9 Image classification

In this exercise you will build neural networks for classifying images. We suggest that you start by a relatively easy task of classifying handwritten digits from a widely used dataset MNIST. Then, you can try a more challenging (but still relatively small) dataset CIFAR-10. And finally, for the competition, we will prepare a brand-new BugNIST 2D dataset.

Datasets The MNIST dataset contains images of handwritten digits of 28×28 pixels as shown in Figure 9.1. Ground truth class labels are given together with the images. MNIST contains 60000 images for training the network. In addition there are 10000 images for testing.

Optionally, you can also try classifying CIFAR-10 images[1]. This dataset is similar to MNIST and contains 50000 images for training and 10000 images for testing. CIFAR-10 images are tiny photograps of size 32×32. Images have 3 color channels, and show significantly higher variation in appearance than handwritten digits. Also CIFAR-10 images are divided in 10 classes representing objects and animals.

MNIST and CIFAR-10 datasets (both training and testing images) can be found on internet. You can also find the training sets for both of these datasets by following the the data link from the course page.

Furthermore, the scikit-learn library comes with even tinier dataset of hand-written digits consisting of 8×8 images with approximately 180 images of each class. This dataset can be loaded using the command `sklearn.datasets.load_digits()`. So if you have scikit-learn installed, this is probably the easiest way of testing your MLP on image data.

For the competition, we will prepare images from a BugNIST dataset. BugNIST is a dataset of 3D images of various bugs, collected and processed at DTU, see Figure 9.2.

Classification accuracy The performance of the classifier is evaluated on the testing set of images, and these images should not be used during training. The aim is to obtain the classification with the highes number of correctly classified images from the testing set, that is, the

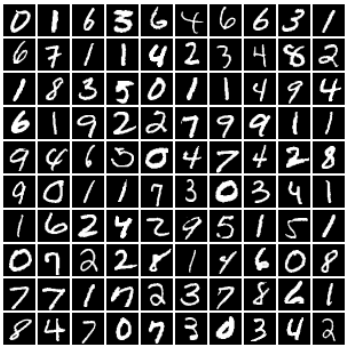

Figure 9.1: Example of the MNIST images.

[1] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009

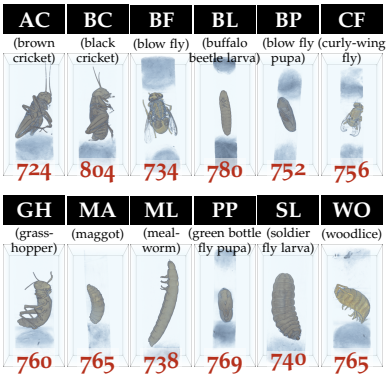

Figure 9.2: Overview of the BugNIST classification data, with 12 classes and approximately 750 volumes for each class. For 2D version of BugNIST dataset we produce several images from each volume.

least number of misclassified images. For this, we will measure the classification accuracy, which is the fraction of correct predictions, and is often given as percentage.

Many additional interesting classification measures can be collected. Figure 9.3 shows confusion matrix for the MNIST classification, where we can see the number of correct predictions for earch class, as well as which classess are likely to be confused by the classifier. Figure 9.4 shows a few examples of correctly classified and misclassified digits.

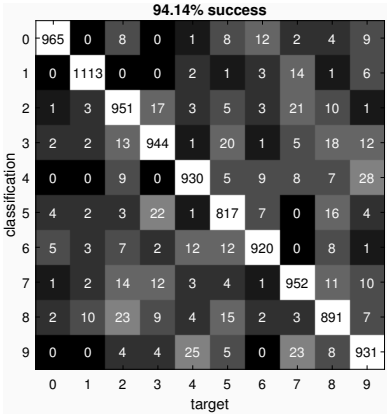

Figure 9.3: Table showing a classification performance example of a classification of the MNIST handwritten dataset.

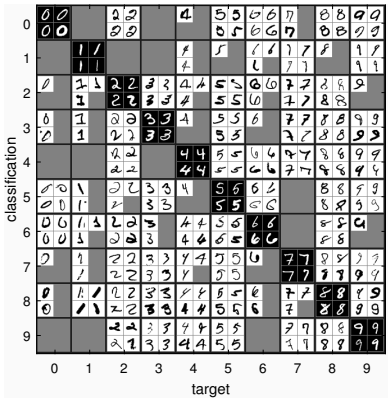

Figure 9.4: Examples of classified handwritten digits and misclassified digits.

9.1 *Handling data*

You should always get a good overview of the dataset you are working with, and visualize a few images from the dataset.

Furthermore, you need to transform your data, such that it can be passed further to your MLP. For image classification you should use a fully connected feed forward neural network, similar to the one you implemented last exercise. But in contrast to last exercise, where data was points in two dimensions, you now have images. Therefore you need to reshape every image into a vector. In case of MNIST images, which are only 28×28 pixels, the vector representation will have 784 dimensions. Therefore, the network classifying MNIST images should take in 784 dimensional vectors and return a 10 dimensional vector for classifying the digits 0 to 9. For the CIFAR-10 data, the input vector becomes 3072 dimensional ($32 \times 32 \times 3$).

