Co-Occurrence Neural Network

Max Wasserman
University of Rochester, USA
mwasser6@ur.rochester.edu

Abstract

In computer vision applications, Convolutional Neural Networks (CNNs) are widely used due to their effectiveness across a wide range of tasks. CNNs rely on convolutions for their speed, ease of use and parameter sharing properties, but these operations are shift invariant and are thus limited in their ability to handle variation on the input.

To ameliorate this, [4] introduced the Co-Occurrence Neural Network. The method uses a non-shift invariant bilateral filter, using pixel locations (spatial information) as well as pixel intensities in the construction of the filter to be applied.

I attempt to replicate their results by implementing their method. I show that I can indeed replicate much of the reported results and go in-depth on the technical details of its implementation as well as critiques and suggestions for future research.

1. Introduction

This section will briefly review the problems Co-Occurrence Neural Networks attempt to solve, as well as motivation for the method.

1.1. Massive Networks

Part of what has makes Neural Networks so effective is their flexibility in architecture to adapt to the application. In computer vision, it is found that deep CNNs are very effective [3], and thus the field has seen an explosion in the the size of these networks. Top performing networks report sizes on the order 20-70 million parameters [1].

Some hypothesize that the reason CCNs need so many layers is because they rely on the convolution operation. The convolution operation has the property of shift invariance: the filter response is independent of where it is applied on the inputs. Due to this, one needs a massive number of filters to deal with different regions on the input [4] and pick out relevant features.

1.2. Attacks on Network Size

Modern methods to reduce network size are: parameter pruning and sharing, low-rank factorization, transferred/compact convolutional filters, and knowledge distillation [1]. Most of these techniques focus on reducing the number of parameters while still relying on the standard shift-invariant convolution operation as discussed above.

1.3. Co-Occurrence Matrix/Filter

A matrix that encodes the co-occurrence statistics of respective pixel intensities is referred to as a co-occurrence matrix. As an example, given a greyscale image, one can compute the (i,j)th entry of the co-occurrence matrix by summing over pixels with intensity i (denoted by I-i), then summing over the set of pixels with intensity j (denoted by I-j) and adding a term inversely proportional to their distance: $C[i,j] \propto \sum_{i \in I_i} \sum_{j \in I_j} 1/(dist(i,j))$

A co-occurrence filter (CoF) is a bilateral filter whose using a spatial filter and a co-occurrence matrix [4]. This filter has the property of smoothing textured regions and preserving boundaries. The co-occurrence term in this filter is unique in that is a function of the pixel values and not the spatial layout, leading to non-shift invariance. Note that the to compute the co-occurrence matrix, you must iterate over the input image, a computationally costly task.

1.4. Deep Co-Occurrence Matrix/Layer

[4] take this idea once step further by using the same bilateral filter structure as the CoF, *but instead of computing the entries of the co-occurrence matrix, they learn them using backpropagation*, thus motivating the name 'deep co-occurrence matrix'. This matrix is then used in the same way as in CoF, but now the filter is optimized with respect to the minimizing the loss function.

A co-occurrence layer (CoL) can replace the traditional convolution operation in mapping from input channels to output channels. It combines a deep co-occurrence matrix with a spatial filter, both learned via backpropagation, to map M input channels to M outputs channels. The exact procedure it uses to do this is described in later sections.

Note: Not all input channels are used in computing each output channel. Output channel i will only use input channels [i-dsf,i+dsf], where dsf := $\lfloor$(depth of spatial filter/2$\rfloor$. Thus the spatial filter fully defines the neighborhood used in the filtering operation.

1.5. CoL: Definition and Usage

Each pixel in each output channel is filtered using a CoF with the first term using the deep co-occurrence matrix and and the second term using the 3D spatial filter (with depth possibly equal to 1). The procedure to calculate the value of the filtered pixel is given in Figure 1. Further clarification is given in the methods sections.

If we construct all output channels using this procedure we call this a Co-Occurrence Layer (CoL). A CoL maps input channels to an equal number of output channels. Any place where a convolution is used to map from M channels to M channels, a CoL layer can be used to replace it.

CoL is differentiable and can be trained using backpropagation without any extra processing.

