National Technical University of Athens
School of Electrical and Computer Engineering

Division of Signals, Control and Robotics
Computer Vision and Signal Processing Lab

# Enhancing Neural Network Compression through Adaptive Pruning Strategies

DIPLOMA THESIS

of

**Aliki Zachou**

**Supervisor**: Petros Maragos
Professor, NTUA

**Co-Supervisor**: Georgios Retsinas
Postdoctoral Researcher, NTUA

Athens, June 2025

**National Technical University of Athens**
School of Electrical and Computer Engineering
Division of Signals, Control and Robotics
Computer Vision and Signal Processing Lab

# Enhancing Neural Network Compression through Adaptive Pruning Strategies

DIPLOMA THESIS

of

**Aliki Zachou**

**Supervisor**: Petros Maragos
Professor, NTUA

**Co-Supervisor**: Georgios Retsinas
Postdoctoral Researcher, NTUA

Approved by the committee on June 18th, 2025.

| | | |
|---|---|---|
| ..... | ..... | ..... |
| Petros Maragos | Gerasimos Potamianos | Ioannis Kordonis |
| Professor, NTUA | Assosiate Professor, UTh | Assistant Professor, NTUA |

Athens, June 2025

.....
**Aliki V. Zachou**
Graduate of Electrical and Computer Engineering NTUA

# Περίληψη

Τα τελευταία χρόνια, τα βαθιά νευρωνικά δίκτυα έχουν επιτύχει εξαιρετικά αποτελέσματα σε διάφορους τομείς με όλο και πιο πολύπλοκα ζητούμενα μηχανικής μάθησης. Ωστόσο, για να επιτευχθεί τέτοια απόδοση, τα μοντέλα γίνονται ολοένα και πιο περίπλοκα και απαιτητικά ως προς την ανάπτυξη και την εκτέλεσή τους. Οι μέθοδοι συμπίεσης μοντέλων, και ιδιαίτερα το κλάδεμα, έχουν αναδειχθεί ως αποτελεσματικές στρατηγικές για την αντιμετώπιση αυτών των προβλημάτων, μέσω της εξάλειψης αχρείαστων παραμέτρων και της μείωσης του υπολογιστικού κόστους. Παρ' όλα αυτά, ακραία επίπεδα αραίωσης συχνά οδηγούν σε αστάθεια και υποβάθμιση της απόδοσης. Η παρούσα εργασία αντιμετωπίζει αυτούς τους περιορισμούς εισάγοντας προσαρμοστικές στρατηγικές κλαδέματος που ενισχύουν την απόδοση ενώ διατηρούν υψηλά επίπεδα συμπίεσης. Η εργασία εξετάζει τις προτεινόμενες στρατηγικές στο Feather, μια πρόσφατη μέθοδο κλαδέματος που αξιοποιεί τον Straight-Through Estimator για την εκπαίδευση αραιών μοντέλων. Αν και το Feather καθώς και άλλες παρόμοιες μέθοδοι προσφέρουν πολύ καλές βάσεις, βασίζονται σε ορισμένες στατικές υπερπαραμέτρους για την κλιμάκωση των κλίσεων και τον προγραμματισμό της αραίωσης, γεγονός που μπορεί να οδηγήσει σε μικρή μείωση της απόδοσης. Η παρούσα εργασία προτείνει δύο συνεισφορές. Η πρώτη συνεισφορά είναι ένας δυναμικός μηχανισμός κλιμάκωσης, εμπνευσμένος από τη στατική υπερπαράμετρο κλιμάκωσης των κλίσεων του Feather που την αντικαθιστά με μία συνάρτηση βασισμένη στη φάση της εκπαίδευσης, επιτρέποντας μεγαλύτερη ροή κλίσεων στα πρώτα στάδια για την αποφυγή πρόωρου κλαδέματος και πιο συντηρητικές ενημερώσεις καθώς αυξάνεται ο λόγος αραίωσης. Η δεύτερη συνεισφορά εισάγει μία οικογένεια προσαρμοστικών συναρτήσεων χρονοπρογραμματισμού κλαδέματος, οι οποίες ρυθμίζουν τον ρυθμό αραίωσης βάσει της σταθερότητας της μάσκας κλαδέματος. Αυτή η προσαρμοστικότητα διασφαλίζει ότι το κλάδεμα γίνεται πιο προσεκτικά όταν οι μάσκες είναι ασταθείς, μειώνοντας την πιθανότητα σημαντικής πτώσης της ακρίβειας. Η αξιολόγηση σε βασικές αρχιτεκτονικές όπως οι ResNet20, DenseNet40-24 και MobileNet V1, εκπαιδευμένες στο CIFAR-100, δείχνει ότι και οι δύο συνεισφορές βελτιώνουν την απόδοση, ιδιαίτερα σε ακραίες ποσοστά αραίωσης. Επιπλέον, η εργασία περιλαμβάνει μία μελέτη σε βάθος για την σημασία των κλαδεμένων βαρών, συγκρίνοντας εκπαίδεύσεις που διατηρούν τις κλαδεμένες συνδέσεις μέσω του STE με εκείνες που τις απορρίπτουν μόνιμα. Πιο συγκεκριμένα, η μελέτη αποσκοπεί στον προσδιορισμό του ποσοστού των βαρών που φέρουν σημασία και εκείνων που μπορούν να αφαιρεθούν οριστικά από τη διαδικασία βελτιστοποίησης, με στόχο την εισαγωγή αραιών κλίσεων στην διαδικασία της αραίωσης χωρίς πτώση της απόδοσης.

**Λέξεις-κλειδιά** — συμπίεση-DNN, αραίωση, κλάδεμα-βαρών, μη-δομημένο-κλάδεμα, αραιή-εκπαίδευση, κλάδεμα-βάσει-μέτρου-μεγέθους

# Abstract

In recent years, deep neural networks have achieved state-of-the-art performance across various and increasingly complicated machine learning tasks. However, in order to achieve such performances the deep neural network models are increasingly complicated and deployment heavy. Model compression methods, particularly pruning, have emerged as effective strategies to address these concerns by eliminating redundant parameters and reducing computational overhead. However, extreme levels of pruning often lead to instability and performance degradation. This thesis addresses these limitations by introducing adaptive pruning strategies that improve performance while maintaining high compression targets. The work tests the proposed strategies on the Feather pruning module, a recent method that utilizes the Straight-Through Estimator to enable gradient-based dense-to-sparse training. Although Feather and many similar modules yield SoA results, they rely on some static hyperparameters for gradient scaling and sparsity scheduling, which may lead to reduced performance. This work proposes two contributions on the basis of those static parameters. The first contribution is a dynamic scaling method inspired by Feather's fixed gradient scaling hyperparameter replacing it with a function based on the training phase's achieved sparsity, allowing larger gradient flow during early iterations to prevent premature pruning and more conservative updates as the sparsity ratio rises. The second contribution introduces an adaptive pruning scheduler function family that adjusts the pruning rate according to the stability of the pruning mask. This adaptiveness ensures that pruning is more cautious when masks become unstable, reducing the likelihood of large accuracy drops. Evaluations on benchmark architectures such as ResNet20, DenseNet40-24, and MobileNet V1, trained on CIFAR-100, show that both contributions enhance performance, especially in extreme sparsity ratios. In addition to these two contributions, the thesis includes an in depth study of the significance of pruned weights, comparing training processes that retain pruned connections via STE against those that permanently remove them. More specifically, the study aims to find what percentage of pruned weights holds significance and what percentage can be permanently dropped from the optimization process, in order to introduce sparse gradients to the sparsification process without loss of accuracy.

**Keywords** – DNN-compression, sparsification, weight-pruning, unstructured-pruning, sparse-training, magnitude-based-pruning

# Ευχαριστίες

Θα ήθελα αρχικά να ευχαριστήσω τον καθηγητή Πέτρο Μαραγκό που από την αρχή των σπουδών μου με ενέπνευσε και εν τέλει μου έδωσε την ευκαιρία να εκπονήσω την διπλωματική μου εργασία στο εργαστήριό του. Επιπλέον, θα ήθελα να ευχαριστήσω τον Δρ. Γιώργο Ρετσινά για την καθοδήγηση και την συνεργασία του που ήταν πολύτιμα για την πορεία και ολοκλήρωση αυτής της διπλωματικής εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου και τα αδέρφια μου για την απόλυτη στήριξή τους όλα αυτά τα χρόνια, καθώς τους φίλους μου και στην σχολή αλλά και εκτός, χάρη στους οποίους τα χρόνια αυτά πέρασαν όμορφα και αξέχαστα.

Αλίκη Ζάχου
Ιούνιος 2025

# Contents

# List of Figures

# List of Tables

# Εκτεταμένη Περίληψη στα Ελληνικά

Contents

# 1 Εισαγωγή

Τα βαθιά νευρωνικά δίκτυα, ενώ αποδίδουν εξαιρετικά σε τομείς όπως η όραση υπολογιστών και η επεξεργασία φυσικής γλώσσας, απαιτούν τεράστιους υπολογιστικούς και ενεργειακούς πόρους [15], [19]. Αυτό καθιστά δύσκολη την ανάπτυξή τους σε συσκευές με περιορισμένες δυνατότητες, όπως κινητά και ρομποτικά συστήματα, ενώ προκαλεί και ανησυχία για το περιβαλλοντικό τους αποτύπωμα [99].

