PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

SUPERMASKS AND A GOOD INITIALIZATION ARE ALL YOU NEED

FRANCISCO RENCORET DOMÍNGUEZ

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Advisor:
ÁLVARO SOTO

Santiago de Chile, December 2020

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

SUPERMASKS AND A GOOD INITIALIZATION ARE ALL YOU NEED

FRANCISCO RENCORET DOMÍNGUEZ

Members of the Committee:
ÁLVARO SOTO
HANS LÖBEL
PABLO ZEGERS
IGNACIO VARGAS

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Santiago de Chile, December 2020

Gratefully to my parents and friends

ACKNOWLEDGEMENTS

To my advisor Álvaro Soto, who guided me throughout this investigation with a clear vision for finding value.

To Julio Hurtado, a key fellow that helped me on a daily basis during the ideation and implementation of this work.

To my office mates from IALab PUC. Alain Raymond, Cristóbal Eyzaguirre, Felipe del Rio, and Raimundo Manterola, who contributed to the realization of this thesis with their regular feedback.

To my parents, who have always supported me to focus and fulfill my dreams.

To my girlfriend Elisa, for encouraging and supporting me every day.

To God, for the gifts, the blessings, and for guiding my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Deep Learning models have shown significant improvements in computer vision tasks, although generally relying on optimizing highly parameterized networks. To overcome this, the Lottery Ticket Hypothesis (Frankle & Carbin, 2019) states that a dense neuronal network contains a subnetwork such that - when trained in isolation - it can match the test accuracy of the original full network. Supermask training (Zhou et al., 2019) is an efficient way of obtaining a Lottery Ticket, but unfortunately, it still faces performance issues. Under Supermask training, the value of the initial weights is key because they are never updated. We hypothesize that by adding prior knowledge from the data to the weight initialization, Supermask training would find a subnetwork with better test performance than random initialization.

In this thesis, we propose a novel method to initialize the weights of a model under Supermask training. We refer to the proposed method as *PatchesInit*. The method initializes the weights with patterns found in the training data to approximate what they should learn on a regular training scheme. To evaluate *PatchesInit*, we train several ConvNets, with different Supermasks configurations over CIFAR-10, CIFAR-100, and TinyImagenet-200 datasets. Results show that *PatchesInit* is an effective weight initialization strategy, showing significant improvements over random initialization. For shallow ConvNets, the proposed method outperforms the baseline under different levels of weight pruning. On the other hand, *PatchesInit* faces problems to initialize weights effectively for deeper networks, but we further propose a variant that does find subnetworks with better performance than random initialization.

Keywords: Deep Learning, Convolutional Neural Networks, Lottery Tickets, Weight Pruning, Supermasks, Weight Initialization.

RESUMEN

Los modelos de aprendizaje profundo han mostrado significativas mejoras en las tareas de visión por computador, aunque generalmente optimizando redes neuronales altamente parametrizadas. Para mejorar esto, la hipótesis de *Lottery Ticket* (Frankle & Carbin, 2019) establece que una red neuronal densa contiene una subred de modo que, cuando se entrena de forma aislada, puede igualar el rendimiento de la red completa original. El entrenamiento de *Supermask* (Zhou et al., 2019) es una forma eficiente de obtener un *Lottery Ticket*, pero desafortunadamente, aún enfrenta problemas de rendimiento. En el entrenamiento de *Supermask*, el valor de los pesos iniciales es clave ya que nunca se actualizan. Nuestra hipótesis es que, al agregar conocimiento previo de los datos a la inicialización de los pesos, el entrenamiento de *Supermask* encontraría una subred con mejor rendimiento en los datos de prueba que la inicialización aleatoria.

En esta tesis, proponemos un método novedoso para inicializar los pesos de un modelo bajo el entrenamiento de *Supermask*. Nos referimos al método propuesto como *PatchesInit*. El método inicializa los pesos con patrones encontrados en los datos de entrenamiento, aproximándose así a lo que deberían aprender en un esquema de entrenamiento regular. Para evaluar *PatchesInit*, entrenamos varias *ConvNets*, con diferentes configuraciones de *Supermask*, sobre los conjuntos de datos CIFAR-10, CIFAR-100 y TinyImagenet-200. Los resultados muestran que *PatchesInit* es una estrategia de inicialización eficaz, mejorando significativamente el rendimiento de la inicialización aleatoria. Para *ConvNets* de poca profundidad, el método propuesto supera a la inicialización aleatoria bajo diferentes niveles de *weight pruning*. Por otro lado, *PatchesInit* enfrenta problemas para inicializar los pesos de manera efectiva para redes más profundas, por lo que proponemos una variante que sí encuentra subredes con mejor rendimiento que la inicialización aleatoria.

Palabras Claves: Aprendizaje profundo, Redes Convolucionales, *Lottery Tickets*, *Weight Pruning*, *Supermask*, Inicialización de pesos.

1. INTRODUCTION

1.1. Motivation

Machine Learning is the study of algorithms that progressively learn through experience. The algorithms learn mathematical models from a set of data points to make predictions without being explicitly programmed to do so.

According to the learning experience, there are several types of Machine Learning approaches (Haldorai & Ramu, 2019). Among them, supervised, unsupervised, and reinforcement learning are some of the most popular. Supervised learning assumes data points to have explicit labels or outputs, learning a mapping function from input to output. Conversely, unsupervised learning does not rely on the labels, forcing the model to find relevant structures from the data. Lastly, reinforcement learning is based on a model that interacts with a dynamic environment and takes actions trying to achieve a particular goal. Each action produces a reward within a specific environment state that the model utilizes as feedback and tries to maximize. Throughout this thesis, we focus on supervised learning, particularly on the task of image classification, where the model is trained to classify an image from a fixed set of categories.

Traditionally, Machine Learning techniques were limited in the ability to process data in their raw form. For decades, implementing these algorithms required careful engineering and domain expertise to extract suitable representations or features from the raw data (frequently called *feature-engineering*). From these features, the algorithm could detect patterns and make predictions (LeCun et al., 2015). For example, in the context of computer vision, Machine Learning algorithms required feature extractors such as SIFT (Lowe, 1999), LBP (Ojala et al., 1996), or HOG (Dalal & Triggs, 2005).

Trying to remove this bottleneck, representation learning techniques come into play with the objective of building models that could create their internal representations from raw input data (Bengio et al., 2013; Zhong et al., 2016). Deep Learning methods were

then developed, as the concept of having multiple levels of representation or sequential learning modules. Each module transforms its representation into a representation at a higher, slightly more abstract level. Complex functions can then be modeled by stacking sufficient modules.

Deep Learning algorithms are parametrized by a set of weights, where learning is encoded. The standard paradigm for training deep learning models is as follows: the weights of the model are randomly initialized, and then the model processes and predicts iteratively over a set of raw training data. An objective function determines the error rate of each prediction, which is used to update the weights of the model by the so-called backpropagation algorithm (Rumelhart et al., 1988). When a termination criterion is achieved (e.g., a target error or convergence point), the training is stopped, and the model is used to predict unseen test data.

In the past few years, Deep Learning models have shown significant improvements in computer vision tasks, but relying on optimizing highly parameterized networks. To overcome this, The Lottery Ticket Hypothesis (Frankle & Carbin, 2019) states that a dense neuronal network contains a subnetwork (a winning ticket) such that - when trained in isolation - it can match the test accuracy of the original full network. In other words, by identifying a winning ticket, we can reduce the model size and make faster predictions without negatively impacting performance. Finding a winning ticket is an open research area and still a computationally expensive procedure because the network needs to be trained to completion several times.

Zhou et al. (2019) introduce Supermask training as a novel and efficient way to identify a winning ticket. The model weights are initialized, frozen, and never trained. Instead, they optimize a binary mask to select a subnetwork that contains those weights that won the lottery. The Supermask procedure finds effective subnetworks reducing the computational cost, but still fall short in performance from the original Lottery Ticket procedure.

Under Supermask procedure, the value of the initial weights is key because they are never updated; the model needs to create a useful internal representation by merely deciding to select or discard each weight. Even though Supermask training finds effective subnetworks within initial random weights, we hypothesize that by adding prior knowledge from the data to the weight initialization, Supermask training would find a more suitable subnetwork.

This thesis proposes a novel method to initialize the weights of a model under Supermask training. We refer to this method as *PatchesInit*. The intuition behind *PatchesInit* is to initialize weights with an approximation of what they should learn during regular training, providing a better starting point (closer to where it should end). In image classification, Convolutional Neural Networks learn internal representations of images by learning to detect patterns. *PatchesInit* tries to mimic this by obtaining adequate patterns from the training data and using them to initialize filter weights, providing prior knowledge that should increase performance on unseen test data.

The Lottery Ticket Hypothesis and the Supermasks seek to make Machine Learning models more efficient, finding subnetworks that predict faster and reduce hardware requirements. This thesis pursues this mission, trying to find even more accurate models by changing the weight initialization.

1.2. Hypothesis

This thesis hypothesizes that initializing the weights of a model with patterns found in the training data will increase the generalization test performance of the subnetworks detected under Supermask training.

Regarding the proposed hypothesis, this thesis covers three specific objectives:

(i) Develop *PatchesInit*: a weight initialization method from patterns present in training data.

(ii) Evaluate the performance of *PatchesInit* on different model configurations and compare it with other initialization methods.

(iii) Analyze the effect of using *PatchesInit* under Supermask training.

The organization of this document is as follows. Section 2 covers related work, diving into details of Convolutional Neural Networks, weight initialization methods, and the Lottery Ticket Hypothesis. Section 3 explains background theory and relevant methods we use throughout this thesis. Section 4 focuses on the first objective by covering the *PatchesInit* method, explaining from the intuition to the specific details. Section 5 focuses on the second and third objectives, diving into the experiments where we evaluate and analyze the impact of *PatchesInit*. Section 6 recaps the main contributions and lessons learnt from this thesis. Finally Section 7 states future research avenues.

2. RELATED WORK

2.1. Convolutional Neural Networks

Convolutional Neural Networks (ConvNets) are a specific type of feed-forward network designed to process data in the form of multidimensional arrays or tensors; as an example, a color image composed of three 2D arrays containing color pixel intensities. The nature of image data presents two distinctive characteristics. First, local groups of values are usually highly correlated, forming distinctive patterns. Second, local statistics of images and other signals are invariant to location. If a pattern appears in one part of the image, it could appear anywhere.

ConvNets have four key concepts that take advantage of the nature of this type of data: local-connections, shared weights, pooling, and multiple layers (LeCun et al., 2015). Units in a convolutional layer are organized in feature maps, where each unit is connected with their corresponding local group of units from the previous layer by a group of shared weights called filters. Filters specialize in finding one specific pattern from a group of locally-connected units from the previous feature map (or input data).

After feature maps are obtained, pooling layers enable semantically similar features to merge, enabling the model to recognize patterns even after small shifts and distortions. Pooling often takes the maximum value of a local patch of units in a feature map, combining them into one value - reducing the dimensionality of the feature maps.

On the other hand, stacking multiple convolutional and pooling layers exploits the property of compositionality in images, where high-level features are obtained through a composition of low-level features. The first layers of a ConvNet identify low-level features such as lines or edges. As the hierarchy advances, low-level features combine into high-level features such as objects or faces (Krizhevsky et al., 2012; LeCun et al., 2015; H. Lee et al., 2009). After the object representations are formed in the last layers, classifiers use those features to predict and separate among classes.

