Technion – Israel Institute of Technology

Electrical Engineering Department

Laboratory of Control, Robotics and Machine Learning

CNN Spatial Optimizations

Winter Semester of 2019

Presenters: Ido Imanuel and Inna Batenkov

Supervisor: Gil Shomron

Table of Contents

1. Abstract

Convolutional neural networks (CNNs) compute their output using weighted-sums of adjacent input elements. This method enables CNNs to achieve state-of-the-art results in a wide range of applications such as computer vision and speech recognition. However, it also comes with the cost of high computational intensity. Shomron et al [1] purposed exploiting the spatial correlation inherent in CNNs and predict activation values, thus reducing the needed computations in the network. They introduced a heuristic that predicts which activations are zero-valued according to nearby activation values, in a scheme they call cross-neuron prediction.

In this work, we further generalize the work done by Shomron et al, with the following steps:

1. Implement a custom CNN layer which we dub a "Spatial" layer, allowing for a fast and efficient prediction statistics collection on any chosen CNN architecture.
2. Create a hierarchal framework that allows for the cross-neuron predictions on any generic CNN architecture and with any generic dataset chosen.
3. Implement the class structure into a few chosen networks: AlexNet, ResNet18, ResNet34
4. Implement a greedy optimization algorithm to choose the best Mask Configuration, allowing for a maximal number of saved MAC operations, under a certain loss in accuracy ε and based on changing optimization granularity.
   Algorithm was implemented in 4 modes: Max Granularity, Uniform Layer, Uniform Filter and Uniform Patch.
5. Test framework on the chosen networks with varying datasets

Steps will be detailed in Sections 4 & 5 with the results displayed on Section 6.

1.1 Achievements

To be filled in with Results

2. Introduction

In this section we will briefly introduce the various topics this project is about.

2.1 Convolution Neural Networks - Basics [2]

2.1.1 Overview

Convolutional Neural Network is a class of deep neural network that is used for Computer Vision or analyzing visual imagery.

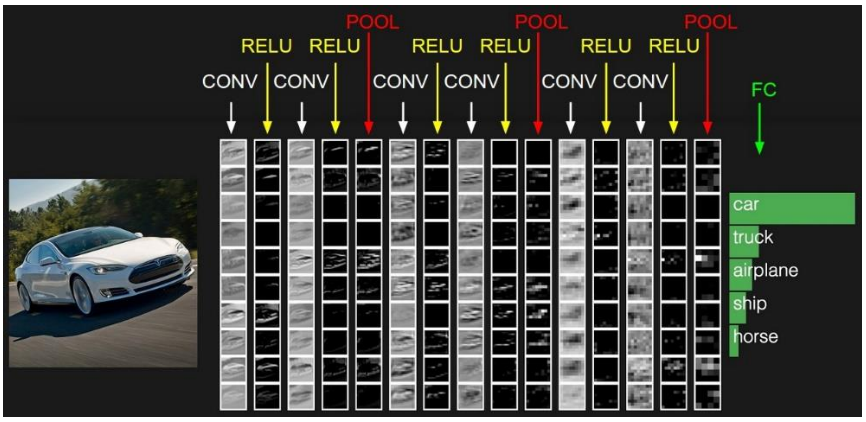


Figure 1: A basic overview of a simple CNN used for vehicle classification, taken from [2]

2.1.2 Convolutional Layer

Computers read images as pixels and it is expressed as matrix (NxNx3)—(height by width by depth). Images makes use of three channels (rgb), so that is why we have a depth of 3.

The Convolutional Layer makes use of a set of learnable filters. A filter is used to detect the presence of specific features or patterns present in the original image (input). It is usually expressed as a matrix (MxMx3), with a smaller dimension but the same depth as the input file.

This filter is convolved (slided) across the width and height of the input file, and a dot product is computed to give an activation map.

Different filters which detect different features are convolved on the input file and a set of activation maps is outputted which is passed to the next layer in the CNN.

