CAP 6618 Spring 2019
Christian Garbin
Module 5 Assignment - Digit classification using convolutional neural networks

Objective of this assignment

The objective of this assignment is to "*Design and implement a complete solution to the problem of handwritten digit recognition, stated as follows: “given an image of a handwritten digit, produce the corresponding numerical value.” Essentially, you will build, fine-tune, and evaluate a 10-class (multi-layer perceptron) convolutional neural network (CNN)-based classifier which will employ the pixel values of the images as features."*

The digits to be recognized are the ones from the MNIST dataset [1].

Files submitted

Two Jupyter notebooks are submitted as part of this assignment:

1. CAP6618_PP3_cgarbin_experiments.ipynb: based on the provided CAP6618_PP3.ipynb notebook, contains the experiments to optimize a CNN with Keras and explore the results.
2. CAP6618_PP3_cgarbin_final.ipynb: contains the analysis and optimization of the CNN classifier using Keras and exploration of the results.

Baseline

The starting point for this assignment was a set of two networks:

- A network with two dense layers, with dropout for regularization. This network achieved 98.38% accuracy after 12 epochs.
- A network with convolution layers and pooling layers, with dropout for regularization. This network achieved 99.16% after 11 epochs.

Since the assignment is specifically about CNNs, the second network was chosen as a baseline for further experiments.

Experiments and conclusions

Instead of chasing raw performance, as measured by better accuracy, in this report we chose to compare the overall efficiency of the network. "Efficiency" in this case is a combination of how large or small the network is for the accuracy it provides, combined with how it takes to train the network to achieve that accuracy.

For those experiments we chose two network topologies:

- A small topology, based on LeNet-5. This network as 20x fewer parameters than the baseline CNN network.
- A large topology, based on VGG-13. This network has 25x more parameters than the baseline CNN network.

The following table shows the results of the tests.

Network	Accuracy (%)	Time per epoch (s)	Epochs to converge	Trainable parameters	Size of model (.h5) file (MB)
Baseline	99.14	4	11	1,199,882	14.4
LeNet like	98.78	3	11	61,706	0.8
LeNet like with dropout	98.40	4	N/A	61,706	0.8
LeNet like with dropout and early stopping	98.69	4	19	61,706	0.8
LeNet like with batch normalization and early stopping	98.76	6	8	62,158	0.8
LeNet like with batch normalization, early stopping and ReLU	**98.79**	**6**	**6**	**62,158**	**0.8**
VGG like	99.30	25	8	32,393,410	388.8

The VGG-like network had the best accuracy, but it is the largest network by a wide margin. As a side effect of that, it is also the slowest to train. Because we are using overall efficiency to choose a network, these numbers disqualified the VGG-like network. The best accuracy it provides is offset by the large memory usage and long training time.

The LeNet-5 network with batch normalization and ReLU (highlighted in the table) was chosen as the most efficient network for the task. It reached the accuracy level of the baseline network with 20x fewer parameters. It also reached that accuracy faster. Although this network has about the same performance as the "LeNet like" network (no regularization or other changes), the fact that it has batch normalization in place makes it a better candidate for further experiments, e.g. data augmentation.

The main conclusion from these experiments is that even a small CNN is able to reach performances comparable to networks that are 500x larger (for this category of problems, of course). Therefore, a general approach for choosing an architecture is to start small and grow it as needed, as opposed to reaching for the largest network from the beginning.

The notebook has more details of the experiments executed to build the table.

