Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

George Emmins

April 30, 2024

Structured Pruning of Convolutional Neural Networks in the Context of Visual Place Recognition

Project Supervisor: Shoaib Ehsan
Second Examiner: Richard Watson

A project report submitted for the award of
BSc Computer Science

Abstract

Visual place recognition (VPR) is the act of recognising previously seen places, only from images. The field of VPR has developed over the years, with most state-of-the-art approaches now utilising deep learning, typically in the form of a convolutional neural network (CNN), for feature extraction. However, the use of CNNs is computationally intensive which can limit the viability of these method's deployment, especially to embedded systems. Network pruning is one of several methods which have been shown to effectively reduce the memory and computational requirements of CNNs, without significantly reducing their accuracy. Structured pruning is a more practical approach to this than unstructured pruning as the resulting networks don't require specialised hardware/libraries to utilise the speedup gained from sparse matrices generated from an unstructured approach. This paper investigates the viability of structured pruning on CNNs for VPR over a number of factors and shows that models can gain a meaningful increase in inference speed and reduction in size and memory requirements, on a standard CUDA GPU, when pruned. However, the ability of the models to maintain their baseline level of performance at higher levels of sparsity appears dependent on a number of factors such as the CNN architecture and pruning method used.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

ECS Statement of Originality Template, updated August 2018, Alex Weddell aiofficer@ecs.soton.ac.uk

Contents

1 Project Description **4**
1.1 The Problem 4
1.2 Goals 4

2 Background Research and Related Work **5**
2.1 Visual Place Recognition 5
2.2 Network Pruning 5
2.3 The 'Lottery Ticket Hypothesis' 6

3 Experiment Design **7**
3.1 High Level Design 7
3.2 Image Representation Methods 7
3.2.1 NetVLAD 7
3.2.2 GeM 8
3.3 Network Architectures 8
3.3.1 ResNet 9
3.3.2 MobileNetV3 9
3.3.3 EfficientNet 10
3.4 Pruning Methods 10
3.4.1 L1 Norm & L2 Norm 10
3.4.2 LAMP 10
3.4.3 FPGM 11
3.5 Dataset 11
3.6 Evaluation methods 12
3.7 Implementation 12
3.8 Justification of Approach 13

4 Results **14**
4.1 Recall 14
4.2 Inference Time 16
4.3 Model Size 17
4.4 Computation 17
4.5 Memory Usage 18

5 Evaluation of Results **20**
5.1 Recall 20
5.2 Inference Time 21
5.3 Memory Usage 21

6 Conclusions and Further Work **22**

7 Planned Schedule and Actual Schedule of Work **23**
7.1 Planned Schedule of Work 23
7.2 Actual Schedule of Work 23

A Tabulated Data **27**

B Archive Contents **33**

1 Project Description

1.1 The Problem

Visual Place Recognition is a powerful tool to localise one's self only using vision information. VPR has many use cases and has been described as "one of the essential and challenging problems in the field of robotics and computer vision" [1]. Chief among these uses is in robotics, where there is a need for real-time processing of images and limits on the availability of memory, compute power and power in general often exist. These constraints exist in opposition to the use of CNNs in VPR as these typically require powerful GPUs with large amounts of VRAM to run at a speed that would be acceptable for use in real-time applications. Although other computational bottlenecks exist within the VPR pipeline, this means that one of the main hurdles to overcome to be able to deploy deep-learning VPR methods to computationally limited systems is the reduction of computational power required to extract image representations using CNNs.

A number of ways exist to do this, and the removal of parameters from CNNs to reduce their size and number of computations (pruning) has been shown to achieve this, whilst maintaining acceptable levels of accuracy in the use case of object recognition. This project investigates whether pruning also successfully achieves this within the domain of VPR.

1.2 Goals

The aim of this project is to investigate the effects of structured pruning of CNNs in use in VPR. This is to ascertain whether or not structured pruning is a viable method of model compression for this use case, possibly enabling VPR models to be deployed on a wider range of systems, due to a decrease in the compute and memory requirements of the models at inference-time.

To achieve this goal, the investigation looks at the effect of 4 different structured pruning algorithms on 2 different VPR image representation methods with 3 different CNN architectures. The performance of these methods is then evaluated at different levels of sparsity, measuring their:

1. Recall

2. Inference time

3. # of model parameters

4. # of direct memory accesses (DMAs)

5. # of multiply-accumulations (MACs)

6. Memory usage at inference-time

These metrics will provide us with an overall picture of the performance and computational requirements of the VPR methods at different levels of sparsity and allow us to make judgements on whether or not the pruning of the networks has provided the expected benefits whilst maintaining the system's ability to perform VPR at an acceptable accuracy.

2 Background Research and Related Work

2.1 Visual Place Recognition

Visual place recognition (VPR) can be described as "the ability to recognize a known place in the environment using vision as the main sensor modality" [2]. VPR has several use cases in the field of computer vision and robotics, e.g in visual SLAM [3] for loop closure detection, and the volume of research surrounding VPR has grown consistently over the past couple decades [4].

The basic problem of VPR can be described as one of image retrieval, whereby a query image is matched with one or more images in a database, indicating a belief that they are of the same place. The basic process for this is to compare some mathematical representation of the query image against those of the database images and use a similarity score to predict whether they are from the same place or not, typically using a predetermined similarity threshold.

Initially, VPR approaches used hand-crafted feature-based models [1] such as Bag-of-Words [5] to represent their images, with techniques such as SIFT [6] and SURF [7] for feature detection and description. However, in 2014 it was shown that the use of deep learning for feature generation can achieve significantly better results than using hand-crafted methods [8]. Since then, the majority of research in the field of VPR has involved methods that utilise deep learning, typically in the form of convolutional neural networks (CNNs). The initial deep learning VPR methods utilized pre-trained CNNs such as OverFeat in [8] or AlexNet pre-trained on ImageNet in [2]. While the performance of these models is far better than traditional methods, custom models trained on VPR-specific data-sets, such as in [9] or [10], typically perform even better and make up a large part of the state-of-the-art techniques.

2.2 Network Pruning

These network architectures, however, can contain millions of parameters and require large amounts of computing power. This can limit the deployment of these models, especially to edge and embedded systems. This is particularly relevant for VPR where the principle use cases are in the field of robotics, and the computing power is typically limited. To remedy this issue, model compression can be used. There are a number of methods of model compression, and these can be used together to great effect [11]. Pruning is one of these methods. "The term “pruning” refers to removing components of a network to produce sparse models for acceleration and compression" [12] and it aims to achieve this whilst maintaining appropriate performance of the model. The idea of pruning was initially put forward in [13] and has since shown to be a viable method for pruning neural networks [14], [15], [16].

Methods of pruning typically fall under one of two categories: structured or unstructured. Unstructured pruning of networks removes the connections in the network and can lead to very sparse networks. However, due to the unstructured nature of the sparsity, it is hard for typical computer hardware to take advantage of the reduction in size of the network. These networks typically require specialised hardware or libraries such as in [17] to achieve genuine acceleration. Structured pruning, on the other hand, removes whole filters from the networks "and can achieve realistic acceleration and compression with standard hardware" [12]. Because of this, structured pruning methods are, today, typically favoured over unstructured methods [18].

Modern pruning methods typically follow an iterative process whereby the original trained network is partially pruned via some heuristic, fine-tuned and pruned again, until the compression requirements have been met, for example in [14] and [19]. However, several methods exist that forego this iterative process and instead prune the network at initialisation [20, 21] in order to avoid the significant time it takes to train the networks pruned using more traditional methods. The 'single-shot' pruning methods, though, typically under perform their more traditional counterparts [22].

As previously mentioned, pruning methods use some form of heuristic to inform which filters to remove in the pruning process. It is typically by these criteria the methods differ. Weight dependent criteria evaluate the importance of the filters in the network by assessing the weights within the filters. Filter norm is often used as a weight dependent criteria to prune filters. For example, in [23] the l_1-norm is used to identify filters that contribute less to the network's output. Whereas [24] uses l_2-norm and instead sets these filters to 0 rather than removing them, claiming improved performance over [23]. In [25], filters are instead removed based on their redundancy and claims that a smaller norm doesn't necessarily indicate a smaller importance, demonstrating performance that improves on the previous studies.

Activation base pruning differs from weight based methods as they "harness the activation maps for pruning decisions" [12]. Methods such as [26] use the current layer's activations to decide which filters to prune and [16] uses the activations of the layer $+ 1$ to make that decision.

2.3 The 'Lottery Ticket Hypothesis'

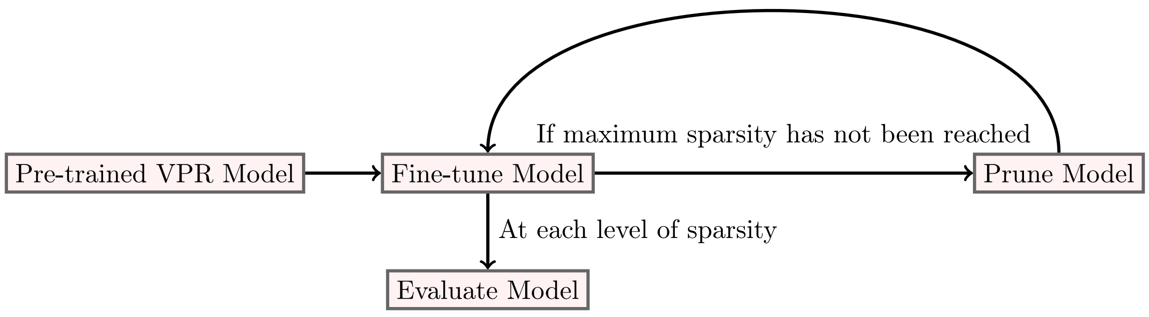
The 'Lottery Ticket Hypothesis' claims that "dense, randomly-initialized, feed-forward networks contain sub networks (winning tickets) that — when trained in isolation — reach test accuracy comparable to the original network in a similar number of iterations" [27]. This suggests that networks can be trained from sparsity when the learning rate and weights are reverted to how they were at epoch 0. However, in [28], it is claimed that networks can be trained from this sparse state, even with randomly initialised weights. The paper suggests that the act of pruning end up being a search for an appropriate network structure that is able to perform at the same level as the over-parameterised network that was started with.

3 Experiment Design

3.1 High Level Design

The experiment will follow the process shown in 1, whereby the selected VPR methods will be pruned by the selected methods and evaluated. The overall pruning process involves pruning away a small amount of the network and then fine-tuning the resulting pruned network. After each pruning/fine-tuning step the network will be evaluated for several metrics, giving a picture of the networks performance at different sparsities for each pruning method. This should then allow a comparison of the different methods, networks and image representation methods across levels of sparsity.

Figure 1: High-level Experiment Design



Pre-trained VPR Model
Fine-tune Model
If maximum sparsity has not been reached
Prune Model
At each level of sparsity
Evaluate Model

3.2 Image Representation Methods

"The most significant part of a visual place recognition system is image representation" [1]. Whilst the feature maps of the outputs of intermediate layers of CNNs can be flattened and used as the image representations, the performance of this method is less competitive due to high dimensionality and a lack of generalisation [1]. A now common method in image retrieval tasks, such as VPR, is to employ some method that can produce an image representation more suitable for image matching from the output of a CNN.

Two different methods of image representation were chosen in this investigation: Generalised Mean (GeM) pooling [29] and NetVLAD [30]. These methods were chosen for the investigation as they have both been, at points, state-of-the-art methods for image representation in VPR and are well known in the field. These layers were also chosen as they have some learnable parameters. Hopefully, this will allow them to adapt to the changing CNN structure and output, giving a better chance for the models to maintain their accuracy through the pruning process. Furthermore, the dimensionality of these descriptors, for a given network architecture, is different, with the dimensionality of the descriptor output from GeM being strictly equal to or smaller than that of NetVLAD. This may also then provide some information on the effects of pruning on representation methods that compress the output of their CNN to different levels, which is important, as smaller image descriptors in turn leads to faster similarity search between the query and database images, a desirable trait in the setting of this investigation.

3.2.1 NetVLAD

NetVLAD can be described as a 'Deep Image Representation' in that it learns parameters during the training phase of the CNN it is attached to, rather than treating all feature map outputs of the CNN as equally 'important'.

NetVLAD employs a pooling layer inspired by the Vector of Locally Aggregated Descriptors (VLAD) [31] typically used with local features such as SIFT [6]. Which, using a set $C = \{c_1, ..., c_k\}$ of k learned visual words with k-means from k-means clustering, creates an image descriptor by accumulating, for each visual word c_i, the differences $x - c_i$ of the local descriptor x assigned to its nearest neighbour: c_i. Given x being of dimension D, the resulting image representation is $v_{i,j}$, where $i = 1...k$ and $j = 1...d$ respectively index the visual word and local descriptor component. Thus, the (i, j) element of v is obtained as follows [31]:

$$v_{i,j} = \sum_{x\, with\, nearest\, neighbour\, c_i} x_j - c_{i,j}$$

NetVLAD adapts this to be compatible with back-propagation, by making the operation of assigning x to a cluster center c_i differentiable. It does this by using a soft assignment of the descriptor to multiple clusters, assigning the weight of the x to cluster c_i proportional to their proximity but relative to proximities to other clusters[30]. This results in an aggregation method with a set trainable parameters for each cluster. Intuitively, the parameters allows for the moving of a cluster's 'anchor' (the origin of the coordinate system within the cluster), to better separate non-matching images.

NetVLAD has been shown to achieve a *recall*@1 on the *pitts*30*k* dataset of around 85% (VGG16 CNN architecture) [30].

3.2.2 GeM

Pooling layers pool the activations of convolution layers into compact image representations typically with a dimension of the number of channels in the corresponding layer of the CNN.

The output of a CNN convolution layer can be represented as a 3D tensor X of dimension $W \times H \times K$ where K is the number of feature maps in the layer. Let X_k be the set of $W \times H$ activations for feature map $k \in \{1...K\}$. Pooling layers take X as an input and produce a vector $\mathbf{f}$, the feature descriptor. In the case of global max pooling this vector is given by:

$$\mathbf{f} = [f_1...f_k...f_K], \; f_k = \max_{x \in X_k} x$$

for average pooling:

$$\mathbf{f} = [f_1...f_k...f_K], \; f_k = \left(\frac{1}{|X_k|} \sum_{x \in X_k} x \right)$$

and in the case of GeM $\mathbf{f}$ is given by the equation:

$$\mathbf{f} = [f_1...f_k...f_K], \; f_k = \left(\frac{1}{|X_k|} \sum_{x \in X_k} x^{p_k} \right)^{\frac{1}{p_k}}$$

The parameter p_k can be set or learned, via back-propagation, for each feature map, as the operation is differentiable. Global max pooling and average pooling can be expressed as special cases of GeM pooling when $p_k \to \infty$ for max pooling and $p_k = 1$ for average pooling[29].