ConvNets have been applied to image classification tasks (LeCun et al., 1989; C.-Y. Lee et al., 2015). Rather shallow ConvNets are generally used for small scale datasets. Deeper architectures, such as AlexNet (Krizhevsky et al., 2012), VGG (Simonyan & Zisserman, 2014), and ResNet (He et al., 2016), are used for large scale datasets. Additional complex components have been included, such as Batch Normalization (Ioffe & Szegedy, 2015) and Residual Connections (He et al., 2016), to effectively train these deep architectures. We use ConvNets from two to six layers for small scale datasets and a ResNet for a large scale dataset.

2.2. Weight Initialization

Neural Networks require an initialization scheme to set the initial weight values before training, which are key for an effective training optimization. An effective initialization of the weights must maximize the stability of the gradient descent algorithm while providing a good starting point, both explained in the following section.

Stability for the gradient descent algorithm is a non-trivial matter that may be impacted by several factors. Throughout this thesis, we focus on the impact that the initial weights have on stability. If the initial weights are not set properly, the algorithm can easily be affected by vanishing or exploding gradients (Pascanu et al., 2013) producing sub-optimal models or even numerical instability.

Under the backpropagation scope, each weight receives an update proportional to the partial derivative of the error w.r.t that weight (Rumelhart et al., 1988). Due to the compounded nature of the chain rule, the partial derivative may sometimes vanish or explode, especially in deeper architectures. If the gradient is too small (particularly smaller than 1), it will decrease exponentially with the sequence of layers, and shallower layers will not update as they should. The opposite happens if the gradient is too big, it will increase

exponentially, and shallower layers would take huge steps breaking stability in the optimization landscape (Bengio et al., 1994). Initial weights must then be small enough to avoid exploding, but big enough to avoid vanishing.

To solve the problems mentioned recently, Glorot and Bengio (2010) introduce the Normalized or Xavier initialization, which maintains the variance of the activation and gradients at a constant value. The weights of each layer are sampled from a 'normalized' distribution, considering the configuration of the previous and next layers; generating values that should neither explode nor vanish for a standardized input. Further investigations (He et al., 2015) propose the Kaiming initialization, a variant of Xavier initialization that samples weights from a different normalized distribution, being more robust to deep architectures with ReLU activations. We use Xavier initialization intuition throughout this thesis (Section 4.2.3) because it provides a more stable training behavior on the rather shallow models we use. Xavier initialization, which we call $Random$, acts as a baseline for our experiments.

Regarding the concept that ideally, initialization schemes should provide a good starting point, advances have been made through pre-training. Pre-training moves the initial weights towards more suitable values (better starting point), which are then used as the initial weights for training the original task. Unsupervised pre-training methods enable weights to adopt knowledge of unlabeled data (Erhan et al., 2010; Frankle et al., 2020). Supervised pre-training methods, such as fine-tuning, provide significant performance advances reusing training from similar tasks than the original. Under the context of Meta-Learning, MAML (Finn et al., 2017) learns representations that learn new tasks quickly, by pre-training the weights to adapt and learn new tasks simultaneously.

The proposed method does not compare with the methods just mentioned because they require an initial pre-training stage optimizing a set of initial weights. Instead, we approximate that knowledge by initializing the weights with patterns found in the training data, avoiding the optimization process and its associated costs.

2.3. The Lottery Ticket Hypothesis

Frankle and Carbin (2019) propose the Lottery Ticket Hypothesis: a dense, randomly-initialized neural network contains a subnetwork (a winning ticket) that is initialized such that — when trained in isolation — it can match the test accuracy of the original network after training for at most the same number of iterations. They show it is possible to find much smaller subnetworks than the original network (even up to 1%), which has impressive reductions in prediction time during inference and storage size of the resulting weights. Also, recent advances preliminarily show that winning tickets have transferable generalization capabilities to other domains (Morcos et al., 2019; Desai et al., 2019).

Unfortunately, finding a winning ticket relies on iterative magnitude pruning, a costly procedure that requires training the network to completion several times. Some research works try to solve this problem. You et al. (2019) state that winning tickets can be identified at an early training stage, while others try to find these with auxiliary tasks (Savarese et al., 2019), self-supervision tasks (Caron et al., 2020), or obtaining stochastic masks with targeted dropout (Gomez et al., 2019).

Follow up work by Zhou et al. (2019) demonstrates that winning tickets achieve better than random performance with their initial random untrained weights. Motivated by this result, they propose Supermask training, a novel way of identifying a winning ticket. The weights of the model are frozen at their initial value, and a stochastic binary mask - with trainable parameters - is trained to select a subnetwork that contains a winning ticket. Supermasks find winning tickets by only training the mask once, a much more efficient method than iterative magnitude pruning.

Ramanujan et al. (2020) further extend this notion into the *edge-popup* algorithm: a deterministic Supermask training method that finds subnetworks with competitive performance compared to the original models (training the weights). The *edge-popup* algorithm finds subnetworks that are not as small as an original Lottery Ticket, but still take advantage reducing prediction time during inference and storage size.

We use this Supermask algorithm for this thesis, where we propose a novel weight initialization so that the algorithm optimizes a mask to select a winning ticket from a more suited set of initial weights. We hypothesize that by adding prior knowledge from the data to the weight initialization, Supermask training would find a subnetwork with better test performance than random initialization.

3. BACKGROUND THEORY

This section provides useful information about relevant methods used in this thesis.

3.1. Xavier Initialization

As mentioned in Section 2.2, the weight initialization scheme of neural networks is a relevant aspect that profoundly affects the convergence of the training procedure. To avoid vanishing or exploding gradients, Glorot and Bengio (2010) propose a method that maintains the variances of the activations and the backpropagated gradients at a constant value as one moves up or down through the network.

They introduce the Normalized or Xavier initialization, where they sample weights in each layer from 'normalized' distribution:

$$W_n \sim \mathcal{N}[0, \frac{\sqrt{2}}{\sqrt{n_{n-1} + n_{n+1}}}], \tag{3.1}$$

where W_n are the weights of layer n; n_{n-1} and n_{n+1} corresponds to the size (number of neurons) of the previous and next layer, respectively.

We use this normalization factor throughout Section 4.2.3 for re-scaling purposes.

3.2. The *edge-popup* algorithm

The *edge-popup* algorithm (Ramanujan et al., 2020) is a deterministic optimization method for finding effective subnetworks within randomly weighted neural networks. The main intuition relies on optimizing a binary mask that learns which weights should be selected as part of the winning ticket and those to be turned off.

For each weight w in the neural network, we include another learnable parameter s which we call popup score. The subnetwork is then chosen by selecting the weights w corresponding to the *top-k%* highest scores s in each layer. The parameter k is calculated

as $k = 1 - prune\_percentage$, where $prune\_percentage$ is a hyperparameter for the model.

A fully connected neural network consists of layers $1, ..., L$ where layer l has n_l nodes $V^{(l)} = \{v_1^{(l)}, ..., v_{n_l}^{(l)}\}$. For any node v, we let I_v denote the input and Z_v denote the output, where $Z_v = \sigma(I_v)$ for some non-linear activation function σ. We write I_v as a weighted sum of all neurons in the preceding layer:

$$I_v = \sum_{u \in V^{(l-1)}} w_{uv} Z_u, \tag{3.2}$$

where w_{uv} are the network parameters from layer l. Under Supermask training, the activation I_v can be rewritten as:

$$I_v = \sum_{u \in V^{(l-1)}} w_{uv} Z_u h(s_{uv}), \tag{3.3}$$

where $h(s_{uv}) = 1$ if s_{uv} is among the *top-k%* highest scores of layer l and $h(s_{uv}) = 0$ otherwise.

The extension to Convolutional Neural Networks is direct. Instead of having neurons, we now have filters. w_{uv}, s_{uv}, I_v, and Z_v are now two dimensional and $w_{uv}^{(k1,k2)}$, $s_{uv}^{(k1,k2)}$, $I_v^{(w,h)}$, and $Z_v^{(w,h)}$ are scalars, where $w \in \{1, ...W\}$ and $h \in \{1, ...H\}$ represents the width and height of the activation map from the node v, respectively. k is the kernel size. $I_v^{(w,h)}$ can be written as:

$$I_v^{(w,h)} = \sum_{u \in V^{(l-1)}} \sum_{k_1=1}^{k} \sum_{k_2=1}^{k} w_{uv}^{(k1,k2)} Z_u^{(w+k_1,h+k_2)} h(s_{uv}^{(k1,k2)}), \tag{3.4}$$

where $h(s_{uv}^{(k1,k2)}) = 1$ if $s_{uv}^{(k1,k2)}$ is among the *top-k%* highest scores of layer l and $h(s_{uv}^{(k1,k2)}) = 0$ otherwise.

Instead of optimizing the weights w, the Supermask freezes them at their initial value and optimizes the scores s. Considering that the function h is not differentiable, one could backpropagate through the hard threshold function by multiplying the gradient of each weight by the result of function h, as Zhou et al. (2019) suggest in their paper.

This method would suffer from the cold start problem, where if a score s_{uv} is initially low, w_{uv} would not be considered in the forward pass and the score s_{uv} would not be updated neither have the chance to popup.

To solve this, the Supermask optimizes with a straight-through gradient estimator (Bengio et al., 2013). Regardless of the forward pass, the gradient is backpropagated as if function h were the identity matrix (i.e., as if h had produced 1 on all weights).

Overcoming the cold start, if Z_u is aligned with the negative gradient (regardless of not being considered in the forward pass), then score s_{uv} will be updated accordingly. If this alignment happens consistently, then the score will continue to increase, and the edge will re-enter the chosen subnetwork (i.e., popup).

The *edge-popup* algorithm is deterministic, making it suitable for posterior analysis. On the other hand, it provides an extra cost of $O(L * S)$ per batch, where L is the number of layers in the network, and S is the cost associated with sorting and selecting the *top-k%* scores.

We use this Supermask method throughout this thesis, trying to make improvements on the matrix W with the initial weights.

3.3. Entropy

In the context of Information Theory (Cover & Thomas, 1991), entropy is a measure of disorder. Claude Shannon introduces entropy as a quantity associated to any random variable, which measures its average level of information or uncertainty (Shannon, 1948).

Entropy is frequently used in Machine Learning, especially in Decision Trees and Random Forests in the context of Information Gain (Breiman, 2001). Similar to Random Forests, we use the following formula to calculate the entropy of a set S of discrete data points:

$$E(S) = \sum_{i=1}^{N} -p_i \log_2 p_i, \tag{3.5}$$

where S contains elements from N different classes; and p_i is the probability of a class i. We use entropy in Section 4.2.2 to measure the level of disorder of a set of discrete labels.

4. PROPOSED METHOD

In this thesis, we propose *PatchesInit*, a novel method to initialize the weights of a Convolutional Neural Network under Supermask training. This section presents the proposed method, where Section 4.1 covers the intuition; Section 4.2 details the procedure; and Section 4.3 shows an outline of the algorithm.

