Computer Vision 1 Project - CNN

Nordin Belkacemi (15096157)
Max Belitsky (13740512)
Thomas Komen (12556963)

October 2023

Contents

1 Introduction

For the classification of objects in images, neural networks are one option to accomplish this task. By training the networks on a collection of pictures with their respective label, such a network could potentially identify images after doing this for a sufficient amount of time.
These networks can have a range of different forms and sizes, with differences in the amount of layers, the amount of neurons in each layer, and the type of operation that each layer performs.

2 Project Setup

For this project, we will be looking into training two of these networks: A two-layer network, and a Convolutional network. In order to train our models, we will be using the CIFAR-100 dataset (Krizhevsky, Hinton, et al., 2009). This is a dataset of 100 classes, with 600 images per class. These are divided into 500 images to train on, with 100 images to test on. Each image is of 32 by 32 pixels, and in RGB. An example of this is shown in figure 1.

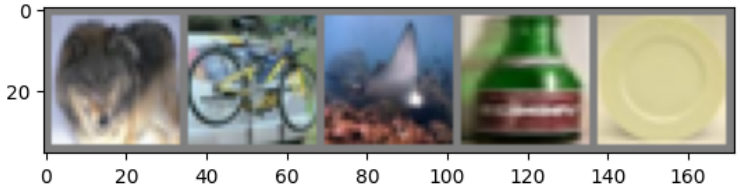

Figure 1: 5 example images of the CIFAR-100 dataset. The classes are wolf, bicycle, ray, bottle, plate

Later on, the convolutional network trained on this dataset will be adapted to be able to train on another dataset, the STL10 dataset (Coates, Ng, & Lee, 2011). This is a dataset containing 500 training images and 800 testing images per class, with 10 classes. These images are 96x96 pixels.

3 Choices for model architecture

In this section, we shall be discussing the architecture of the two networks we use for this project, with explanations as to why these choices were made.

3.1 Two-Layer Model

The first model used in this project is an ordinary two-layer model. This model has an input size of 3*32*32, for images of 32 by 32 pixels, and 3 colour channels. The network contains one singular hidden layer of variable size, which can be tuned in order to produce better results. Since the dataset used contains 100 classes, the output size will be this 100, with each output corresponding to one of these classes. Each layer is fully connected, and makes use of the ReLu activation function.
Given a hidden layer size of H nodes, this results in 3*32*32*H+H trainable parameters in the first layer, and H*100+100 in the second layer, resulting in a total of 3173*H+100 parameters for the model in total.

3.2 Convolutional network

The second model used in this project is a convolutional network, with its architecture based on LeNet-5 (Lecun, Bottou, Bengio, & Haffner, 1998). The first layer is a convolutional layer with 3 channels input and 6 channels output, and has a kernel size of 5. Next, a subsampling layer is applied to this output, where the maximum value of each 2x2 window is taken, effectively cutting the size of each dimension in half. The same convolution is done again, this time from 6 channels to 16 channels, and the same subsampling is applied to these values as well. From this point on, the convoluted matrices are flattened into a vector, after which a linear layer maps these values together to 120 outputs. These 120 values are then mapped to 84 as described in the paper of LeNet-5, after which the output layer maps this to 100 outputs once again representing the 100 classes. Each layer except for the subsampling layers use the tanh activation function.
This results in 5*5*3*6+6 = 456 trainable parameters for the first convolutional layer and 5*5*6*16+16 = 2416 for the second layer. The first linear layer has 400*120+120 = 48120 trainable parameters, 120*84+84 = 10164 for the second layer and 84*100+100 = 8500 for the output layer. This results in a total of 69656 trainable parameters for this network.

4 Training and Validation

4.1 Initial training setup

As mentioned before, the dataset used to train the two networks is the CIFAR-100 set. In order to train the networks, the dataset was transformed in two ways: the data was normalized, and turned into a tensor to allow training. This transformation is the same for both training and testing.
For optimization, a stochastic gradient descent algorithm provided by pytorch (Paszke et al., 2019) was used to train the parameters, initially set with a learning rate of 0.001 and a momentum of 0.9. These values will later be tuned in section 4.3.
The loss function used is CrossEntropyLoss, which is also provided by pytorch.

4.2 Initial results

For the two layer network, as a baseline we trained with a hidden layer size of 50 neurons. Both networks were trained for 50 epochs in order to get initial results without any tuning.

Two Layer Network

When plotting the loss over time of the two layer network as shownn in figure 2, it is visible that the loss starts to flatten out to a degree.

