Improved Regularization of Convolutional Neural Networks with Data Augmentation

By Sanjog Kumar Handique

University of Massachusetts Lowell

1. Abstract:

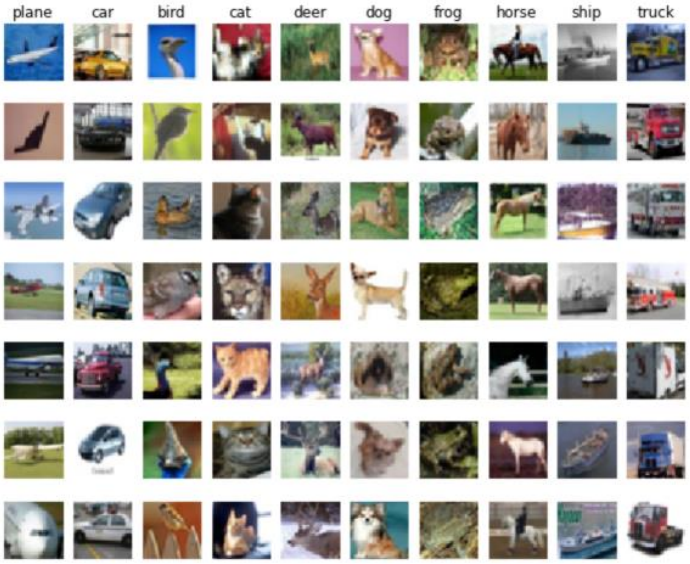
Convolutional neural networks are capable of learning powerful representational spaces, which are necessary for tackling complex learning tasks. However, due to the model capacity required to capture such representations, they are often susceptible to overfitting and therefore require proper regularization in order to generalize well. In this paper, we apply **Convoluted Neural Networks** to a publicly available famous dataset – CIFAR 10 The CIFAR-10 dataset consists of 60000 32x32x3 color images in 10 equal classes, (6000 images per class). Each class of images corresponds to a physical object (automobile, cat, dog, airplane, etc). It was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. I will try to improve the accuracy by introducing a regularization technique called **Dropout** which is recently invented. Also I'll be using another technique called Data Augmentation which generates more training data from old data, so that we may have enough training and avoid overfitting. We evaluate this method by applying it to current state-of-the-art architectures on the CIFAR- 10, Code available at https://github.com/sanjogh777/Ciifar-10.

2. Introduction:

In recent years deep learning has contributed to considerable advances in the field of computer vision, resulting in state-of-the-art performance in many challenging vision tasks such as object recognition, semantic segmentation, image captioning, and human pose estimation. Much of these improvements can be attributed to the use of convolutional neural networks (CNNs) [1], which are capable of learning complex hierarchical feature representations of images. As the complexity of the task to be solved increases, the resource utilization of such models increases as well: memory footprint, parameters, operations count, inference time and power consumption. Modern networks commonly contain on the order of tens to hundreds of millions of learned parameters which provide the necessary representational power for such tasks, but with the increased representational power also comes increased probability of overfitting, leading to poor generalization. In order to combat the potential for overfitting, several different regularization techniques can be applied, such as data augmentation or the judicious addition of noise to activations, parameters, or data. In the domain of computer vision, data augmentation is almost ubiquitous due to its ease of implementation and effectiveness. Simple image transforms such as mirroring or cropping can be applied to create new training data which can be used to improve model robustness and increase accuracy [2]. Large models can also be regularized by adding noise during the training process, whether it'be added to the input, weights, or gradients. One of the most common uses of noise for improving model

accuracy is dropout [3], which stochastically drops neuron activations during training and as a result discourages the co-adaptation of feature detectors.

In this work we consider applying noise in a similar fashion to dropout, but with two important distinctions. The first difference is that units are dropped out only at the input layer of a CNN, rather than in the intermediate feature layers. The second difference is that we drop out contiguous sections of inputs rather than individual pixels, as demonstrated in Figure 1.



In this fashion, dropped out regions are propagated through all subsequent feature maps, producing a final representation of the image which contains no trace of the removed input, other than what can be recovered by its context. This technique encourages the network to better utilize the full context of the image, rather than relying on the presence of a small set of specific visual features.

3. Background:

3.1. Dropout:

Because a fully connected layer occupies most of the parameters, it is prone to overfitting. One method to reduce overfitting is dropout. At each training stage, individual nodes are either "dropped out" of the net with probability {1-p} or kept with probability {p}, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. Only the reduced network is trained on the data in that stage. The removed nodes are then reinserted into the network with their original weights.

In the training stages, the probability that a hidden node will be dropped is usually 0.5; for input nodes, this should be much lower, intuitively because information is directly lost when input nodes are ignored.

At testing time after training has finished, we would ideally like to find a sample average of all possible {2^{n}} dropped-out networks; unfortunately this is unfeasible for large values of {n}.

