LEAN: GRAPH-BASED PRUNING FOR CONVOLUTIONAL NEURAL NETWORKS BY EXTRACTING LONGEST CHAINS

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural network pruning techniques can substantially reduce the computational cost of applying convolutional neural networks (CNNs). Common pruning methods determine which convolutional filters to remove by ranking the filters individually, i.e., without taking into account their interdependence. In this paper, we advocate the viewpoint that pruning should consider the interdependence between operators in the computational graph. We propose the **LongEst-chAiN** (LEAN) method that prunes CNNs by using graph-based algorithms to select relevant chains of convolutions. A CNN is interpreted as a graph, with the operator norm of each operator as distance metric for the edges. LEAN pruning iteratively extracts the highest value path from the graph to keep. In our experiments, we test LEAN pruning on several image-to-image tasks, including the well-known CamVid dataset, and a real-world X-ray CT dataset. Results indicate that LEAN pruning of CNNs (including U-Net and ResNet) maintains performance where individual pruning methods break down, sometimes achieving a 5–10x sparser network.

1 INTRODUCTION

In recent years, convolutional neural networks (CNNs) have become state-of-the-art machine learning models to solve a wide variety of tasks LeCun et al. (2015), including image classification Krizhevsky et al. (2012), segmentation Ronneberger et al. (2015), and denoising Tian et al. (2020). On many occasions, high-accuracy CNNs require a large amount of memory and computational resources. Long evaluation times can limit the applicability of neural networks in real-time applications, and large memory requirements can limit their usefulness for memory-limited applications Yang et al. (2017). Small networks may be more applicable in such settings, but may lack accuracy.

Neural network *pruning* Mozer & Smolensky (1989); Karnin (1990) has recently gained popularity as a technique to reduce the size of neural networks Blalock et al. (2020). Neural networks consist of learnable parameters, e.g., the parameters that make up a convolutional filter, that determine the network output. When pruning, the neural network is reduced in size by removing parameters while trying to maintain a high accuracy. The learnable parameters can be considered as single data elements (referred to as *individual parameters*), or structured in groups. A common pruning method for CNNs is to remove individual parameters Han et al. (2016). However, removing individual parameters may not directly lead to improved computational efficiency in CNNs Park et al. (2017). Because CNNs contain convolution operations, which are parameterized by a *filter* matrix, a convolution needs to be performed unless all filter parameters are pruned. Therefore, to reduce computational effort, it is necessary to remove or keep entire filters. Common methods for pruning CNNs determine what convolutional filters to remove by ranking filters separately Luo et al. (2017). This can be done by computing the operator norm of a filter, e.g., the L_1-norm Li et al. (2017).

We argue that as filters are not applied independently, i.e., the network output is formed by a composition of chains of filters and other operators, pruning individual filters can result in sub-optimal filter selection. Furthermore, to significantly reduce evaluation time, a severe pruning regime must be considered, i.e., a *pruning ratio* (percentage of remaining parameters after pruning) of 1–10%. In these severe pruning regimes, pruning CNNs by removing individual filters can result in *network*

disjointness, i.e., suboptimal network structures that contain sections that are not part of some path from the input to the network output. Some existing pruning methods take into account network structure to a limited degree. For instance, some methods prune entire groups of filters Molchanov et al. (2017); Liu et al. (2017); Luo et al. (2017). These methods consider the decomposition of a CNN into layers and set separate pruning criteria for each. However, these methods do not contain safeguards to avoid network disjointness.

In this paper, we present a novel pruning method called longest-chain (LEAN) pruning, which as opposed to conventional pruning approaches uses graph-based algorithms to keep or prune chains of dependent operations collectively. In LEAN, a CNN is represented as a graph that contains all the CNN operators, with the operator norm of each operator as distance metric for the edges. We argue that the strongest subnetworks in a CNN can be discovered by extracting the longest (multiplicative) paths, using linear time graph algorithms. The main focus of this work is to highlight how LEAN pruning can significantly improve the computation speed of CNNs for image-to-image tasks, and obtain high accuracy in the severe pruning regime that is difficult to achieve by pruning the operators individually.

This paper is structured as follows. In section 2, we explore existing pruning approaches. In section 3, we outline the preliminaries on CNNs, pruning filters, and the operator norm. Next, in section 4, we introduce LEAN pruning. We discuss the setup of our experiments in section 5. In section 6, we obtain experimental results for several datasets: (i) the CamVid dataset Brostow et al. (2008; 2009); (ii) a real-world dynamic CT dataset Coban & Lucka (2019); Coban et al. (2020); and (iii) a simulated high-noise segmentation dataset Pelt & Sethian (2018). For the CT experiment, we report practically realized wall time speedup. Also, we discuss the various pruned substructures that arise from the pruning methods. Our final conclusions are presented in section 7. Lastly, the various mathematical steps to compute the operator norms can be found in Appendix A.

2 Related work

Reducing the size of neural networks by removing parameters has been studied for decades Mozer & Smolensky (1989); Karnin (1990); Hassibi et al. (1993). Neural network pruning can be used for compressing a network to reduce memory requirements Han et al. (2016); Gong et al. (2015), and for creating computationally efficient networks Ye et al. (2018); Luo et al. (2017). Recent results focus on sparsifying the individual parameters in a neural network Han et al. (2015), or prune filters at runtime based on the feature maps Lin et al. (2017). Alternatively, one can prune entire channels He et al. (2017), or decide which channels to keep so that the feature maps approximate the output of the unpruned network over several training examples Luo et al. (2017). Apart from pruning the convolutional filters, it is possible to sparsify the parameters of other common components of CNNs, such as the batch normalization layers Ye et al. (2018); Liu et al. (2017); Huang & Wang (2018).