3.3 Network Architectures

This investigation chose to look at three separate CNN architectures. These being ResNet[32], EfficientNet[33] and MobileNet[34]. These specific models were chosen for various reasons.

The ResNet architecture was chosen as it is a well-known and performant network. Because of this, it appears in many papers surrounding the fields of both network pruning and VPR. This gives a reference to compare our results against and can inform conclusions surrounding successful compression of our models.

MobileNets are "a class of efficient models for mobile and embedded vision applications"[35] with MobileNetV3 being the newest in the family. The choice to include this network architecture is due to their design being geared towards mobile and embedded vision applications, of which VPR is often one. Along with this, V3 is tuned to run on mobile CPUs, allowing this model to effectively deployed to a variety of systems which is in concordance with the goals of this investigation.

Similarly, EfficientNet is the result of designing CNNs to be scalable, in order to match the computational limits of the system they're in. Once again, this aligns with the goals of the investigation and provides us with an already resource efficient architecture that has the possibility of being compressed further through pruning.

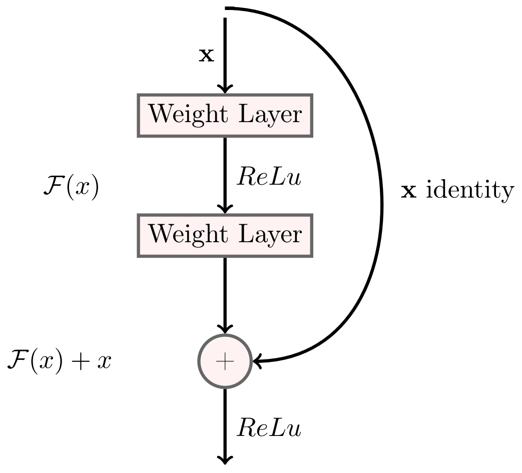
All the models were taken with pre-trained weights from PyTorch and then fine-tuned on our VPR-specific dataset.

3.3.1 ResNet

The specific model used in this investigation is ResNet18 with the aggregation layers taking the output from the 5^{th} and final convolutional layer.

The ResNet architecture was designed in order to be able to effectively train deeper CNN networks. In [32], it is noted that deep networks suffer a degradation problem, whereby the "accuracy [of the network] gets saturated and then degrades rapidly" [32]. To address this a deep residual learning framework is used:

Figure 2: Skip connections within ResNet



Instead of hoping each few stacked layers directly fit a desired underlying mapping, the layers are allowed to fit a residual mapping. With the desired underlying mapping being $\mathcal{H}(\mathbf{x})$, the stacked nonlinear layers are allowed to fit another mapping of $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$, meaning the original mapping is recast into $\mathcal{F}(\mathbf{x}) + \mathbf{x}$ [32]. "The formulation of $\mathcal{F}(\mathbf{x}) + \mathbf{x}$ can be realized by feedforward neural networks with "shortcut connections""[32] as shown in 2. These shortcut layers perform identity mapping and their outputs are added to those of the stacked layers.

The ResNet18 model and most up-to-date weights used in the investigation contains 11.7 million parameters and 1.81 gigaflops in its feedforward computation. Accuracy@1 on ImageNet-1K is reported to be 69.758. (Flops, parameters and performance figures taken from https://pytorch.org/vision/stable/models.html)

3.3.2 MobileNetV3

MobileNets are a family or networks that use depth-wise separable convolutions to reduce the computation required during inference. The depth-wise separable layers "replace a full convolutional opera- tor with a factorized version that splits convolution into two separate layers."[36]. The first layer applies one convolution per input channel for lightweight filtering. The next layer is known as a pointwise convolution, having dimension 1x1. This layer builds "new features through computing linear combinations of the input channels" [36].

During standard convolution, an input tensor with dimensions $h_i \times w_i \times d_i$ is convolved with a kernel $K \in \mathcal{R}^{k \times k \times d_i \times d_j}$ to produce an output tensor in the shape $h_j \times w_j \times d_j$. The computational cost of this layer would be: $h_i \cdot w_i \cdot d_i \cdot d_j \cdot k \cdot k$. The depthwise separable convolutions only cost: $h.w_i \cdot d_i(k^2 + d_j)$ and can be used as a direct replacement for standard convolutional layers with similar performance [36]. This is a reduction in computational cost of almost k^2.

The models can also be optimised for specific platforms using a platform aware network search, finding the optimal global structure and number of filters per layer [35].

The MobileNetV3-Large model and most up-to-date weights used in the investigation contains 5.5 million parameters and 0.22 gigaflops in its feedforward computation. Accuracy@1 on ImageNet-1K is reported to be 75.274. (Flops, parameters and performance figures taken from https://pytorch.org/vision/stable/models.html)

3.3.3 EfficientNet

The EfficientNet architecture is a result of the use of a compound scaling method, that scales network width, depth and resolution to reach some level of computational requirement.

The EfficientNet_B3 and most up-to-date weights used in the investigation contains 12.2 million parameters and 1.83 gigaflops in its feedforward computation. Accuracy@1 on ImageNet-1K is reported to be 82.008. (Flops, parameters and performance figures taken from https://pytorch.org/vision/stable/models.html)

3.4 Pruning Methods

"Neural network pruning is an art of removing “unimportant weights” from a model" [37]. However, defining the 'importance' of a weight is still an unclear notion. The aim of a pruning method is to be able to choose the least 'important' weights within a network and remove them.

Structured pruning (the focus of this investigation) removes not single weights but entire filters from the network. This allows for acceleration and model compression using standard hardware. The form of structured pruning used in these experiments is group-level pruning. This method is used as, in many CNN architectures, "parameters from different layers are inherently dependent on each other"[38], meaning these parameters need to be pruned at the same time - in a group. This means that the importance scores used to decide on the pruned structures are the mean importance of structures within a group. The python library used to facilitate this is Torch-Pruning, a tool designed along-side the paper [38].

This investigation covers four weight-based pruning methods for the standard train-prune-fine tune paradigm. Other categories of methods exist, but weight-based methods methods don't require input-data to calculate importance and therefore have lower computational costs, whilst still being able to achieve high performance - hence their choice for the investigation.

3.4.1 L1 Norm & L2 Norm

Structured filter norm pruning uses the norm values of the filters as their 'importance'. The l_p-norm of a filter can be defined as:

$$\|\mathbf{F}_i^l\|_p = \sqrt[p]{\sum_{n=1}^{N_l} \sum_{k_1=1}^{K_l} \sum_{k_2=1}^{K_l} |\mathbf{F}_i^l(n, k_1, k_2)|^p}$$

where $i \in N_{l+1}$ represents the i-th filter in the l-th layer, N_l is the input channel size and K_l is the kernel size [12]. The parameter p is the order of the norm. The values of p used in this investigation are 1 and 2 - the most commonly used values in pruning.

3.4.2 LAMP

Layer-adaptive sparsity for magnitude based pruning is a method by which layer-wise sparsity is chosen in conjunction with magnitude based pruning to achieve better results than magnitude pruning alone [37]. The importance score used in LAMP "is a rescaled version of weight magnitude that incorporates the model-level l_2 distortion incurred by pruning" [37].

A feedforward neural network with depth d has the weight tensors $W^{(1)}...W^{(d)}$ associated with each convolutional layer. For these 2d convolutional layers, the tensors are four-dimensional. Initially, these tensors are flattened so the method can work with both fully connected and convolutional layers. For each of these flattened vectors, "the weights are sorted in an ascending order according to the given index map" [37]. The LAMP score for the uth index of the weight tensor W is then defined as:

$$\text{score}(u; W) := \frac{(W[u])^2}{\sum_{v \geq u} (W[v])^2}$$

The connections with the lowest LAMP score are then pruned up to the desired level of sparsity. "Informally, the LAMP score measures the relative importance of the target connection among all surviving connections belonging to the same layer" [37]. LAMP was a method designed for unstructured pruning but has been adapted in Torch-Pruning to work in a structured manner.

3.4.3 FPGM

Filter Pruning via Geometric Median (FPGM) shows that a smaller norm is not always 'less important' in the context of pruning, as the effectiveness of norm pruning is contingent on: "the norm deviation of the filters being large and the minimum norm of the filters being small" [39]. Rather than pruning filters with 'less importance' FPGM seeks to remove filters with redundancy.

The pruning method is as follows: given a set of n points $a^{(1)}...a^{(n)}$ with each $a^{(i)} \in \mathbb{R}^d$, find a point $x^* \in \mathbb{R}^d$ that minimizes the sum of the Euclidean distances to them:

$$x^* = \arg\min_{x \in \mathbb{R}^d} \text{ where } f(x) = \sum_{i \in [1,n]} \|x - a^{(i)}\|_2$$

where $[1, n] = \{1...n\}$. This notion of geometric median can estimate centrality for data in Euclidean spaces, so can be used to get the common information for all filters in the single ith layer of the network:

$$x^{GM} = \arg \min_{x \in \mathbb{R}^{N_i \times K \times K}} \sum_{j' \in [1, N_{i+1}]} \|x - \mathcal{F}_{i,j'}\|_2$$

where N_i and N_{i+1} represent the number of input and output channels of the ith layer of the network, and $\mathcal{F}_{i,j}$ represents the jth filter of the ith layer, with dimension $\mathbb{R}^{N_i \times K \times K}$.

Then, the filter to prune in the ith layer is the one closest to the geometric median of that layer:

$$\mathcal{F}_{i,j^*} = \arg\min_{\mathcal{F}_{i,j'}} \|\mathcal{F}_{i,j'} - x^{GM}\|_2, \text{ s.t } j^{'} \in [1, N_{i+1}]$$

This filter is chosen as it can be represented by the other filters, so pruning it from the network should have little negative impact on its performance [39].

3.5 Dataset

The dataset chosen for training and testing in this investigation is knows as pitts30k, a subset of the larger pitts250k dataset [40], with the choice to use the smaller of the two datasets coming down to practicality: a smaller dataset provides faster training times and takes up considerably less disk space, allowing for the running of more individual experiments, whilst still being large enough to train the models on.

The pitts dataset was also chosen as the dataset is widely used in VPR model evaluation, this allows for comparison of our models with others in separate papers to gain better reference for their performance.

Furthermore, the dataset encompasses some of the main challenges faced by VPR models in achieving high performance. Namely, the dataset contains images which provide both illumination and viewpoint changes, with images being collected in multiple seasons and with multiple view directions [1]. This means that the dataset is more representative of the real world conditions that a deployed VPR system might face, making it directly relevant to us investigating VPR systems so that they *can* be deployed.

The pitts30k model uses 10,000 images each for training, validation and testing.

3.6 Evaluation methods

The evaluation of a VPR model can't be performed with just one measurement. When looking at deploying VPR systems into real world applications, a number of factors must be considered.

Recall@k is certainly the most important, and is the main metric by which VPR models are measured. Recall@k measures the proportion of correct suggestions in the top k suggestions provided by the model. $k = 1$ is the most common value for k and is the measurement of what percentage of the time the model's best matching database image is a correct match. As the very function of the VPR model depends on its ability to correctly match images, this metric must be considered in the investigation.

Inference time is also an important measure for VPR systems that will be used in the real world. Many VPR applications depend on the ability of the system to match images and a high rate. The time taken for this to happen is dependent on both parts of the system: feature extraction and representation, and image matching. While inference time only effects the former of the two parts, it is still relevant to the whole time taken by the system and within these experiments image matching time will remain constant within image representation methods.

Inference-time is also generally representative measure of the computational requirement of a network. This is pertinent to the investigation as a reduction in computational requirements of the systems is a goal of pruning and by measuring this we can see if that has been achieved by the pruning.

Peak Memory Usage was chosen to be measured as it indicates the ability for a given system to use a given VPR model. If the memory required is too high then the system would not be able to run that model. By measuring this across levels of sparsity we will be able to see the change in memory usage, and hopefully lower the memory requirements so that a wider range of systems can use the VPR methods tested.

Params and MACs both provide and insight into the structure of the network. They are both relevant to the computational requirements of the network but collecting the data on these statistics may also provide some insight into the practical function of the different pruning methods used, perhaps informing further research into the topic.

3.7 Implementation

The implementation of the experiments was done wholly in python using the PyTorch framework for all of the models. As stated previously, the pretrained models were downloaded from https://pytorch.org/vision/stable/models.html . These models then had the aggregation layers appended to their final convolutional layer to produce the VPR models that were then used in the investigation.

The implementation of these aggregation layers and the training process of the system was taken and adapted from https://github.com/gmberton/deep-visual-geo-localization-benchmark, the accompanying code repository to [41]. This repository was also used to evaluate the recall performance of the network.

For the pruning of the networks, the Torch-Pruning library from https://github.com/VainF/Torch-Pruning was used [38]. This library came with the pruning methods that were evaluated in the investigation built-in. This library also provided the tools to benchmark the inference time and memory usage of the models.

For the collection of the other model parameters, torch-scan was used, which is a PyTorch implementation of tf.Keras.model.

The system was designed around these tools and has integrated them, allowing for easy training, pruning, fine-tuning and evaluation in on python command. The system also has a large amount of parameters that can be selected for each run, relating to all parts of the run. Pruning method; network aggregation; pruning-step size and network architecture are just a few of the most important ones within the system, and can be chose with just one command-line argument. An example of the command for one run would be:

```
python main.py -run_type=tpe -run_path=../pruning -backbone=mobile -aggregation=netvlad
-datasets_folder=../datasets_vg/datasets -pruning_method=l2_norm -max_sparsity=1
-pruning_step=0.1 -train_batch_size=4 -recall_curve=True
```

The system is also easily extensible to custom pruning methods; new network architectures; different datasets and network aggregations, needing little more than the implementation of these in-line with the design of the current methods.

All evaluations were run on the same machine in order to get consistent results for inference time and memory usage. The machine used and NVidia RTX 6000 Ada GPU with 48GB of VRAM. However, the changes in inference time and memory usage were also observed on a separate RTX 3090 with 24GB of VRAM. For the evaluation of inference time and memory usage, a batch size of 512 was used as much smaller sizes provided inconsistent results between runs. This fact may be relevant to 'online VPR' where each query image is processed separately (batch size of 1) and further research could be done on this.

All evaluation was done using `model.eval()` and `with torch.no_grad()`.

3.8 Justification of Approach

The approach of this investigation can be seen as justified as it follows the process typically employed by papers investigating pruning methods on CNNs. Many papers that explore methods of network pruning use a comparison of network performance at sparsity to justify the viability of the approach and typically these are the principle metrics by which VPR and other computer vision techniques are evaluated. Furthermore, the choice of VPR methods and pruning techniques is appropriate as they have at one point been state-of-the-art or near state-of-the-art and perform similar to other methods in their category. Thus, the investigation should provide relevant data and inform real-world implementations of pruning networks for VPR.

4 Results

In this section, the most pertinent results from the experiments will be displayed. These results are discussed further in the following section.

Unless otherwise stated, the dataset used to measure recall performance is the *pitts*30*k* dataset.

4.1 Recall

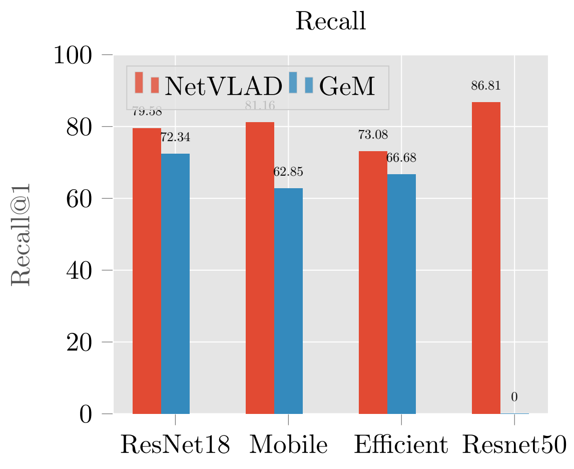


Figure 3: Baseline Model Performance

It can be seen in both graphs in 3 that the performance of the VPR models that use the NetVLAD aggregation level is higher than those which use GeM. Interestingly MobileNetV3 outperforms ResNet18 and EfficientNet using the NetVLAD layer, but is the worst of the three when using GeM. Resnet50 has significantly higher accuracy than the other models, indicating that model depth is a factor in performance.

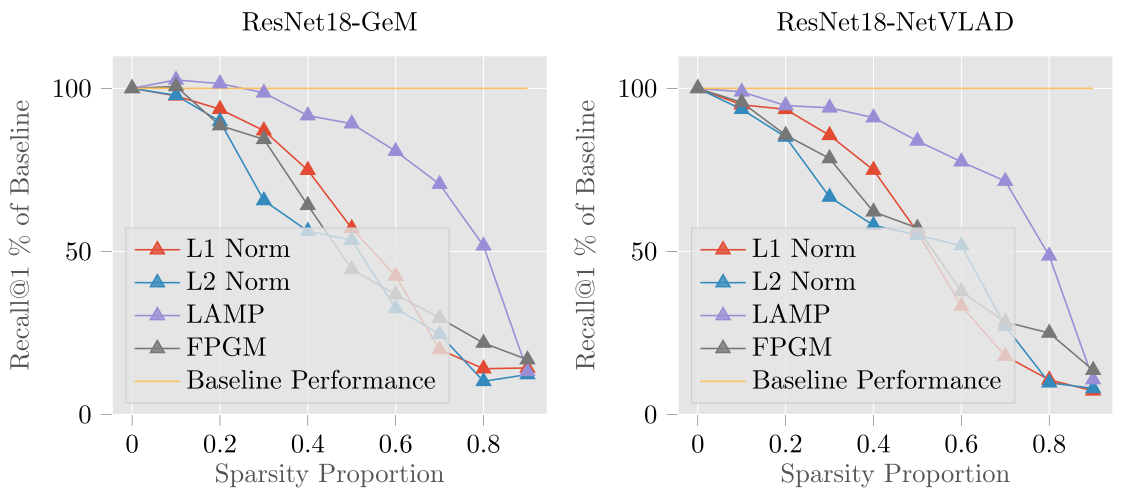


Figure 4: ResNet18 Pruning

The figures 4, 5, 6 and 7 show the performance of the pruned models at different levels of sparsity having been pruned with the four chosen pruning methods.

It can be noted that the models using MobileNetV3, EfficientNet and ResNet18 all gain performance over their baseline at certain levels of sparsity in at least one of the experiments. It's also clear, for all models, that past a certain - generally high - level of sparsity, the performance of the model drops considerably, and they effectively cease being able to perform their desired function.

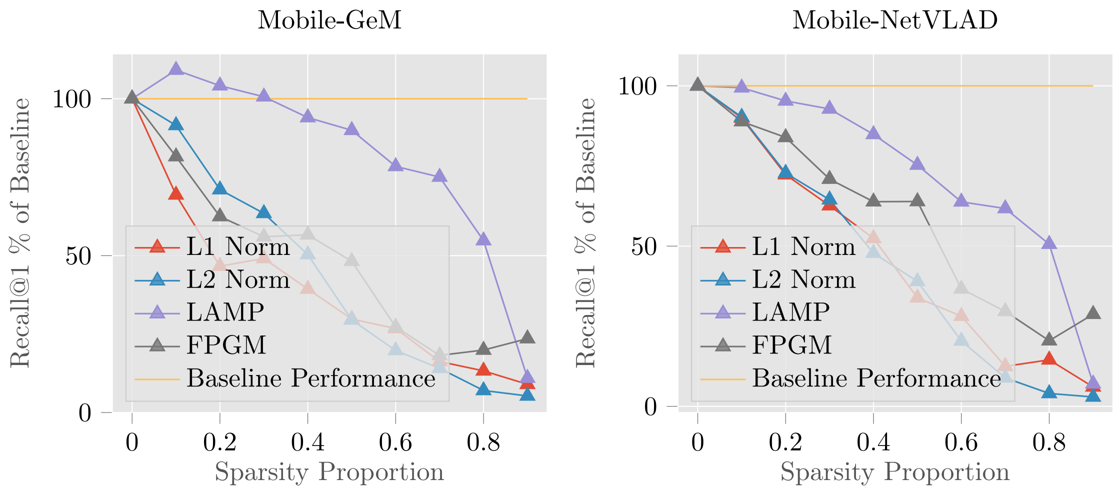


Figure 5: MobileNetV3 Pruning

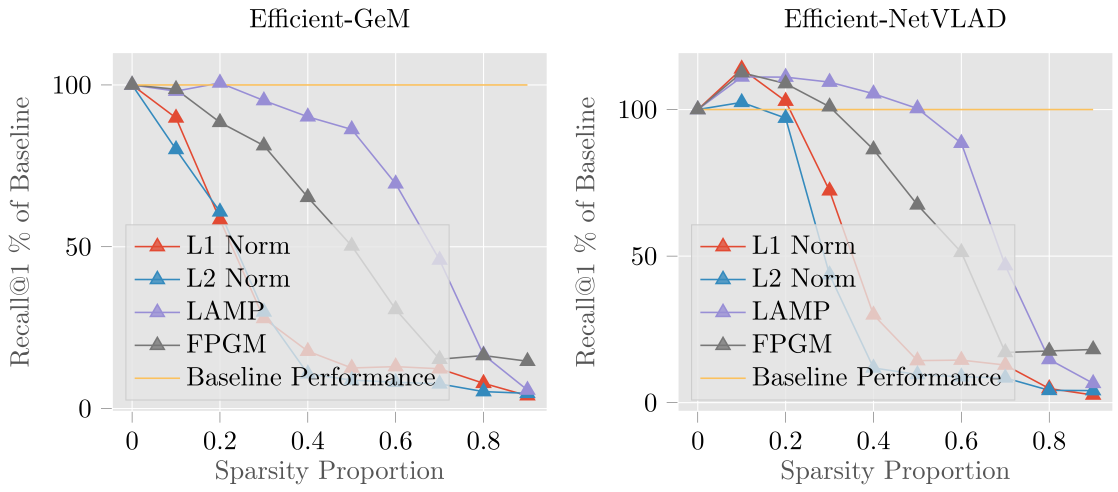


Figure 6: EfficientNet Pruning

The difference in performance loss between all three experimental variables can be seen, with the pruning method used appearing to have the largest impact on model performance. Of these pruning methods, LAMP significantly outperforms the others in all tests, with the models pruned using this method maintaining their performance at higher levels of sparsity far better than the others.

Notable also, is the difference in receptiveness of the CNN architectures to pruning. At levels of sparsity below 0.5, the network architecture that fairs best in a given set of circumstances varies. However, past this point, generally, ResNet18 performs the best, as its drop off in performance is more gradual than those of the other two networks.

Differences in performance between aggregation layers, however, is harder to characterise. In the EfficientNet model, all pruning methods perform better using the NetVLAD aggregation. However, in the MobileNet and ResNet models, LAMP pruning performs better with the GeM layer than the other. For the other pruning methods, though, the models using NetVLAD seemed to produce better results.

The pruning run performed on ResNet50 has also been included in 7. This was to see if a deeper network of the same architecture would respond differently to pruning. This does appear to be the case with the L2 Norm pruning performing much better than on its shallower variant - ResNet18.

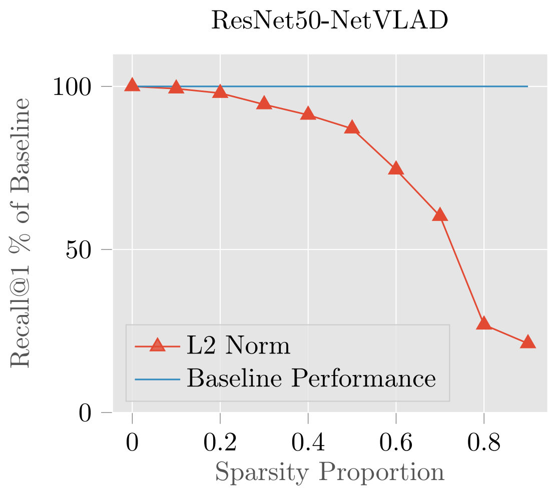


Figure 7: ResNet50 Pruning

4.2 Inference Time

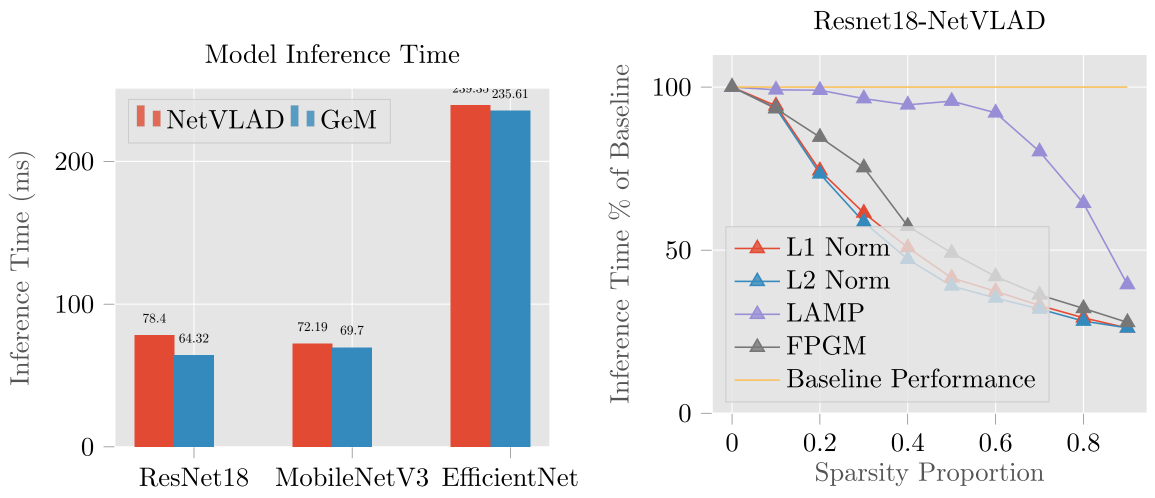


Figure 8: Inference Time

The baseline inference time for the networks can bee seen in 8. ResNet18 and MobileNetV3 take comparable amounts of time, although interestingly, ResNet is slower than Mobile with NetVLAD and faster with GeM. Also notable is the far greater inference time of EfficientNet, even though it is smaller in size than ResNet18.

The inference time characteristics across levels of sparsity are largely similar across network architectures and aggregations. As such, only the graphs for the VPR methods with NetVLAD aggregations have been included in 8 and 13. The networks with GeM aggregations do achieve slightly larger percentage decreases due to the GeM layer taking up less of the total inference time; the trends, however, are the same.

From these graphs it can be seen that all pruning methods provide, at least, some reduction in the inference time of the network. The obvious difference is in the data is the significantly lower reduction which the LAMP pruning method provides the network. For all networks, LAMP only provides a speedup equivalent to the other methods at sparsities around 30-40% higher than them. For the other pruning methods, the reduction in inference time is similar. This suggest that the structure in the networks removed by LAMP pruning are less computationally intensive than

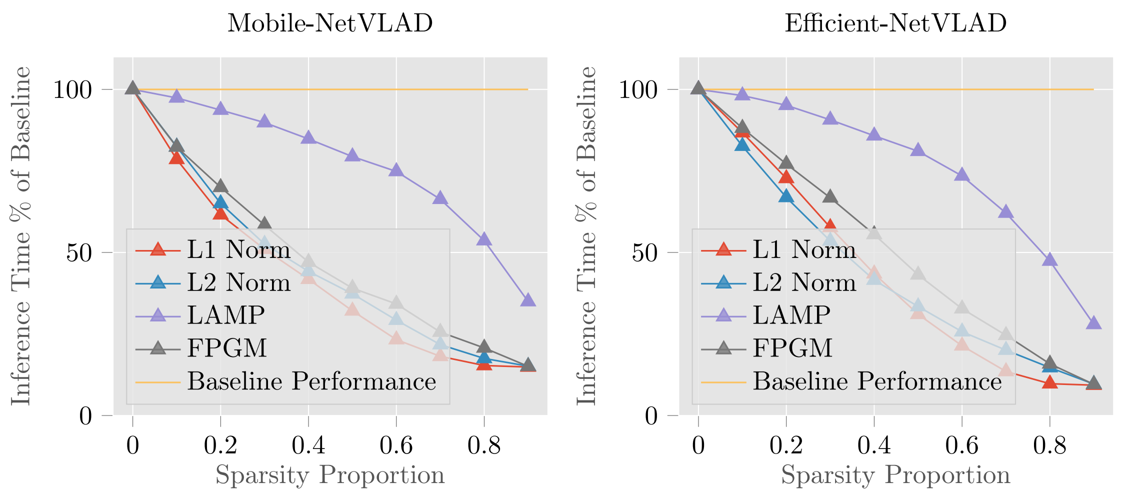


Figure 9: MobileNetV3 and EfficientNet Inference Time

the others. For earlier levels of sparsity, inference time decreases over 10% of baseline for a 10% increase in sparsity, although it is slightly less for the ResNet models.