Με αφορμή αυτά τα ζητήματα, έχει δοθεί μεγάλη έμφαση στη συμπίεση των δικτύων, με το κλάδεμα να αποτελεί μια από τις πιο αποτελεσματικές τεχνικές [15], [16], [19], [59]. Μέσω της αφαίρεσης περιττών παραμέτρων, όπως βάρη, νευρώνες, φίλτρα ή και ολόκληρα επίπεδα, επιτυγχάνονται πιο ελαφριά και αποδοτικά μοντέλα χωρίς σημαντική απώλεια απόδοσης. Η παρούσα εργασία προτείνει κάποιες δυναμικές μεθόδους για την αντικατάσταση στατικών παραμέτρων, εξετάζοντας την απόδοσή τους στο πλαίσιο Feather [27], το οποίο κλαδεύει το μοντέλο ταυτόχρονα με την εκπαίδευσή του χρησιμοποιώντας έναν σταθερό μηχανισμό κλιμάκωσης κλίσεων και ένα κυβικό χρονοδιάγραμμα αραιότητας.

Η κύρια συνεισφορά της εργασίας είναι η ανάπτυξη δύο δυναμικών μηχανισμών που αντικαθιστούν τις στατικές παραμέτρους που αναφέρθηκαν. Εισάγεται μια δυναμική συνάρτηση κλιμάκωσης των κλίσεων, η οποία επιτρέπει καλύτερη ροή πληροφορίας κατά την εκπαίδευση, αποφεύγοντας την υπερβολική αποκοπή βαρών στα αρχικά στάδια. Δεύτερον, προτείνεται ένας προσαρμοστικός προγραμματισμός κλαδέματος που ρυθμίζει τον ρυθμό αραίωσης με βάση τη σταθερότητα της μάσκας κλαδέματος. Τα πειραματικά αποτελέσματα δείχνουν ότι οι προτεινόμενες μέθοδοι βελτιώνουν την ακρίβεια και τη σταθερότητα του μοντέλου, ειδικά σε συνθήκες ακραίας αραιότητας. Επιπλέον, η εργασία περιλαμβάνει μια ποσοτική μελέτη για τη σημασία των κλαδεμένων βαρών, προσφέροντας χρήσιμες γνώσεις για το ποια βάρη μπορούν να αφαιρεθούν οριστικά χωρίς επιπτώσεις στην απόδοση, με στόχο την εισαγωγή αραιών κλίσεων στην αραιή εκπαίδευση του μοντέλου.

# 2 Θεωρητικό Υπόβαθρο

## 2.1 Μηχανική Μάθηση

Η μηχανική μάθηση είναι ένα από τα πιο σημαντικά υποπεδία της τεχνητής νοημοσύνης που επιτρέπει στα συστήματα να μαθαίνουν από δεδομένα χωρίς να προγραμματίζονται. Ουσιαστικά τα συστήματα μαθαίνουν να προσαρμόζονται και αποκτούν ικανότητα γενίκευσης από προηγούμενες εμπειρίες, κάτι που επιτρέπει στους αλγορίθμους να αποδίδουν σε νέα, άγνωστα δεδομένα [8], [52].

### Τύποι Μηχανικής Μάθησης

Η μηχανική μάθηση χωρίζεται σε τρεις διαφορετικούς τύπους ανάλογα με τον τρόπο που μαθαίνει το σύστημα. Η επιβλεπόμενη μάθηση εκπαιδεύεται με δεδομένα με ετικέτες, με στόχο το μοντέλο να δημιουργήσει συνδέσεις μεταξύ των δεδομένων και των ετικετών, ώστε να βρίσκει την κατάλληλη ετικέτα σε νέα δεδομένα. Χρησιμοποιείται κυρίως για ταξινόμηση και παλινδρόμηση, με αλγορίθμους όπως τα δέντρα απόφασης [70] και ο αλγόριθμος των πλησιέστερων γειτόνων. Η μη επιβλεπόμενη μάθηση αντίθετα, μαθαίνει από δεδομένα χωρίς ετικέτες και στοχεύει στην ανακάλυψη κρυφών προτύπων ή δομών μέσα σε αυτά, χρησιμοποιώντας αλγορίθμους όπως το k-means [84] και την ιεραρχική ομαδοποίηση [71]. Τέλος, η ενισχυτική μάθηση χαρακτηρίζεται από έναν πράκτορα που αλληλεπιδρά με ένα περιβάλλον και μαθαίνει από ανταμοιβές ή ποινές, ώστε να λαμβάνει διαδοχικές αποφάσεις.

### Κατηγορίες Επιβλεπόμενης Μάθησης

Η επιβλεπόμενη μάθηση χωρίζεται σε δύο κατηγορίες, την παλινδρόμηση και την ταξινόμηση, ανάλογα με την εργασία που πρέπει να πραγματοποιήσει. Η παλινδρόμηση αποτελείται από αλγορίθμους που προβλέπουν συνεχείς αριθμητικές τιμές, όπως η θερμοκρασία ή οι τιμές μετοχών, προσαρμόζοντας μια συνάρτηση στα δεδομένα εκπαίδευσης. Η ταξινόμηση κατηγοριοποιεί εισόδους σε διακριτές κατηγορίες, μοιράζοντας ουσιαστικά τα δεδομένα. Για παράδειγμα, στην ανίχνευση ανεπιθύμητης αλληλογραφίας το μοντέλο μαθαίνει από επισημασμένα παραδείγματα ώστε να προβλέπει εάν ένα μήνυμα είναι ανεπιθύμητο ή όχι.

### Σετ Εκπαίδευσης, Επικύρωσης και Δοκιμής

Για την αξιολόγηση και βελτίωση της απόδοσης ενός μοντέλου μηχανικής μάθησης, το σύνολο δεδομένων διαχωρίζεται συνήθως σε τρία υποσύνολα: το σετ εκπαίδευσης, το σετ επικύρωσης και το σετ δοκιμής. Το σετ εκπαίδευσης χρησιμοποιείται για την εκπαίδευση του μοντέλου, το σετ επικύρωσης βοηθά στη ρύθμιση των υπερπαραμέτρων και στην αποφυγή υπερπροσαρμογής, ενώ το σετ δοκιμής χρησιμοποιείται για την τελική αξιολόγηση του μοντέλου σε άγνωστα δεδομένα.

### Μετρικές Αξιολόγησης

Για την αξιολόγηση της απόδοσης ενός μοντέλου, ειδικά σε προβλήματα ταξινόμησης, μπορούν να χρησιμοποιηθούν τέσσερις βασικές μετρικές. Η ακρίβεια (accuracy) μετρά το συνολικό ποσοστό σωστών προβλέψεων. Η ακρίβεια (precision) εστιάζει στο ποσοστό των πραγματικά θετικών μεταξύ όλων των προβλέψεων θετικών, ενώ η ανάκληση εξετάζει το ποσοστό των πραγματικά θετικών που εντοπίστηκαν μεταξύ όλων των υπαρκτών θετικών. Το F1-score συνδυάζει ακρίβεια και ανάκληση σε μία τιμή μέσω του αρμονικού μέσου, αποτελώντας μια ισορροπημένη μετρική ιδιαιτέρως χρήσιμη σε μη ισορροπημένα σύνολα δεδομένων.

### Συναρτήσεις Σφάλματος

Οι συναρτήσεις σφάλματος είναι πολύ σημαντικές για την εκπαίδευση των μοντέλων, καθώς ποσοτικοποιούν τη διαφορά μεταξύ της προβλεπόμενης και της πραγματικής εξόδου. Στην παλινδρόμηση, η Μέση Τετραγωνική Απόκλιση τιμωρεί περισσότερο τα μεγαλύτερα σφάλματα, ενώ η Μέση Απόλυτη Απόκλιση αντιμετωπίζει όλα τα σφάλματα εξίσου. Για την ταξινόμηση, η απώλεια διασταυρούμενης εντροπίας (cross-entropy loss) είναι ιδιαίτερα διαδεδομένη, καθώς μετρά τη διαφορά μεταξύ της προβλεπόμενης κατανομής πιθανοτήτων και των πραγματικών ετικετών. Η απόκλιση Kullback-Leibler είναι μια άλλη μετρική που ποσοτικοποιεί τη διαφορά μεταξύ δύο κατανομών πιθανοτήτων, κυρίως σε προχωρημένα πιθανοκρατικά μοντέλα.

### Υπερπροσαρμογή και Υποπροσαρμογή

Η υπερπροσαρμογή συμβαίνει όταν ένα μοντέλο είναι υπερβολικά περίπλοκο και μαθαίνει τον θόρυβο των δεδομένων εκπαίδευσης, οδηγώντας σε κακή απόδοση σε νέα δεδομένα. Η υποπροσαρμογή αντιθέτως εμφανίζεται όταν το μοντέλο είναι υπερβολικά απλό και αδυνατεί να εντοπίσει τα βασικά μοτίβα, παρουσιάζοντας χαμηλή απόδοση τόσο στο εκπαιδευτικό όσο και στο δοκιμαστικό σετ. Ένα αποδοτικό μοντέλο βρίσκει την ισορροπία μεταξύ των δύο αυτών άκρων, μαθαίνοντας καλά τα μοτίβα αλλά όχι τον θόρυβο.