CoL uses a small number of parameters to generate a large number of filters as explained below. CoL can handle varying patterns in the input, as well as distributions (i.e., histograms) or re-arrangement of values, because of its co-occurrence term.

1.6. CoL Can Access a Large Number of Filters

After training a network with an embedded CoL there are k^{m*m*m} filters that can be used by the network (assuming k x k deep co-occurrence term and an m x m x m spatial filter) on the forward pass. This is compared to the single filter that is always used in the forward pass of a standard convolution operation. [4] hypothesize this is the reason CoL is able to achieve a reduction in the number of parameters in many architectures while maintaining accuracy.

2. Related Work & Method

Being that this work is mostly consists of confirming results from [4], the Related Work and Method section are highly related, and thus merged into one section. This section will be split into 2 parts: a short section describing the CoL method at an abstract level, and a section describing the implementation details of this method used by the author.

2.1. CoL Theory

The filtering procedure is defined by Figure 1. The value of output activation J_p is a weighted sum of the activations in N_p, the neighborhood of pixel p. The *Neighborhood is fully defined by the spatial filter*. One can visualize centering an m x m x m cube around input pixel p, as shown in Figure 2 . All pixels in the intersection of (1) the volume of the cube and (2) the input channels is included in the neighborhood of pixel p.

$$J_p = CoL(I, w, L([I_p], [I_q])) = \sum_{q \in N_p} w_q L([I_p], [I_q]) I_q$$

Figure 1. Filtered Output Activation J_p

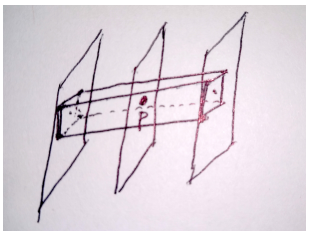

Figure 2. Neighborhood N_p

2.2. Methods: CoL Implementation

2.2.1 Normalizing Input Channels

CoL takes M input channels and outputs M output channels. CoL requires all inputs be in [0,1], due to the uniform binning procedure which follows. To achieve this one must (1) add the minimum value of the set of activations to each activation in this set, then (2) divide this set of activations by the maximum value (in this new shifted set).

The authors do not specify if the set of activations is all-channels or per-channel. A fast implementation of both is provided by leveraging broadcasting (formatting the computation such that PyTorch can call C subroutines to carry out computation).

2.2.2 Neighborhood Calculation

The neighborhood used in the filtering calculation is fully defined by the spatial filter. Centering the spatial filter over a pixel p and taking the intersection with the input channels gives the set of neighbors (Figure 2). The implementation of this is more complicated than it would first seem and is prone to error. This is due to (1) the cases where the pixel p lies on the border of the image (defined as the set of pixels where the support of the spatial filter extends past the support of the image), and (2) the channel of pixel p is in the beginning/end, and one must 'wrap around' to fetch the neighboring pixels.

The implementation of finding the neighboring pixels involves splitting the 2D image into 9 areas (Top & Left, Left, Bottom & Left, ... , Top & Right, Top, Middle). An input pixel p is then classified as being in one of these areas. Based on this classification, a set of 2D indices are returned. After determining which channels are to be used, channel indices are appended onto these 2D indices, thus producing the full set of neighboring pixels.

2.2.3 Speeding Up Neighborhood Calculation

The computation described above must be performed for each pixel, in each channel, in each image, in each batch. For a batch size of 10, channel size of 10x10, and 32 channels, that is O(10^5) computations on each forward pass. To reduce this, these neighborhood sets are precomputed and store in a lookup table. The index into the look up table is the pixels index (batch, channel, width, height) and the table returns a set of indeces which is the neighborhood of that pixel.

2.2.4 Transformation: Global Index to Local (relative) Index

In order to perform the filtering, one needs to transform the global indeces of the nieghbors into indeces which can be used to index into the local 3D spatial filter. Indeces of neighbors q are function of its relative to location with respect to p.

$q_{local\_row} = |p_{row} - q_{row} - borderSize|$
$q_{local\_col} = |p_{col} - q_{co}l - borderSize|$
$q_{depth} = |p_{channel} - q_{channel} + borderSize|$

Where q_{local_row} is the row index for p's neighbor q. p_{row} is the row index of p in the network. q_{row} is the row index of p's neighbor q in the network. borderSize is the $\lfloor$depth of the spatial filter/2$\rfloor$. The edge cases for first/last channels are not shown here.

