Master's Thesis

A Novel Filter Pruning Algorithm for Vision Tasks based on Kernel Grouping

Lee Jongmin

Supervisor Chong Nakyoung

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
(Information Science)

Month, Year

Abstract

Computer vision was researched since the late 1960's but image classification is still a challenging task. After Geoffrey Hinton won the ImageNet Large Scale Vision Recognition Challenge(ILSVRC)[1] as known as ImageNet with AlexNet[2] on 2012, people started researching about Convolutional Neural Networks(CNN) for image classification. Modern neural networks[3, 4] achieved nearly 90% accuracy on the ImageNet dataset, however the number of parameters are tremendously large. The most popularly used CNN models are VGG16[5], InceptionV3[6], and ResNet18[7], which have 138M, 24M, 11M parameters respectively. Involution[8] successfully reduced the number of parameters of CNNs by replacing all the 3×3 convolution kernels with involution kernels, which use 1×1 convolution for the convolution layer's kernel generation. In result, involution networks achieved similar performance with 34.1% fewer parameters. Although the models introduced above have great performance on image tasks, they still require high computational cost therefore applying deep neural networks on mobile devices remain challenging.

There were several approaches for reducing the number of parameters trying not to lose the performance. Knowledge distillation[9] is a method which use a dense network as a teacher model, and a sparse network as a student model. The term distillation means that the student model learns the soft label of the teacher model. By learning both hard label which is the loss function of the prediction and the ground truth, and the soft label which is the loss between the prediction of the student and the teacher model, the student model can mimic the output of the teacher model. If properly trained, the student model will act similarly with the teacher model with fewer parameters. Raphael Gontijo Lopes et al. proposed a knowledge distillation algorithm which does not require data when training the student model by reconstructing the input image using the activation statistics and layer gradients. Gongfan Fang et al. improved the data free distillation algorithm by training an image generator for data reconstruction.

Filter pruning[10] is a method for reducing the number of filters in CNN. When pruning the filters we sort by the sum of weights for each filter for every layer. Since filters that have weights that are close to 0 will not affect much of the performance of the model, therefore when given a proper threshold we can get a sparse model with a similar performance.

However, it is hard to apply the conventional pruning method for involution since it requires sorting the filters. Involution has reshaping layers therefore if the filters are sorted, they lose the spatial information. To overcome this problem we need to rewrite the code for the model which is hard to implement and time consuming. In this research we propose a pruning method called the model diet which is easy to implement, and effective for CNN models including involution. Instead of sorting the filters for each layer, we reduce a certain portion of the filters by grouping the kernel weights therefore for involution, the spatial information is not lost. Since the model depth is maintained but the filters are reduced, we call this pruning method a model diet, and we will show that diet models have faster convergence compared with randomly initialized models.

The model diet is consisted of 2 stages, the kernel grouping stage and the group selection stage. kernel grouping is an algorithm that splits the kernel weights into groups. The kernel weights are split in order therefore when applied to involution, the involution kernel does not lose the spatial information. Once the kernel weights are split into groups, we take the sum of the weights for each group. Then we use the group that has the biggest sum ans we call this operation group selection.

Deep learning frameworks such as Tensorflow or Pytorch save the model's weights as matrices. For involution the kernel weights are saved as a vector. When the weights are loaded, the vector reshapes itself into the corresponding shape. Therefore the element of the vector indicates a certain location in an image. If we apply conventional pruning algorithms, the weights of the involution kernel will be sorted also, resulting a loss of spatial information. However the kernel grouping keeps the order of the weights, therefore when applied to involution the loss of the spatial information does not happen. Also the computational complexity of the model diet is $\mathcal{O}(n)$ where the computational complexity of the conventional pruning is $\mathcal{O}(n \log n)$. Since model diet has the same computational complexity with selecting the maximum element in a vector where conventional methods have the same computational complexity with sorting.

In this research we show the effectiveness of the model diet in 3 vision tasks, image classification, image segmentation, and depth estimation. We test the performance of the diet model and the random initialized model. The diet model showed faster convergence and performance compared with the random initialized model. For image segmentation the dataset was easy to generalize and lacked difficulty therefore the diet model and the random initialized model showed equal performance but the diet model had faster convergence. For depth estimation both the diet model and the random initialized model showed poor performance even though the loss converged. Also the difference between groups was studied. We split the full model into 2 groups and compared the performance. Both the group with the bigger sum and smaller sum showed equal performance and speed of convergence. Since pruning can be regarded as weight initialization, we hypothesize that both the group with bigger sum and smaller sum started from a different location, but shared the same local optimum point.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

After the AlexNet was proposed by Hinton[2], people started to gain interest in Convolutional Neural Networks(CNNs). VGG[5] used 3×3 kernels for convolution and stacked a lot of layers and made the network deep, Inception replaced big sized kernels into multiple 3×3 kernels and used 1×1 convolutional kernels for channel reduction, ResNet[7] used skip connections to make the network even deeper. Although CNNs successfully increased the accuracy for ImageNet, they require a big amount of computational cost and memory.

1.1 Background

1.1.1 Number of parameters

The number of parameters in deep learning models mean the power of expression for a given dataset. Therefore reducing the parameter also means losing the power for generalizing the data. Recent CNN models reduced the redundancies by replacing big kernels with smaller ones like Inception, or even generate the kernels for convolution with 1×1 convolution like Involution. According to the lottery ticket hypothesis[11], there might exist a sparse network that has similar performance with the dense network which means that CNN models also have redundancies which can be reduced.

Model	Parameters(M)	Top-1 Accuracy	Top-5 Accuracy
VGG16	138.4	74.4%	91.9%
InceptionV3	23.9	77.12%	93.7%
ResNet50	25.6	77.15%	93.29%
EfficientNet-L2	480	90.2%	98.8%
NFNet-F4+	527	89.2%	97.6%
RedNet26	9.23	75.96%	93.19%

Table 1.1: The number of parameters and ImageNet accuracy

1.1.2 Disadvantages of CNN

According to Duo Li et al.[8] convolution have "channel-specific and spatial-agnostic" features. The term spatial-agnostic means that the we limit the receptive field of convolution kernels to extract the features in various positions. However, if we have same features with different scale or features that are far apart, convolution lacks the ability of capturing these kind of features. This in result makes the classifier to make filters that capture different features of the same object, causing the inter channel redundancy.

1.2 Research purpose

This research aims to reduce the redundancies of CNN using a filter pruning algorithm while keeping the implementation as simple as possible. This algorithm is applicable on various models including Involution models. Involution kernels are different from convolution kernels since they include reshaping layers so conventional pruning algorithms are hard to be applied. If we sort the filters like conventional methods[10], it causes the involution kernel to lose the spatial information. In this research we propose a method which is easy to implement and reduces about 50% of the model's weights. Also we show that this method could be applied in other CNN models without losing much accuracy.

1.3 Thesis outline

Chapter2: This chapter explains the related work such as involution kernels, filter pruning for CNN.

Chapter3: This chapter explains the details of the proposed method.

Chapter4: This chapter explains the experiments and evaluation we earned including the dataset used.

Chapter5: This chapter gives the overall summary and the future work.

Chapter 2

Related Works

2.1 Involution

Involution is a type of a kernel that can reduce the inter channel redundancy of CNNs. It reversed the inherence of convolutions and have spatial-specific and channel-agnostic features. This term means that Involution kernels refer to the channels for each pixel when generating the kernel weights. Unlike randomly generated kernels like CNN, Involution takes the channel information when generating kernels. This makes a difference because for Involution kernels the kernel weights for each pixel differs. Convolution kernels on the other hand, share the same weights for every pixel. This makes Involution kernels have wider receptive fields compared to convolution kernels. This makes Involution more similar to self attention rather than convolution. Self attention is a relation between 2 pixels but involution takes more pixels in regard, so we can consider Involution a more generalized form of self attention.

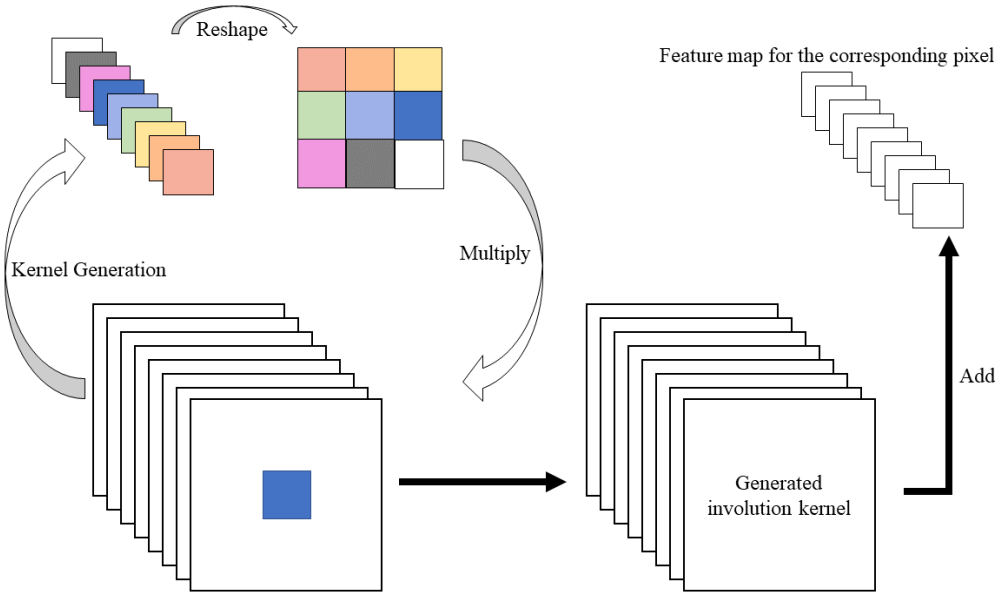

Figure 2.1: Overview of involution

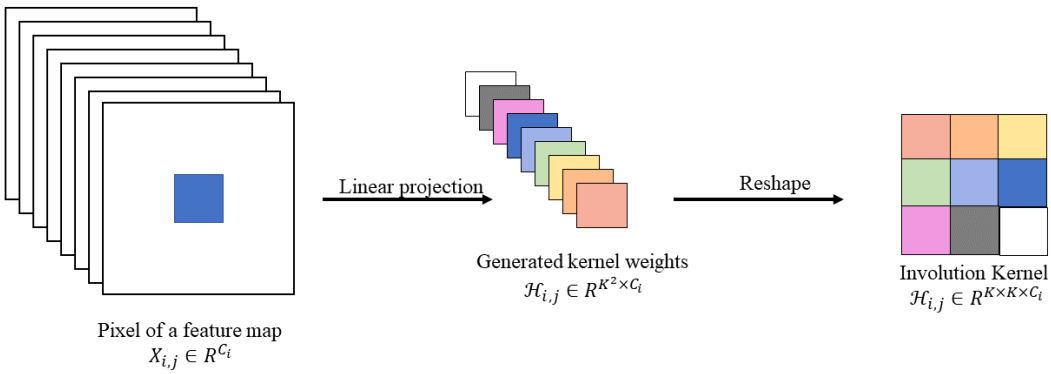

Figure 2.2: Involution kernel generation

2.1.1 Involution kernels

Let $X \in \mathbb{R}^{H \times W \times C_i}$ be a feature map and $X_{i,j} \in \mathbb{R}^{C_i}$ be a pixel in the feature map where C_i denotes the number of channels of the feature map. We also define involution filters $\mathcal{H} \in \mathbb{R}^{H \times W \times K \times K \times C_i}$ and a specific involution filter $\mathcal{H}_{i,j} \in \mathbb{R}^{K \times K \times C_i}$. We then define a linear projection $W : \mathbb{R}^{C_i} \mapsto \mathbb{R}^{K^2 \times C_i}$ where K denotes the involution kernel size. For each pixel $X_{i,j}$, linear projection W will be operated and the output will be reshaped into C_i, $K \times K$ shaped kernels. Since the involution filter is a concatenated tensor of involution kernels, we can write the entire kernel generation as below.