### Κανονικοποίηση

Οι τεχνικές κανονικοποίησης χρησιμοποιούνται για τη μείωση της υπερπροσαρμογής, επιβάλλοντας ποινές στην πολυπλοκότητα του μοντέλου. Η κανονικοποίηση $L_1$ προσθέτει ποινή ίση με την απόλυτη τιμή των συντελεστών του μοντέλου, ενθαρρύνοντας την αραιότητα. Η κανονικοποίηση $L2$ τιμωρεί το τετράγωνο των συντελεστών, αποτρέποντας τη δημιουργία μεγάλων βαρών. Το dropout είναι μια άλλη δημοφιλής τεχνική, κυρίως σε νευρωνικά δίκτυα η οποία μηδενίζει τυχαία ένα ποσοστό των νευρώνων, υποχρεώνοντας το δίκτυο να μάθει πιο δυναμικές αναπαραστάσεις και βελτιώνοντας την ικανότητα γενικευσης.

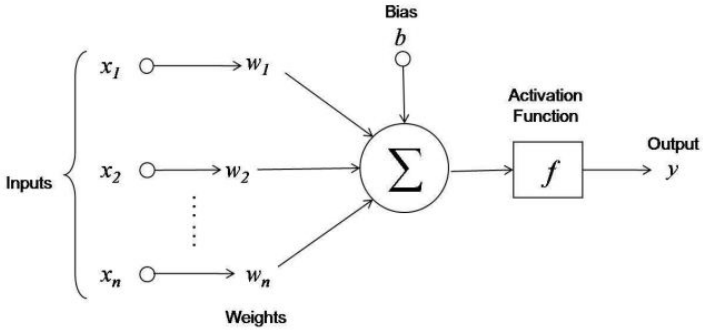
## 2.2 Βαθιά Μάθηση και Νευρωνικά Δίκτυα

Το πεδίο της βαθιάς μάθησης αποτελεί υποσύνολο της μηχανικής μάθησης και βασίζεται στη χρήση νευρωνικών δικτύων πολλαπλών επιπέδων για την επεξεργασία μεγάλων και μη επεξεργασμένων δεδομένων, όπως εικόνες, κείμενο ή ήχος. Αντιθέτως, με την παραδοσιακή μηχανική μάθηση που απαιτεί χειροκίνητη εξαγωγή χαρακτηριστικών, τα βαθιά νευρωνικά δίκτυα μαθαίνουν αυτόματα τα απαραίτητα χαρακτηριστικά κατά τη διάρκεια της εκπαίδευσης [1], [52].

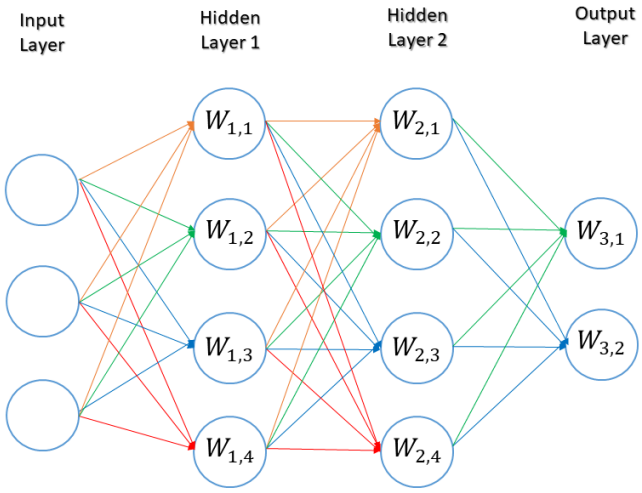
### Το Perceptron και τα FeedForward Νευρωνικά Δίκτυα

Το perceptron είναι η βασική δομική μονάδα των νευρωνικών δικτύων που λειτουργεί ουσιαστικά ως δυαδικός ταξινομητής. Αποτελείται από εισόδους, βάρος, όρο μετατόπισης και μια συνάρτηση ενεργοποίησης. Κατά την εκπαίδευση, τα βάρη ενημερώνονται ώστε να μειωθούν τα σφάλματα ταξινόμησης. Υπάρχουν διάφορες

συναρτήσεις ενεργοποίησης, όπως η ReLU, η Sigmoid, η Tanh και η Softmax. Συνδυάζοντας πολλαπλά perceptron δημιουργούνται πολυεπίπεδα δίκτυα, γνωστά ως MLPs ή FNNs. Όταν κάθε νευρώνας συνδέεται με όλους του επόμενου επιπέδου, έχουμε τα πλήρως συνδεδεμένα δίκτυα (FCNNs), ενώ τα συνελικτικά νευρωνικά δίκτυα (ΝNs) αποτελούν εξειδίκευση των FNNs για εικόνες, τα οποία εξετάζονται αργότερα αφού παρουσιαστούν πρώτα οι βασικές έννοιες εκπαίδευσης.




Σχήμα 1: Οπτικοποίηση νευρώνα περσέπτον που χρησιμοποιείται στα νευρωνικά δίκτυα. Από [92]




Σχήμα 2: Παράδειγμα αρχιτεκτονικής πλήρως συνδεδεμένου νευρωνικού δικτύου. Από [55]

### Οπισθοδιάδοση

Η εκπαίδευση των νευρωνικών δικτύων βασίζεται σε υπολογισμό παραγώγων της συνάρτησης σφάλματος ως προς τα βάρη του μοντέλου. Η μέθοδος που χρησιμοποιείται για τον υπολογισμό αυτών των παραγώγων ονομάζεται οπισθοδιάδοση και βασίζεται στη χρήση του κανόνα της αλυσίδας. Η εκπαίδευση περιλαμβάνει δύο φάσεις, τη φάση εμπρόσθιας διάδοσης, όπου υπολογίζεται η έξοδος του δικτύου, και τη φάση οπισθοδρόμησης, όπου υπολογίζονται οι παράγωγοι της απώλειας και διανέμονται προς τα πίσω για την ενημέρωση των βαρών, με στόχο την ελαχιστοποίηση του σφάλματος.

### Βελτιστοποιητές

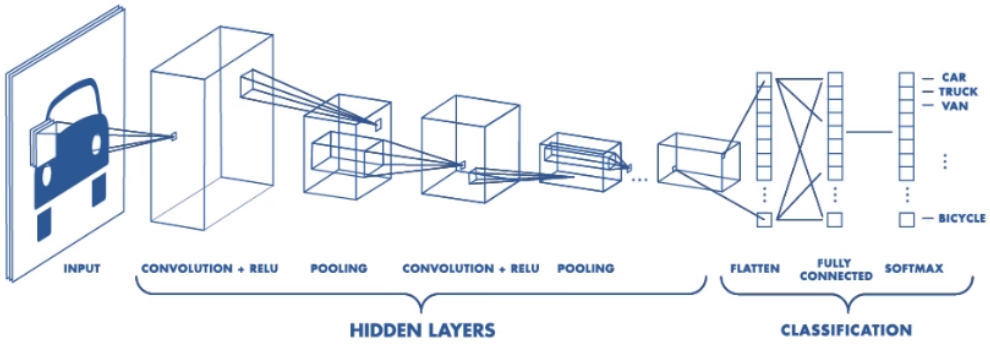
Οι βελτιστοποιητές είναι αλγόριθμοι που προσαρμόζουν τις παραμέτρους του μοντέλου με σκοπό τη μείωση της συνάρτησης κόστους. Οι πιο συνηθισμένοι είναι παραλλαγές της καθοδικής κλίσης (gradient descent). Η βασική της μορφή είναι η απλή καθοδική κλίση, η οποία είναι αργή και εξαρτάται έντονα από τον ρυθμό μάθησης. Η στοχαστική εκδοχή υπολογίζει την κλίση ανά δείγμα, προσφέροντας ταχύτητα και τυχαιότητα.

Η mini-batch εκδοχή εξισορροπεί τις προηγούμενες μεθόδους, υπολογίζοντας την κλίση σε μικρά πακέτα δεδομένων αντί να τα χρησιμοποιεί όλα. Η προσθήκη ορμής στην στοχαστική εκδοχή βοηθά στη σταθερότητα και ταχύτερη σύγκλιση. Οι Adagrad και RMSprop εισάγουν προσαρμοστικούς ρυθμούς μάθησης, ενώ ο Adam συνδυάζει τα πλεονεκτήματα των μεθόδων ορμής και προσαρμογής, καθιστώντας τον έναν από τους πιο διαδεδομένους αλγόριθμους στη βαθιά μάθηση.

### Αρχιτεκτονική Συνελικτικών Νευρωνικών Δικτύων

Τα συνελικτικά νευρωνικά δίκτυα (CNNs) είναι αρχιτεκτονικές ειδικά σχεδιασμένες για επεξεργασία και ταξινόμιση εικόνας. Δέχονται ως είσοδο εικόνες με διαστάσεις $W \times H \times D$ και εξάγουν προβλέψεις για τις αντίστοιχες κατηγορίες που μπορεί να ανήκουν. Κάθε επίπεδο του CNN έχει ξεχωριστό ρόλο και μπορεί να είναι συνελικτικό, υποδειγματοληψίας (pooling), κανονικοποίησης πακέτων (batch normalization) ή πλήρως συνδεδεμένο (fully connected). Αυτά τα επίπεδα λειτουργούν μαζί για να εξαγάγουν και να επεξεργαστούν χαρακτηριστικά της εικόνας, οδηγώντας στην κατανόηση του περιεχομένου της και σε προβλέψεις.

