Stephanie Kaes <kaes@vision.rwth-aachen.de>
George Lydakis <lydakis@vision.rwth-aachen.de>

Exercise 5: Convolutional Neural Networks

due **before** 2024-01-17

Important information regarding the exercises:

- The exercises are not mandatory. Still, we strongly encourage you to solve them! All submissions will be corrected. If you submit your solution, please read on:
- Use the Moodle system to submit your solution. You will also find your corrections there.
- Due to the large number of participants, we require you to submit your solution to Moodle **in groups of 3 to 4 students**. You can use the **Discussion Forum** on Moodle to organize groups.
- If applicable submit your code solution as a zip/tar.gz file named `mn1_mn2_mn3.{zip/tar.gz}` with your **matriculation numbers** (`mn`).
- Please do **not** include the data files in your submission!
- Please upload your pen & paper problems as PDF. Alternatively, you can also take pictures (.png or .jpeg) of your hand written solutions. Please make sure your handwriting is legible, the pictures are not blurred and taken under appropriate lighting conditions. All non-readable submissions will be discarded immediately.

Question 1: Classification with CNN ..... $(\Sigma = 25)$

In this exercise you will implement a CNN for classifying image patches. A basic code framework is provided in the Moodle room. In the following tasks, you will complete the basic steps necessary for building a simple CNN with PyTorch framework that can classify images with acceptable accuracy. A dataset (500MB) is provided at `https://omnomnom.vision.rwth-aachen.de/data/cityscapesExtracted.zip` and contains examples for training, validation and testing. It consists of RGB images of three categories: persons, riders and cars which were down-scaled to at most 100×100 pixels for memory reasons.

(a) Have a brief look at how to use TensorBoard with PyTorch so that you can visualize the training progress. Basic learning error outputs have already been added. **(0 pts)**
Hint: You may also look at the tutorial on the official PyTorch website under the following link: `https://pytorch.org/tutorials/recipes/recipes/tensorboard_with_pytorch.html`

(b) The dataset contains extraction artifacts e.g. small parts of cars or far away pedestrians. In order to remove images that do not contain any reasonable content, you should complete `image_dims` function in `cs_dataset.py` and filter out these artifacts by thresholding the size of the content to at least 900 pixels. **(1 pt)**

(c) We also need to resize, turn into Tensors and normalize the images before passing them to the CNN. Resize the image 64 by 64 pixels, then turn them into Tensors and finally normalize the Tensor images using PyTorchs's `Transforms` classes. **(3 pts)**
Hint: You may check the tutorial on the offical PyTorch website "TRANSFORMING AND AUGMENTING IMAGES" under the following link: `https://pytorch.org/vision/stable/transforms.html` in order to understand PyTorchs's `Transforms` classes.

(d) The main building blocks of CNNs are convolution and pooling layers, respectively. For our purpose, a small CNN with 3 layers of each type will be sufficient where each convolutional layer is succeeded by a ReLU activation and a max-pooling layer. The convolutional layers should have a receptive field of 5 by 5 pixels. The layers may comprise 24, 32, and 50 filters which are applied with a stride size of 1. You may pool over blocks of 3 by 3 with a stride size of 2 for all pooling layers. Implement the proposed convolution and pooling layers in `network.py`. **(5 pts)**
Hint: For implementing the convolution in PyTorch you will need to use PyTorch's `torch.nn.Conv2d` class for defining receptive fields, layers and stride by following this link: `https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html`
Hint: In order to implement the pooling layers, please use PyTorch's `torch.nn.MaxPool2d` class under the following link: `https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html#torch.nn.MaxPool2d` *Hint:* Here, you should not implement a ReLU activation layer of the main building blocks of CNNs, since we will use its implementation later on.

(e) In order to map the representation after the last convolutional layer to a class label, you should implement three fully connected layers of size 100, 50, and the number of categories in `network.py`. **(2 pts)**
Hint: For implementing the linear layers, you can take a look at PyTorch's `torch.nn.Linear` class under the following link `https://pytorch.org/docs/stable/generated/torch.nn.Linear.html#torch.nn.Linear`

(f) To start training, we need to feed a batch of input Tensor images to the network, i.e. we need to run a forward pass with a batch of input Tensor images. Initially, you feed a batch of input Tensor images to the main building blocks of CNNs. **(3 pts)**
Hint: Here, you should implement the ReLU activation layers in the main building blocks of CNNs. For implementing ReLU activation layers, you can check PyTorch's `torch.nn.functional.relu` class by following this link `https://pytorch.org/docs/stable/generated/torch.nn.functional.relu.html#torch.nn.functional.relu`

(g) After getting the output of the main building blocks of CNNs, we need to feed this output to the fully connected layers of CNNs. **(2 pts)**
Hint: Please be careful about the shape of the output Tensors.

(h) For training purposes, we need to specify a proper loss on top of a softmax which will be appropriate to predict the final class. In our case, a suitable loss function is a cross-entropy loss which should be implemented as the last step in the `forward` function in `network.py`. **(1 pt)**
Hint: You may use PyTorch's `torch.nn.functional.cross_entropy`. Please check out the following link `https://pytorch.org/docs/stable/generated/torch.nn.functional.cross_entropy.html#torch.nn.functional.cross_entropy`

(i) Implement the training loop in `main.py`. First, you should calculate the training loss and training accuracy, then call backwardpass and update the network weights. **(4 pts)**
Hint: Please look at PyTorch's `torch.optim.SGD` class and its functions by following this link `https://pytorch.org/docs/stable/generated/torch.optim.SGD.html`.

(j) One possibility to regularize the model is early stopping based on the validation set error. After training for a certain number of epochs, the error on a validation set is measured. One may save the best model or stop training if the error increases too much. Although not crucial for this simple task, you should implement a validation **(3 pts)**

step that is used to save the best model. Complete the `val` function in `main.py` which computes the accuracy of the trained model on the validation set.

(k) Finally, we want to use the model to predict class labels for the images in the test set. Complete the `test` function in `main.py` which should return the test accuracy. Finally, you should compute the confusion matrix i.e. a matrix where entry (*prediction*, *label*) corresponds to how often the model predicted *prediction* for an instance of label *label*. What do you observe and why? Is there a problem with this dataset? **(1 pt)**