4.1. Intuition

For image classification tasks, the Supermask procedure finds subnetworks that predict faster and reduce hardware requirements but still fall short in performance compared with the full networks (Frankle & Carbin, 2019; Zhou et al., 2019; Ramanujan et al., 2020). We focus on reducing the performance gap of these subnetworks by changing the initial set of weights of the neural network.

The Supermask procedure optimizes a binary mask to select a subnetwork within the randomly-initialized untrained weights of a network. Even though Supermasks find effective subnetworks using random initial weights, we believe that by adding prior knowledge from the data to the weight initialization, Supermask training would find subnetworks with better performance on unseen test data. In this thesis, we propose a method to initialize the weights of a ConvNet with prior knowledge from the data.

4.2. Method

ConvNets learn internal representations of images by learning to detect patterns (features), where each filter specializes in detecting one specific pattern. Low-level layers learn to identify low-level features, which are then composed, enabling high-level layers to identify high-level features (Krizhevsky et al., 2012; LeCun et al., 2015; H. Lee et al., 2009). For example, in image classification, a ConvNet may learn to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer,

and then use these shapes to detect higher-level features, such as facial shapes in higher layers. Figure 4.1 shows an example of this.

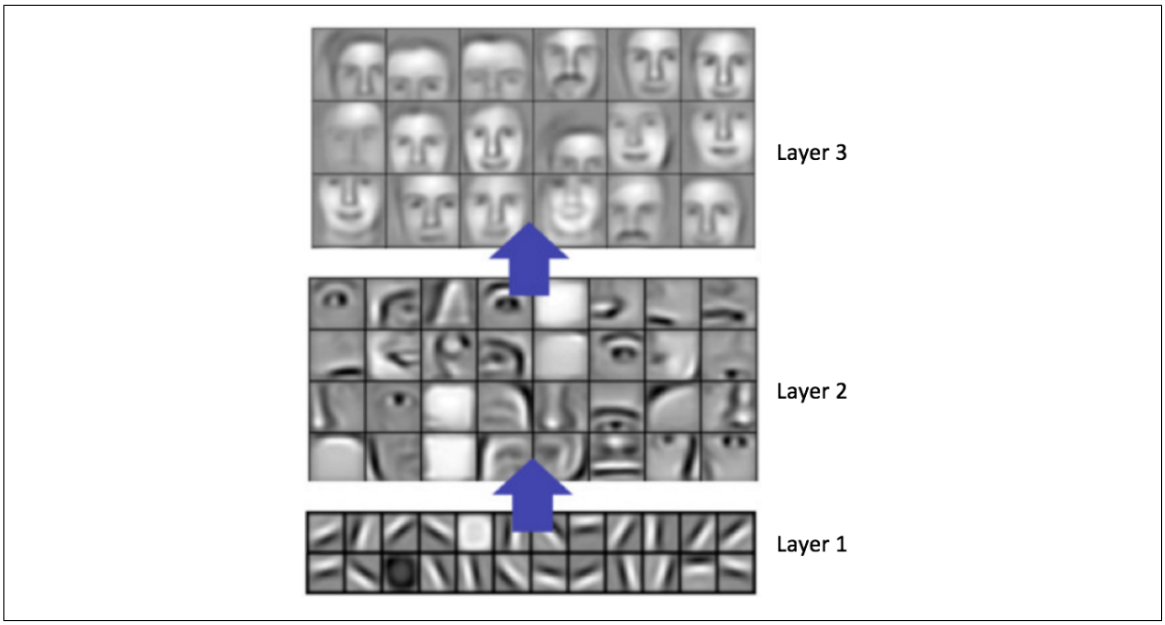


Figure 4.1. Learned features showing increasing compositionality of features through different convolutional layers of a neural network (H. Lee et al., 2009).

ConvNets begin with initial random filter weights, which logically do not detect patterns. Instead, *PatchesInit* initializes the filter weights with common patterns we find in the training data, enabling the network to detect common patterns early during training.

PatchesInit initializes the filter weights of each layer sequentially, where we use the same procedure but only changing the compositionality level of the patterns we find. We extract patterns from the raw pixels to initialize the filter weights of the first layer. On the other hand, we compose the low-level patterns into higher-level patterns to initialize the filter weights of the higher-level layers accordingly. We divide our method into three steps, which are then repeated sequentially to initialize the weights of each layer.

First, *PatchesInit* extracts a set of candidate patches from the training images (Section 4.2.1). Patches are image segments that we use to form common patterns. We want to

form patterns that help recognize objects, so we use an objectness algorithm and sample patches from possible object locations in the image.

Second, *PatchesInit* clusters the candidate patches to separate and identify common groups, from where we then form patterns (Section 4.2.2). We group all the available patches into a fixed number of clusters and then select a subset of adequate ones depending on the initializing layer. For low-level layers, we select clusters that produce patterns that are common to all classes, whereas for high-level layers, we select the ones that produce patterns that help to discriminate among the classes.

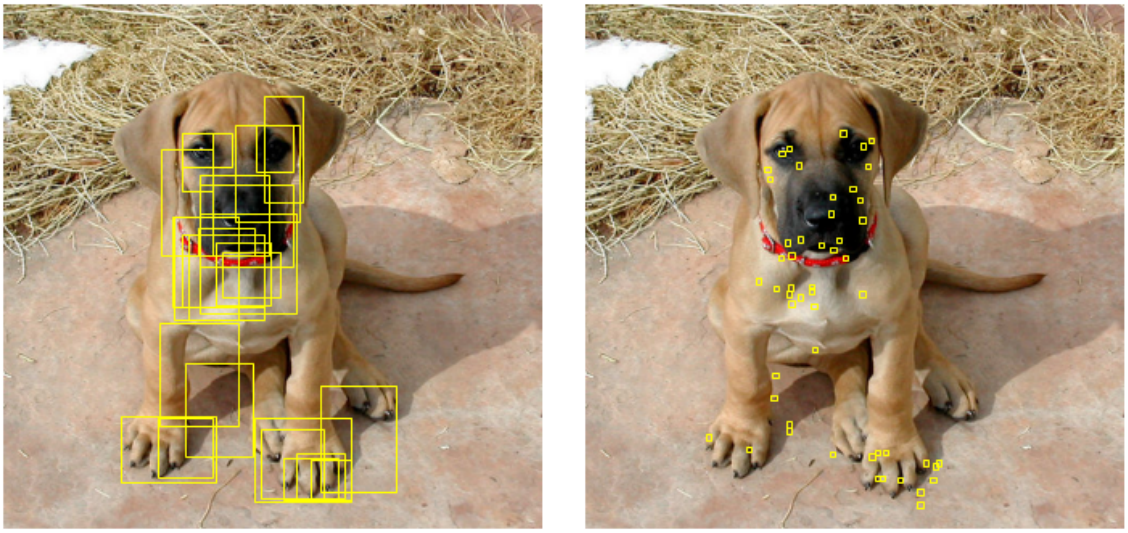
Third, *PatchesInit* transforms patterns into filter weights (Section 4.2.3). We normalize and scale each pattern to form the weights of one filter. This step finalizes one iteration of the *PatchesInit* procedure, initializing the filter weights of one layer.

We explain the details behind these recently mentioned steps and the compositionality strategy we use in the following subsections.

4.2.1. Extracting patches from training data

As a first step, *PatchesInit* obtains candidate patches from the input data, which we then cluster and form patterns. A patch is a segment of the image or a group of locally-connected units from the input data received. In both cases, we represent patches as an array of values.

To initialize the weights of the first layer, we extract patches from the input images by first running an objectness algorithm, as shown in lines 3-4 from Algorithm 1. We use selective search (Uijlings et al., 2013), which combines the strength of both an exhaustive search and segmentation, generating bounding boxes that represent possible object locations. We then sample patches from these bounding boxes (line 8 from Algorithm 1). Figure 4.2 shows an example, where we generate bounding boxes and sample patches from them.



(a) Bounding boxes

(b) Sampled patches

Figure 4.2. Extracting patches from an example image from Imagenet (Deng et al., 2009). (a) Shows bounding boxes generated by a selective search algorithm (Uijlings et al., 2013), and (b) shows randomly sampled patches from these bounding boxes.

To initialize the weights of the rest of the layers, instead of extracting patches from the input images, we use the previous layers (whose weights we already initialized in line 17 from Algorithm 1) to compose and generate higher-level patterns, from where we then extract patches (lines 5-6 from Algorithm 1). We give in-depth details of this compositionality procedure in Section 4.2.4.

We sample the same number of patches P for each layer in the network (line 8 from Algorithm 1). We require P as a hyperparameter, which we further discuss and experiment in Section 5.6.1. We extract patches of the same size as the filters in each layer, avoiding an upsample or downsample that may alter the patch information.

4.2.2. Clustering patches and forming patterns

As a second step, *PatchesInit* clusters all the candidate patches to separate and identify common groups, which we then use to form common patterns. In particular, the proposed method groups all the available patches into a fixed number of clusters, selects a subset of adequate ones, and then forms patterns. We then use each pattern to initialize one filter weights.

Before clustering patches, we need to make sure they have an appropiate dimensionality for comparison. Data with very high dimensionality can suffer from *the curse of dimensionality* (Bellman, 1966), where the volume of the space increases so fast that the available data becomes sparse; therefore, clustering data becomes a very difficult task (Steinbach et al., 2004).

We use Principal Component Analysis PCA (Pearson, 1901) to reduce the dimensionality of the patches (line 10 from Algorithm 1). The main idea of PCA is to reduce the dimensionality of a dataset by transforming the variables into a new set of orthogonal variables (eigenvectors of a covariance matrix). The algorithm then chooses those variables that maximize the variance of the projected data. Following the intuition from VB and David (2015), we only select the first few projected variables to reduce the dimensionality of each patch.

Now that patches have a suitable dimensionality, *PatchesInit* executes a KMeans algorithm (Jin & Han, 2010) through the candidate patches to produce C clusters (line 11 from Algorithm 1). KMeans receives the number of clusters C as a hyperparameter and iteratively distributes the data between C groups minimizing the sum of the euclidean distance of all the data points to their corresponding group centroid. KMeans then yields each cluster with their corresponding patches and centroid. The hyperparameter C is thoroughly experimented in Section 5.6.1.

After forming the clusters, we need to select a subset of adequate ones depending on the initializing layer. As a reminder, we use one pattern to initialize one filter, and we

form one pattern from one cluster, so we select as many clusters as filters in each layer. As mentioned in Section 3.3, we use entropy to understand how common or discriminative clusters are for the classes present in the dataset. We calculate the entropy of the labels of the patches each cluster contains, where each patch has the same label as the image where it came from (line 12 from Algorithm 1). We use the following equation:

$$E(c) = \sum_{i=1}^{N} -p_i \log_2 p_i, \tag{4.1}$$

where c is a cluster that contains patches from N different classes (set of discrete data points); and p_i is the probability of class i. If a cluster has maximum entropy, it contains patches from all classes; therefore, it forms a common pattern and vice versa.

Low-level layers select clusters with high entropy because they need patterns shared by all classes to create general representations of the data. On the other hand, high-level layers select clusters with low entropy (i.e., groups that contain patches from only a few classes or even one) because they need discriminative power to separate data from different classes. We refer to the exact layer where the selection criteria changes as layer D, which we require as a hyperparameter. Choosing the layer D is a non-trivial decision and is further discussed in Section 5.6.2.