Τα συνελικτικά επίπεδα είναι τα βασικότερα των CNNs, υπεύθυνα για την εξαγωγή χαρακτηριστικών μέσω εφαρμογής φίλτρων στην είσοδο και παράγουν χάρτες χαρακτηριστικών. Τα φίλτρα έχουν διαστάσεις $F \times F$ και είναι υπεύθυνα για την αναγνώριση διαφόρων χαρακτηριστικών της εικόνας. Τα επίπεδα υποδειγματοληψίας μειώνουν τις διαστάσεις των χαρτών χαρακτηριστικών ανάμεσα σε δύο συνελικτικά επίπεδα, μειώνοντας έτσι τον αριθμό των παραμέτρων και τον υπολογιστικό φόρτο. Το επίπεδο κανονικοποίσης πακέτων κανονικοποιεί τις εισόδους έτσι ώστε να μην απέχουην πολύ οι κατανομές των βαρών ανά επίπεδο. Τέλος, το πλήρες συνδεδεμένο επίπεδο βρίσκεται στην έξοδο του δικτύου και πραγματοποιεί την πρόβλεψη κατηγορίας της εικόνας, αφού έχει λάβει ως είσοδο τα εξαγόμενα χαρακτηριστικά της.




Σχήμα 3: Παράδειγμα αρχιτεκτονικής συνελικτικού νευρωνικού δικτύου. Από [100]

# 3 Βιβλιογραφική Επισκόπηση

Τα τελευταία χρόνια, με τη συνεχή πρόοδο των βαθιών νευρωνικών δικτύων, η τεχνητή νοημοσύνη έχει σημειώσει σημαντικά άλματα σε διάφορους τομείς όπως η Υπολογιστική Όραση [34], [49], η Επεξεργασία Φυσικής Γλώσσας [21], η Αναγνώριση Ομιλίας [37] και οι πολυτροπικές εφαρμογές [60], [76]. Ωστόσο, η εξαιρετική απόδοση αυτών των εφαρμογών απαιτεί δίκτυα με εκατομμύρια έως και δισεκατομμύρια παραμέτρους, γεγονός που οδηγεί σε υψηλό υπολογιστικό κόστος [19]. Αυτό συνεπάγεται με αυξημένες απαιτήσεις σε μνήμη, επεξεργαστική ισχύ και κατανάλωση ενέργειας, καθιστώντας δύσκολη την υλοποίησή τους σε συστήματα με περιορισμένους πόρους, όπως κινητές συσκευές, ενσωματωμένα συστήματα και ρομπότ.

Για την αντιμετώπιση αυτών των προκλήσεων, έχει αναπτυχθεί σημαντική έρευνα πάνω στη συμπίεση και επιτάχυνση νευρωνικών δικτύων [15], [16], [19], [59], με στόχο τη μείωση του μεγέθους τους χωρίς σημαντική απώλεια απόδοσης. Αυτό διευκολύνει την ανάπτυξη σε περιορισμένα περιβάλλοντα, ενώ παράλληλα μπορεί να ενισχύσει την ευρωστία και τη γενίκευση.

## 3.1 Σχετικές Μέθοδοι Συμπίεσης

Η ενότητα αυτή παρουσιάζει τις κυριότερες μεθόδους συμπίεσης νευρωνικών δικτύων εκτός του κλαδέματος.

### Κβαντοποίηση

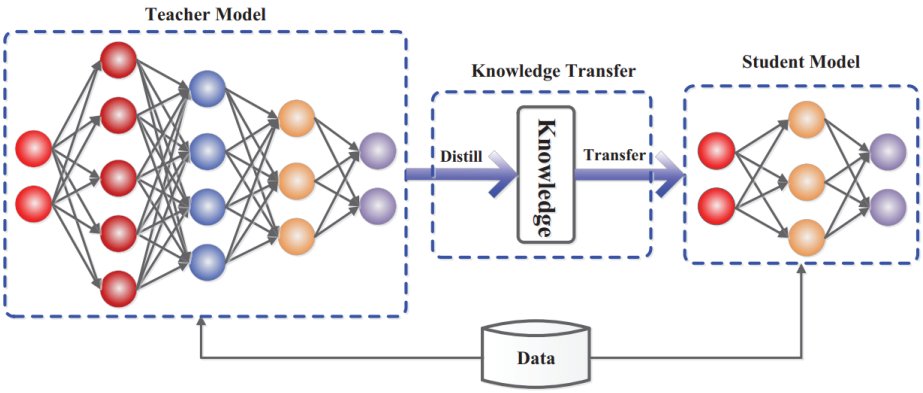
Κβαντοποίηση είναι η χαρτογράφηση συνεχών τιμών σε διακριτές αναπαραστάσεις, συνήθως ακέραιες. Στα νευρωνικά δίκτυα, η τεχνική αυτή επιτρέπει μείωση της υπολογιστικής πολυπλοκότητας και της κατανάλωσης ενέργειας, διατηρώντας ικανοποιητική απόδοση [59]. Η αριθμητική κβαντοποίηση χαμηλών bit περιλαμβάνει την απευθείας ποσοτικοποίηση των βαρών ενός δικτύου από 32- ή 16-bit float τιμές σε 8-, 4-, 2-bit ή ακόμη και 1-bit ακέραιες τιμές [42], [88]. Η συσταδοποίηση και η κοινή χρήση βαρών επιτυγχάνεται μέσω αλγορίθμων ομαδοποίησης όπως το k-means [84], όπου παρόμοια βάρη ομαδοποιούνται και αντικαθίστανται από μια κοινή τιμή.

### Αποσύνθεση Τανυστών

Η αποσύνθεση τανυστών στοχεύει στη διάσπαση μεγάλων πινάκων βαρών σε μικρότερες και χαμηλής τάξης αναπαραστάσεις [83], μειώνοντας παραμέτρους και FLOPs. Οι κύριες τεχνικές είναι η SVD [20], [73], η αποσύνθεση Tucker [47], [57], η αποσύνθεση CP [51] και το Dictionary Learning [79].

### Απόσταξη Γνώσης

Η απόσταξη γνώσης είναι τεχνική συμπίεσης κατά την οποία ένα μικρότερο δίκτυο (μαθητής) εκπαιδεύεται να μιμείται την έξοδο ενός μεγαλύτερου προκαθορισμένου μοντέλου (δάσκαλος) [11], [29], [38]. Η βασική τεχνική παρουσιάζεται στην Εικόνα 4.




Σχήμα 4: Οπτικοποίηση απόσταξης γνώσης με το μοντέλο-δάσκαλο και το μοντέλο-μαθητή. Από [29]

### Σχεδιασμός Συμπαγών Μοντέλων και Διερεύνηση Δικτύων

Αντί για συμπίεση μεγάλων δικτύων, μια εναλλακτική είναι ο σχεδιασμός μικρών, αποδοτικών αρχιτεκτονικών για χρήση σε συστήματα με περιορισμένους πόρους. Τα MobileNets [40] και DenseNets [41] είναι χαρακτηριστικά παραδείγματα.

Η διερεύνηση αρχιτεκτονικών (neural architecture search) [77] είναι μια αυτοματοποιημένη διαδικασία εύρεσης βέλτιστων αρχιτεκτονικών, σχεδιασμένων να ανταποκρίνονται σε συγκεκριμένους περιορισμούς. Στο πλαίσιο της συμπίεσης, δημιουργούνται αποδοτικά και συμπιεσμένα μοντέλα από το μηδέν.

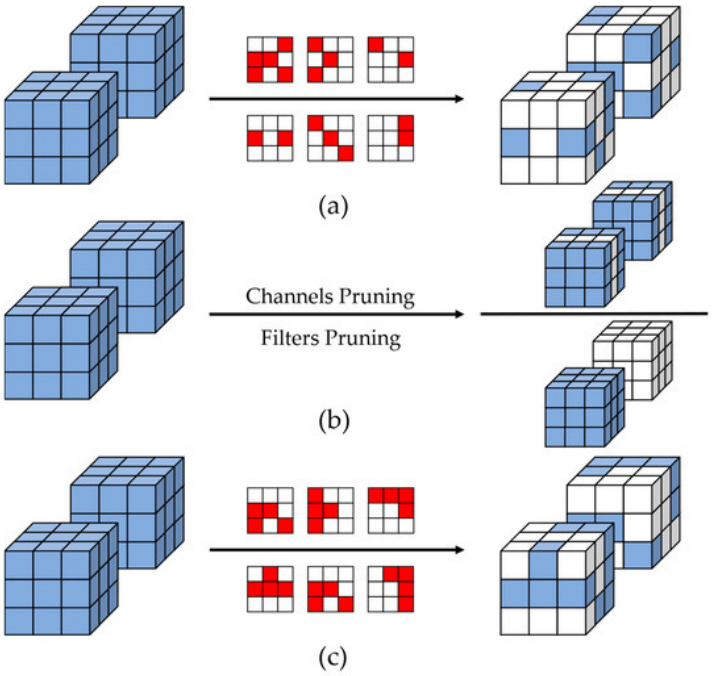
## 3.2 Κλάδεμα Νευρωνικών Δικτύων

Το κλάδεμα στα νευρωνικά δίκτυα είναι μια τεχνική συμπίεσης μοντέλων που αποσκοπεί στη μείωση του μεγέθους και της πολυπλοκότητας ενός δικτύου, αφαιρώντας στοιχεία που θεωρούνται λιγότερο σημαντικά

ή πλεονάζοντα. Από τις πρώτες σχετικές εργασίες είναι οι "Optimal Brain Damage" [53] και "Optimal Brain Surgeon" [33]. Η τεχνική αυτή σήμερα παραμένει στο επίκεντρο της έρευνας, λόγω της συνεχούς αύξησης των παραμέτρων στα σύγχρονα δίκτυα [15].

