Assignment 2

Julia Putz, Dario Giovannini

May 2023

Contents

1 Introduction

During this exercise, iterative optimization and parametric models for image classification were covered. Therefore, several experiments to determine the most suitable combination of hyperparameters, optimizer types and different network architectures including regularization using dropout and weight decay, as well as data augmentation were carried out.

2 Gradient descent - Part 1

2.1 How it works

Basically, gradient descent is used to find a local minimum of a function and hence minimize the loss. Therefore, iterative steps are taken into the opposite direction of the so called gradient. The gradient is a vector of partial derivatives that tells us the direction in which the cost function is increasing the most rapidly. It is calculated by computing the partial derivative of the cost function with respect to each parameter. As we are dealing with a two-dimensional array, which represents the function, we take the value of the array at position x1 and subtract it from the array at position x1 + epsilon. The same has to be done for x2 as well. If plotting the gradient, one can see that it shows into the direction of the local minimum.

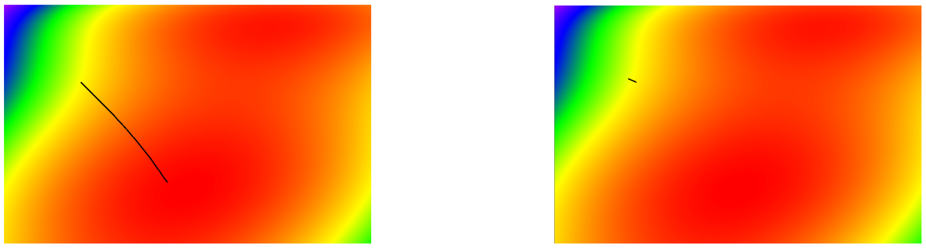

Figure 1: visualized gradient (epsilon = 10, learning rate = 10, beta = 0, nesterov = False) using SGD (right) and Adam (left)

2.2 Findings

When experimenting with the example data, provided as images, we found that the results obviously depend heavily on the chosen point which is evaluated. In the implementation of the gradient computation, we decided to round the locations x1 and x2. Hence, the epsilon should not be chosen too small, e.g. if x1 = 300.0 and epsilon is set to 0.4, the values used to compute the gradient would be taken from the same location (array[300][x2] - array[300][x2]). Therefore, only epsilon values larger than 0.5 were used for experimenting. For the final comparison we experimented mainly using Adam and SGD as optimizer. In most of the compared settings, Adam reached a lower loss than SGD, even within way less iterations needed. This was somehow expected, as SGD is known to get stuck on local minima. This however can be counteracted by setting the momentum. When e.g. setting momentum = 1, the visualized result looks very similar to the result using Adam when using the same parameters.

3 Network Architecture - Part 2

The initial architecture for the neural network consists of 2 rounds of convolutional layers followed by a ReLU and Max-Pooling layer, then two linear layers to produce the output vector of size 2, as shown in figure 2. The combination of a convolutional layer followed by ReLU and Max-Pooling layers follows the recommendation from the lecture, and a stack of two such blocks is close to the limit of what is reasonable given the small image size, as each block cuts the dimensions in half due to the pooling layers, bringing the height and width after the second pooling layer down to only 8 ”pixels”.

Following the blocks of convolutional and pooling layers, the output is flattened for input to the linear layers, which are fully connected and finally output their result in a layer with only two nodes, corresponding to the number of classes. To obtain actual predictions, the ”predict” method of the CNN wrapper class passes the network output through a simple Softmax layer.

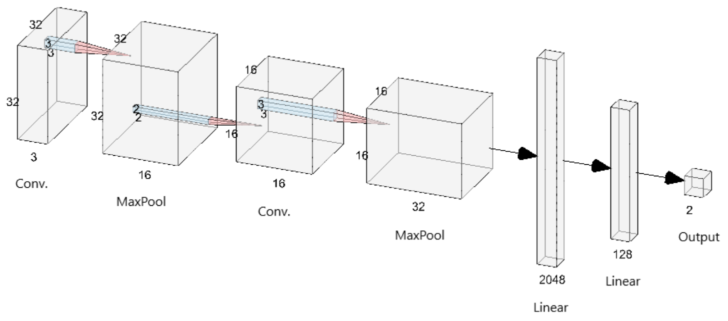

Figure 2: Diagram of initial network architecture, created with NN-SVG[1]

3.1 Initial Parameter Selection

The principal hyperparameters for this network are the learning rate and the weight decay, both applied to the Stochastic Gradient Descent Optimizer (SGD). The initial values, chosen essentially at random, were 10^{-4} for the learning rate, and 0.2 for the weight decay. A test run with 100 epochs showed that these were

[1] `http://alexlenail.me/NN-SVG/AlexNet.html` using the webGL render, captured using the Windows snipping tool. Layer labels added manually using Paint3D.

not good values, resulting in a final test accuracy of only 45% - worse than a random guess. Experiments with a few different values found a learning rate of 0.02 and weight decay of 0.005 as reasonable values, resulting in a minimum training loss of 0.361, best validation accuracy of 0.76 and final test accuracy of 67.5%. The training loss never reached values below 0.35 in any experiments.