After selecting the adequate clusters (line 13 from Algorithm 1), we form each pattern as the patch closest to its centroid (line 14 from Algorithm 1). Even though we compare distances in the reduced space, we use patches in their original space to form the patterns. We experimented with forming the patterns in three different ways but got worse results: using the centroid itself, using the average of the five nearest patches to the centroid, or optimizing weights to maximize the dot product with the patches of the corresponding cluster.

4.2.3. Initializing filter weights from patterns

As a final step, *PatchesInit* procedure normalizes and scales patterns into suitable weights. The values of the weights are fundamental for a stable training, as mentioned in Section 2.2.

Following the intuition from Glorot and Bengio (2010), we normalize and scale each pattern with the normalization factor used by the Xavier initialization (line 16 from Algorithm 1). After the scaling is done, each pattern is used as the weights of one filter (line 17 from Algorithm 1):

$$filter\_weights = \frac{(pattern - \mu)}{\sigma} \times \frac{\sqrt{2}}{\sqrt{n_{l-1} + n_{l+1}}}, \tag{4.2}$$

where μ and σ are the mean and standard deviation of the pattern, respectively; and n_{l-1}, n_{l+1} the number of neurons in the previous and next layer of the network, respectively.

This step finalizes one iteration of the *PatchesInit* procedure, initializing the filter weights of one layer. The procedure then runs sequentially for the rest of the layers of the network (line 1 from Algorithm 1), where we compose patterns to generate accordingly higher-level patterns.

4.2.4. Composing patterns for higher-level layers

PatchesInit seeks to initialize each layer with patterns of their corresponding compositionality level. To achieve this, we use the previous layers (whose weights we already initialized) to compose and generate higher-level patterns during each iteration of the sequential process.

As mentioned in Section 4.2.1, we extract patches from the raw pixels to initialize the weights of the first layer. To initialize the weights of the rest of the layers, we partially forward pass the images through the model until the previous layer (whose weights we already initialized in line 17 from Algorithm 1) and generate activation maps. These activation maps represent the training images on a higher compositionality level based on the patterns found by the previous iterations of *PatchesInit*. We then sample patches from these activation maps, extracting patches with an accordingly-higher compositionality level for each layer. We do not run an objectness algorithm for these cases; instead, we sample patches from the entire activation maps.

Algorithm 1 shows the compositionality strategy we use. In lines 3-4, where l is 1, we execute an objectness algorithm over X to generate bounding boxes. Conversely, we show the procedure for the rest of the layers l in lines 5-6, where we execute a partial forward pass of the images X through the model f, generating the activation maps for layer $l-1$. We then sample patches from these bounding boxes or activation maps, as shown in line 8.

After the proposed method runs sequentially for all the layers in the network, the *PatchesInit* procedure ends, and the model is ready for Supermask training.

4.3. Algorithm

Algorithm 1 *PatchesInit*

Input: $\{X, Y\}$ dataset
Input: f model with L layers
Input: Hyperparameters: P number of patches, C number of clusters, D layer where cluster selection criteria changes

for layer $l = 1, 2, \ldots L$ **do**
    # Step 1 : Extracting patches from training data
    if $l == 1$ **then**
        Execute an objectness algorithm to generate bounding boxes from X
    else
        Generate activation maps of layer $l - 1$ by forward passing X through f
    end if
    Sample P candidate patches from line 4 or 6
    # Step 2 : Clustering patches and forming patterns
    Reduce dimensionality of candidate patches
    Cluster candidate patches into C clusters
    Calculate the entropy of each cluster using the labels Y of their patches
    Select adequate clusters depending on layer D
    Form the pattern for each selected cluster
    # Step 3 : Initializing filter weights from patterns
    Normalize and scale each pattern
    Assign each pattern to one filter weights in layer l from model f
end for

Output: Initialized weights of model f

5. EXPERIMENTS

This section covers the details of the experiments and analysis of *PatchesInit*. Section 5.1 and Section 5.2 cover the datasets and implementation details, respectively. Section 5.3, Section 5.4, and Section 5.5 dive into experiments and evaluation metrics. Section 5.6 performs ablation studies over relevant design and implementation issues behind our proposed method. Finally, Section 5.7 compares *PatchesInit* against other frequently used initialization methods.

5.1. Datasets

We test *PatchesInit* under the task of image classification. In particular, we test on three datasets: CIFAR-10, CIFAR-100, and TinyImagenet-200.

Krizhevsky and Hinton (2009) collected and published these frequently used datasets CIFAR-10 and CIFAR-100. Both datasets consist of 60,000 32x32 color images divided into 50,000 for training and 10,000 for test. CIFAR-10 and CIFAR-100 present 10 and 100 equally divided classes, respectively. Figure 5.1 shows all the classes in CIFAR-10 with several example images from each class.

TinyImagenet-200 [1] is a modified subset of the original Imagenet dataset (Deng et al., 2009). It was created by Fei-Fei Li, Andrej Karpathy, and Justin Johnson as part of their CS231N course at Stanford university [2]. The dataset consists of 110,000 64x64 color images divided into 100,000 for training and 10,000 for test. The dataset presents 200 balanced classes.

[1] Extracted from: https://tiny-imagenet.herokuapp.com/
[2] http://cs231n.stanford.edu/

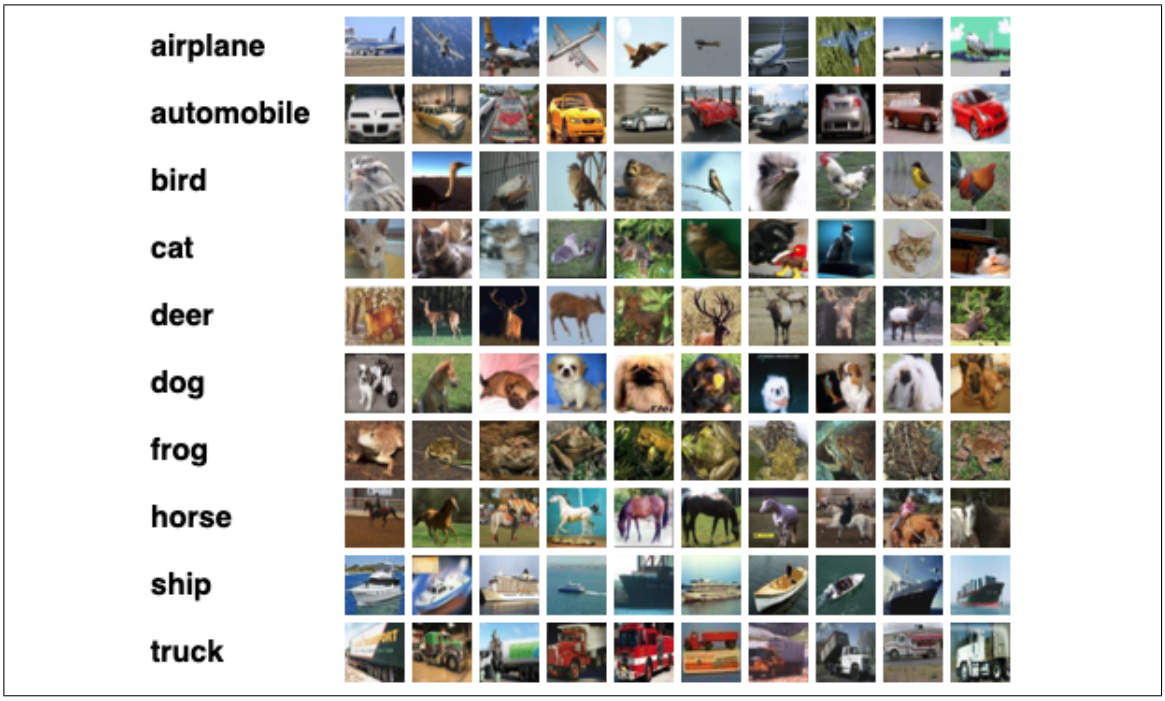


Figure 5.1. Example of the classes in the CIFAR-10 dataset. Image retrieved from (Krizhevsky & Hinton, 2009).

5.2. Implementation and hyperparameters

In our experiments, we use similar models to (Frankle & Carbin, 2019; Zhou et al., 2019; Ramanujan et al., 2020). We refer to the models as Conv2, Conv4, Conv6, and ResNet-18. Table 5.1 shows the details behind each of these models.

We optimize the models with a cross-entropy loss and Stochastic Gradient Descent. We train all models for 100 epochs and report the Top1 test accuracy of the last epoch. We use cosine learning rate decay. For CIFAR-10 and CIFAR-100, we train the models with learning rate 0.1, weight decay 1e-4, momentum 0.9, and batch size 128. For TinyImagenet-200, we train the models with learning rate 0.256, weight decay 1e-5, momentum 0.875, and batch size 256.

Table 5.1. Architecture details of models Conv2, Conv4, Conv6, and ResNet-18 used in (Frankle & Carbin, 2019; Zhou et al., 2019; Ramanujan et al., 2020). Each model first performs convolutions (Conv Layers) followed by fully connected layers (FC). For Conv Layers, each cell shows the number of filters and for FC Layers, each cell shows the number of neurons. Pool denotes max pooling operations and square brackets denote residual connections.

Model	Conv2	Conv4	Conv6	ResNet-18
				64, pool
				2x[64, 64]
			64, 64, pool	2x[128, 128]
Conv		64, 64, pool	128, 128, pool	2x[256, 256]
Layers	64, 64, pool	128, 128, pool	256, 256, pool	2x[512, 512]
FC Layers	256, 256, 10	256, 256, 10	256, 256, 10	avg-pool, 200

We compare the performance of *PatchesInit* against the performance of models whose weights we initialize with Xavier initialization. We call these *Random* and they act as baselines. We use the same training parameters for these.

Diving into the *edge-popup* algorithm parameters, we use the same as in (Ramanujan et al., 2020). We use $prune\_percentage$ of $\{10, 30, 50, 70, 90\}$ for a comprehensive analysis of the method.

The details of *PatchesInit* are the following. We implement a selective search using OpenCV (Bradski, 2000) and filter out bounding boxes covering more than 50% of the image. We randomly sample 50 patches from each input data point without replacement. We implement PCA with *sklearn* (Pedregosa et al., 2011) and reduce the dimensionality of the patches to 27; following the intuition from VB and David (2015). We also implement KMeans with the *MiniBatchKMeans* method from *sklearn*, using batch size 256 and 10 maximum iterations. We filter out clusters that contain less than five patches.

For CIFAR-10 and CIFAR-100, the number of patches P we extract is 50,000, and the number of clusters C is 500. For TinyImagenet-200, the number of patches P we extract is 100,000, and the number of clusters C is 1,000. Layer D, where we change the selection criteria of clusters, is always the last layer of the network. These three hyperparameters are experimented in Section 5.6.1 and Section 5.6.2.

5.3. Training the Supermask on CIFAR-10

We first evaluate the performance of *PatchesInit* on CIFAR-10. To get a broader sense of performance, we train Conv2, Conv4, and Conv6 models with all the pruning rates mentioned in Section 5.2. We show the exact results in Tables 8.1 - 8.3 from Appendix 1.

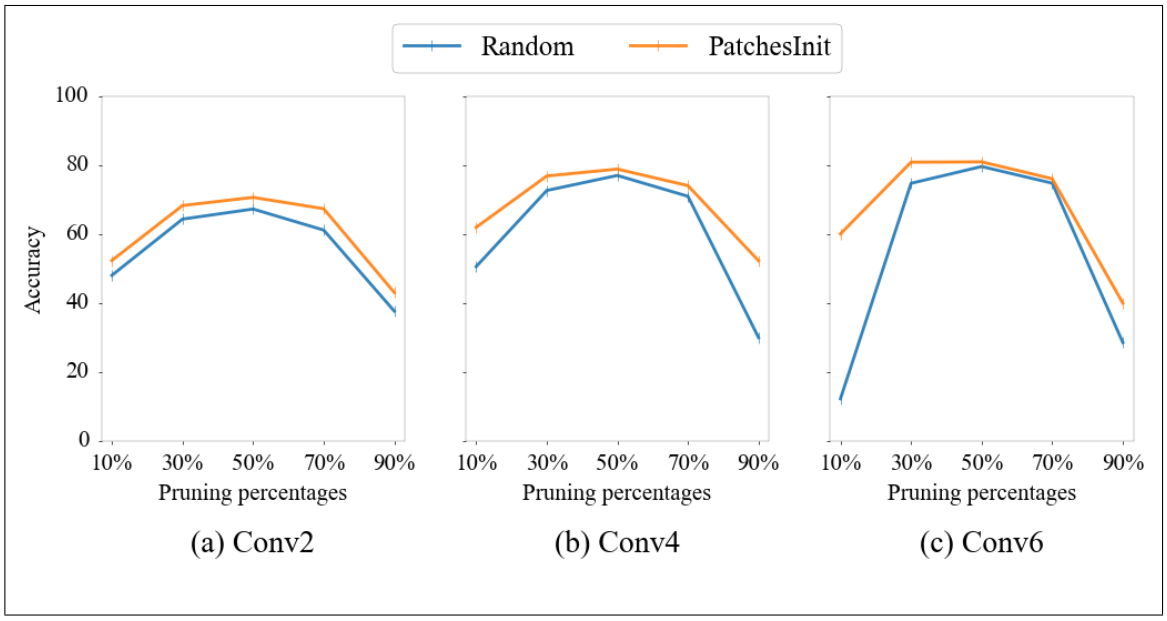


Figure 5.2. Training on CIFAR-10: Supermask test performance of Conv2, Conv4, and Conv6 models achieved with 10-90% pruning rates. *PatchesInit* outperforms *Random* for all pruning rates. Subplot (a) shows the performance of Conv2, (b) Conv4, and (c) Conv6 models.

As illustrated by Figure 5.2, *PatchesInit* outperforms the baseline for all models and pruning rates. For Conv2, *PatchesInit* increases performance for all the pruning rates, with 3% and 6% accuracy improvements for 50% and 70% pruning rates, respectively. Results

for Conv4 and Conv6 are slightly different. In those cases where Supermask training seems to find effective subnetworks (30-70% pruning rates), initializing the weights with *PatchesInit* produces subnetworks with a 2-4% accuracy increase for Conv4 and 1-6% accuracy increase for Conv6. In other cases, initializing the weights with *PatchesInit* has a remarkable effect over Supermask training. For Conv6, we find subnetworks that achieve up to 48% and 11% accuracy improvements for 10% and 90% pruning rates, respectively.

Additionally, Figure 5.3 further shows that *PatchesInit* does provide a better starting point by showing a clear difference in performance during the early epochs of training, followed by a faster convergence. On the very first epoch, *PatchesInit* beats *Random* by more than 28% in accuracy and presents a steeper accuracy curve, inducing a faster convergence rate. Also, by analyzing the error curves, we deduce that *PatchesInit* initialization produces a stable training scheme, where the loss starts at a reasonable point and tends to drop. Figure 8.1 from Appendix 1 shows a similar trend while training Conv6 for different pruning percentages.

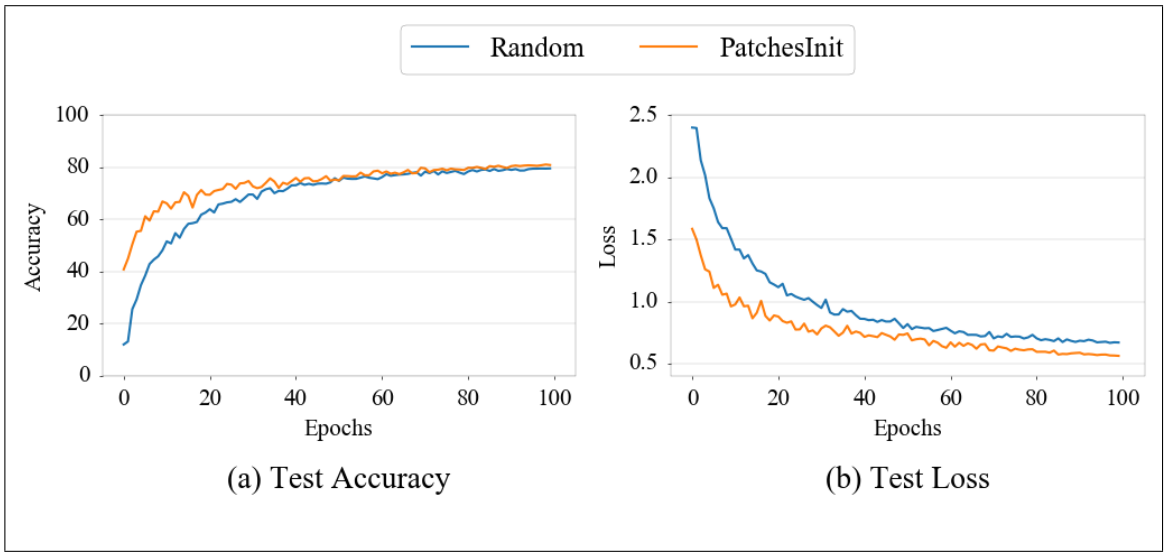


Figure 5.3. Training of Conv6 on CIFAR-10 with 50% pruning: *PatchesInit* converges faster than *Random* and to higher performance value. Subplot (a) shows the test accuracy and (b) the loss during training.

5.4. Training the Supermask on CIFAR-100

In this section, we experiment and evaluate the performance of *PatchesInit* on CIFAR-100. Similar to Section 5.3, we train Conv2, Conv4, and Conv6 models with all the pruning rates mentioned in Section 5.2. In this case, we analyze the Top5 accuracy for a more representative performance measure and report the Top1 results in Appendix 2. Additionally, we run each experiment with three different seeds, where each line represents the mean result, and the shaded area represents the standard deviation we obtain (Figure 5.4). We show all the exact results in Tables 8.4 - 8.9 from Appendix 2.

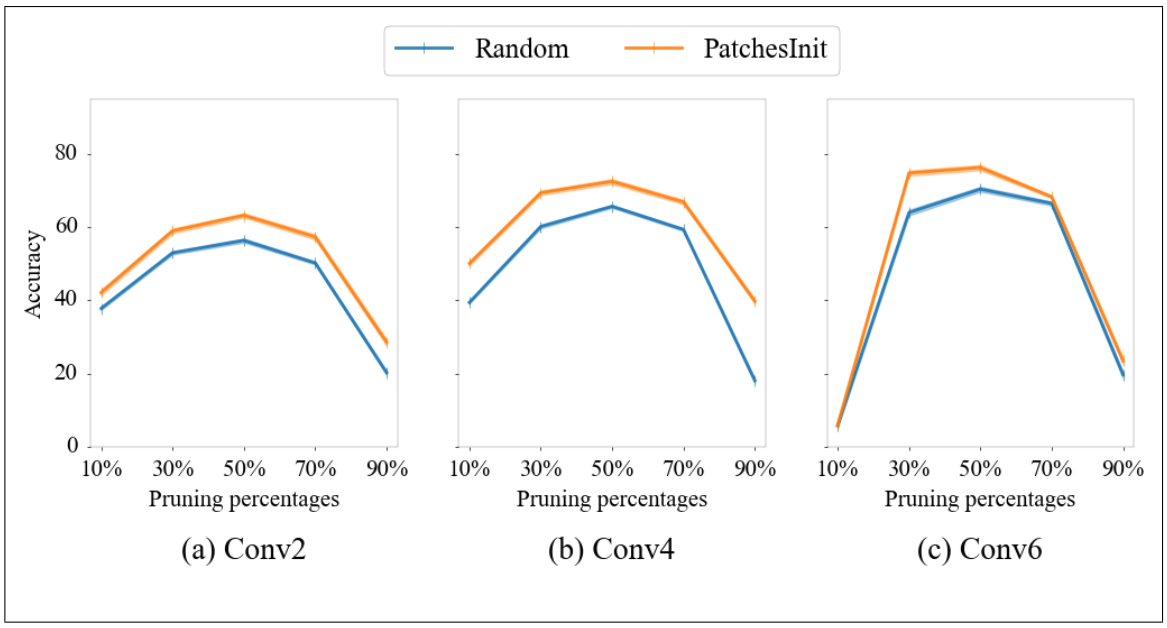


Figure 5.4. Training on CIFAR-100: Supermask Top5 test performance of Conv2, Conv4, and Conv6 models achieved with 10-90% pruning rates. *PatchesInit* outperforms *Random* for all pruning rates. Subplot (a) shows the performance of Conv2, (b) Conv4, and (c) Conv6 models.

Figure 5.4 shows a similar trend with even better results than those we obtain for CIFAR-10, especially for 30-70% pruning rates where Supermask training does find effective subnetworks. For Conv2, *PatchesInit* outperforms the baseline for all pruning rates. We see 6-7% accuracy improvements for 30-70% pruning rates. The same goes for Conv4

and Conv6, where we now see a remarkable performance difference. For Conv4, *PatchesInit* increases accuracy from 7-9% for 30-70% pruning rates, but now shows inferior improvements for those cases where Supermask training does not seem to find effective subnetworks; only improving by 11% and 22% for 10% and 90% pruning rates, respectively. For a fair comparison, Table 8.6 shows a 7% and 12% Top1 accuracy increase for 10% and 90% pruning rates, respectively; which is inferior to the 11% and 22% Top1 accuracy increase presented in Table 8.2.

Results for Conv6 are similar to Conv4. For 30-70% pruning rates, Figure 5.4.c shows that *PatchesInit* outperforms the baseline, improving accuracy between 2-11%. Different from CIFAR-10, in this case, *PatchesInit* could not find an effective subnetwork for Conv6 and 10% pruning. Also, For 90% pruning, our method achieved better performance than *Random* (improved by 3%), but with a smaller accuracy improvement than on CIFAR-10 (11% increase in accuracy).