$$\mathcal{H}_{i,j} = Reshape(W(X_{i,j}))$$

We define the kernel generation function as $\phi : \mathbb{R}^{C_i} \mapsto \mathbb{R}^{K \times K \times C_i}$ such that

$$\mathcal{H}_{i,j} = \phi(X_{i,j})$$

Once the involution kernel is generated we perform a multiply-add operation through the channel dimension. The output Y will be shown below.

$$Y_{i,j,k} = \sum_{(u,v) \in \Delta_K} \mathcal{H}_{i,j,u,v,k} \cdot X_{i+u,j+v,k} \tag{2.1}$$

where u, v denotes relative kernel index and k denotes the channel index.

$$\Delta_K = \{-\lfloor \frac{K}{2} \rfloor \cdots \lfloor \frac{K}{2} \rfloor\} \times \{-\lfloor \frac{K}{2} \rfloor \cdots \lfloor \frac{K}{2} \rfloor\} \in \mathbb{Z}^2$$

2.1.2 Compared with convolution

Let $\mathcal{F} \in \mathbb{R}^{C_o \times C_i \times K \times K}$ be a set of convolution filters. For each filter $\mathcal{F}_k \in \mathbb{R}^{C_i \times K \times K}$, it operates a multiply-add operation for each pixel of the feature map X. Therefore we write the output as below.

$$Y_{i,j,k} = \sum_{c=1}^{C_i} \sum_{(u,v) \in \Delta_K} \mathcal{F}_{k,c,u,v} \cdot X_{i+u,j+v,c} \tag{2.2}$$

Unlike Involution kernels, the weights of convolution kernels are randomly initialized without conditioning. That means for each pixel $X_{i,j}$, they share the same kernel. On the other hand, involution kernels are conditioned by the pixel's values, every pixel have different kernel weights. From equation 2.1 and 2.2, the big difference between Involution and convolution is that filters $\mathcal{F}$ is randomly generated. Involution kernels are conditioned by the pixel's channel information and makes the channel-agnostic features of Involution. Each weight of the involution kernel is a linear transform of the channels for the corresponding pixel.

Since convolution shares the same kernel for every pixel, this also means that we have limited the receptive field down to the kernel size. Involution on the other hand, different weights for each pixel is applied therefore the receptive field is not limited. This makes Involution similar to self attention.

2.1.3 Compared with self attention

Attention mechanisms were first invented for natural language processing tasks. The basic idea is to attend every component of the input when we compute the output. For example, in machine translation tasks, when we output the words of the target language we consider the entire sentence of the source sentence. While attention was a model designed for different sources, eg. different languages in machine translation, however self attention was a model designed for the same source which means it attends itself when producing the output.

There were many attempts to apply attention mechanisms in vision tasks. Vision transformers[12, 13] have remarkable performance on image classification tasks and Kelvin Xu et al.[14] used visual attention for image captioning. Trying to apply attention on vision tasks was another attempt to avoid the limited receptive field and get a wider vision. Self attention on images work as the equation below

$$Y_{i,j,k} = \sum_{u,v} (\mathbf{Q} \cdot \mathbf{K}^\top)_{i,j,u,v,k} \cdot \mathbf{V}_{u,v,k}$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathcal{R}^{H \times W \times d_k}$ are the query, key and value respectively. Note that $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ are linear transforms of the input feature map and $\mathbf{Q} \cdot \mathbf{K}^\top \in \mathcal{R}^{H \times W \times H \times W \times d_k}$ is the attention value $\mathbf{A}$.

If we set the involution kernel to be the same size with the input feature map and apply the multiply-add operation to the entire image, we get the same shaped tensor with the attention value $\mathbf{A}$. Therefore we can regard the attention value as a type of an involution kernel which relies only on $\mathbf{Q}$ without $\mathbf{V}$. Since involution kernel can be an arbitrary shape, we can think involution as a generalized version of self attention.

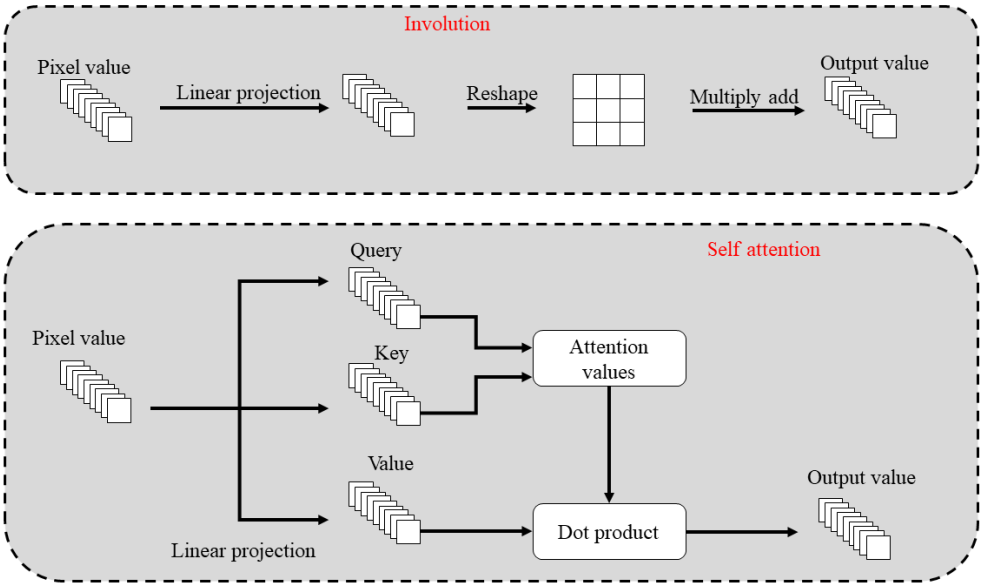

Figure 2.3: Difference between self attention and involution

2.1.4 RedNet

RedNet is the first neural network which use Involution. The architecture follows the work of ResNet but the convolution layers are replaced with involution layers and all the 1×1 convolutions for channel projection and fusion are retained. RedNet has less parameters compared to ResNet and has better performance in various vision tasks.

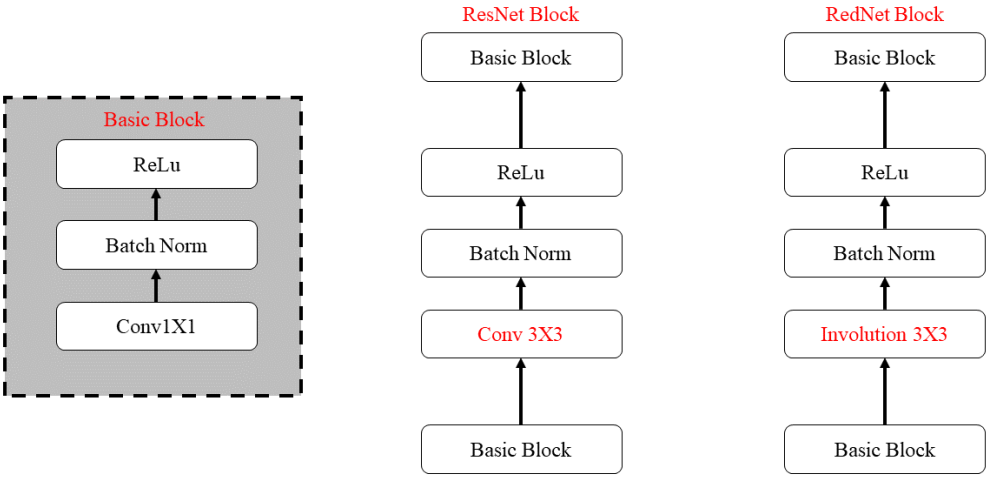

Figure 2.4: ResNet block(figure on the left) and RedNet Block(figure on the right)

Model	Parameters(M)	FLOPs(G)	Top-1 Accuracy(%)
ResNet-26	13.7	2.4	73.6%
RedNet-26	9.2	1.7	75.9%
ResNet-38	19.6	3.2	76.8%
RedNet-38	12.4	2.2	77.6%
ResNet-50	25.6	4.1	76.8%
RedNet-50	15.5	2.7	78.4%
ResNet-101	44.6	7.9	78.5%
RedNet-101	25.6	4.7	79.1%
ResNet-152	60.2	11.6	79.3%
RedNet-152	34.0	6.8	79.3%

Table 2.1: Comparison between RedNet and ResNet proposed by Duo Li et al.[8]

2.2 Model compression

Since CNN can handle a lot of tasks for vision, various models were developed to apply CNN models in mobile devices[15–17]. An object detection model using depthwise convolution were proposed in [15]. Depthwise convolution is an algorithm, which uses the same convolution kernel for every channel. [18] proposed an algorithm which use multiple sized kernels. Their study showed that using different sized kernels and concatenating the feature maps have better performance compared with the conventional depthwise convolution. Models proposed in [16] and [17] are image classification models that use channel shuffle for group convolution to reduce the computational complexity. Their proposal ghost module uses a low computational costly operation to reduce the number of parameters and computational complexity. However depthwise convolution has less power of expression, and channel shuffle regard the channel groups mutually independent therefore the information within channel groups is not exchanged. However the number of parameters are strongly related with the models performance.

