Aufgabe 17 (Optimierte Implementierung eines Convolutional Neural Networks)
In `conv_net_opt.zip` findet sich eine leicht verbesserte modulare Implementierung neuronaler Netze[1], die im Vergleich zu Aufgabe 16 eine effizientere Umsetzung der Convolutional und Max-Pool Layers enthält.[2]

(a) Trainieren Sie das `BasicConvNet` mittels `run_training_mnist.py`. Das Training sollte im Vergleich zu A16 (c) nun viel weniger Zeit benötigen. Welche maximale Klassifizierungsgenauigkeit beim Testdatensatz können Sie erreichen, wenn Sie 100 (anstatt 20) Epochen lang trainieren? Wiederholen Sie nun das Training mit allen verfügbaren 50000 Ziffern (wobei eine Epoche genügt). Auf welche Klassifizierungsgenauigkeit kommen Sie jetzt? [2 Punkte]

Hinweis: Beachten Sie, dass der Trainingsdatensatz in dieser Teilaufgabe zur direkten Vergleichbarkeit mit A16 (c) zunächst auf insgesamt nur 40 Ziffern eingeschränkt ist!

Sie können das trainierte Netzwerk natürlich auch nochmals an Ihren eigenen handschriftlichen Ziffern ausprobieren.

(b) Den CIFAR-10 Datensatz[3] bestehend aus 60000 32×32 Farbbildern kennen Sie bereits aus A11. Laden Sie den Datensatz (nochmals) herunter (siehe Beschreibung in `get_cifar10.md`), und lassen Sie anschließend `run_training_cifar.py` laufen. Visualisieren Sie nun die Filter, d.h. die Gewichte $w_{f,c,k}$ der Convolutional Layer des trainieren Netzwerks, mit Hilfe der Funktion `visualize_conv_weights` in `visualization.py`. Worin spiegeln sich die Dimensionen von w (Anzahl Features, Anzahl Channels, Höhe und Breite) im Ausgabebild wider? [2 Punkte]

Hinweis: Die Klassifizierungsgenauigkeit beim Testdatensatz sollte knapp über 40% liegen.

(c) Realisieren Sie nun ein Convolutional Network (etwa mit Architektur ähnlich zu `BasicConvNet`) mittels der TensorFlow Keras "Sequential API" (d.h. unabhängig vom bisherigen Code), und trainieren Sie das Netzwerk anhand CIFAR-10. In der Vorlage `tfkeras_conv_cifar.py` wird der CIFAR-10 Datensatz im (für TensorFlow) geeigneten Format geladen. [2 Punkte]

Hinweis: Siehe Abschnitt "Alternative high-level Keras Sequential API" im Jupyter-Notebook `tensorflow_intro.ipynb` auf der Vorlesungswebsite als Orientierung. Zwischen einer Convolutional und Dense Layer muss eine zusätzliche Schicht `tf.keras.layers.Flatten()` eingefügt werden, um die drei Dimensionen "Höhe, Breite und Channel" zu einer einzigen Dimension zusammenzufassen.

Aufgabe 18 (Grundlagen von TensorFlow und Artistic Style Transfer)

(a) Vervollständigen Sie folgendes Python-Programm, um die Ableitung der Funktion $x \mapsto 2x\sin(\pi x)$ an der Stelle $x = 3/2$ mittels TensorFlow auszurechnen, und vergleichen Sie das Ergebnis mit dem analytischen Referenzwert -2. [2 Punkte]

```
import numpy as np
import tensorflow as tf

x = tf.placeholder(tf.float64)
y = ...
dydx = ...
with tf.Session() as sess:
    y_eval, dydx_eval = sess.run((y, dydx), feed_dict=...)
    print('y_eval:', y_eval)
    print('dydx_eval:', dydx_eval)
```

Hinweis: Siehe wiederum das Jupyter-Notebook `tensorflow_intro.ipynb` auf der Vorlesungswebsite als mögliche Orientierung.

[1] basierend auf http://cs231n.github.io/assignments2018/assignment2
[2] Die `im2col`-Operation verwendet einen Trick basierend auf den "strides" eines NumPy-Arrays.
[3] https://www.cs.toronto.edu/~kriz/cifar.html

Der Artistic Style Transfer-Algorithmus[4] verwendet sogenannte Gram-Matrizen zur Repräsentation des Zeichenstils. Die Gram-Matrix der ℓ-ten Schicht ist (basierend auf der Netzwerk-Aktivierung) definiert als

$$G^\ell_{ff'} = \sum_j a^\ell_{fj}\, a^\ell_{f'j}, \tag{1}$$

wobei f, f' Feature-Indizes bezeichnen und $j \in \mathbb{Z}^2$ den räumliche (Pixel-)Index. In Matrixschreibweise lässt sich G^ℓ also ausdrücken als $G^\ell = (a^\ell) \cdot (a^\ell)^T$.