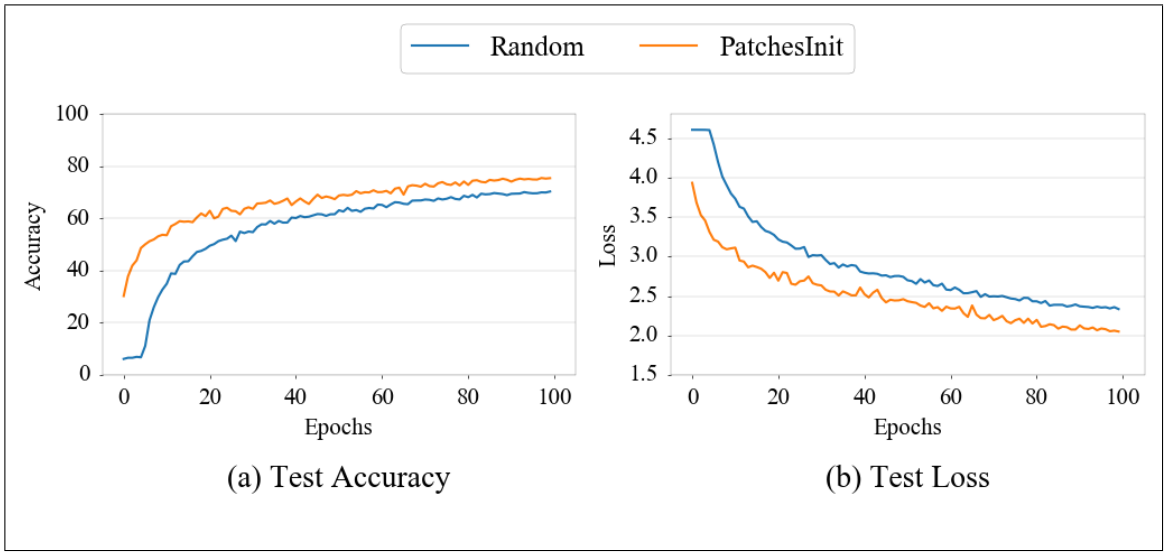


Figure 5.5. Training of Conv6 on CIFAR-100 with 50% pruning: *PatchesInit* converges faster than *Random* and to higher performance value. Subplot (a) shows the Top5 test accuracy and (b) the loss during training.

In Figure 5.5, we show training curves for Conv6 with 50% pruning rate, from where we can see that *PatchesInit* again provides a better starting point than random initialization.

Our method surpasses the baseline for 24% in accuracy in the first epoch and presents a faster convergence. Analyzing Figure 5.5.b, we can also note that initializing the weights with *PatchesInit* does provide a stable training scheme. Figure 8.2 from Appendix 2 shows a similar trend while training Conv6 for different pruning percentages.

We conclude that initializing the weights of shallow ConvNets with *PatchesInit* increases the performance of the subnetworks we find under Supermask training. Additionally, we can also conclude that *PatchesInit* generates weights that provide a better starting point than random initialization and produce a stable training scheme.

5.5. Training the Supermask on TinyImagenet-200

In this section, we experiment and evaluate the performance of *PatchesInit* on TinyImagenet-200. We train a ResNet-18 model for 10-70% pruning rates, excluding 90%, because the Supermask does not seem to find effective subnetworks for those cases.

Table 5.2. Test results achieved for *PatchesInit* on a ResNet-18 model trained on TinyImagenet-200: *Partial-PatchesInit* outperforms the baseline for all pruning rates.

Init method	10% pruning	30% pruning	50% pruning	70% pruning
Random	50.0	54.1	52.7	51.1
PatchesInit	47.6	53.0	52.5	48.9
Partial-PatchesInit	**50.5**	**55.1**	**54.0**	**51.9**

On TinyImagenet-200, we observe different trends to CIFAR-10 and CIFAR-100. Results from Table 5.2 show that *PatchesInit* did not improve the performance of the baseline, decreasing performance from 0.2-2.4% in accuracy for 50% and 10% pruning rates, respectively.

The main reason for this decay lies in the fact that the method cannot initialize the weights effectively under high compositionality levels. We investigate other possible causes in Appendix 3 that help us conclude this.

We create a modified version of *PatchesInit* that only initializes the first block of layers of a ResNet-18 model, leaving the rest of the network with their initial random weights. We refer to this method as *Partial-PatchesInit*. We train *Partial-PatchesInit* on TinyImagenet-200, where we initialize the rest of the network with the same random values as the baseline for a fair comparison.

Table 5.2 shows that *Partial-PatchesInit* does improve the performance of the baseline for all pruning rates. We see accuracy increases varying from 0.5-1.3% for 10% and 50% pruning rates, respectively; demonstrating that by only initializing the first quarter of the

layers with *PatchesInit*, Supermask training finds subnetworks with better performance than random initialization.

5.6. Ablation studies

In this section, we perform ablation studies over relevant design and implementation issues behind our proposed method. We perform all experiments with a Conv6 model with 30%, 50%, and 70% pruning rates on CIFAR-100 using the hyperparameters mentioned in Section 5.2. We report the Top5 accuracy and the time cost is in minutes. We use the same random seed for all experiments.

5.6.1. Patch extraction and clustering

First, we experiment over the number of patches P to extract, and the number of clusters C. We analyze the effect both variables have when initializing the weights of a model with *PatchesInit*, comparing the accuracy obtained and the computational cost associated.

KMeans has a complexity of $\mathcal{O}(x * y * z)$, where x is the number of data points, y the number of clusters, and z the dimensionality of each data point. As mentioned in Section 4.2.1, we extract patches of the same size as the filters in each layer, so we do not experiment over z. On the other hand, we expect a linear increase in computational complexity when increasing x or y, which would be P or C in our case.

We implement five different P and C configurations, varying from 50,000-1,000,000 patches and 500-10,000 clusters, respectively. Additionally, we include a variant, which we refer to as *SimpleInit*, where C is exactly the number of filters in each layer (64, 64, 128, 128, 256, 256 clusters for Conv6), skipping the selection step. *SimpleInit* selects the minimum number of clusters, therefore we consider it as a floor.

Table 5.3 displays results for 50% pruning. We find subnetworks with the best performance when P is 50,000 and C is 500, outperforming *SimpleInit* by 1.6% in terms of accuracy. Nevertheless, for P and C values less than 200,000 and 2,000, respectively, *PatchesInit* finds subnetwork with competitive results. When P and C are greater than

Table 5.3. Top5 accuracy and time cost in minutes achieved by varying the number of patches P to extract and the number of clusters C, for Conv6 model trained on CIFAR-100 with 50% of pruning. The best performance is achieved when P is 50,000 patches and C is 500 clusters.

Number patches P	Number clusters C	Accuracy %	Time Cost (minutes)
10k	*SimpleInit*	74.7	2.0
50k	**500**	**76.3**	**11.5**
100k	1,000	75.5	23.4
200k	2,000	74.4	41.6
500k	5,000	73.7	147.1
1M	10,000	71.8	308.9

those values, we notice a considerable decrease reaching 4.5% in terms of accuracy. In Appendix 4, we show the results for 30% and 70% pruning, similar to those just mentioned.

Table 5.3 shows that computational complexity does increase linearly with P and C. *PatchesInit* takes around 12 minutes to initialize the weights when P is 50,000 and C is 500, which we consider as an affordable cost considering our hardware restrictions. However, *SimpleInit* finds subnetworks with competitive performance at the expense of a low computational cost. This variant could be interesting for other research environments with lower computational resources.

Additionally, we study the average entropy of the selected clusters in each layer (Table 5.4). As a reference, the minimum and maximum entropy for CIFAR-100 is 0 and 6.64, respectively. All configurations of P and C, except *SimpleInit*, effectively select high-entropy clusters for low-level layers and low-entropy clusters for high-level layers, following the intuition from Section 4.2.2. As mentioned in Section 5.2, we chose layer D as the last layer of the model (layer 6 in this case), which is reflected in the decrease of the average entropy of the selected clusters between layers 5 and 6. On the other hand, *SimpleInit* struggles to select clusters with high entropy for low-level layers.

Table 5.4. Mean entropy of the selected clusters of each layer achieved by varying the number of patches P to extract and the number of clusters C, for Conv6 model trained on CIFAR-100 with 50% of pruning.

Number clusters C	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Layer 6
SimpleInit	4.75	5.02	4.41	4.40	3.78	3.62
500	**5.75**	**5.85**	**5.57**	**5.63**	**5.44**	**4.19**
1,000	5.87	5.95	5.75	5.78	5.59	4.09
2,000	6.08	6.03	5.94	6.18	6.03	4.02
5,000	6.07	6.05	6.11	6.19	6.11	3.54
10,000	6.18	6.24	6.09	6.18	6.00	2.55

5.6.2. Cluster selection

The second relevant design and implementation issue relies on the layer D, where the selection criteria for clusters changes. As mentioned in Section 4.2.2, low-level layers should be initialized with patterns representing all classes, while deeper layers should be initialized with discriminant patterns. The question is, where do we set the change? Accuracy is the only evaluation criterion this time (no extra computational cost associated).

To test this hyperparameter, we experiment with six configurations varying layer D. *3C-3D* is a Conv6 model where we initialize the first three layers selecting common clusters (3C) and the last three layers selecting discriminant clusters (3D). In this case, we set the change on layer 4 (4th layer being initialized with discriminant clusters), so D would be 4. *6C-0D* initializes all layers with common clusters (D does not apply) and acts as a floor for this experiment. On the other hand, *0C-6D* initializes all layers with discriminant clusters (D is 0) and acts as a ceiling.

Table 5.5 provides the results of these experiments. As we can see, *5C-1D* presents the highest results (D is 6), where we initialize the first five layers selecting common clusters and only the last layer with discriminant clusters. Table 8.12 and Table 8.13 (Appendix 4) show similar results for different pruning rates.

Table 5.5. Accuracy achieved by varying layer D where the selection criteria of clusters changes, for Conv6 model trained on CIFAR-100 with 50% of pruning.

Selection configuration	D	Accuracy %
6C-0D	-	74.8
5C-1D	**6**	**76.3**
4C-2D	5	75.3
3C-3D	4	75.2
2C-4D	3	74.5
1C-5D	2	73.8
0C-6D	1	70.9

We notice that initializing at least some high-level layers with discriminant clusters increases the performance of the subnetworks we find. *6C-0D* finds subnetworks with competitive performance, but they are outperformed by *5C-1D*, *4C-2D*, and *3C-3D*. Additionally, we see that performance decreases when we initialize more than two or three layers with discriminant clusters, suggesting that the model needs to have at least more than half of the layers initialized with common clusters.

Another interesting insight relies on the necessity of initializing at least one low-level layer with common clusters. Table 5.5 shows a remarkable 3% decrease in accuracy for *0C-6D* compared with *1C-5D*, suggesting that initializing at least the first layer with common clusters is substantial.

5.7. Comparing with other initializations

The last experiment of this investigation compares *PatchesInit* with respect to other frequently used initialization methods. This verifies if *PatchesInit* presents the state of the art performance for Supermask training. We test on CIFAR-100 and run each experiment with three different seeds, reporting the mean Top5 accuracy and standard deviation.

We compare *PatchesInit* with respect to four other initialization methods. Xavier Normal $\mathcal{N}(0, \sigma)$ (previous baseline), Xavier Uniform $\mathcal{U}(-\sigma, \sigma)$, Kaiming Normal $\mathcal{N}(0, \sigma)$ (He et al., 2015) and Kaiming Uniform $\mathcal{U}(-\sigma, \sigma)$. The parameter σ represents the normalization factor used in Xavier and Kaiming initialization methods.