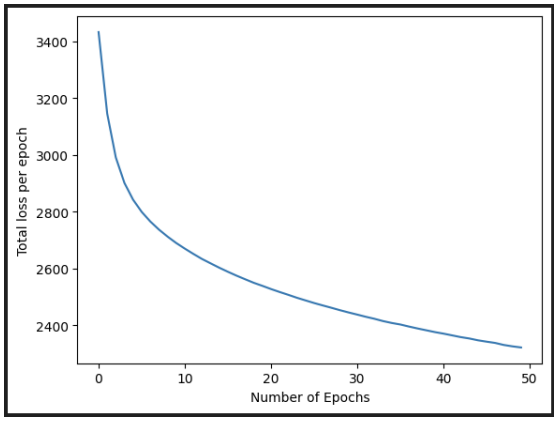

Figure 2: The loss over time for the initial twolayer

When testing the accuracy of the network on the test images at this point, we got 22.5%. This will be used as the baseline for future tuning.

Convolutional Network

Having plotted the initial run for convnet with 50 epochs, as shown in figure 3, the amount of epochs was not sufficient to get close to convergence of the loss yet. However, the accuracy of the model at this point was already at 27.0%.

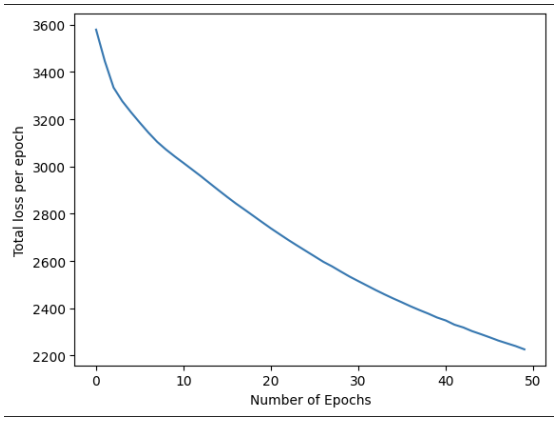

Figure 3: The loss over time for the initial twolayer

4.3 Hyperparameter tuning

Two Layer Network

For this network we decided to tune the amount of neurons in the hidden layer. We tested between 10 and 100 neurons with steps of 20 neurons. As higher amounts of neurons significantly increase the amount of parameters, the lowest amount at which we get a good accuracy would be preferred.
As the amount of parameters increases, we are also training these networks for 70 epochs.

Convolutional Network

For this network, we have tuned the amount of channels both the first and the second network produce, and we tuned the learning rate and momentum. Each tuning is trained for 70 epochs, and the accuracies on the validation set are compared.
For the learning rate and momentum, we first tuned learning rate, with the rates 0.1, 0.01, 0.001 and 0.0001. With a momentum of 0.9, this produced the following accuracies:

Learning Rate	Accuracies after 70 epochs
0.1	14%
0.01	22%
0.001	28.5%
0.0001	8.5%

Because of this, we continue with the learning rate of 0.001. When tuning the momentum, we got the following results:

Momentum	Accuracies after 70 epochs
0.9	28,5%
0.5	11%
0	8%

So for the optimization, we kept using the learning rate 0f 0.001 and momentum of 0.9.
For the first layer, we convolve to 3, 6 or 10 outputs. The second convolutional layer produces 12, 16 or 18 outputs.

4.4 Comparing the models

In terms of architecture, the end of the two models are fairly similar. Both models use just a few linear layers to determine the class of each image. The key difference between the two models here is that the convolutional network has a couple of layers before that convolve and subsample the inputs in order to learn how to highlight features making later classification better. The improvement of this convolution shows already without tuning in section 4.1, where in 50 epochs time the accuracy is increased by almost 25%.

5 Finetuning the convolutional network for STL-10

5.1 Preparation

In order to finetune the convolutional network for the STL-10 dataset, both the data and the network needed to be adapted for the best performance. For the data, we extracted 5 classes out of this set, those classes being "car", "deer", "horse", "monkey" and "truck". This, because these were the classes already trained on with the CIFAR-100 dataset in section 4. The transformers used on the data were the same as before, except the images were also downscaled to 32x32 pixels. This way, the structure of the previous network could be preserved as much as possible, without having to accommodate for larger images. This improves the time it takes for the network to adapt to the new dataset significantly.
For the network itself, we reused the best convolutional network from section 4.3. Only the last fully connected layer was changed. Since there were only 5 classes to predict, the last layer was changed to map to 5 outputs instead of 100.
This also reduces the amount of trainable parameters for the ouput layer. This goes from 84*100+100 to 84*5+5=425 trainable parameters, reducing the parameters for the final layer by 8075.

References

Coates, A., Ng, A., & Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (pp. 215–223).

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324. doi: 10.1109/5.726791

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc.