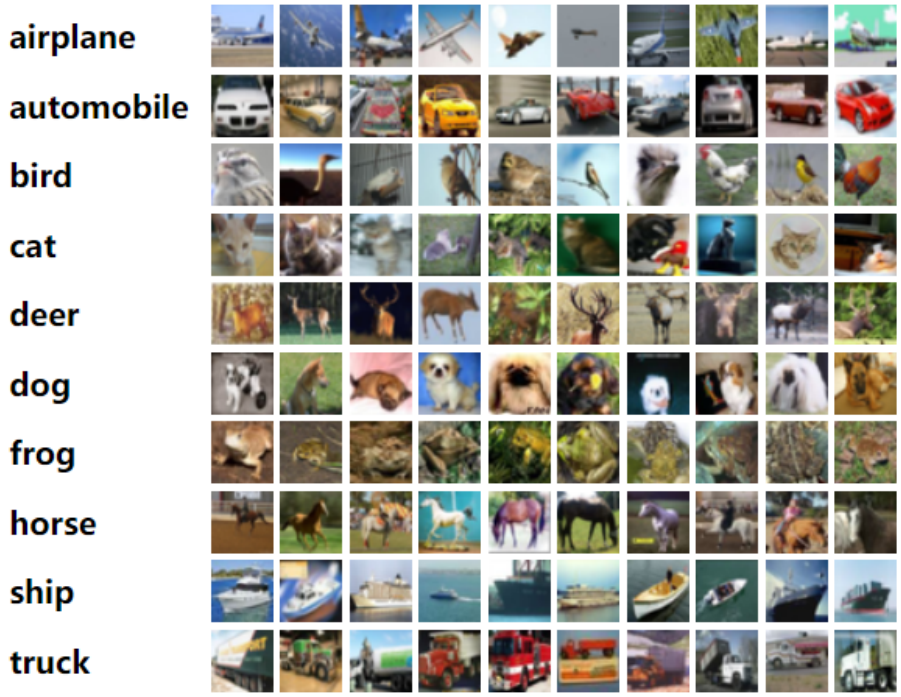
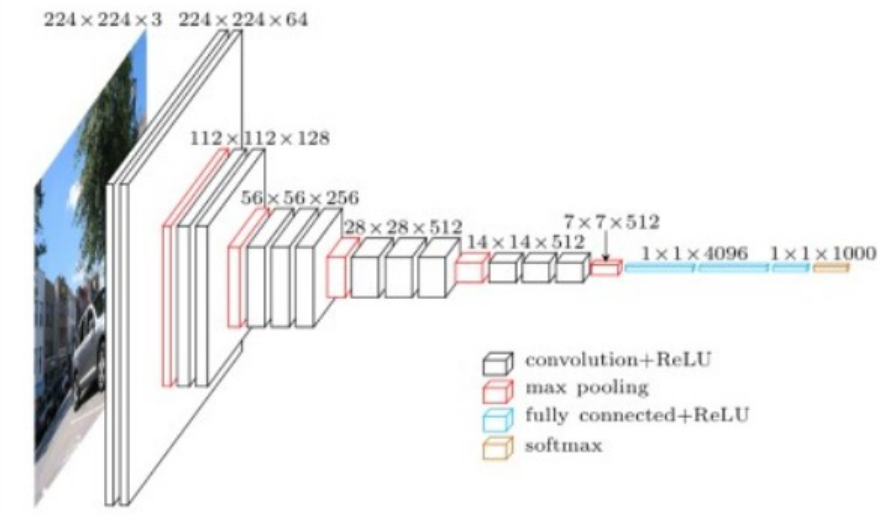
Exercise for MA-INF 2213 Computer Vision SS23
26.05.2023
Submission deadline: 14.06.2023
Neural Networks

In this assignment, you are required to train a neural network to solve a classical computer vision task: classification of CIFAR-10 images. This dataset contains 50000 training images and 10000 validation images of 10 classes, each having a resolution of 32×32. Here is what the images look like:



For this task, we will be training a VGG11 model. This is a relatively lightweight architecture, which should be well-suited for our task. Here is the scheme of the architecture:



As one can notice, the presented architecture is tuned for processing larger images (224×224). Nonetheless, it can be easily adapted to our smaller images of CIFAR-10, which are 7 times smaller in each dimension. This requires simple modifications:

