On Compact Deep Neural Networks for Visual Place Recognition, Object Recognition and Visual Localization

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science in Computer Science and Engineering
by Research

by

Soham Saha
201507509
soham.saha@research.iiit.ac.in
sohamsaha.cs@gmail.com

International Institute of Information Technology, Hyderabad
Deemed University
Hyderabad - 500 032, INDIA
June 2019

Copyright © Soham Saha, 2019
All Rights Reserved

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled " On Compact Deep Neural Networks for Visual Place Recognition, Object Recognition and Visual Localization" by Soham Saha, has been carried out under our supervision and is not submitted elsewhere for a degree.

Date

Advisor: Prof. C.V. Jawahar
Center for Visual Information Technology
Kohli Center on Intelligent Systems
IIIT Hyderabad, India

Date

Advisor: Dr. Girish Varma
Machine Learning Lab
Kohli Center on Intelligent Systems
IIIT Hyderabad, India

To my Parents, Mr.Partha Pratim Saha and Mrs.Sangita Saha

Acknowledgments

I would like to express my sincere gratitude to my advisor Prof. C.V. Jawahar for his continuous guidance, patience, motivation, and immense knowledge. His guidance helped throughout my research and writing of the thesis. He regularly motivated me to strive for excellent research and has driven me to continue with it even after completing my MS degree. He consistently allowed me to work freely, but steered me in the right direction whenever he thought I needed it.

I thank my co-advisor Dr. Girish Varma who guided me continuously through my research journey. His door was always open whenever I needed any kind of help and guidance and he always encouraged technical discussions. His perseverance, dedication and motivation have inspired me immensely throughout.

I would also like to thank my colleagues Viral Parekh and Harish Krishna, seniors Soumyajit Ganguly, Riddhiman Dasgupta, Koustav Mullick, Avijit Dasgupta, Aditya Arun, Saurabh Daptardar, Praveen Krishnan, Pritish Mohapatra and Dr. Ajoy Mondal for their constant support and for providing a positive learning environment. I am specially grateful to Soumyajit Ganguly, Riddhiman Dasgupta and Koustav Mullick for being a constant motivator during my toughest months and for always being there when I needed them.

I also take this opportunity to thank Texas Instruments, India, for allowing me be a part of their project which acted as a stepping stone for me to get into research in Deep Learning.

Last but not the least, I express my deepest gratitude toward my parents, Shri Partha Pratim Saha and Mrs. Sangita Saha for their unconditional support in all my endeavours and without whom the journey would have been impossible.

Abstract

There has been an immense increase in the use of Deep Neural Networks in recent times due to the availability of more data and greater computing power. With their recent success, it has been a trend to use them extensively in real-time applications. However, the size of deep models can render them incapable to be used in devices with memory-constraints. In this thesis, we explore the several neural network compression techniques for three separate tasks namely i) visual place recognition, ii) object recognition and iii) visual localization. We explore explicit compression methods for the visual place recognition task and the object recognition task, achieved by making modifications to the learned weight matrices. Furthermore, we look at compression attained through architectural modifications in the network itself, proposing novel training procedures and new loss functions for object recognition and visual localization.

The task of visual place recognition requires us to correctly identify a place given its image, by finding out images of the same place in the world(dataset). Performing this on low memory devices such as mobile phones and robotics systems, is a challenging problem. The state of the art models for this task uses deep learning architectures having close to 100 million parameters which take over 400MB of memory. This makes these models infeasible to be deployed in low memory devices and gives rise to the need of compressing them. Hence, we study the effectiveness of explicit model compression techniques like trained quantization and pruning, on one of the most effective visual place recognition models. We show that a compressed network can be created by starting with a pre-trained model and then fine-tuning it via trained pruning and quantization. Through this training method, the compressed model is able to produce the same mAP as the original uncompressed network. We achieve almost 50% parameter reduction through pruning with no loss in mAP and 70% reduction with close to 2% mAP reduction, while also performing trained 8-bit quantization. Furthermore, together with 5-bit quantization, we perform about 50% parameter reduction by pruning and get only about 3% reduction in mAP. The resulting compressed networks have sizes of around 30 MB and 65 MB which makes them easily usable in memory constrained devices.

We next move on to compression through low rank approximation for the task of image classification. Traditional compression algorithms in deep networks involves performing low-rank approximations on the learned weight matrices after the training procedure has been completed. We propose to perform low rank approximation during training itself and make the parameters of the approximated matrix learnable too by using a suitable loss function. We show that by using our method, we are able to compress a

base-model providing 89% accuracy, by 10x, with some loss in performance. Using our compression based training procedure, our compressed model is able to achieve an accuracy of about 84%.

Next, we focus on developing compressed models for the object recognition task and propose a novel architecture for the same. Deep neural networks for image classification typically consists of a convolutional feature extractor followed by a fully connected *classifier* network. The predicted and the ground truth labels are represented as one hot vectors. Such a representation assumes that all classes are equally dissimilar. However, classes have visual similarities and often form a hierarchy. We propose an alternate architecture for the classifier network called the Latent Hierarchy Classifier which can discover a latent hierarchy of the classes while simultaneously reducing the number of parameters used in the original classifier. We show that, for some of the best performing architectures on CIFAR and Imagenet datasets, our proposed alternate classifier and training procedure, recovers the accuracy. Also, our proposed method significantly reduces the parameter complexity of the classifier. We achieve a reduction in the number of parameters of the classification layer by 98% for CIFAR 100 and 41% for the Imagenet 1K dataset. We also verify that many visually similar classes are grouped together, under the learnt hierarchy.

Finally, we address the problem of Visual Localization where the task is to predict the camera orientation and pose of the given input scene. We propose an anchor point classification based solution for this task by using single camera images only. Our proposed three-way branching of the feature extractor into an Anchor Point Classifier, a Relative Offset Regressor and an Absolute Regressor, is able to achieve $<$2m translation localization and $<5^\circ$ pose localization on the Cambridge Landmarks dataset, while also obtaining state-of-the-art in median distance localization for orientation for all the 6 scenes. Our method not only uses fewer parameters than previous deep learning based methods but also improves on memory footprint as well as test-time over nearest neighbour based approaches.

Contents

List of Figures

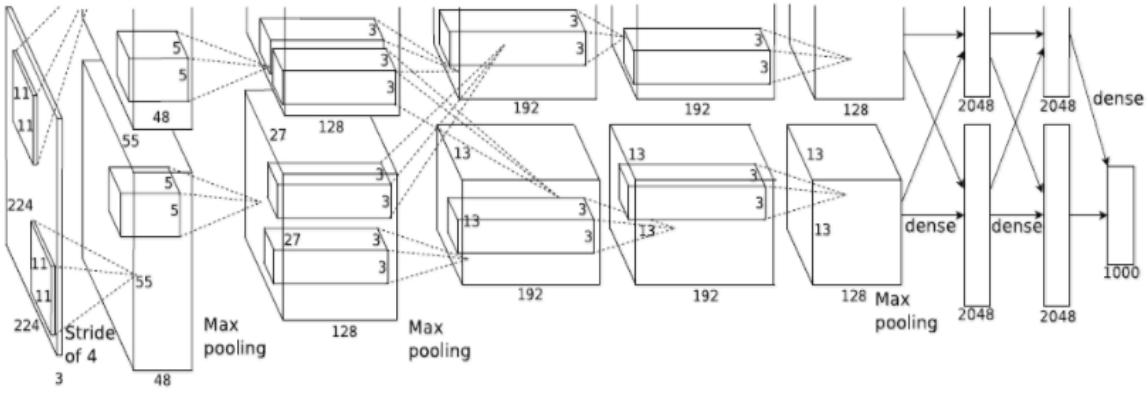
List of Tables

Chapter 1

Introduction and Background

1.1 Introduction to Model Compression

Deep Neural Networks have gained much significance over the last few years in a variety of tasks where machine learning solutions are a necessity. This has resulted in the state=of-the-art being enhanced by deep learning. The primary reasons for this boom in deep learning has been the recent availability of a large amount of data and computing power in the form of GPUs. Thus, deep neural networks are being extensively used for a lot of tasks in computer vision, text and speech processing to get the best results. The need therefore arises to use these models in real time tasks and applications used in our day to day lives. It is a tempting proposition to be able to do so. However, the larger size of the models is a hindrance to some extent, for them to be deployed in memory-constrained devices. It would therefore be extremely advantageous to have compressed neural network architectures which provide the same performance as their original uncompressed counterparts. However, the new models would be highly compressed and would only have a fraction of the memory footprint. They might also be an improvement in terms of test-time, from the initial models. It would be easier to use these models for several mobile applications since they would be easy to download with their small sizes. Compression in networks in such a way that the main memory footprint reduces itself would be another added advantage. All of these efficiencies added with the reduction in test-time of compressed networks will surely open-up exciting possibilities for them to be used in real-time applications. We show the a typical size of a CNN (Alexnet) next along-with the parameters in every layer and the number of floating point operations required during a single forward propagation.



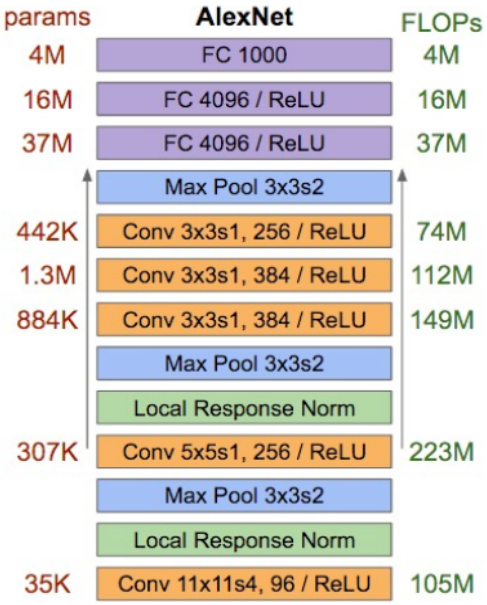


Figure 1.1 Network Architecture of Alexnet alongwith number of parameters and number of floating point operations in every layer. It is interesting to observe that while the number of parameters are more in the fully connected layer than in the convolutional layers, the convolution operations account for a considerably larger number of FLOPs than their fully connected counterparts.

1.2 Types of Model Compression

Deep Neural Network compression can be performed at different stages of the training pipeline which are in turn determined on the type of compression we want to perform on our model. In this section we will discuss the various types of Network Compression.

1.2.1 Explicit Compression methods

We begin by elaborating on explicit compression methods which involves manipulating the learnt weight matrices in an efficient manner.

1.2.1.1 Compression after Training

The training of the deep neural network is completed on the task at hand keeping all the pre-defined parameters of the network intact. Thereafter, the aim is to reduce the number of parameter of the neural net by analyzing the values of the learned weights. One of the most popular methods is perform a low rank approximation on the weight matrix itself. In order to do this, one can perform a Singular Value Decomposition [37] on the weight matrix itself and achieve a compression ratio based on the number of singular values one is willing to take into account. This type of compression method has not been found to do well in practice as explained in more details in the subsequent sections.

Another approach which is popularly followed as a compression procedure is pruning [25] [46]. In this method, the connections, nodes or filters of the neural network are removed permanently after training, based on some criterion. This makes the architecture have much less parameters than before which results in impressive compression gains. In this type of compression procedure, the results are somewhat better than in low rank approximation. We further show the effect of pruning neurons and synapses in a neural network in Figure 1.2

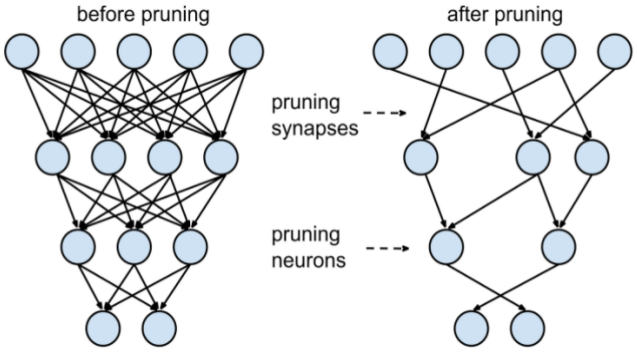


Figure 1.2 Illustration of pruning neurons and synapses in a neural network. Pruning synapses resembles removing weights while pruning neurons resembles removing the entire node of the neural network with all its connections.

Alongwith pruning, one out of the several quantization procedures, [16] is typically followed in order to boost compression performance. The key idea of quantization is to represent each weight value with a less number of bits than what was used before. This essentially means that there will less number of distinct weight values than what was present earlier. The most popular quantization techniques in literature are as follows:

1. **Uniform Quantization:** In this type of quantization, we uniformly divide the weight space into k equal partitions also known as anchors. We then assign each weight value to the nearest anchor by calculating the Euclidean distance or any other distance metric between a weight value and the anchors. Thus, the nearest neighbour strategy helps to assign each weight to the correct anchor. The number of anchors depends on the number of bits we wish to use, for representing each weight and is hence a design choice. For example, we if we choose to use 8-bits, then the weight space will be divided into 256 equally spaced anchors which means that we can have only 256 unique weights. In order to store the matrix efficiently, all we need to so is to store the 8-bit values and calculate the weight value with the help of the following equation:

$$w^i = min + \frac{max - min}{2^8 - 1}[a^i] \tag{1.1}$$

where w^i is the i-th weight value and a^i is the value of the i-th anchor point index. This is an example of uniform quantization with 8-bits.

2. **Non-Uniform Quantization:** In this type of quantization, specific weight values are shared by multiple connections in the network. Hence, here, for every index assigned, a mapping needs to be stored with the corresponding weight value. The most popular method to learn the distinct weight values is to perform k-means clustering [38]. Thus, k-means clustering is performed on the entire weight space as an unsupervised learning method and the the optimum cluster centers are allowed to be discovered. The value to be used for 'k' is a design choice in itself and depends on how many bits once chooses to represent each weight value in the network. For example, for 8-bit quantization, one should perform 256-means clustering on the weight space. It is to be noted however, that in this method, all the learned cluster centers are stored with the corresponding index of the cluster center each weight is assigned to. Thus, before a forward pass, the network needs to decompressed. However, the storage in HDD can be done most efficiently without any loss in performance. The k-means clustering is typically calculated based on the following equation:

$$min \sum_{i}^{mn} \sum_{j}^{k} ||w_i - c_j||_2^2 \tag{1.2}$$

where w_i is the i-th weight value and c_j is the j-th cluster center.

3. **Product Quantization:** This type of quantization requires a pre-processing step and post-processing step in between the non-uniform quantization procedure mentioned earlier. In this case, the weight matrix is divided into several sub-matrices say 'n'. Then, k-means clustering is applied on each of the sub-matrices independently. Now k-cluster centers of the n sub-matrices are concatenated to form a single quantized representation of the original weight matrix. The division of the initial weight matrix can be explained in the following manner.

$$W = [W^1, W^2, W^3 ... W^s] \tag{1.3}$$

where a weight matrix W has been divided into s sub-matrices.

4. **Residual Quantization:** This method is again an interesting variant of the k-means quantization. In this method, the first step is to perform the k-means clustering on the weight space as mentioned in the case of non-uniform quantization. Now, instead of assigning each weight value to a cluster center as was the case previously, we calculate the residue of a weight value from each of the cluster centers. Now, we have k-dimensional vector for each weight value. We then perform k-means clustering one he residues again and repeat this procedure. The exact time to stop this method is a choice which is to be made by the user. The typical practice is to stop when the error is less than a certain threshold. As a final step, all the k cluster centers sets are added to form a unified k cluster centers. This procedure thus serves as much better approximation to the weights of the neural network since the cluster centers are more accurate. Residual quantization reduces the accumulation of errors than was the case with non-uniform quantization. The equation for calculating the resultant weight vector is given by the following equation:

$$\hat{w_z} = c_j^1 + c_j^2 + c_j^3 + ... + c_j^t \tag{1.4}$$

where $\hat{w_z}$ is the approximated weight value and c_j represents the j-th cluster center.

1.2.1.2 Compression During Training

Compression can be achieved during training as well. All of the previously described methods of pruning and quantization can be performed during training with slight modifications. Here we will go through the details of how it can be done.

Pruning of network weights, nodes as well as convolution filter can all be done during training itself. Typically, while pruning the weights, an additional constraint is added to the loss function which makes criterion based pruning easily achievable. One way to prune the weights is to set the connections to 0 based on a threshold. This is then followed by retraining the network. Retraining the network allows for the other parameters to make adjustments as if the pruned parameters were non-existent in the first place. Thus, pruning acts as a sort of regularizer on the network parameters and highly avoids over-fitting. Instead of retraining the network only once, an alternative is to iteratively train the network for a few epochs and then perform the pruning. This iteration can be carried out till saturation. We illustrate the trained pruning algorithm in Figure 1.3

Pruning can also be performed on the nodes of a neural network itself. This is generally done by adding a regularizer to the loss function so that the incoming as well as outgoing connections from a node of neural network become as close to 0 as possible thereby rendering it useless to be used in the forward computation entirely. Such a node can be dropped from the neural network and this strategy reduces a large number of parameters eventually. The effect of dropping a particular node (synapse) is shown in Figure 1.2. In order to prune filters of a CNN, one needs to apply a suitable constraint to

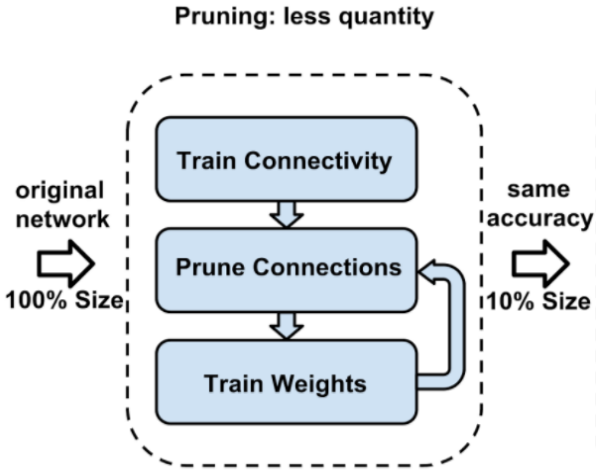


Figure 1.3 Trained pruning and its effect on the network parameters. Retraining the network after pruning allows adjustment for non-pruned weights and allows them to recover the original performance. Image Credit: Han et.al [22]

the loss function while also making the required changes to the non-linear activation. In this case, the activation function has parameters which is also made learnable and an identical activation signifies that a particular layer is learning nothing, and can be therefore dropped. CNN filters can be pruned in this manner [71].

Quantization is a popular technique in which compression is achieved by representing each weight value by a less number of bits than what was used before. Quantization performed during training is beneficial because it helps to represent the weights better and achieve a reduction in error than what was achieved while performing quantizaion after training. We have already described the most popularly used quantization techniques in literature in the previous section. All of those techniques can be performed during training iteratively. Other than those techniques, we can also perform trained quantization with weight sharing during training which we shall be describing here. Typically, the quantization pipeline during training has the procedure shown in Figure 1.4

Weight sharing is performed along-with the quantization step. First, the weight space is subjected to k-means clustering as described before in non-uniform quantization. Then, similar cluster indices are assigned to the weights as well as the gradients. The weights with the same cluster centers and the gradients with the same cluster centers are then added up to form a codebook of weights and gradients respectively. The final step is the updation of the codebook of weights with the codebook of gradients. This gives an updation in each step. This procedure of weight sharing can be performed once in between the training and then at the end or it can be performed once every few epochs iteratively. The pipeline for the weight sharing procedure with trained quantization in shown in Figure 1.5 [22]

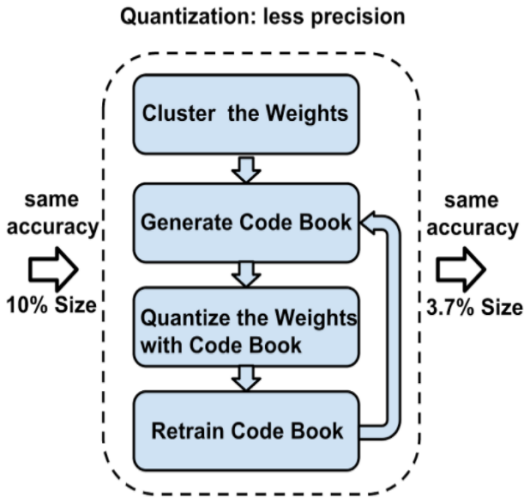


Figure 1.4 Quantization pipeline typically followed during training. The weight code book is updated through retraining. This helps to improve upon the lost performance as a result of the quantization. The model is able to recover the accuracy eventually. Image Credit: Han et.al [22]

1.2.2 Architectural Modifications

1.2.2.1 Compression Before Training

Until now, we have talked about attaining compression through modifications to the weights and connections of the DNN. However, compression can also be performed right before the training takes place as well. This is typically done through careful and more efficient design of the neural network architecture. Thus, in this case the network has fewer trainable parameters since the beginning itself and thus novelty in network design and loss function is essential, such that performance is not compromised. In this subsection, we will describe some of the architectural modifications in popular use.

1. One of the most significant works which has been done in this domain is the proposal of student-teacher networks. In this setup there are two networks- one network with a huge number of parameters referred to as the teacher network, and another network with much less number of the parameters known as the student network. The task of the student network is to mimic the outputs of the teacher network using those less number of parameters with which it has been defined initially. The training procedure followed is to first train the teacher network on a task till saturation. Then, we use the outputs of the teacher network as ground truth labels for training the student network. The outputs of the teacher network are used in place of the hard labels which are present originally in the dataset. This simplifies the task of the student network since now it only has to produce soft labels. [28] [58] [82] [42] The student teacher network setup can be clearly understood from the from Figure 1.6.

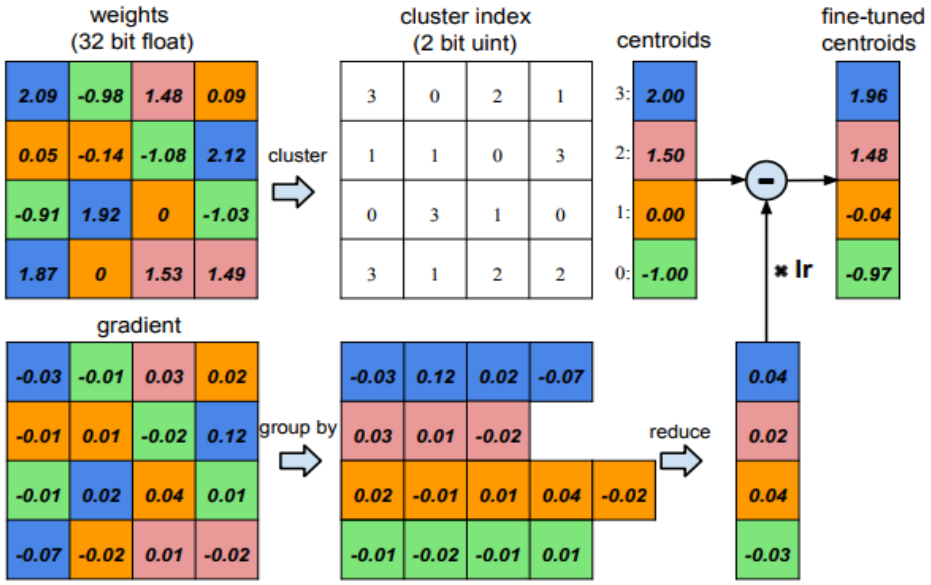


Figure 1.5 Weight sharing during scalar quantization and fine-tuning the centroids (cluster centers). The centroids are fine-tuned using the clustered gradients of the weight values. Image Credit: Han et.al

2. Traditional deep neural network have the structure of a convolution feature extractor followed by a classifier which is a series of fully-connected layers, generally. The output of the feature has multiple filters and connection of all these weights to a fully connected layer results in a parameter explosion. Almost 95% of the parameters are present in the fully connected layers in traditional deep neural network models. In order to address this problem, Global Average Pooling layer of GAP was developed which calculates the average signal of a filter before mapping the output to a fully connected layer. This is generally deployed at the end of the convolution feature extractor right before the classifier begins. Since we are taking an average signal across a filter, the parameters of the model are highly reduced. For example, for 7x7x2208 dimensional output of the feature extractor as used in Facebook's DenseNet [33], the GAP is applied o each of the 2208 7x7 filters resulting in an output of 1x1x2208. This gives a direct compression of 49x. The advantages of using GAP instead of the traditional feature extractor can be further understood in the the Figure 1.7.

3. Deep Neural Network architectures are generally over-parameterized and subject to design choices. Hence designing the optimum architecture is a challenge and may be a trial-and-error procedure sometimes. Therefore, a third method to compress deep neural networks is to learn the optimum architecture itself. This approach has found a lot of importance in recent times and is an interesting future research topic also. This can be done in two ways. The first way to modify activation functions and/or loss functions in such a way such that the layers and filter can be removed(or

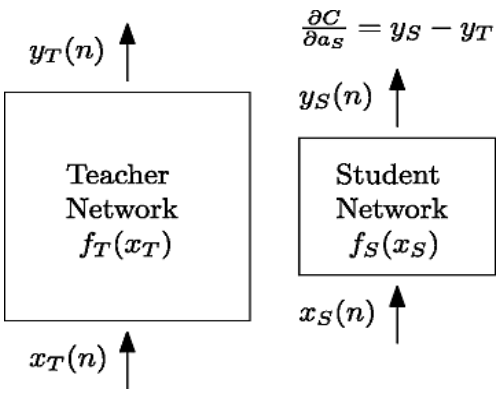


Figure 1.6 Setup of the student-teacher network. The teacher network typically has a lot more parameters than the student network. Therefore, when the student network is used only at test time, it results in a significant compression gain.

added) during the training phase itself [71]. A second way is to use a reinforced loss function which assigns penalty to the the sub-optimally performing architectures [7].

There are a lot of other architectural modifications and novel learning mechanisms one can explore in order to achieve a network compression [56] and also improve the test-time performance. This is an interesting and active research problem.

1.3 Contributions to the thesis

The contributions in this thesis can be summarized as follows:

1.3.1 Visual Place Recognition

We begin by proposing a compression pipeline for the visual place recognition task. This task typically involves recognizing the place from its corresponding image by matching it with similar existing images of that place in the dataset. Typical solutions use a discrete landmark assignment based approach where landmarks for a particular dataset are predefined and an input scene location is identified based on those landmarks. Deep learning based solutions for this task consist of huge models and developing compressed and efficient solutions is a necessity.

We propose a compression pipeline for this task which can be achieved through end-to-end training. Our solution involves performing trained pruning of the neural network parameters followed by trained quantization of the same. We perform this iteratively during training which allows enough time for the network to adapt itself using the non-pruned parameters in such a way that it does not compromise on the performance. We also show that enforcing sparsity in the loss function and performing the steps iteratively are essential for the method to be effective.

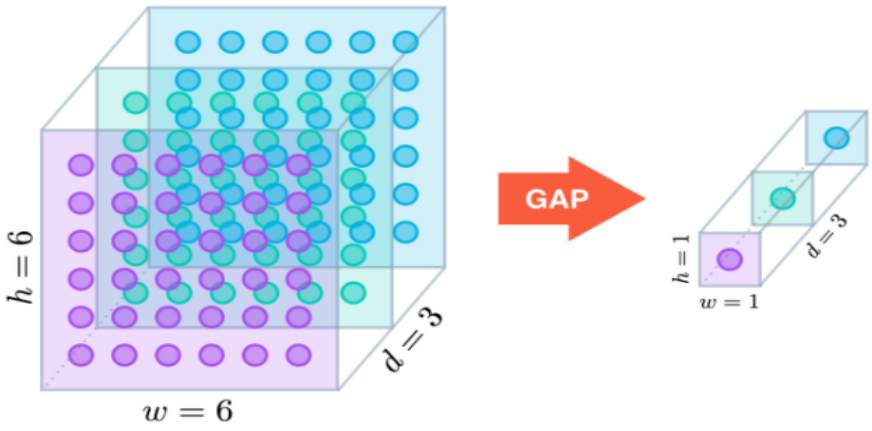


Figure 1.7 Parameter Reduction is the most important advantage of using a Global Average Pooling layer before applying a fully connected classifier to the network. This has been demonstrated here as a 6x6 average pooling is applied across each feature map to get a 3 dimensional vector. Image Credit: Alexis Cook

Our results highlight the effectiveness of the method. We use a pre-trained neural network for the visual place recognition task and fine-tune on different datasets to report our results. The compressed model is able to achieve about 8x compression overall without any loss in performance and about 12x compression gain with only 2% loss in performance. The resultant models have sizes of about 30MB and 65MB which makes it feasible for them to be used on mobile devices.

1.3.2 Object Recognition

This is one of the most fundamental tasks of computer vision in which for a given an image, containing an object, the task is to identify the object present in the image. Alternatively known as image classification, this task uses a Convolution Neural Network architecture to achieve state of the art results. We propose explicit as well as architectural modifications for these models.

The first approach which we take, is attaining compression through low rank approximation. We propose to make the approximated weight matrices trainable end-to-end using a modified loss function such that the smaller matrix can adjust its parameters in order to provide for a better approximation. This procedure provides us with significant compression gains. Secondly, we propose to the fully connected classifier typically present in CNN based solutions for the object recognition task. Our proposed classifier called the Latent Hierarchy Classifier attains compression by learning an efficient embedding of the label space while also learning inherent latent hierarchy present in classes of the dataset.

We show that through low-rank approximation, we can compress a base model providing 89% accuracy on the CIFAR-10 dataset by 10x. The compressed architecture is able to achieve an accuracy of 84 % on the same dataset. Using our novel Latent Hierarchy Classifier, we show that we are able to recover the accuracy for some of the best performing architectures on the CIFAR-100 and the Imagenet

1K dataset. We achieve a reduction in the number of parameters of the classification layer by 98% for the CIFAR-100 dataset and by 41% for the Imagenet 1K dataset. Thus, we empirically prove that our classifier is efficient alternative to the existing ones. Moreover, it is verified by us that visually similar classes are grouped together through this training procedure.

1.3.3 Visual Localization

The task of visual localization involves predicting the camera orientation and pose from the image of a given scene. In order to do this, regression and nearest-neighbour based methods are generally followed. We propose an efficient architecture design for this purpose which improves the state of the art for camera orientation localization on the Cambridge Landmarks dataset.

We propose an anchor point classification based novel architecture for this task. Our architecture consists of a pre-trained feature extractor followed by an anchor point classifier, a relative offset regressor and an absolute regressor. This three way network is combined using a novel differentiable loss function with multiple components. This training procedure of ours helps to attain significant improvements on the regression based methods in terms of performance and over traditional nearest neighbour methods in terms of memory and test-time.

We report significant results for the 6 scenes of the Cambridge Landmarks dataset. We are able to achieve $<$2m orientation localization and $<5^{\circ}$ camera pose localization for the scenes. This gives us state of the art results for camera orientation localization for all the 6 scenes. Our architecture is therefore efficient in terms of performance, over previous deep learning solutions for this task on this dataset.

1.4 Organization of the Thesis

The rest of the thesis is organized as follows:

1. In Chapter 2, we discuss about the iterative trained pruning and trained quantization approach for network compression, for the Visual Place Localization task.

2. Chapter 3 talks about the trainable low rank approximation method for the weight matrices of the CNN.

3. In Chapter 4, we focus on a new architecture for the object recognition task, where we aim to replace the fully-connected layer with the latent hierarchy classifier and achieve significant compression ratios as well as learn a multi-level class hierarchy.

4. In Chapter 5 we present a new architecture for the Visual Localization task where anchors are used for the ease of localizing a place correctly. We also propose a new loss function for this task here.

5. In Chapter 6, we conclude our work and findings, as well as provide directions for future work.

Chapter 2

Compressing Deep Neural Networks for Recognizing Places

2.1 Introduction and Related Work

Recent approaches in solving pattern recognition problems are focused on deep learning techniques. Visual place recognition in computer vision is one such task, which is typically cast as an image retrieval problem, where a database of images is queried using a query image and the system produces relevant images. Some of the major challenges for the visual place recognition task is to correctly retrieve the image of the desired place even though it has a different perspective or is under a different ambient lighting. Thus we would want our retrieval system to be scale and light invariant.

Traditionally, this problem is solved by extracting feature vectors using the BOF or SIFT representations and finding the approximate nearest neighbours of the query image in this feature space. Another popular method, the vector of locally aggregated descriptors (VLAD) was originally proposed by Jegov et al. [36] as an improvement over BOF representation of images and the Fisher kernel vector representation. This method essentially provides a vector representation of images and is quite similar to the Fisher kernel representation. It aggregates the descriptors based on a locality criterion in the feature space. Arandjelovic et al. [3] improved on VLAD by introducing vocabulary adaptation and intra-normalization. Over the years, contributions have been made on quality improvement of the aggregated features, optimizing the indexing scheme and other modifications for improving the performance of VLAD in [70], [10].

Starting with the work of Krizhevsky et al. [44], there has been a significant improvement in the performance of various computer vision tasks by using deep neural networks. For the task of image retrieval, Arandjelović et al. [2] proposed a deep learning model that can be trained in an end to end fashion. Their architecture consists of the convolution layers of popular deep image classification models like Alexnet or VGG16 (Simonyan et al. [69]). This is followed by a NetVLAD layer (see Section 2.2) which has trainable parameters unlike the original VLAD. Finally, they also have a PCA whitening layer that reduces the dimension of the feature vectors.

The recent trend for improving the performance of computer vision tasks has been to train deep learning models with an increased number of parameters. For example the popular Alexnet has 60

million parameters and requires about 240 MB of memory while VGG16 has 130 million parameters and has takes around 500 MB of memory. However, these models need to be deployed in memory-constrained devices like a mobile phone or a robotics system. Hence, there is a need for reducing the size of the models without deteriorating the performance as well as reducing the test-time evaluation (Denton et al.[13]). In this work, we address the former.

Gong et al. [16] studied some of the standard quantization approaches for storing network weights in compressed format after the training process is completed. Through parameter pruning, we want to remove those parameters which do not have an adverse effect on performance of task at hand. This method of avoiding incorrect pruning was proposed by Geo et al.[19] where they implement on-the-fly connection pruning followed by splicing. Recently, Han et al. [22] proposed a combination of pruning of weights and quantizing them during the training process itself since such huge models are likely to contain many redundant parameters. They further reported the energy savings in the compressed networks in [20].

In our work, we study the effectiveness of trained pruning and quantization methods (see Section 2.3) proposed by Han et al., for compressing the NetVLAD model (see Section 2.2) of Arandjelović et al. [2]. We demonstrate our results on the Oxford and Paris buildings datasets (see Section 2.4). We implement trained pruning and trained quantization iteratively while fine-tuning the original network. This makes the network drop the redundant parameters as well as allows it to fine tune the other parameters so that the network can learn to deliver a similar performance as before by making do with the existing parameters. Such a compact network finds importance in enabling better performance in small memory devices. While performing 8-bit quantization, we can achieve about 50% reduction in parameters (factor of 8X) with same performance and about 70% reduction (factor of 12X) with just about 2% reduction in performance (see Section 2.5).

Han et al. [22] performed pruning, quantization as well as Huffman encoding to achieve a compression ratio of 35X. However, Huffman encoding is only useful while saving to disk. The network needs to be uncompressed before doing a forward pass during testing. In this work, we are more concerned about the memory required during the forward pass and do not apply the Huffman encoding technique. Hence our compression rate of 12X is lower that the 35X as claimed by Han et al. [22]. Moreover, we are doing it for the task of image retrieval rather than for classification as in the case of Han et al. [22].

2.2 NetVLAD

Vector of Locally Aggregated Descriptors (VLAD) is a well-known method used for instance level retrieval. Here we first compute a feature descriptor for each image and define k cluster centers from among the descriptors. This is followed by the computation of sum of residuals i.e the difference between the feature descriptor for each image and each of the cluster centers. Thus, if we have a d dimensional feature descriptor, we will have k residues each of d dimension. These vectors are then

concatenated resulting in a $k \times d$ length descriptor for each image. This is followed by vocabulary adaptation and intra normalization, the details of which are mentioned in [3].

The VLAD comprises a series of operations which were presented in the form of a Generalized VLAD layer for neural nets called NetVLAD by Arandjelović et al. [2]. The input to the NetVLAD layer (denoted by x_i where i is a spatial location) is the convolutional features of dimension $H \times W \times D$ which is considered as $N = H \times W$ D-dimensional vectors. It has some trainable parameters W, b and cluster centers C having dimensions $K \times D$, K and $K \times D$ respectively. The output feature vector denoted by V (of dimension $K \times D$) is given by the following equation:

$$V(j,k) = \sum_{i=1}^{N} \left(\frac{e^{w_k^T x_i + b_k}}{\sum_{k'} e^{W_{k'}^T + b_{k'}}} \right) (x_i(j) - c_k(j)) \tag{2.1}$$

Figure 2.1 shows NetVLAD layer.

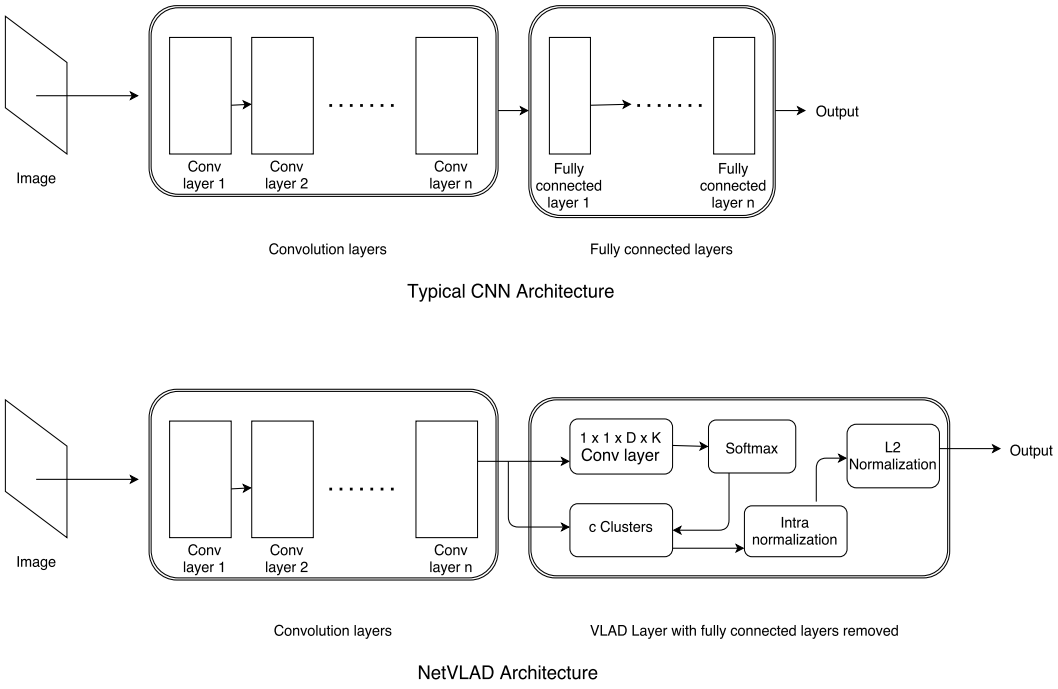


Figure 2.1 CNN + Fully Connected Layer (top) and NetVLAD architecture (bottom). Traditional image classification methods use a feature extractor followed by a fully connected layer (top). NetVLAD proposed to replace the FC layer with a layer performing the VLAD operations.

2.2.1 Triplet loss

The standard loss function used for retrieval problems is the triplet loss. This loss takes as inputs the feature vectors corresponding to a query image as well as a positive and negative vector corresponding to that query. In our experiments, for each query, the positive example is a relevant image (typically a

image of the same building with a different perspective or lighting conditions) and a negative example is randomly sampled from the rest. The triplet loss is defined in equation 2.2.

$$L(q,p,n) = \frac{1}{2}h\left(m + \|q-p\|^2 - \|q-n\|^2\right) \tag{2.2}$$

Here, q denotes the query, p is a positive example for q and n is a negative example for q. m denotes the margin which is usually taken to be 1. h denotes the hinge loss where $h(x) = max(0,x)$. Figure 2.2 illustrates how this loss affects the feature vectors while training.

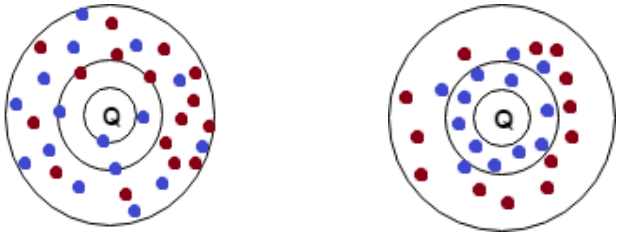


Figure 2.2 Toy image showcasing the distribution of the samples before and after the fine tuning using triplet loss. The red/blue dots indicate negative/positive examples. The innermost circle resembles the boundary for the Query image. The outermost circle is imaginary resembling the maximum distance a sample image can be from the query. The middle circle is indicative of the margin which we use to separate the positive samples from the negative samples.

2.3 Compression

We perform *trained pruning* and *trained quantization* iteratively on some networks for the task of visual place recognition and report the performance on well known datasets. We explain these compression techniques in the subsequent sections.

2.3.1 Trained Network Pruning

Network Pruning has been widely used for compressing neural nets as shown by LeCun et al. [46]. They help to reduce the number of parameters thereby reducing complexity and avoiding redundancy.

We build on top of these approaches. Our aim is to drop all connections which have a value less than some threshold. We start with a model which has been trained to a desired precision on the visual place recognition dataset using the architecture mentioned in Section 2.2. Now, we must ensure that the network learns to do its task even after the parameters have been dropped, and in order to maximize it we must make the network learn in such a way that more parameters are closer to zero. In other words, the weight matrix should be sparse. In order to achieve that, we add a ℓ_1 regularizer to the weights which ensures that sparse matrices are learnt. This works better than using ridge regression as regularization i.e using a ℓ_2 regularizer. These have been demonstrated in Tibshirani et al. [77] and Hoerl et al.[31]. Thus, we add the ℓ_1 regularizer to every layer in our network.

The weight regularization can be framed as follows:

$$J(\theta) = Loss + \lambda \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} |W_{ji}^{l}| \tag{2.3}$$

where λ is a tuning parameter and can be considered as a hyperparameter, l is the layer number in the neural network architecture, n_l is the number of layers and s_l is the number of hidden units in the lth layer. j, i are indices of the weight matrix W in the lth layer. The effect of adding the regularizer can be viewed in Figure 2.3

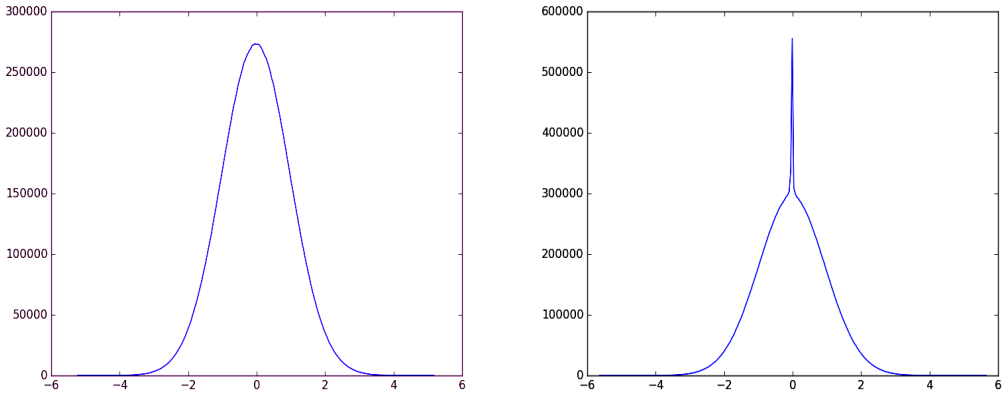


Figure 2.3 Distribution of weights before and after regularization. X-axis denotes the values of weights while Y-axis denotes the frequency of weights. The effect of adding a sparsity constraint (regularizer) to the loss function can be understood from the way the weights skew towards 0 (right).

After training the network by backpropagating the gradients from the loss function for some time, we prune the network connections below a threshold and retrain the network using the new modified loss function for a few epochs. This is followed by trained quantization which is described in the next section. This iterative process of pruning weights, quantizing and retraining is repeated for a certain number of epochs until the loss stabilizes. We ensure pruning by altering the gradients for the already pruned parameters to zero after every pruning operation. If any other parameters enter the threshold as a result of the gradient updates, then they are also pruned in the next iteration. Retraining the network ensures that the other parameters are able to compensate for the drop of the existing parameters. The parameters which are not pruned are able to adapt accordingly after realizing that the pruned parameters are non-existent.

An important hyperparameter which we need to fix is the threshold. Let us denote it by θ. This is significant because it determines the range in which weights will be dropped. Since our aim is to make the weight matrix as sparse as possible without compromising on the original mAP, we experiment with several values of θ ranging from 0.1 to 0.001 as long as it sparsifies the weight matrix without loss in

mAP. We prune the weights according to the following rule.

$$W_i = \begin{cases} W_i & W_i < -\theta \\ 0 & -\theta \leq W_i \leq \theta \\ W_i & \theta < W_i \end{cases} \tag{2.4}$$

where W is the weight matrix and θ is the threshold value.

2.3.2 Trained Quantization

This step is carried out after every pruning step. The way in which we compress the network is by reducing the number of bits required to represent each weight. The trained pruning step is accompanied by weight sharing where we reduce the number of effective weights being stored, by having multiple connections share the same weight.

The steps for trained quantization are listed below:

1. Perform Network pruning on the already trained network as mentioned in Section 2.3.1
2. Use k-means clustering for non-uniform quantization on the weights. For 8-bit non uniform quantization we shall get 256 cluster centers.
3. Now, for each time we update the weights, we accumulate the gradients having the same cluster indices, and add them up. Then we add these gradient cluster centers to the original codebook of cluster centers. This codebook is what we need to store in memory.
4. At the end, we replace the original weights with these 256 distinct updated cluster centers.

   Figure 2.4 gives an overview of the process.

2.4 Dataset and Experiments

We use transfer learning for our experiments i.e we use a pretrained network which has been trained on the visual place recognition task and we fine tune it on *one of Oxford and Paris Buildings* dataset and *test on the other*. This is helpful because it uses the knowledge of a previously learnt task and uses it for a similar task, but on different data. This fine tuning is required because it lets the network learn how to perform the visual place recognition task with less number of parameters. Some of the landmarks present in the Oxford and Paris Buildings dataset are shown Figure 2.5

We test our results in two ways and report the same. First, we fine tune on Oxford Buildings dataset and validate the performance of our method by using the Paris Buildings dataset. Then, we repeat this process by using Paris Buildings for fine tuning and Oxford Buildings for testing. This approach is justified due to the following reason. There are 55 queries in each of these datasets. Had we made a

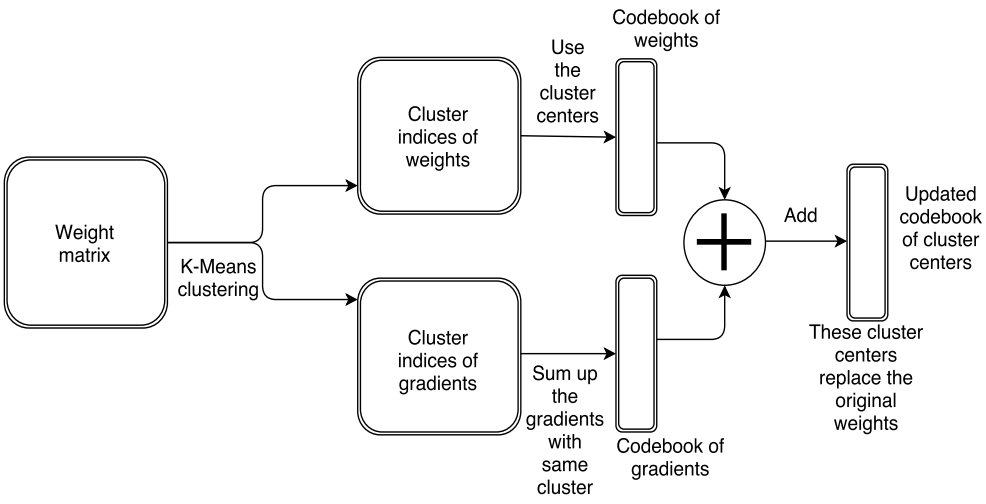


Figure 2.4 Steps involved in Trained Quantization: k-means clustering produces a weight cluster and a gradient cluster which are then summed up to produce two separate codebooks. The final step involves the weight codebook update with the gradient codebook update.

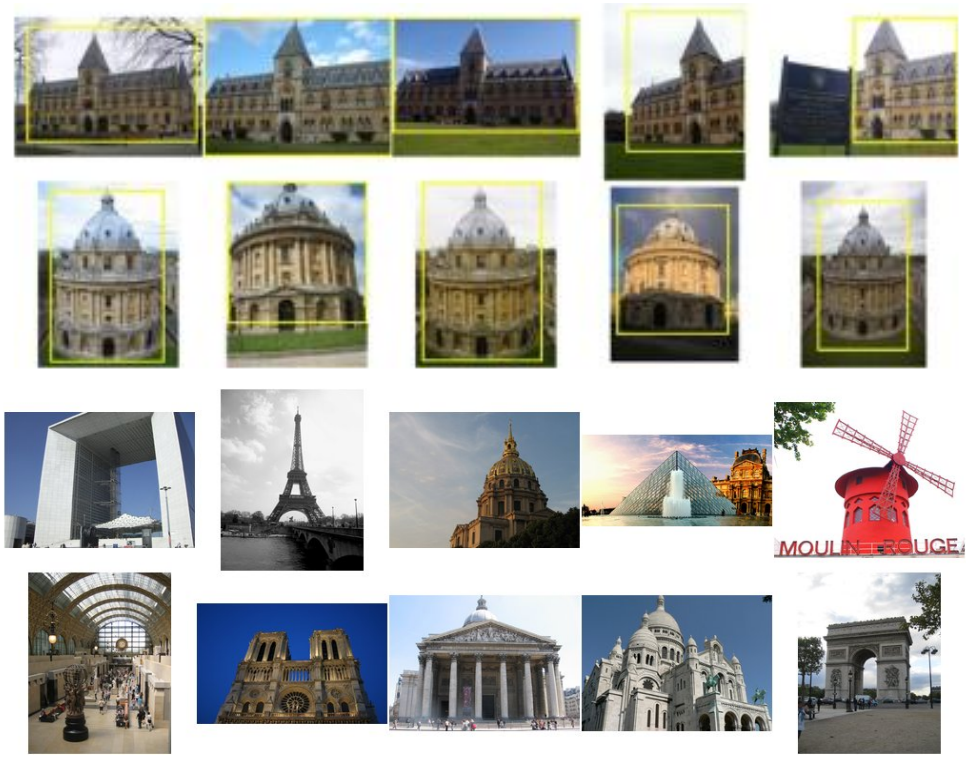


Figure 2.5 Landmarks present in the Oxford (top) and Paris (bottom) Buildings dataset. The datasets are challenging because they contain images of the same landmark taken from different viewpoints and in varying illumination.

split of one dataset say into fine tuning and testing query sets, and reported our performance on that, it would not have reflected a realistic scenario. This is because once the model has been deployed on a mobile phone or any other memory constrained device for that matter, there is no further scope of fine tuning. Thus, recording the performance on a completely new dataset such as that which has not been seen by the network beforehand makes much more sense.

We study the effect of the compression techniques mentioned in Section 2.3 on the following pretrained NetVLAD models Arandjelović et al.[2].

1. Alexnet + NetVLAD : Consists of the convolutional layers of Alexnet (pretrained on Imagenet), followed by the NetVLAD layer and a whitening layer, pretrained on Pittsburgh 30k dataset.

2. VGG16 + NetVLAD : Consists of the convolutional layers of VGG16 (pretrained on Imagenet), followed by the NetVLAD layer and a whitening layer, pretrained on Pittsburgh 30k dataset.

We report our results on the following datasets

1. Oxford 5K : Images of Oxford buildings [53].

2. Paris 6K : Images of Paris buildings. [54]

Both the datasets, contain search results corresponding to 55 query images. There are 'good' and 'ok' images in each dataset which are considered as positive images for a query in our triplet loss function. The 'junk' images and other non-positive images for each query are considered as negative images. As mentioned above we use one of these for fine tuning and the other for testing.

We observe the change in mAP for several values of threshold which results in different number of parameters being discarded (Figure 2.6). From such a plot, it is easier for us to determine what optimum value we should select for the θ.

Table 2.1 enlists the results of trained pruning and quantization on aforementioned networks which we used.

2.5 Results and Discussion

We show that our method is able to reduce the memory usage of the network with negligible loss in precision. We are able to prune close to 50% of the parameters with no loss in mAP and almost 70% parameters with only 2% drop in mAP while fine tuning with 8-bit quantization. Overall, this corresponds to 8X and 12X compression rates respectively. Also, fine tuning with 5-bit quantization allows us to attain about 50% parameter reduction with about 3% mAP drop and this corresponds to about 12X compression rate. These results are reported in Table 2.1.

Table 2.1 Compression results on Alexnet and VGG16 with NetVLAD and PCA whitening pre-trained on Pittsburgh30k dataset. The time taken per query is about 0.31 seconds in each case. Arandjelović et al. reports the MAP for Oxford Buildings and Paris Buildings as 69.8% and 76.5% respectively with Alexnet+NetVLAD and 71.6% and 79.7% respectively with VGG16 + NetVLAD which we use as a baseline for our experiments.

Method	Threshold for pruning	Percentage of Parameters Pruned	Drop in MAP (Oxford Buildings)	Drop in MAP (Paris Buildings)	Memory usage (MB)
Alexnet + NetVLAD + whitening (base model)	0	0	0%	0%	248.6
8 bits quantization	0.001	25.77	0%	0%	41.4
	0.005	48.44	0%	0%	32.4
	0.01	69.92	2.1%	1.8%	20.0
	0.05	85.77	14.2%	13.3%	10.3
5 bits quantization	0.005	52.39	2.9%	3.4%	19.5
	0.01	74.95	7.3%	6.7%	10.6
VGG16 + NetVLAD + whitening (base model)	0	0	0%	0%	529.5
8 bits quantization	0.001	25.52	0%	0%	89.6
	0.005	51.77	0%	0%	65.1
	0.01	68.23	2%	2.1%	40.5
	0.05	84.68	11.8%	14.1%	21.7
5 bits quantization	0.005	55.77	2.2%	3.6%	42.1
	0.01	75.66	6.8%	5.6%	21.2

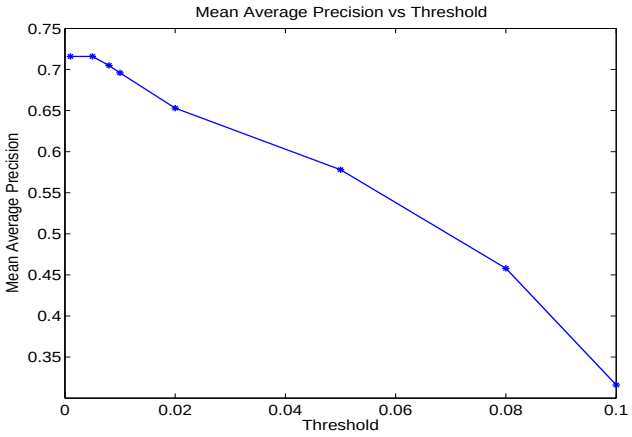


Figure 2.6 This plot shows the Drop in MAP (final performance) with the variation in Threshold used for pruning in VGG + NetVLAD on the Oxford Buildings dataset. As the threshold is increased beyond a certain point, more parameters are pruned thereby explaining the performance drop.

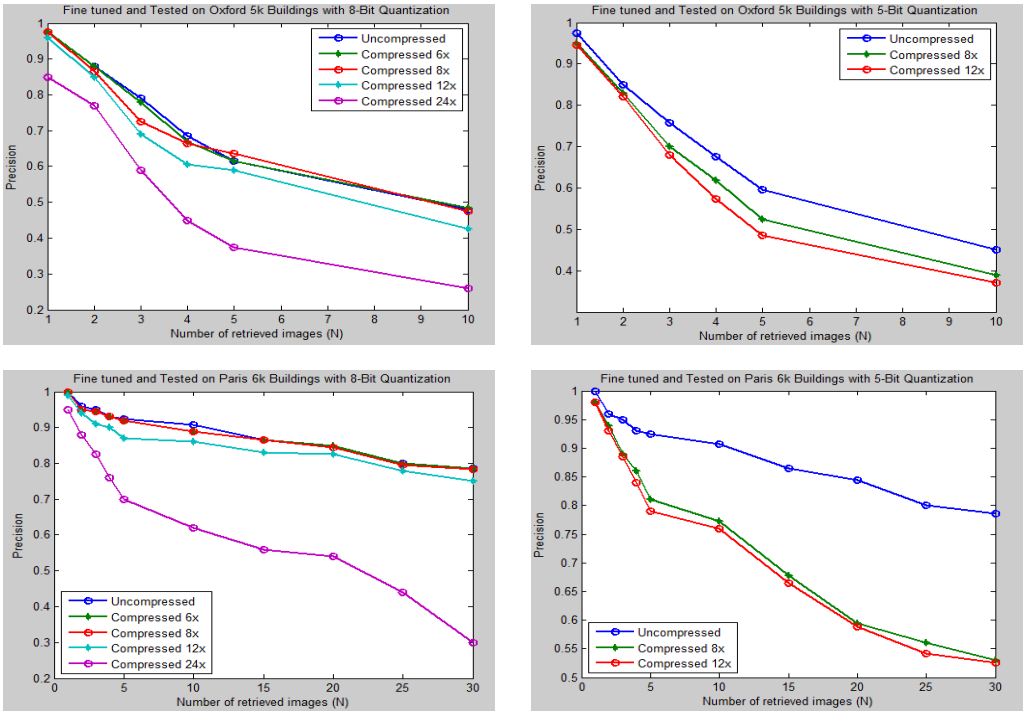


Figure 2.7 Results for VGG16 + NetVLAD + whitening pre-trained on Pittsburgh30k dataset. Plots contain the corresponding titles.

2.5.1 Alexnet+NetVLAD

We evaluate the performance of our compressed network by fine-tuning with Oxford Buildings dataset and testing our performance using Paris buildings dataset and vice-versa. We report the mAP and the corresponding plots. In this architecture, there are 5 convolutional layers followed by a NetVLAD layer. Additionally, a whitening operation after the NetVLAD layer is used. We show that the original network size is 248.6 MB and it can be compressed to 32.4 MB without loss in precision. Furthermore, we show that it can be compressed to about 20 MB with only around 2% loss in precision. Both of these are achieved while performing 8-bit quantization. These compression ratios enable the network to be used in mobile applications and real-time systems.

2.5.2 VGG16 + NetVLAD

We also look at compressing VGG16 + NetVLAD network since the bigger size of this network requires for it to be compressed even more in order to be used in a mobile application. This network with PCA whitening has a size of about 528 MB. However, we compress the network to about 65 MB without any loss in precision. The network can further be compressed to about 40 MB with about 2% loss in precision. The overall compression achieved is 12X with only 2% loss in performance.

It should be observed that when we are pruning the network an important hyperparameter is the value of θ used for pruning. This determines what percentage of weights will be discarded in the network and

eventually has an impact on the final performance of the network. We observe that the mAP reduces drastically with increasing threshold value in Figure 2.6. This result also holds true intuitively since a higher threshold value means that more number of parameters are dropped according to Equation 2.2.