4.3 Model Size

To inform us about why the pruning methods provide the speedups they do, it is also pertinent to look at the number of parameters left in the models once they have been pruned.

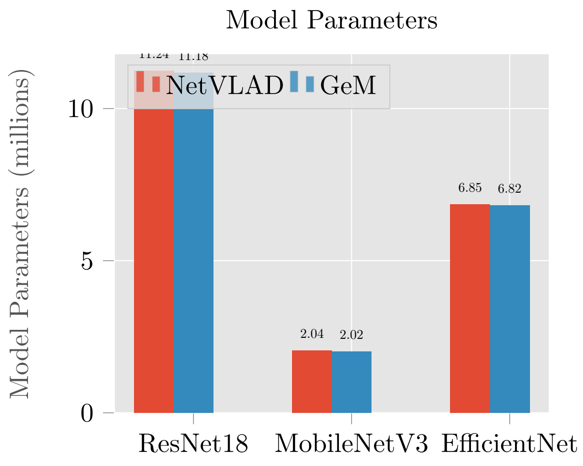


Figure 10: Model Baseline Parameters

The number of parameters of the baseline models can be seen in 10. The models are all of distinctly different sizes with ResNet18 having roughly 5× the number of parameters as MobileNetV3 and just under 2× that of EfficientNet.

The graphs in 11 are representative of all the experiments done. It can be seen that the number of parameters in each model decreases practically linearly and at the same rate as the increase in sparsity. Of note here, though, is the clear that the LAMP pruning method appears to be removing a slightly higher number of parameter than the other methods. This appears contradictory to the fact that it provides less of a speed-up than the other methods.

4.4 Computation

The relative number of MACs in each network in 12 is similar to that of parameters.

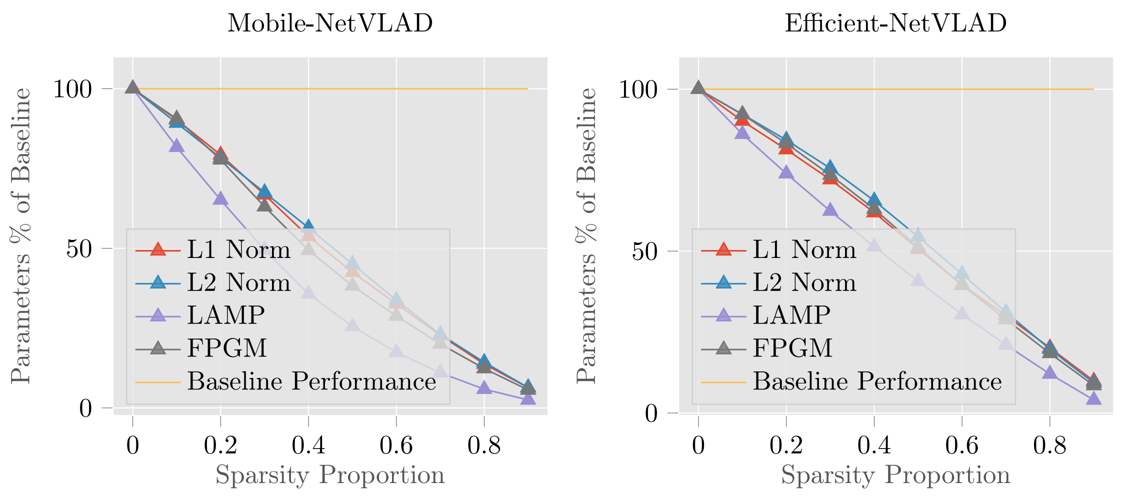


Figure 11: MobileNetV3 and EfficientNet Parameters

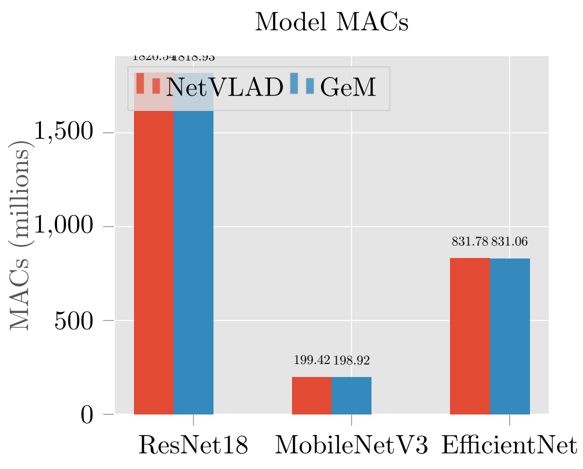


Figure 12: Model Baseline MACs

Similar to the number of parameters, the number of MACs scales almost linearly with sparsity. However, in contrast to the parameter data, the LAMP pruning method reduces the number of MACs in the forward pass of the networks less than the other pruning methods. This correlates with the inference time results, as it would be expected that a network that requires more computations in its forward pass would take a longer time to compute its output.

4.5 Memory Usage

In line with the inference time of the models, the baseline memory usage of EfficientNet is far higher than the other two networks, with ResNet18 and MobileNet being fairly similar 14.

The peak memory usage trends shown in 7.2 and 16 follow for all the experiments run.

From this data it can be seen that the structured pruning of the CNNs in the VPR models can reduce the peak memory usage at inference-time. Once again, LAMP is the outlier among pruning methods, failing to meaningfully reduce the peak memory requirements of the model until high levels of sparsity. The three other methods, as in other metrics, all follow a simliar trend of reducing the peak memory usage of the model. This would also be expected given the inference time and MACs data.

For all models, the reduction in memory requirement is greater at lower levels of sparsity and there are less gains to be made pruning to higher levels, except in the case of LAMP.

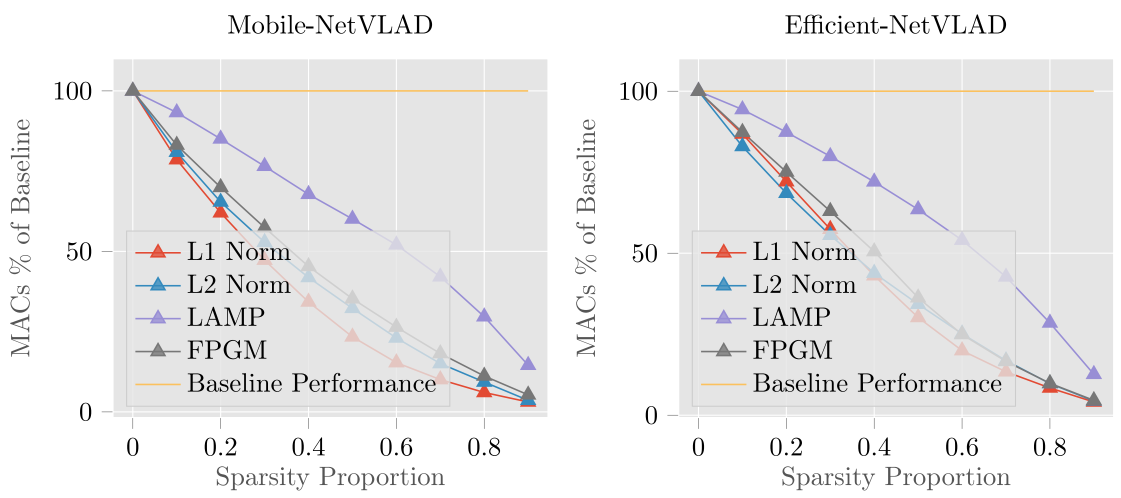


Figure 13: MobileNetV3 and EfficientNet Parameters

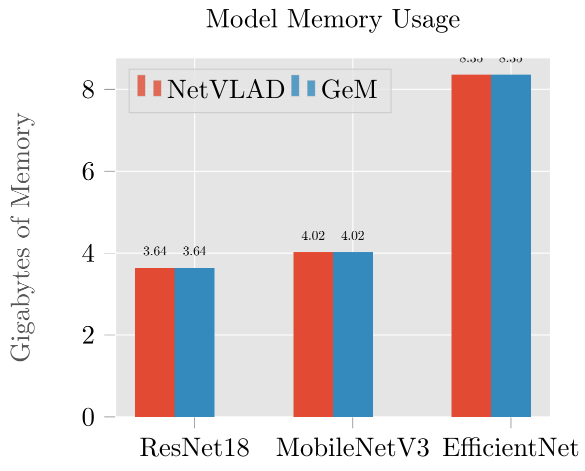


Figure 14: Model Baseline Memory Usage

It appears that MobileNetV3 has the largest drop in peak memory requirement, compared to EfficientNet and ResNet18, at least at lower levels of sparsity, after which the change in memory flattens out.

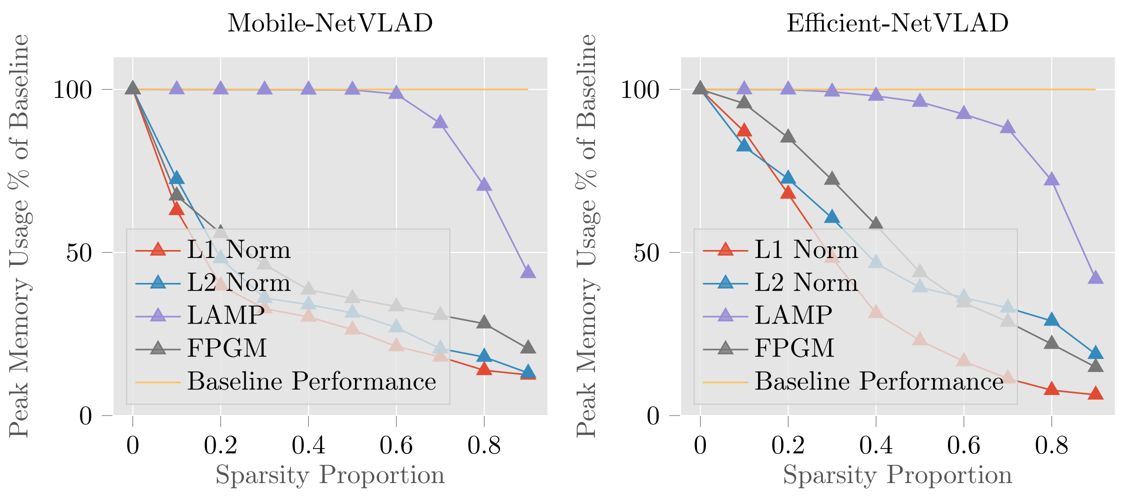


Figure 15: MobileNetV3 and EfficientNet Memory Usage

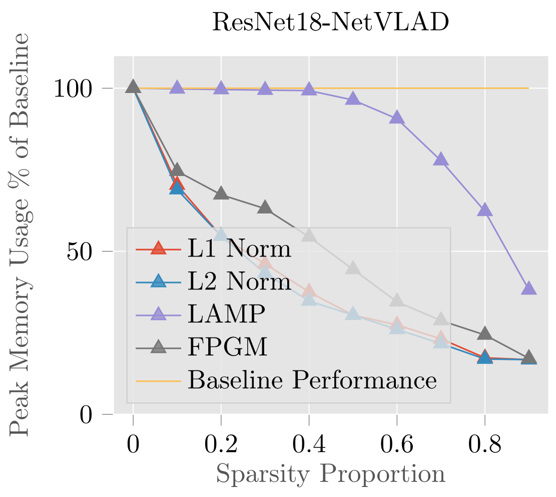


Figure 16: ResNet18 Memory Usage

5 Evaluation of Results

The implications of the results in the section above are discussed in this section.

5.1 Recall

The data taken from all the experiments show a clear change in the recall of a model due to its sparsity. Whilst most models do, at some point, gain performance over their baseline, on average an increase in sparsity leads to a decrease in performance.

The drop off in performance from the pruning of these models is quite high compared to examples of CNN pruning in other papers. Obviously, this investigation is working in a different problem space, but it still a notable difference. For example, in [42] ResNet18 is prune up to 40% of its flops, with only a reduction in top-1 accuracy of less than 2%. This is in contrast to our model, where performance drops to 78% of baseline performance at a similar level of reduction in network computation.

Given that we have shown that changing the aggregation layer of a model, changes how it responds to pruning, it would be expected that having an aggregation layer would make the performance of the model change as well. The results of this investigation suggest that the change it causes is a reduction in the ability of a model to maintain its accuracy at sparsity. The performance at sparsity may be lower than the non-VPR counterparts but this may be less of an issue for the VPR systems. This is because, in SLAM, the capability of the system isn't entirely dependent on the accuracy of the VPR model. This is because "since the advent of robust graph optimization techniques for SLAM, the avoidance of wrong loop closures has become less relevant. With robust graph optimization, it is more important to find enough correct loop closures than to avoid wrong loop closures" [4]. This may mean that regardless of a certain level of decrease in performance, the model may well still be viable and the faster, pruned network, would be more suitable than the slower, more accurate one.

5.2 Inference Time

A decrease in inference time of the models at sparsity is one of the key objectives of pruning the VPR-models. It can be seen that all pruning methods achieve this, and provide a gain in speed that can at least be harnessed by CUDA GPUs. The speed-up gained by these networks is certainly significant, with the inference time being halved at 40% sparsity in some experiments.

Although the pruning methods have proven they can speed up the network, it remains to be seen if the trade off between speed and performance is one that makes sense for any VPR use-case. However, given that pruning does speed up inference time, there is now incentive to find or create other pruning methods that will provide better accuracy given that they can also provide a speed up.

The collection of the timing results used a batch size of 512. This was chosen as, with batches smaller than this, the results for inference time and memory usage were erratic and showed little pattern. This may impact the number of use cases that see a speed up in inference time as 'online' VPR-systems that process one image at a time may not see a benefit. However, the phenomenon wasn't explored conclusively, and further changes to settings may have still produced a speedup at increasing sparsity for small batch numbers.

It's interesting that the LAMP pruning method produced far less of a speedup than the other methods. Although it can't be said for sure, this may be due to its ability to choose layer-wise sparsity. This means that at lower levels of sparsity, it would be able to aggressively prune certain layers far more than others - and in this case, those layers would be contributing far less to the total inference time of the network than others. This would allow it to prune away large amounts of parameters whilst still keeping the inference time high.