Jonathan Frankle and Michael Carbin[11] hypothesize that there exists a sparse network that has similar or even better performance compared with the dense network. Although there might exist a winning ticket i.e. the sparse network which has similar performance with the dense network, it is tough to find the initial weights of the parameters. Rather than finding the winning ticket, pruning the weights from a pre-trained model is easier. Knowledge distillation distills the knowledge from a dense model i.e. a teacher model to a sparse one i.e. the student model. Filter pruning prunes the filters of a CNN model and re-train the pruned model.

Ofir Zafrir et al. proposed an algorithm named prune once for all(PruneOFA) in [19]. PruneOFA is consists of 2 steps, teacher preperation and student pruning. The teacher preperation step use a pre-training dataset to train the teaher model. After training the teacher model, the student model is generated and pre-trained using the teacher model's weights. Then we prune the teacher model using gradual

magnitude pruning(GMP) and learning rate rewinding(LRR). After the pruning is done, we use the task dataset to finetune the pruned model.

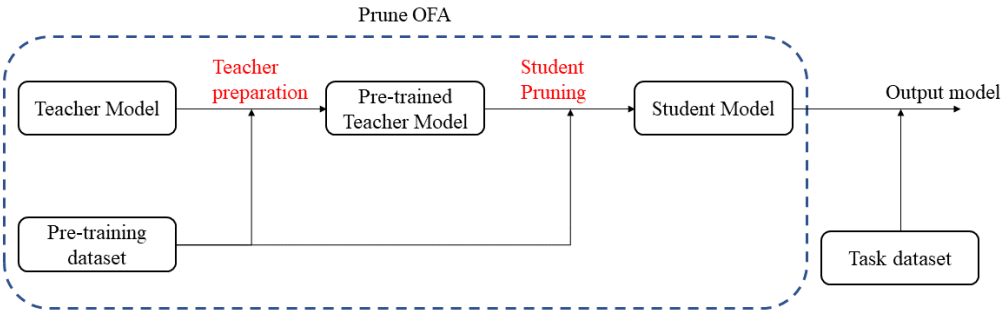

Figure 2.5: The pipeline of PruneOFA

GMP was first proposed in [20]. GMP has a 3 step training pipeline. The steps are as below.

1. Train connectivity
2. Prune connections
3. Train weights

The train connectivity step learns which connection of a network is important. The prune connection step prunes the weights with a certain threshold. The final step decides the final values of each connection. The second and third step is iteratively repeated so that the model is gradually pruned. LRR[21] is an algorithm to reduce a model's sparsity into a certain rate. It's similar to GMP but the training mechanism is slightly different. GMP had both updating and pruning step in one epoch, but LRR repeats to train the network and prune 20% of the total weights.

2.2.1 Knowledge distillation

Hinton et al.[9] proposed knowledge distillation to train a sparse network i.e. the student model by learning the soft labels of the dense network i.e. the teacher model. The student model use both hard labels from the ground truth and the soft labels from the teacher model. The term "soft" from soft label comes from

the temperature constant applied in the softmax function. The softmax function is defined as below.

$$softmax(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)}$$

We can penalize the softmax function using the temperature constant T. Then the penalized softmax can be written as below.

$$softmax_p(x_i) = \frac{exp(x_i/T)}{\sum_j exp(x_j/T)}$$

To show the effect of this temperature constant, we let a, b as the logits for a binary classification where $a \geq b$. Then, the difference between logits is $\lim_{T\to\infty} e^{a/T} - e^{b/T}$. Which is also $\lim_{T\to\infty} e^{a/T}(1 - e^{\frac{b-a}{T}})$. If $T \to \infty$, $\lim_{T\to\infty} e^{a/T}(1 - e^{\frac{b-a}{T}}) = 0$ therefore the output becomes a uniform probability distribution. When $T \to 0^+$, the difference between logits i.e. $\delta(a, b) = \lim_{T\to 0^+} e^{a/T}(1 - e^{\frac{b-a}{T}}) \to \infty$ therefore $\lim_{T\to 0^+} \frac{e^{a/T}}{e^{a/T}+e^{b/T}} = 1$ and similarly $\lim_{T\to 0^+} \frac{e^{b/T}}{e^{a/T}+e^{b/T}} = 0$. Hence we can say as T gets bigger, the output gets harder and as it gets smaller the output get softer.

As shown in Figure 2.6 we define the soft label loss i.e. kd loss $\mathcal{L}_{KD}$ as

$$\mathcal{L}_{KD} = Softmax_p(MSE(S(x, \theta_S), T(x, \theta_T)), \tau)$$

where MSE is the mean square error and $S(x, \theta_S)$, $T(x, \theta_T))$ stands for the output of the student and teacher model respectively. The hard label loss is the common cross entropy loss for multi label classification.

$$\mathcal{L}_{CE} = CE(S(x, \theta_S), T(x, \theta_T))$$

The overall loss function is defined as

$$\mathcal{L} = \sum_{\mathbb{B}} \mathcal{L}_{KD} + \mathcal{L}_{CE}$$

which means the sum of all losses along the batch dimension. The penalized softmax function makes the student model to mimic the characteristics of the teacher model, where the hard label from the ground truth guarantees the accuracy of the prediction. If successfully learned, we can train a model which acts similar with fewer parameters.

Data free distillation[22] is a knowledge distillation method which doesn't require data for knowledge distillation. The conventional knowledge distillation method is to minimize the 2 loss functions. One, the error between the ground truth and second, the error between the teacher model and the student model. However data free distillation methods does not require the original data used when training the teacher model. Instead reconstruts the input data by using a random gaussian noise for the teacher model's input. The gaussian noise goes through the top layer i.e. the classification layer to the bottom layer i.e. the input layer. We use the gradients of each layer to reconstruct the input image.

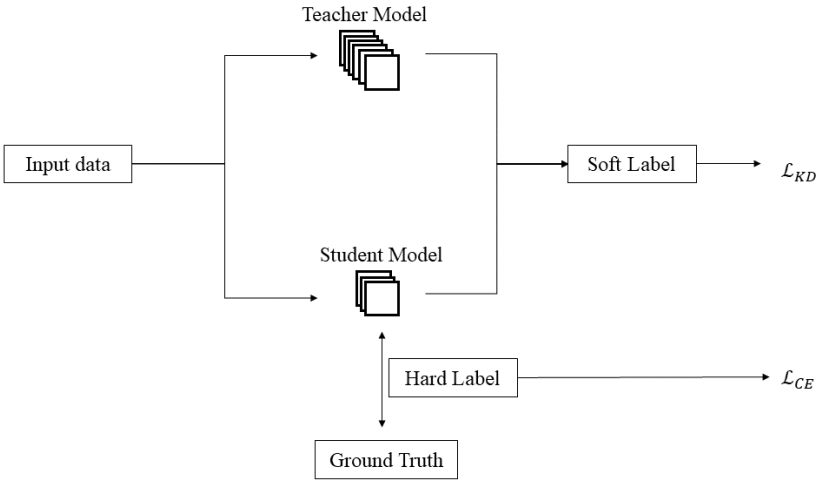

Figure 2.6: The pipeline of knowledge distillation

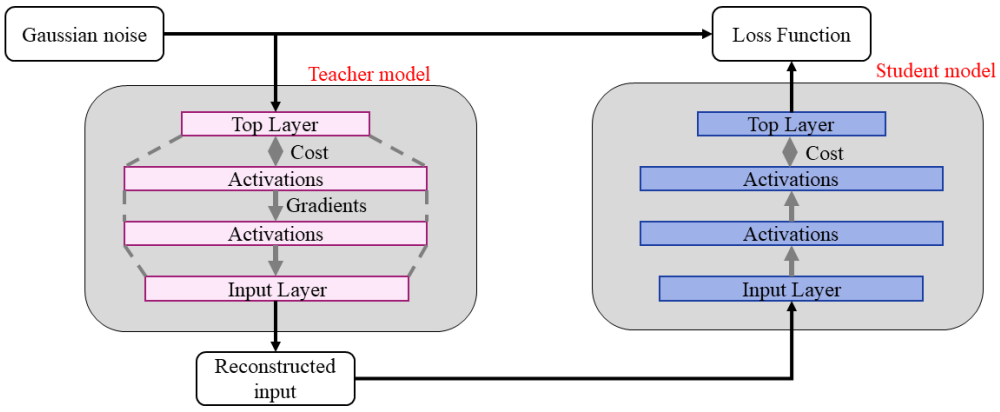

Figure 2.7: The pipeline of top layer statistics

When training the student model we can use each layer's activation as a ground truth. There are 2 ways using the teacher model's activations. We can either use the classification layer only or all layer's activation. If we use the classification layer, we call this method top layer activation statistics and if we use all layer's activation we call this all layers statistics.

Data free adversarial distillation[23](DFAD) is an algorithm that use a GAN's generator for sample generation. It mimics the learning process of humans. The generator will be trained to sample harder examples. Unlike the previous data free

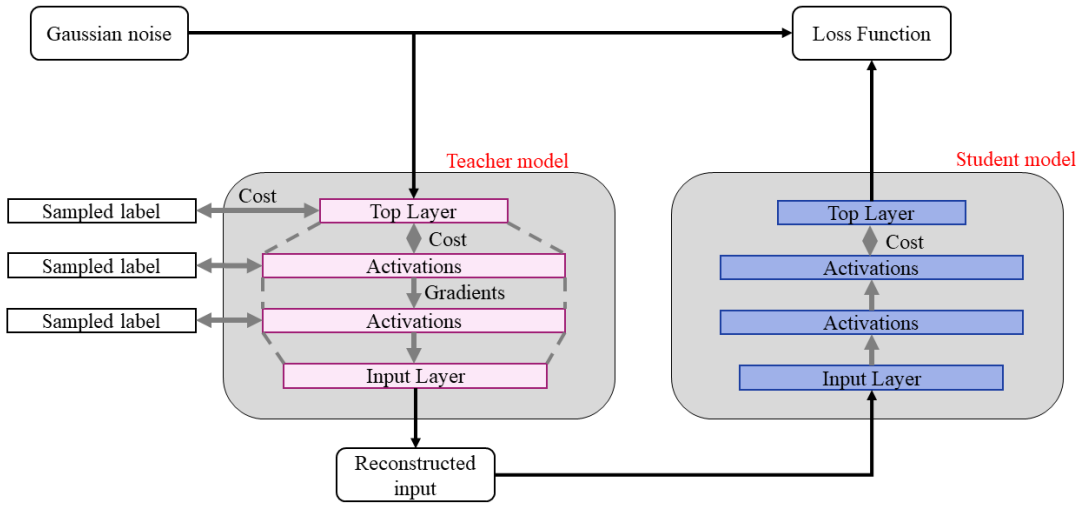

Figure 2.8: The pipeline of all layer statistics

distillation, this algorithm does not require the layer's activation statistics. Instead it generates images itself to train the student network. The training process of DFAD is split into 2 stages.

The first step which we call the imitation step, we fix the weights of the generator and train the parameters of the student model. In the imitation step we minimize the loss between the teacher model and the student model. The second step is called the generation step and we fix the student model's parameters and train the generator. The student and the teacher model acts as the discriminator when training the generator. The generator is trained to maximize the loss between the teacher model and the student model.

2.2.2 Filter pruning

Pruning was widely used throughout deep learning and it can also be used in model compression. Hao Li et.al. [10] proposed a filter pruning algorithm to reduce the number of weights in CNN. The pruning process is as below.

For each filter in the network, we take the sum of all weights in the filter. Then we sort the filters by the sum of their weights and remove the filters with a certain proportion, and if the sum is smaller than the threshold, we discard the filter. Filters which have weights that are close to 0 means that they are likely to have less effect on the performance. Removing the filters that less affect the performance can result in reducing the parameters of the network.

Fig.2.10 shows the entire process of filter pruning for convolution kernels. Each kernels are colored differently. First, we take the sum weights for each kernel. Then, we prune kernels with a certain threshold by the sum of the kernel's weights. As a result we can remove kernels that less affect the model's performance.

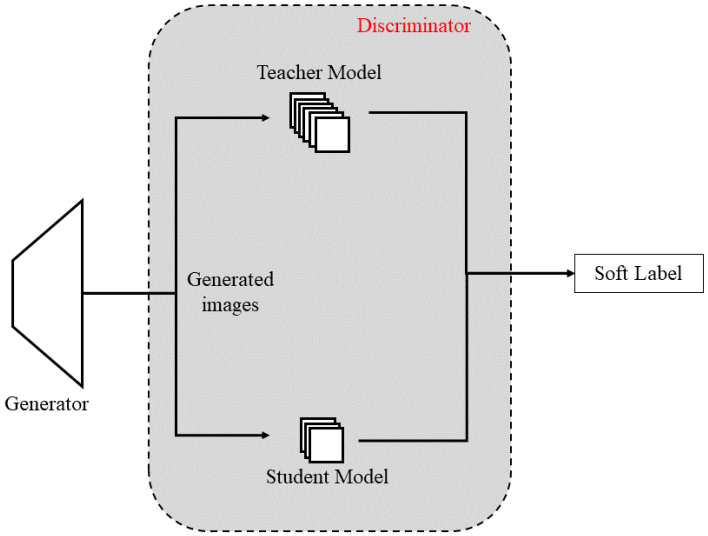

Figure 2.9: The pipeline of data free adversarial distillation

Algorithm 1 Filter pruning

Initialize $Newfilter = \emptyset$
for each $f \in \mathcal{F}$ **do**
    $f_{C_o,C_i} \leftarrow \sum_K f_{C_o,C_i}$
end for
Sort $\mathcal{F}$
$Newfilter \leftarrow Newfilter$[:len($Newfilter$)*ratio]
for each $f \in \mathcal{F}$ **do**
    if $f_{C_o,C_i} \geq$ Threshold **then**
        $Newfilter$.append(f_{C_o,C_i})
    else
        Pass
    end if
end for

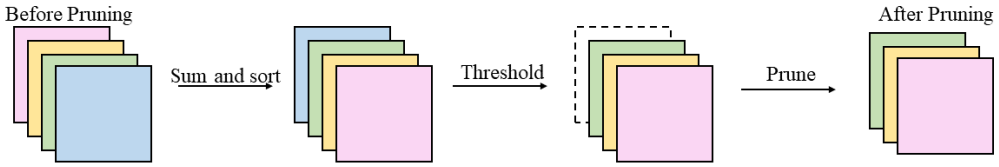

Figure 2.10: Filter pruning

Chapter 3

Proposed method

The key point of pruning is that kernels are mutually independent. Therefore it is possible to remove kernels that less affect the performance of the model. However, for algorithms such as involution, deciding which kernel to remove might be difficult. In this chapter we explain 2 features of our proposed method.

1. Kernel grouping
2. Model Diet

3.1 Kernel grouping

When we try to save the model of a CNN, we save the kernel weights as a dictionary. Every entry of the dictionary corresponds to each kernel, therefore summing and sorting the kernel weights was easy in CNN. However for involution kernels the weights are saved as a vector since the kernels act as a linear layer. Then the output of the kernel is then reshaped to a $K \times K$ kernel. The kernel size K can be an arbitrary number so we can not decide the kernel size by loading the saved weights. To overcome this problem we split kernels into N groups for each layer.

In order to use pre-trained models, the weights of the model need to be loaded and the weights of the saved model differ from their shape. For example, convolutional layers are saved as (output channels, input channels, K, K) where K stands for the kernel size, and linear projection layers are saved as (output features, input features). Since Involution kernels are generated from linear projection and a reshaping layer, the linear projection of the involution kernel is saved as (output features, input features). The saved model does not contain the reshaping layer since reshaping is an operation rather than a neural network layer.

The reshaping permutes the output of the linear projection into a convolution kernel, therefore the kernel size is unknown when the weights of the pre-trained model are loaded. Since the kernel size could be an arbitrary number, the entire model structure is needed to be known and read from the file in order to use the conventional filter pruning methods. To overcome the unknown kernel size problem we intended to keep the algorithm as simple as possible. Let N be the number of the weights. The weights are first split into g groups, where each group

G_i takes the index from $(N/g) \times i$ to $(N/g) \times (i+1)$ where i is a natural number smaller than g. We call this operation kernel grouping.

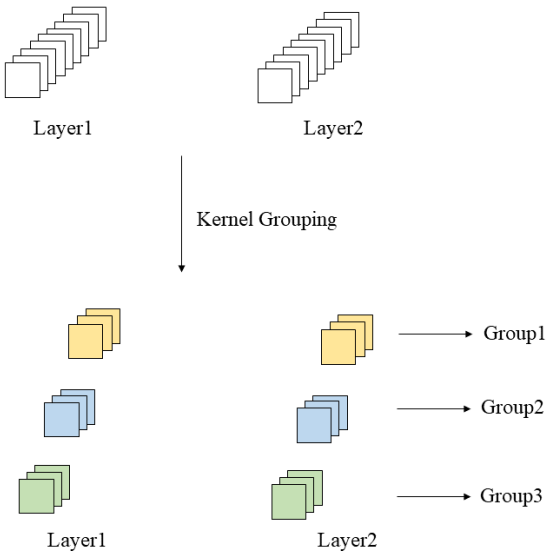

Figure 3.1: Kernel Grouping for convolution kernels

Fig.3.1 shows the kernel grouping for convolution kernels. The white kernels refer to the kernels for each layer. The figure on the bottom shows the convolution kernels after kernel grouping is applied. Different groups are colored in different colors. Fig.3.1 shows when we try to group kernels into 3 groups.

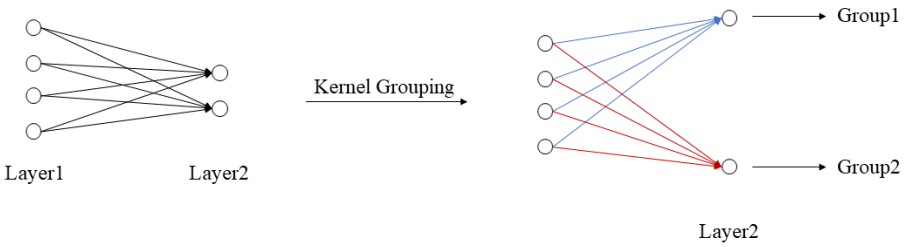

Figure 3.2: Kernel Grouping for fully connected layers

Fig.3.2 shows kernel grouping for vectors. In different words, weights for fully connected layers and involution kernels. The figure on the lefts shows the original model before kernel grouping is applied. The figure on the right shows the weights after kernel grouping is applied. Different groups are colored in differently.

3.2 Model diet

The term *Model diet* comes from reducing the weights in half while maintaining the model depth. The weights of each group are summed and the group that has the

biggest sum is used for the diet model. Since the bias and batch normalization with the convolution results must be matched, the sum of groups is needed instead of the sum of kernels. Our aim is to change the shape of the weight (output channel, input channel, K, K) into (output channel/g, input channel/g, K, K) therefore the weights from index $(N/g)\times i$ to index $(N/g)\times(i+1)$ where i indicates the index of the group which has the max sum of elements are kept. Any whole number g where g is a divisor of N/K^2 can be chosen. For example, if $g = 2$, half of the weights will be used. Fig. 3.3 depicts the graphical explanations of conventional pruning methods and the proposed model. The left vector corresponds to a weight vector of a linear projection. The right side indicates the involution kernel after the reshaping layer. Note that the darker color means the bigger value of the weight and the red cross means that the weight is pruned. In order to prune the correct filter, we need to add K^2 number of weights before sorting where K is the kernel size. However saved model files does not provide any information about the kernel size therefore deep technical details about the model are needed in order to apply pruning methods.

The green and brown indicates different kernels and K_i stands for the i-th kernel. If the results are different from the middle, the spatial information is lost. Conventional pruning sorts the weights before reshaping therefore the result differs from the actual kernel. Diet in contrast prunes the weights before reshaping therefore the results are equal to the actual kernel.

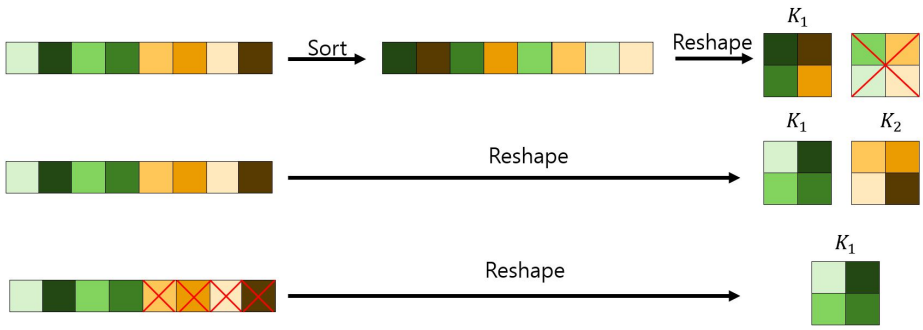

Figure 3.3: Conventional filter pruning (top), the actual involution kernel (middle), and the diet operation (bottom).

Fig.3.4 shows the visual explanation about the model diet algorithm. The black and white kernels refer to the original model i.e. the full model. Then we group the kernels into N groups. After grouping the kernels we get the total sum of the weights of kernels for each group. Finally we choose the group that has the biggest sum. We call this model the diet model.

Algorithm 2 shows the pseudo code of the diet operation. State dict is the dictionary where the model weights are saved. When the full model's state dict is loaded, the algorithm first checks the type of weight. Length 1 corresponds to the bias of a layer, length 2 corresponds to the weights to a fully connected layer and length 4 corresponds to the weights of a convolutional layer. The term half means the half of the shape with the corresponding dimension. As can be seen in

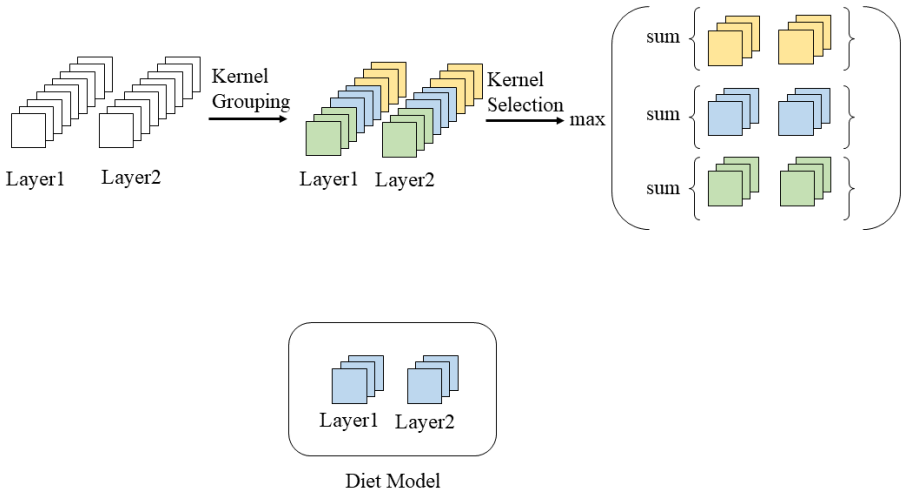

Figure 3.4: Visual explanation about model diet algorithm.

the algorithm, diet operation can be done regardless of the kernel size.

3.3 Computation Cost

Let K be the width and height of a kernel. Normally a convolution kernel has the same width and height therefore we let the number of weights per kernel is K^2. We define the depth of a model L and the number of kernels per layer as C.

The computation cost of conventional pruning methods are computed as below. Summing all the kernels need $K^2 \cdot C$ complexity. After the summing the weights of a kernel, we sort the kernels in order by their sum. The computation cost of sorting kernels depend on the number of channels. Therefore sorting requires $C \log C$ complexity. After sorting, we apply a certain threshold to the sum of weights which cost C complexity. In total, the computation complexity for a single layer is $C(K^2 + \log C + 1)$.

Model diet contains 2 components. Kernel grouping and group selection. The computation cost of kernel grouping is computed as follows. Since kernel grouping sums the kernel weights in order, the computation cost of kernel grouping is $K^2 \cdot C$. Unlike conventional pruning algorithms, sorting is omitted in the model diet algorithm. Group selection selects a group that has the biggest sum of kernel weights, resulting a G computational cost where G is the number of groups. The total amount of computation is $K^2 \cdot C + G$ for model diet.

Since the number of groups and kernel size is negligibly small compared to the number of channels, we can summarize the time complexity for conventional pruning algorithms as $\mathcal{O}(n \log n)$. Also $\mathcal{O}(n)$ for model diet.

Algorithm 2 Model diet pseudo code

```
Load Full model state dict
for each key in Full model do
    for i, g in enumerate(groups) do
        start=(N/g) * i
        end = (N/g) * (i + 1)
        if full[key].shape is not diet[key].shape then
            if len(full[key].shape) is 1 then
                g[key]=full[key][start:end]
                groupsum[g]+=sum(g[key])
            end if
            if len(full[key].shape) is 2 then
                g[key]=full[key][:, start:end]
                groupsum[g]+=sum(g[key])
            end if
            if len(full[key].shape) is 4 then
                g[key]=full[key][start:end,start:end,:,:]
                groupsum[g]+=sum(g[key])
            end if
        end if
    end for
end for
diet = group[argmax(groupsum)]
Save diet as state dict
```

Chapter 4

Experimental results

4.1 Image classification

We follow the work in [8] using RedNet model to test our method. RedNet is a successor model of ResNet [7] in image recognition but the convolution layers are replaced with Involution layers. The full model refers to the model without the diet algorithm while the diet model is the model after applying the diet algorithm. A randomly initialized model is a model that has the same architecture as the diet model but has random weights. First, we test the models using RedNet to show that diets are effective to Involution. Then we use different CNN models such as [5, 7] to show that the diet operation also works with conventional CNN models. Conventional CNN models follow the implementation of PyTorch. The test accuracy was measured with both newly trained diet model and knowledge distilled model.

4.1.1 Dataset

The Imagenette [1] dataset is a small subset of the ImageNet dataset [1] that is composed of 10 different class labels for image classification. It contains about 9000 images for training and 3000 images for testing. Imagenette is a useful dataset as it allows for faster processing since it has fewer images compared with ImageNet. Since benchmark datasets usually have millions of images, it might require more computational power and time to process.

We first fine tune the ImageNet pre-trained full model with Imagenette. Then we apply the diet operation to the full model reducing the number of weights and build a new diet model. We compare the change of top-1 classification accuracy during the training epochs between the diet models i.e. the model that inherits the pre-trained weights and the model that was randomly initialized.

4.1.2 Parameter setup

Images in the dataset are 3-channel RGB images. The values of the channels might be biased in a certain color, for example an image of a red flower will have

[1] https://github.com/fastai/imagenette

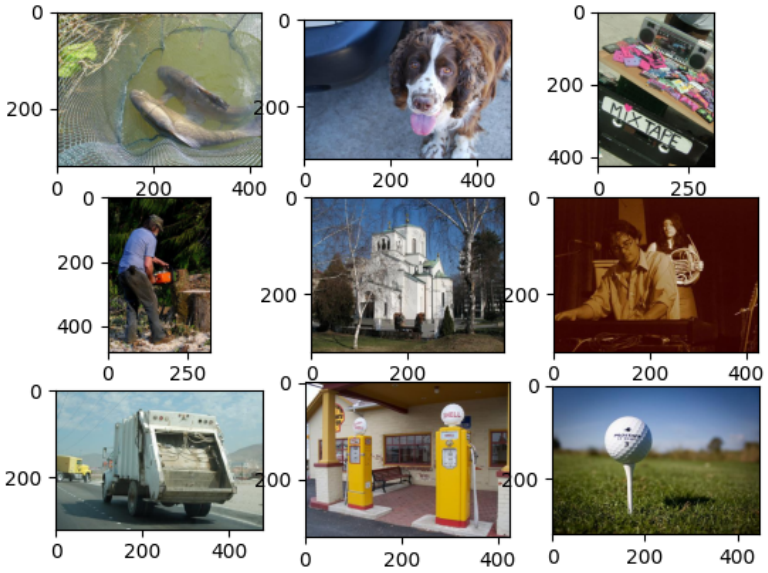

Figure 4.1: Sample images of the Imagenette dataset.

high pixel values for red channels, while an image of a blue sky might have large values for blue channels. This bias will affect the ability to capture the visual patterns therefore we apply normalization to each channel to remove the channel bias. All the input images are normalized with the mean (0.4914, 0.4822, 0.4465) and the standard deviation (0.247, 0.243, 0.261) indicating for each channel(red, green, blue in order) correspondingly. The mean and standard deviation follows the CIFAR10 dataset. We crop each image with random axes and flip the image with 0.5 probability to avoid overfitting.

We also used constant learning rate of $1e-5$ with batch size 32 using Adam optimization. The training epochs were same during training both the diet model and the randomly initialized model. For RedNet we use 700 epochs for training, and for ResNet and VGG we used 100 epochs for training.

Table 4.1: Size comparison for each model (in MB)

Model Size	RedNet26			ResNet18			VGG16		
	Full	Diet	Reduction Rate	Full	Diet	Reduction Rate	Full	Diet	Reduction Rate
Inference	492.54	287.04	41.72%	62.79	52.06	17.09%	322.13	161.07	50.00%
Paramaters	27.48	7.00	74.53%	42.65	11.21	73.72%	512.35	128.15	74.99%
Total	520.60	297.62	42.83%	106.01	63.84	39.78%	835.06	289.79	65.30%

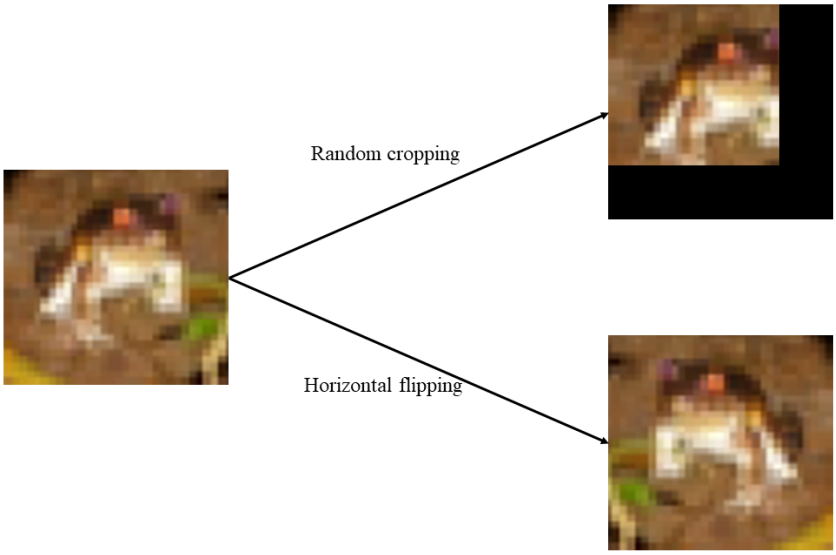

Figure 4.2: Visual explanation about data augmentation.

