General constraints for code submissions Please adhere to these rules to make our and your life easier! We will deduct points if your solution does not fulfill the following:

- If not stated otherwise, we will use exclusively Python 3.5.
- If not stated otherwise, we expect a Python script, which we will invoke exactly as stated on the exercise sheet.
- Your solution exactly returns the required output (neither less nor more) – you can implement a `--verbose` option to increase the verbosity level for developing.
- Add comments and docstrings, so we can understand your solution.
- (If applicable) The `README` describes how to install requirements or provides addition information.
- (If applicable) Add required additional packages to `requirements.txt`. Explain in your `README` what this package does, why you use that package and provide a link to it's documentation or GitHub page.
- (If applicable) All prepared unittests have to pass.
- (If applicable) You can (and sometimes have to) reuse code from previous exercises.

Now that you have learned about different search spaces and optimization methods for neural architecture search (NAS), you will use these concepts to implement a specialized NAS method.

1. **Search Spaces** [2 points]

   In this exercise you will have to compute the search space size in a cell search space. Suppose we have a *normal* and *reduction* cell stacked to form the search model (see Slide 29 Lecture 7). As illustrated in Slide 30 of Lecture 7, each of these cells can be represented as a Directed Acyclic Graph (DAG), where the yellow-labeled nodes are the inputs from the previous and previous-previous cells, the blue-labeled nodes are the so-called intermediate nodes, where the output tensors of the incoming edges are added element-wise. The dashed edges are a mixture/combination of operations in the operation set $\mathcal{O}$. The solid lines going to the output node (red-labeled node) just denote that the feature maps out of the intermediate nodes are concatenated to form the cell output tensor, and do not perform any operation as the dashed edges.

   Decisions are made only on the dashed edges and intermediate nodes. On each of these edges we want to select one out of $|\mathcal{O}|$ possible operations and on each intermediate node the *top-K* inputs (see Slide 32 Lecture 7). Given $|\mathcal{O}| = 7$, $K = 2$ and 4 intermediate nodes in both normal and reduction cells, compute the total number of possible architectures in this search space.

2. **Differentiable Architecture Search** [5 points]

   In this second part of the exercise you will run DARTS for finding an optimal CNN architecture on MNIST. The search model is defined in `model_search.py`. It contains three stacked cells: reduction-normal-reduction. The architecture search problem is to find an optimal operation out of $\mathcal{O} = \{conv\_3x3, max\_pool\_3x3, avg\_pool\_3x3, Identity\}$ in each edge of these cells. The number of intermediate nodes is 2.

   (a) In order to create the architecture continuous relaxation, we need to define a *MixedOp*, which is a convex combination of the operations outputs connecting two nodes in the cells. It is defined as follows: [3pt.]

   $$x^{(j)} = \sum_{i<j} \tilde{o}^{(i,j)}(x^{(i)}) = \sum_{i<j} \sum_{o \in \mathcal{O}} \frac{e^{\alpha_o^{(i,j)}}}{\sum_{o' \in \mathcal{O}} e^{\alpha_{o'}^{(i,j)}}} o(x^{(i)})$$

   Based on this formulation you have to fill in the code in lines 46-57 of `model_search.py` in order to compute the output tensor $x^{(j)}$ of the MixedOp.

   (b) Having defined the search model, we now need to run the DARTS optimization loop (Algorithmi 1, Slide 31 of Lecture 7). We will use the first-order approximation. Your task is going to be only to write the lines of code that compute the architectural updates in lines 43-49 of `train_search.py`. Afterwards, you should be able to run `python train_search.py` without any errors. This will conduct the search for 5 epochs and write in a directory named `logs/` the output logs and a file [2pt.]

with the optimal architecture configuration.

In the end, to generate a visualization of the found cells run `python visualize.py`. This should generate two `.pdf` files named `normal.pdf` and `reduction.pdf`. Push the contents written in `logs/` together with the two `.pdf` files generated by the visualization script to your Bitbucket repository.

NOTE: Running `train_search.py` on a GPU machine takes a few minutes. This might scale to more than 1h when running on a CPU machine.

3. **Feedback** [Bonus: 0.5 points]

For each question in this assignment, state:

- How long you worked on it.
- What you learned.
- Anything you would improve in this question if you were teaching the course.

This assignment is due on 28.06.19 (10:00). Submit your solution for the tasks by uploading a PDF to your groups BitBucket repository. The PDF has to include the name of the submitter(s).