Die "Kostenfunktion" $\mathcal{L}^\ell_{\text{style}}$ quantifiziert die Abweichung vom vorgegebenen Zeichenstil auf der ℓ-ten Schicht durch

$$\mathcal{L}^\ell_{\text{style}} = \frac{1}{(2N^\ell)^2} \left\| G^\ell - \mathsf{G}^\ell_{\text{style}} \right\|^2_F ,$$

wobei die symmetrische Matrix $\mathsf{G}^\ell_{\text{style}}$ fest vorgegeben ist, G^ℓ in Gl. (1) definiert ist, N^ℓ die Anzahl der Einträge von a^ℓ bezeichnet und $\|X\|_F = \sqrt{\mathrm{Tr}[XX^T]}$ die Frobenius-Norm einer Matrix X ist.

(b) Verifizieren Sie, dass

$$\frac{\partial \mathcal{L}^\ell_{\text{style}}}{\partial a^\ell_{fj}} = \frac{1}{(N^\ell)^2} \left(\left(G^\ell - \mathsf{G}^\ell_{\text{style}} \right) \cdot a^\ell \right)_{fj} .$$

[2 Punkte]

(c) In `style_transfer.zip` findet sich eine Python/TensorFlow Implementierung des Artistic Style Transfer-Algorithmus.[5] Vervollständigen Sie die fehlende Zeile innerhalb der Python-Funktion `gram_matrix` in `style_transfer.py` zur Berechnung der Gram-Matrix mittels TensorFlow. Lassen Sie nun als Demonstration `run_style_transfer.py` laufen. [2 Punkte]

Hinweis: In TensorFlow wird die Aktivierung a^ℓ_{fj} als Tensor der Dimension $1 \times H \times W \times F$ gespeichert, wobei die Höhe H und Breite W die räumlichen Dimensionen der Ausgabe der aktuellen Schicht angeben und F die Anzahl der Features, d.h. die Feature-Dimension kommt zum Schluss. Sie können Gl. (1) mittels `tf.tensordot` und geeignete Wahl des `axes`-Parameters auswerten; insbesondere sind keine for-Schleifen nötig.

Bemerkung: Oft findet sich in der Literatur auch die Bezeichnung "channels" anstatt "features", da ja die Ausgabe-Features der ℓ-ten Schicht als Eingabe-Channels der $(\ell+1)$-ten Schicht aufgefasst werden.

Tutoraufgabe 9 (Visualisierung von Klassifikations-Netzwerken)
Wir wollen ein bereits trainiertes Netzwerk zur Bildklassifizierung untersuchen; das Netzwerk ordnet ein Eingabebild einer Klasse (z.B. Katze, Haus, Auto, ...) zu, wobei die möglichen Klassen fest vorgegeben sind. Beim "ImageNet Large Scale Visual Recognition Challenge" gibt es beispielsweise 1000 Klassen. Die Idee besteht darin, zu einer vorgegebenen Klasse ein synthetisches Eingabebild zu finden, das das Netzwerk maximal "anspricht".[6] Konkret besitze das Netzwerk für jede Klasse y eine separate Ausgabezelle, die jeweils einen "Score" S_y liefere. (Die "Scores" werden typischerweise noch mittels einer Softmax-Schicht in eine Wahrscheinlichkeitsverteilung umgewandelt, was hier allerdings nicht benötigt wird.) Es sei nun eine Klasse y fest ausgewählt; generieren Sie ein synthetisches Bild $x \in \mathbb{R}^{H \times W \times C}$, das die Netzwerk-Ausgabe für diese Klasse maximiert:

$$x = \arg\max_{x'} S_y(x') - \lambda \|x'\|^2.$$

Der zweite Term dient zur Regularisierung. Implementieren Sie diese Idee für das "SqueezeNet"[7].

[4] siehe L. A. Gatys, A. S. Ecker, M. Bethge. Image style transfer using convolutional neural networks. CVPR (2016)
[5] basierend auf http://cs231n.github.io/assignments2018/assignment3
[6] siehe K. Simonyan, A. Vedaldi, A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. ICLR (2014)
[7] siehe F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, K. Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. arXiv:1602.07360