The initial inference time for EfficientNet is also far higher than the other two networks, despite being having fewer params and MACs than Resnet18. This may have been due to the convolutional layer that the aggregation layers were appended to was of high dimension. As the aggregation layers scale to the size of the output layer, this would mean that they would be far larger on EfficientNet than the others. With that being said, this wasn't verified, as the differences in inference time and memory usage were only noticed after all the experiments had run.

5.3 Memory Usage

All pruning methods showed a reduction in peak memory usage as sparsity increases. This is inline with the goals of pruning and will help make deploying VPR models on different systems possible. For example EfficientNet-NetVLAD took over 8 gigabytes of memory to run at baseline but, at 30% sparsity, it used only 6. This reduction illustrates the benefit of this pruning as 8 Gb is an often used amount of VRAM in graphics cards. This means that a common 8 Gb graphics card wouldn't have been able to run the baseline model, however, it would be able to use the pruned model, with no loss of accuracy. This demonstrates the benefit that pruning can provide and that the reduction in memory that this investigation has shown is meaningful in allowing a larger number of computers to run certain VPR-models.

6 Conclusions and Further Work

The results in the sections above show that the pruning of CNNs *can* provide a reduction in the computational requirements of a VPR system whilst maintaining an appropriate level of precision with certain combinations of pruning method, CNN architecture and network aggregation. For example, EfficientNet-NetVLAD, pruned to 30% sparsity with FPGM, has the same recall performance as its baseline, but an inference time 33% lower and peak memory usage of 28% lower than baseline.

The research has also shown that all 3 of the tested variables have an effect on the performance of pruning. Along with depth of specific architecture also indicating it has an impact. Given that there exist a very large number of combinations of the three, that remain untested, it's plausible to imagine that more performant combinations exist that will make the pruning of CNNs in VPR more viable that it has already shown to be.

Further work may be taken to look at the viability of one-shot pruning methods as the training time for these pruned networks becomes quite long. For reference: the time taken to run these experiments was in excess of 1000 hours on an RTX 4090. Although training time doesn't effect the final system, it may make those implementing VPR systems less willing to use pruning due to the extra time it would take them.

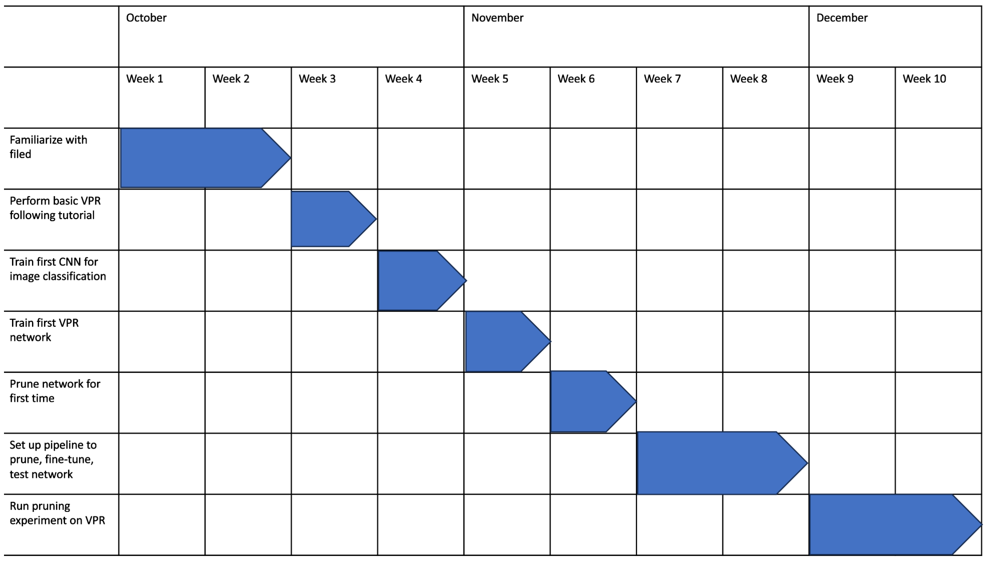
The the performance at sparsity of the VPR systems is notably lower than that of the networks in the literature on this topic, this may be due to errors within my implementation. Further work on this system would provide more certainty of the results, a possible free gain in performance and also allow further research into the topic of pruning CNNs for VPR.

Overall, I would say that the goals of this investigation have been met, given that, in testing, the desired effects of pruning a network for VPR have been shown, along with evidence to showing that these pruned networks would possibly be able to be deployed on a wider range of systems in the real-world.

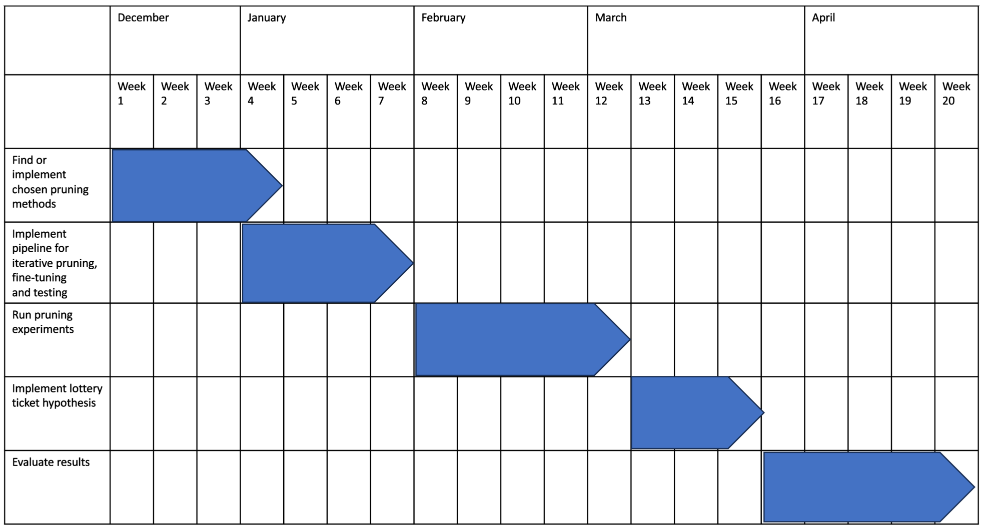
7 Planned Schedule and Actual Schedule of Work

7.1 Planned Schedule of Work

This schedule of work has been taken directly from the progress report submitted in December.

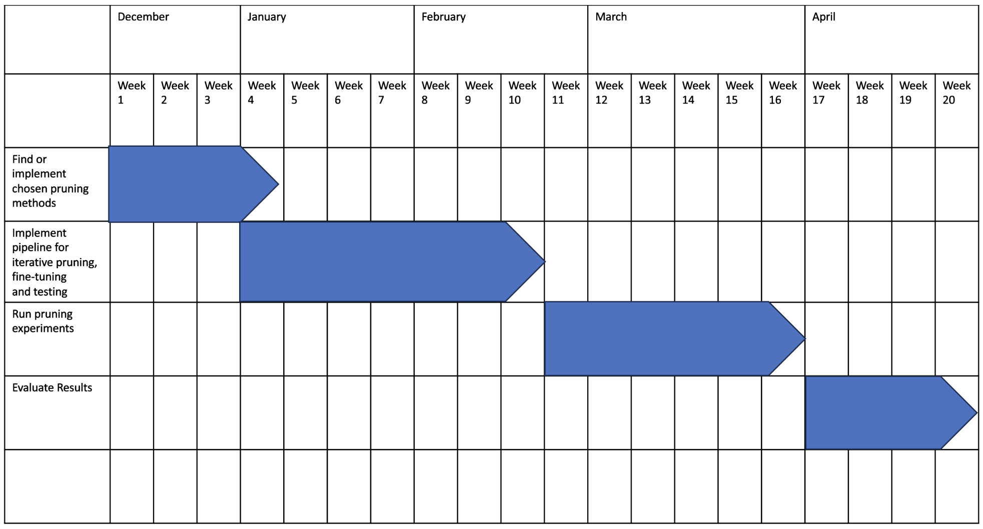


	October				November				December	
	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10
Familiarize with filed	🟦	🟦								
Perform basic VPR following tutorial			🟦							
Train first CNN for image classification				🟦						
Train first VPR network					🟦					
Prune network for first time						🟦				
Set up pipeline to prune, fine-tune, test network							🟦	🟦		
Run pruning experiment on VPR									🟦	🟦



	December			January				February				March					April			
	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16	Week 17	Week 18	Week 19	Week 20
Find or implement chosen pruning methods	🟦	🟦	🟦	🟦																
Implement pipeline for iterative pruning, fine-tuning and testing				🟦	🟦	🟦	🟦													
Run pruning experiments								🟦	🟦	🟦	🟦	🟦								
Implement lottery ticket hypothesis													🟦	🟦	🟦					
Evaluate results																🟦	🟦	🟦	🟦	🟦

7.2 Actual Schedule of Work



References

[1] X. Zhang, L. Wang, and Y. Su, "Visual place recognition: A survey from deep learning perspective," *Pattern Recognition*, vol. 113, p. 107760, May 2021.

[2] N. Sünderhauf, S. Shirazi, F. Dayoub, B. Upcroft, and M. Milford, "On the performance of ConvNet features for place recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4297–4304, Sept. 2015.

[3] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *IEEE Robotics & Automation Magazine*, vol. 13, pp. 99–110, June 2006. Conference Name: IEEE Robotics & Automation Magazine.

[4] S. Schubert, P. Neubert, S. Garg, M. Milford, and T. Fischer, "Visual Place Recognition: A Tutorial," *IEEE Robotics & Automation Magazine*, pp. 2–16, 2023.

[5] H. Jégou, F. Perronnin, M. Douze, J. Sánchez, P. Pérez, and C. Schmid, "Aggregating local image descriptors into compact codes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 9, pp. 1704–1716, 2012.

[6] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, pp. 91–110, 2004.

[7] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[8] Z. Chen, O. Lam, A. Jacobson, and M. Milford, "Convolutional Neural Network-based Place Recognition," Nov. 2014. arXiv:1411.1509 [cs].

[9] R. Arandjelović, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "Netvlad: Cnn architecture for weakly supervised place recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 6, pp. 1437–1451, 2018.

[10] H. J. Kim, E. Dunn, and J.-M. Frahm, "Learned contextual feature reweighting for image geo-localization," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3251–3260, 2017.

[11] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," Feb. 2016. arXiv:1510.00149 [cs].

[12] Y. He and L. Xiao, "Structured Pruning for Deep Convolutional Neural Networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–20, 2023. arXiv:2303.00566 [cs].

[13] Y. LeCun, J. Denker, and S. Solla, "Optimal Brain Damage," in *Advances in Neural Information Processing Systems*, vol. 2, Morgan-Kaufmann, 1989.

[14] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both Weights and Connections for Efficient Neural Network," in *Advances in Neural Information Processing Systems*, vol. 28, Curran Associates, Inc., 2015.

[15] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures," July 2016. arXiv:1607.03250 [cs].

[16] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression," in *2017 IEEE International Conference on Computer Vision (ICCV)*, (Venice), pp. 5068–5076, IEEE, Oct. 2017.

[17] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *SIGARCH Comput. Archit. News*, vol. 44, p. 243–254, jun 2016.

[18] J. Liu, S. Tripathi, U. Kurup, and M. Shah, "Pruning Algorithms to Accelerate Convolutional Neural Networks for Edge Applications: A Survey," May 2020. arXiv:2005.04275 [cs, stat].

[19] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning Efficient Convolutional Networks through Network Slimming," in *2017 IEEE International Conference on Computer Vision (ICCV)*, (Venice), pp. 2755–2763, IEEE, Oct. 2017.

[20] C. Wang, G. Zhang, and R. Grosse, "Picking Winning Tickets Before Training by Preserving Gradient Flow," Aug. 2020. arXiv:2002.07376 [cs, stat].

[21] N. Lee, T. Ajanthan, and P. H. S. Torr, "SNIP: Single-shot Network Pruning based on Connection Sensitivity," Feb. 2019. arXiv:1810.02340 [cs].

[22] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin, "Pruning Neural Networks at Initialization: Why are We Missing the Mark?," Mar. 2021. arXiv:2009.08576 [cs, stat].

[23] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning Filters for Efficient ConvNets," Mar. 2017. arXiv:1608.08710 [cs].

[24] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, p. 2234–2240, AAAI Press, 2018.

[25] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4335–4344, 2019.

[26] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 1398–1406, 2017.

[27] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations*, 2019.

[28] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *International Conference on Learning Representations*, 2019.

[29] F. Radenović, G. Tolias, and O. Chum, "Fine-Tuning CNN Image Retrieval with No Human Annotation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, pp. 1655–1668, July 2019. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[30] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "NetVLAD: CNN Architecture for Weakly Supervised Place Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Las Vegas, NV, USA), pp. 5297–5307, IEEE, June 2016.

[31] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3304–3311, 2010.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[33] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 6105–6114, PMLR, 09–15 Jun 2019.

[34] A. Howard, M. Sandler, B. Chen, W. Wang, L. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, "Searching for mobilenetv3," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, (Los Alamitos, CA, USA), pp. 1314–1324, IEEE Computer Society, nov 2019.

[35] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[36] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2019.

[37] J. Lee, S. Park, S. Mo, S. Ahn, and J. Shin, "Layer-adaptive sparsity for the magnitude-based pruning," *arXiv preprint arXiv:2010.07611*, 2020.

[38] G. Fang, X. Ma, M. Song, M. Bi Mi, and X. Wang, "DepGraph: Towards Any Structural Pruning," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (Vancouver, BC, Canada), pp. 16091–16101, IEEE, June 2023.

[39] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. 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*, pp. 4340–4349, 2019.

[40] A. Torii, J. Sivic, T. Pajdla, and M. Okutomi, "Visual place recognition with repetitive structures," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 883–890, 2013.

[41] G. Berton, R. Mereu, G. Trivigno, C. Masone, G. Csurka, T. Sattler, and B. Caputo, "Deep visual geo-localization benchmark," in *CVPR*, June 2022.

[42] Y. He, P. Liu, L. Zhu, and Y. Yang, "Filter Pruning by Switching to Neighboring CNNs with Good Attributes," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, pp. 8044–8056, Oct. 2023. arXiv:1904.03961 [cs].

A Tabulated Data

This appendix provides tabulated data from each pruning run. The title of each table indicates the CNN architecture, aggregation layer and pruning method used, respectively.

Resnet18-NetVLAD-L1-Norm

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	79.58	75.57	74.46	68.12	59.64	44.56	26.47	14.25	8.45	5.77
Recall@1 percentage	100.0	94.97	93.57	85.6	74.94	55.99	33.26	17.9	10.62	7.25
Area under recall@k curve	95.55	94.37	94.55	92.99	91.57	87.31	76.2	64.26	58.62	57.63
Area under recall@k curve percentage	100.0	98.77	98.95	97.32	95.83	91.38	79.75	67.25	61.35	60.31
Inference time (ms)	78.4	73.9	58.31	48.14	39.82	32.53	29.31	25.9	22.93	20.48
Inference time percentage	100.0	94.26	74.38	61.4	50.79	41.48	37.38	33.03	29.24	26.12
Params (millions)	11.24	10.18	9.21	8.34	7.52	6.68	5.67	4.65	3.3	1.73
Params percentage	100.0	90.59	81.9	74.17	66.87	59.41	50.48	41.36	29.37	15.41
DMAs (millions)	1837.4	1285.73	957.09	736.94	559.24	434.09	344.76	265.59	181.07	92.33
DMAs percentage	100.0	69.98	52.09	40.11	30.44	23.62	18.76	14.45	9.85	5.02
MACs (millions)	1820.54	1271.53	944.78	726.09	549.8	425.98	337.92	260.09	177.22	90.3
MACs percentage	100.0	69.84	51.9	39.88	30.2	23.4	18.56	14.29	9.73	4.96
Peak memory usage (Gigabytes)	3.64	2.56	1.99	1.68	1.37	1.11	1.0	0.84	0.63	0.61
Peak memory usage percentage	100.0	70.31	54.65	46.15	37.51	30.42	27.43	23.14	17.32	16.76

Resnet18-NetVLAD-L2-Norm

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	79.58	74.47	67.71	53.05	46.17	43.78	41.21	21.55	7.73	6.37
Recall@1 percentage	100.0	93.58	85.08	66.67	58.02	55.01	51.79	27.08	9.72	8.0
Area under recall@k curve	95.55	94.5	93.31	89.98	86.66	85.83	85.08	74.32	45.3	51.49
Area under recall@k curve percentage	100.0	98.9	97.66	94.17	90.7	89.83	89.04	77.78	47.41	53.89
Inference time (ms)	78.39	73.33	57.53	45.91	37.01	30.58	27.67	25.04	22.13	20.5
Inference time percentage	100.0	93.54	73.39	58.57	47.21	39.01	35.3	31.95	28.24	26.15
Params (millions)	11.24	10.24	9.3	8.45	7.68	6.89	5.89	4.74	3.41	1.77
Params percentage	100.0	91.05	82.69	75.15	68.35	61.32	52.42	42.13	30.32	15.75
DMAs (millions)	1837.4	1271.98	946.14	706.1	530.46	421.82	338.1	261.98	181.22	95.66
DMAs percentage	100.0	69.23	51.49	38.43	28.87	22.96	18.4	14.26	9.86	5.21
MACs (millions)	1820.54	1257.8	933.79	695.34	521.11	413.63	331.2	256.49	177.37	93.56
MACs percentage	100.0	69.09	51.29	38.19	28.62	22.72	18.19	14.09	9.74	5.14
Peak memory usage (Gigabytes)	3.64	2.51	1.99	1.58	1.27	1.11	0.95	0.79	0.62	0.61
Peak memory usage percentage	100.0	68.9	54.64	43.25	34.76	30.42	26.04	21.68	16.94	16.77

Resnet18-NetVLAD-LAMP

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	79.58	78.73	75.38	74.82	72.43	66.74	61.66	56.94	38.72	8.51
Recall@1 percentage	100.0	98.93	94.73	94.03	91.02	83.87	77.49	71.55	48.65	10.69
Area under recall@k curve	95.55	95.27	94.99	94.33	94.01	92.6	92.17	89.99	84.66	62.56
Area under recall@k curve percentage	100.0	99.71	99.41	98.72	98.39	96.91	96.46	94.18	88.6	65.47
Inference time (ms)	78.63	77.96	77.88	75.85	74.34	75.24	72.42	63.08	50.62	31.02
Inference time percentage	100.0	99.15	99.04	96.46	94.55	95.69	92.1	80.22	64.37	39.45
Params (millions)	11.24	9.34	7.52	5.92	4.52	3.34	2.4	1.64	0.99	0.47
Params percentage	100.0	83.07	66.89	52.66	40.22	29.73	21.37	14.55	8.84	4.21
DMAs (millions)	1837.4	1742.2	1625.97	1468.36	1322.82	1125.13	890.86	601.14	350.08	126.1
DMAs percentage	100.0	94.82	88.49	79.91	71.99	61.23	48.48	32.72	19.05	6.86
MACs (millions)	1820.54	1727.26	1612.89	1456.94	1312.88	1116.61	883.69	595.55	346.08	123.98
MACs percentage	100.0	94.88	88.59	80.03	72.11	61.33	48.54	32.71	19.01	6.81
Peak memory usage (Gigabytes)	3.64	3.63	3.63	3.62	3.61	3.51	3.3	2.83	2.27	1.39
Peak memory usage percentage	100.0	99.79	99.59	99.41	99.26	96.38	90.62	77.78	62.25	38.15

Resnet18-NetVLAD-FPGM

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	79.58	76.16	68.18	62.47	49.46	45.5	29.97	22.58	19.88	10.81
Recall@1 percentage	100.0	95.7	85.67	78.5	62.15	57.17	37.67	28.37	24.98	13.59
Area under recall@k curve	95.55	94.53	93.41	92.51	87.24	86.99	78.93	74.76	70.38	62.32
Area under recall@k curve percentage	100.0	98.93	97.76	96.82	91.3	91.04	82.61	78.24	73.66	65.22
Inference time (ms)	78.54	73.33	66.49	59.15	45.04	38.58	32.91	28.49	25.23	21.87
Inference time percentage	100.0	93.37	84.65	75.31	57.35	49.12	41.9	36.27	32.13	27.85
Params (millions)	11.24	10.33	9.32	8.33	7.41	6.44	5.45	4.33	2.99	1.46
Params percentage	100.0	91.86	82.86	74.09	65.88	57.32	48.5	38.55	26.61	12.95
DMAs (millions)	1837.4	1305.31	1002.23	781.34	602.18	468.27	361.32	273.04	185.07	87.48
DMAs percentage	100.0	71.04	54.55	42.52	32.77	25.49	19.66	14.86	10.07	4.76
MACs (millions)	1820.54	1290.82	989.37	769.9	592.23	459.83	354.39	267.55	181.19	85.52
MACs percentage	100.0	70.9	54.35	42.29	32.53	25.26	19.47	14.7	9.95	4.7
Peak memory usage (Gigabytes)	3.64	2.72	2.45	2.3	1.98	1.62	1.26	1.05	0.89	0.62
Peak memory usage percentage	100.0	74.54	67.33	63.06	54.42	44.44	34.51	28.75	24.33	17.15

Mobile-NetVLAD-L1-Norm

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	81.16	73.02	58.66	50.82	42.55	27.48	22.83	10.14	11.78	4.93
Recall@1 percentage	100.0	89.97	72.27	62.62	52.42	33.86	28.13	12.49	14.52	6.07
Area under recall@k curve	95.65	94.27	90.63	88.05	85.56	77.36	74.05	60.61	64.61	55.06
Area under recall@k curve percentage	100.0	98.56	94.75	92.05	89.45	80.88	77.42	63.37	67.55	57.56
Inference time (ms)	72.19	56.71	44.44	36.81	30.17	23.16	16.8	13.1	11.07	10.74
Inference time percentage	100.0	78.55	61.56	50.99	41.79	32.09	23.28	18.15	15.33	14.88
Params (millions)	2.04	1.84	1.62	1.36	1.1	0.87	0.67	0.46	0.28	0.13
Params percentage	100.0	90.28	79.28	66.77	53.72	42.5	32.61	22.77	13.74	6.35
DMAs (millions)	208.04	163.34	128.8	98.45	71.77	49.34	32.6	21.48	12.9	6.6
DMAs percentage	100.0	78.51	61.91	47.32	34.5	23.72	15.67	10.33	6.2	3.17
MACs (millions)	199.42	156.7	123.7	94.29	68.31	46.62	30.62	20.08	12.07	6.17
MACs percentage	100.0	78.58	62.03	47.28	34.25	23.38	15.35	10.07	6.05	3.09
Peak memory usage (Gigabytes)	4.02	2.53	1.6	1.32	1.21	1.06	0.85	0.72	0.56	0.5
Peak memory usage percentage	100.0	62.93	39.83	32.78	30.23	26.32	21.18	17.97	13.86	12.47

Mobile-NetVLAD-L2-Norm

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	81.16	73.25	59.07	52.33	38.81	31.65	16.56	7.22	3.27	2.38
Recall@1 percentage	100.0	90.26	72.78	64.48	47.81	38.99	20.41	8.89	4.03	2.93
Area under recall@k curve	95.65	94.53	91.36	89.63	84.76	80.39	65.69	54.8	54.54	52.63
Area under recall@k curve percentage	100.0	98.83	95.51	93.71	88.61	84.05	68.68	57.29	57.02	55.02
Inference time (ms)	72.23	59.54	46.97	37.97	31.94	26.96	21.12	15.7	12.64	10.93
Inference time percentage	100.0	82.44	65.03	52.57	44.21	37.32	29.24	21.74	17.5	15.14
Params (millions)	2.04	1.82	1.6	1.38	1.15	0.92	0.69	0.47	0.29	0.13
Params percentage	100.0	89.24	78.34	67.5	56.43	45.07	33.74	23.06	14.3	6.38
DMAs (millions)	208.04	168.32	135.96	109.87	87.27	67.44	48.36	31.76	19.72	7.77
DMAs percentage	100.0	80.9	65.35	52.81	41.95	32.42	23.25	15.27	9.48	3.73
MACs (millions)	199.42	161.35	130.4	105.37	83.52	64.32	45.93	30.03	18.54	7.2
MACs percentage	100.0	80.91	65.39	52.84	41.88	32.26	23.03	15.06	9.3	3.61
Peak memory usage (Gigabytes)	4.02	2.91	1.93	1.45	1.37	1.26	1.08	0.82	0.72	0.52
Peak memory usage percentage	100.0	72.47	48.17	36.02	34.02	31.47	26.97	20.52	17.94	13.06

Mobile-NetVLAD-LAMP

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	81.16	80.66	77.35	75.31	68.87	61.11	51.79	50.1	41.07	5.72
Recall@1 percentage	100.0	99.39	95.3	92.79	84.85	75.29	63.81	61.73	50.6	7.05
Area under recall@k curve	95.65	95.59	95.04	95.09	94.03	92.67	90.99	89.24	87.43	54.83
Area under recall@k curve percentage	100.0	99.94	99.36	99.41	98.31	96.88	95.13	93.3	91.41	57.32
Inference time (ms)	72.23	70.34	67.65	64.88	61.26	57.35	54.06	47.9	38.74	25.24
Inference time percentage	100.0	97.39	93.66	89.82	84.81	79.4	74.84	66.32	53.63	34.94
Params (millions)	2.04	1.67	1.33	1.01	0.73	0.52	0.36	0.22	0.12	0.05
Params percentage	100.0	81.68	65.16	49.52	35.72	25.52	17.41	11.0	5.87	2.49
DMAs (millions)	208.04	194.15	177.28	159.84	141.94	126.19	109.64	89.04	63.05	31.32
DMAs percentage	100.0	93.32	85.21	76.83	68.23	60.66	52.7	42.8	30.31	15.05
MACs (millions)	199.42	186.0	169.6	152.62	135.19	119.85	103.8	83.96	59.13	28.99
MACs percentage	100.0	93.27	85.05	76.53	67.79	60.1	52.05	42.1	29.65	14.54
Peak memory usage (Gigabytes)	4.02	4.01	4.01	4.01	4.01	4.01	3.96	3.6	2.83	1.75
Peak memory usage percentage	100.0	99.96	99.93	99.9	99.87	99.85	98.58	89.58	70.38	43.61

Mobile-NetVLAD-FPGM

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	81.16	72.11	68.1	57.54	51.82	51.88	29.75	24.05	16.62	23.33
Recall@1 percentage	100.0	88.85	83.91	70.9	63.85	63.92	36.66	29.63	20.48	28.74
Area under recall@k curve	95.65	94.05	93.04	90.76	89.39	89.46	78.09	77.27	67.98	79.16
Area under recall@k curve percentage	100.0	98.33	97.27	94.89	93.46	93.53	81.64	80.78	71.07	82.76
Inference time (ms)	72.23	59.46	50.54	42.16	33.97	28.21	24.71	18.49	14.97	10.89
Inference time percentage	100.0	82.33	69.97	58.38	47.04	39.06	34.21	25.59	20.72	15.08
Params (millions)	2.04	1.85	1.59	1.29	1.01	0.78	0.59	0.41	0.25	0.11
Params percentage	100.0	90.57	77.74	63.05	49.38	38.24	28.79	20.04	12.34	5.62
DMAs (millions)	208.04	172.62	145.24	119.23	94.17	73.3	55.09	38.04	23.54	11.36
DMAs percentage	100.0	82.98	69.81	57.31	45.27	35.23	26.48	18.28	11.31	5.46
MACs (millions)	199.42	165.77	139.48	114.52	90.37	70.3	52.68	36.28	22.3	10.59
MACs percentage	100.0	83.13	69.94	57.42	45.32	35.25	26.42	18.19	11.18	5.31
Peak memory usage (Gigabytes)	4.02	2.71	2.24	1.86	1.55	1.44	1.34	1.23	1.13	0.82
Peak memory usage percentage	100.0	67.4	55.88	46.24	38.54	35.94	33.4	30.75	28.18	20.49

