Aufgabe 15 (Faltung mittels Fast-Fourier-Transformation)
Die *zirkuläre diskrete Faltung* $*$ zweier Vektoren $x, y \in \mathbb{C}^n$ liefert wieder einen Vektor aus $\mathbb{C}^n$ und ist definiert als

$$(x * y)_j = \sum_{\ell=0}^{n-1} x_\ell \, y_{(j-\ell) \bmod n} \quad \text{für} \quad j = 0, 1, \ldots, n-1. \tag{1}$$

Wir verwenden zudem folgende Konvention für die diskrete Fourier-Transformation:

$$\mathcal{F} : \mathbb{C}^n \to \mathbb{C}^n, \quad \mathcal{F}(x) = z \quad \text{mit} \quad z_k = \sum_{j=0}^{n-1} \mathrm{e}^{-2\pi \mathrm{i} jk/n} \, x_j \quad \text{für} \quad k = 0, 1, \ldots, n-1,$$

mit entsprechender inversen Fourier-Transformation

$$\mathcal{F}^{-1} : \mathbb{C}^n \to \mathbb{C}^n, \quad \mathcal{F}^{-1}(z) = x \quad \text{mit} \quad x_j = \frac{1}{n} \sum_{k=0}^{n-1} \mathrm{e}^{2\pi \mathrm{i} jk/n} \, z_k \quad \text{für} \quad j = 0, 1, \ldots, n-1.$$

(a) Rechnen Sie das Faltungstheorem nach: für beliebige $x, y \in \mathbb{C}^n$ gilt

$$x * y = \mathcal{F}^{-1}\big(\mathcal{F}(x) \cdot \mathcal{F}(y)\big), \tag{2}$$

wobei $\cdot$ die punktweise Multiplikation bezeichnet. [3 Punkte]

Hinweis: Sie können ohne Beweis die Orthogonalität der Fourier-Vektoren verwenden, d.h.

$$\frac{1}{n} \sum_{k=0}^{n-1} \mathrm{e}^{2\pi \mathrm{i}(j-\ell)k/n} = \delta_{(j-\ell) \bmod n, 0} \quad \forall j, \ell = 0, 1, \ldots, n-1.$$

(b) Schreiben Sie jeweils eine Python-Funktion, die die Faltung zweier Vektoren via Gl. (1) bzw. Gl. (2) mittels FFT berechnet. Überprüfen Sie anhand eines kleinen Beispiels, dass beide Funktionen (bis auf numerische Rundungsfehler) auch tatsächlich dasselbe Ergebnis liefern. Was ist die asymptotische Laufzeit der Funktionen? [2 Punkte]

Hinweis: Die Fourier-Transformationen lassen sich mit `numpy.fft.fft` bzw. `numpy.fft.ifft` berechnen. Die asymptotische Laufzeit der FFT beträgt $\mathcal{O}(n \log(n))$.

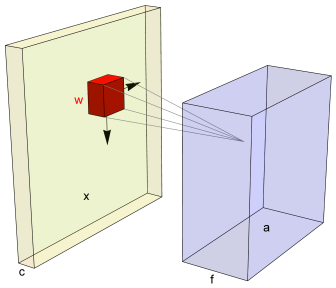
(c) Eng verwandt mit der Faltung ist die *Kreuzkorrelation*

$$(x \star y)_j = \sum_{\ell=0}^{n-1} \overline{x_\ell} \, y_{(j+\ell) \bmod n} \quad \text{für} \quad j = 0, 1, \ldots, n-1,$$

also $j + \ell$ anstatt $j - \ell$ in Gl. (1), wobei $\overline{x_\ell}$ die komplexe Konjugation von x_ℓ bezeichnet. Wie muss der Vektor x modifiziert werden, um wieder auf die Form (1) zu kommen, d.h. für welchen Vektor x' gilt $x \star y = x' * y$? [1 Punkt]