### Μέθοδοι Κλαδέματος

Οι μέθοδοι κλάδεμα διακρίνονται ανάλογα με τη λεπτομέρεια της αραίωσης σε μη-δομημένο, δομημένο και ημι-δομημένο [15]. Το μη-δομημένο αφαιρεί μεμονωμένα βάρη, το δομημένο αφαιρεί ολόκληρες δομές όπως φίλτρα ή κανάλια, ενώ το ημι-δομημένο προσπαθεί να συνδυάσει τα πλεονεκτήματα και των δύο. Το Σχήμα 5 οπτικοποιεί τις διαφορές μεταξύ τους [86].




Σχήμα 5: Οπτικοποίηση μεθόδων κλαδέματος σε ένα συνελικτικό νευρωνικό δίκτυο: **(α)** Μη-δομημένο Κλάδεμα **(β)** Δομημένο Κλάδεμα **(γ)** Ημι-δομημένο Κλάδεμα. Από [86]

Το μη-δομημένο κλάδεμα αφαιρεί μεμονωμένα βάρη ανεξαρτήτως θέσης [15], δημιουργώντας αραιές αλλά ακανόνιστες δομές [25]. Παρότι επιτυγχάνει υψηλή αραίωση χωρίς σημαντική απώλεια ακρίβειας, δεν μπορεί να αξιοποιηθεί εύκολα από συμβατικό υλικό χωρίς εξειδικευμένες βιβλιοθήκες [96]. Το δομημένο κλάδεμα αφαιρεί δομές όπως νευρώνες [64], φίλτρα [101] και κανάλια [62], διατηρώντας ομαλή αρχιτεκτονική που ευνοεί την επιτάχυνση σε απλό υλικό [36]. Το ημι-δομημένο κλάδεμα [81], [96] προσπαθεί να ισορροπήσει μεταξύ των άλλων δύο μεθόδων.

### Κριτήρια Κλαδέματος

Τα κριτήρια κλάδεμα καθορίζουν ποια βάρη, φίλτρα ή δομές αφαιρούνται [15], [96].

Το πιο διαδεδομένο κριτήριο είναι το μέγεθος, και εξετάζει το απόλυτο μέγεθος των βαρών. Τα βάρη με χαμηλή τιμή θεωρούνται λιγότερο σημαντικά και αφαιρούνται [32].

Ένα άλλο βασικό κριτήριο είναι η ευαισθησία, βάσει της οποίας χρησιμοποιούνται παράγωγοι δευτέρου βαθμού για να βρεθεί η σημασία των βαρών στην συνάρτηση σφάλματος. Αν και πιο ακριβείς, είναι υπολογιστικά δαπανηρές, κάτι που στην αρχή δεν αποτελούσε πρόβλημα εφόσον τα νευρωνικά δίκτυα ήταν ακόμα ρηχά. Πλέον με τα τεράστια δίκτυα προτιμώνται μέθοδοι που προσεγγίζουν τις παραγώγους αντί να τις υπολογίζουν ευθέως [85], [94].

Το κλάδεμα με βάση την εξομάλυνση είναι μία τρίτη κατηγορία κριτηρίων κλαδέματος, κατά την οποία η αραίωση εισάγεται πλαγίως αντί ευθέως όπως στις άλλες περιπτώσεις με την εισαγωγή ενός όρου εξομάλυνσης ($L_0, L_1, L_2$ νόρμες) στην συνάρτηση σφάλματος [31], [66], [98].

Τέλος, το τυχαίο κλάδεμα χρησιμοποιείται ως μέθοδος ελέγχου, χωρίς κριτήριο επιλογής έτσι ώστε να συγκριθούν μαζί του πιο εξειδικευμένες μέθοδοι και να αποδείξουν ότι έχουν βάση [9], [24].

### Χρονικό Πλαίσιο Κλαδέματος

Το χρονικό πλαίσιο καθορίζει το πότε εφαρμόζεται το κλάδεμα σε σχέση με την εκπαίδευση, πριν, κατά τη διάρκεια ή μετά. Η πιο ενστικτώδης εκδοχή είναι το κλάδεμα να γίνει μετά το πέρας της εκπαίδευσης, εφόσον θα έχουν ήδη βρεθεί τα πιο σημαντικά βάρη [31], [33], [53].Το κλάδεμα πριν την εκπαίδευση εκπαιδεύει ήδη αραιά δίκτυα [24], [54], [91], [95]. Το κλάδεμα κατά τη διάρκεια της εκπαίδευσης είναι πλέον από τις πιο διαδεδομένες προσεγγίσεις, κατά την οποία το επιθυμητό ποσοστό αραίωσης εισάγεται σταδιακά [4], [27], [89], [93], [104], [105].

### Συνδυασμός Κλαδέματος με Άλλες Μεθόδους Συμπίεσης

Αν και το κλάδεμα είναι μια από τις πιο διαδεδομένες τεχνικές συμπίεσης, τα τελευταία χρόνια διερευνώνται τρόποι ενίσχυσής του μέσω συνδυασμού με άλλες τεχνικές συμπίεσης [15], [36].

Ο πιο συχνός συνδυασμός στη βιβλιογραφία είναι του κλαδέματος και της κβαντοποίησης. Το κλάδεμα αφαιρεί μη απαραίτητες συνδέσεις ή δομές, ενώ η κβαντοποίηση μειώνει την ακρίβεια αναπαράστασης των απομένοντων βαρών [31]. Το κλαδεμα σε συνδυασμό με την αποσύνθεση τανυστών εξετάζει ταυτόχρονα τον πλεονασμό σε βάρη και δομές. Το κλάδεμα αφαιρεί άχρηστες συνδέσεις, ενώ η αποσύνθεση τανυστών μειώνει τις διαστάσεις των βαρών [57]. Ο συνδυασμός κλαδέματος με απόσταξη γνώσης χρησιμοποιείται για να μετριάσει την απώλεια απόδοσης σε υψηλές αραιώσεις. Η απόσταξη γνώσης επιτρέπει στο μικρότερο κλαδεμένο μοντέλο να μάθει από το πλήρες [13]. Τέλος, η διερεύνηση αρχιτεκτονικών σε συνδυασμό με κλάδεμα στοχεύει στην σχεδίαση αποδοτικών αρχιτεκτονικών. Αναζητούνται αρχιτεκτονικές με περιορισμούς αραιότητας και εκτελείται το κλάδεμα κατά την αναζήτηση, στοχεύοντας σε αποδοτικά μοντέλα ανάλογα με τις απαιτήσεις του υλικού [12].

## 3.3 Το Feather και Άλλες Παρόμοιες Μέθοδοι

### Αδόμητο Κλάδεμα από Πυκνό σε Αραιό βάσει Μεγέθους Βάρους

Το GMP [105] αποτελεί μια από τις βασικές μεθόδους αφαίρεσης βαρών με μικρή απόλυτη τιμή σε νευρωνικά δίκτυα, εφαρμόζοντας σταδιακά έναν σκληρό τελεστή κατωφλίου σε προκαθορισμένα σημεία κατά την εκπαίδευση, με στόχο μια ομαλή μετάβαση από πυκνό σε αραιό μοντέλο. Η μη διαφορίσιμη φύση του όμως και η σταθερή προγραμματισμένη αφαίρεση βαρών μπορεί να περιορίσουν την προσαρμοστικότητά του στις δυναμικές της εκπαίδευσης.

Το STR [50] αντιμετωπίζει αυτούς τους περιορισμούς εισάγοντας ένα διαφορίσημο απαλό κατώφλι που επιτρέπει στα βάρη να συρρικνώνονται ομαλά προς το μηδέν, αντί να αφαιρούνται απότομα. Αυτό επιτρέπει στο δίκτυο να προσαρμόζεται πιο ευέλικτα και να μαθαίνει καλύτερα ποια βάρη πρέπει να διατηρηθούν, οδηγώντας σε πιο σταθερή εκπαίδευση και καλύτερη διατήρηση της ακρίβειας.

Το ST-3 [93] βελτιώνει περαιτέρω το STR με τη χρήση ενός Straight-Through Estimator [7], που επιτρέπει τη ροή των βαθμίδων κατά την οπισθοδιάδοση ακόμα και μέσω μη διαφορίσιμων σημείων, καθώς και με έναν μηχανισμό δυναμικής επανακλιμάκωσης των βαρών. Αυτές οι βελτιώσεις εξασφαλίζουν πιο αποτελεσματική και σταθερή μάθηση, επιτρέποντας την εκμάθηση πιο αραιών μοντέλων χωρίς σημαντική απώλεια ακρίβειας.

Το Feather [27], το οποίο αποτελεί αντικείμενο της εργασίας, βασίζεται στον τρόπο κλαδέματος του GMP και του ST-3 αλλά εισάγει καινοτομίες στον τρόπο προγραμματισμού της αφαίρεσης και στην κλιμάκωση των βαθμίδων, βελτιώνοντας τη σταθερότητα της εκπαίδευσης ακόμη και σε περιπτώσεις εξαιρετικά υψηλής αραίωσης.