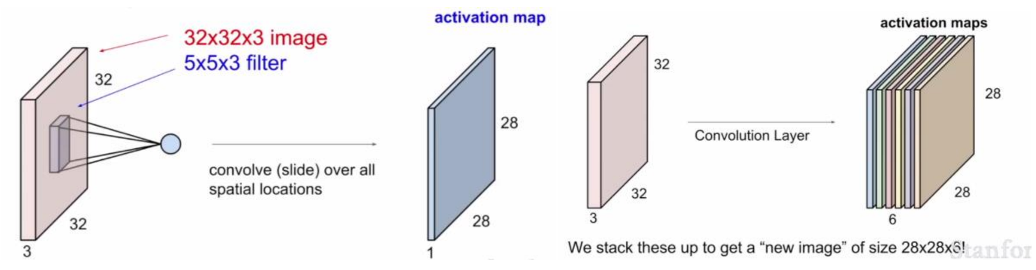
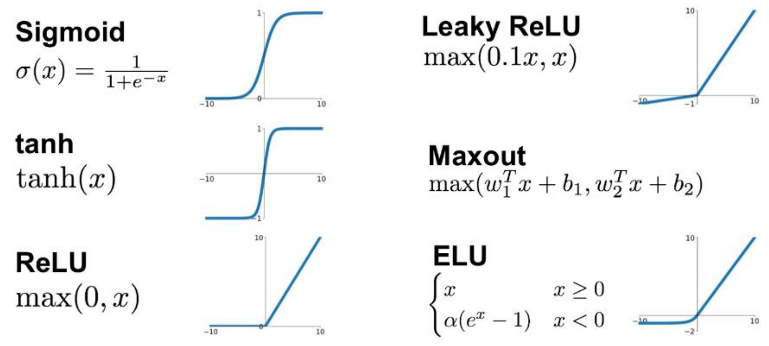

32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all spatial locations
activation map
28
28
1
32
32
3
Convolution Layer
activation maps
28
28
6
We stack these up to get a "new image" of size 28x28x6!

Figure 2: A basic view of input and output of a CNN layer, from [3]

2.1.3 Activation Function

Activation function is a node that is put at the end of or in between Neural Networks. They help to decide if the neuron would fire or not.

“The activation function is the non linear transformation that we do over the input signal. This transformed output is then sent to the next layer of neurons as input.” — Analytics Vidhya



ReLU function is the most widely used activation function in neural networks today. One of the greatest advantage ReLU has over other activation functions is that it does not activate all neurons at the same time. From the image for ReLU function above, we’ll notice that it converts all negative inputs to zero and the neuron does not get activated. This makes it very computational efficient as few neurons are activated per time. It does not saturate at the positive region. In practice, ReLU converges six times faster than tanh and sigmoid activation functions.

2.1.4 Fully Connected Layer (Dense layers)

In this layer, the neurons have complete connection to all the activations from the previous layers. Their activations can hence be computed with a matrix multiplication followed by a bias offset. This is the last phase for a CNN network.

The final output of the the last dense layer is inputted into a k-Softmax probability function, allowing for a classification into k classes.

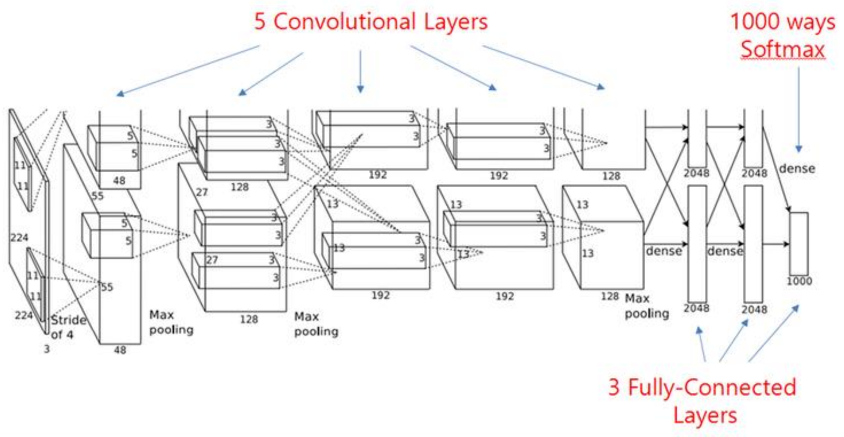


Figure 3: The AlexNet architecture, as shown in [2]

2.2 Convolution Neural Networks – Power Considerations

2.2.1 Field of View

Let us consider a simply CNN architecture for classifying digits, for example, on the well-known MNIST dataset. The propagation of data to a specific neuron in a given convolution layer is reliant only on some of the input pixels, but not all of them. Only the pixels that are involved in the actual convolution operation of that neuron are relevant – these stem directly from the size of the filter used. This result is demonstrated in the following [4], allowing for clear visualization of the data propagation.