The plots for the experimental results are shown in Figure 2.7.

Additionally, we show some of the qualitative results for the retrieval of similar landmarks on the Oxford Buildings dataset using the VGG16 + NetVLAD architecture in Figure 2.8.

Figure 2.8 Qualitative results on the Oxford Buildings dataset. The yellow boundaries symbolize the query, the green boundaries symbolize correctly retrieved images while the red boundaries represent incorrectly retrieved images for the query landmark.

2.6 Summary and Contributions

Deep learning models have improved the accuracy of various pattern recognition and machine learning tasks. However, the improvements were achieved by using models with increasingly larger number of parameters, making them infeasible to be run in memory constrained devices. Hence, the problem of compressing these models assumes significance. In this work, we studied the effect of model compression techniques like quantization and pruning for the task of visual place recognition. We succeed in reducing the model size from 248.6 MB to 32 MB and from 529.5 MB to 65.1 MB without any drop in MAP. This enables our model to be be run locally on memory-constrained devices rather than sending the image to a server, thereby avoiding additional latency. Thus,during processing, we do not need to send the image to the server.

From our experiments, we show that a greater percentage of parameters can be pruned with 5-bit quantization primarily by retraining the network iteratively. Also, the iterative retraining of the network with pruning and quantization is significant as it allows the un-pruned network parameters to adapt themselves to the parameter modifications.

We therefore make the following contributions in this work. We propose an iterative trained quantization and pruning procedure in order to achieve deep neural network compression while maintaining performance. A loss function enforcing sparsity is found to be beneficial by us for this type of training procedure. Finally, we address the task of visual place recognition and conclude that our compressed architecture is much more suitable to be used in memory-constrained environments than its counterparts since we report comparable results for the same.

Chapter 3

Learning Low-Rank Approximations for Model Parameters

In this chapter we will discuss about the experiments on attaining model compression through low rank approximations. We observe and analyze the results when these parameters are made trainable end to end.

3.1 Introduction

As we discussed in the previous chapter, the size of deep neural networks can make it problematic for them to be deployed on real-time systems where a high test-time and model size can be a hindrance. Low rank approximations have been popularly used for mathematical modelling and data compression. It is typically framed as an optimization(minimization) problem in which the cost function tries to measure the fit between the input data matrix and the approximating matrix. The constraint for this optimization problem is that the approximating matrix should have a reduced rank. The reduced rank enables in obtaining a compressed representation for the input matrix. Neural Network compression through low rank approximation has been a popularly studied approach for compressing memory intensive models, where the matrices considered are the weight matrices of the network. From among the different low rank approximation algorithms available in literature, the one which has received prime importance for neural network compression is Singular Value Decomposition (SVD). SVD provides for an optimum approximation of a matrix from among the various other approximation techniques. Hence this has been extensively studied in neural network compression with not such impressive results, however.

Denton et. al. in 2014 [14] first proposed low rank tensor approximations on deep neural networks. They proposed a simple method. They start by performing low rank approximation on the convolutional layers and then they fine-tune the upper layers in order to ensure that the performance of the network is restored. In the current work, we experimented with an extension on that work. We first train a deep neural network to saturation on the image classification task. Then for each learned weight matrix in the network, we perform a low rank tensor approximation and add the Frobenious norm of the residue between the original weight matrix and the approximated weight matrix to the loss function. This makes sure that the approximated matrix acts as a sort of regularizer. We further include a sparsity constraint

in our loss, the significance of which is explained later. In the final step, we fine-tune the entire network with all the sub-parts of the loss function. Our aim is that the approximated matrix will be able to recover the performance of the original network without any loss in accuracy during test-time. We are able to achieve decent results with this method although significant improvements on existing works is not attained. We provide further analysis of our method in the subsequent sections.

3.2 Related Works

The necessity of having a compressed network architecture provide similar results as its uncompressed form to render it usable in memory constrained devices has driven research in this direction in the last few years. Jaderburg et. al [35] exploited low-rank decomposition of Convolution tensors for scene text character recognition proving that low rank tensor approximations are applicable for a variety of tasks. Denton et. al [14] explore the same for the image classification task and show that a low rank tensor approximation followed by fine-tuning of the initial layers can recover the performance considerably. They do this on a considerably larger network than was done previously. They are able to achieve 5x compression rate on the convolutional layers with about 0.7% increase in error rate. They evaluate their performance layer-wise by compressing each layer and reporting the error increase as a result of that. The total error for all the layers is then accumulated to give a total error for the compressed network. For the fully connected layers they compress between 2x-13x with about 2% increase in error. Han et. al [22] and Xie et.al further provide comparisons on how low rank approximations fare with other compression methods.

3.3 Methodology

Although the Singular Value Decomposition is the best low rank approximator for a given matrix, Han et al. [22] show that there is a significant drop in performance when one tries to achieve 10x compression with this method. The performance cannot be recovered even by retraining the initial convolution layers. Hence our aim was to propose a method which would allow for learning the optimum low-rank approximation by making the parameters of the approximated matrix end-to-end trainable itself. This would allow the smaller matrix to adapt its parameters to best approximate the original matrix and recover whatever information has been lost, as a result of the approximation. Thus, making the approximation end-to-end trainable allows for adjustment of the weights through fine-tuning. We propose the following training steps for achieving this:

1. We first train a CNN end-to-end till saturation for the object recognition task on one of the popular image classification datasets. We use the popularly used cross-entropy loss for this, which is defined as follows:

$$H(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \tag{3.1}$$

2. Next, we modify the loss function to include the parameters for the approximated matrix. As we know the SVD of any matrix is given by the following equation:

$$W = UDV^T \tag{3.2}$$

where W is a nxd dimensional matrix, U is an nxn orthogonal matrix, V is a dxd orthogonal matrix and D is an nxd diagonal matrix with non-negative entries with the diagonal entries stored from high to low.

The residue obtained from the difference of the original matrix and the approximated matrix would give the error as $Error = W - UDV$. Now, in order to improve our approximation, we allow a highly sparse matrix S to be added to the original approximation and having the same dimension as the weight matrix W. Then, the overall approximation would amount to $Approximation = UDV + S$, where S is highly sparse. This approximation is now subtracted from the original weight matrix and its Frobenius norm is added to the loss function. The sparsity constraint would benefit us highly interms of memory used, while saving the model. Thus, we get the following equations so far:

$$Error = Original - Approximation = W - UDV - S \tag{3.3}$$

$$Loss = H(y, \hat{y}) + |W - UDV - S|_2^2 \tag{3.4}$$

Now, as mentioned earlier, it is required that the matrix S should be highly sparse. Therefore, we should apply an additional constraint to the loss function in order to account for sparsity. It is known that the least absolute deviation regressor or the lasso is commonly used in literature to enforce sparsity and we do the same here as well. Thus, we add the L1 norm of the matrix S to our loss function since we require it to become sparse during training itself. The overall loss function now becomes the following:

$$Loss = Cross\_Entropy + Error + Sparsity\_Constraint \tag{3.5}$$

$$Loss = H(y, \hat{y}) + |W - UDV - S|_2^2 + |S|_1 \tag{3.6}$$

3. Using this loss function in Equation 4.6 we train the entire network end-to end. The parameters of U,D,V and S are all trainable which accounts for the approximation to recover the error to some extent. The sparsity of S ensures that the storage of that matrix in memory, is highly inexpensive.

3.4 Dataset and Experiments

We experimented using the CIFAR 10 dataset which is an extremely popular image classification dataset. It consists of 32 x 32 colour images and presents a 10-class classification problem. We used the LeNet architecture with batch normalization and dropout for our experiments. Our baseline model was able to produce an accuracy of 88.9% on the test set. In this section , we provide a description of the experiments performed with our proposed method, in the directions of recovering this accuracy even with the compressed model. We performed 3 variants of the method described in the previous section.

1. We perform the SVD of every weight matrix independently and store them as weight parameters along with the original weights. The sparsity matrix is initialized with the residue between the Original matrix and the Approximated matrix. It goes without saying that all these parameters are learnable. We were able to reach an accuracy of about 84% with this procedure.

2. In a different experiment, we tried training the weights of the approximated matrices from scratch instead of initializing them with SVD Factorization. In this experiment we were able to achieve an accuracy of around 73%.

3. Finally, we tried training the entire network from scratch instead of using transfer learning and pre-training the base model, with only the cross entropy functions. Thus, in this case the weights of the neural network as well as approximated parameters were subjected to joint optimization. For this method we included an additional L1 loss between the original weights and the approximated weights. With these modifications, we achieved a performance of around 71 %.

From these experiments, we can clearly conclude that pre-training the neural network is the best thing to be done for such a procedure as shown by the first experiment. Even with using an additional L1 loss between the original weights and the approximated weights in the loss function, we were unable to get any further improvement in the above mentioned results.

An important point worth noting is that we use a weighted loss function which essentially means that we assign weights to the different components of our loss function. These weights of the different loss components are hyper-parameters and getting the optimum result is subject to critical tuning of these hyper-parameters. This is because the overall optimization is incredibly sensitive to the different components used in the loss.

3.5 Results and Discussion

In this section we report the results for the experiments conducted in a way as mentioned in the previous section on the CIFAR 10 dataset. The accuracy of the base model is around 89%. We show that we are able to achieve 10x Compression as well reach around 83.7% accuracy by using a pretrained CNN and when the approximated weights are initialized with SVD. When the approximator initialized

randomly, we can achieve 10X Compression while achieving an accuracy of about 72%. However, when the weights of both the CNN as well the approximator are initialized randomly, we are able to achieve only 5X Compression with about 72% accuracy. Hence there is a significant performance drop in the latter two cases. We summarize our results in Table 4.1.

CIFAR 10 Dataset	Pretrained CNN + Approximator initialized with SVD	Pretrained CNN + Approximator initialized Randomly	Randomly initialized CNN+ Approximator initialized randomly
Base Model	88.9%	88.9%	88.9%
2X	86.2%	83.6%	73.7%
5X	83.9%	73.4%	71.8%
10X	83.7%	72.2%	61.4%
20X	76.4%	65.8%	48.6%

Table 3.1 Results of the above mentioned experiments on the CIFAR 10 dataset. As shown, a pre-trained CNN with the approximated weight matrices being initialized through SVD, provides the best results. We are able to reach an accuracy of 83.7% even while achieving 10x compression with a pretrained CNN and approximator initialized with Singular Value Decomposition.

The reason why we were unable to recover the overall performance of the base model with our proposed loss function might be due to the fact that the initial training of the CNN to saturation means that the optimizer has already found a sub-optimal local minima in the highly non-convex error surface. Therefore, the addition of these additional terms to the loss function is pulling the solution away from the current local minima to a random space on the error surface rather than redirecting it back to a better local minima or a global minima. This is a possible explanation for why our method did not work.

3.6 Contributions and Future Work

It has already been established that there is an immense need for compressed architectures in deep learning. Therefore, in this work we explore the idea of compressing deep neural networks through low rank approximation and extend the existing work on it. Rather than performing low rank tensor approximations after the training of the network has been completed, we propose to do it during training itself. We draw inspiration from the positive results of trained pruning and quantization and expect that low rank approximation done during training, by making the approximating parameters trainable in itself, would provide significant gains to the existing compression ratios. However, that is not the case as we found out, due to the reasons mentioned in the previous section. We hope that these experiments would provide valuable insights and throw more insight for further research to be carried out in this direction.

Chapter 4

On Discovering Latent Hierarchies in Classes

4.1 Introduction

Deep neural networks for image classification typically consist of a *feature extractor* network with alternating convolutional and pooling layers [45, 26, 74]. The feature extractor is followed by a *classifier network*, which maps the feature vectors to the class probabilities. This layer is typically a multi layered perceptron (multiple fully connected layers separated by a non linear activation function). The fully connected architecture is preferred since the predicted and the ground truth labels are represented as one hot vectors. The one hot representation assumes that the classes are independent of each other. However the classes typically are not completely independent. They have visual similarities and often form a hierarchy. For example the CIFAR 100 dataset has super classes, each of which consists of 20 subclasses. The Imagenet dataset is organized according to the WordNet hierarchy [60]. Even in datasets where a hierarchy is not mentioned explicitly, the classes might have some latent hierarchical structure. Learning this latent hierarchy explicitly, via a deep learning architecture could improve the performance of the model as it increases the ease of classification. Moreover the fully connected classifier is known to have highly redundant parameters. Alternate well designed architectures for the classifier could improve the parameter size also.

Main Contributions. In this work, i.) we propose to discover multi-level latent hierarchies in classes for improving image classification performance. We encode the latent hierarchy as a binary tree and map each leaf node to a binary string. ii.) We design a deep learning architecture which consists of Latent Hierarchy Classifier (which replaces the Fully Connected classifier), Class2Str and Str2Class networks for discovering the hierarchy represented, as a string embedding (see Section 4.3). iii.) We identify the constraints that needs to be satisfied by the model to achieve high accuracy and propose a structured loss function for learning the latent hierarchy (see Section 4.4). iv.) We benchmark the performance of the proposed architecture and training methods, against some of the best performing architectures in popular image classification datasets and show comparable accuracies (see Section 4.6). Compared to HD-CNN [84] which also discovers a 2 level hierarchy, our method gives better accuracies using only a one third the number of parameters (See Table 4.6). v.) We achieve a reduction in the number of

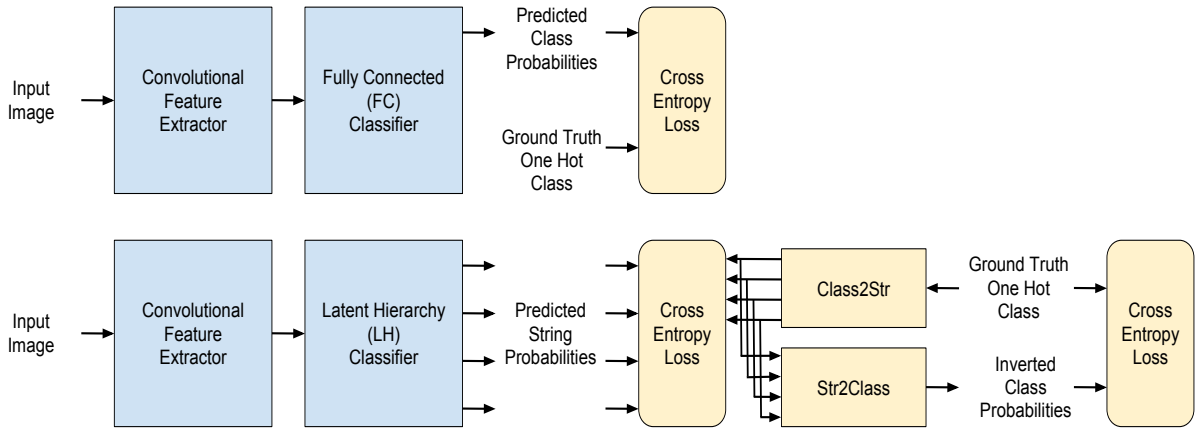


Figure 4.1 The top half of the image shows a traditional classification architecture. The bottom half of the image shows the proposed architecture. The yellow blocks denote those parts which are used only during training phase. During testing these can be removed and therefore the parameter contributions from this part of the network can also be ignored.

parameters of the classification layer by $> 96\%$ for CIFAR 10, CIFAR 100, 76% for Imagenet Rand200 and 41% for the Imagenet 1K dataset. Compared to the method of Deep Compression by Han et al., we get a parameter reduction of 97% as compared to 92% obtained by them in the classifier for the Lenet architecture on the MNIST dataset. vi.) We verify that the string embedding learnt corresponds to some latent hierarchy, by running the same experiments using a hard-coded random string embedding where we observe a decrease in accuracy (see Section 4.6).

We therefore conclude that our proposed model is fit to be used for image classification tasks in memory constrained environments without compromising on performance.

4.2 Related Works

In this section, we survey the related works.

Latent Hierarchy Learning. Though there is a vast literature in using class hierarchical structures in image classification (see [80] for a survey), it is not very well studied using deep neural network based models. An early attempt at learning latent hierarchies was done by Srivastava et al. [72]. Their main goal is transferring knowledge between classes to improve the results with insufficient training examples. Deng et al. [12] uses prior knowledge about relations and hierarchies among classes to improve the classification accuracy.

Yan et al. [84] proposed a deep neural network model called HD-CNN to learn a latent hierarchy. They first train a fine grained classifier and then use the confusion matrix to identify some coarse grained categories. A separate fine grained classifier is trained for each of these coarse grained categories to improve the accuracy. Thus, HD-CNN discovers a 2 level hierarchy with a 2 stage training process, while we discover a multilevel hierarchy using a single stage end to end training. The comparison with

HD-CNN is further enlisted in Table 4.6. The Network of Experts [1] model experiments with different strategies to group fine categories into coarse ones. However both these results have worked with a 2 level hierarchy (the coarse and the fine categories). Our proposed method focuses on mutli-level hierarchies.

Compression. Recently there has been lot of interest in compressing deep neural networks, due to its enormous practical value. In the work of Han et al. [23], a base model is trained to convergence to get a high accuracy classifier. Then a pruning/quantization procedure is applied along with alternate retraining steps, to reduce the number and bit complexity of the parameters, while maintaining the accuracy. Dean et al. [29] proposes a different approach to compression where soft outputs from a high parameter complexity model is used to train a low parameter complexity model. Romero et al. [59] takes this method further by using intermediate outputs of the high complexity model for training the low complexity one.

To our knowledge, this is the first work to focus on reducing the classifier parameter complexity, by learning a latent hierarchy. Our methods are partly orthogonal to [23]. Han et al. [23] applies their method to the full network, while we are focused on the classifier. For the MNIST datasets, we show a reduction in parameters of the classifier by 97% compared to the 92% obtained by Han et al. It is to be noted that pruned (sparse) matrices, requires specialized sparse matrix multiplication algorithms and storage formats to have the claimed memory and runtime improvements. Also, the non uniform quantization used in [23] have a runtime cost of table look up in traditional hardware. In a related work [21], they propose a customized hardware to avoid this runtime cost. Our methods involve replacing the architecture of the classifier, by a low parameter complexity LH Classifier. This can be implemented in all deep learning frameworks as well as existing hardware. In this sense, it is more similar to the works of Iandola et al. and Szegedy et al. [34] [74] where architectural blocks (called Fire Modules, Depthwise separable convolutions respectively) with low parameter complexity while having high generalization accuracy is designed. Some of the newer image classification architectures use the global average pooling layer [51] instead of the fully connected classifier. However this method requires that the number of channels in the final convolutional output is equal to the number of classes. In our proposed method, the parameter complexity of the model can be set completely independent of the number of classes (see Section 4.6).

Our proposed method also has the following differences with Squeezenet [34]. SqueezeNet performs parameter reduction in the feature extractor by reducing the filter sizes of convolutions (5x5 to 3x3 and 3x3 to 1x1) while our proposed method achieves a parameter reduction in the classifier by replacing the fully connected layer with a novel architecture. Furthermore, Squeezenet uses Alexnet as a base model for compression whereas we use VGG 16 as a base model, and therefore achieve higher accuracies.