Το Spartan [89], σε αντίθεση με τις άλλες μεθόδους, προσεγγίζει το πρόβλημα της αφαίρεσης βαρών με μια θεωρητικά θεμελιωμένη μέθοδο βασισμένη στη θεωρία βέλτιστης μεταφοράς. Ορίζει την αραιότητα

μέσω ενός κανονικοποιημένου προβλήματος βέλτιστης μεταφοράς και χρησιμοποιεί το διπλό μέσο για τη σταθεροποίηση της μάθησης υπό περιορισμούς αραίωσης.

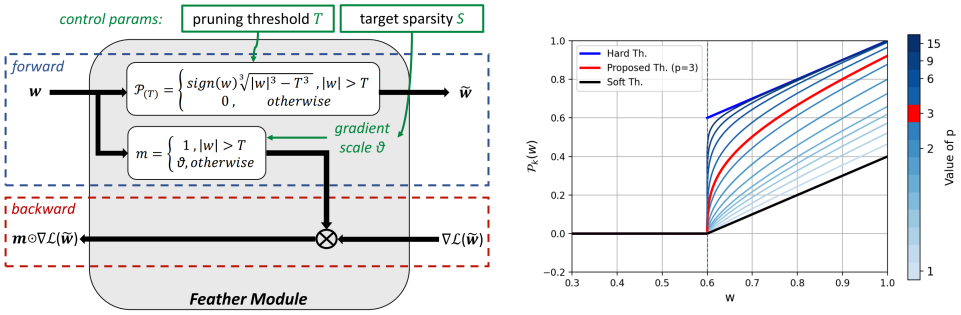
Η MAP [4] είναι μια δυναμική μέθοδος αραιής εκπαίδευσης που χρησιμοποιεί προσοχή βασισμένη στο μέτρο των βαρών. Αντί για δυαδική μάσκα, εφαρμόζει συνεχείς τιμές προσοχής ώστε να ενισχύει σημαντικές συνδέσεις και να επιτρέπει προσαρμογές στις λιγότερο σημαντικές. Η εκπαίδευση χωρίζεται σε φάση εξερεύνησης και φάση εκμετάλλευσης, οδηγώντας σε υψηλή απόδοση ακόμη και σε ακραία επίπεδα αραίωσης.

### Feather

Το Feather είναι μία μη-δομημένη, βασισμένη στο μέγεθος τεχνική αραίωσης, η οποία εφαρμόζει κλάδεμα βαρών κατά τη διάρκεια της εκπαίδευσης. Το Feather ενσωματώνει μια βελτιωμένη εκδοχή του Straight-Through Estimator (STE), επιτρέποντας τη ροή των κλίσεων μέσα από τα κλαδεμένα βάρη κατά την οπισθοδιάδοση, αντιμετωπίζοντας τον τελεστή κατωφλίου ως την ταυτοτική συνάρτηση.

Ο μηχανισμός αυτός επιτρέπει στο δίκτυο να διατηρεί τα πλεονεκτήματα ενός πυκνού μοντέλου ενώ ταυτόχρονα προοδευτικά αποκτά αραιότητα. Η διαδικασία κλαδέματος καθοδηγείται από έναν κυβικό χρονοπρογραμματιστή που αυξάνει σταδιακά το επίπεδο αραιότητας κατά την εκπαίδευση. Ο τελεστής κατωφλίου δεν είναι ούτε πλήρως σκληρός ούτε πλήρως μαλακός, αλλά επιτυγχάνει μία ισορροπία μέσω της παραμέτρου $p = 3$, επιτρέποντας ομαλότερη δυναμική αραιότητας.

Για την αντιμετώπιση της αστάθειας της μάσκας σε υψηλά ποσοστά αραιότητας (98% ή 99%), το Feather εισάγει μηχανισμό κλιμάκωσης των κλίσεων, όπου οι κλίσεις των κλαδεμένων βαρών πολλαπλασιάζονται με έναν σταθερό παράγοντα $\theta$ κατά την οπισθοδιάδοση. Η τιμή του $\theta$ παραμένει σταθερή καθ' όλη τη διάρκεια της εκπαίδευσης, συνήθως ορίζεται ως 1 για μέτριες αραιώσεις και ως 0.5 για ακραίες, αν και αυτή η στατική προσέγγιση έχει περιορισμούς που εξετάζονται σε επόμενα κεφάλαια της εργασίας.




(a) (b)

Σχήμα 6: Η μέθοδος κλαδέματος Feather. Από [27]

# 4 Συνάρτηση Κλιμάκωσης Κλίσεων

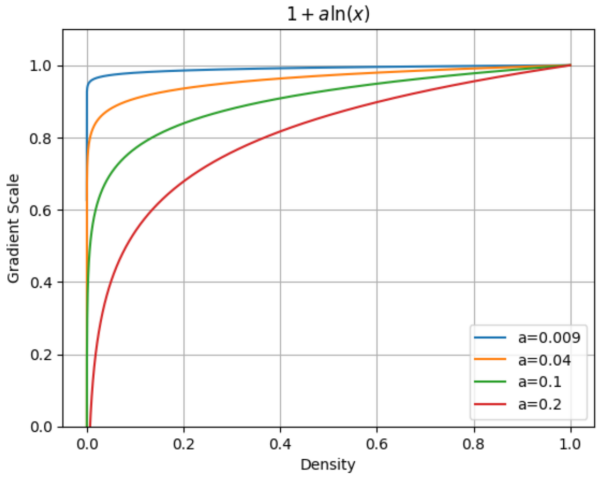
Η τεχνική αραίωσης Feather προτείνει τη χρήση της βαθμωτής παραμέτρου $\theta$ για να σταθεροποιήσει τις μάσκες κλαδέματος όσο περισσότερο αυξάνεται το ποσοστό αραίωσης, εφόσον τότε αποσταθεροποιείται η μάσκα περισσότερο. Η παράμετρος αυτή επηρεάζει τη ροή των κλίσεων (gradients) στα κλαδεμένα βάρη. Ωστόσο, η στατική προσέγγιση της επιλογής τιμής $\theta$ (είτε 1 είτε 0.5 ανάλογα με τον στόχο αραίωσης) περιορίζει τη γενίκευση και μπορεί να οδηγήσει σε υποπροσαρμογή. Το κεφάλαιο αυτό προτείνει μια δυναμική συνάρτηση που υπολογίζει το $\theta$ κατά τη διάρκεια της εκπαίδευσης, ώστε να επιτυγχάνεται βελτιωμένη ακρίβεια.

## 4.1 Προτεινόμενη Μέθοδος

### Επιλογή Συνάρτησης

Προτείνεται η χρήση της συνάρτησης $f(x) = 1 + \alpha \cdot \ln(x)$, όπου $x$ είναι η εκάστοτε πυκνότητα του μοντέλου ή επιπέδου του δικτύου. Αυτή η επιλογή γίνεται ώστε το $\theta$ να ξεκινάει από μεγαλύτερες τιμές στα αρχικά

στάδια εκπαίδευσης (όπου η πυκνότητα είναι υψηλή) και να μειώνεται σταδιακά καθώς αυξάνεται η αραιότητα. Με αυτόν τον τρόπο διευκολύνεται η ροή πληροφοριών όταν το δίκτυο είναι ακόμα πυκνό, ενώ περιορίζεται όταν η μάσκα σταθεροποιείται.



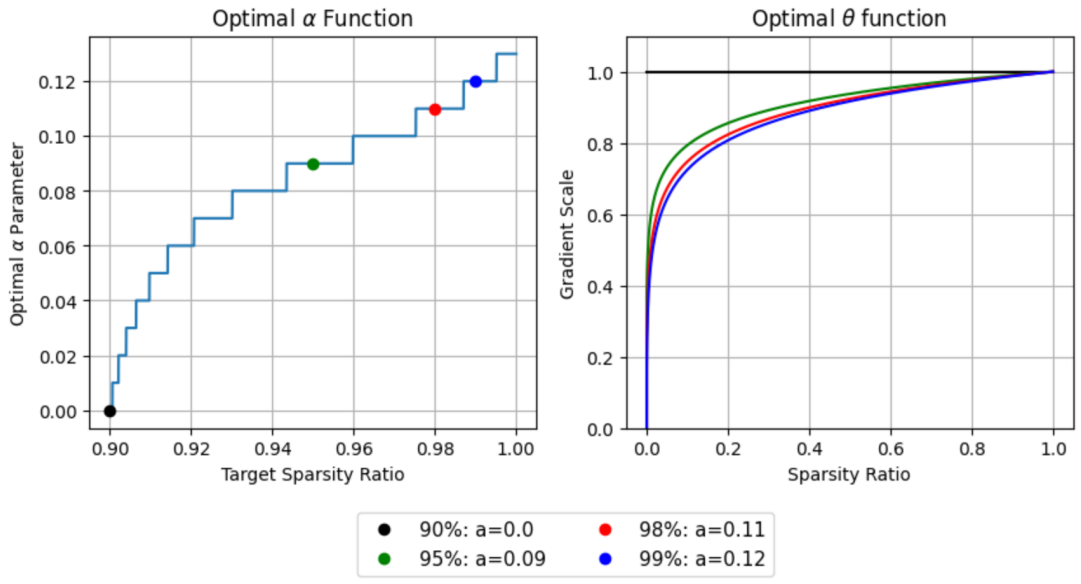

Σχήμα 7: Συνάρτηση $1 + \alpha \cdot \ln(x)$ για διαφορετικά $\alpha$

### Επιλογή της Υπερπαραμέτρου $\alpha$

Για να καθοριστεί η παράμετρος $\alpha$ η οποία επηρεάζει τον ρυθμό μεταβολής του $\theta$, διενεργήθηκε εμπειρική αξιολόγηση σε τέσσερις διαφορετικούς στόχους αραιότητας. Όσο μεγαλύτερος είναι ο στόχος, τόσο πιο γρήγορα πρέπει να μειώνεται το $\theta$. Προέκυψε ότι η βέλτιστη τιμή του είναι συνάρτηση του στόχου αραιότητας και προσεγγίζεται με μια τριγωνομετρική συνάρτηση:

$$\alpha(S) = 0.026 \cdot \tan(23.09 \cdot S + 22.08) + 0.093$$

Η εξίσωση αυτή επιτρέπει τη γενίκευση της μεθόδου και την εφαρμογή της και σε ενδιάμεσες ή μη δοκιμασμένες τιμές αραιότητας.