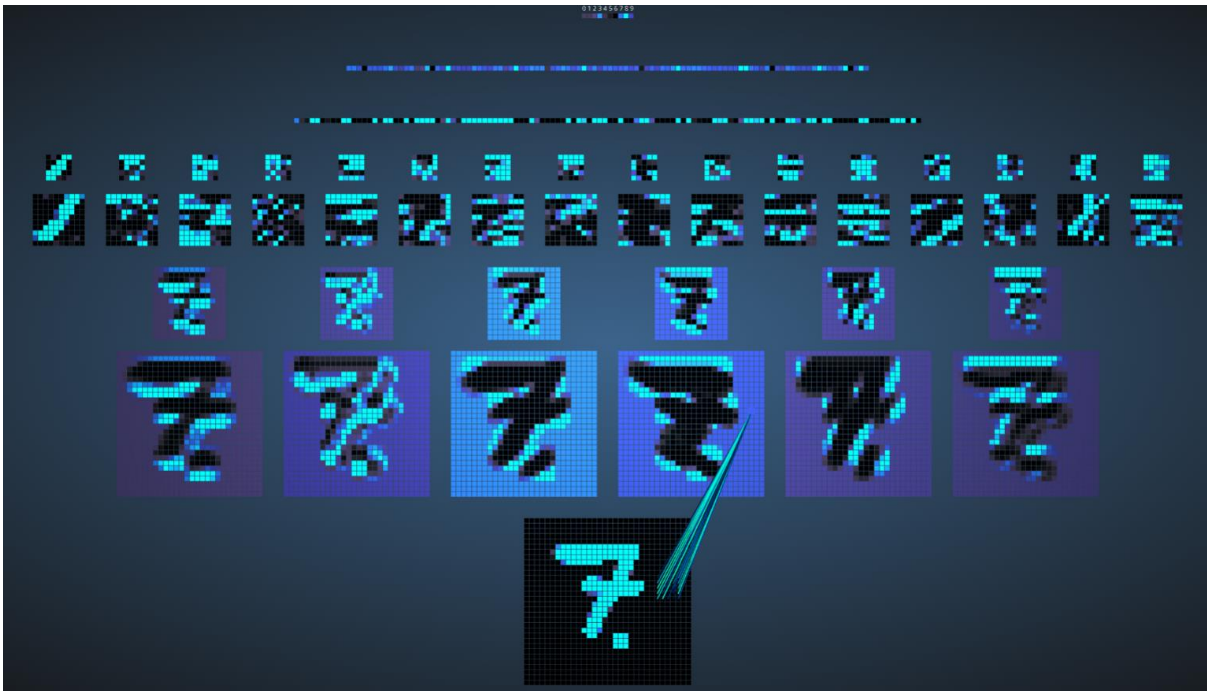


Figure 4: CNN output filter maps visualization on MNIST trained CNN classifier for the digit 7 [3]

2.2.2 MAC Operations and Relu considerations

Consider a given filter i of layer j. Ignoring the bias term, the output of the neuron is calculated by:

$$0+\sum_{k\in\text{Field of View}} x_k w_k^{ij}$$

For a filter of size 3x3 for example, we would calculate $3\cdot 3=9$ mult operations, and sum $3\cdot 3-1\ +1=9$ times, thus resulting in 9 MAC (Multiply and Accumulate operations). As with any arithmetic computations, these cost us power. Notice that when this sum is negative, a Relu activation on that layer truncates the sum to zero.

3. Project Specification

In this section we will briefly motivate this project and explain its targets.

3.1 Problem Description

Convolution neural networks have become a state-of-the-art approach to solving image related tasks. However, they suffer from being highly computationally intensive [5].

Predicting zero-valued activations has already been proven effective to reduce some of the computations [1] [5]. In this project we will optimize the prediction pattern to achieve the highest performance possible within a model's accuracy constraint.

Project Targets:

1. Learn how CNNs work
2. Learn how cross neuron activations work prediction method works
3. Learn how Aklaghi et al. [5] optimized their solution
4. Optimize the mask patterns used in [1] to be more efficient, utilizing the optimization ideas from [5]
5. Evaluate prediction performance and model accuracy

3.2 Past Work Overview

We review the two main articles this project is based on.

3.2.1 SnaPEA [5]

In [5], Aklaghi et al proposed SnaPEA, a system for reducing compute-heavy convolution operations, exploiting the fact that the common Relu activation layers truncate any results below zero. The offered two modes of optimization:

1. Exact mode: Assuring no loss in classification accuracy. This includes adding additional logic on top of CNN layers, which statically re-orders the weights based on their sign. Upon a convolution with said layer, the logic periodically performs a single-bit sign check on the partial sum. Once the partial sum sums over the negative values and drops below zero, the rest of the computations can simply be ignored, since the output value will be zero in any case.
2. Predictive mode: Trades classification accuracy for larger savings. Speculatively cuts the computation short even earlier than exact mode. To control the accuracy, they develop a multi-variable optimization algorithm that thresholds the degree of speculation.

Results:

- SnaPEA in the exact mode, yields, on average 28% speedup and 16% energy reduction in various modern CNNs without affecting their classification accuracy.
- With 3% loss in classification accuracy, on average, 67.8% of the convolutional layers can operate in the predictive mode. The average speedup and energy saving of these layers are 2.02× and 1.89×, respectively. The benefits grow to a maximum of 3.59× speedup and 3.14× energy reduction.