Efficient-NetVLAD-L1-Norm

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	73.08	83.22	75.18	52.83	21.89	10.49	10.61	9.38	3.52	1.91
Recall@1 percentage	100.0	113.87	102.87	72.29	29.95	14.35	14.52	12.83	4.82	2.61
Area under recall@k curve	93.48	96.14	94.49	87.62	69.56	48.71	48.46	25.65	40.41	38.76
Area under recall@k curve percentage	100.0	102.85	101.08	93.73	74.41	52.11	51.84	27.44	43.23	41.46
Inference time (ms)	239.35	207.58	174.02	138.09	104.1	74.3	51.18	32.35	23.34	22.29
Inference time percentage	100.0	86.73	72.71	57.69	43.49	31.04	21.38	13.51	9.75	9.31
Params (millions)	6.85	6.18	5.57	4.94	4.24	3.47	2.72	2.06	1.38	0.68
Params percentage	100.0	90.18	81.36	72.09	61.9	50.64	39.67	30.05	20.13	9.9
DMAs (millions)	851.25	738.91	614.01	489.36	367.57	258.24	171.27	115.3	72.33	35.12
DMAs percentage	100.0	86.8	72.13	57.49	43.18	30.34	20.12	13.55	8.5	4.13
MACs (millions)	831.78	721.94	599.53	477.44	358.13	250.98	165.87	111.51	69.92	33.92
MACs percentage	100.0	86.79	72.08	57.4	43.06	30.17	19.94	13.41	8.41	4.08
Peak memory usage (Gigabytes)	8.35	7.27	5.67	4.03	2.62	1.92	1.39	0.95	0.65	0.53
Peak memory usage percentage	100.0	87.05	67.95	48.25	31.35	23.01	16.59	11.33	7.76	6.35

Efficient-NetVLAD-L2-Norm

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	73.08	74.87	70.94	31.9	8.66	7.01	6.46	6.18	3.07	3.01
Recall@1 percentage	100.0	102.45	97.07	43.65	11.85	9.6	8.83	8.45	4.2	4.12
Area under recall@k curve	93.48	94.37	94.35	79.81	57.94	55.03	50.53	50.25	46.59	42.47
Area under recall@k curve percentage	100.0	100.95	100.93	85.38	61.98	58.87	54.05	53.75	49.84	45.43
Inference time (ms)	239.39	197.74	160.12	128.14	99.42	80.01	61.51	47.99	35.2	23.08
Inference time percentage	100.0	82.6	66.88	53.53	41.53	33.42	25.7	20.05	14.7	9.64
Params (millions)	6.85	6.32	5.77	5.18	4.49	3.73	2.94	2.13	1.35	0.64
Params percentage	100.0	92.25	84.31	75.56	65.57	54.41	42.9	31.14	19.71	9.31
DMAs (millions)	851.25	706.5	584.01	473.98	374.61	292.98	215.51	145.05	84.6	38.51
DMAs percentage	100.0	83.0	68.61	55.68	44.01	34.42	25.32	17.04	9.94	4.52
MACs (millions)	831.78	689.93	569.84	462.08	364.87	284.98	209.09	140.19	81.28	36.77
MACs percentage	100.0	82.95	68.51	55.55	43.87	34.26	25.14	16.85	9.77	4.42
Peak memory usage (Gigabytes)	8.35	6.89	6.06	5.06	3.9	3.28	3.02	2.76	2.42	1.57
Peak memory usage percentage	100.0	82.45	72.57	60.54	46.68	39.26	36.16	33.02	29.0	18.81

Efficient-NetVLAD-LAMP

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	73.08	81.21	81.16	79.91	77.02	73.34	64.69	34.18	10.77	4.81
Recall@1 percentage	100.0	111.12	111.06	109.36	105.4	100.36	88.52	46.78	14.74	6.59
Area under recall@k curve	93.48	94.99	94.74	94.75	93.75	93.22	92.18	76.24	51.58	45.9
Area under recall@k curve percentage	100.0	101.62	101.35	101.36	100.29	99.72	98.61	81.56	55.18	49.1
Inference time (ms)	239.6	235.08	227.93	217.24	205.52	194.15	176.04	148.73	113.5	66.95
Inference time percentage	100.0	98.11	95.13	90.67	85.78	81.03	73.47	62.08	47.37	27.94
Params (millions)	6.85	5.9	5.06	4.27	3.52	2.79	2.09	1.44	0.83	0.28
Params percentage	100.0	86.11	73.91	62.37	51.35	40.69	30.46	21.03	12.05	4.02
DMAs (millions)	851.25	803.07	743.8	680.37	613.75	541.48	461.2	364.89	244.09	109.12
DMAs percentage	100.0	94.34	87.38	79.93	72.1	63.61	54.18	42.87	28.67	12.82
MACs (millions)	831.78	784.8	726.72	664.49	599.11	528.18	449.44	355.07	236.85	105.03
MACs percentage	100.0	94.35	87.37	79.89	72.03	63.5	54.03	42.69	28.48	12.63
Peak memory usage (Gigabytes)	8.35	8.35	8.34	8.29	8.18	8.03	7.72	7.35	6.02	3.5
Peak memory usage percentage	100.0	99.95	99.91	99.27	97.99	96.14	92.39	88.05	72.04	41.85

Efficient-NetVLAD-FPGM

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	73.08	82.28	79.55	73.78	63.1	49.31	37.49	12.51	12.88	13.26
Recall@1 percentage	100.0	112.59	108.85	100.96	86.35	67.48	51.29	17.13	17.63	18.15
Area under recall@k curve	93.48	95.47	94.28	93.86	92.79	89.25	83.27	51.23	51.05	55.38
Area under recall@k curve percentage	100.0	102.13	100.86	100.41	99.26	95.47	89.08	54.8	54.61	59.24
Inference time (ms)	239.67	211.21	184.83	160.0	133.08	103.4	78.52	58.98	37.96	22.9
Inference time percentage	100.0	88.13	77.12	66.76	55.53	43.14	32.76	24.61	15.84	9.56
Params (millions)	6.85	6.32	5.7	5.03	4.31	3.51	2.7	1.98	1.26	0.58
Params percentage	100.0	92.21	83.24	73.51	62.99	51.19	39.44	28.9	18.43	8.5
DMAs (millions)	851.25	743.88	639.92	537.08	431.42	310.66	214.16	143.3	85.28	39.85
DMAs percentage	100.0	87.39	75.17	63.09	50.68	36.5	25.16	16.83	10.02	4.68
MACs (millions)	831.78	726.55	624.53	523.7	420.14	301.65	207.24	138.17	81.92	38.07
MACs percentage	100.0	87.35	75.08	62.96	50.51	36.27	24.91	16.61	9.85	4.58
Peak memory usage (Gigabytes)	8.35	7.99	7.11	6.03	4.87	3.66	2.89	2.4	1.83	1.24
Peak memory usage percentage	100.0	95.67	85.18	72.23	58.36	43.87	34.61	28.73	21.9	14.82

Resnet18-GeM-L1-Norm

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	72.34	70.67	67.72	62.97	54.2	41.3	30.69	14.41	10.15	10.31
Recall@1 percentage	100.0	97.69	93.61	87.04	74.91	57.09	42.43	19.91	14.03	14.26
Area under recall@k curve	95.17	93.83	93.45	91.74	90.23	85.37	78.91	63.09	58.94	60.97
Area under recall@k curve percentage	100.0	98.59	98.19	96.4	94.81	89.7	82.91	66.29	61.93	64.06
Inference time (ms)	64.32	59.89	44.22	35.05	25.69	19.11	15.62	12.9	9.52	6.76
Inference time percentage	100.0	93.1	68.75	54.5	39.94	29.71	24.29	20.06	14.8	10.51
Params (millions)	11.18	10.12	9.15	8.29	7.48	6.63	5.64	4.61	3.25	1.68
Params percentage	100.0	90.52	81.83	74.14	66.93	59.33	50.43	41.2	29.08	14.99
DMAs (millions)	1835.76	1284.56	958.97	735.29	557.57	432.92	343.9	264.84	180.0	91.07
DMAs percentage	100.0	69.97	52.24	40.05	30.37	23.58	18.73	14.43	9.8	4.96
MACs (millions)	1818.93	1270.39	946.7	724.46	548.14	424.83	337.08	259.36	176.17	89.07
MACs percentage	100.0	69.84	52.05	39.83	30.14	23.36	18.53	14.26	9.69	4.9
Peak memory usage (Gigabytes)	3.64	2.56	1.99	1.68	1.37	1.11	1.0	0.84	0.63	0.52
Peak memory usage percentage	100.0	70.31	54.65	46.14	37.51	30.42	27.43	23.11	17.29	14.3

Resnet18-GeM-L2-Norm

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	72.34	70.8	65.02	47.46	40.68	38.66	23.53	17.81	7.35	8.86
Recall@1 percentage	100.0	97.87	89.88	65.61	56.24	53.44	32.53	24.62	10.16	12.25
Area under recall@k curve	95.17	94.25	93.42	88.01	86.82	85.55	75.15	71.02	46.95	60.36
Area under recall@k curve percentage	100.0	99.03	98.16	92.48	91.23	89.89	78.96	74.62	49.33	63.42
Inference time (ms)	64.2	60.12	43.56	32.67	23.4	18.08	14.03	12.34	8.5	6.98
Inference time percentage	100.0	93.63	67.85	50.89	36.44	28.17	21.85	19.22	13.24	10.88
Params (millions)	11.18	10.16	9.23	8.38	7.59	6.79	5.83	4.65	3.35	1.71
Params percentage	100.0	90.94	82.56	74.93	67.9	60.79	52.18	41.61	29.95	15.29
DMAs (millions)	1835.76	1268.69	944.08	710.03	531.01	420.37	336.47	259.71	179.5	94.22
DMAs percentage	100.0	69.11	51.43	38.68	28.93	22.9	18.33	14.15	9.78	5.13
MACs (millions)	1818.93	1254.56	931.77	699.26	521.7	412.23	329.61	254.28	175.68	92.15
MACs percentage	100.0	68.97	51.23	38.44	28.68	22.66	18.12	13.98	9.66	5.07
Peak memory usage (Gigabytes)	3.64	2.51	1.99	1.63	1.27	1.11	0.95	0.79	0.58	0.52
Peak memory usage percentage	100.0	68.9	54.65	44.66	34.75	30.41	26.04	21.68	15.89	14.31

Resnet18-GeM-LAMP

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	72.34	74.21	73.4	71.36	66.3	64.51	58.41	51.1	37.41	9.57
Recall@1 percentage	100.0	102.58	101.46	98.64	91.64	89.17	80.73	70.63	51.71	13.22
Area under recall@k curve	95.17	95.15	95.24	94.72	92.65	92.5	90.88	89.47	84.31	63.99
Area under recall@k curve percentage	100.0	99.98	100.07	99.53	97.35	97.19	95.49	94.01	88.59	67.24
Inference time (ms)	64.86	63.78	64.03	61.43	60.1	61.51	57.79	49.49	37.62	16.93
Inference time percentage	100.0	98.34	98.72	94.72	92.66	94.83	89.1	76.3	57.99	26.11
Params (millions)	11.18	9.27	7.46	5.85	4.46	3.27	2.33	1.57	0.92	0.41
Params percentage	100.0	82.97	66.73	52.35	39.87	29.27	20.87	14.06	8.23	3.64
DMAs (millions)	1835.76	1740.56	1623.91	1467.12	1321.18	1132.66	889.06	607.76	352.14	124.83
DMAs percentage	100.0	94.81	88.46	79.92	71.97	61.7	48.43	33.11	19.18	6.8
MACs (millions)	1818.93	1725.65	1610.86	1455.74	1311.27	1124.12	881.93	602.15	348.17	122.74
MACs percentage	100.0	94.87	88.56	80.03	72.09	61.8	48.49	33.1	19.14	6.75
Peak memory usage (Gigabytes)	3.64	3.63	3.63	3.62	3.61	3.56	3.3	2.89	2.27	1.39
Peak memory usage percentage	100.0	99.77	99.6	99.39	99.24	97.74	90.6	79.23	62.23	38.14

Resnet18-GeM-FPGM

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	72.34	72.77	64.08	61.05	46.41	32.19	26.6	21.41	15.85	12.16
Recall@1 percentage	100.0	100.59	88.58	84.38	64.15	44.49	36.77	29.59	21.9	16.81
Area under recall@k curve	95.17	95.08	93.73	92.69	88.98	82.82	78.83	75.97	66.2	61.71
Area under recall@k curve percentage	100.0	99.91	98.49	97.39	93.5	87.02	82.83	79.83	69.56	64.84
Inference time (ms)	64.29	59.54	52.95	45.77	30.9	24.91	19.51	14.88	11.86	8.41
Inference time percentage	100.0	92.61	82.36	71.2	48.07	38.75	30.35	23.14	18.45	13.09
Params (millions)	11.18	10.26	9.25	8.26	7.34	6.38	5.39	4.27	2.93	1.39
Params percentage	100.0	91.81	82.76	73.94	65.68	57.07	48.2	38.19	26.18	12.44
DMAs (millions)	1835.76	1303.66	1000.59	779.7	600.54	466.62	359.68	271.4	183.43	85.83
DMAs percentage	100.0	71.01	54.51	42.47	32.71	25.42	19.59	14.78	9.99	4.68
MACs (millions)	1818.93	1289.21	987.77	768.29	590.62	458.22	352.78	265.95	179.58	83.91
MACs percentage	100.0	70.88	54.3	42.24	32.47	25.19	19.39	14.62	9.87	4.61
Peak memory usage (Gigabytes)	3.64	2.71	2.45	2.3	1.98	1.62	1.26	1.05	0.89	0.62
Peak memory usage percentage	100.0	74.54	67.32	63.05	54.41	44.43	34.51	28.74	24.31	17.08

Mobile-GeM-L1-Norm

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	62.85	43.59	29.31	30.85	24.74	18.72	16.83	10.21	8.35	5.65
Recall@1 percentage	100.0	69.35	46.64	49.09	39.36	29.79	26.77	16.25	13.28	8.99
Area under recall@k curve	93.66	88.32	80.93	80.82	75.66	69.13	68.91	58.65	56.47	53.05
Area under recall@k curve percentage	100.0	94.3	86.41	86.29	80.78	73.81	73.57	62.62	60.29	56.64
Inference time (ms)	69.7	54.13	42.19	34.15	28.08	21.04	15.0	10.6	7.45	6.62
Inference time percentage	100.0	77.66	60.54	49.0	40.29	30.19	21.52	15.21	10.69	9.5
Params (millions)	2.02	1.82	1.6	1.34	1.07	0.84	0.64	0.44	0.26	0.11
Params percentage	100.0	90.21	79.11	66.36	53.08	41.62	31.56	21.73	12.76	5.32
DMAs (millions)	207.53	162.81	128.97	98.4	72.17	49.49	33.22	21.35	12.62	6.21
DMAs percentage	100.0	78.45	62.15	47.41	34.78	23.85	16.01	10.29	6.08	2.99
MACs (millions)	198.92	156.18	123.86	94.23	68.67	46.75	31.17	19.95	11.82	5.79
MACs percentage	100.0	78.51	62.26	47.37	34.52	23.5	15.67	10.03	5.94	2.91
Peak memory usage (Gigabytes)	4.02	2.53	1.6	1.32	1.21	1.06	0.93	0.72	0.56	0.5
Peak memory usage percentage	100.0	62.93	39.82	32.78	30.23	26.32	23.12	17.97	13.9	12.56