2.2.5 Constructing a Filter

One can think of Figure 1 as the sum of the element wise product of three tensors as shown below in Figure 3

tensor 1: Input Activations
tensor 2: Deep Co-Occurrence's
tensor 3: Spatial Filter

The tensor created by the element-wise product of tensor 2 and 3 is the constructed filter applied to the input pixels. There are k^{m*m*m} such filters. To see this, consider that pixel activation I_p is fixed, and thus the index into the k x k Deep Co-Occurrence Matrix L is fixed. For each neighbor q there are k possible choices for its binned activation I_q. Thus the number of possible filters are $k*k*k.....*k = k^{size-of-neighborhood} = k^{size-of-spatialfilter}$

3. Experiments

The experiments are performed on a synthetic dataset. A network containing a CoL is compared against other network architechtures. The CoL network shows significatly faster convergence in loss as well as train and test accuracy.

3.1. Loss Function and Hyperparameters

The loss functions was copied from [4]'s implementation which was a Cross Entropy Loss, and the optimization

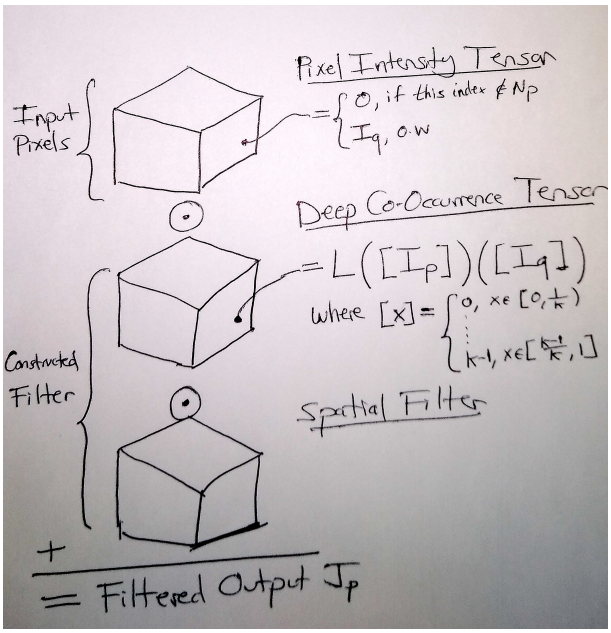

Figure 3. Network Constructs Filters

was performed using the Adam stochastic gradient descent package, with hyperparamers of learning rate =.0004, betas=(0.9, 0.999), eps=1e-08, weight decay=0.

[4] did not specify learning rate, batch size, betas, eps, or wieght decay parameters in their paper or released code (for this particular experiment). Differences in these parameters, as well as the particular implementations of these algorithms in Pytorch vs TensorFlow (the package used by [4]), account for much of the difference found in the shape of the plotted curves to follow.

3.2. Note: Bilateral Filtering Implementation

As discussed previously, the CoL layer is a bilateral filter. Performing filtering with such a bilateral filter, particularly one that is non-shift invariant, is very computationally expensive, and limited the author in the ability to fully replicate [4]'s results (due to limited time to implement the optimizations, and limited funds to train a large network from scratch).

[4] implements numerous optimizations in the implementation of CoL in order to speed up this computation. They cite [2] in their method.

3.3. Generation of Synthetic Dataset

From [4]:

> We illustrate the power of CoL on a toy example and compare its performance with two other popular layers: fully-connected and convolutional. We generated a synthetic set of 6000 training images and 1000 test images of size 10x10 and 4

pixel values that were sampled i.i.d from two different distributions (i.e., histograms). The pixel values of the first distribution came from the histogram [0.4, 0.1, 0.1, 0.4], and the pixel values of the second came from the histogram [0.1, 0.4, 0.4, 0.1]. The mean pixel value of both histograms is the same.

Each image was created using the Numpy scientific computing package. The dataset was created implementing PyTorch's VisionDataset interface. This allows for use of PyTorch's built in randomization/batching of the dataset.