3.2.2 Cross-Neuron Predictions [5]

To ease the compute intensity of CNNs, Shomron et al applies a common technique applied in GPPs – value prediction. They purpose a value prediction method which exploits the spatial correlation of activations inherent in CNNs. The arguments is as follows: neighboring activations in the CNN output feature maps (ofmaps) share close values (illustrated in the figure below).

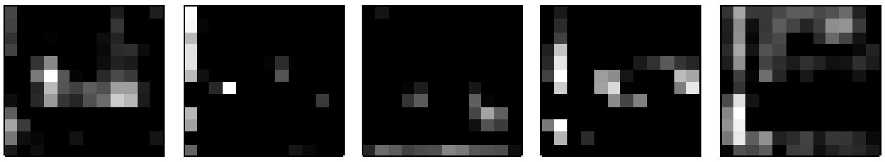
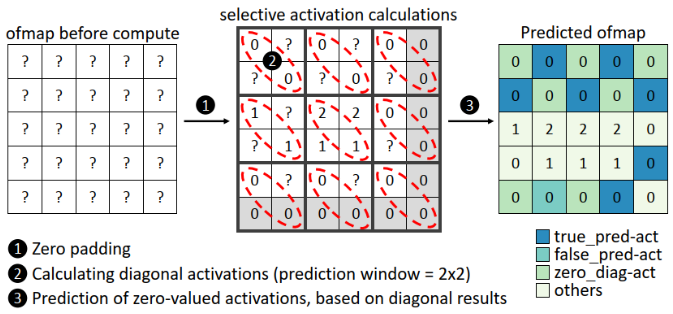


Figure 5: Taken from [1], examples of AlexNet CONV3 ofmaps (after Relu) for five arbitrary filter channels. Spatial correlation is evident. Black pixels are zero.

Therefore, some activation values may be predicted according to the values of adjacent activations. By predicting an ofmap activation, an entire convolution operation between the input feature map (ifmap) and the kernel maybe saved.

Method:

They propose a prediction method by which zero-valued activations are predicted according to nearby zero-valued activations. First, ofmaps are divided into square, non-overlapping prediction patches. The ofmaps are padded with zeros so predictions can also be made in the presence of margins. Next, the activations positioned diagonally in each patch are calculated. If these activations are zero-valued, the remaining activations within that patch are predicted to be zero-valued as well, thereby saving their MAC operations. The figure below depicts this method, with 2x2 patches.



ofmap before compute
selective activation calculations
Predicted ofmap
true_pred-act
false_pred-act
zero_diag-act
others
1 Zero padding
2 Calculating diagonal activations (prediction window = 2x2)
3 Prediction of zero-valued activations, based on diagonal results

Figure 6: Taken from [1], Illustration of the prediction method

Results:
2x2 Patch:
- 34.8% MAC savings under a Top-1 degradation accuracy of 1.9% on the AlexNet network.
- 20.8% MAC savings under a Top-1 degradation accuracy of 1.9% on the ResNet18 network.

3x3 Patch:
- 40.8% MAC savings under a Top-1 degradation accuracy of 4% on the AlexNet network.
- 23.5% MAC savings under a Top-1 degradation accuracy of 17.6% on the ResNet18 network.

4. Architecture

In this section we will review the architecture proposed in this project, and explain its function.

4.1 Spatial Layer

We build upon the work done by Shomron et al [1], and purpose a generalization of its idea. We propose a custom CNN layer which we dub a "Spatial" layer which computes the cross-neuron prediction for any given prediction mask. We recognize the fact that the flat prediction mask (pmask) used by Shomron et al may not be optimal, and other, more varied masks may achieve superior results, while supplying lower accuracy degradation. In general, for a patch of size pxp, may receive as many as $\Omega_p^* \triangleq 2^{p \cdot p} - 2$ possible singleton masks (relevant for a single patch in the ofmap), removing the degenerate all 1s or all 0s mask.

$$
\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix} \Rightarrow
\begin{array}{cccc}
\begin{bmatrix}1 & 0\\0 & 0\end{bmatrix} & \begin{bmatrix}0 & 1\\0 & 0\end{bmatrix} & \begin{bmatrix}0 & 0\\1 & 0\end{bmatrix} & \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix} \\
 & \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix} & \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix} & \\
\begin{bmatrix}1 & 0\\1 & 0\end{bmatrix} & \begin{bmatrix}0 & 1\\0 & 1\end{bmatrix} & \begin{bmatrix}0 & 0\\1 & 1\end{bmatrix} & \begin{bmatrix}1 & 1\\0 & 0\end{bmatrix} \\
\begin{bmatrix}1 & 0\\1 & 1\end{bmatrix} & \begin{bmatrix}0 & 1\\1 & 1\end{bmatrix} & \begin{bmatrix}1 & 1\\1 & 1\end{bmatrix} & \begin{bmatrix}1 & 1\\0 & 1\end{bmatrix}
\end{array}
$$

Figure 7: The 2x2 prediction mask used by Shomron et a(left)l, versus all possible masks(right). The degenerate all ones or all zeros masks were removed, seeing they define deterministic and degenerate predictions

For a given mask, the activations positioned with "1" in each patch are calculated. If these activations are zero-valued, the remaining activations within that patch are predicted to be zero-valued as well, thereby saving their MAC operations.