Σχήμα 8: Ιδανική συνάρτηση για το $\alpha$ με τις ιδανικές τιμές για κάθε στόχο αραίωσης σημειωμένες και με τις παραγόμενες συναρτήσεις $f(x)$ δίπλα

### Παγκόσμια και Ανά Επίπεδο Προσεγγίσεις

Η συνάρτηση $\theta$ μπορεί να εφαρμοστεί με δύο τρόπους:

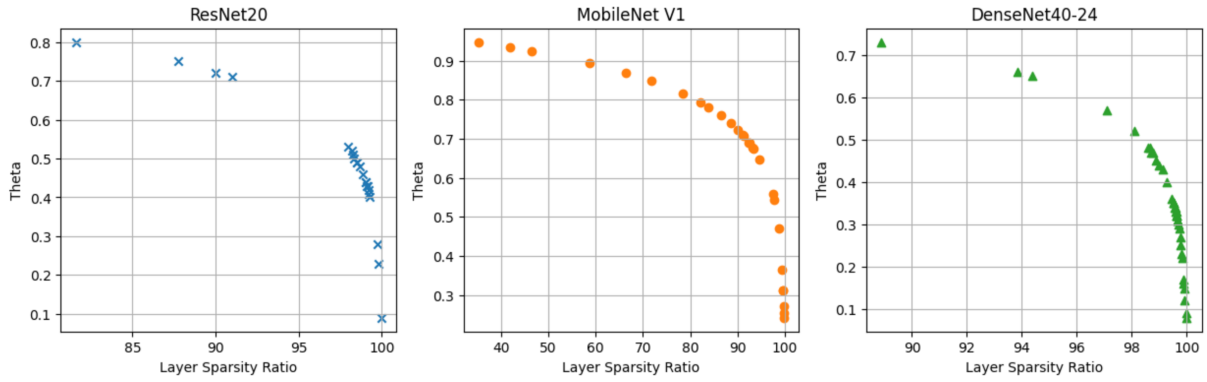
- Παγκόσμια (Global): υπολογίζεται ένας μέσος όρος $\theta$ με βάση τις αραιότητες που επιτυγχάνει κάθε επίπεδο και εφαρμόζεται σε όλα τα επίπεδα.
- Ανά επίπεδο (Layer-wise): κάθε επίπεδο έχει το δικό του $\theta$, υπολογιζόμενο από τη δική του πυκνότητα.

Η παγκόσμια προσέγγιση είναι πιο απλή και λιγότερο επιρρεπής σε υπερπροσαρμογή, ενώ η προσέγγιση ανά επίπεδο επιτρέπει μεγαλύτερη ευελιξία και καλύτερη αξιοποίηση της τοπικής πληροφορίας.

## 4.2 Πειραματική Αξιολόγηση

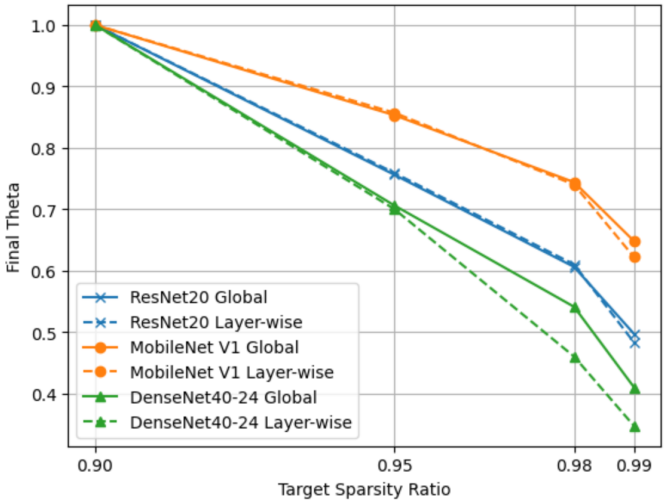
### Σύγκριση Παγκόσμιας και Ανά Επίπεδο Προσέγγισης

Η σύγκριση των δύο προσεγγίσεων πραγματοποιήθηκε σε τρία διαφορετικά μοντέλα (ResNet20, MobileNet V1, DenseNet40-24). Παρατηρήθηκε ότι σε μέτριες αραιότητες η απόδοση ήταν παρόμοια, ωστόσο σε ακραία ποσοστά αραιότητας η παγκόσμια μέθοδος ξεπέρασε την ανά επίπεδο. Αυτό αποδίδεται στο γεγονός ότι η ανά επίπεδο εφαρμογή μπορεί να παράγει πολύ μικρά $\theta$ σε ορισμένα επίπεδα, οδηγώντας σε ανεπαρκή οπισθοδιάδοση κλίσεων και υποεκπαίδευση. Όπως φαίνεται στην Εικόνα 9, κάποια επίπεδα λαμβάνουν τελική τιμή για το $\theta$ σχεδόν 0.05, κάτι το οποίο σημαίνει ότι οι κλίσεις κατά την οπισθοδιάδοση σχεδόν μηδενίζονται, με αποτέλεσμα να μην ανανεώνονται τα βάρη καν. Αυτό έχει ως αποτέλεσμα την υποεκπαίδευση των μοντέλων σε πολύ υψηλά ποσοστά αραιότητας, καθώς όλο και περισσότερα επίπεδα αποκτούν πολύ χαμηλά βάρη τα οποία κλαδεύονται σε πολύ μεγάλο ποσοστό. Πιθανώς αυτό το πρόβλημα να είναι περιορισμός της συνάρτησής μας και μία πιο προσεκτικά σχεδιασμένη συνάρτηση να λύνει αυτό το πρόβλημα.



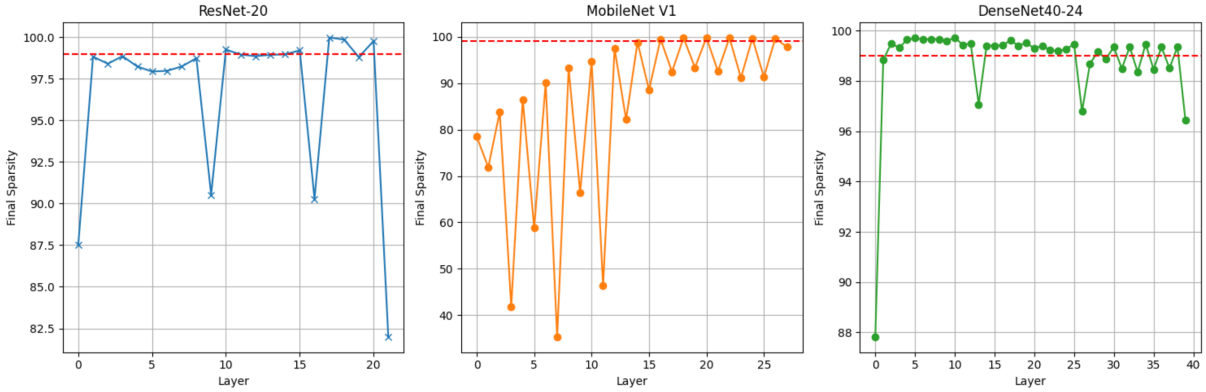

Σχήμα 9: Τελικές τιμές του $\theta$ και τελικό ποσοστό αραιότητας για κάθε επίπεδο κάθε μοντέλου με στόχο 99% χρησιμοποιώντας την ανά επίπεδο προσέγγιση

Η μέθοδος της παγκόσμιας ανανέωσης $\theta$ φαίνεται να λύνει αυτό το πρόβλημα, καθώς λαμβάνονται υπόψιν όλες οι μεγάλες μεταβολές μεταξύ των επιπέδων, δίνοντας έτσι μία καθολική τιμή που φέρνει ισορροπία. Στην Εικόνα 10 φαίνεται το μέσο $\theta$ και για τις δύο προσεγγίσεις. Ενώ για χαμηλούς στόχους αραίωσης φαίνεται να παραμένουν οι τιμές κοντά, για υψηλά ποσοστά είναι ξεκάθαρο ότι διατηρείται λίγο υψηλότερη η μέση τιμή, κάτι που αποδίδεται στο γεγονός ότι πλέον πραγματοποιείται η σωστή αραίωση των επιπέδων.