3.4. Network Architectures Compared

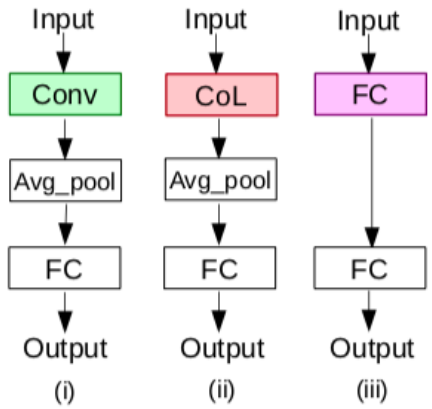

Figure 4. Network Architectures [4]

Five architectures were compared: 3 of type (i) - convolutional, 1 of type (ii) - fully connected, and 1 of type (iii) - CoL.

1: Conv(1,1,9)$\rightarrow avg(9,9) \rightarrow fc(36,2)$
2: Conv(3,3,2)$\rightarrow avg(7,7) \rightarrow fc(32,2)$
3. Conv(3,3,9)$\rightarrow avg(9,9) \rightarrow fc(36,2)$
4. FC(100,36) $\rightarrow fc(36,2)$
5. CoL(4,4)$\rightarrow avg(5,5) \rightarrow fc(36,2)$

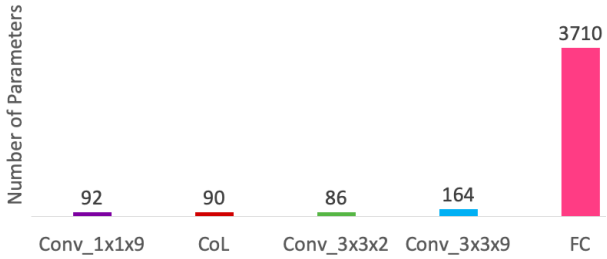

Figure 5. Size of Network Architectures

4. Results

4.1. Loss

The main result in this paper was the replication of the fast convergence in loss (number of epochs basis, not runtime).

Below is Figure 6, from [4] showing Loss vs Iteration Number, and a replicated plot generated by the author in Figure 7, showing Loss vs Epoch. [4] did not provide a specification of their batch size. Given batch size, Iteration Number is simply a multiple of Epochs.

The results are the same in both plots. The CoL network converges in Loss faster than all other architectures. The subsequent ordering, in terms of convergence rate, is Conv_3x3x9, Conv_3x3x2, Conv_1x1x9, and FC. This order is the same in both plots.

One can observe that the shapes of the the curves are slightly different. The main reasons for this is (1) differences in the hyperparameters used (see 3.1) (2) randomness associated with the generation of the synthetic dataset (see 3.3), (3) differing initilization/effects of randomness in parameter weights initialization, and (4) whether one chooses to report the 0th loss value as before/after the first epoch of training.

Regarding (3): [4] used a truncated normal (with resampling) which did not have an equivalent implementation in PyTorch. Randomness associated with the initizialization of network parameters can also influence starting loss values.

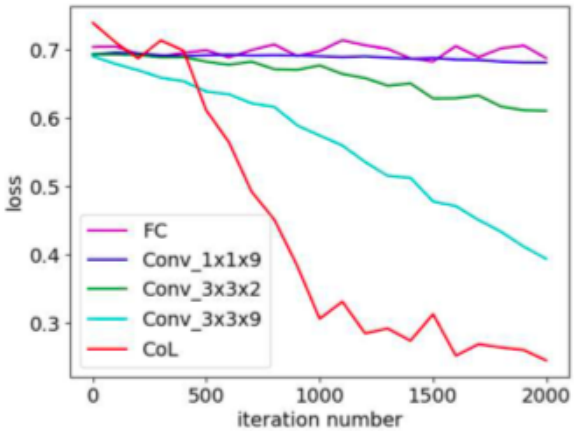

Figure 6. Loss vs Epochs: Original Result

4.2. Training/Test Accuracy

Plots from Training/Test Accuracy are included below. These follow much the same trend as loss, with CoL and the Conv networks network converging faster than the FC networks.