This custom layer was implemented with Python utilizing the well-known PyTorch module, and runs efficiently with a GPU compatible implementation. The layer receives a given full pmask (combined from singleton patch-size pmasks, targeted to be in the shape of the ofmap), and collects statistics such as the number of MAC operations saved via the supplied pmask.

4.2 Spatial Network Framework

Recognizing the difference between different network architectures, such as VGG16 or AlexNet, we model a new class dubbed "SpatialNet". This class inherits from the PyTorch neural network class, and implements generic functions that deal with the control and statistical analysis of all Spatial layers placed inside the network's architecture definition. This allows an easy and simple way to implement Spatial layers into any desired CNN. We implement the framework on the AlexNet, ResNet18 and ResNet34 classes.

5. Optimization

In this section will detail the optimization algorithm and its various control parameters

5.1 Overview

The main idea of this project is as follows:

Given a desired SpatialNet classification network and chosen dataset, along with an allowed parameter ε detailing the maximal accuracy loss allowed, find the optimal set of prediction masks, allowing for the maximal amount of MAC operations saved, with under ε classification accuracy loss.

The steps are as follows:

1. Insert Spatial layers after any chosen Convolution layers with Relu activations
2. Vanilla train the network on the dataset with the Spatial layers disabled. Save baseline 0-1 classification accuracy.
3. Configure the chosen optimization hyper-parameters, choosing the desired mode of optimization.
4. Using the trained model, run the optimization algorithm, looking for the best set of full prediction masks with under ε classification accuracy loss from the baseline accuracy. We note that for every Spatial layer in the architecture, exists an optimal prediction mask.

5.2 Optimization Algorithm

5.2.1 Combinatorial Troubles

We begin with a worrying problem – the combinatorial explosion expected when looking for the optimal mask. We note the patch size hyper-parameter that controls the number of combinatorial options to choose from (detailed in section 5.2.4.1). Notice that while taking into account all valid mask options, the growth in the options for a singleton pmask options is exponential:

$$\Omega_p^* \triangleq 2^{p^2} - 2 = \begin{cases} 2^4 - 2 & p = 2 \\ 2^9 - 2 & p = 3 \\ 2^{16} - 2 & p = 4 \end{cases} = \begin{cases} 14 & p = 2 \\ 510 & p = 3 \\ 65534 & p = 4 \end{cases}$$

This is only the beginning of the problem, seeing that multiple singleton pmasks are to be inserted into the creation of a full pmask, relevant for a single Spatial layer. We explain the combinatorial calculation with an example:

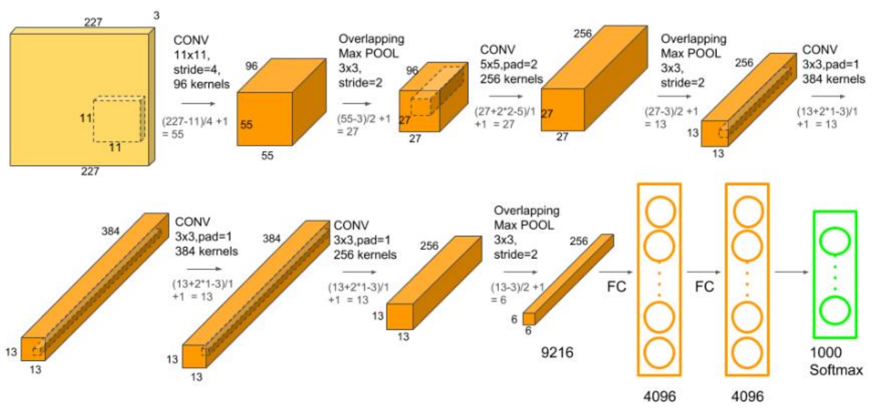

227
3
11
11
227
CONV 11x11, stride=4, 96 kernels
(227-11)/4 +1 = 55
96
55
55
Overlapping Max POOL 3x3, stride=2
(55-3)/2 +1 = 27
96
27
27
CONV 5x5,pad=2 256 kernels
(27+2*2-5)/1 +1 = 27
256
27
27
Overlapping Max POOL 3x3, stride=2
(27-3)/2 +1 = 13
256
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 256 kernels
(13+2*1-3)/1 +1 = 13
256
13
13
Overlapping Max POOL 3x3, stride=2
(13-3)/2 +1 = 6
256
6
6
9216
FC
FC
4096
4096
1000 Softmax

