1 Overview

In this exercise sheet, you will train neural networks for image classification.

1.1 Network Architecture and Initialisation

Using the notebook `image_classification_vgg16.ipynb` as a starting point, train a network using:

a) A pretrained `vgg16` from the model zoo of pytorch (this involves simply executing the notebook)

b) The same network, but initialize the weights with random weights `torch.nn.init.xavier_uniform_`.

c) Compare the training procedure and describe your findings. You probably want to re-organize the source code to facilitate these types of comparisons.

d) A drawback of the `vgg16` is the default input size of 244×244 pixels. In the notebook, the images are scaled as part of the `batch_transforms`. What happens if a `vgg16` is applied to input size 18×18?

1.2 Custom Network Architectures

We will now consider developing our own network architecture, which can handle the 18×18 directly.

a) Implement an encoder network as a sequential model. We will operate in the MNIST dataset, which has a very low resolution of $28x28$ pixels. The network should start by a first convolutional layer with `n_features_1=32` features and kernel size 3×3, to bring the input to 32 channels.

   The main part of our architecture should then consists of `n_blocks=3` encoder blocks. Each block should consist of the following layers:

   (a) a first convolutional layer with `n_features_1=32` features and kernel size 3×3.

   (b) a ReLU layer.

   (c) a second convolutional layer with `n_features_2=32` features and kernel size 3×3.

   (d) a 2×2 max pooling layer.

   After the encoder part, the network should be flattened, such that the last two hidden layers can be fully conncected layers with `n_fully_1=144` respectively `n_fully_2=72` features.

   The network should be initialized by random values (which is done using `torch.nn.init.xavier_uniform`). Train the network and compare the performance to the randomly initialized `vgg16` .

b) Now implement your own version of ResNet. You do so by subclassing (inheriting from) the class `nn.module`. The model should follow the same basic structure as the first model, except that the encoder blocks are replaced by residual blocks. A residual block adds a bypass connection that adds the input to the output.

   Override the forward method to apply the layers to the input x.

c) Compare the performance of the resnet to the performance of the other two networks. Discuss the findings in several sentences.

I needed the following time to complete the task:

1.3 Network Architecture Search

Find out, which architecture works best for the task of MNIST classification.

a) Describe your strategy to test different parameters of the architecture search. What combinations should be tested, how does training time pose a limit?

b) Implement a network architecture search, i.e. an experiment that automatically determines good parameters for your architecture.

I needed the following time to complete the task:

1.4 Diagnosis and Data Cleanup

We are now considering the training data in the directory `data/mnist_png_mislabelled`. The directory contains the same training data as the original directory, but some (100) labels are incorrect.

a) Train your custom network from the imperfact data. Make a quantitative description of the effect on the training.

b) Use your networks and diagnosis tools to identify what is going on. Use the confusion matrix to see if all classes are affected or it the problem is limited to some classes.

c) Use the `top_losses` function to identify mislabelled images. Remove the images from the training data and describe the effect of the data cleanup on the training.

I needed the following time to complete the task: