ENPM 673 Project 6

Daniel Sahu
Brenda Scheufele

16 May 2020

1 Introduction

Convolutional Neural Networks (CNN) are a popular form of deep learning often used in image processing. They perform well in the task of image classification with little image pre-processing required, and are adept at extracting useful features from datasets. Our implementation focuses on the classification of images into ”cats” or ”dogs” using this data set. Our implementation uses the Pytorch module in Python3, and our baseline code is modified from this Google Colab tutorial.

2 Initial results

The baseline code resulted in roughly 50% accuracy, which corresponds to the proportion of labels in the dataset. This, along with the fact that the loss rates were not improving, implies no useful features were being extracted. This was using one conv layer, resizing the images to 300 x 300, using a batch size of 32, and running it for 10 epochs.

Epoch	Loss	Train Accuracy
0	0.8240	50.45
1	0.8272	50.45
2	0.8276	50.45
3	0.8264	50.45
4	0.8248	50.45
5	0.8264	50.45
6	0.8272	50.45
7	0.8256	50.45
8	0.8272	50.45
9	0.8276	50.45

3 Improving accuracy

One of the most difficult tasks of training via CNN is figuring out ”optimal” hyperparameters: the number and type of layers, their size, how to pre-process or normalize the data set, the batch size, the learning rate, etc. The number of permutations are staggering and clearly a perfect solution is not feasible. To narrow down the possibilites we rely on some of the work people have put together in the past documenting best practices and methods for improving overall training accuracy. Many of our changes were following the recommendations to the layers from this tutorial.

Our improvements were the following:

1. Added two convolutional layers with kernel size (5x5).
2. Added an addition linear layer.
3. Added two pooling layers.
4. Normalized the training data.
5. Resized and standardized the training data to (50x50).
6. Used a batch size of 32.

The resulting accuracy was improved, and the loss values consistently went down:

Epoch	Loss	Train Accuracy
0	0.6935	56.50
1	0.6679	61.09
2	0.6458	64.97
3	0.6297	68.04
4	0.6096	69.10
5	0.5927	71.77
6	0.5858	73.03
7	0.5571	76.21
8	0.5466	78.38
9	0.5364	79.23
10	0.5143	81.60
11	0.5016	83.01
12	0.4808	85.03
13	0.4775	85.28
14	0.4591	87.60
15	0.4486	88.41
16	0.4409	89.36
17	0.4337	90.07
18	0.4210	91.33
19	0.4048	93.04

4 Overfitting and Test results

A large pitfall to avoid when using CNN is to make sure you're not overfitting the data. Overfitting in this context means the model finds and extracts salient features that are only useful in the training data, and don't generalize to new situations. For example, supposed in our training data all the dogs happened to be facing to the right, and all the cats happened to be facing to the left. The CNN might pick this up as an import distinguishing feature between cats and dogs, when really it was just a red herring. The best way to validate that your dataset is performing as expected it to run your model against a test set of images that it has never encountered before, and check to make sure it's producing the correct labels. We see in Figure 1 that our validation data follows with our training data, indicating that we are not overfitting.

The results for training and testing are :

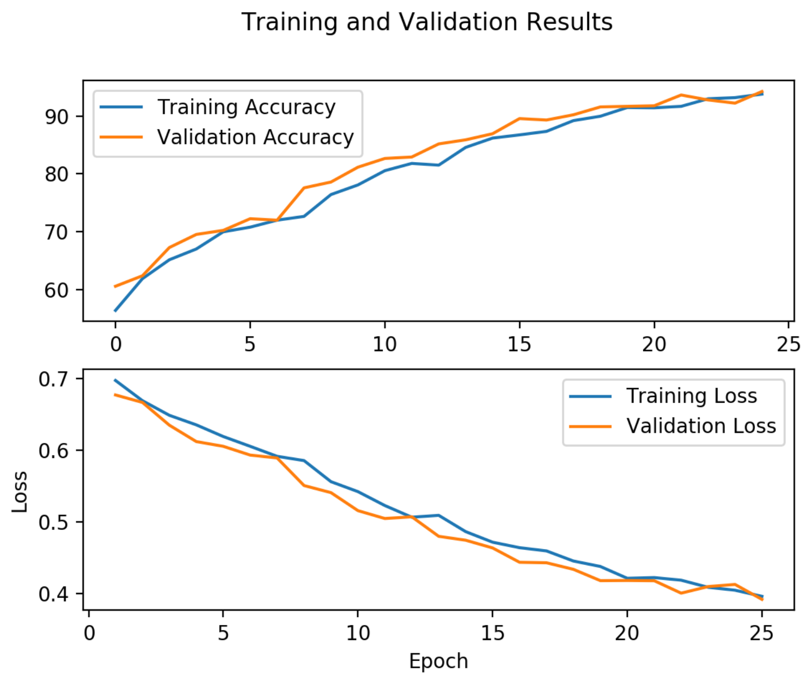

Figure 1

Epoch	Loss	Train Accuracy
0	0.7268	54.18
1	0.6762	60.48
2	0.6616	62.65
3	0.6537	64.11
4	0.6199	69.10
5	0.6077	70.16
6	0.5817	73.84
7	0.5627	75.55
8	0.5556	76.51
9	0.5306	79.28
10	0.5134	81.10
11	0.4934	83.77
12	0.4616	87.20
13	0.4572	87.90
14	0.4539	88.10
15	0.4366	89.67
16	0.4174	91.63
17	0.4113	92.59
18	0.4123	92.14
19	0.4016	93.20

5 Code

The code is found at this GitHub repository.