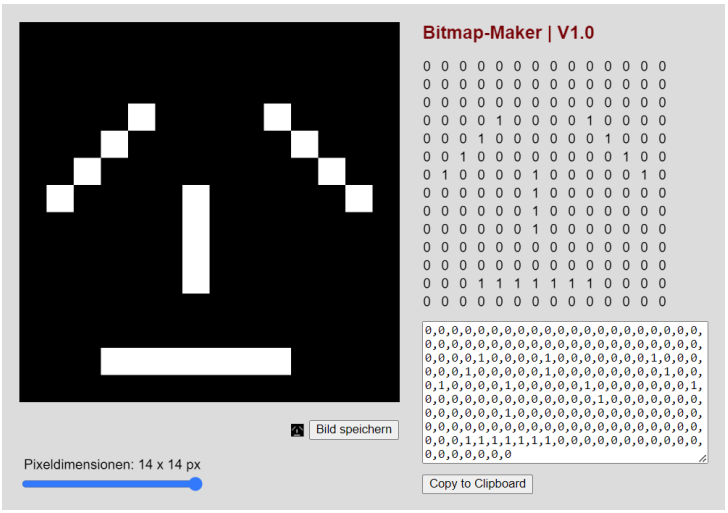
Unterrichts-Idee für ‚Einführung in Bitmaps und Conv2D-Layer‘

1. Mustervorlage entwerfen

https://iludis.de/bitmapMaker/index.html

a. Erstelle mit der Bitmap-App ein Bild, in dem solche Muster vorkommen, die man erkennen möchte. Das könnte zum Beispiel so aussehen:



b. Mit dem Button „Copy to Clipboard” kann das 1-Bit Bitmap-Muster in die Zwischenablage kopiert werden.
c. Man kann das Bitmap auch als PNG-Bild speichern – allerdings wird dann der Wert ‚1‘ in den Wert ‚255‘ um gewandelt, und außerdem erhält das Bild die üblichen drei RGB-Kanäle.

2. Bitmaps im Jupyter-Notebook anzeigen

https://iludis.de/bitmapMaker/Simple_Bitmaps.ipynb

Optional: Wenn man mit den ‚richtigen Tools‘ etwas herumspielen will, kann man das 1D-Bitmaparray in eine Python-Umgebung einfügen. Dazu musst du in der Befehls-Zeile

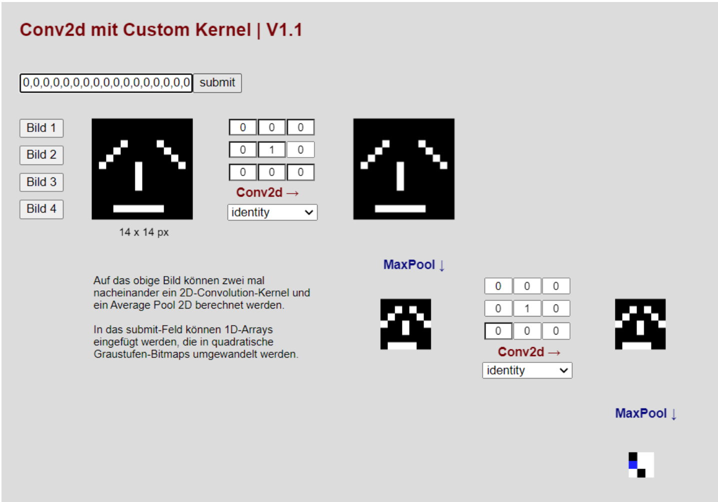
meinArray = np.array([***1,0,1,0,1,0,1,0,1***])

… die roten Zahlen gegen deine kopierten Zahlen ersetzen. Dann erzeugt das Skript daraus wieder ein Bild:

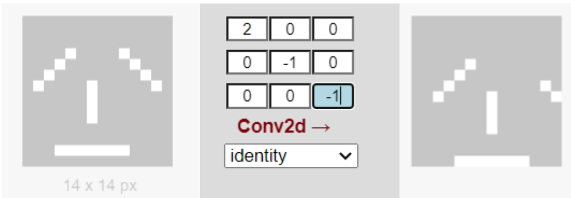
3. Convolution-Kernel entwerfen

https://iludis.de/konvolution/index.html

a) Füge das kopierte 1D-Array mit dem serialisierten Bitmap-Muster in das „submit“-Feld ein und drücke den Knopf. Die App wird das 1D-Array wieder quadratische umformen („reshape“) und als Bild darstellen



b) Nun kannst du mit den Conv2D-Kernels ‚herumspielen‘: Wähle entweder im Dropdown unterschiedliche Standard-Kernel oder gib deine Kernel manuell ein:



c) Probiere folgende Kernel aus:

$$\begin{pmatrix} 0 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 0 \end{pmatrix} \qquad \begin{pmatrix} -1 & -1 & 0 \\ -1 & 1 & -1 \\ 0 & -1 & -1 \end{pmatrix} \qquad \begin{pmatrix} -1 & 0 & -1 \\ -1 & 1 & -1 \\ -1 & 0 & -1 \end{pmatrix}$$

4. Mit ‚richtigen‘ Grayscale-Bildern im MNIST-Format ausprobieren:

https://iludis.de/GrayscaleMaker/index.html

Male mit der Maus (oder dem Finger) ein Bild und füge es per Copy-Paste wieder in die Conv2D-App ein. Funktionieren deine Kernels noch?