Embeddings. Our methods resembles some of the embedding techniques that have been proposed. Target coding [85] suggests to replace the 1-hot representation of the labels by an error correcting code (which has a minimum distance property. i.e. any pair of code words have Hamming distance at least d where d is the minimum distance) like Hadamard code. The mapping between the labels and the code

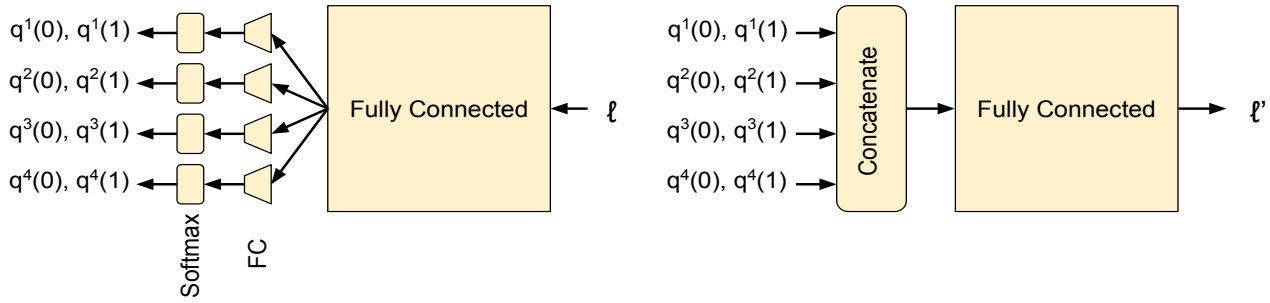


Figure 4.2 The architectures of the proposed Class2Str (left) and Str2Class (right) networks. Class2Str converts the label space representation into a latent binary string representation while Str2Class does vice-versa.

words are fixed statically. However in our case, we are mapping labels to strings, which does not satisfy any non trivial minimum distance property ($d = 1$ is trivial, since any two strings differ in at least one position). Moreover the mapping between labels to strings is learnt in an end to end fashion, unlike in their case. Our method is more inspired by Huffmans coding which naturally gives a hierarchy rather than error correcting codes which has a minimum distance property.

DeViSe ([15] and the follow up works related to semantic embedding and zero shot learning) suggests to replace the 1-hot vector representation of the labels by a word embedding which is learnt separately to model language (using text data). Their main goal is to use the word embedding to get semantic meaning of the labels. Hence the model is able to assign labels to images even if the dataset is small (zero-shot learning). Our goal is to learn a latent hierarchy among the classes that aids in better classification accuracy using a model with lesser parameters. Latent hierarchy learning is ensured using a structured loss function, while they use similarity loss (for eg. cosine) between the label embeddings and image features. Our embedding to binary strings requires considerably less memory than their word embedding to real vectors (to be saved as a table). Also, they need to use a nearest neighbor search for finding the label, while we can do it by a simple binary tree traversal.

4.3 Approach

We propose an alternate architecture and training method for the classifier network, which can discover a latent hierarchy of the classes of arbitrary depth (see Figure 4.1). We take advantage of the fact that hierarchies can be represented by a binary tree with the leaf nodes being the classes (see Figure 4.8). Any path in the tree from the root to a leaf can be represented as a unique binary string. Hence a binary tree defines a one to one mapping from classes to binary strings. Conversely, any one to one mapping from classes to binary strings could also be converted into a hierarchy, by considering the prefix tree of the strings (reminiscent of Huffman's codes from coding theory). Hence learning a hierarchy is equivalent to learning a one to one mapping from classes to strings (See Figure 4.8). The proposed architecture consists of the following networks:

Class2Str. The Class2Str network takes the one hot encoding of the ground truth class labels (denoted by ℓ) and produces a probability distribution of strings $\in \{0,1\}^L$. The probability distribution is given by $2L$ variables, $\left(q^1(0), q^1(1)\right), \cdots, \left(q^L(0), q^L(1)\right)$ where $q^i(b)$ denotes the probability of the ith bit to be b. We obtain the string encoding by taking the maximum probability value in every bit. This is defined by the function:

$$s(q) = a_1, a_2, \cdots a_L \text{ where } a_i = \text{argmax}_{b\in\{0,1\}}\ q^i(b). \tag{4.1}$$

We experimented with a fully connected layer for this network. The probabilities for each bit in the string ($q^i(0), q^i(1)$) by using a sequence of fully connected layers which maps the latent space to 2 dimensions and applying a softmax (see Figure 4.2). During testing phase, we can replace this network by a look up table with entries corresponding to the string encoding for each class (see Section 4.5, for the method of operation during testing phase). This serves as the ground truth hierarchy for our method. The end to end training with the structured loss ensures that the embedding represents an optimal ground truth.

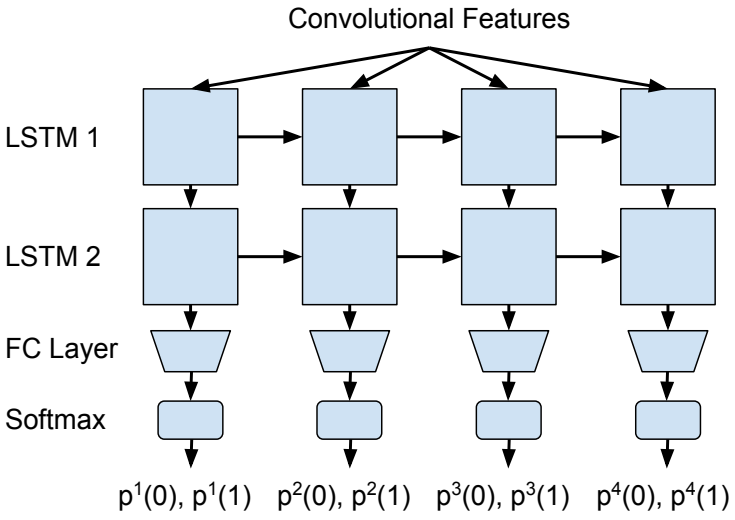


Figure 4.3 The architecture of the proposed LH Classifier which takes as input the convolutional features. This Classifier is plugged into the baseline network after removing the Fully Connected Layers.

Latent Hierarchical (LH) Classifier. The fully connected classifier in the traditional architectures, is designed to predict class probability scores. The Class2Str network replaces the classes with strings. Hence we need the classifier network to predict a probability distribution over strings given by $2L$ variables. We will denote the probability distribution which is the output of the Class2Str network by $\left(p^1(0), p^1(1)\right), \cdots, \left(p^L(0), p^L(1)\right)$, where $p^i(b)$ denotes the probability that the ith bit is $b \in \{0,1\}$. Each bit of the predicted string, is supposed to be a binary classifier which given the previous bits, tries to best divide the data in the subclasses. With different prefixes, the classifier should be able classify according to a different criterion. This dependence on the previously predicted bits is essential to forming a hierarchy. Recurrent Neural Networks are best known to model these kind of dependencies.

Hence we experiment with a multi layered unidirectional LSTM network [30]. The input is fed into a recurrent neural network repeated for L time steps, where L is the length of the string. We use single or double layered LSTM networks for the LH Classifier. The output of the LSTM network is passed through a fully connected layer which reduces the dimension to 2. Applying the softmax function, we obtain the probability scores for each bit of the string (see Figure 4.3).

An important property of the LH Classifier which is useful in parameter reduction is that, its parameter complexity can be set independent of the number of classes. The LSTM is composed of 4 gates. Hence the number of parameters for a single layered network is roughly $f \times h + 4 \times h \times h$ where f is the feature vector size and h is the hidden state size of the LSTM. Note that this is completely independent of the number of classes. Even a single layered fully connected classifier has parameter complexity of $C \times f$, where C is the number of classes. By setting h to be much less than C, we can have have a much smaller LH classifier. Also adding multi-layered LSTMs incurs only little cost in term of number of parameters, if h (the hidden size) is small. The number of parameters in a second layer of LSTM with hidden size h' is roughly $4 \times h' \times h'$ which is much less than $f \times h + 4 \times h \times h$, since f is much bigger than h and h' typically.

Str2Class. Apart from these two, we have a third network called Str2Class, which is essential for making sure that the string encoding learned by Class2Str is one to one. The Str2Class network tries to decode the string produced by Class2Str to class probabilities (denoted by ℓ') which is close to ℓ (the one hot vectors of the ground truth labels). For Str2Class, we experiment with a fully connected network similar to the Class2Str network. We first concatenate the 2 dimensional probability vectors for each bit in to a $2L$ dimensional vector and pass it through a fully connected layer (see Figure 4.2). This network can also be removed during the testing phase (see Section 4.5, for the method of operation during testing phase).

Dataset	Convolutional Feature Extractor	Fully Connected Classifier	RNN Classifier	Class2Str	Str2Class
MNIST	16 - pool - 32 - pool - 64	3136-500-10	3136 - 10 - 10	10 - 500	500 - 10
CIFAR 10	64 - 64 - pool - 128 - 128 - pool - 256 - 256 - 256 - pool - 512 - 512 - 512 - pool - 512 - 512 - 512 - pool	512 - 1024 - 1024 - 10	512 - 20 - 20	10 - 500	500 - 10
CIFAR 100	64 - 64 - pool - 128 - 128 - pool - 256 - 256 - 256 - pool - 512 - 512 - 512 - pool - 512 - 512 - 512 - pool	512 - 1024 - 1024 - 100	512 - 40 - 40	100 - 1000	1000 - 100
Imagenet Rand200	64 - 64 - pool - 128 - 128 - pool - 256 - 256 - 256 - pool - 512 - 512 - 512 - pool - 512 - 512 - 512 - pool	25088 - 4096 - 4096 - 200	25088 - 85 - 85	200 - 1000	1000 - 200
Imagenet 1K	64 - 64 - pool - 128 - 128 - pool - 256 - 256 - 256 - pool - 512 - 512 - 512 - pool - 512 - 512 - 512 - pool	25088 - 4096 - 4096 - 1000	25088 - 2100 - 100	1000 - 2000	2000 - 1000

Table 4.1 The architectures for each of the networks used in the experiments are summarized here. The sequence represents number of channels in each layer, the number of hidden units, the hidden state size of LSTM for the feature extractor, FC Classifier and LH Classifier respectively. This pattern is followed for the feature extractor, classifiers and other networks alike.

4.4 Training with Structured Loss

Training the proposed architecture present some difficulties. One of the main problem is of ensuring that the *Class2Str* encoding is one to one (ie. distinct classes are encoded as distinct strings). This is a discrete constraint and we need to design a continuous loss function that is minimized when this constraint is satisfied. Our solution consists of the *Str2Class* network which inverts the Class2Str mapping. We design a continuous loss function which using the outputs of the Str2Class function, ensures that the Class2Str function is one to one on convergence. Furthermore, we use a structured loss function (Equation 4.4), so that the string embedding gives rise to a latent hierarchy.

The loss function is defined by first identifying the constraints that need to be satisfied to get a high accuracy image classifier. The constraints that need to be satisfied are the following:

1. The Class2Str encoding must map distinct ℓ (classes) to distinct $s(q)$'s. Without this, we cannot obtain a unique class label from the predicted string. This is satisfied if Str2Class is able to invert the Class2Str mapping. That is ℓ must be equal to ℓ' (notation from the previous section).

2. The string encoding given by the Class2Str network much match the string predicted by the LH classifier. That is $s(p) = s(q)$ where s is defined in Equation (4.1).

The first constraint can be modeled by a cross entropy term $H(\ell, \ell')$. For the second constraint, we first strengthen it, to say that the distribution p, q are equal. Hence this can also be modeled by a cross entropy term $H(p, q) = \sum_{i=1}^{L} H(p^i, q^i)$. So a possible loss function is the following:

$$\alpha H(\ell, \ell') + \beta \sum_{i=1}^{L} H(p^i, q^i) \tag{4.2}$$

where H denotes the cross entropy between distributions.

The above loss function is straightforward, but we encounter two problems when we train the network.

1. On back-propagating the above loss function, $q^i(0), q^i(1)$ tends to converge towards the uniform distribution. When the q^i's are close to unbiased, the string encoding function s is not robust i.e a small change in the value of q^i changes the string encoding. This results in low generalization accuracy. Hence we add an additional constraint that q^i's should be biased towards 0 or 1. We ensure this by using the following regularizer which is maximized when these distributions are fully biased:
$$\sum_{i=1}^{L} \left(q^i(0)^2 + q^i(1)^2 \right). \tag{4.3}$$

2. Equation (4.2) gives equal weight for each position of p^i, q^i. This implies that an error in the first as well as last position is equally penalized. However making an error at the top level of a hierarchy is in some sense greater than a error at the bottom level. An error at the first position,

Dataset	% Acc of FC Classifier	% Acc of LH Classifier	% Acc of reduced FC Classifier	#parameters in FC Classifier	#parameters in LH Classifier	Reduction in parameters	Reduction in test time per image
MNIST	99.38	99.36	98.45	1.61 M	31 K	98%	13.9%
CIFAR 10	90.43	90.51	88.40	1.58 M	6 K	99%	15.9%
	% Acc in Maxout: 90.65, Network in Network: 91.2, Deeply Supervised Networks : 91.78						
CIFAR 100	64.65	64.67	57.90	1.58 M	30 K	98%	14.8%
	% Acc in Maxout: 61.43, Network in Network: 64.32, Deeply Supervised Networks : 65.43						
Imagenet Rand200	76.47	75.93	70.14	120.35 M	29.28 M	76%	7.4%
Imagenet 1K	70.51	70.11	67.88	123.63 M	72.42 M	41%	5.5%

Table 4.2 The accuracy and parameter reductions achieved by the proposed method on the respective datasets. Also accuracies of some of the best performing models (Maxout [18], Network in Network [51], Deeply Supervised Networks [47]) on the given datasets are shown. The third column shows the accuracy of a model where an FC classifier with the same number of parameters as the LH Classifier is used.

results in classification of the input to a class which is highly dissimilar to the ground truth. However strings having long common prefix, correspond to similar classes and the cost of error must be less. Hence we use the following loss term instead, where $\mu \in (0, 1)$ is hyperparameter, to achieve better accuracies. μ^i is evidently the most important hyperparameter which helps in learning of the correct hierarchy since this ensures that misclassification at the initial bits of string incur a larger penalty in the eventual loss. Hence this factor serves as a decay constant for the sequential penalty incurred in the subsequent bits of the predicted string. After careful tuning of this sensitive hyperparameter, we found that a decay factor of 0.8 for μ gives the best results.

$$\sum_{i=1}^{L} \mu^i H(p^i, q^i) \tag{4.4}$$

We also use a L^2 regularizer for all the weight in the network. Hence the final loss function that we minimize is the sum of the losses for each constraint, given bellow:

$$\alpha H(\ell, \ell') + \beta \sum_{i=1}^{L} \mu^i H(p^i, q^i) - \gamma \sum_{i=1}^{L} \left(q^i(0)^2 + q^i(1)^2\right) + \delta L^2(W) \tag{4.5}$$

where H denote the cross entropy function, $L^2(W)$ denotes the sum of squares of all the weights and $\alpha, \beta, \gamma, \delta$ are hyper parameters. Note that the term corresponding to Equation (4.3) is negative, since it needs to be maximized.

4.5 Dataset and Experiments

Our experiments were performed on MNIST, CIFAR10, CIFAR100 [43], Imagenet Rand200 [11] and Imagenet 1K [61] datasets (ordered according to gradual increasing complexity). MNIST consists

Network	Dataset	Error Rate	# Parameters
HD-CNN	CIFAR-100	32.62	∼15M
LHC (Ours)	CIFAR-100	35.33	**14.7M**
HD-CNN	Imagenet 1K	31.34	∼220M
LHC (Ours)	Imagenet 1K	**29.89**	**87.2M**

Table 4.3 Comparison on parameter reduction and error rates with HD-CNN on the CIFAR-100 and Imagenet 1K dataset. Our method does better than HD-CNN in terms of number of parameters on CIFAR-100 and both in terms of error rate as well as number of parameters on Imagenet 1K.

of single channel, (28×28) images with 10 classes. CIFAR10 images has 3 channels with slightly bigger (32×32) sizes, and 10 classes. As a next step we moved to CIFAR100, which has 100 classes but with the same image size. we followed this up by experimenting on Imagenet Rand200 which is a subset with images of 200 classes randomly sampled from images of 1000 classes in Imagenet [11]. For this dataset, We used color images of size 224×224 randomly cropped from the resized images. Finally, we ran experiments on Imagenet 1K [61] which contains 1.28 million images from 1000 classes.

We apply our technique to the VGG16 architecture on Imagenet. For CIFAR, since the image sizes are smaller, we use a VGG16 variant and for LeNet architecture for MNIST. The exact details of the architectures for the experiments are provided in Table 4.1.

Training Phase. We train a traditional network, for the task of image classification, that consists of convolutional and fully connected layers (denoted as the Base Model) to convergence. Then, we replace the FC classifier with the proposed LH classifier, without altering the weights of the convolutional feature extractor. The LH classifier network is then trained along with the Class2Str and Str2Class networks by back-propagating the gradients calculated from the loss function (details mentioned in Section 4.4). The convolutional weights are not updated with the gradients during the backward pass. We use the Adam algorithm for the optimization. Hyperparameters $\alpha, \beta, \gamma, \delta$ are initially chosen so that each term in the loss function has the same order of magnitude. Furthermore γ is reduced to as small a value as possible in order for the probabilities q to remain highly biased.

Testing Phase. During the testing phase the Class2Str and Str2Class networks are removed. Given an image, the LH classifier predicts a string $s(p)$. We maintain a look up table containing the mappings of strings corresponding to each of the classes, learnt by the Class2Str network. The predicted class for the image is the the one corresponding to the string $s(p)$, in the look up table. The predicted strings for the test images are compared with the learnt strings for each class by the Class2Str network. If all bits match, we count it as a correct prediction.

4.6 Results and Discussion

We describe our results about the proposed model, first by comparing the accuracies achieved, then in terms of the parameters of the classifier used and finally analyze the latent hierarchy discovered by the model.

Network	Acc	#Params Classifier	Compression Rate
Lenet-300-100	98.36 %	266 K	0%
DC Pruned (ip)	98.42%	22 K	92%
LH Classifier	98.51%	8 K	97%

Table 4.4 Comparison of parameter reduction with Lenet-300-100 (top) and Lenet-5 (bottom) on the MNIST dataset, with the results reported by Deep Compression (DC) method of Han et al. [23]

Accuracy. We report the accuracy of the base model with the FC Classifier as well as of the proposed model with the LH Classifer. Our base models (see Table 4.1) have accuracies comparable to the best performing ones in the corresponding datasets. We can recover the classification accuracy of the base model in each case (see Table 4.2). In Table 4.6 we demonstrate how our method compares with HD-CNN [84] which also discovers a 2 level latent hierarchy, on CIFAR-100 and Imagenet 1K datasets. The proposed LHC is able to outperform HD-CNN in terms of the number of parameters for both the mentioned datasets while also having a lower error rate for Imagenet 1K.

Compression. Our proposed method has advantages in terms of compression. We replace a high complexity Fully Connected (FC) classifier by a low complexity LH Classifier that is trained according the proposed method. The proposed architecture has the scope of reducing parameters of the classifier due to the following reasons: 1.) By discovering the latent hierarchy among the classes explicitly in the architecture, should allow even a small complexity model to give good accuracy. 2.) Since the Class2Str network converges to a one to one mapping, it can be replaced by a look up table during testing phase. The look up table maps each string to the unique class that has been learnt. During the testing phase, the class predictions are given by first getting the string predicted by the LH classifier network, followed by a binary tree traversal. This adds negligible memory and running time to the system. 3.) Even if the base model has only a single fully connected layer for the classifier, it will still have parameter size = number of classes $\times$ feature size. This can easily become expensive when either the number of classes or the convolutional feature size is large. However the LH classifier network is designed such that its parameters can be set independent of the number of classes. Moreover it can be seen from the table that, had we used the same number of parameters in a FC Classifier as those used by LH Classifier, we would have got significantly inferior performance thereby proving LH Classifier to be a beneficial alternative.

Replacing the FC Classifier with the LH classifier reduces parameters. The parameter reduction is around 96% (see Table 4.2), for CIFAR100 and around 76% for Imagenet Rand200. For Imagenet 1K, we get reduction of 41%. We also compare our results on MNIST with that of Han et al. [23]. Han et al. [23] achieves parameter reduction by way of pruning the weights. Our method gives a reduction of 97% as compared to 92% in their work, on the classifier. In Figure 4.4, we show how the accuracy of the model varies with different hidden state sizes of the LSTM used by the LH Classifier. As can be seen from the Figure 4.4, decreasing the hidden state size to 1000, still gives a accuracy $> 60\%$ for VGG16 trained on Imagenet.