Table 4.2: Top 1 accuracy(%) of the diet model.

Model	Full	Diet	with KD	random initialized
RedNet26	93.9%	87.1%	89.7%	60.2%
ResNet18	97.4%	90.2%	90.8%	75.9%
VGG16	98.2%	90.2%	90.8%	79.6%

4.1.3 Computational Results and Analysis

Table 4.2 and Table 4.3 shows the number of parameters and the test accuracies before and after the diet algorithm. We could reduce the number of parameters more than half by simply halving the number of filters. The diet could reduce up to approximately 75% of the parameters. Table 4.1 and Fig. 4.5 shows the size in MB of each model and the test accuracy respectively. Although the accuracy was lowered about 7% compared to the full model, the results still remain in a reasonable area. The accuracy is lower than the previous work but its implementation is much easier as well as the computational cost needed. Model diet can

Table 4.3: Number of parameters ($\times 10^6$) before and after diet.

Model	Full	Diet	Reduction Rate
RedNet26	7.20	1.84	74.44%
ResNet18	11.18	2.94	73.70%
VGG16	134.31	33.59	74.99%

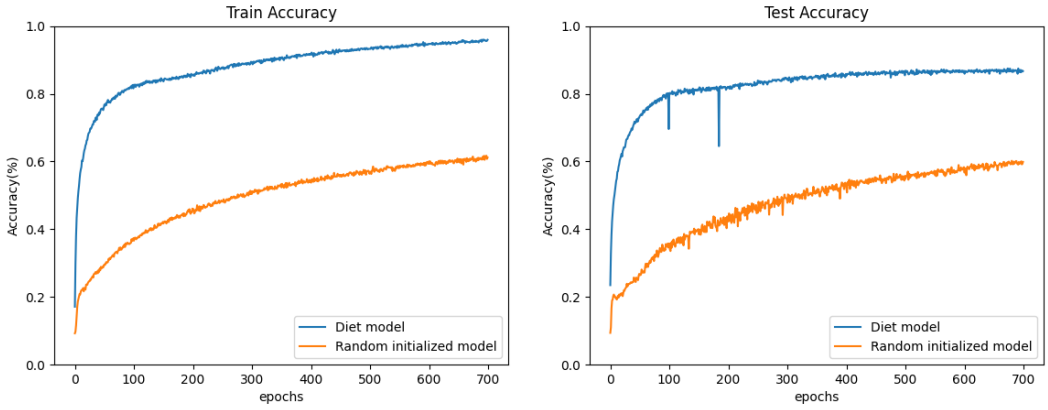

Figure 4.3: The train, test accuracy of random initialized and the diet model(Tested with Involution).

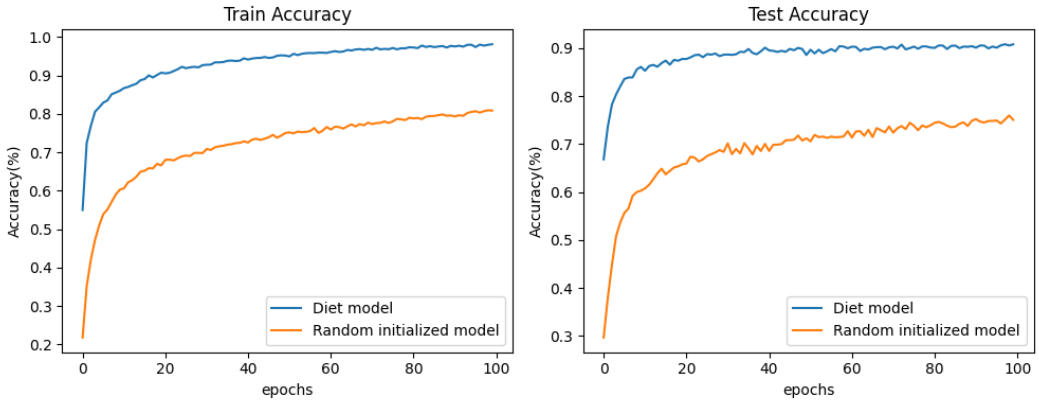

Figure 4.4: The train, test accuracy of random initialized and the diet model(Tested with ResNet).

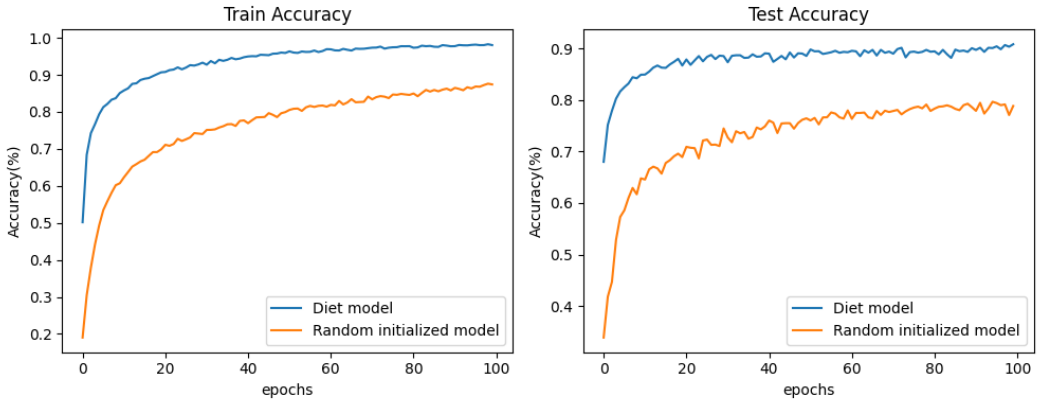

Figure 4.5: The train, test accuracy of random initialized and the diet model(Tested with VGG).

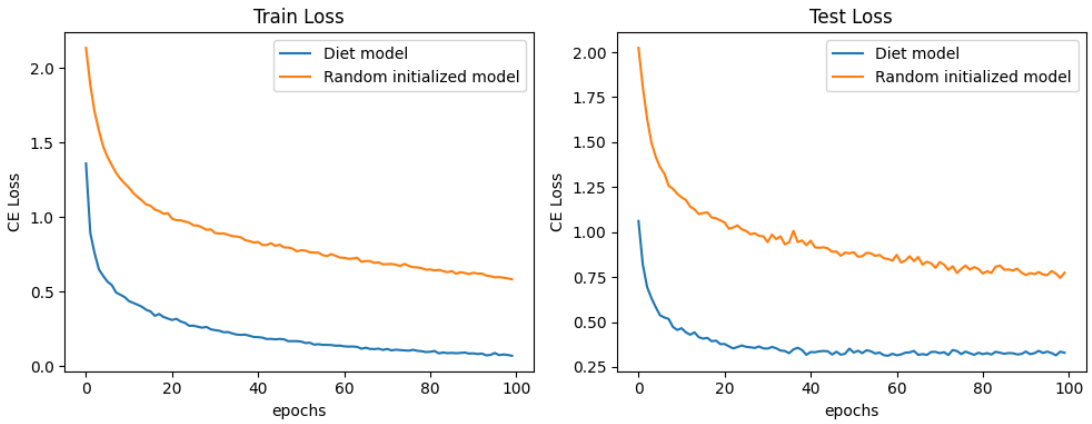

Figure 4.6: The train, test loss of random initialized and the diet model(Tested with ResNet).

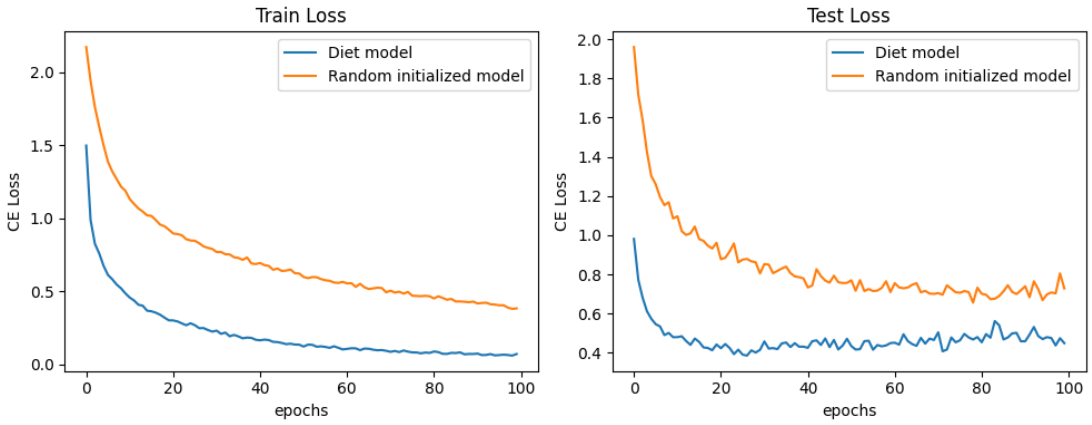

Figure 4.7: The train, test loss of random initialized and the diet model(Tested with VGG).

be operated by simply using if-then rules depending on the weight shape without requiring extra data manipulation operations such as adding or sorting.

Table 4.4: Number of GFLOPs before and after diet.

Model	Full	Diet	Reduction Rate
RedNet26	1.75	0.67	61.71%
ResNet18	1.83	0.92	49.72%
VGG16	15.54	3.93	74.71%

Table 4.4 shows the number of floating-point operations in a billion scale. It can be seen that the floating-point operations have reduced about 50-60%. This difference will make a drastic difference in the inference time when it is needed to operate in real time.