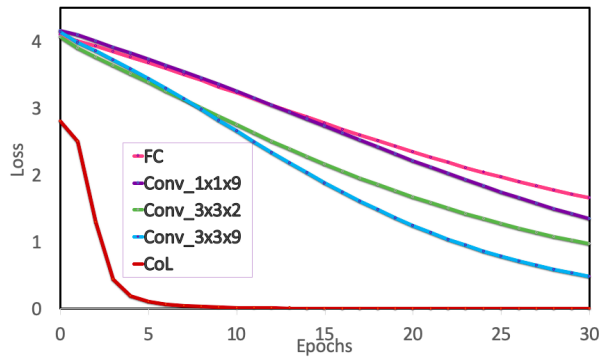

Figure 7. Loss vs Epochs: Replicated Result

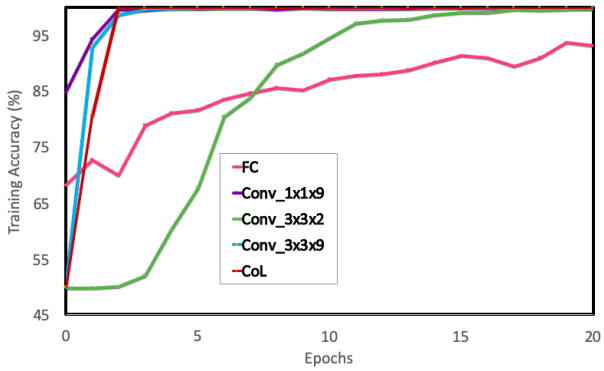

Figure 8. Training Accuracy

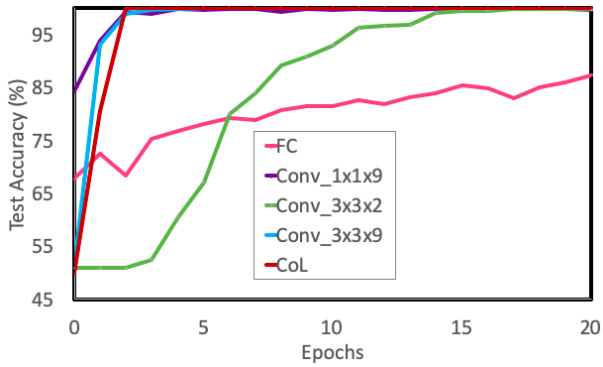

Figure 9. Test Accuracy

5. Future Work

(1) Filter Distribution Analysis on a Better Basis
(2) Binning Procedure Exploration
(3) CoL Optimizations

5.1. Filter Distribution Analysis on a Better Basis

The authors point out that when CoL layers are inserted into the ResNet model, they provide better performance when inserted deeper in the network. The authors suspect that when CoL is inserted deeper, the network learns to utilize a wider set of filters that CoL can represent. Recall that given a CoL with a kxk Deep Co-Occurrence Matrix and an $mxmxm$ spatial filter, k^{mxmxm} can be created. [4] hypothesize that if this was the case, this increased representation power would lead to improved performance.

> We suspect that at the deeper layers of a network the number of different filters captured by CoL is higher than in the early layers, and this might lead to increased representation power that leads to improved performance. To quantify this, we collected the filters used by CoL, clustered them into prototypes using k-means, and calculated a histogram of filters. That is, we count, for each prototype (i.e., cluster center) filter, how many similar filters were generated by the CoL layer.

The authors *do not* mention in which domain this clustering occurs. It is also not included in their released code on github. It is assumed that the constructed filter, shown in 3 is unrolled into a vector, and the clustering occurs directly in this domain.

This is a flawed technique, which may have resulted in flawed conclusions. K-means clustering performs euclidean distance calculations on the raw input vectors. ***In order for clustering results to be significant, close vectors in the input space should encode high similarity on some relevant objective/feature.***

In this case we are comparing filters. While the filters in CNNs are often more convienent to work with in their spatial representations, a filter's properties are more clearly viewed in the frequency domain. Below in Figure 10, I show a representative example of why *comparing filters in the spatial domain does not encode similarity for what we are interested in: the filter response.*

In Figure 10 the input filter are identical. Their filter response is the same. Yet taking a euclidean distance in the spatial domain would indicate that these filters are very different. A better domain (i.e. basis) is needed.