Figure 8: AlexNet ofmaps for input of size 227x227 as detailed on [6]

Given an input of size 227x227, the AlexNet architecture has 96 kernels/filters of size 11x11. This creates a 96x55x55 ofmap, which will then be inserted into the Spatial layer. If we choose p=2, we need to split this map into 2x2 blocks. We therefore pad the map with a section of zeros, resulting in an ofmap of size 96x56x56. We can now insert:

$$N = \frac{96 \cdot 56 \cdot 56}{2 \cdot 2} = 75,264$$

N different patches. We now have the task of choosing which of the $\Omega_{p=2}^* = 14$ options for singleton masks is best for each patch, resulting in:

$$\Omega = 14^{75264}$$

Ω options for a single full pmask, and this is only for a single Spatial layer.

Note that for the AlexNet architecture, as detailed [6], if we place a Spatial layer after each convlution layer, a total of

$$N = \frac{96\cdot56\cdot56}{2\cdot2} + \frac{96\cdot28\cdot28}{2\cdot2} + \frac{256\cdot28\cdot28}{2\cdot2} + \ 2\cdot384 + 2\cdot256 \ \frac{14\cdot14}{2\cdot2} = 206976$$

N different slots for singleton pmask placement are possible, creating $\Omega = 14^{206976}$ different options.

If we optimize by this granularity utilizing an exact exhaustive search, with each forward run needed to compute the number of operations saved and loss in accuracy, taking 1msec, it would many more years than the number of atoms in the known universe to finish the computation.

5.2.2 Greedy Optimization

The problem of optimizing over the mask set is therefore most likely NP-Hard, and exponential in the required runtime complexity required to choose the best possible mask configuration for the entire network, thus unfeasible computationally.

We borrow ideas from [5] and design a multilevel greedy optimization algorithm, running in 4 possible modes, as follows:

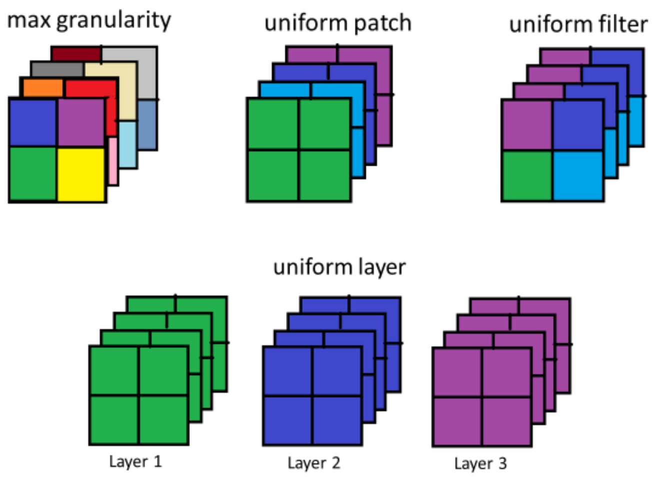


Figure 9: Implemented optimization modes, detailed with a rough sketch. Each square is a kernel ofmap mask, and each set of squares is a full ofmap for a given Spatial layer. Different colors relate to different singleton pmasks

5.2.2.1 *Optimization mode sketch*

We roughly describe the optimization algorithm behind each mode. For formal pseudo-code, please see Appendix 1.

For each mode of operation, we define its optimization target as the following:

- Max Granularity: Targets the patch level
- Uniform Patch: Targets the filter level
- Uniform Layer and Uniform Filter: Targets the Spatial layer level

The key presumption behind the algorithm is that targets are uncorrelated. Given N targets, optimizing jointly for all targets is equivalent to N sole optimization tasks of each. This presumption is clearly not withheld, seeing ofmaps from deep layers are derived directly from ofmaps of shallower layers in the architecture, but it allows us to deal the combinatorial explosion, making the optimization feasible. We roughly describe each optimization mode:

5.2.2.2 *Exact solution under the Uniform Layer mode*

We would like to work a higher granularity, reducing combinatorial troubles. For the Uniform Layer setup, we define the following scheme:

Given a set of singleton pmasks Ω, construct a uniform full pmask, where all patches originate from the same singleton pmask. This means that for a given layer, there are Ω_p^* options to test. The number of combinatorial options to test is therefore:

$$\Omega = L^{\Omega_p^*}$$

Where L is the number of Spatial layers in the network architecture. For L=5 and p=2 for example, we receive:

$$\Omega = 5^{14} \approx 6.1 \cdot 10^9$$

This is the only mode where an exact solution may be computed, under the assumption of a very small L and p.