- If we simply feed CIFAR-10 images to this network, all the intermediate features will become proportionally 7 times smaller in horizontal and vertical dimensions. Thus, the features before the last pooling layer will become 1×1×512 instead of 7×7×512. The need for this last pooling layer is, therefore, eliminated.
- As the task of classifying 32×32 images is simpler, we should accordingly balance the capacity of the model. You are suggested to make two modifications to make the model a bit simpler: (1) decrease the channel multiplier of each layer by a factor of 2. For example, all layers with 128 channels should have 64 channels, 256 should be 128, and so on. (2) make the network a bit shorter by using fewer layers. (3) drop the fully connected layer before the final classification layer. In the scheme above this corresponds to removing the layer with dimensions 1×1×4096.
- Finally, as we now have 10 classes, the final classification layer should map features to 10 activations.

You are provided with a code template *main.py* that you are required to fill with your code. The code uses PyTorch, which is the most popular library for implementing neural networks in Python. If you are not familiar with PyTorch, spend some time reading tutorials on its official website. It should be relatively easy to get started with the template and tutorials. The code already provides some helper functions like dataloading. You are required to program the neural network, define the loss function and optimization, and implement the training loop and evaluation. More specifically:

1. Define the VGG11 architecture in the __init__ method of the Our_Neural_Network class. The main body of the network should contain 8 convolutional blocks. The structure of each convolutional block is the following:

   - Conv2D(kernel_size=3, padding=1)
   - BatchNorm2d
   - ReLU

   The overall structure of the network should be the following:

   - ConvBlock1 (32 channels)
   - MaxPool(kernel_size=2, stride=2)
   - ConvBlock2 (64)
   - MaxPool(kernel_size=2, stride=2)
   - ConvBlock3 (128)
   - ConvBlock4 (128)
   - MaxPool(kernel_size=2, stride=2)
   - ConvBlock5 (256)
   - ConvBlock6 (256)
   - MaxPool(kernel_size=2, stride=2)
   - ConvBlock7 (256)

- ConvBlock8 (256)
- MaxPool(kernel_size=2, stride=2) (after this pooling the image becomes 1×1)
- Linear Fully Connected layer to 10 classes

2. Define the forward pass function of the neural network.
3. For training, we will be using the Cross Entropy loss and a standard SGD optimization with the learning rate of 0.1, the momentum of 0.9, and the weight_decay of 5e-4. Define them before the training loop.
4. Implement the training procedure, training the model for 50 epochs. The training loop should contain dataloading, zeroing out gradients of the model, forward pass, loss computation, backpropagation, and the step of the optimizer. Please log the training accuracy and training loss after each epoch.
5. Implement evaluation after each training epoch, computing the accuracy of the model on validation set. Do not forget to switch the model to test mode. Please log the validation accuracy and validation loss at each epoch.
6. If the accuracy is better than the previous best, save the model's checkpoint at checkpoint/ best_ckpt.pth.
7. After the training is done, visualize the progression of the training/validation accuracy and losses.

How to report on this assignment. Please include the following files in your solution:

- **main.py** This is your code.
- **log.txt.** In this file, copypaste all the output of your training script. In Lunux, this could be done by running the python command with '> log.txt' suffix. If your run the training using Google Colab, simply copypaste the output.
- **accuracy.png.** Draw a plot with the training/validation accuracy at different epochs.
- **losses.png.** Provide a plot with the training/validation losses at different epochs.
- **overfitting.txt.** Do you observe overfitting in the model? Provide a short answer in this file. Base your answer on the accuracy/loss plots.
- **checkpoints/best_ckpt.pth.** This should be the saved best checkpoint that shows the best accuracy.
- **test.py** Complete the provided template so that we can easily check the performance of your model.

Grading. The solutions will be primarily graded by the (1) accuracy of following the implementation instructions (2) and whether the performance of the trained model is adequate. The teacher's solution reaches an accuracy of around 84%, which should be a rough estimate of what the model should be able to achieve. Reaching a bit smaller scores is acceptable.
The whole solution gives you 20 points. A fully correct main.py gives 12 points. A correct combination of log.txt, test.py, and the checkpoint showing reasonable performance gives

4 points on top. Finally, 4 additional points are given for a correct combination of accuracy.png, losses.png, and overfitting.txt.

TIP: If you do not have a GPU on your machine, you can use Google Colab. This service provides an easy interface and free GPUs to train your models. In this case, please still use the reporting format with 7 files, as described above.