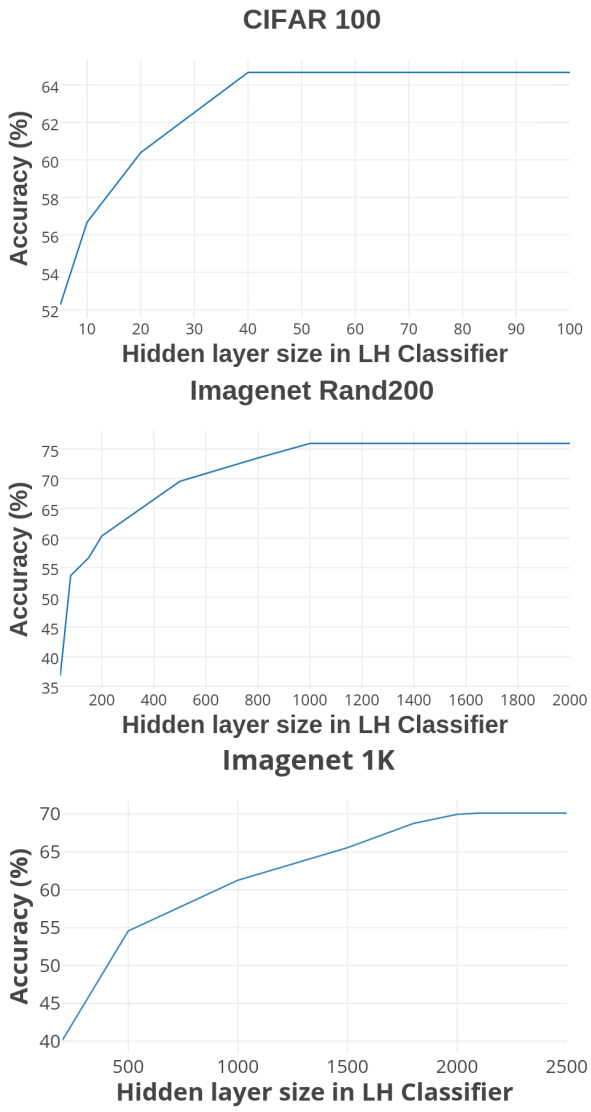


Figure 4.4 Plots for the change in accuracy with varying hidden layer sizes of the LSTM.

Latent Hierarchy. Next we examine if the string embedding learnt by the Class2Str network represents a latent hierarchy. We replace the Class2Str network by a random embedding of labels to strings that is one to one. The random embedding is given as a look up table, which gives a string from each label. We train the LH classifier with the structured loss with the targets being the strings from the random embedding corresponding to the ground truth labels. We obtain that the accuracy reduction is 5% for CIFAR10 and 4.3% for CIFAR100.

The hierarchies learnt by our models for MNIST and CIFAR10 is given in Figure 4.8. As can be seen from the figure, visually similar classes seem to be grouped together. For CIFAR100, we observe some examples of visually similar classes getting mapped to strings which have a long common prefix

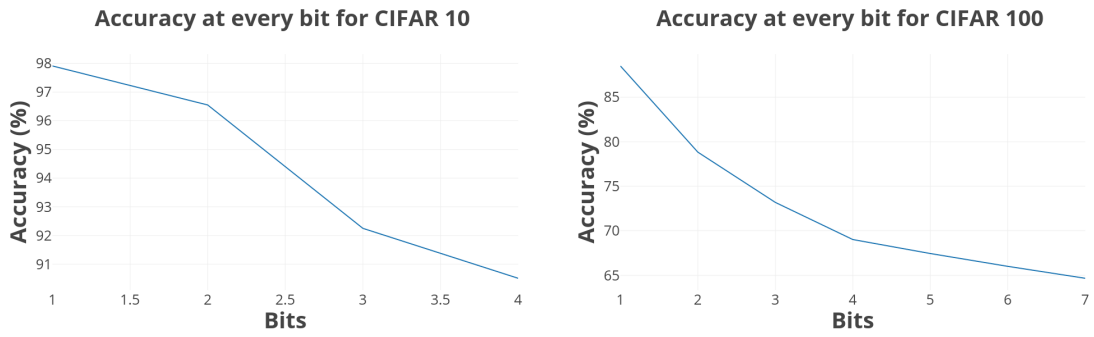


Figure 4.5 Classification Accuracy at every bit of the predicted string as a result of using Structured Loss.

(see Figure 4.6). We also analyze effect of changing the values of string length L on the accuracy (see Figure 4.7). Best accuracy is obtained by choosing $L = \lceil \log_2 C \rceil$ as is evident from the plot.

We use a structured loss (see Section 4.4) during training for backpropagating the error, which assigns more penalty to misclassification at the initial bits and lower penalty to the subsequent bits. This essentially means that the network will learn to classify initial bits of the string more correctly than the later bits, and at test time we will get the best accuracy for classification of the 1st bit of the string. The accuracy reduces as we move ahead for bits 2,3 and so on (see Figure 4.5). This seemingly natural result is verified by us by reporting the accuracy at every bit position of the string for some of the datasets.

Note that the flipping 0's to 1's doesn't change the latent hierarchy. Hence, there are multiple Class2Str mappings corresponding to the same latent hierarchy. The system is stochastic and converges to one of them randomly as of now. This is not necessarily a problem. However, explicit symmetry breaking in the model or training process might improve the results.

4.7 Summary and Contributions

In this work, we have proposed a deep neural network architecture for object recognition, that explicitly learns a latent hierarchy of the classes. We have empirically demonstrated that this architecture is equally good as the traditional architectures in terms of accuracy. Moreover, it achieves these accuracies, with a fraction of the parameter complexity, making it a compelling addition to the known model compression techniques. We believe it is an interesting and open research direction to use prior knowledge of hierarchies in the classes along with the LH Classifier, in to get better accuracy. Furthermore, the added benefit of obtaining a compressed model without any loss in performance on the base model, underlines the usability of our architecture in applications with limited memory bandwidth.

To sum up, our contributions here are not only to design a memory efficient model for object recognition with no drop in performance but also to learn an efficient embedding of the label space in the process, thereby learning a multi-label latent hierarchy. We hope that these contributions will throw

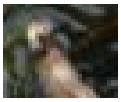
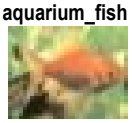
Image Examples (Classes)					Strings	Length of the Longest Common Prefix
shrew 	shrew 	porcupine 	otter 	otter 	0011100 (shrew) , 0011110 (porcupine) , 0011111 (otter)	5
lobster 	aquarium_fish 	flatfish 	crocodile 	ray 	0101010 (lobster) , 0101011 (aquarium fish) , 0101100 (flatfish) , 0101110 (crocodile) , 0101111 (ray)	4
maple_tree 	lawn_mower 	lawn_mower 	tiger 	worm 	1011011 (maple_tree) , 1011101 (lawn_mower), 1011110 (tiger), 1011111 (worm)	4
pine_tree 	pine_tree 	palm_tree 	palm_tree 	table 	0111101 (pine_tree) , 01111110 (palm_tree), 0111111 (table)	5
beaver 	skunk 	rabbit 	rabbit 	wolf 	1111000 (beaver), 1111001 (skunk) , 1111010 (rabbit) , 1111011 (wolf)	4

Figure 4.6 While classes like shrew, porcupine and otter have similar visual features. Images of crocodile had water around them in the images which justifies its proximity with the other fishes, images of tiger and worm had a lot of greenery around it which has resulted in it having a long common prefix with maple tree and lawn mower. Pine tree, palm tree (with brown trunks) and table have brown as a common color while beaver, skunk, rabbit and wolf are not only visually similar but also of similar color.

light to the interesting research direction of learning latent hierarchies inherent among the classes of object recognition challenges.

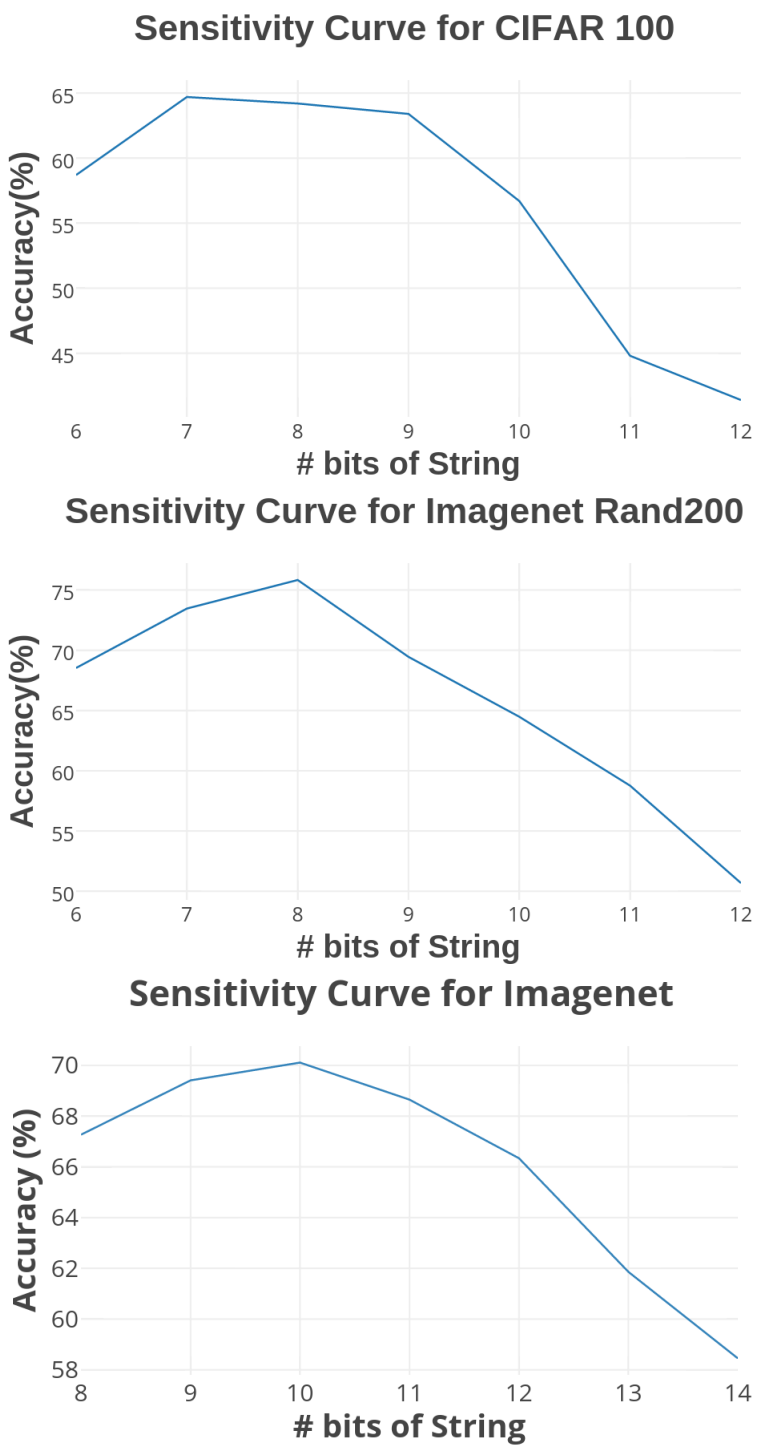


Figure 4.7 Accuracy is plotted vs the the number of bits of string to be used for a latent representation of the label space, after the loss saturates and no further learning takes place. The optimal number of bits can be found from such plots for each dataset.

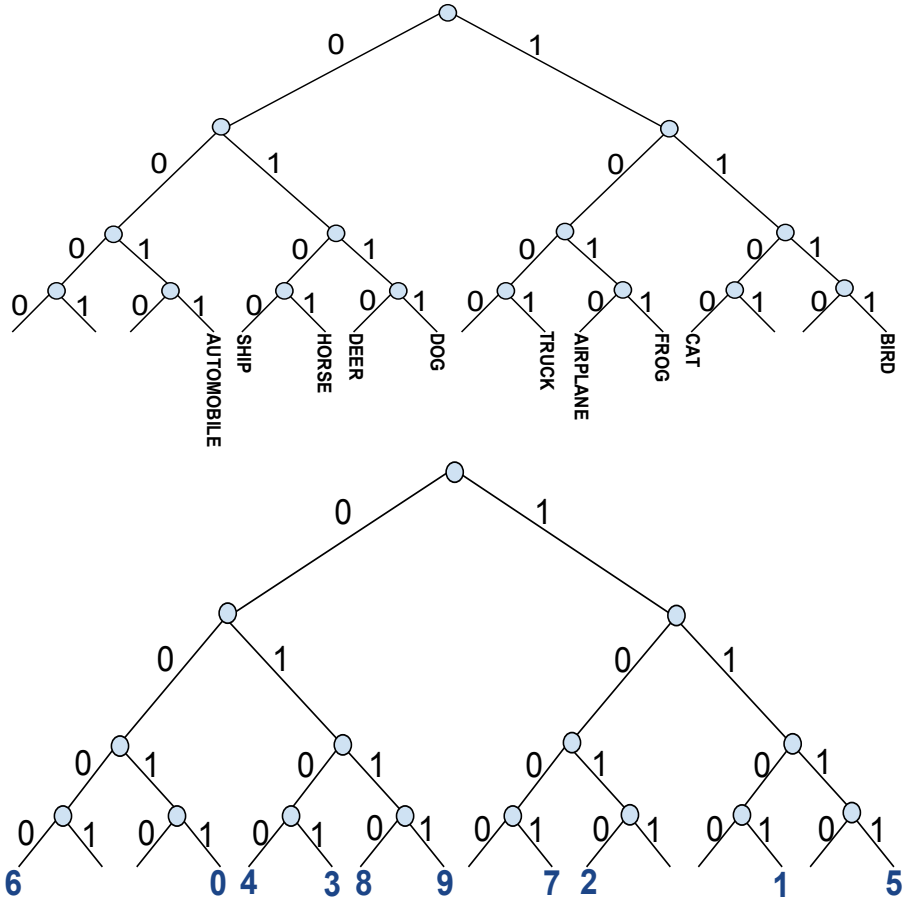


Figure 4.8 Prefix tree learnt for the CIFAR10 (top) and MNIST (bottom) datasets. Some of the visually similar digits in MNIST such as 3,8 and 9 have a common prefix. Hence their close proximity in the tree. Even 7 and 2 are in proximity since they are visually similar.

Chapter 5

Anchor Point based Visual Localization

5.1 Introduction

In chapters 2 and 3, we explored explicit compression of deep neural networks while the previous chapter concentrated more on compressing the classifier in the network. In this chapter, our focus is on deigning an efficient architecture and using it for the task of visual localization of a scene for autonomous devices. Compressing feature extractors has been studied in previous works where architectural modifications have been performed before commencing the training process of the neural network itself, in order to develop a more efficient architectural design. Some of the recent architectural designs which focus on a more efficient feature extractor can enlisted as under:

1. **GoogleNet:** The key contribution to this architecture was the introduction of the inception module. This module has 1x1 convolutions before the 3x3 and 5x5 convolutions. The 1x1 convolutions (or network in network layer) provide a method of dimensionality reduction thereby accounting for compression. The network in network type of convolution is able to extract information about the very fine grain details in the volume, while the 5x5 filter is able to cover a large receptive field of the input, and thus able to extract its information as well. [75]

2. **ResNet:** The major idea introduced through this architecture is the Residual Block. The input x typically goes through conv-relu-conv series resulting in F(x). Now, the input to the next layer H(x) is modified by adding this quantity to the original input as H(x) = F(x) + x. This avoids the problem of vanishing gradients and allows very deep networks to be used. One can also completely remove the fully-connected layers with this architecture. [27]

3. **DenseNet:** This architecture makes use of a growth rate parameter for each layer. The primary difference of this architecture from ResNet is that this has connections from one layer to all its subsequent layers instead of to the next layer only. We further describe this in more detail in the subsequent sections. [33]

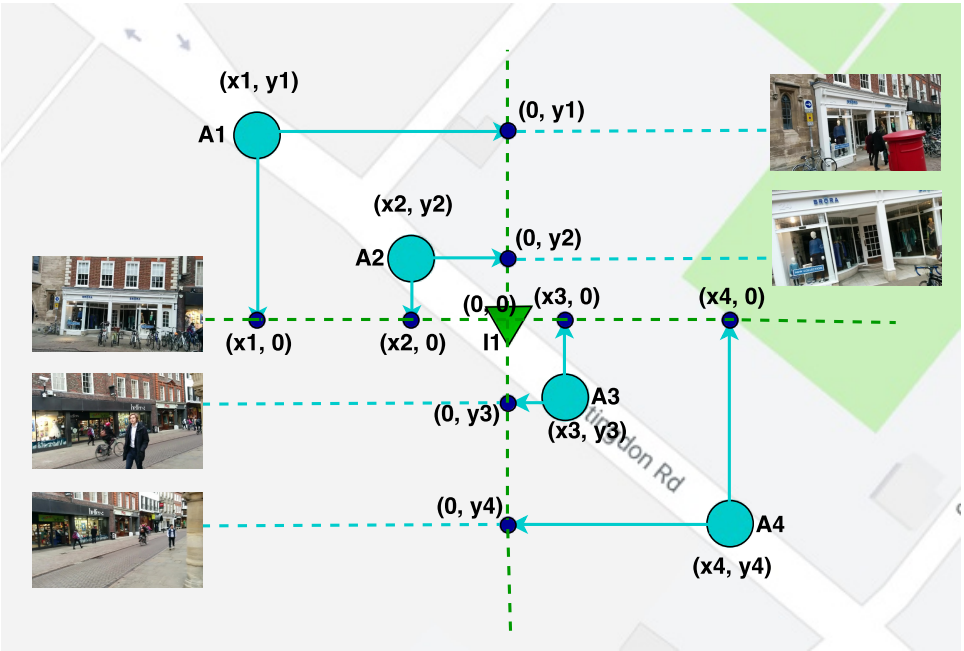


Figure 5.1 An example of anchor point allocation for a sample road. The blue points (A1-A4) denote the anchor points which are predefined uniformly. The green triangle(I1) is the input frame which denotes the current location. The coordinates of the anchor points are shown relative to the coordinate of the current location (denoted as (0,0)). The nearest anchor point A3 or A2, might not be visible properly from the view. Our system can find the most relevant anchor point and the relative offsets from it, giving a more accurate prediction.

The visual relocalization problem is an essential component of many practical systems like autonomous navigation, augmented reality, drone navigation etc. Although GPS sensors could be used for localization in these applications, it is often noisy and will not work in indoor environments. Hence an independent source of location information is essential for safety as well as applicability in a wide variety of environments. Though location information could be accurately estimated using 3D point cloud data, gathering and processing such data can be expensive and slow. The visual relocalization problem is estimating the location as well as camera pose, given just the observed camera frame without using any other sensor data.

Early solutions modeled this problem as an image retrieval problem. However, these solutions required that image features for a large collection of images to be stored. Also, a nearest neighbour search was needed at test time. Thus the memory and runtime increased proportionately with the data set size. The most accurate approaches requires 3D point cloud data of the region [66], which is expensive to gather and process. PoseNet proposed to model this problem directly as a regression problem and used a deep neural network as the image feature extractor [41]. Extensions to this work aim at modeling the uncertainty in estimating the location and pose by using a Bayesian neural network [40]. Furthermore, a spatial LSTM based approach to regress the pose was found to improve performance [81]. A recent approach has been to improve points through a geometric loss function which tries to minimize the reprojection error [40].

We propose a visual relocalization system inspired by how humans determine location. We typically identify some landmarks which we deem as important. The coordinates of these landmarks are well known. Then, we try to estimate our position by determining the offset to our position relative to the landmark. The landmark we choose need not be the nearest one, as it might not be visible, due to the viewing angles. Inspired by this idea, we propose a system which assigns anchor points relative to which the 6DOF (six degrees of freedom of the 3 spatial and 3 pose coordinates) can be predicted accurately. We do this by modeling the problem as a multi task problem, of classifying to an anchor point and then finding the offsets relative to the anchor point. However a direct approach for training such a network requires ground truth of the anchor points present in each image. The datasets typically provide only the ground truths of the 6DOF and not the anchor point present in the image. The anchor point visible in the image need not be the nearest one, because of the pose. Hence we propose a new loss function, which during training automatically finds the appropriate anchor point relative to which the offsets needs to be regressed in an end to end fashion.