5.2.2.3 *Greedy algorithm under the Uniform Layer mode*

For each layer, set one of the Ω_p^* options, and run a full classification forward run through the system. Compute the number of operations saved, along with the relevant 0-1 classification accuracy.

Collect all results, resulting in an array of: $L \times \Omega_p^*$ where each cell holds a 0-1 accuracy and operations saved. Choose the best option from all layers, with respect to op savings, and define it as the global pmask for the network.

Under the no correlation assumption, this is assured the best configuration. We have a possible problem of possibly not withholding the accuracy ε requested – this is handled by an iterative scheme to choose less accuracy-expensive pmask options, at the cost of lowering our op savings, and is described in Appendix 1.

Note that the number of forward runs needed is a mere: $\Omega = L \cdot \Omega_p^*$, which is vastly less than the numbers shown above, but also means that we are strictly looking at a much small subspace of possible options, most likely resulting in a sub-optimal solution.

5.2.2.4 *Greedy algorithm under the Uniform Patch mode*

This mode merely adds more options to the possible layer configurations tested in the Uniform Layer mode. Instead of choose a single singleton pmask for the entire full pmask, it looks at all options possible for a single kernel ofmap, and duplicates to all other kernel ofmaps from the same layer.

5.2.2.5 *Greedy algorithm under the Uniform Patch mode and the Max Granularity mode*

These copy the scheme of the Uniform layer, while reducing the target granularity. We now test at the kernel level (Uniform Patch) or even the patch level (Max Granularity), instead of the layer level, increasing the number of options to choose from.

The kernel level optimization looks for the best mask configuration for a specific layer. We keep the top M options (M is a hyper-parameter) and continue with a full Uniform Layer optimization, with the M options from each layer.

The patch level looks for the best mask configuration for a specific kernel. We keep the top M options, and continue with a full Uniform Patch optimization (and afterwards, a full Uniform Layer optimization).

5.2.3 Hyper-parameters

We allow for a full configuration of many hyper-parameters, allowing for greater freedom of optimization and analysis.

5.2.3.1 *Patch Size (p parameter)*

The singleton size of the estimator block. In the baseline case (Diagonal in [1]) for a value of p=2 we would receive the singleton pmask:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

For a patch size of 3 in the baseline case, this would amount to:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We note that increasing the prediction window size presents a tradeoff. On the one hand, a large window size increases the number of activations that may be predicted per window, thus allowing for increased operations saved.On the other hand, spatial correlation diminishes as the size of the prediction patch increases, which: (1) decreases the number of prediction patches inside the entire layer pmask; and (2) increases the false prediction rate, since the area around the with non-1s possible increases. These false predictions will decrease the model accuracy.

5.2.3.2 *Ones Range (ones_range parameter)*

This parameter controls the number of 1s we would like to see in the singleton pmask, and is supplied as a range between $1: p^2 - 1$.

We note that there is an inherent tradeoff between different masks – The more “0s” supplied with the mask – the higher the chance to save MAC operations, and the more operations that are saved. With this, the higher the chance to predict wrongly, thus causing possible classification accuracy degradation.

5.2.3.3 *Granularity Threshold Parameter*

We note that, for example, on the Max Granularity mode, we test the effect of a small p x p patch on the accuracy of the entire network. Seeing this is a tiny change when compared to the entirety of the network, we look for some way to strengthen its effect. We increase the spread of each target checked to be something more substantial, by increasing its area of effect. For a given target pmask choice, we define the GT parameter. This parameter clusters close by patches and defines them as a single target for optimization. When GT is highest, the whole kernel would be a single patch, and we receive the Uniform Patch mode. When lowest, we receive the true Max Granularity mode defined above.

5.2.3.4 *Network Choice*

As noted in [1], some network architectures supply sparser ofmaps than others. We tested 2 different networks – ResNet18 [7] and AlexNet [8]. The number of Spatial layers themselves were changed – and a coupled set of layers were introduced in the ResNet architecture. This means that after every convolution layer a Spatial layer was introduced, but some of the layers were optimized jointly, relaxing the combinatorial optimization troubles, while allowing op savings after every convolution layer.

5.2.3.5 *Dataset Choice*

We implemented many different dataset choices into the framework, including ImgNet, TinyImgNet, STL10, MNIST, CIFAR10, and FashionMNIST. Dataset choice is a primary factor when looking at computation runtime, where ImgNet is over 1,000,000 photos large, RGB with an average size of 256x256, and CIFAR10 only 60,000 photos of size 32x32, grayscale. This decides the forward runtime of the network, F. Seeing these forward runs are the most substantial part of the algorithm, we can deduce that the total runtime of different optimization modes may be approximated to be: $T \approx \Omega \cdot F$ where Ω is the total number of options checked at each granularity mode.