Many pruning approaches are aimed at reducing neural network size with very little accuracy drop Dong & Yang (2019); He et al. (2019); Molchanov et al. (2019); Zhao et al. (2019), as opposed to sacrificing accuracy in favor of computation speed. These approaches rarely exceed a pruning ratio of 12–50% Blalock et al. (2020); Luo et al. (2017); Lin et al. (2019); Han et al. (2015). When a high pruning ratio is used, e.g., a range 5–10% Lin et al. (2017); Liu et al. (2019), a significant drop in accuracy is observed. Pruning ratios of 2–10% can be achieved with an accuracy drop of 1-3% by learning-rate rewinding Renda et al. (2020). However, the reduction in FLOPs was less substantial (1.5–4.8 times), as individual parameters were pruned. A more extensive survey of the many different pruning methods can be found in Blalock et al. (2020).

Criteria for deciding which elements of a neural network to prune have been extensively studied. A common choice is to use the absolute value of a parameter to score parameter importance. However, whether the magnitude of a parameter is a reasonable metric to rank parameter importance has been questioned LeCun et al. (1990). In relation to using the operator norm for pruning, singular values can be used to compress network layers Denton et al. (2014), and for pruning feed-forward neural networks Abid et al. (2002). With respect to CNNs, several publications discuss computing the singular values of convolutional layers Sedghi et al. (2019). Furthermore, a definition of ReLU singular values was proposed recently with an accompanying upper bound Dittmer et al. (2019).

3 PRELIMINARIES

3.1 CNNS FOR SEGMENTATION

In this work, we demonstrate the results of the proposed pruning approach on a series of image segmentation problems. The goal of semantic image segmentation is to assign a class label to each pixel in an image. A segmentation CNN computes the function $f : \mathbb{R}^{m \times n} \rightarrow [0,1]^{k \times m \times n}$, which specifies the probability of each pixel being in one of the k classes for an $m \times n$ image.

CNNs are composed of layers of *operators* which pass images from each layer to the next. Concretely, every layer, for example convolutional and ReLU layers, has an input $\mathbf{x}$ and output $\mathbf{y}$. The input and output consist of one or more images, called channels. For instance, in a convolutional layer input images $\mathbf{x}_i$ are convolved with learned filters to produce an output channel $\mathbf{y}_j$

$$\mathbf{y}_j = \left(\sum_{i=1}^{n} h_{ij} * \mathbf{x}_i \right) + b_i. \tag{1}$$

Here h_{ij} represents a convolution filter, and b_i an additive bias parameter. In this way, every CNN operator produces an output which consists of a number of channels. The exact arrangement of operators, and connections between them, depends on the architecture.

A common operator to downsample images is the *strided convolution*. The stride defines the step size of the convolution kernel. A convolution with stride s defines a map $h : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{\frac{m}{s} \times \frac{n}{s}}$. Upsampling images can be done by *transposed convolutions*. Transposed convolutions intersperse the input image pixels with zeroes so that the output image has larger dimensions.

In addition to convolution operations, other common operators such as pooling and batch normalization are often used. A pooling operation downsamples an image by combining pixel values into one. Usually, the average or maximum value of a 2×2 patch is used. A batch normalization layer Ioffe & Szegedy (2015) is used to normalize the input images for convolutional layers. A batch normalization layer takes a feature map $\mathbf{x}_i$ and scales and shifts it by

$$\mathbf{y}_i = \gamma \frac{\mathbf{x}_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta. \tag{2}$$

Here, γ and β are the batch normalization parameters and biases which are learned during training, and μ_B and σ_B^2 are the running mean and variance of the mini-batch, i.e., the set of images used for the current training step. For an overview of CNN components we refer to Goodfellow et al. (2016).

3.2 PRUNING CONVOLUTION FILTERS

Pruning techniques aim to remove extraneous parameters from a neural network. Several schemes exist to prune parameters from a network, but retraining the network after pruning is critical to avoid significantly impacting accuracy Han et al. (2015). Pruning where a trained network is pruned once is called *one-shot pruning*. Alternatively, a network can be *fine-tuned*, where the network is repeatedly pruned by a certain percentage, and every step the network is retrained for a few epochs. Fine-tuning typically gives better results than one-shot pruning Renda et al. (2020).

Generic pruning algorithm: All pruning methods used in this work make use of the *fine-tuning* pruning algorithm outlined in Algorithm 1. The selection criteria for determining which filters to keep for each step defines the different pruning methods. The pruning ratio `pRatio` is the fraction of remaining convolutions we ultimately want to keep, and `stepRatio` is the fraction of convolutions that is pruned at each step.

Algorithm 1 Fine-tuning pruning algorithm

```
procedure PRUNE(MODEL, PRATIO, NSTEPS, EPOCHS)
    stepRatio ← e^(ln(pRatio)/nSteps)
    for step ← 0 to nSteps do
        pruneParams ← selectPrunePars(model, stepRatio)
        model ← removePars(model, prunedParams)
        for k ← 0 to epochs do
            model ← trainOneEpoch(model, trainData)
    return model
```

One approach for deciding which parameters are to be pruned is to use *magnitude pruning*, where the parameters with the lowest absolute value are pruned Han et al. (2015). Here, parameters can be pruned individually, or in a structured manner where entire filters or channels are pruned. Pruning individual parameters does not necessarily lead to improved computational efficiency. The threshold magnitude for the parameters to be pruned can be chosen per layer or globally. Pruning a network globally can give higher accuracy than setting thresholds per layer Blalock et al. (2020).

As we are interested in improving computational efficiency, we focus on structured pruning. When using *structured magnitude pruning*, a convolution filter $\mathbf{h} \in \mathbb{R}^{k \times k}$ is pruned by its L_1 vector norm $||\tilde{\mathbf{h}}||_1$, $\tilde{\mathbf{h}} \in \mathbb{R}^{k^2}$. Note that the norm is in this case the sum of the absolute values of the parameters.

3.3 Operator norm

As an alternative to the L_1-norm, one can interpret a convolution h as a linear operator which acts on the input feature maps, and score it according to an operator norm. The (induced) operator norm $\|\cdot\|_p$ is defined as

$$\|h\|_p := \sup \left\{ \|h * \mathbf{x}\|_p \;\middle|\; \mathbf{x} \in \mathbb{R}^n, \|\mathbf{x}\|_p = 1 \right\}.$$

A common operator norm is the *spectral norm*, which is induced by the Euclidean norm ($p = 2$). The spectral norm can be obtained by calculating the largest singular value of the matrix H associated with h, $\|h\|_2 = \sigma_{max}(H)$ Meyer (2000). A property that we will use is that the spectral norm is submultiplicative, i.e., we have

$$\|AB\| \leq \|A\| \cdot \|B\|, \quad \text{for all} \quad A, B \in \mathbb{R}^{n \times n}. \tag{3}$$

4 Method

The idea behind LEAN is to construct a weighted graph structure formed by the operators of the CNN and having their respective norms as weights, such that the longest paths in this graph are selected as important subnetworks. The remaining unselected operators will then be pruned.

4.1 LEAN: creating the pruning graph

In this section we define the *pruning graph* that will be the basis of the LEAN pruning algorithm. Previously we outlined the viewpoint that every CNN operator has an input $\mathbf{x}$ and output $\mathbf{y}$, which each consist of channels $\mathbf{x}_i$ and $\mathbf{y}_j$ (section 3.1). For each channel, we add a single node in the pruning graph. An edge connects two nodes corresponding to input channel $\mathbf{x}_i$ and output channel $\mathbf{y}_j$ if channel $\mathbf{x}_i$ is used in the computation of channel $\mathbf{y}_j$. For an output channel of n convolution operations, according to equation 1 we add n edges running from $\mathbf{x}_i$ to $\mathbf{y}_j$, each corresponding to exactly one filter h_{ij}. Certain CNN operators may perform computations in-place in practice, but we consider these operators as separate nodes in the pruning graph. For example, a batch normalization operation computes the output $\mathbf{y}_i$ according to equation 2. This yields a single edge between nodes $\mathbf{x}_i$ and $\mathbf{y}_i$ in the pruning graph.

We assign the edges *weights corresponding to the operator norm* of the operator that acts between those channels. An example CNN architecture and associated pruning graph is given in Figure 1. We describe the calculation of these operator norms in Appendix A. Since many CNN architectures change the sizes of images, we introduce a novel approach for computing the operator norm efficiently when convolutions are used to down- or upsample images in the Appendix. Edges of operations of which we cannot compute the operator norm are given a weight of 1. In this work we *only consider linear operators*, meaning that we assign edges corresponding to non-linear operators a weight of 1. Furthermore, we do not include the additive bias terms in the pruning graph. However, when pruning, such terms are removed from the CNN when all associated convolution or batch normalization operations are pruned.

Some CNNs contain operators that are meant to distribute features throughout the network, but are not implemented with learnable parameters, for example many residual connections in ResNet He et al. (2016). We include such operations in the pruning graph to model the dependency of operations. However, as these operations are not learnable we label them as unprunable. When an edge is unprunable, the corresponding CNN operation cannot be removed from the network.

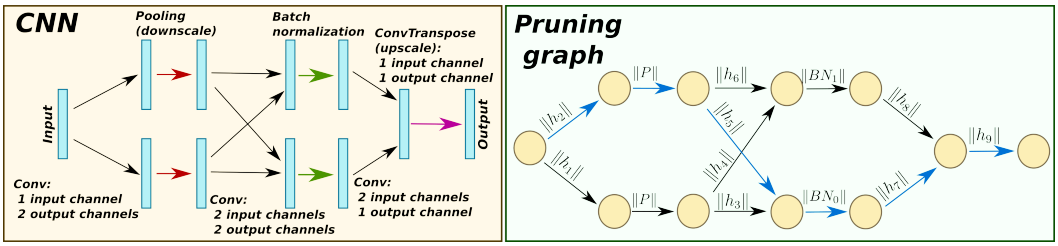

Figure 1: (Left) Example CNN architecture consisting of convolution, pooling, batch normalization, and upscaling (transposed convolution) layers. As edges corresponding to ReLU activation functions are given a weight of 1 in the pruning graph, they are omitted from the diagram. (Right) Example pruning graph associated to the CNN architecture on the left. Every channel is a node, and every operator is an edge if the parent node is used in the computation of the child node. The edge weights are the corresponding operator norms. For convenience, the convolution operations are labeled as h_i for $i = 1, \ldots, 9$. A possible longest, i.e., highest multiplicative value, path is illustrated in blue.

In this work, we do not consider CNNs with recurrent connections. Therefore, the pruning graph is a Directed Acyclic Graph (DAG). In section 5.1, we comment on the approximations LEAN makes in relation to non-linear components, and what adaptations to CNNs can be made to mitigate this.

4.2 LEAN: EXTRACTING CHAINS FROM THE GRAPH

The LEAN method prunes chains of convolutions based on paths in the graph; we **keep** the *highest operator norm paths*. Specifically, because operator norms are composed submultiplicatively (as in equation 3), we retain the highest *multiplicative* operator norm paths. We refer to this as LongEstchAiN (LEAN) pruning. When we perform LEAN pruning, we iteratively extract such paths from the graph until we have reached the pruning ratio. This means that *the edges that are **not** extracted are pruned*.

Algorithm 2 LEAN

```
procedure LEAN(MODEL, PRUNERATIO)
    graph ← createPruningGraph(model)
    retainedConvs ← []
    while fractionRemainingConvs < 1 − pruneRatio do
        bestChain ← longestPath(graph)
        retainedConvs ← retainedConvs + bestChain
        graph ← removeFromGraph(graph, bestChain)
    return retainedConvs
```

For each pruning step in the generic pruning algorithm outlined in Algorithm 1, Algorithm 2 is used to select the parameters to prune (line 4 in the pseudo code). Note that while unprunable edges can be part of a longest path to extract operator chains, they are not taken into account when determining the pruning ratio. As we prune at least one operator for every pruning step, Algorithm 2 is run at most once for each operator. For DAGs, the longest (highest value) path in a graph can be found in $\mathcal{O}(|V| + |E|)$ time, where $|V|$ is the number of nodes, and $|E|$ is the number of edges Sedgewick & Wayne (2011). For a CNN with k operators (edges), and m channels (nodes), this means that the worst-case complexity of LEAN is $\mathcal{O}(kp(k + m))$ for a pruning ratio p.

After every LEAN pruning step is concluded, we prune redundant channels. A channel is considered redundant if it is a homogeneous constant image for all input data. We try to prune such channels by removing from the pruning graph: 1) nodes without incoming edges; or 2) nodes where the succeeding batch normalization node has running variance below some threshold (10^{-40} by default). Case 2) is to remove convolutions which still have input channels, but whose output is always zero after applying the ReLU activation function.

5 EXPERIMENTAL SETUP

In our experiments, we compare LEAN pruning to two different individual pruning filter methods. The first method we compare to is **structured magnitude pruning**, i.e., pruning entire filters by

CNN	Convolution Strided	Convolution Transposed	Convolution Dilated	Pooling	Batch normalization	# Parameters	Edges pruning graph
MS-D	No	No	Yes	No	No	$4.57 \cdot 10^4$	$5.05 \cdot 10^3$
ResNet50	Yes	No	Yes	Yes	Yes	$3.29 \cdot 10^7$	$1.44 \cdot 10^7$
UNet4	No	Yes	No	Yes	Yes	$1.48 \cdot 10^7$	$1.84 \cdot 10^6$

Table 1: Operators present in MS-D, FCN-ResNet50, FCN-U-Net4 architectures.

their L_1-norm (see section 3.2), and the second method we compare to is **operator norm pruning**, i.e., pruning individual filters using the operator norm. Here, we choose to use the spectral norm. The difference between LEAN and structured magnitude pruning is 1) the operator norm; 2) the consideration of network structure. The comparison of LEAN to structured operator norm pruning measures the effect of incorporating the networks structure, serving as a kind of ablation study.

As explained in section 3.2, for both structured magnitude and operator norm pruning, we can choose between setting the pruning thresholds globally, or per layer. In the severe pruning regime that we consider in these experiments, setting pruning thresholds per layer can be detrimental as entire layers will be pruned at severe pruning ratios, creating a disjoint network. Therefore, we set pruning thresholds globally in these experiments for structured magnitude pruning and operator norm pruning. Similarly to for LEAN pruning, for both these methods, we remove additive bias terms from the CNN when all associated convolution or batch normalization operations are pruned.

5.1 CNN ARCHITECTURES

In this section, we present three fully-convolutional CNN architectures that are used in the experiments; the mixed-scale dense convolutional neural network (MS-D) network Pelt & Sethian (2018), U-Net Ronneberger et al. (2015), and ResNet He et al. (2016). These networks all have skip connections which makes them interesting from a graph theory point of view.

In our experiments we use MS-D networks implemented as in Pelt & Sethian (2018). We chose a depth of either 100 or 150 layers, each with 1 channel per layer. The dilations for layer i were $1 + (i \mod 10)$, and all filters have a size of 3×3, except the final 1×1-layer. As U-Net architecture we use a fully-convolutional (FCN) U-Net4 network, i.e., a U-Net with 4 scaling operations. As ResNet architecture, we use an FCN-ResNet50 network. This is a ResNet with 50 layers, and a bottleneck block-structure He et al. (2016). Table 1 outlines which operators are present in the networks.

In all experiments we convert the CNNs to a pruning graph as described in section 4. To be able to calculate the operator norms of the pooling layers, we replace the max pooling layers of ResNet and U-Net with average pooling layers. In previous work it was shown that in some cases U-Net with average pooling can perform better than U-Net with max pooling Astono et al. (2020).

5.2 EXPERIMENTAL SETUP

Since we are interested in computational speedup, not memory compression, we want to prune entire filters in our experiments. Therefore, we measure pruning ratio as the fraction of convolutions remaining, rather than the fraction of parameters that is pruned, when performing the pruning process. This means that other parameters, such as batch normalization parameters, are only pruned when they become redundant, but do not count towards the pruning ratio for each step of the algorithm.

For all pruning experiments on the MS-D network, we exclude the output layer from pruning. In preliminary experiments, for both structured magnitude pruning and operator norm pruning we observed that almost the entire final layer is immediately pruned, which would create an unfair comparison to LEAN. This may be due to the lack of batch normalization layers, or zero-initialization of the output layer Pelt & Sethian (2018). As the output layer of the MS-D networks uses few FLOPs, it has little impact on computational efficiency.

We assess the performance of pruned neural networks across 5 independent runs of fine-tuning (Algorithm 1). The MS-D networks were pruned to a pruning ratio of 1% in 45 steps. The U-Net4, and ResNet50 networks were pruned to a ratio of 0.1% in 30 steps. All were retrained for 5 epochs after each step. We chose relatively severe pruning ratios because we are interested in significant

computational speedup. U-Net4 and ResNet50 are pruned to a lower ratio as they have orders of magnitude more convolutions than the MS-D network.