4.1.4 Comparison between groups

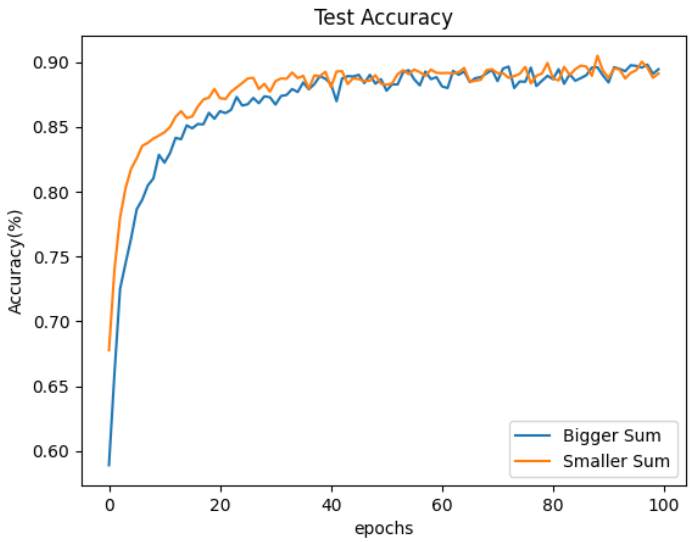

Figure 4.8: Comparison between the test accuracy(%) of VGG

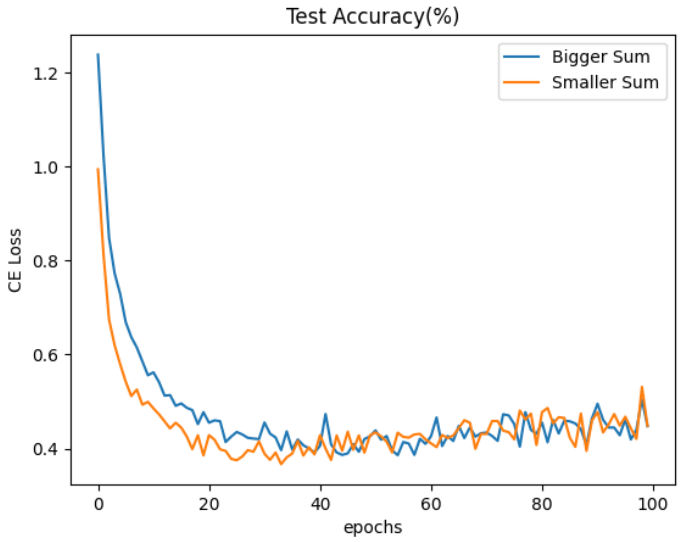

Figure 4.9: Comparison between the test loss(CE) of VGG

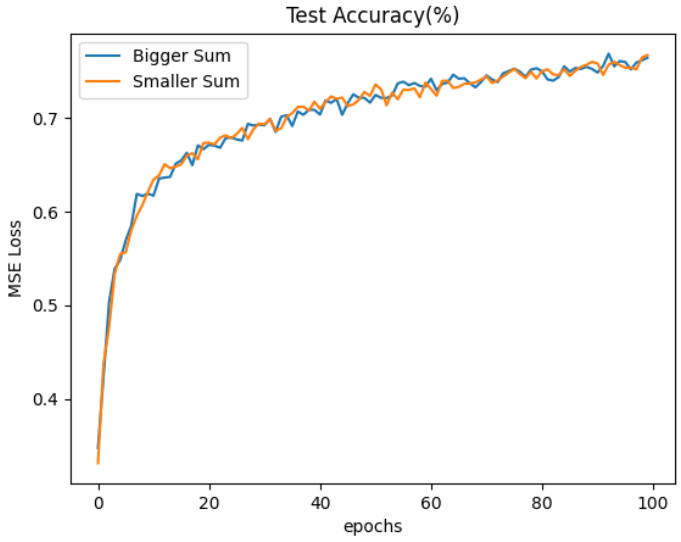

Figure 4.10: Comparison between the test accuracy(%) of ResNet

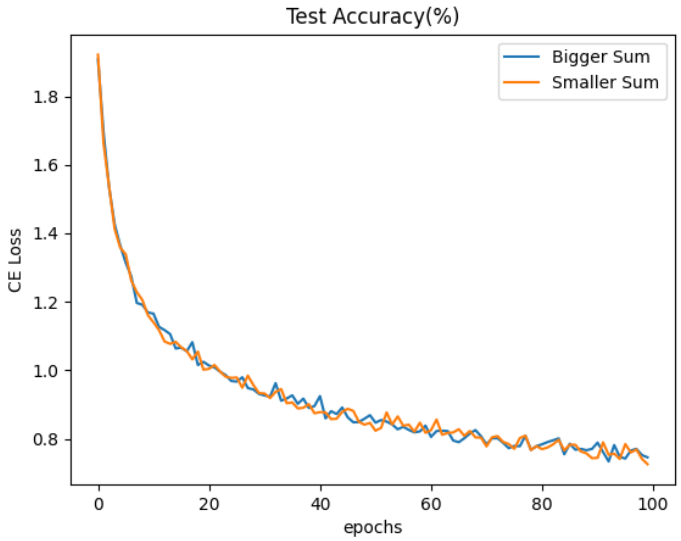

Figure 4.11: Comparison between the test loss(CE) of ResNet

Fig.4.8 and Fig.4.9 shows the test accuracy and test loss for the VGG model respectively. Fig.4.10 and Fig.4.11 shows the test accuracy and test loss for the ResNet model respectively. The blue plot indicates the diet model which trained with the bigger sum of groups, and the orange one indicates the diet model which used the smaller sum of groups. Either way the convergence and performance is similar. Since the model diet algorithm trains the diet model with training data, we can regard it as a weight initialization algorithm.

Model diet prunes the kernels of the full model which converged into a certain place on the feature space. This means that the optimization lead the weights into a local optimum. For the 2 models, the model which used the bigger sum and the one with the smaller sum, will start from a different location in the feature space. Although the 2 models have completely different weights, they share the common convergence point of the full model therefore we hypothesize that the 2 models also share the same convergence point. Hence the 2 models start with the different loss and accuracy, but as the weights are trained, the 2 models head to the same point. We can see that the 2 models have similar performance from the figures mentioned before.

4.2 Image segmentation

Figure 4.12: The input and ground truth for the carvana dataset

The image segmentation task is a task where we classify each pixel in an image. The segmentation model takes an image as an input and outputs a mask. each mask corresponds to a certain label. We test the model diet algorithm on the U-Net model[24]. Other experimental settings are same with image classification. The U-Net model was trained using the carvana image masking dataset. Fig.4.12 shows a sample input and ground truth image of the carvana image masking dataset.

Fig.4.13 shows the test loss of the diet model and the random initialized model of U-Net. We use the binary cross entropy loss(BCE Loss) which is a cross entropy loss when there is 2 classes. The diet model and the random initialized model both shows similar performance. However we can still observe that the diet model converge faster than the random initialized model. There was no difference on the performance between the diet model and the random initialized model since the dataset lacks difficulty. Therefore either models showed similar performance.

Fig.4.14 shows the input image, the ground truth binary mask, the prediction of the diet model and the prediction of the random initialized model. The prediction between the diet model and the random initialized model look almost same since the dataset is easy to distinguish the background and the car.

4.3 Depth Estimation

Depth estimation is close to image segmentation but the classification is changed to regression. We predict the depth for each given pixel in an image. The NYU depth dataset V2 was used for training. Also the same U-Net model used for

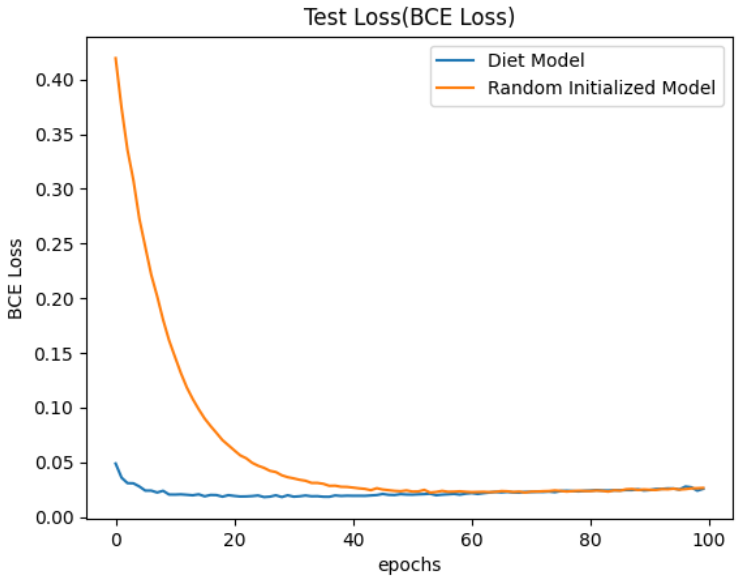

Figure 4.13: The test loss of random initialized and the diet model(Tested with U-Net)

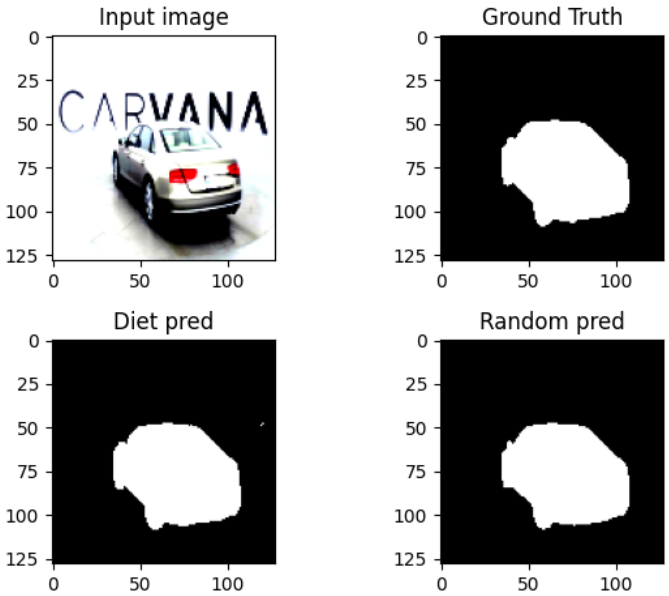

Figure 4.14: The input image(top left), the ground truth binary mask(top right), the prediction of the diet model(bottom left) and the prediction of the random initialized model(bottom right).

image segmentation was used for depth estimation. The training settings stays same with image classification.