Aufgabe 16 (Implementierung eines Convolutional Neural Networks)
Im Allgemeinen berechnet eine Convolutional Layer die Operation

$$a_{f,j} = \sigma\big(z_{f,j}\big), \quad z_{f,j} = \sum_{c,k} w_{f,c,k}\, x_{c,j+k} + b_f \tag{3}$$



für Eingabe x, wobei $j \in \mathbb{Z}^2$ und $k \in \mathbb{Z}^2$ räumliche (Pixel-)Indizes bezeichnen, c den "Channel" der Eingabe (z.B. den Farbkanal) und f das jeweilige "Feature" der Ausgabe. Wir erlauben zusätzlich ein "zero-padding" der Eingabe x bezüglich der räumlichen (Pixel-) Dimensionen, d.h. eine Umrandung mit Nullen, so dass z.B. für `pad=2` und einen 5×5 Gewichts-Filter die Höhe und Breite der Ausgabe gleich der Eingabe ist. Die "stride length" gibt an, um wie viele Pixel der Faltungskern jeweils nach rechts bzw. unten verschoben wird; typisch ist `stride=1`.

In `conv_net.zip` auf der Webseite zur Vorlesung[1] finden sich Python-Programme für eine modulare Implementierung neuronaler Netze[2], wobei `layers.py` die Feedforward- und Backpropagation-Funktionen für verschiedene Schicht-Typen enthält. Aus Effizienzgründen wird stets ein "mini-batch" aus N Ein- und Ausgabevektoren auf einmal abgearbeitet; diese werden in Tensoren mit erster Dimension N zusammengefasst. Zur flexiblen Kombination verschiedener Typen von Aktivierungsfunktionen (Sigmoid, ReLU, ...) wird σ in der obigen Gleichung (3) als eigene Schicht behandelt.

(a) Implementieren Sie die Funktionen `conv_forward` und `conv_backpropagate` in `layers.py`, und verifizieren Sie Ihre Lösung mittels `python test_layers.py`. [3 Punkte]

(b) In `cnn_basic.py` findet sich eine Vorlage für ein einfaches "convolutional neural network". Vervollständigen Sie den Code und verifizieren Sie Ihre Lösung mittels `python test_cnn_basic.py`. [2 Punkte]

(c) Trainieren Sie nun das Netzwerk anhand eines kleinen Ausschnitts des MNIST Datensatzes (bestehend aus nur 40 Ziffern), indem Sie `run_training.py` laufen lassen. Woran lässt sich Overfitting erkennen? [1 Punkt]
In zukünftigen Aufgaben soll eine effizientere Implementierung der Convolutional Layer auch das Trainieren anhand größerer Datensätze ermöglichen.

Tutoraufgabe 8 (Convolutional Layer zur Bildbearbeitung)
Durch geeignete Wahl der Gewichte $w_{f,c,k}$ und Bias-Werte b_f in Gl. (3) können Convolutional Layers als Bildbearbeitungs-Werkzeuge fungieren, wie im Folgenden demonstriert werden soll. Der Einfachheit halber sei σ stets die Identitätsfunktion, also $a_{f,j} = z_{f,j}$. Die Anzahl der Features betrage 3 (interpretiert als Farbkanäle) oder 1 (interpretiert als Graustufen-Wert).

(a) Erstellen Sie einen *Graustufen-Konvertierer*. Welche Dimension besitzt der Gewichtstensor w?

(b) Implementieren Sie einen 5×5 Gauß-Unschärfe-Filter durch Wahl von

$$w_{f,c,k} = \delta_{f,c} \frac{\mathrm{e}^{-\frac{1}{4}(k_x^2+k_y^2)}}{\sum_{k'} \mathrm{e}^{-\frac{1}{4}(k_x'^2+k_y'^2)}} \quad \text{für} \quad |k_x| \le 2, \quad |k_y| \le 2.$$

(c) Ein gebräuchlicher Kantendetektions-Filter ist der *Sobel-Operator*, der (pro Farbkanal) durch Faltung mit

$$\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \quad \text{bzw.} \quad \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

die horizontale bzw. vertikale "Ableitung" der Helligkeitswerte approximiert. Implementieren Sie den Sobel-Operator. Wie lassen sich die beiden Ableitungen für die jeweiligen Farbkanäle zu einem einzelnen Wert kombinieren?

[1] https://tu-dresden.de/mn/math/wir/mendl/studium/courses/mosim_2019_sose
[2] basierend auf http://cs231n.github.io/assignments2018/assignment2