To get access to this week's code use the following link: `https://classroom.github.com/a/r_3a2ZOp`

General constraints for submissions: Please adhere to these rules to make our and your life easier! We will deduct points if you fail to do so.

- Your code should work with *Python 3.8*.
- You should only fill out the *TODO-gaps* and not change anything else in the code.
- Add comments to your code, to help us understand your solution.
- Your code should adhere to the PEP8 style guide. We allow line lengths of up to 120.
- While working on the exercise, push all commits to the `dev` branch (details in assignment 1). Only push your final results to the `main` branch, where they will be automatically tested in the cloud. If you push to `main` more than 3 times per exercise, we will deduct points.
- All provided unit tests have to pass: In your *GitHub* repository navigate to *Actions* → your last commit → Autograding → education/autograding to see which tests have passed. The points in autograding only show the number of tests passed and have nothing to do with the points you get for the exercise.
- `for` loops can be slow in Python, use vectorized `numpy` operations wherever possible (see assignment 1 for an example).
- Submit *a single PDF* named `submission.pdf`. Include your matriculation numbers on the top of the sheet. Add the answers and solution paths to all non-coding questions in the exercise. Do not leave answers to any questions as comments in the code. You can use Latex with the student template (provided in exercise 1 / ILIAS) or do it by hand.
- Please help us to improve the exercises by filling out and submitting the `feedback.md` file.
- We do not tolerate plagiarism. If you copy from other teams or the internet, you will get 0 points. Further action will be taken against repeat offenders!
- Passing the exercises ($\geq$ 50% in total) is a requirement for passing the course.

How to run the exercise and tests

- See the `setup.pdf` in exercise 1 / ILIAS for installation details.
- We always assume you run commands in the *root folder* of the exercise repository.
- If you use miniconda, do not forget to activate your environment with `conda activate mydlenv`
- Install the required packages with `pip install -r requirements.txt`
- Python files in the *root folder* of the repository contain the scripts to run the code.
- Python files in the `tests/` folder of the repository contain the tests that will be used to check your solution.
- Test everything at once with `python -m pytest`
- Run a single test with `python -m tests.test_something` (replace `something` with the test's name).
- To check your solution for the correct code style, run `pycodestyle --max-line-length 120 .`
- The scripts `runtests.sh` (Linux/Mac) or `runtests.bat` (Windows) can be used to run all the tests described above. If you are on Linux, you need execution rights to run `runtests.sh`.

This exercise focuses on CNN architectures. Using PyTorch, we will:

- Implement basic architectures.
- Experiment with architectures, hyperparameters and batch normalization.
- Design and implement a Convolutional network that improves classification accuracy.
- Experiment with transfer learning using a ResNet pre-trained on the ImageNet dataset.

1. Coding Tasks

We will be using the CIFAR-10 dataset for our experiments. It consists of 60000 color images in 10 classes, 50000 training images, and 10000 test images of size 32x32. It can be downloaded from here.

PyTorch provides a package called torchvision, which automatically downloads the CIFAR-10 dataset, preprocesses it, and iterates through it in minibatches.

To keep the training fast, we use only 10% of the training dataset. Naturally, this will mean that final accuracies of our models will not be very good.

To get an idea what the data looks like, you can run `python show_images.py`

1) [4 points] **Todo:** Define a Convolutional Neural Network (class `ConvNet1` in `lib/models.py`).

A simple Convolutional Network is a sequence of layers. The most used layers to build ConvNets are Convolutional Layers, Pooling Layers, and Fully-Connected Layers.

Your task is to stack these three types of layers to form a full ConvNet architecture following the pattern in CIFAR-10:

[INPUT - CONV - RELU - MAXPOOL - FC]

- INPUT [32x32x3]: the RGB raw pixel values of the image.
- CONV: convolutional layer with 3 filters and filter size 5 (use padding value 2 to preserve the input dimension).
- RELU: applies an elementwise ReLU activation function.
- MAXPOOL: downsampling operation with kernel size = 2.
- FC: fully-connected layer that computes the class scores (also called logits), with output size [1x1x10], where each of the 10 outputs corresponds to a class score. Each neuron in this layer will be connected to all the neurons in the previous layer.

Implement the `forward` function, remember that the `backward` pass is computed automatically in PyTorch.

Run `python -m tests.test_convnet1` to test your implementation.

Train and evaluate the model by running `python run_conv.py -x 1`. You should get around 32% accuracy.