All CNN architectures are implemented in PyTorch. The fully-convolutional ResNet50 model is adapted from PyTorch's model zoo code, and U-Net4 is adapted from the PyTorch-UNet repository Milesi (2020). For the MS-D network we used a PyTorch adaptation Hendriksen (2020). The networks were trained using ADAM Kingma & Ba (2014) with $lr = 0.001$, minimizing the negative log likelihood function. For each dataset, we have trained a single model as a starting point for pruning. For every experimental run, the same trained model was fine-tuned. All experiments were run on a workstation with an AMD Ryzen 3800X processor and NVIDIA GeForce RTX 2070 Super GPU, or on a server with two 8-core Xeon CPUs and four Titan X (12GB) GPUs.

5.3 DATASETS

For our experiments, we consider three datasets: a high-noise, but relatively simple, segmentation dataset Pelt & Sethian (2018); the well-known CamVid dataset Brostow et al. (2008; 2009); and a real-world CT dataset Coban & Lucka (2019); Coban et al. (2020) to test the methods in practice.

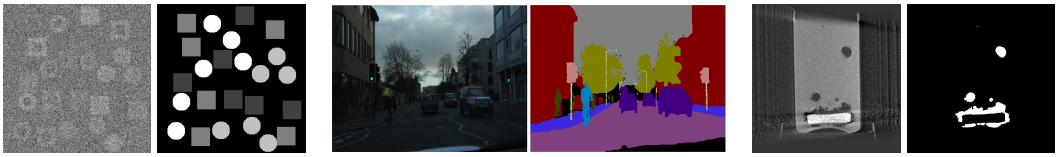

Figure 2: Example input and target images of the (Left) Circle-Square (CS), (middle) CamVid, (right) real-world dynamic CT datasets.

5.3.1 SIMULATED CIRCLE-SQUARE (CS) DATASET

We used a simulated high-noise 5-class segmentation dataset containing 256×256 images of randomly placed squares and circles (CS dataset) Pelt & Sethian (2018) (see Figure 2). The objects were assigned a random grey value and Gaussian noise was added to the images. In total, we generated 1000 training, 250 validation, and 100 test images. Experimental results on the CS dataset are quantified using global accuracy. As a starting point for pruning, we trained an MS-D network with 100 layers which was able to obtain a global accuracy of 97.5%. The trained ResNet50 and U-Net4 networks had an 95.8%, and 97.6% respectively.

5.3.2 CAMVID

The Cambridge-driving Labeled Video Database (CamVid) Brostow et al. (2008; 2009) is a collection of videos with labels, captured from the perspective of a driving automobile. In total, 700 labeled frames are split into 367 training images, 100 validation images, and 233 test images. As the CamVid dataset contains few input images, we combined the training and validation datasets and trained for a fixed 500 epochs. Similar to other papers that apply CNNs to CamVid Brostow et al. (2009); Badrinarayanan et al. (2017); Paszke et al. (2017), we use 11 classes representing various roadside objects, and a single class representing unlabeled pixels (see Figure 2).

We used median frequency balancing Eigen & Fergus (2015) to balance classes for training, and set the class weight for *unlabeled* to zero. During training, we used data augmentation by randomly cropping and randomly (horizontally) flipping input images. Similar to other papers, experimental results on the CamVid dataset are quantified using mean Intersection-over-Union (mIoU) Badrinarayanan et al. (2017); Paszke et al. (2017). As a starting point for pruning, we trained an MS-D network with 150 layers with an mIoU of 0.5183, a ResNet50 with an mIoU of 0.7075, and a U-Net4 with an mIoU of 0.6534.

5.3.3 REAL-WORLD DYNAMIC CT DATASET

Lastly, we used a real-world dynamic X-ray CT dataset of a dissolving tablet suspended in gel Coban & Lucka (2019); Coban et al. (2020). The dataset contains noisy tomographic images of bubbles which are to be segmented within a glass container filled with gel Schoonhoven et al. (2020) (see Figure 2). The dataset consists of 512×512 images, split into 9216 training images, 2048 validation

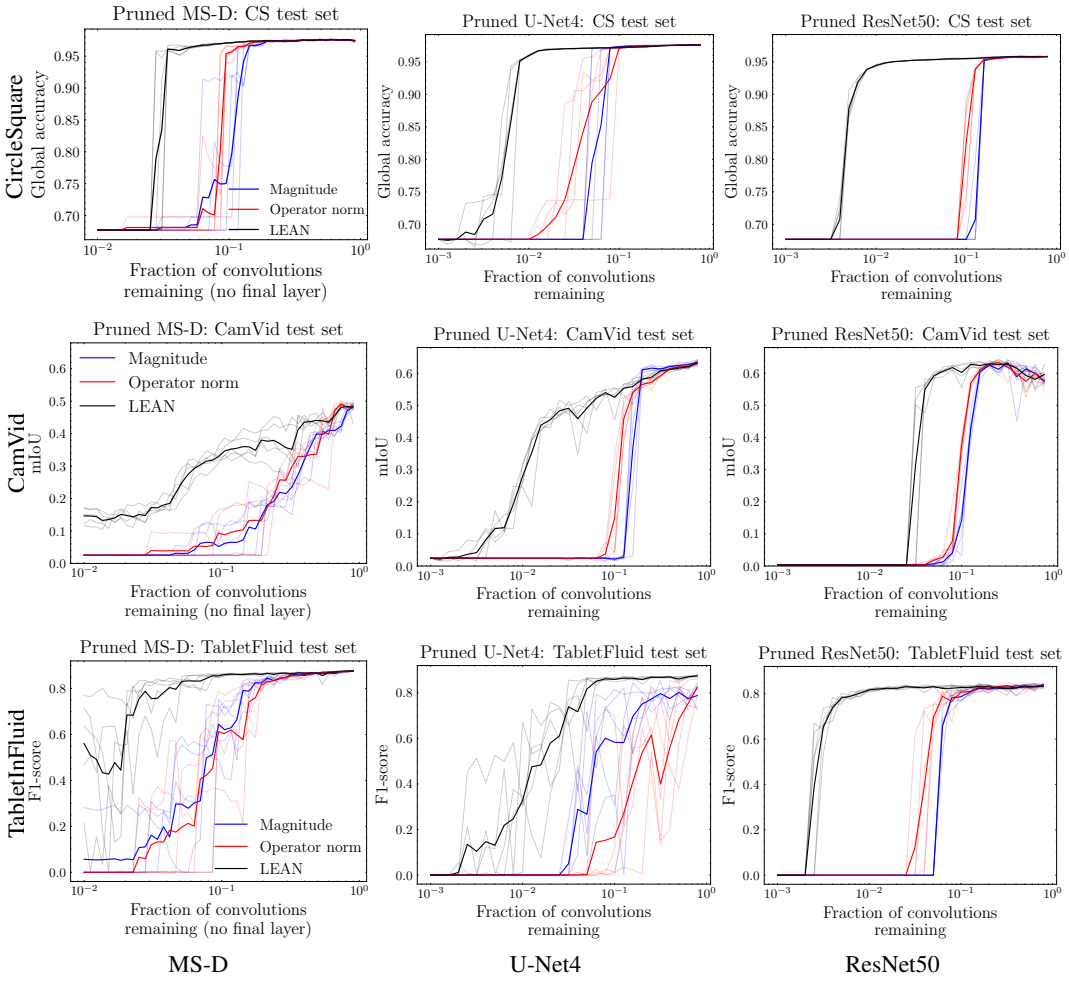

Figure 3: Experimental results for pruned networks on the test datasets for structured magnitude, operator norm, and LEAN pruning. (Top) CircleSquare (CS) dataset, (middle) CamVid dataset, (bottom) TabletInFluid dataset. From left to right we have MS-D, U-Net4, and ResNet50. Base model is pruned to a ratio of 1% (MS-D) or 0.1% (ResNet50 U-Net4) for all datasets. Every pruning step, the mIoU on the test set is separately recorded and displayed in the graph. Displayed is the mean of 5 runs, with each individual run as a translucent line.

images, and 1536 test images. We use the F1-score as was done in the original paper Schoonhoven et al. (2020) because the large amount of background pixels make global accuracy an unsuitable metric. As a starting point for pruning, we trained an MS-D network with 100 layers with an F1-score of 0.8816, a ResNet50 with an F1-score of 0.8506, and a U-Net4 with an F1-score of 0.8811.

6 Results

6.1 Experimental results for severe pruning

The results of the pruning experiments are displayed in Figure 3. We see that LEAN pruning is able to create pruned networks with better accuracy than structured magnitude and operator norm pruning for all networks, and all datasets. The pruning ratio that can be achieved without significant loss of accuracy depends on the network architecture and the complexity of the dataset.

For the CS dataset, the MS-D network achieved a pruning ratio of 3.4% with an average reduction in accuracy of 1.5% using LEAN pruning. On the CamVid dataset, the mIoU declines as pruning progresses which may be due to the higher difficulty of the segmentation problem. For the X-ray CT dataset, LEAN achieved a pruning ratio of 5.1% with an F1-score drop of on average 5.7%.

	Evaluation time (seconds)					
Batch size	1	2	4	6	8	16
Base MS-D	52.1	55.6	60.8	66.9	70.7	84.3
Pruned MS-D	7.0	5.7	5.2	5.1	5.0	4.8

Table 2: Practically realized wall time speedup of pruned MS-D networks evaluated on the entire test dataset.

For U-Net4, LEAN achieved a 0.79% pruning ratio with an accuracy reduction of on average 2.5% for CS. Similarly to MS-D, on the CamVid dataset mIoU steadily decreased, although it did not drop off as severely for LEAN. Nevertheless, at a 3.1% pruning ratio a large mIoU reduction of on average 24.8% was observed. For the X-ray CT dataset, LEAN achieved a pruning ratio of 6.3% with an F1-score drop of on average 3.4%.

For ResNet50, LEAN achieved a 0.79% pruning ratio with an accuracy reduction of on average 2.1% for CS, and 5.0% pruning ratio with an average mIoU reduction of 16.3% for CamVid. For the X-ray CT dataset, LEAN achieved a pruning ratio of 1.0% with an average F1 reduction of 4.1%.

6.2 Speedup real-world dynamic X-ray CT segmentation

In addition to measuring the achievable pruning ratios, we are interested in measuring the practically realized wall-time speedup. We tested this for the real-time X-ray CT dataset as practical speedup has immediate benefits for such experiments. Existing pruning support in PyTorch only masks pruned parameters, thereby not creating a faster network. Therefore, we implemented a custom MS-D model which loads only the unpruned parameters. During the experiments, one MS-D network was able to achieve an F1-score of 0.8343 (drop of 5.4%) at a pruning ratio of 2.5% (using LEAN). The practically realized wall times for evaluating the test set are given in Table 2. Using the fastest batch sizes (1 and 16 respectively), the pruned network is 10.9 times faster than the base network.

7 Conclusion

In this paper, we have introduced a novel pruning method (LEAN) for CNN pruning by extracting the highest value paths of operators. We incorporate existing graph algorithms, and computationally efficient methods for determining the operator norm. We show that LEAN pruning permits removing significantly more operations while retaining better network accuracy than pruning based on individual filters across different norms. Our results show that LEAN pruning can increase the speed of the network, both in theoretical speedup (FLOPs reduction) and in practice. In conclusion, LEAN enables severe pruning of CNNs while maintaining a high accuracy, by effectively exploiting the interdependency of network operations.