4 Further Experiments - Part 3

Several experiments were conducted to find ways to improve classificaton performance of the network, of which some worked and some did not. Significant improvements were seen by including data augmentation techniques, specifically flipping and random cropping, but the attempt to further improve data augmentation through random rotations led to a decrease in performance. Early stopping was part of the training process from the beginning, as this technique had already been used in the first assignment and the process largely reused, though experiments with an increased number of epochs showed increases in test accuracy, indicating that model capacity was usually not reached yet. Regularization was implemented via the weight decay parameter of the SGD optimizer, which had already been used in the initial network evaluation in the previous section, as well as a single dropout layer with varying dropout chance and position within the network, either after the first or after the second convolutional block.

Experiments with network architecture were limited, primarily due to the small input image size which made it difficult to effectively apply multiple convolutional layers. One experiment was conducted using a higher number of feature maps and an additional linear layer, but this experiment resulted in a reduction in test accuracy compared to the previous best result achieved in addition to a noticeable increase in training time, and was thus reverted - it is not reflected in table 1.

input augmentations	dropout	learning rate	weight decay	test accuracy
None	No	0.0001	0.2	0.45
None	No	0.02	0.005	0.675
flip, crop	50% after conv 1	0.02	0.005	0.662
flip, crop, rotate	50% after conv 1	0.02	0.005	0.588
flip, crop, rotate	50% after conv 1	0.02	0.005	0.662
flip, crop, rotate	30% after conv 1	0.02	0.005	0.537
flip, crop, rotate	30% after conv 2	0.02	0.005	0.55
None	50% after conv 2	0.02	0.005	0.662
flip	50% after conv 2	0.02	0.005	0.688
flip	50% after conv 2	0.02	0.02	0.65
flip, crop	50% after conv 2	0.02	0.02	0.65
crop, rotate	50% after conv 2	0.04	0.002	0.662
flip, crop	50% after conv 2	0.04	0.002	0.725
flip, crop, rotate	50% after conv 2	0.04	0.002	0.637

Table 1: Table of basic experiments, with identical architecture and 100 epochs.

An overview of the basic experiments performed can be found in table 1. All of these experiments were ran on the same architecture except the dropout layer, and a single training run with 100 epochs each. Initially, the dropout layer was placed after the first convolutional block, with a 50% chance for any given node to be dropped. A lower value was tried but resulted in a significant reduction in test accuracy combined with the input augmentation, indicating the necessity of this regularization technique to avoid overfitting, as both of those experiments had comparable validation accuracies and training losses to experiments with higher test accuracies.

In addition to experiments with augmentation and regularization techniques, further variations were done on the learning rate and weight decay parameters. It was observed that a lower weight decay performed somewhat better, while a higher learning rate enabled slightly quicker growth of validation accuracies during training, and were thus preferred. As only a limited amount of experiments could be performed and often several factors were changed these are not very robust results, but mainly served as an exploration of the possible space of parameters as well as network and training configurations.

4.1 Impact of the "rotate" augmentation

The "rotate" augmentation operation was introduced in an attempt to further increase test accuracy of the model by further augmenting the training data. The operation consists of rotating the input image by 90 degrees 0, 1, 2 or 3 times, resulting in a quarter chance each for no rotation, a quarter rotation, half a rotation and three quarters of a rotation.

In order to compare variance between training runs as well as more closely assess the impact of the additional rotate augmentation operation, the last two experiments in table 1 were ran another 10 times each, using 200 epochs of training rather than 100. The resulting average test accuracy over the ten training runs for the experiment using only flip & crop operations was $0.7735\% \pm 0.025\%$, with a maximum of 0.812. In comparison, the experiment using the rotate operation in addition to flip & crop showed an average test accuracy of only 0.655 ± 0.036, with a maximum of 0.713.

5 Conclusion

Even simple convolutional neural network architectures are capable of achieving decent accuracies in image classification tasks. Given limited training data, some techniques can aid in improving model performance, specifically image augmentation and regularization. The former consists of presenting the model with variations of an image instead of the same image repeatedly over epochs, which improves its performance on unseen data, and the latter helps prevent overfitting of complex models. However, not all techniques are equally useful, and the "rotate" technique tried here was shown to not lead to a further increase in model performance when combined with the "flip" and "crop" operations.