Fig.4.15 shows the test loss(MSE loss) for the diet model and the random initialized model. Both diet model and random initialized model showed convergence around epoch 20. The performance of the diet model is slightly higher than the

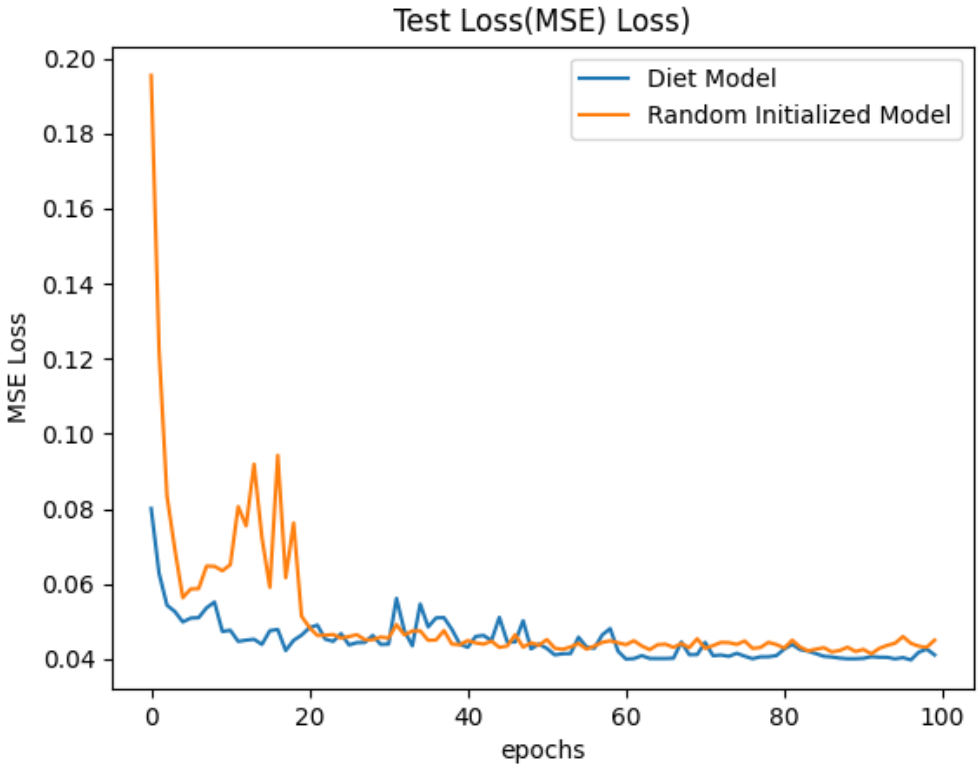

Figure 4.15: The test loss of random initialized and the diet model for depth estimation(Tested with U-Net)

random initialized model.

Fig.4.16 show the predictions for both models. Although the loss was reduced, the prediction of the U-Net model was poor. Since we used MSE loss for training, and MSE loss does not guarantee the local information the decrease of the MSE loss does not mean that the performance is getting better.

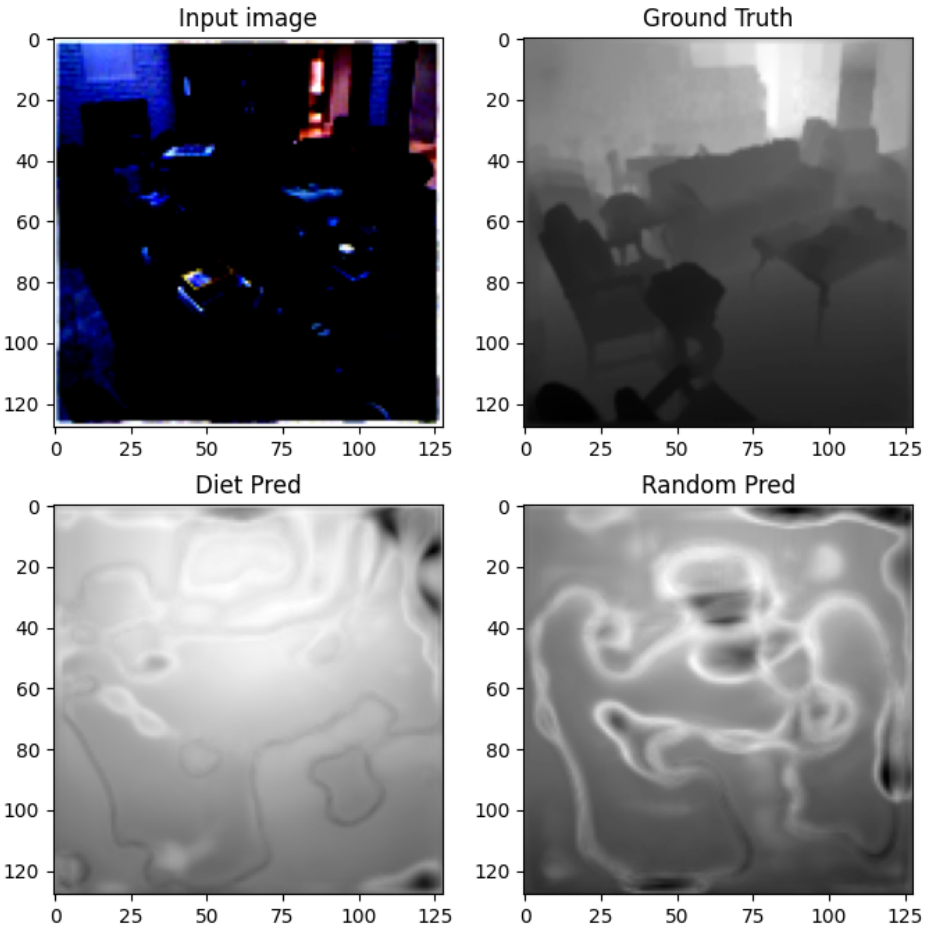

Figure 4.16: The input image(top left), the ground truth binary mask(top right), the prediction of the diet model(bottom left) and the prediction of the random initialized model(bottom right).

Chapter 5

Conclusion

In this research, we proposed a novel method for compressing the neural network models by reducing the weights used. Our experimental results provided that it can reduce up to 75% of the parameters without requiring much computational efforts. Also Involution kernels have reshaping layers so the kernel size remains unknown from weight files. Even if the kernel size is known, we still need extra manipulation such as summation and sorting. The proposed Model diet overcomes these limitations since it does neither need the kernel size nor require extra manipulation.

Diet models can learn faster compared to randomly initialized models and can be applied on conventional CNN models as well. We showed the results on both involution model and conventional models. The accuracy is decreased by approximately 7% but still remains on a reasonable area. Since diet RedNet26 only have 1.84M parameters, model diet can be applied on devices that require less parameters such as mobile devices or mobile robots.

We tested the performance and convergence on 3 different tasks. For image classification the diet model showed better performance and faster convergence compared to the random initialized model. For image segmentation, both diet model and the random initialized model showed similar performance. However the diet model still had faster convergence compared to the random initialized model. For depth estimation we used the same model architecture with image segmentation. The performance was poor even though the loss converged. The summation of groups did not affect much on the performance. We hypothesize that both groups with the bigger sum and the smaller sum exists in the same local optimum.

Model diet has its limitations on the program design since it becomes complicated to implement when the model architecture does not consist of basic blocks. Pytorch implements ResNet by using basic block layers and RedNet implementation follows the work of Pytorch. Although the diet algorithm can still be applied to those models, its implementation may not be as simple as done by if-then rules. Also, more sophisticated methods can be developed on how to choose weights that would be kept in the reduced model.

For future work, some experimental and performance analysis will be targeted in other models for object detection, image generation and similar other tasks.

Also, more sophisticated ways to select a group of weights apart from the summation of weights and the effect of the sum of the groups will be studied. To check if the weights are located into a similar local optimum, similarity functions such as cosine similarity will be used to compare the values of the model's weights. If the 2 models have big similarity we can say that 2 models converged into a same local optimum. Also if the models share the same convergence point, it means that we can select either group therefore the algorithm can be simplified.

We failed to train the U-Net model for depth estimation. To check whether model diet works on depth estimation, we must try with state of the art depth estimation models or other training methods. In this research monocular depth estimation was tested with U-Net with MSE loss trained with the Adam optimizer. For future work, other state of the art models and training methods must be tested.

The 2 models trained for image segmentation has similar performance. This happened due to the lack of difficulty of the dataset. To test the effectiveness of the model diet the U-Net model must be tested on harder datasets such as the COCO image segmentation dataset.

Bibliography

[1] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[3] Hieu Pham et al. "Meta Pseudo Labels". In: *CoRR* abs/2003.10580 (2020). arXiv: 2003.10580. URL: https://arxiv.org/abs/2003.10580.

[4] Andrew Brock et al. "High-Performance Large-Scale Image Recognition Without Normalization". In: *CoRR* abs/2102.06171 (2021). arXiv: 2102.06171. URL: https://arxiv.org/abs/2102.06171.

[5] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[6] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV].

[7] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[8] Duo Li et al. *Involution: Inverting the Inherence of Convolution for Visual Recognition*. 2021. arXiv: 2103.06255 [cs.CV].

[9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML].

[10] Hao Li et al. *Pruning Filters for Efficient ConvNets*. 2017. arXiv: 1608.08710 [cs.CV].

[11] Jonathan Frankle and Michael Carbin. *The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks*. 2019. arXiv: 1803.03635 [cs.LG].

[12] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020. arXiv: 2010.11929 [cs.CV].

[13] Ze Liu et al. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. arXiv: 2103.14030 [cs.CV].

[14] Kelvin Xu et al. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*. 2016. arXiv: 1502.03044 [cs.LG].

[15] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV].

[16] Xiangyu Zhang et al. *ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices*. 2017. arXiv: 1707.01083 [cs.CV].

[17] Kai Han et al. *GhostNet: More Features from Cheap Operations*. 2020. arXiv: 1911.11907 [cs.CV].

[18] Mingxing Tan and Quoc V. Le. "MixConv: Mixed Depthwise Convolutional Kernels". In: *CoRR* abs/1907.09595 (2019). arXiv: 1907.09595. URL: http://arxiv.org/abs/1907.09595.

[19] Ofir Zafrir et al. "Prune Once for All: Sparse Pre-Trained Language Models". In: *CoRR* abs/2111.05754 (2021). arXiv: 2111.05754. URL: https://arxiv.org/abs/2111.05754.

[20] Song Han et al. "Learning both Weights and Connections for Efficient Neural Networks". In: *CoRR* abs/1506.02626 (2015). arXiv: 1506.02626. URL: http://arxiv.org/abs/1506.02626.

[21] Alex Renda, Jonathan Frankle, and Michael Carbin. "Comparing Rewinding and Fine-tuning in Neural Network Pruning". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=S1gSj0NKvB.

[22] Raphael Gontijo Lopes, Stefano Fenu, and Thad Starner. "Data-Free Knowledge Distillation for Deep Neural Networks". In: *CoRR* abs/1710.07535 (2017). arXiv: 1710.07535. URL: http://arxiv.org/abs/1710.07535.

[23] Gongfan Fang et al. "Data-Free Adversarial Distillation". In: *CoRR* abs/1912.11006 (2019). arXiv: 1912.11006. URL: http://arxiv.org/abs/1912.11006.

[24] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.