Future work could be split along two lines. On the one hand, there are more CNN operators for which methods to compute their operator norms could be developed. Notably, we have disregarded non-linear operators in this work. If we could include the non-linear activation functions, and max-pooling operators, perhaps LEAN could be further improved. On the other hand, new graph theoretic approaches for extracting meaningful paths from the graph could be explored. Such algorithms are already abundant in the field of graph theory, and could quite readily be carried over to neural network pruning research.

References

S. Abid, F. Fnaiech, and M. Najim. A new neural network pruning method based on the singular value decomposition and the weight initialisation. In *2002 11th European Signal Processing Conference*, pp. 1–4, 2002. 2

Indriani P. Astono, James S. Welsh, Stephan Chalup, and Peter Greer. Optimisation of 2D U-Net Model Components for Automatic Prostate Segmentation on MRI. *Applied Sciences*, 10 (7), 2020. ISSN 2076-3417. doi: 10.3390/app10072601. URL https://www.mdpi.com/2076-3417/10/7/2601. 6

Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017. 7

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *Proceedings of Machine Learning and Systems*, 2020. 1, 2, 4

Gabriel J Brostow, Jamie Shotton, Julien Fauqueur, and Roberto Cipolla. Segmentation and recognition using structure from motion point clouds. In *European conference on computer vision (ECCV)*, pp. 44–57. Springer, 2008. 2, 7

Gabriel J Brostow, Julien Fauqueur, and Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97, 2009. 2, 7

Sophia Bethany Coban and Felix Lucka. Dynamic 3D X-ray micro-CT data of a tablet dissolution in a water-based gel, 2019. URL https://doi.org/10.5281/zenodo.3610187. 2, 7

Sophia Bethany Coban, Felix Lucka, Willem Jan Palenstijn, Denis Van Loo, and Kees Joost Batenburg. Explorative Imaging and Its Implementation at the FleX-ray Laboratory. *Journal of Imaging*, 6(4):18, 2020. URL "https://doi.org/10.3390/jimaging6040018". 2, 7

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pp. 1269–1277, 2014. 2

S. Dittmer, E. J., and P. Maass. Singular values for relu layers. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, 2019. 2

Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. In *Advances in Neural Information Processing Systems*, pp. 760–771, 2019. 2

David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pp. 2650–2658, 2015. 7

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *International Conference on Learning Representations (ICLR)*, 2019.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *International Conference on Learning Representations (ICLR)*, 2015. 2

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016. 3

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015. 2, 3, 4

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016. 1, 2

Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pp. 293–299. IEEE, 1993. 2

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. 4, 6

Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 1389–1397, 2017. 2

Allard A. Hendriksen. Mixed-scale dense networks for pytorch. https://github.com/ahendriksen/msd_pytorch, 2020. 7

Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 304–320, 2018. 2

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning (ICML)*, 2015. 3

Anil K. Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall, Inc., USA, 1989. ISBN 0133361659. 13

E. D. Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE Transactions on Neural Networks*, 1(2):239–242, 1990. 1, 2

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2014. 7

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012. 1

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990. 2

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. 1

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient ConvNets. In *5th International Conference on Learning Representations, ICLR 2017*, 2017. 1

Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *Advances in neural information processing systems*, pp. 2181–2191, 2017. 2

Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured CNN pruning via generative adversarial learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2736–2744, 2017. 2

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *International Conference on Learning Representations (ICLR)*, 2019. 2

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015. 14

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pp. 5058–5066, 2017. 1, 2

Carl D Meyer. *Matrix analysis and applied linear algebra*, volume 71. Siam, 2000. 4

Alexandre Milesi. Unet: semantic segmentation with pytorch. https://github.com/milesial/Pytorch-UNet, 2020. 7

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *International Conference on Learning Representations (ICLR)*, 2017. 2

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

Michael C Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems 1*, pp. 107–115. Morgan-Kaufmann, 1989. 1, 2

Jongsoo Park, Sheng Li, Wei Wen, Ping Tak Peter Tang, Hai Li, Yiran Chen, and Pradeep Dubey. Faster CNNs with direct sparse convolutions and guided pruning. *International Conference on Learning Representations (ICLR)*, 2017. 1

Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. ENet: A deep neural network architecture for real-time semantic segmentation. *International Conference on Learning Representations (ICLR)*, 2017. 7

Daniël M. Pelt and James A. Sethian. A mixed-scale dense convolutional neural network for image analysis. *Proceedings of the National Academy of Sciences*, 115(2):254–259, 2018. ISSN 0027-8424. doi: 10.1073/pnas.1715832114. URL http://www.pnas.org/content/115/2/254. 2, 6, 7

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. *International Conference on Learning Representations (ICLR)*, 2020. 2, 3

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, 2015. 1, 6

R. Schoonhoven, J. W. Buurlage, D. M. Pelt, and K. J. Batenburg. Real-time segmentation for tomographic imaging. In *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, 2020. doi: 10.1109/MLSP49062.2020.9231642. 7, 8

Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-wesley professional, 4 edition, 2011. 5

Hanie Sedghi, Vineet Gupta, and Philip M Long. The singular values of convolutional layers. *International Conference on Learning Representations (ICLR)*, 2019. 2, 13

Chunwei Tian, Yong Xu, and Wangmeng Zuo. Image denoising using deep CNN with batch renormalization. *Neural Networks*, 121:461–473, 2020. 1

T. Yang, Y. Chen, and V. Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6071–6079, 2017. 1

Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *International Conference on Learning Representations (ICLR)*, 2018. 2

Chenglong Zhao, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian. Variational convolutional neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

A Appendix: Operator norm calculation

In addition to the structure of nodes and edges, the pruning graph contains edge we assign edges in the pruning graph weights related to the operator norms of the CNN operators. In this section we will discuss how the operator norms for various common CNN operators are computed.