You should also standardize your data, and there are different approaches. For example, MNIST images which are saved as uint8 integers with values between 0 and 255, and if you divide pixel values with 2/255 and subtract 1, you will ensure that all pixel values are between -1 and 1. Alternatively, you can standardize the dataset such that it has zero-mean and standard deviation of 1. Furthermore, when working with images you can safely assume that overall image intensity is not important for the classification. You can therefore standardize each image individually, for example ensuring that each image vector has zero mean and a unit length (using 2-norm).

It is also worth considering the data type. Single precision is faster than double, so you should consider if you want faster computations, at the cost of lower precision. You can experimentally evaluate if the high precision is necessary.

As for the target values, in most datasets these are saved as class labels, in case of MNIST, the digits 0 to 9. You should transform those to one-hot encoding used by your networ, that is a 10 dimensional vectors with 1 in the dimension representing the target class and zeros elsewhere.

Validation If you use all available training images for training the weights of your network, it is difficult to estimate how the network will perform on testing images, which it has not seen during training. Your network could overfit (learn to classify training data very well, but still do poorly on unseen data) without you realizing. To observe how a model performs on the data it has not trained on, you can reserve a fraction of your training images (for example 20%) for validating the performance of the model. Then you can compare between different networks (different hyper-parameters) based on the validation accuracy. And you can monitor the perfrmance of the network during training, and stop the training if the network starts to overfit, which is seen by by a drop in classification performance of the validation data.

9.2 *Network and optimization improvements*

A part of optimizing the neural network is by changing its architecture. Therefore, you should implement your network such that you can change the number of layers and the number of neurons in each layer.

A large number of techniques for optimizing the performance of the neural network has been proposed. You can use the book on deep learning by Goodfellow et al.[2] to get ideas for many improvements, and below we mention a few commonly-used approaches.

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016

Minibatches Obtaining a strong classifier using 50000 training images requires many iterations of the backpropagation algorithm.

Therefore, it is a good idea to utilize vectorized code in your implementation. This can be done by computing the gradients for subsets of the training data using minibatches. You can have a minibatch as a matrix and compute the forward propagation and gradients using matrix operations. By averaging the gradients obtained from the minibatch, the backpropagation can be carried out in the same way as you would do when training with one sample at a time. Due to the averaging, the obtained gradients are less affected by noise and it is typically possible to have higher learning rates.

Momentum and adaptive learning rate Optimization with stochastic gradient decent can be slow, but updating the gradients using momentum can accelerate the learning. Momentum is obtained by remembering past gradient, and computing the update as a weighted combination of the previous gradient and the new gradient. Hereby, the update is computed as a moving average with exponential decay. Another way of ensuring convergence of the gradient decent is by adapting the learning rate. Here you can adapt the learning rate to the individual gradient estimates.

9.3 Regularization

It is important that the neural network generalizes well such that it can classify new unseen data. Since neural networks often have many parameters it is easy to overfit the model, especially on small datasets where a very low training error can be obtained, but the validation error will be high.

One way to overcome the problem with training a neural network on small datasets is through dataset augmentation, where fake data is fabricated by small modifications of the input data. This can be done by small permutations or by adding small amounts of noise. Hereby a much larger dataset can be obtained, which can help the training.

Instead of adding noise to augment the training data, small amounts of noise may be added to the hidden units in the network. You can add random noise in each minibatch iteration. Noise can also be added to the output targets for obtaining better performance.

Dropout is another method for regularizing the neural network. Here a random selection of neurons are set to zero during each minibatch iteration leaving out these neurons in that iteration. Setting the neurons to zero resembles having a number of different neural networks and is inspired by ensemble methods.

9.4 Exercise and classification competition

The tasks for this week are:

1. Implement a fully connected neural network for image classification. Use MNIST (or scikit-learn digits) dataset while making the necessary changes to handle image data. Provided is a short script showing how scikit-learn digits can be prepared before training.

2. Train the neural network on a part of the training data (e.g. 50000 images) and validate it on another part (e.g. the remaining 10000 images). Plot the training and validation error for each iteration (epoch).

3. Implement one or more optimization strategies and document how this affects the obtained result. As minimum, train your network in minibatches.

4. Try experimenting with regularization methods.

5. (Optional) Use your network for classifying CIFAR-10 images.

6. Train your network to classify BugNIST dataset. When you are satisfied with the obtained result upload the trained network together with the code for running it. The submission details will be written

on the course page. It should be made clear how your code should be run – you can either document the code or attach a `README` document. The participation in the competition is not mandatory, but we strongly encourage all students to participate.

7. The winners will be announced based on the accuracy achieved on the training set. Hovewer, we will also measure how fast does your network predict. So do an effort in making the code efficient.