das CNN? Kannst du bspw. einige gelernten *features* aus der faltenden Schicht visualisieren?
8. Ändere die Konfiguration der Schichten, bspw. Anzahl der *Conv.* Filter, Größe der Filter, das *pooling* Fenster und die Anzahl der Neuronen in der voll verbundenen Schicht. Welche Auswirkung hat das auf die Performance?
9. Was ist die kleinst mögliche Konfiguration, die noch gute Ergebnisse liefert?
10. Versuche es ohne *Pooling*. Ändert sich die Klassifikationsgenauigkeit oder die Trainingszeit?

- Was kann man überhaupt Einstellen, welche Variablen gibt es.
- Was haben wir zusätzlich eingebaut?
- Was bedeutet pooling und convolution, hier genau erklären was da passiert.
- Dropout bringt was?

2 Die Architektur

3 Das Training

Wir haben die gelernten Gewichte der beiden faltenden Schichten durch umformen als zweidimensionale Grauwertbilder visualisiert. Man erknnt lediglich ein Rauschen, ähnlich wie man es von einer zufälligen Besetzung erwarten würde. Das zweite Bild hat eine höhere Auflösung, da die Eingabe für den Filter nach der ersten faltenden Schicht eine deutlich höhere Dimension hat, als der eindimensionale Kanal für die Graustufenwerte des ursprünglichen Bildes.

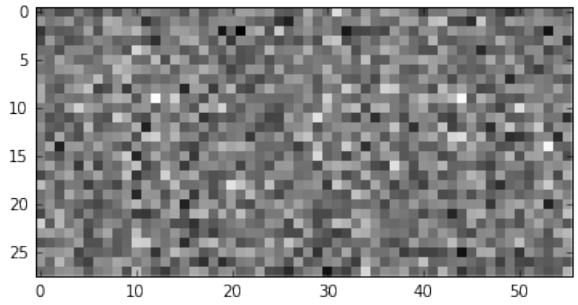

Abbildung 1. Visualisierung der gelernten Gewichter der ersten faltenden Schicht.

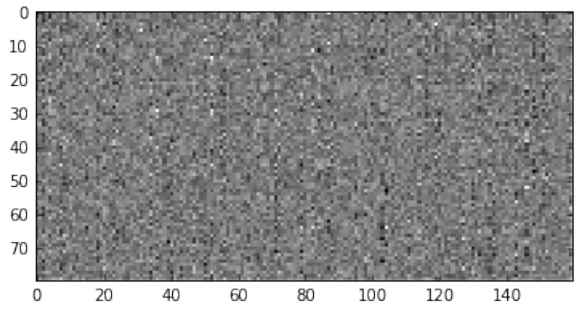

Abbildung 2. Visualisierung der gelernten Gewichter der ersten faltenden Schicht.

Dass das Netz tatsächlich inrgendwelche features erlernt hat, ist anhand der letzten beiden Abbildungen zu sehen. Faltet man eine homogene Bildfläche mit den gelernten Filtern, so entstehen linienartige Muster. Natürlich kann man keinen vierdimensionalen Tensor ohne manipulation darstellen, sodass wir diese auf zwei dimensionen umgeformt haben.

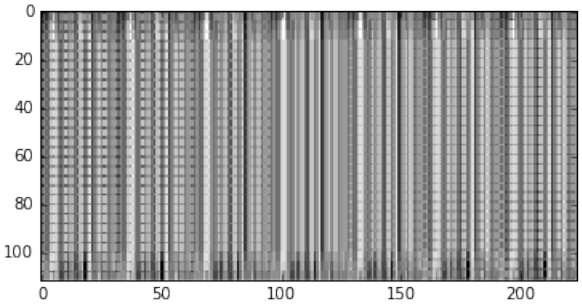

Abbildung 3. Ausgabe der ersten faltenden Schicht für eine homogene Bildfläche.

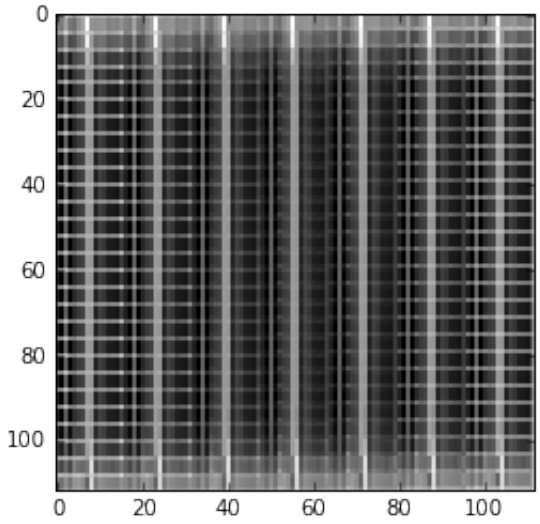

Abbildung 4. Ausgabe der zweiten faltenden Schicht.

4 Test und Auswertung

Listing 1

ANHANG A
CODE-SCHNIPSEL

```
def weight_variable(shape):
    """Creates a tf Variable of shape 'shape' with
    random elements"""
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)
```

Listing 1. Erzeugt ein Gewichts-Array mit zufällig initialisierten Werten. Die Dimensionen werden als Parameter entegengenommen.

```
def bias_variable(shape):
    """Create a small bias of shape 'shape' with
    contants values of 0.1"""
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)
```

Listing 2. Erzeugt ein bias-Array mit zufällig initialisierten Werten. Die Dimensionen werden als Parameter entegengenommen.

```
def conv2d(x, W):
    """Use a conv layer with weights W on zero
    padded input x and stride 1"""
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1],
    padding='SAME')
```

Listing 3. Eine zweidimensionale Faltung mit Eingabe x und Filter W.

```
def max_pool_2x2(x):
    """Create a 2x2 pooling layer with strides 2,
    reducing the resolution"""
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
    strides=[1, 2, 2, 1], padding='SAME')
```

Listing 4. Hier wird 2x2 max pooling auf die Eingabe x angewendet.

LITERATUR

[1] TODO: Namen einfügen, *Vorlesung: Machine Learning II - Deep Learning*, WS 2016/17