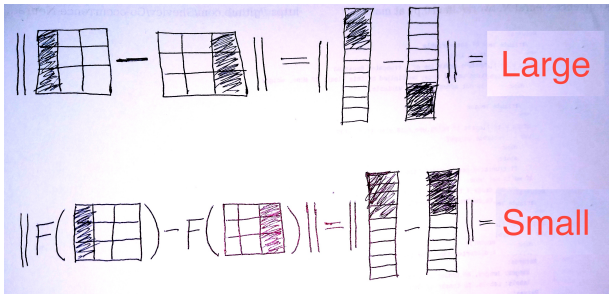

Figure 10. The Clustering Domain Matters: Identical Filters Should Have 0 Distance. F is some COB Transformation

I propose repeating this experiment, but before clustering is performed, transform the filters into a space which

encodes the similarity we are interested in. A good representative basis is the (1) 2D DFT basis, (2) a basis constructed using the eigenvectors of the graph of the filter support (a 2D/3D lattice), or (3) a learned basis using examples of similar vectors in the space with a cost function encoding desired distance outputs.

5.2. Binning Procedure Experimentation

The input to the CoL is first normalized, so all activations are between 0 and 1. The activations must then be binned for indexing the Deep Co-Occurrence matrix L. [4] uses uniform binning: the bins are static and split the activation interval [0,1] into equal sub-intervals. This binning procedure may have detrimental effects if the distribution of input activations follow a highly non-uniform distribution like the one presented in Figure 11. This distribution would under utilize many of the parameters in the Deep Co-Occurrence matrix, thus greatly reducing the representational power of the CoL.

To fix this problem, there are a few potential solutions.

(1) Make bin cutoffs differentiable parameters with a strong prior over the equal sub-interval value (= bin number/number of bins) enforced with some regularization in the loss function.

(2) Make bin cutoffs hyperparmeters of the model and retrain on these different values.

(3) Make the bin cutoffs a function of the input activation distribution, enforcing some uniformity over the created bins, and thus forcing equal use of all parameters in the Deep Co-Occurrence matrix.

Ultimately the experiment will have to be run to explore the effectiveness of these methods.

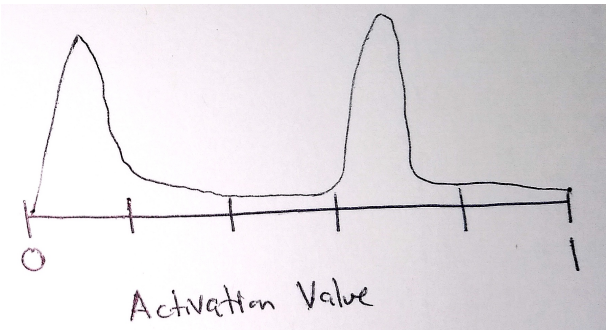

Figure 11. A Bad Activation Distribution for Uniform Binning

5.3. CoL Optimizations

The author was not able to replicate the ResNet results because of the time and cost of training such a model. [4] implemented an optimization to help overcome this which would slightly reduce such barriers to replication.

6. Conclusions

I have replicated the fundamental results of the Co-Occurrence Neural Network method introduced by [4], while uncovering some heavy computational costs of the method. I have introducing some corresponding techniques to deal with these costs, like caching neighborhood calculations and leveraging broadcasting for normalization operations. The main result which remains to be confirmed is the ResNet compression, and maintenance of accuracy, when CoL is inserted in the deepest conv layer. This was due to lack of time to optimizations in CoL implementation given by [2], as well as lack of resources (funds) to train such a model.

Furthermore, a fundamental flaw in [4]'s analysis was uncovered (section 5.1), motivating an exploration of domains better suited to analyze the similarity of filters with respect to their filter response. Other avenues for further research were proposed in detail.

References

[1] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[2] Michaël Gharbi, Jiawen Chen, Jonathan T Barron, Samuel W Hasinoff, and Frédo Durand. Deep bilateral learning for real-time image enhancement. *ACM Transactions on Graphics (TOG)*, 36(4):118, 2017.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[4] Irina Shevlev and Shai Avidan. Co-occurrence neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4797–4804, 2019.