6. Results

In this section we describe various tests, and display some of the resulting prediction masks.

7. Further Work

8. Bibliography

[1] G. Shomron and U. Weiser, "Exploiting Spatial Correlation in Convolutional Neural".

[2] Medium, "CNN Overview," [Online]. Available: https://medium.com/@udemeudofia01/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17.

[3] S. CS231, "Training Neural Networks - Part 1," [Online]. Available: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf.

[4] A. W. Harley, "An Interactive Node-Link Visualization," [Online]. Available: http://scs.ryerson.ca/~aharley/vis/conv/flat.html.

[5] V. Akhlaghi and A. Yazdanbakhsh, "SnaPEA: Predictive Early Activation for Reducing Computation in Deep Convolutional Neural Networks".

[6] S. Nayak, "LearnopenCV," [Online]. Available: https://www.learnopencv.com/understanding-alexnet/.

[7] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2015.

[8] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks".

9. Appendix 1: Optimization Algorithm Psuedo Code

Inputs:	CNN	– a CNN model
	$\mathcal{D}$	– an optimization dataset
	ϵ	– maximal acceptable loss in classification accuracy
	mode	– mode of optimization, i.e. max granularity, uniform layer, uniform patch or uniform filters
	ps	– patch size
	patterns	– all possible patterns. Patterns can be selected directly or by specifying a range of number of '1's in the patch
	gran	– granularity
outputs:	mask	– final mask
	LQrec	– LayerQuantizier stage results
	CQrec	– ChannelQuantizier stage results
	PQrec	– PatchQuantizier stage results
	FLrec	– first level results

```
initialize Optimizer(CNN, $\mathcal{D}$, $\epsilon$, mode, ps, patterns, gran)
FLrec = Optimizer.FirstLevelOptimizationPass()
PQrec = Optimizer. PatchQuantizierPass(FLrec)
CQrec = Optimizer. ChannelQuantizierPass(PQrec)
LQrec = Optimizer. LayerQuantizierPass(CQrec)
mask = FindFinalMask(LQrec, $\epsilon$)
```

```
function FirstLevelOptimizationPass(Optimizer)
    initialize FLrec[layer][channel][patch][pattern] → None
    for layer $\ell$ in CNN do
        for channel $c$ in $\ell$ do
            for patch $p$ in $c$ do
                for $patt$ in patterns do
                    tmp_mask = CreateFirstLvlMask(mode, $\ell, c, p, patt, gran$)
                    ops_saved, err = Simulate(CNN, $\mathcal{D}$, tmp_mask)
                    FLrec[$\ell$][ $c$][$p$][$patt$] = (ops_saved, err)
    return FLrec
```

```
function PatchQuantizierPass(Optimizer , FLrec)
    for record in FLrec do
        if err $>$ $\epsilon$ then
            remove record
    sort FLrec based on ops_saved
    initialize PQrec[layer][channel][pattern] $\rightarrow$ None
    for layer $\ell$ in CNN do
        for channel $c$ in $\ell$ do
            for $i$ in range(0, #patterns) do
                channel_pattern = CreateChannelPattern(mode, $\ell, c$, FLrec$[\ell][c][:][i]$)
                ops_saved, err = Simulate(CNN, $\mathcal{D}$, channel_pattern)
                PQrec $[\ell][c]$[channel_pattern] = (ops_saved, err)
    return PQrec
```

```
function ChannelQuantizierPass(Optimizer , PQrec)
    for record in PQrec do
        if err $>$ $\epsilon$ then
            remove record
    sort PQrec based on ops_saved
    initialize CQrec[layer][pattern] $\rightarrow$ None
    for layer $\ell$ in CNN do
        for $i$ in range(0, #patterns) do
            layer_pattern = CreateLayerPattern(mode, $\ell$, CQrec$[\ell][:][i]$)
            ops_saved, err = Simulate(CNN, $\mathcal{D}$, layer_pattern)
            CQrec $[\ell]$[layer_pattern] = (ops_saved, err)
    return CQrec
```

```
function LayerQuantizierPass(Optimizer, CQrec)
    for record in CQrec do
        if err $>$ $\epsilon$ then
            remove record
    sort CQrec based on ops_saved
    initialize LQrec[pattern] $\rightarrow$ None
    mask = CreateNetworkPattern(mode, CQrec[:][0])
    while mask is not None do
        ops_saved, err = Simulate(CNN, $\mathcal{D}$, mask)
        LQrec[mask] = (ops_saved, err)
        mask = UpdateMask(mask, CQrec)
    return LQrec
```