Comparison with previous programming projects

The classifiers in the previous assignments differ from the classifier in this assignment in these ways:

Module 3 - Non-neural network classifier

Module 3 used only classifiers not based on neural networks. The major differences between the CNN classifier used in this module and the ones used in module 3 are:

- **Training algorithms:** classifiers in module 3 use a number of training algorithms that are, generally speaking, based on partitioning or distance measures to classify the instances. Neural networks, on the other hand, are based on one training algorithm: use SGD to minimize a cost function.
- **Feature engineering**: as a consequence of the nature of their training algorithms, classifiers in module 3 are more sensitive to the data used for training. For example, feature engineering that helps increase the distance between classes (e.g adding a quadratic feature to create another dimension), may have a significant effect on accuracy. Neural networks can extract features from the dataset on their own (with the exception of feature scaling, they need little help to learn features).
- **Memory usage at training time**: The first challenge with these classifiers is that many of them do not support batch or mini-batch training. In those cases, it means the classifier needs to load the entire training dataset in memory for training, which may be infeasible for some dataset. All CNN classifiers (and NN classifiers in general) can be trained with smaller batch sizes, reducing memory usage.
- **Memory usage at runtime**: some classifiers in module 3 are very compact at runtime. A decision tree, for example, can be modeled as a sequence of if/else statements, which is compact and fast to execute. Neural networks models require a large set of weights, which results in hundreds or thousands of kilobytes of memory.
- **Training time optimization**: to reduce training time, optimizers in module 3 usually spawn more processes that work on different pieces of data. Besides spawning parallel jobs, because of the nature of its SGD training algorithm (based on matrix algebra), neural network classifiers are also able to optimize training time by making use of specialized hardware, i.e. GPUs and TPUs.
- **Reusing of a trained classifier**: CNNs can also be generalized to other tasks with *transfer learning*, transferring the lower layers of a network to other tasks. This comes from the fact that these layers "learned" general concepts about the input data that can be applied to other tasks. The classifiers from module 3 are specific to the task they have been trained on.
- **Interpretability**: several classifiers in module 3 are easily interpretable, i.e. it is easy to explain why they are making certain decisions. For example, decision trees, when shown in a graphical format with the parameters that split each node, can be explained to lay people. The decision process of a neural network is obscure even for trained professionals (there are some advances in this area, but it is still not close to being explainable).
- **Stability of the solution**: classifiers in module 3 can suffer from instability: adding just one more training instance can modify the solution in significant ways (e.g. change the root node of a decision tree). This may or may not be a problem. It can be argued that flexibility to new information is good, but on the other hand, missing just a few data points when collecting a training set may result in poor performance in real life. Neural networks are more stable to small changes in the training data set (but see the next point).
- **Amount of data needed for training**: continuing from the previous item, the stability of the solution, it can be argued that neural networks are stable to small changes in the training data set because they need large training data sets in the first place. Classifiers in module 3 can be trained with small data sets and be usable (albeit limited).

In terms of accuracy, the classifier in module 3 reached 97% accuracy, which is impressive but seems to be the ceiling for these classifiers. Pushing beyond that limit may require dimensionality reduction, feature engineering, and other techniques. In contrast to that, even a simple CNN classifier reaches over 98% accuracy.

Module 4 - MLP classifier

Module 4 used MLP, a classifier that is already in the same family of "neural network classifiers" as the ones used in module 5. It has more similarities than differences, starting with being trainable with SGD.
The major differences appear when MLP networks are compared with CNNs.

- **Layers**: MLPs are composed of dense (fully connected) layers only. CNNs have convolution and pooling layers.
- **Input shape**: because the input layer of an MLP is fully connected to the input data, it expects all instances in the training and test data to have the same shape. If, for example, it is being trained with pictures, all pictures must have the same dimensions. A CNN "scans" the input with a filter of small dimensions (compared to the dimensions of the input data), so it can adapt itself to the shape of the input.
- **Abstraction**: CNNs layers learn abstractions about the data, with each successive layer consolidating information from the previous layers and learning higher-level abstractions.
- **Transfer learning**: as a result of their ability to abstract information during training, CNNs can be repurposed to other tasks (the lower layers of a CNN, to be precise).

Sources

[1] LeCun et al., The MNIST Database, http://yann.lecun.com/exdb/mnist/, accessed on February 7th, 2019.