2) [2 points] **Todo:** Change the parameters of the network - Number of filters (class `ConvNet2` in `lib/models.py`).

Use the previous ConvNet and change the number of filters used in the convolutional layer to 16.

Run `python -m tests.test_convnet2` to test your implementation.

Train and evaluate the model by running `python run_conv.py -x 2`.
Describe what happens when you change the number of filters. Do more or fewer perform better?

3) [2 points] **Todo:** Change the parameters of the network - Filter size (class `ConvNet3` in `lib/models.py`).

In `ConvNet2` we use 5x5 filters, now use 3x3 filters and change the padding value to preserve input dimension.

Run `python -m tests.test_convnet3` to test your implementation.

Train and evaluate the model by running `python run_conv.py -x 3`.
Describe what happens. What are benefits of using smaller filters?

4) [2 points] **Todo:** Batch normalization (class `ConvNet4` in `lib/models.py`).

Insert batch normalization after the convolution in `ConvNet3` and describe what happens. Is the network performing better? If so, why?

Run `python -m tests.test_convnet4` to test your implementation.

Train and evaluate the model by running `python run_conv.py -x 4`.

5) [4 points] **Todo:** Transfer learning (`lib/transfer_learning.py`)

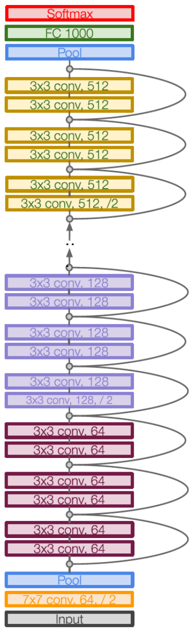
In this part of the exercise, we are going to load a `ResNet18` model, which has been pre-trained on the ImageNet dataset, and re-train it to perform classification on the CIFAR10 dataset.

We have specified five different ways to use the pretrained model for transfer to our dataset in the code comments. Implement the models by replacing or "freezing" the necessary layer(s). Frozen layers do not change during training.

Each implementation can be tested by running its respective testcase. For example, to test the first implementation, run `python -m pytest tests.test_modified_resnet1`.

Train and evaluate the models by running `python run_transfer_learning.py`.

Briefly discuss your observations.



Architecture of ResNet34. ResNet18 is similar, but has only 2 residual blocks before each downsampling. Check the ResNet paper for more details.

6) [2 points] You may have noticed while implementing the CNN models that the number of input features to the linear layer at the end depends on the output of the convolutional layer before it. E.g., a convolutional layer which produces an output of shape `(5, 3, 10, 10)` (in BCHW format) will result in the linear layer after it having `3*10*10` input features. Further, the height and width of this output (10 and 10, in this case) also depend on the size of the input image.

How is the ResNet model able to take images of variable sizes as inputs?

2. [2 bonus points] **Design your own ConvNet**

Todo: Taking into account the previous results, design your own ConvNet to achieve at least 55% accuracy in maximum 20 epochs (class `ConvNet5` in `lib/models.py`).

You can change the network architecture. The most common ConvNet architecture follows the pattern:
`INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC`
where the * indicates repetition, and the POOL? indicates an optional pooling layer. Moreover, $N \geq 0$ (and usually $N \leq 3$), $M \geq 0$, $K \geq 0$ (and usually $K < 3$)

Remember that deeper networks will take a lot more time to train.

Train and evaluate the model by running `python run_conv.py -x 5`.

Note: You are only allowed to play with the architecture and the hyperparameters of the layers for this exercise. Do not warm start with ResNet or any other models.

3. [1 bonus point] **Code Style**

On each exercise sheet, we will also be using `pycodestyle` to adhere to a common Python standard. `pycodestyle` checks your Python code against some common style conventions in PEP 8 and reports any deviations with specific error codes.

Your code will be automatically evaluated on submission (on push to `main`). You can run `pycodestyle --max-line-length=120 .` to manually evaluate your code before submission.

One bonus point will be awarded if there are no code style errors detected in your code by the `pycodestyle --max-line-length=120 .` command.

4. [1 bonus point] **Feedback**

Todo: Please give us feedback by filling out the `feedback.md` file.

- Major Problems?
- Helpful?
- Duration (hours)? For this, please follow the instructions in the `feedback.md` file.
- Other feedback?

This assignment is due on 25.12.2024 (23:59 CET). Submit your solution for the tasks by uploading (`git push`) the PDF, txt file(s) and your code to your group's repository. The PDF has to include the name of the submitter(s). Teams of at most 3 students are allowed.