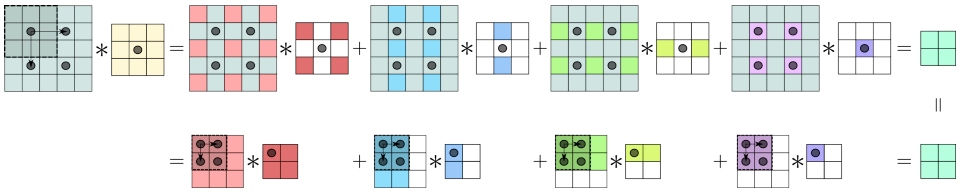

Figure 4: The output of a strided convolution of stride 2 can be obtained by summing the result of regular convolutions. First, split the image, and the original convolution filter, into 4 smaller parts (corresponding sections given in color) with white entries representing zeroes. Next, sum the outputs of 4 regular convolutions pixel by pixel. The diagram outlines the procedure for a 5×5 image and 3×3 kernel. The convolution is performed without padding and produces a 2×2 output image. The dots in the image represent the positions of the dot in the filter, during computation of the convolution.

A.1 Operator norm of convolutions

As mentioned in section 3.3, the spectral norm of a convolution with filter h can be determined by the singular values of the associated matrix H. A substantially more efficient method than constructing H is to make use of Fourier transforms Sedghi et al. (2019). The matrix H is diagonalized by the two-dimensional unitary Discrete Fourier Transform (DFT), and the eigenvalues of H are given by the two-dimensional DFT of h^+ (see section 5 of Jain (1989)). For an image of size $n \times n$, h^+ is the filter padded with zeroes to size $n \times n$. Practically, the singular values of H are the magnitudes of the complex entries of $F_{2D}(h^+)$. Here F_{2D} is the 2D Fourier transform.

$$\sigma_{max}(H) = \max\left\{\left|F_{2D}(h^+)\right|\right\}$$

Note that the Fourier-based method used in this work assumes a doubly-blocked circulant definition of H, which corresponds to periodic boundary conditions as opposed to zero padding. When n is chosen sufficiently large, the difference in operator norm is negligible.

A.2 Downsampling: operator norm for strided convolutions

The operator norm of a strided convolution can be computed by observing that it is equal to the sum of several regular convolutions on smaller input images. Consider a single parameter for a stride 2 convolution kernel. Because of the stride, it is multiplied only with every other pixel (both horizontally and vertically). Similarly, parameters in the kernel which are exactly 2 apart are multiplied with the same pixels. In Figure 4 we outline the procedure for a 3×3 kernel, with stride 2. Here, there are 4 different convolution positions before the pattern repeats itself; every combination of odd and even numbered indices for the horizontal and vertical axes (each pattern is marked by a different color). In general, the same properties hold for other values of s. We will use these properties to compute the operator norm.

Denote the partitioned convolution kernels as $h^{[i]}$, and the partitioned input images as $X^{[i]}$. Both $h^{[i]}$ and $X^{[i]}$ are zero-padded to the correct size (illustrated as white cells in Figure 4). Then we have that

$$h *_2 X = \sum_{i=1}^{4} h^{[i]} * X^{[i]} \tag{4}$$

where $*_2$ denotes a stride 2 convolution. As $h^{[i]} * X^{[i]}$ is a regular convolution, we can determine the singular values of $h^{[i]}$ using Fourier transforms as described above.

In general, equation 4 is analogous to the equation of a convolutional layer with 4 input channels and 1 output channel. The operator norm of such a convolutional layer is described in Sedghi et al. (2019). There, they collect the singular values of the convolutional layer into a tensor $P \in \mathbb{R}^{4\times 1\times n\times n}$, with

$$P_{c_{in},c_{out},i,j} = F_{2D}(h^{[c_{in}]+})_{i,j}.$$

According to Theorem 6 of Sedghi et al. (2019), the spectral norm of the convolutional layer equals the maximum of the singular values of the 4×1 matrices $P_{:,:,i,j}$. Since the matrices are single-column, their singular value equals their L_2-norm. Therefore, the spectral norm of h equals

$$\|h\| = \max_{i,j}\left\{\sum_{c_{in}} P^2_{c_{in},:,i,j}\right\}$$

In the case of 2×2 kernels, and stride 2, a relatively simple calculation can be performed. In this case, each $h^{[i]}$ contains just one non-zero parameter, and the convolution $h^{[i]} * X^{[i]}$ is in fact a scalar multiplication. So for a 2×2 kernel with parameters h_{ij}, we get that $\sigma_{max}(h) = \sqrt{\sum_{i,j} h_{i,j}^2}$.

A.3 UPSAMPLING: OPERATOR NORM FOR TRANSPOSED CONVOLUTIONS

The operator norm of a transposed convolution can be computed in a similar fashion as the strided convolution. The matrix associated with a transposed convolution of stride s is exactly the transpose of the matrix for a stride s convolution Long et al. (2015). In general we have for any linear map A between Euclidean vector spaces that $\|A\| = \|A^T\|$. Therefore, for a given transposed convolution kernel h, the operator norm is equal to that as if h were applied as a strided convolution. As such, we can utilize the procedure outlined in section A.2.

A.4 OPERATOR NORM FOR BATCH NORMALIZATION AND POOLING OPERATORS

As operator norm for a batch normalization channel B we take the multiplicative scaling it applies to images (see section 3.1)

$$\|B\| = \left| \frac{\gamma}{\sqrt{\sigma_B^2 + \epsilon}} \right|.$$

Note that we do not include the additive β term in the computation of the operator norm, but rather interpret it as a separate bias term.

In addition to convolutional operators, there are other CNN operators to which we can attribute an operator norm. Average pooling can be seen as a convolution operation. *Average pooling* with stride s is the same as a stride s convolution, with an $s \times s$ kernel with values $1/s^2$ everywhere. As such, we can calculate the operator norm efficiently as outlined in section A.2.