We benchmark our model on an outdoor dataset covering a large area (Cambridge landmarks [41] covering few 1000 m^2) as well as an indoor dataset (7 Scenes [67] covering few m^2) suited for small robot navigation. We improve the median error in all the scenes of both the datasets, from the previous best model, PoseNet (with geometric reprojection loss [40]) when using the same GoogleNet [76] feature extractor. We achieve $<$1.5m and $<4^\circ$ in localization performance in 4 out of the 6 outdoor scenes in Cambridge Landmarks. In the specific case of Street scene, our method improves the median error by over 8m. Furthermore, we localize to within $<$0.2m for all the indoor scenes of the 7 Scenes dataset which is a significant improvement over the previous deep learning based approaches. We also experiment with various feature extractors like DenseNet [33] (giving better accuracy) and MobileNet [32] (giving better runtime performance, while maintaining accuracies).

The contributions of this work can therefore be summarized as follows:

1. We design a novel deep neural network architecture for achieving 6-DOF regression for visual localization of a scene.

2. We propose an anchor point based end-to-end training algorithm using the anchor point classifier, relative offset regressor and the absolute offset regressor.

3. We propose a novel and differentiabe loss function for our task and address the problem of discontinuity for the anchor point classifier.

4. We achieve state-of-the-art results for translation localization and comparable results for the camera pose localization on the Cambridge Landmarks dataset.

5.2 Related Work

In this section, we briefly survey the literature. For a detailed survey see [55].

5.2.0.0.1 Visual Place Recognition. The visual place recognition task has been traditionally solved by modeling it as an image retrieval problem [4], [9], [79], [68]. This enables Bag of Words (BOW) and VLAD [36] representations to be used in scalable retrieval approaches. More recently, deep learning models have also been used successfully for creating efficient image representations, which was combined with retrieval methods [57], [17], [5], [78] [6]. However, retrieval based solutions require image features for the entire dataset to be stored. Also, a nearest neighbour search needs to be triggered at test time. Thus, the memory and runtime increased proportionately with the data set size. PlaNet [83] modeled the problem as a classification problem for localization at a world scale. However, estimation of a fine-grained 6-DOF localization requires continuous output and discrete methods have not worked so far for this task.

In contrast, metric localization (visual localization) techniques approach this as a 2D-3D mapping between the 2D coordinate system of the image space to the 3D coordinate system of the world space. This is typically done by matching the representations of the learnt image [49, 8, 65, 73, 50, 66] and adapting a nearest neighbour approach. The full 6-DOF pose of the camera can be estimated quite precisely. These methods however require a large database of features and efficient retrieval methods, which makes them suffer at test time since most retrieval methods have an addition feature matching latency.

5.2.0.0.2 PoseNet. PoseNet [41] addressed this idea by introducing a regression based deep neural network technique to estimate the metric localization parameters. It achieves a combination of strengths of place recognition and localization approaches and can localize without a prior estimate on the pose. This solves the disadvantage of storing the features in memory since they are learnt during training and estimated during testing.

Extensions to this work have been on RGB-D input images for localization [48], learning relative ego-motion of the camera [52], improving the context of features used for localization [75], using video sequence information for localization [39], modeling the uncertainly in localization using Bayesian Neural Networks [39], exploring a number of loss functions for learning camera pose which are based on geometry and scene re-projection error [40].

Although PoseNet and its extensions are robust and scalable, they lag behind some of the traditional methods [66] in estimating the pose . In this work, we propose a novel discrete anchor point assignment based regression technique which provides significant improvement over these methods while maintaining the scalability and robustness. We keep intact the advantages of PoseNet and its variants as well as improve over the nearest neighbour based methods where feature vectors and required to be stored in memory entirely.

5.3 Methodology

The visual re-localization problem involves inferring the 3D location coordinates and the camera pose (given by 3 angles) from just the image. In the previous works [39, 40, 41], a deep neural network

based approach was proposed which directly predicts the 6-DOF by regressing against the ground truth, relative to a global coordinate system. Here, we propose a system which assigns anchor points relative to which the 6-DOF can be predicted accurately. We do this by modeling the problem as a multi-task problem. The first task is of classifying the input camera image to one of the anchor points and the second task is of finding the offsets relative to that anchor point. We use standard CNN feature extractors for image feature extraction. The feature extractors are typically followed by a Global Average Pooling (GAP) layer in order to get a single vector instead of a feature map. This is illustrated in Figure 5.3. Our proposed method then branches this output into 3 heads.

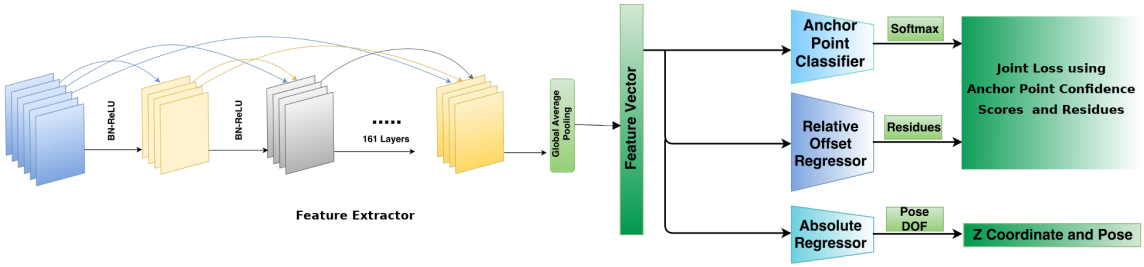


Figure 5.2 Block diagram of the proposed architecture. We experiment with different CNN feature extractors including GoogleNet, DenseNet and MobileNet. The CNN feature extractor is followed by a global average pooling layer to get the final feature vector. The feature vector is fed to an anchor point classifier, relative offset regressor and an absolute regressor for pose.

5.3.0.0.1 Anchor Point Classification. Given a route map where visual relocalization is desired, we subdivide the route into equally spaced intervals. The end points of the intervals are designated as anchor points for the 2D spacial coordinates. Note that the global coordinates of these anchor points are known, since they are fixed apriori. We then model the problem of finding the most relevant anchor point as an image classification problem. Given an image, the predicted class should be the most prominent anchor point in the image. Note that the most prominent anchor point need not always be the one nearest to the location of the image. Hence we frame our problem in such a way that we need not have the ground truth information about the prominent anchor point directly as part of the training data set. We treat the predicted probabilities as a confidence score for each of the anchor points. We combine this information in our loss with the relative offsets. Furthermore, we experiment by considering the nearest anchor point to the image as the ground truth and incorporate the corresponding cross entropy loss in our overall loss function. We report results for both cases.

The Anchor Point Classifier head is obtained mapping the global average pooling output of the feature extractor to the number of predefined classes using a Fully Connected (FC) layer. For our case, the number of classes is equal to the number of anchor points defined during preprocessing. The output of the FC layer is then subjected to a softmax classifier to get a probability mapping. However, instead of using these values as probability predictions, we use them as confidence scores in our experiments as will be explained in more detail in Section 5.3.1.

5.3.0.0.2 Relative Offset Regression. Along with anchor point classification, our system also produces the relative offsets from the anchor point. We train these offsets through the regression loss.

However, we face the same problem of not having the ground truth for the most relevant anchor point. Hence our system produces the relative offsets with respect to all the anchor points and our model tries to regress these relative offsets. Note that since the global coordinates of all the anchor points are known, the relative offset ground truth information for all the images can also be calculated from the global ground truth information. The output of the GAP layer of the feature extractor is passed through a fully connected neural network layer to obtain the regressing head whose task is to predict the X,Y Coordinate offset values relative to each anchor point defined previously. Thus if there were N anchor points, say, the output of this head would be a $2N$ dimensional vector.

5.3.0.0.3 Absolute Offset Regression for Z and Pose. Our approach is motivated from human scene recognition through relative identification. Hence, having the anchor points can be efficient only for the X and Y coordinates and not for the Z and camera angle coordinates. Predicting relative offsets for Z coordinate and pose is not carried out due to the following reasons:

1. The Z coordinate and the camera pose of a scene is independent of the previous scene. It is thus counter intuitive and does not capture the continuity of the scene like the X,Y coordinates do.

2. Regressing relative offset for the Z coordinate as well as the pose complicates the regression task further thereby leading to inferior results as we found out in our experiments.

Hence, our model separately predicts the the remaining 4-DOF in accordance with the global coordinate system. The third and final head is assigned the task of predicting the absolute values of the Z coordinate and the remaining DOF values for determining the camera pose. Therefore, the GAP layer of the feature extractor is mapped to a 5 dimensional vector in this case. This head also consists of a fully connected layer for regressing the absolute values of the remaining DOF (i.e Z coordinate and pose).

5.3.1 Loss Function

We train our model with a loss function which is the sum of 3 components. Let us denote the output of the anchor point classification head to be $\hat{C}$, which gives confidence scores for each of the N anchor points, for the current input image. X, Y denotes the vectors of dimension N of ground truth offsets of the frame in the horizontal plane, with respect to each of the N anchor points and Z the vertical distance. The ground truths X, Y, though not provided with the dataset, can easily be computed as a preprocessing step, by changing the origin of the coordinate system to the respective anchor points. P denote the three angles of the camera. Let $\hat{X}, \hat{Y}, \hat{Z}, \hat{P}$, be the predictions of the model.

Since we do not have ground truths for the most relevant anchor point in the image, we cannot train the anchor point classification head with a cross entropy loss. However, we formulate a joint loss function that uses $\hat{C}$ and the offsets $\hat{X}, \hat{Y}$ from each of anchor points. We weight the squared loss of offsets with confidence scores of the respective anchor points. The resultant loss function therefore looks like the following:

$$\sum_i \left[\left(X_i - \hat{X}_i \right)^2 + \left(Y_i - \hat{Y}_i \right)^2 \right] \hat{C}_i \tag{5.1}$$

This is motivated by the fact that the accuracy is determined by the offset of the most relevant anchor point. If an anchor point is completely irrelevant ($\hat{C}_i = 0$), then we do not need the corresponding offset predictions.

Now, we include the predictions of the absolute regressor which is responsible for predicting the Z coordinate values and the remaining angular DOF for determining the pose. In order to do this, add the following component to the loss function:

$$\sum_i \left[\left(Z_i - \hat{Z}_i\right)\right]^2 + \left\| P_i - \frac{\hat{P}_i}{\left\|\hat{P}_i\right\|} \right\|^2 \tag{5.2}$$

Additionally, we also experiment with adding a cross entropy loss with the ground truth being nearest anchor point.

$$\text{Confidence loss} = H\left(C_i, \hat{C}_i\right) \tag{5.3}$$

All the 3 components of our loss function mentioned in equations 5.1, 5.2, 5.3 are then assigned weights (hyperparameter) and the summed up to get the overall loss for our task. Therefore, we now have the following resultant loss function:

$$\alpha_1 H(C_i, \hat{C}_i) + \alpha_2 \sum_i \left[\left(X_i - \hat{X}_i\right)^2 + \left(Y_i - \hat{Y}_i\right)^2\right] C_i + \alpha_3 \sum_i \left[\left(Z_i - \hat{Z}_i\right)\right]^2 + \left\| P_i - \frac{\hat{P}_i}{\left\|\hat{P}_i\right\|} \right\|^2 \tag{5.4}$$

where $\alpha_1, \alpha_2, \alpha_3$ are weights assigned to the separate components of the loss function.

5.4 Dataset and Experiments

We benchmark our method on an outdoor and as well as indoor localization dataset. A summary of the datasets can be found in Table 5.1. We rescale all the images to 224×224 since our convolutional feature extractors are trained on those dimensions.

As described in Section 5.3, we initialize our feature extractor (DenseNet architecture) pretrained on the Imagenet 1K dataset trained on the image classification task. This gives better performance for outdoor scenes than using a network pre-trained on the Places dataset as shown in earlier work by Kendall et.al [41].

Table 5.1 Summary of localization datasets used for benchmarking.

Dataset	Type	Scale	Imagery	Scenes	Train Images	Test Images	Spatial Area
Cambridge Landmarks [41]	Outdoor	Street	Mobile phone camera	6	8,380	4,841	100×500 m
7 Scenes [67]	Indoor	Room	RGB-D sensor (Kinect)	7	26,000	17,000	4×3 m

As a pre-processing step, we first divide the scene space into several anchor points by uniformly selecting every n-th frame from among all the frames in a scene video. The distance of x,y coordinates for each image is then calculated relatively from each of these anchor points. This is the ground truth

information for the Relative Offset Regressor. We illustrate some of the selected anchor points for 3 scenes, King's College, Shop Facade and Street in Figure 5.4.

Our proposed loss function has 3 separate components. We assign separate weights to each of these components, all of which are hyperparameters and are subject to be optimized through grid-search or any other hyperparameter optimization technique. We found specific ranges for these weights for outdoor scene localization task. The first part of the loss function is the cross-entropy calculated between the classifier output and the nearest anchor point to an input scene, which acts as its label. For this classification loss, we use a weight value ranging between 1-3 across all the scenes. The second component of the loss function is the Relative Offset (Translation) Regressor which predicts the relative distance of the x,y coordinates of the input scene from each of the anchor points. As a weight for this component, we use a value ranging between 4-30. Finally, the pose regressor predicts the remaining 5-DOF and a weight of 0.1-2 is assigned to it.

We use a learning rate varying from 0.00005 to 0.0005 for the scenes in the dataset. The Adam optimizer is used for optimization with a scheduler to decay the learning rate by half after every 30th epoch. The evaluation of accuracy is a subjective procedure. We consider a prediction for the scene to be accurate if the orientation is $<$2m and the corresponding pose is $<5^{\circ}$. Other thresholds for accuracy in evaluation, will naturally produce variations in performance calculation.

5.5 Results and Discussions

For validating the effectiveness of our approach, we first compare the median errors in the spatial and angular dimensions using our approach as well as the previous best PoseNet model [41] . For fairness of comparison, we experiment with using the same GoogleNet [76] feature extractor as the PoseNet result. Secondly we also experiment with newer feature extractors like DenseNet [33] and MobileNet [32] to drastically improve the accuracy and runtime speed. This also demonstrates that our approach generalizes to a variety of feature extractors. Furthermore we give more detailed analysis of anchor points as well as qualitative results.

5.5.0.0.1 Comparison with PoseNet on GoogleNet. We compare our method with PoseNet. We use the same GoogleNet feature extractor used in the PoseNet results, so that we can observe the improvement due to our approach rather than using a improved feature extractor. We report results without the cross entropy loss applied to the classification head since letting the network discover relevant the anchor points gave better results. We report the median error for all the scenes in the Cambridge Landmarks and the 7 Scenes datasets in Table 5.2. Also, it can be seen from Table 5.2, that we are doing better than the previous best method of PoseNet (with geometric reprojection loss) [40] in all the scenes, and improve the localization median error by around 8m in the Street scene. We also improve upon the previous best performance for each of the indoor scenes in the 7 Scenes dataset.

5.5.0.0.2 Improved Accuracy using DenseNet and Abalation study. We then experiment with the state of the art CNN feature extractors. DenseNet [33] is known to give improved accuracies for image

Scene	Area or Volume	Active Search (SIFT) [66]	Posenet Spatial LSTM [81]	Posenet sigma^2 weight [40]	Posenet Geom. Rep. [40]	Ours(DenseNet) (cross entropy)	Ours(DenseNet) (w/o cross entropy)	Ours(GoogleNet) (w/o cross entropy)
Great Court	$8000m^2$	-	-	$7.00m$, 3.65°	$6.83m$, 3.47°	$5.85m$, 3.61°	$4.64m$, 3.42°	**$5.89m$, 3.53°**
King's College	$5600m^2$	$0.42m$, 0.55°	$0.99m$, 3.65°	$0.99m$, 1.06°	$0.88m$, 1.04°	$0.55m$, 0.97°	$0.57m$, 0.88°	**$0.79m$, 0.95°**
Old Hospital	$2000m^2$	$0.44m$, 1.01°	$1.51m$, 4.29°	$2.17m$, 2.94°	$3.20m$, 3.29°	$1.45m$, 3.16°	$1.21m$, 2.55°	**$2.11m$, 3.05°**
Shop Facade	$875m^2$	$0.12m$, 0.40°	$1.18m$, 7.44°	$1.05m$ 3.97°	$0.88m$, 3.78°	$0.49m$, 2.42°	$0.52m$, 2.27°	**$0.77m$, 3.25°**
St. Mary's Church	$4800m^2$	$0.19m$, 0.54°	$1.52m$, 6.68°	$1.49m$, 3.43°	$1.57m$, 3.32°	$1.12m$, 2.84°	$1.04m$, 2.69°	**$1.22m$, 3.02°**
Street	$50000m^2$	$0.85m$, 0.83°	-	$20.7m$, 25.7°	$20.3m$, 25.5°	$8.19m$, 25.5°	$7.86m$, 24.2°	**$11.8m$, 24.3°**
Chess	$6m^2$	$0.04m$, 1.96°	$0.24m$, 5.77°	$0.14m$, 4.50°	$0.13m$, 4.48°	$0.06m$, 3.95°	$0.06m$, 3.89°	**$0.08m$, 4.12°**
Fire	$2.5m^2$	$0.03m$, 1.53°	$0.34m$, 11.9°	$0.27m$, 11.8°	$0.27m$, 11.3°	$0.16m$, 10.4°	$0.15m$, 10.3°	**$0.16m$, 11.1°**
Head	$1m^2$	$0.02m$, 1.45°	$0.21m$, 13.7°	$0.18m$, 12.1°	$0.17m$, 13.0°	$0.08m$, 10.7°	$0.08m$, 10.9°	**$0.09m$, 11.2°**
Office	$7.5m^2$	$0.09m$, 3.61°	$0.30m$, 8.08°	$0.20m$, 5.77°	$0.19m$, 5.55°	$0.11m$, 5.24°	$0.09m$, 5.15°	**$0.11m$, 5.38°**
Pumpkin	$5m^2$	$0.08m$, 3.10°	$0.33m$, 7.00°	$0.25m$, 4.82°	$0.26m$, 4.75°	$0.11m$, 3.18°	$0.10m$, 2.97°	**$0.14m$, 3.55°**
Red Kitchen	$18m^2$	$0.07m$, 3.37°	$0.37m$, 8.83°	$0.24m$, 5.52°	$0.23m$, 5.35°	$0.08m$, 4.83°	$0.08m$, 4.68°	**$0.13m$, 5.29°**
Stairs	$7.5m^2$	$0.03m$, 2.22°	$0.40m$, 13.7°	$0.37m$, 10.6°	$0.35m$, 12.4°	$0.13m$, 10.1°	$0.10m$, 9.26°	**$0.21m$, 11.9°**

Table 5.2 Comparison of median error. As reported in the last column, our model performs better than the best deep learning model PoseNet [40], making the gap with active search methods [66] (which uses 3D point cloud data unlike us) lesser. Use of improved feature extractors like DenseNet reduces the errors further.

Scene	**Mean Distance**	**Median Distance**	**Accuracy (<2m,<5°)**	**#Anchor Points**
Great Court	10.48m	5.85m	69.51 %	154
King's College	0.76m	0.57m	93.52 %	122
Old Hospital	2.93m	1.45m	85.94 %	110
Shop Facade	0.72m	0.52m	93.76 %	33
St. Mary's Church	1.62m	1.12m	88.95 %	146
Street	17.45m	9.89m	11.26 %	201

Table 5.3 We report the mean and median distances for each scene of the Cambridge Landmarks dataset using the DenseNet feature extractor. The accuracy is calculated by considering the orientation localization within 2m and the pose localization with 5° to be a correct prediction.

classification. In Table 5.2, we report the results of our proposed method with the DenseNet feature extractor and compare with all the previous methods. We report the results with and without the cross entropy term (see Section 5.3.1). For the Cambridge Landmarks, 5 out of the 6 scenes, the model performed better without the cross entropy loss, validating our approach of discovering the appropriate anchor point relative to which offsets needs to be calculated.