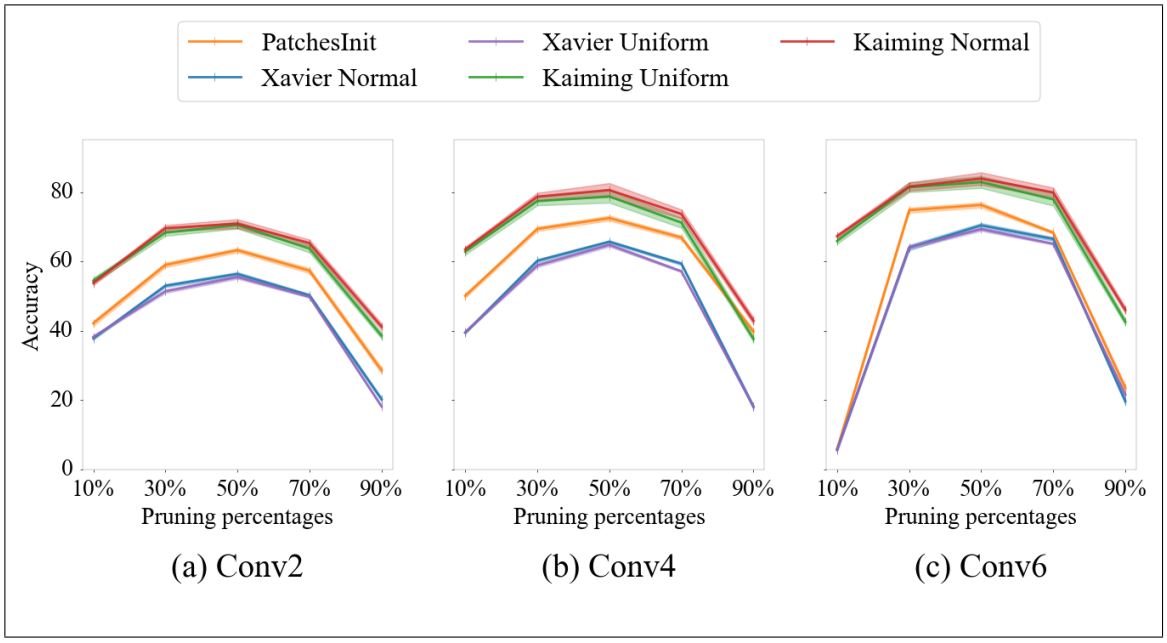


Figure 5.6. Comparing different initialization methods on CIFAR-100. *PatchesInit* does not outperform Kaiming initialization. Subplot (a) shows the performance of Conv2, (b) Conv4, and (c) Conv6 models.

Results are shown in Figure 5.6. First, we can notice that *PatchesInit* does not outperform Kaiming initialization, and therefore is not the state of the art for Supermask training. Even though they present better results, Kaiming initialization presents unstable results that may vary up to 4% in terms of accuracy for different seeds. Our preliminary

results had shown similar results, which was the main reason for deciding to use Xavier instead of Kaiming for our method. Future work plans to use Kaiming intuition for our method and solving their stability problem under Supermask training.

Another interesting insight is that for both, Xavier and Kaiming, Normal distributions outperformed Uniform distributions. Additionally, *PatchesInit* beat Xavier Uniform for all pruning rates and model configurations.

6. CONCLUSIONS

This thesis hypothesizes that initializing weights with patterns found in the training data, will increase the generalization test performance of the subnetworks detected under Supermask training. We propose a novel weight initialization method for ConvNets under Supermask training, which we refer to as *PatchesInit*.

This hypothesis is fulfilled by initializing the weights of the model with *PatchesInit*. Section 5.3 and Section 5.4 show that for rather shallow ConvNets, initializing the weights of the model with *PatchesInit* increases the performance of the subnetworks we find. For CIFAR-10, we see increases in Top1 accuracy varying from 3-6% for Conv2, 2-22% for Conv4, and 1-48% for Conv6. For CIFAR-100, we see increases in Top5 accuracy varying from 4-8% for Conv2, 7-22% for Conv4, and 2-11% for Conv6. Additionally, we can also conclude that *PatchesInit* generates weights that provide a better starting point than random initialization and produce a stable training scheme.

In Section 5.5, we can notice that, in the case of a ResNet-18 model, *PatchesInit* does not outperform the baseline. We conclude that *PatchesInit* cannot initialize the weights effectively under high levels of compositionality. However, we further propose a modified version of the method, which we refer to as *Partial-PatchesInit*, that effectively outperforms the baseline by 0.5-1.3% in Top1 accuracy under different pruning rates.

The ablation studies from Section 5.6 allow us to find the best configuration of hyperparameters required by *PatchesInit*. We demonstrate that our method performs best when extracting 50,000 patches from the input data, generating 500 clusters, and only initializing the last layer selecting discriminant clusters. We find subnetworks with the best performance with this configuration, and additionally, initialize the layers with our desired combination of common and discriminant clusters.

From our last experiment, shown in Section 5.7, we can conclude that *PatchesInit* is not state of the art for Supermask training. However, we can conclude that for rather

shallow ConvNets, *PatchesInit* outperforms both Xavier Normal and Xavier Uniform initializations.

The Lottery Ticket Hypothesis and the Supermask seek to make models more efficient, finding subnetworks that predict faster and reduce hardware requirements. This thesis aims to pursue this mission and propose a method to find subnetworks with even better performance. We find subnetworks that use 70%, 50%, or even 30% of the full model weights reaching better performance than Xavier initialization.

7. FUTURE WORK

Even though this investigation showed advances in this area, there is still a long road ahead.

The first line of future work would be to make *PatchesInit* more robust under high levels of compositionality. In this way, *PatchesInit* could then initialize all layers of a ResNet-18 model and find subnetworks with even better performance. Additionally, it would then be interesting to apply *PatchesInit* to even deeper models such as ResNet-50 and ResNet-101, or other deep architectures such as VGG or AlexNet.

Another line of future work is to include the Kaiming initialization intuition in our method, to improve even further the results of *PatchesInit* and hopefully obtain stable results.

One last line of future work lies in verifying whether initializing the weights of a model with *PatchesInit* effectively improves the performance of the subnetworks found using other Supermasks methods. In particular, it would be relevant to try the Supermask procedure from Zhou et al. (2019).

REFERENCES

Bellman, R. (1966). Dynamic programming. *Science*, *153*(3731), 34–37.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1798-1828.

Bengio, Y., Léonard, N., & Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*, 157-66.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Caron, M., Morcos, A., Bojanowski, P., Mairal, J., & Joulin, A. (2020). Pruning convolutional neural networks with self-supervision. *arXiv preprint arXiv:2001.03554*.

Cover, T. M., & Thomas, J. A. (1991). *Elements of information theory*. Wiley-Interscience.

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Vol. 1, pp. 886–893).

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 248–255).

Desai, S., Zhan, H., & Aly, A. (2019). Evaluating lottery tickets under distributional shifts. *arXiv preprint arXiv:1910.12708*.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., & Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, *11*(19), 625-660.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning* (Vol. 70, p. 1126–1135).

Frankle, J., & Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations.*

Frankle, J., Schwab, D. J., & Morcos, A. S. (2020). The early phase of neural network training. *arXiv preprint arXiv:2002.10365*.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Vol. 9, pp. 249–256).

Gomez, A. N., Zhang, I., Kamalakara, S. R., Madaan, D., Swersky, K., Gal, Y., & Hinton, G. E. (2019). Learning sparse networks using targeted dropout. *arXiv preprint arXiv:1905.13678*.

Haldorai, A., & Ramu, A. (2019). Supervised, unsupervised and reinforcement learning - a detailed perspective. *Journal of Advanced Research in Dynamical and Control Systems*, *11*, 429-433.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 1026–1034).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition*.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jin, X., & Han, J. (2010). K-means clustering. In *Encyclopedia of Machine Learning* (pp. 563–564).

Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (pp. 1097–1105).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541–551.

Lee, C.-Y., Xie, S., Gallagher, P., Zhang, Z., & Tu, Z. (2015). Deeply-supervised nets. In *Artificial Intelligence and Statistics* (pp. 562–570).

Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual International Conference on Machine Learning* (p. 609–616).

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE International Conference on Computer Vision* (Vol. 2, pp. 1150–1157).

Morcos, A., Yu, H., Paganini, M., & Tian, Y. (2019). One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *Advances in Neural Information Processing Systems* (pp. 4932–4942).

Ojala, T., Pietikäinen, M., & Harwood, D. (1996). A comparative study of texture measures with classification based on featured distributions. *Pattern recognition*, *29*(1), 51–59.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning* (pp. 1310–1318).

Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *2*(11), 559–572.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., & Rastegari, M. (2020). What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 11893–11902).

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1988). Learning representations by back-propagating errors. In *Neurocomputing: foundations of research* (p. 696–699).

Savarese, P., Silva, H., & Maire, M. (2019). Winning the lottery with continuous sparsification. *arXiv preprint arXiv:1912.04427*.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, *27*(3), 379-423.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale

image recognition. *arXiv preprint arXiv:1409.1556*.

Steinbach, M., Ertöz, L., & Kumar, V. (2004). The challenges of clustering high dimensional data. In *New directions in Statistical Physics* (pp. 273–309).

Uijlings, J. R., Sande, K. E., Gevers, T., & Smeulders, A. W. (2013). Selective search for object recognition. *International Journal of Computer Vision*, *104*(2), 154–171.

VB, S., & David, J. (2015). Significance of dimensionality reduction in image processing. *Signal Image Processing : An International Journal*, *6*, 27-42.

You, H., Li, C., Xu, P., Fu, Y., Wang, Y., Chen, X., ... Lin, Y. (2019). Drawing early-bird tickets: Towards more efficient training of deep networks. *arXiv preprint arXiv:1909.11957*.

Zhong, G., Wang, L.-N., Ling, X., & Dong, J. (2016). An overview on data representation learning: From traditional feature learning to recent deep learning. *The journal of Finance and Data Science*, *2*(4), 265 - 278.

Zhou, H., Lan, J., Liu, R., & Yosinski, J. (2019). Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems* (pp. 3597–3607).

8. APPENDICES

8.1. Appendix 1: Results for CIFAR-10

Table 8.1. Top1 test accuracy results achieved for *PatchesInit* on a Conv2 model trained on CIFAR-10.

Init method	10% pruning	30% pruning	50% pruning	70% pruning	90% pruning
Random	48.1	64.4	67.3	61.2	37.7
PatchesInit	**52.4**	**68.3**	**70.7**	**67.4**	**43.1**

Table 8.2. Top1 test accuracy results achieved for *PatchesInit* on a Conv4 model trained on CIFAR-10.

Init method	10% pruning	30% pruning	50% pruning	70% pruning	90% pruning
Random	50.6	72.7	77.0	71.0	30.0
PatchesInit	**62.0**	**76.9**	**78.9**	**74.1**	**52.3**

Table 8.3. Top1 test accuracy results achieved for *PatchesInit* on a Conv6 model trained on CIFAR-10.