However, we can find an approximation by using the full network with each node's output weighted by a factor of {p}, so the expected value of the output of any node is the same as in the training stages. This is the biggest contribution of the dropout method: although it effectively generates {2^{n}} neural nets, and as such allows for model combination, at test time only a single network needs to be tested. By avoiding training all nodes on all training data, dropout decreases overfitting in neural nets. The method also significantly improves the speed of training. This makes model combination practical, even for deep neural nets. The technique seems to reduce node interactions, leading them to learn more robust features that better generalize to new data.

3.2. Data Augmentation:

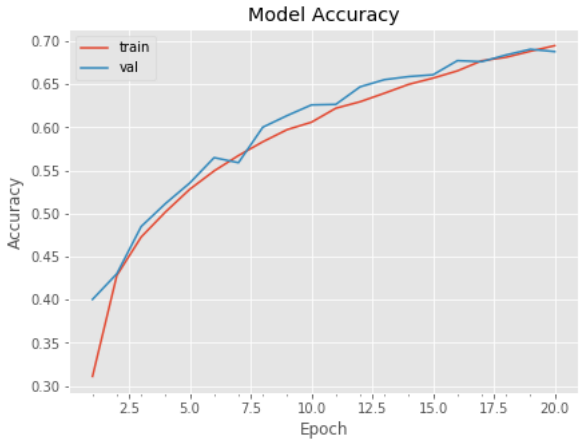
Data augmentation has long been used in practice when training convolutional neural networks. When training LeNet5 for optical character recognition, LeCun et al. apply various affine transforms, including horizontal and vertical translation, scaling, squeezing, and horizontal shearing to improve their model's accuracy and robustness. In, Bengio et al. demonstrate that deep architectures benefit much more from data augmentation than shallow architectures.

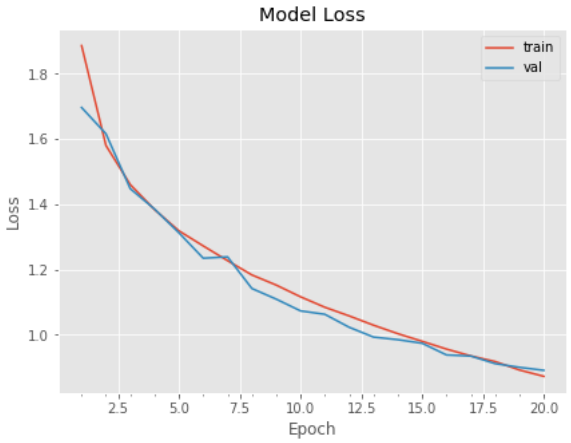
They apply a large variety of transformations to their handwritten character dataset, including local elastic deformation, motion blur, Gaussian smoothing, Gaussian noise, salt and pepper noise, pixel permutation, and adding fake scratches and other occlusions to the images, in addition to affine transformations.

4. Approach:

There were 2 major steps of my implementation:

1. My first CNN classification model was simple. This was my approach:
   1.1. Load the cifar10 dataset using keras.datasets.cifar10.load_data()
   1.2. Normalize the pixels by dividing each pixel by 255 and separate the training and test data.
   1.3. The CNN model had 3 hidden layers and were convolved with a Max pool kernel of 2*2 matrix. Each node had 32 images with width and height 3.
   1.4. The activity function used in the hidden layers was RELU.
   1.5. The output layer had 10 nodes. The loss function used was cross entropy and the optimizer was RMSPROP.
   1.6. When the model was run on the Test sample, with over 20 epochs, it got an accuracy of 0.68.





2. 2.1. I made the 2nd model the same way except I had 5 hidden layers, and added dropout probabilities of 0.25 for the first three hidden layers, then 0.5 on last two hidden layers. Activation function was RELU for all the layers except the output which was ADAM. While compiling, I put ADAM as my optimizer.
   2.2 Then also did Data Augmentation by using 'ImageDataGenerator' function which I imported from the Keras library.
   2.3 I trained the model for 20 Epochs as well.
   2.4 Accuracy was 87.94% which is great considering I didn't use the big sets like RESNET, IMAGENET and also it was just for 20 epochs.

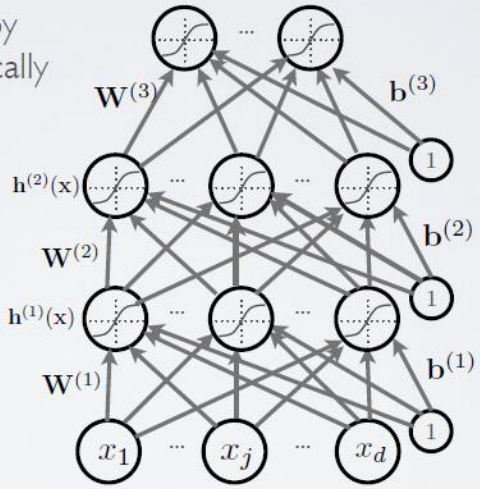
Also Data Augmentation plays a big role to get a good accuracy and avoiding over-fitting. This was done by shifting and rotating the images and fitting them back in the model.