In Table 5.3, we also report the accuracy, the mean and the median distance localized to in orientation, and the median camera angle pose localization for each of the 6 scenes of the Cambridge Landmarks dataset. It can be seen from Table 5.3, that the accuracy for the Street scene is 11.26%. The Street scene is the most challenging scene from among the used scenes for the visual localization task since it covers more than $50000m^2$ in area and also contains multiple landmarks. We achieve state-of-the-art performance for translation and pose on the Street scene as well. Our method is able to significantly improve the translation and orientation localization for this particular scene from 20.3m to 7.86m (see Table 5.2) which is a considerable improvement.

Next, we perform an experiment with DenseNet as the feature extractor connected to a 6-DOF regressor directly. This is a sort of control for our proposed method and we do not use an anchor point classifier here. We provide a comparison of our proposed method and directly using the regressor in the

Cambridge Landmarks dataset. It is observed that our proposed method is able to perform much better than simply regressing the DOF from the DenseNet feature extractor. The results are summarized in Table 5.4. As can be seen our method gives lesser errors. This verifies that the improvement in median errors is indeed happening due to our approach and not due to the feature extractors only.

Scene	**Median Dist. (DenseNet + DOF Regressor)**	**Median Dist. (Our method)**	**Accuracy (DenseNet + DOF Regressor)**	**Accuracy (Our Method)**
Shop Facade	1.32m	**0.52m**	82.64%	**93.76%**
King's College	1.45m	**0.57m**	81.80%	**93.52%**

Table 5.4 Comparison between the DenseNet feature extractor followed by a simple regressor and our proposed method.

5.5.0.0.3 Runtime Analysis. The MobileNet [32] feature extractor is known to be significantly faster while maintaining accuracies. We illustrate the results when using it in Table 5.5.0.0.3. For some scenes in Cambridge Landmarks dataset, we empirically compare the performance and efficiency of 3 popular feature extractors, namely GoogleNet, DenseNet and MobileNet, when used with our proposed method. We observe from Table 5.5.0.0.3 that MobileNet provide a considerable better runtime performance, while still giving errors lesser than GoogleNet.

Scene	**DenseNet (Feature Extractor)**		**GoogleNet (Feature Extractor)**		**MobileNet (Feature Extractor)**	
	Performance	**FLOPs**	**Performance**	**FLOPs**	**Performance**	**FLOPs**
Kings College	$0.57m$, 0.88°	5998 M	$0.79m$, 0.95°	760 M	$0.67m$, 0.94°	569 M
Shop Facade	$0.52m$, 2.27°		$0.77m$, 3.25°		$0.60m$, 2.31°	

Table 5.5 Comparison of different feature extractors on the basis of performance and FLOPs using our proposed anchor point based method for visual re-localization on the Cambridge landmarks dataset.

5.5.0.0.4 Analysis of Anchor Points and Qualitative Results. An important hyperparameter for our approach is the number of frames between consecutive anchor point which is required for assigning anchor points as a preprocessing step. The outcome for this selection also determines the number of classes the classifier should predict since it is equal to the number of anchor points. We therefore plot the behavior of the median distance localization for translation and the overall accuracy, with the varying frame number selection (say k). This means that for a particular scene, we select every k^{th} frame. We observe in Figure 5.3, that we are able to get an optimum accuracy for a specific frame number which in turn makes the task easy of deciding the number of classes since it depends on k.

Finally, we showcase some qualitative results for the Cambridge Landmark scenes as well as for the 7 scenes dataset. It can demonstrated from those results that not only is the learned anchor point different from the nearest one but also better in terms of camera angle and avoiding occlusion.

5.6 Conclusion

We propose a novel approach for solving the visual relocalization problem, inspired by how humans estimate their location, by observing suitable landmarks. We model it as a multi task problem of

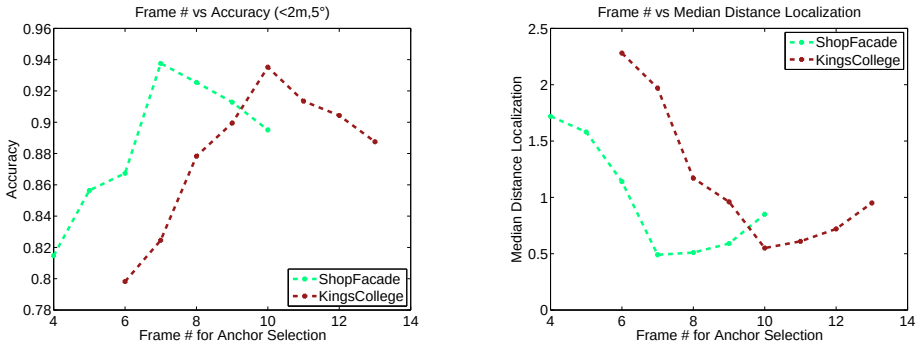


Figure 5.3 Plots showing how accuracy and median distance varies with different choices of frames interval between anchor points. We chose the optimum frame number for the dataset for anchor point selection.

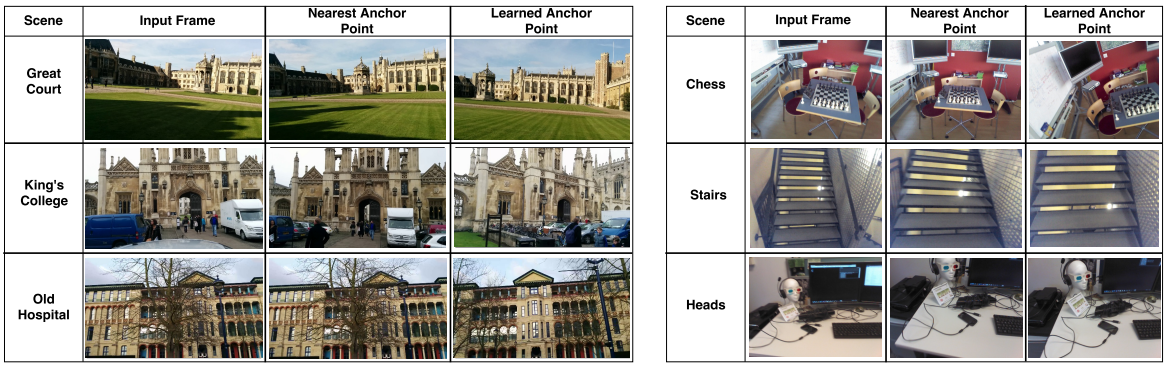


Figure 5.4 (Left) We contrast the nearest anchor point and the learned anchor point for an input query for the Cambridge dataset. Note that for the Old Hospital scene, the relevant anchor point learned is not blocked by a tree while the nearest anchor point is. Generalizing, learned anchor points gives a clear view of a landmark as compared to the nearest anchor point, validating our approach of discovering the relevant anchor point. (Right) Learned anchor points from the 7 Scenes dataset. In this case we observe a more zoomed in version of the input image is learned as the reference anchor point.

classification and relative offset regression. We propose a deep learning architecture and loss function which automatically discovers the anchor point relative to which accurate offset estimates can be predicted. We do not require each image to be tagged with the relevant landmark to train the classification part. Through our experiments, we show that our method achieves an improvement over PoseNet and its extensions in all scenes of the Cambridge Landmarks dataset as well as the indoor scenes of the 7 Scenes dataset. We achieve 1.5m and 4° in localization performance in 4 out of the 6 outdoor scenes in Cambridge Landmarks and 0.2m localization for the 7 indoor Scenes. Furthermore, our method outperforms simple replacement of the feature extractor followed by regression which further showcases the advantages of an anchor point classification and relative offset regression based method for the visual localization task.

Chapter 6

Conclusion and Future work

Traditional model compression techniques have been based on explicit compression and architectural modifications, for achieving compression. This has been a recent area of research. These traditional approaches have suffered from several drawbacks, the most important of which are the complicated training procedures as demonstrated by Han et.al. in Dense Sparse Dense Training [24]. Some of these methods which perform explicit compression only, suffer from significant performance drop as well, as is the case with pruning and quantization without retraining, and traditional low rank approximation methods. This has led to designing efficient architectures for model compression which other than designing a novel model requires one to propose a design specific training procedure as well. These new training procedures are mostly accompanied by new loss functions which are primarily task and dataset specific.

In this thesis, we touch on both these aspects of model compression. We aim to achieve an improvement over the traditional explicit neural network compression methods by making them suitably trainable. In order to do this, we demonstrate an iterative training procedure with pruning and quantization for the visual place recognition task as well propose a trained low rank approximation for the object recognition task. Both these approaches require us to make certain modifications to the loss function, to make the learning efficient and the compression effective. The second part of this thesis focuses on the more newer architectural modifications to provide efficient network design. We develop a new architecture for the object recognition task which not only achieves a significant reduction in parameters over the existing base model but also learns a latent hierarchy in the process. We propose a structured loss function and this helps us to learn an efficient embedding of the label space for this task. We also address the visual localization task by designing a novel deep neural network architecture, which follows an anchor point based training method. This is also accompanied by a proposed loss function which aids this training, thereby producing significantly improved results for the DOF Regression. Through this approach, we achieve better performance even though we use less parameters overall, and also improve on the memory and test-time over existing traditional nearest neighbour based approaches.

Recent research in this direction has also seen work in proposing such a training mechanism and loss function which learns the optimum architecture itself. The learnt architecture might vary from task to

task and across different datasets. Most popular approaches in this domain have been to define a model or mesh with all possibilities beforehand and go on reducing connections as the training progresses, defining a parametric activation function which can learn whether or not a layer is important, and designing a reinforced loss function which calculates pay-offs based on the performance of a particular architecture on a specific task. These are extremely interesting directions which can be focused on in the future as extensions to the thesis.

Related Publications

1. Soham Saha, Girish Varma, C.V.Jawahar. “Compressing Deep Neural Networks for Recognizing Places.” 4th Asian Conference on Pattern Recognition (ACPR), 2017. Nanjing,China, IEEE, 2017 (Spotlight Presentation). [63]

2. Soham Saha, Girish Varma, C.V.Jawahar. “On Learning Latent Hierarchies in Classes” International Conference on Pattern Recognition (ICPR), 2018. Beijing, China, IEEE, 2018. [64]

3. Soham Saha, Girish Varma, C.V.Jawahar. “Improved Visual Relocalization by Discovering Anchor Points” British Machine Vision Conference (BMVC), 2018. Northumbria University, Newcastle, United Kingdom, Springer, 2018. (Spotlight Presentation) [62]

Bibliography

[1] K. Ahmed, M. H. Baig, and L. Torresani. *Network of Experts for Large-Scale Image Categorization*, pages 516–532. Springer International Publishing, Cham, 2016.

[2] R. Arandjelović, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. NetVLAD: CNN architecture for weakly supervised place recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[3] R. Arandjelovic and A. Zisserman. All about vlad. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '13, pages 1578–1585, Washington, DC, USA, 2013. IEEE Computer Society.

[4] R. Arandjelović and A. Zisserman. Dislocation: Scalable descriptor distinctiveness for location recognition. In *Asian Conference on Computer Vision*, pages 188–204. Springer, 2014.

[5] A. Babenko and V. Lempitsky. Aggregating local deep features for image retrieval. In *Proceedings of the IEEE international conference on computer vision*, pages 1269–1277, 2015.

[6] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky. Neural codes for image retrieval. In *European conference on computer vision*, pages 584–599. Springer, 2014.

[7] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[8] S. Choudhary and P. Narayanan. Visibility probability structure from sfm datasets and applications. In *European conference on computer vision*, pages 130–143. Springer, 2012.

[9] M. Cummins and P. Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 27(6):647–665, 2008.

[10] J. Delhumeau, P.-H. Gosselin, H. Jégou, and P. Pérez. Revisiting the vlad image representation. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 653–656. ACM, 2013.

[11] J. Deng, A. C. Berg, K. Li, and L. Fei-Fei. What does classifying more than 10,000 image categories tell us? In *Proceedings of the 11th European Conference on Computer Vision: Part V*, ECCV'10, pages 71–84, Berlin, Heidelberg, 2010. Springer-Verlag.

[12] J. Deng, N. Ding, Y. Jia, A. Frome, K. Murphy, S. Bengio, Y. Li, H. Neven, and H. Adam. *Large-Scale Object Classification Using Label Relation Graphs*, pages 48–64. Springer International Publishing, Cham, 2014.

[13] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

[14] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

[15] A. Frome, G. Corrado, J. Shlens, S. Bengio, J. Dean, M. Ranzato, and T. Mikolov. Devise: A deep visual-semantic embedding model. In *Neural Information Processing Systems (NIPS)*, 2013.

[16] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[17] Y. Gong, L. Wang, R. Guo, and S. Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. In *European conference on computer vision*, pages 392–407. Springer, 2014.

[18] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[19] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1379–1387. Curran Associates, Inc., 2016.

[20] 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. In *International Symposium on Computer Architecture (ISCA 2016)*, 2016.

[21] 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. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 243–254, Piscataway, NJ, USA, 2016. IEEE Press.

[22] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR'16 best paper award)*, 2015.

[23] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016.

[24] S. Han, J. Pool, S. Narang, H. Mao, S. Tang, E. Elsen, B. Catanzaro, J. Tran, and W. J. Dally. Dsd: Regularizing deep neural networks with dense-sparse-dense training flow. *arXiv preprint arXiv:1607.04381*, 2016.

[25] B. Hassibi, D. Stork, and G. Wolff. Optimal brain surgeon and general network pruning. In *International Conference on Neural Networks*, volume 1, pages 293–299, 1993.

[26] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[27] 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*, pages 770–778, 2016.

[28] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[29] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[30] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[31] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[32] 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. *CoRR*, abs/1704.04861, 2017.

[33] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[34] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[35] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman. Deep structured output learning for unconstrained text recognition. *arXiv preprint arXiv:1412.5903*, 2014.

[36] H. Jegou, F. Perronnin, M. Douze, J. Sanchez, P. Perez, and C. Schmid. Aggregating local image descriptors into compact codes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(9):1704–1716, Sept. 2012.

[37] D. Kalman. A singularly valuable decomposition: the svd of a matrix.

[38] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, 24(7):881–892, 2002.

[39] A. Kendall and R. Cipolla. Modelling uncertainty in deep learning for camera relocalization. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 4762–4769. IEEE, 2016.

[40] A. Kendall and R. Cipolla. Geometric loss functions for camera pose regression with deep learning. *arXiv preprint arXiv:1704.00390*, 2017.

[41] A. Kendall, M. Grimes, and R. Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015.

[42] J. Kim, M. El-Khamy, and J. Lee. Bridgenets: Student-teacher transfer learning based on recursive neural networks and its application to distant speech recognition. *CoRR*, abs/1710.10224, 2017.

[43] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[44] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, page 2012.

[45] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[46] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. *Advances in neural information processing systems 2, NIPS 1989*, 2:598–605, 1990.

[47] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *International conference on artificial intelligence and statistics*, 2015.

[48] R. Li, Q. Liu, J. Gui, D. Gu, and H. Hu. Indoor relocalization in challenging environments with dual-stream convolutional neural networks. *IEEE Transactions on Automation Science and Engineering*, 2017.

[49] Y. Li, N. Snavely, and D. P. Huttenlocher. Location recognition using prioritized feature matching. In *European conference on computer vision*, pages 791–804. Springer, 2010.

[50] Y. Li, N. Snavely, D. P. Huttenlocher, and P. Fua. Worldwide pose estimation using 3d point clouds. In *Large-Scale Visual Geo-Localization*, pages 147–163. Springer, 2016.

[51] M. Lin, Q. Chen, and S. Yan. Network in network. *International Conference on Learning Representations (ICLR)*, abs/1312.4400, 2014.

[52] I. Melekhov, J. Ylioinas, J. Kannala, and E. Rahtu. Relative camera pose estimation using convolutional neural networks. In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 675–687. Springer, 2017.

[53] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[54] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[55] N. Piasco, D. Sidib, C. Demonceaux, and V. Gouet-Brunet. A survey on visual-based localization: On the benefit of heterogeneous data. *Pattern Recognition*, 74:90 – 109, 2018.

[56] A. Prabhu, H. Krishna, and S. Saha. Adversary is the best teacher: Towards extremely compact neural networks, 2018.

[57] A. S. Razavian, J. Sullivan, S. Carlsson, and A. Maki. Visual instance retrieval with deep convolutional networks. *ITE Transactions on Media Technology and Applications*, 4(3):251–258, 2016.

[58] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[59] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. In *In Proceedings of ICLR*, 2015.

[60] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[61] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[62] S. Saha, G. Varma, and C. Jawahar. Improved visual relocalization by discovering anchor points. *arXiv preprint arXiv:1811.04370*, 2018.

[63] S. Saha, G. Varma, and C. V. Jawahar. Compressing deep neural networks for recognizing places. In *2017 4th IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 352–357, Nov 2017.

[64] S. Saha, G. Varma, and C. V. Jawahar. Class2str: End to end latent hierarchy learning. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 1000–1005, Aug 2018.

[65] T. Sattler, B. Leibe, and L. Kobbelt. Improving image-based localization by active correspondence search. In *European conference on computer vision*, pages 752–765. Springer, 2012.

[66] T. Sattler, B. Leibe, and L. Kobbelt. Efficient effective prioritized matching for large-scale image-based localization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(9):1744–1756, Sept 2017.

[67] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *CVPR*, pages 2930–2937, 2013.

[68] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2930–2937, 2013.

[69] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[70] E. Spyromitros-Xioufis, S. Papadopoulos, I. Kompatsiaris, G. Tsoumakas, and I. Vlahavas. A comprehensive study over vlad and product quantization in large-scale image retrieval. *IEEE Transactions on Multimedia*, 2014.

[71] S. Srinivas and R. V. Babu. Learning neural network architectures using backpropagation. *arXiv preprint arXiv:1511.05497*, 2015.

[72] N. Srivastava and R. R. Salakhutdinov. Discriminative transfer learning with tree-based priors. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2094–2102. Curran Associates, Inc., 2013.

[73] L. Svarm, O. Enqvist, M. Oskarsson, and F. Kahl. Accurate localization and pose estimation for large 3d models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 532–539, 2014.

[74] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence,February 4-9, 2017, San Francisco, California, USA.*, pages 4278–4284, 2017.

[75] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[76] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.

[77] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[78] G. Tolias, R. Sicre, and H. Jégou. Particular object retrieval with integral max-pooling of cnn activations. *arXiv preprint arXiv:1511.05879*, 2015.

[79] 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*, pages 883–890, 2013.

[80] A.-M. Tousch, S. Herbin, and J.-Y. Audibert. Semantic hierarchies for image annotation: A survey. *Pattern Recogn.*, 45(1):333–345, Jan. 2012.

[81] F. Walch, C. Hazirbas, L. Leal-Taixé, T. Sattler, S. Hilsenbeck, and D. Cremers. Image-based localization with spatial lstms. *ICCV*, 2016.

[82] S. Watanabe, T. Hori, J. Le Roux, and J. R. Hershey. Student-teacher network learning with enhanced features. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 5275–5279. IEEE, 2017.

[83] T. Weyand, I. Kostrikov, and J. Philbin. Planet-photo geolocation with convolutional neural networks. In *European Conference on Computer Vision*, pages 37–55. Springer, 2016.

[84] Z. Yan, H. Zhang, R. Piramuthu, V. Jagadeesh, D. DeCoste, W. Di, and Y. Yu. Hd-cnn: Hierarchical deep convolutional neural networks for large scale visual recognition. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2740–2748, Dec 2015.

[85] S. Yang, P. Luo, C. C. Loy, K. W. Shum, and X. Tang. Deep representation learning with target coding. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 3848–3854. AAAI Press, 2015.