Mobile-GeM-L2-Norm

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	62.85	57.51	44.64	39.89	31.63	18.54	12.43	8.85	4.42	3.32
Recall@1 percentage	100.0	91.5	71.03	63.47	50.33	29.51	19.77	14.08	7.03	5.28
Area under recall@k curve	93.66	92.5	89.27	86.3	82.07	69.77	60.1	58.98	54.84	52.99
Area under recall@k curve percentage	100.0	98.76	95.31	92.14	87.63	74.49	64.17	62.97	58.55	56.58
Inference time (ms)	69.72	57.08	44.42	35.6	30.52	24.63	19.51	14.29	10.88	6.66
Inference time percentage	100.0	81.88	63.71	51.06	43.78	35.33	27.98	20.5	15.61	9.55
Params (millions)	2.02	1.8	1.57	1.35	1.12	0.89	0.66	0.44	0.26	0.1
Params percentage	100.0	89.14	78.0	66.83	55.29	43.92	32.72	21.77	12.93	5.17
DMAs (millions)	207.53	167.94	135.3	110.33	88.76	67.3	47.89	31.13	18.58	7.23
DMAs percentage	100.0	80.92	65.2	53.17	42.77	32.43	23.08	15.0	8.95	3.48
MACs (millions)	198.92	160.97	129.75	105.8	84.96	64.16	45.44	29.32	17.34	6.6
MACs percentage	100.0	80.92	65.23	53.19	42.71	32.26	22.84	14.74	8.72	3.32
Peak memory usage (Gigabytes)	4.02	2.91	1.93	1.45	1.37	1.31	1.13	0.9	0.72	0.53
Peak memory usage percentage	100.0	72.47	48.17	36.01	34.02	32.72	28.24	22.44	17.94	13.29

Mobile-GeM-LAMP

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	62.85	68.59	65.46	63.25	59.08	56.53	49.3	47.17	34.43	6.9
Recall@1 percentage	100.0	109.13	104.15	100.63	94.0	89.94	78.43	75.05	54.79	10.97
Area under recall@k curve	93.66	93.99	93.57	93.1	92.76	92.19	90.67	90.33	84.48	52.13
Area under recall@k curve percentage	100.0	100.35	99.9	99.4	99.04	98.43	96.81	96.44	90.2	55.66
Inference time (ms)	69.77	67.84	65.17	62.47	58.71	54.87	51.62	45.5	36.38	22.87
Inference time percentage	100.0	97.23	93.41	89.54	84.16	78.64	73.99	65.22	52.15	32.78
Params (millions)	2.02	1.65	1.31	0.99	0.71	0.5	0.34	0.2	0.1	0.03
Params percentage	100.0	81.48	64.8	48.98	35.17	24.79	16.61	10.05	4.93	1.48
DMAs (millions)	207.53	193.64	176.8	159.37	141.29	125.63	109.16	88.06	62.62	30.72
DMAs percentage	100.0	93.31	85.19	76.79	68.08	60.54	52.6	42.43	30.18	14.8
MACs (millions)	198.92	185.51	169.13	152.16	134.56	119.31	103.32	83.01	58.72	28.38
MACs percentage	100.0	93.26	85.03	76.49	67.64	59.98	51.94	41.73	29.52	14.27
Peak memory usage (Gigabytes)	4.02	4.01	4.01	4.01	4.01	4.01	3.96	3.6	2.78	1.75
Peak memory usage percentage	100.0	99.96	99.93	99.9	99.87	99.85	98.58	89.58	69.1	43.61

Mobile-GeM-FPGM

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	62.85	51.26	39.26	35.21	35.59	30.3	17.15	11.44	12.51	14.8
Recall@1 percentage	100.0	81.56	62.46	56.02	56.63	48.2	27.29	18.21	19.91	23.55
Area under recall@k curve	93.66	90.77	86.47	84.45	84.76	81.83	66.26	57.89	57.95	70.07
Area under recall@k curve percentage	100.0	96.91	92.32	90.17	90.5	87.37	70.75	61.81	61.87	74.81
Inference time (ms)	69.73	56.95	47.99	39.62	31.43	25.64	22.07	15.81	12.34	8.28
Inference time percentage	100.0	81.68	68.83	56.82	45.08	36.78	31.66	22.68	17.7	11.88
Params (millions)	2.02	1.83	1.57	1.27	0.99	0.76	0.57	0.39	0.23	0.09
Params percentage	100.0	90.48	77.52	62.68	48.86	37.62	28.07	19.23	11.45	4.68
DMAs (millions)	207.53	172.11	144.73	118.72	93.66	72.78	54.57	37.52	23.02	10.86
DMAs percentage	100.0	82.93	69.74	57.21	45.13	35.07	26.3	18.08	11.09	5.23
MACs (millions)	198.92	165.27	138.98	114.01	89.87	69.79	52.18	35.78	21.8	10.11
MACs percentage	100.0	83.09	69.87	57.32	45.18	35.09	26.23	17.99	10.96	5.08
Peak memory usage (Gigabytes)	4.02	2.71	2.24	1.86	1.55	1.44	1.34	1.23	1.13	0.82
Peak memory usage percentage	100.0	67.39	55.88	46.24	38.54	35.94	33.4	30.75	28.17	20.48

Efficient-GeM-L1-Norm

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	66.68	59.89	38.91	18.57	11.83	8.36	8.64	8.2	5.21	2.66
Recall@1 percentage	100.0	89.81	58.35	27.85	17.73	12.54	12.96	12.3	7.81	3.98
Area under recall@k curve	92.13	90.96	83.67	63.21	51.01	43.52	33.29	30.08	33.37	36.54
Area under recall@k curve percentage	100.0	98.73	90.82	68.61	55.37	47.24	36.13	32.65	36.22	39.66
Inference time (ms)	235.61	204.93	169.57	135.03	99.6	70.01	47.5	27.9	20.47	17.25
Inference time percentage	100.0	86.98	71.97	57.31	42.28	29.71	20.16	11.84	8.69	7.32
Params (millions)	6.82	6.15	5.54	4.91	4.21	3.44	2.68	2.02	1.35	0.65
Params percentage	100.0	90.14	81.3	71.94	61.79	50.42	39.34	29.68	19.79	9.53
DMAs (millions)	850.5	738.17	612.65	488.87	369.04	257.93	170.73	114.17	71.55	34.48
DMAs percentage	100.0	86.79	72.03	57.48	43.39	30.33	20.07	13.42	8.41	4.05
MACs (millions)	831.06	721.21	598.22	476.97	359.6	250.67	165.35	110.4	69.16	33.29
MACs percentage	100.0	86.78	71.98	57.39	43.27	30.16	19.9	13.28	8.32	4.01
Peak memory usage (Gigabytes)	8.35	7.27	5.62	4.03	2.62	1.94	1.4	0.94	0.65	0.53
Peak memory usage percentage	100.0	87.05	67.34	48.25	31.43	23.26	16.82	11.25	7.76	6.34

Efficient-GeM-L2-Norm

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	66.68	53.39	40.54	19.91	7.17	5.87	5.41	5.09	3.49	3.14
Recall@1 percentage	100.0	80.07	60.79	29.86	10.76	8.8	8.12	7.63	5.24	4.71
Area under recall@k curve	92.13	90.16	83.84	69.15	55.9	53.11	49.83	49.29	48.19	44.29
Area under recall@k curve percentage	100.0	97.86	91.0	75.06	60.68	57.65	54.09	53.5	52.31	48.07
Inference time (ms)	235.53	193.15	157.54	124.83	97.31	76.85	59.36	44.77	31.99	18.03
Inference time percentage	100.0	82.01	66.89	53.0	41.32	32.63	25.2	19.01	13.58	7.66
Params (millions)	6.82	6.29	5.74	5.14	4.45	3.69	2.9	2.1	1.31	0.61
Params percentage	100.0	92.22	84.18	75.34	65.33	54.1	42.55	30.72	19.27	8.89
DMAs (millions)	850.5	705.76	583.75	473.97	375.87	293.97	215.56	144.76	84.29	37.85
DMAs percentage	100.0	82.98	68.64	55.73	44.19	34.56	25.35	17.02	9.91	4.45
MACs (millions)	831.06	689.21	569.58	462.09	366.11	285.94	209.14	139.9	80.98	36.13
MACs percentage	100.0	82.93	68.54	55.6	44.05	34.41	25.17	16.83	9.74	4.35
Peak memory usage (Gigabytes)	8.35	6.94	6.11	5.06	3.95	3.33	3.02	2.76	2.42	1.57
Peak memory usage percentage	100.0	83.06	73.21	60.54	47.28	39.86	36.16	33.02	28.99	18.81

Efficient-GeM-LAMP

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	66.68	65.4	67.09	63.47	60.12	57.51	46.32	30.6	11.08	3.81
Recall@1 percentage	100.0	98.09	100.62	95.18	90.17	86.25	69.46	45.9	16.61	5.72
Area under recall@k curve	92.13	92.14	92.54	92.06	91.26	90.91	87.94	79.41	56.63	49.12
Area under recall@k curve percentage	100.0	100.01	100.45	99.92	99.06	98.68	95.45	86.19	61.47	53.32
Inference time (ms)	235.78	231.5	223.4	212.84	202.04	190.18	172.56	145.34	109.94	63.29
Inference time percentage	100.0	98.19	94.75	90.27	85.69	80.66	73.19	61.64	46.63	26.84
Params (millions)	6.82	5.87	5.03	4.24	3.49	2.76	2.06	1.41	0.79	0.25
Params percentage	100.0	86.02	73.79	62.18	51.11	40.42	30.15	20.65	11.66	3.61
DMAs (millions)	850.5	802.41	743.05	679.84	613.2	540.85	460.82	364.73	243.31	108.1
DMAs percentage	100.0	94.34	87.37	79.93	72.1	63.59	54.18	42.88	28.61	12.71
MACs (millions)	831.06	784.15	725.98	663.98	598.57	527.56	449.07	354.92	236.1	104.03
MACs percentage	100.0	94.36	87.36	79.9	72.03	63.48	54.04	42.71	28.41	12.52
Peak memory usage (Gigabytes)	8.35	8.35	8.34	8.29	8.18	8.03	7.72	7.35	6.02	3.5
Peak memory usage percentage	100.0	99.95	99.91	99.27	97.99	96.14	92.39	88.05	72.04	41.85

Efficient-GeM-FPGM

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	66.68	65.79	58.98	54.18	43.53	33.51	20.42	10.17	10.93	9.76
Recall@1 percentage	100.0	98.66	88.45	81.25	65.28	50.25	30.63	15.25	16.39	14.63
Area under recall@k curve	92.13	91.96	90.39	90.35	86.62	83.57	68.01	43.24	47.53	43.5
Area under recall@k curve percentage	100.0	99.82	98.11	98.07	94.02	90.71	73.82	46.93	51.59	47.22
Inference time (ms)	235.8	207.5	181.04	156.48	129.15	99.54	74.7	55.11	34.28	18.12
Inference time percentage	100.0	88.0	76.77	66.36	54.77	42.21	31.68	23.37	14.54	7.68
Params (millions)	6.82	6.29	5.67	5.0	4.28	3.48	2.67	1.95	1.23	0.55
Params percentage	100.0	92.18	83.17	73.4	62.83	50.98	39.18	28.59	18.08	8.1
DMAs (millions)	850.5	743.14	639.18	536.33	430.67	309.92	213.41	142.55	84.54	39.1
DMAs percentage	100.0	87.38	75.15	63.06	50.64	36.44	25.09	16.76	9.94	4.6
MACs (millions)	831.06	725.83	623.8	522.97	419.41	300.92	206.51	137.44	81.2	37.34
MACs percentage	100.0	87.34	75.06	62.93	50.47	36.21	24.85	16.54	9.77	4.49
Peak memory usage (Gigabytes)	8.35	7.99	7.11	6.03	4.87	3.66	2.89	2.4	1.83	1.24
Peak memory usage percentage	100.0	95.67	85.18	72.23	58.36	43.86	34.61	28.73	21.9	14.82

Resnet50-NetVLAD-L2-Norm

Sparsity Proportion	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Recall@1	86.81	86.21	85.02	82.0	79.21	75.56	64.64	52.27	23.36	18.38
Recall@1 percentage	100.0	99.31	97.94	94.46	91.25	87.04	74.46	60.22	26.91	21.18
Area under recall@k curve	96.46	96.21	96.04	95.74	95.18	94.15	92.24	90.42	72.84	68.31
Area under recall@k curve percentage	100.0	99.74	99.56	99.25	98.67	97.61	95.63	93.74	75.51	70.82
Inference time (ms)	379.98	365.99	331.12	301.64	279.89	254.13	235.09	219.27	208.6	199.94
Inference time percentage	100.0	96.32	87.14	79.38	73.66	66.88	61.87	57.71	54.9	52.62
Params (millions)	8.67	7.75	6.77	5.76	4.79	3.89	2.98	2.16	1.36	0.66
Params percentage	100.0	89.3	78.04	66.42	55.23	44.8	34.39	24.86	15.67	7.63
DMAs (millions)	3335.42	2615.09	2060.94	1626.09	1263.25	958.47	698.05	480.13	291.28	133.92
DMAs percentage	100.0	78.4	61.79	48.75	37.87	28.74	20.93	14.39	8.73	4.02
MACs (millions)	3312.12	2595.29	2044.38	1612.12	1251.66	949.06	690.74	474.57	287.29	131.22
MACs percentage	100.0	78.36	61.72	48.67	37.79	28.65	20.86	14.33	8.67	3.96
Peak memory usage (Gigabytes)	5.69	5.03	4.16	2.97	2.5	2.13	2.12	2.12	2.12	2.12
Peak memory usage percentage	100.0	88.48	73.18	52.23	44.02	37.43	37.36	37.3	37.24	37.2

B Archive Contents

1. VPR-Pruning: System designed to perform training, pruning and evaluation of VPR methods. 2.Tables: Contains the data for all the tables in appendix A in csv format. 3.Plots: Contains the LaTeXcode for all the plots used in this paper.