Init method	10% pruning	30% pruning	50% pruning	70% pruning	90% pruning
Random	12.3	74.7	79.6	74.8	28.6
PatchesInit	**60.1**	**80.9**	**81.0**	**76.1**	**40.0**

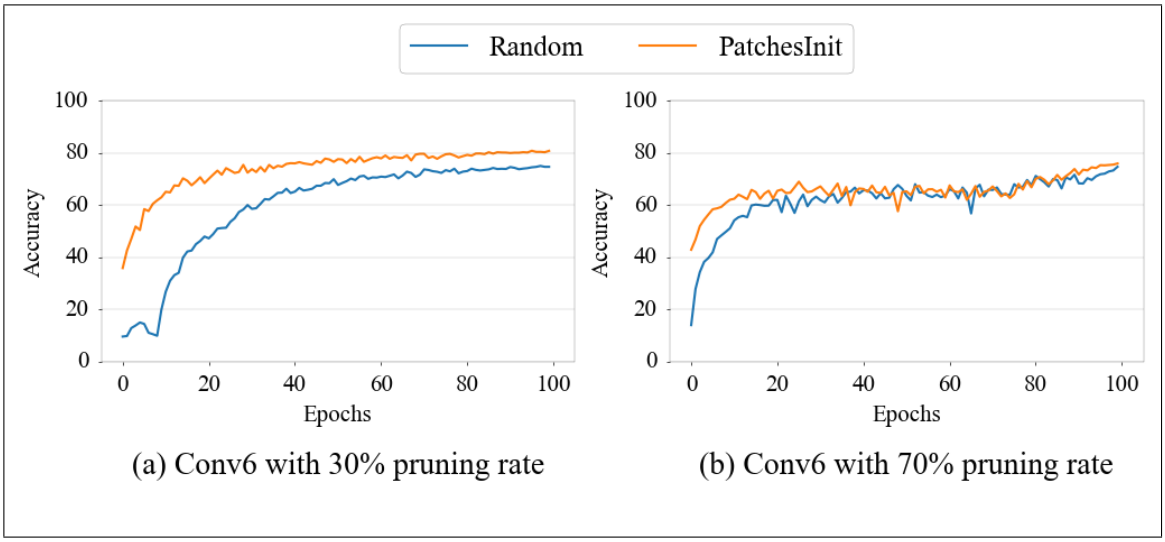


Figure 8.1. Training of Conv6 on CIFAR-10 with 30% and 70% pruning rates. Subplot (a) shows the Top1 test accuracy during training for 30% pruning rate, and (b) for 70% pruning rate.

8.2. Appendix 2: Results for CIFAR-100

Table 8.4. Top1 test accuracy results achieved for *PatchesInit* on a Conv2 model trained on CIFAR-100.

Init method	10% pruning	30% pruning	50% pruning	70% pruning	90% pruning
Random	15.3 $\pm$0.7	25.5 $\pm$0.5	28.3 $\pm$0.5	23.8 $\pm$0.4	6.3 $\pm$0.6
PatchesInit	**17.9** $\pm$0.6	**30.1** $\pm$0.5	**33.5** $\pm$0.4	**28.7** $\pm$0.6	**11.5** $\pm$0.5

Table 8.5. Top5 test accuracy results achieved for *PatchesInit* on a Conv2 model trained on CIFAR-100.

Init method	10% pruning	30% pruning	50% pruning	70% pruning	90% pruning
Random	37.8 $\pm$0.4	53.0 $\pm$0.4	56.4 $\pm$0.5	50.2 $\pm$0.4	20.3 $\pm$0.5
PatchesInit	**42.2** $\pm$0.7	**59.0** $\pm$0.6	**63.3** $\pm$0.4	**57.4** $\pm$0.6	**28.6** $\pm$0.5

Table 8.6. Top1 test accuracy results achieved for *PatchesInit* on a Conv4 model trained on CIFAR-100.

Init method	10% pruning	30% pruning	50% pruning	70% pruning	90% pruning
Random	16.3 $\pm$0.3	31.0 $\pm$0.4	36.7 $\pm$0.4	30.1 $\pm$0.5	4.9 $\pm$0.5
PatchesInit	**23.2** $\pm$0.7	**40.0** $\pm$0.6	**42.6** $\pm$0.6	**37.3** $\pm$0.7	**16.6** $\pm$0.6

Table 8.7. Top5 test accuracy results achieved for *PatchesInit* on a Conv4 model trained on CIFAR-100.

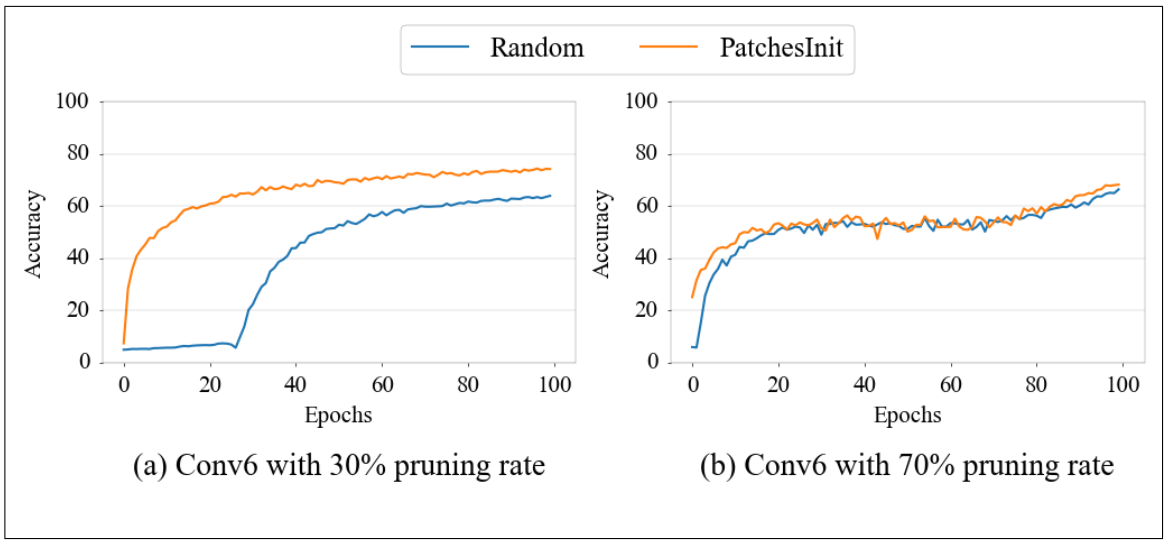
Init method	10% pruning	30% pruning	50% pruning	70% pruning	90% pruning
Random	39.5 $\pm$0.3	60.2 $\pm$0.5	65.7 $\pm$0.4	59.4 $\pm$0.4	18.1 $\pm$0.6
PatchesInit	**50.1** $\pm$0.6	**69.4** $\pm$0.6	**72.5** $\pm$0.5	**66.9** $\pm$0.5	**39.8** $\pm$0.4

Table 8.8. Top1 test accuracy results achieved for *PatchesInit* on a Conv6 model trained on CIFAR-100.

Init method	10% pruning	30% pruning	50% pruning	70% pruning	90% pruning
Random	1 ±0.5	34.4 ±0.6	40.9 ±0.5	37.7 ±0.6	5.4 ±0.7
PatchesInit	1.4 ±0.6	**46.3** ±0.5	**47.7** ±0.6	**38.9** ±0.4	**8.2** ±0.6

Table 8.9. Top5 test accuracy results achieved for *PatchesInit* on a Conv6 model trained on CIFAR-100.

Init method	10% pruning	30% pruning	50% pruning	70% pruning	90% pruning
Random	5.7 ±0.5	64.0 ±0.6	70.4 ±0.6	66.5 ±0.5	19.7 ±0.6
PatchesInit	**5.9** ±0.5	**74.8** ±0.7	**76.3** ±0.6	**68.3** ±0.4	**23.5** ±0.8



(a) Conv6 with 30% pruning rate (b) Conv6 with 70% pruning rate

Figure 8.2. Training of Conv6 on CIFAR-100 with 30% and 70% pruning rates. Subplot (a) shows the Top5 test accuracy during training for 30% pruning rate, and (b) for 70% pruning rate.

8.3. Appendix 3: Analyzing *PatchesInit* for ResNet-18 on TinyImagenet-200

Table 5.2 shows that *PatchesInit* does not outperform the baseline for ResNet-18 on TinyImagenet-200. We analyze three different possible problems and conclude that the main problem relies on initializing the weights under deep compositionality levels.

First, we thought that there was a problem while reducing the dimensionality of the patches in the deeper layers. In the case of ResNet-18, patches in the last block have size 4,608, so we experimented reducing the dimensionality to 64 instead of 27. The subnetworks we found did not improve the performance of our original method.

Second, we thought there was a problem with the number of patches P to extract and the number of clusters C for the deeper layers. Considering that deeper layers have more filters than shallower layers, we tested increasing P to 200,000 and C to 2,000 clusters for the deeper layers. This experiment produced worse results than our original method.

Third, we thought only initializing the last layer of the network with discriminant clusters was not enough. We experimented with initializing the last block of layers with discriminant clusters but obtained similar results. We also tried initializing all layers with common clusters, but again achieved similar results.

We then conclude that the main reason for this decay lies in that the patterns we extract from the more in-depth activation maps are not useful for initializing weights. Our method fails to compose patches for deep layers. To prove that this was the problem, we experimented using *PatchesInit* to initialize only some network layers, leaving the rest with random values. We refer to this variant as *Partial-PatchesInit*.

We experimented initializing the weights with *Partial-PatchesInit* only up to the first, second, and third block of layers (up to the fourth would be equivalent to *PatchesInit*). We concluded that *Partial-PatchesInit* produces effective weights when initializing up to the second block (9 convolutional layers) but then decreased in performance when initializing

up to the third block. The best configuration found was by initializing only the first block of layers, whose results we report in Table 5.2.

8.4. Appendix 4: Results for ablation studies

Table 8.10. Top5 test accuracy and time cost in minutes achieved by varying the number of patches P to extract and the number of clusters C, for Conv6 model trained on CIFAR-100 with 30% of pruning.

Number patches P	Number clusters C	Accuracy %	Time Cost (minutes)
10k	*SimpleInit*	74.2	2.0
50k	**500**	**74.8**	**9.4**
100k	1,000	74.4	35.7
200k	2,000	74.5	84.1
500k	5,000	73.8	226.4
1M	10,000	73.1	335.1

Table 8.11. Top5 test accuracy and time cost in minutes achieved by varying the number of patches P to extract and the number of clusters C, for Conv6 model trained on CIFAR-100 with 70% of pruning.

Number patches P	Number clusters C	Accuracy %	Time Cost (minutes)
10k	*SimpleInit*	65.7	2.0
50k	**500**	**68.3**	**12.2**
100k	1,000	67.1	23.9
200k	2,000	65.7	44.7
500k	5,000	65.5	104.7
1M	10,000	64.9	219.6

Table 8.12. Top5 test accuracy achieved by varying layer D where the selection criteria of clusters changes, for Conv6 model trained on CIFAR-100 with 30% of pruning.

Selection configuration	D	Accuracy %
6C-0D	-	73.4
5C-1D	**6**	**74.8**
4C-2D	5	74.3
3C-3D	4	74.0
2C-4D	3	73.1
1C-5D	2	72.0
0C-6D	1	70.2

Table 8.13. Top5 test accuracy achieved by varying layer D where the selection criteria of clusters changes, for Conv6 model trained on CIFAR-100 with 70% of pruning.

Selection configuration	D	Accuracy %
6C-0D	-	66.5
5C-1D	**6**	**68.3**
4C-2D	5	67.5
3C-3D	4	66.7
2C-4D	3	65.0
1C-5D	2	63.9
0C-6D	1	60.0