Dropout Regularization:
This was added to get rid of over-fitting by randomly assigning probabilities to any nodes of each layer, with o.5 being the base. So any node below p=0.5 would have been eliminated, hence the weights have to be wisely distributed by the nodes, so the issue of over-fitting got eliminated.

DROPOUT

Topics: dropout

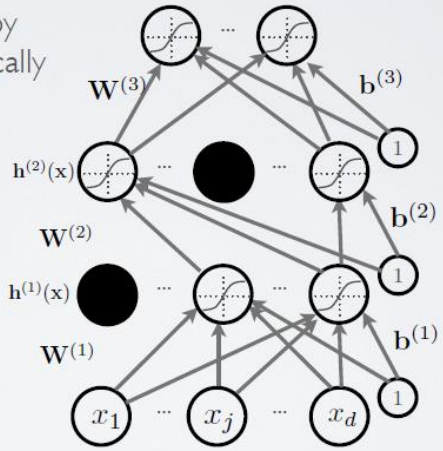
- Idea: «cripple» neural network by removing hidden units stochastically
  - each hidden unit is set to 0 with probability 0.5
  - hidden units cannot co-adapt to other units
  - hidden units must be more generally useful
- Could use a different dropout probability, but 0.5 usually works well



DROPOUT

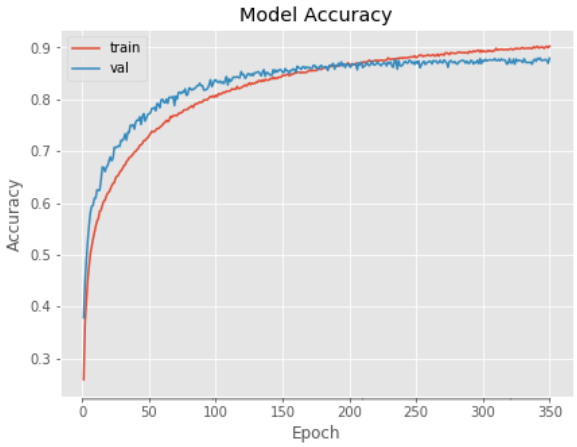
Topics: dropout

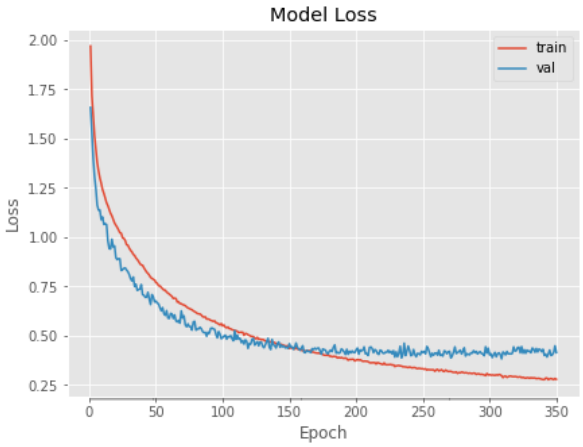
- Idea: «cripple» neural network by removing hidden units stochastically
  - each hidden unit is set to 0 with probability 0.5
  - hidden units cannot co-adapt to other units
  - hidden units must be more generally useful
- Could use a different dropout probability, but 0.5 usually works well



5. Result:

5.1. Accuracy and Loss of the second model. This model has an accuracy of 87.4%. No sign of overfitting in these curves.





5.2. However there were some images which couldn't be classified as the model is not perfect.

6. Conclusion:

The model performed better than expected mainly because we never used the deep nets like RESNET, IMAGENET also the number of epochs were less, still it at 87.4%, it performed much better than expected. Regularizers like Dropout helped the model not to overshoot and Data Augmentation helped with some good accuracy. More accuracy can only be reached with more epochs and better neural nets like RESNET, IMAGENET.
The concise takeaway from this experiment was that even without large dataset, and lower epochs, a model can give good accuracy if we know how to optimize the accuracy curves better.

7. Reference:

[1] Y. Bengio, A. Bergeron, N. Boulanger-Lewandowski, T. Breuel, Y. Chherawala, et al. Deep learners benefit more from out-of-distribution examples. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pages 164–172, 2011.

[2] A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. In IEEE International Symposium on Circuits & Systems, 2016.

[3] A. Coates, A. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pages 215–223, 2011.

[4] X. Gastaldi. Shake-shake regularization. arXiv preprint arXiv:1705.07485, 2017.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In European Conference on Computer Vision, pages 630–645. Springer, 2016.

[6] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580, 2012.

[7] https://github.com/fchollet/keras/blob/master/examples/cifar10_cnn.py.