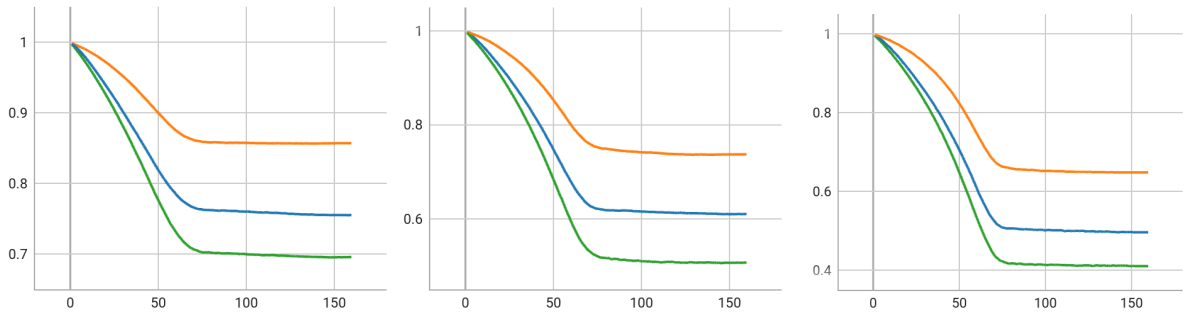



Σχήμα 10: Μέσο $\theta$ και για τις δύο προσεγγίσεις για όλα τα μοντέλα και τα επιθυμητά ποσοστά αραίωσης

Στην Εικόνα 12 φαίνονται οι τιμές $\theta$ που αποκτούν όλα τα μοντέλα ανά εποχή εκπαίδευσης για όλους τους στόχους. Είναι εμφανές ότι φτάνουν διαφορετικές τελικές τιμές, γεγονός που βασίζεται στη διαφορά μεταξύ των αραιοτήτων που καταφέρνει να επιτύχει κάθε επίπεδο, όπως φαίνεται στην Εικόνα 11.




Σχήμα 11: Τελικά ποσοστά αραίωσης που πετυχαίνουν τα επίπεδα των τριών μοντέλων με ολικό στόχο 99%χρησιμοποιώντας την παγκόσμια μέθοδο. Με μπλε είναι το ResNet20, με πορτοκαλί το MobileNet V1 και με πράσινο το DenseNet40-24




(α′) 95% (β′) 98% (γ′) 99%

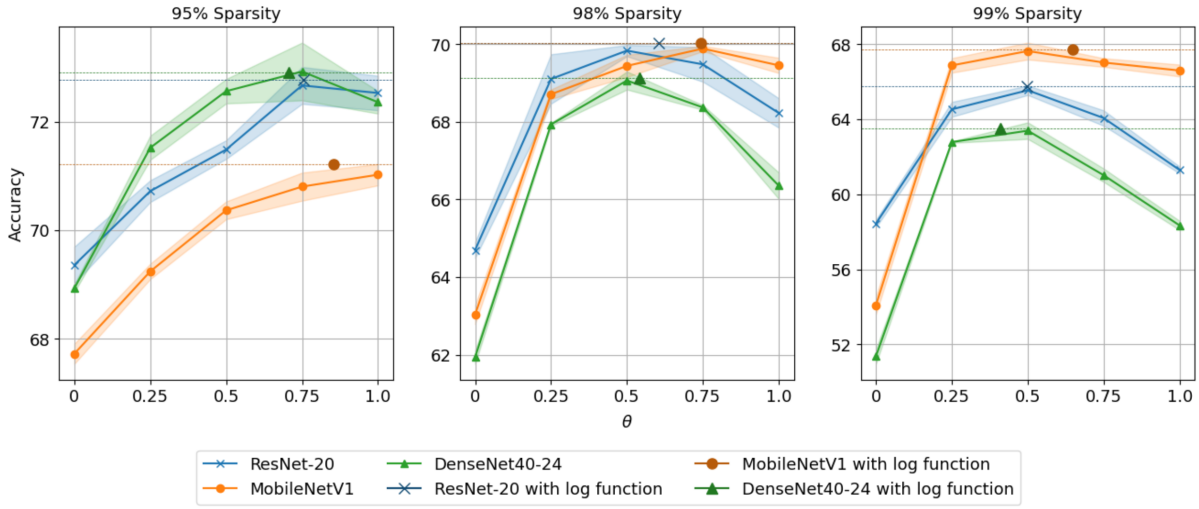
Σχήμα 12: Κλιμάκωση κλίσεων ανά εποχή για κάθε μοντέλο και κάθε ποσοστό αραίωσης με την παγκόσμια μέθοδο. Με μπλε είναι το ResNet20, με πορτοκαλί το MobileNet V1 και με πράσινο το DenseNet40-24

### Σύγκριση με το Feather

Οι δύο προσεγγίσεις συγκρίνονται απευθείας με το αρχικό Feather, κάποια SoA frameworks καθώς και με το Feather τροποποιημένο ώστε το $\theta$ να είναι το ιδανικό όπως προέκυψε από τις μελέτες του. Τα αποτελέσματα δείχνουν ότι η δυναμική συνάρτηση $\theta$ αποδίδει σταθερά καλύτερα ακόμα και όταν η τελική τιμή του είναι ίδια με την ιδανική του Feather, γεγονός που αποδίδεται στην υψηλή τιμή που διατηρεί το $\theta$ στην αρχή της εκπαίδευσης. Η ανά επίπεδο προσέγγιση δίνει καλύτερα αποτελέσματα σε μεγάλα και βαθιά δίκτυα όπως το MobileNet V1, αλλά σε μικρά μοντέλα όπως το DenseNet η παγκόσμια προσέγγιση αποδεικνύεται πιο αξιόπιστη.

| Ratio | 90% | 95% | 98% | 99% |
|---|---|---|---|---|
| ResNet-20 (1.096M Params): 73.59 $\pm$0.44 | | | | |
| ST-3 | 72.81$_{\pm 0.13}$ | 71.72$_{\pm 0.20}$ | 67.53$_{\pm 0.53}$ | 58.32$_{\pm 0.17}$ |
| Spartan | 72.56$_{\pm 0.35}$ | 71.60$_{\pm 0.40}$ | 67.27$_{\pm 0.14}$ | 61.70$_{\pm 0.21}$ |
| Feather-Global | **73.74** $\pm$0.17 | 72.53 $\pm$0.32 | 69.83 $\pm$0.14 | 65.55 $\pm$0.25 |
| Feather with best $\theta$ | **73.74** $\pm$0.17 | 72.67 $\pm$0.34 | 69.83 $\pm$0.14 | 65.55 $\pm$0.25 |
| Global $\theta$ | **73.74** $\pm$0.17 | **72.77** $\pm$0.30 | **70.04** $\pm$0.13 | **65.75** $\pm$0.28 |
| Layer-wise $\theta$ | **73.74** $\pm$0.17 | 72.66 $\pm$0.11 | 69.54 $\pm$0.20 | 64.94 $\pm$0.34 |
| MobileNetV1 (3.315M Params): 71.15 $\pm$0.17 | | | | |
| ST-3 | 70.94$_{\pm 0.25}$ | 70.44$_{\pm 0.23}$ | 69.40$_{\pm 0.06}$ | 66.63$_{\pm 0.15}$ |
| Spartan | 70.52$_{\pm 0.51}$ | 69.01$_{\pm 0.11}$ | 65.52$_{\pm 0.24}$ | 60.65$_{\pm 0.22}$ |
| Feather-Global | **71.55** $\pm$0.30 | 71.03 $\pm$0.20 | 69.44 $\pm$0.29 | 67.64 $\pm$0.45 |
| Feather with best $\theta$ | **71.55** $\pm$0.30 | 71.03 $\pm$0.20 | 69.89 $\pm$0.07 | 67.64 $\pm$0.45 |
| Global $\theta$ | **71.55** $\pm$0.30 | 71.22 $\pm$0.20 | 70.04 $\pm$0.24 | **67.73** $\pm$0.18 |
| Layer-wise $\theta$ | **71.55** $\pm$0.30 | **71.33** $\pm$0.12 | **70.29** $\pm$0.58 | 67.64 $\pm$0.31 |
| DenseNet40-24 (0.714M Params): 74.70 $\pm$0.51 | | | | |
| ST-3 | 72.56$_{\pm 0.31}$ | 71.21$_{\pm 0.35}$ | 65.48$_{\pm 0.18}$ | 56.18$_{\pm 0.60}$ |
| Spartan | 73.13$_{\pm 0.25}$ | 71.61$_{\pm 0.04}$ | 65.94$_{\pm 0.07}$ | 58.64$_{\pm 0.18}$ |
| Feather-Global | **73.75** $\pm$0.36 | 72.36 $\pm$0.21 | 69.06 $\pm$0.23 | 63.40 $\pm$0.44 |
| Feather with best $\theta$ | **73.75** $\pm$0.36 | **72.93** $\pm$**0.53** | 69.06 $\pm$0.23 | 63.40 $\pm$0.44 |
| Global $\theta$ | **73.75** $\pm$0.36 | 72.92 $\pm$0.04 | **69.13** $\pm$0.10 | **63.54** $\pm$0.32 |
| Layer-wise $\theta$ | **73.75** $\pm$0.36 | 72.53 $\pm$0.09 | 68.72 $\pm$0.31 | 61.93 $\pm$0.35 |

Table 1: Σύγκριση της ακρίβειας Top-1 στο CIFAR-100




Σχήμα 13: Τελικά αποτελέσματα της παγκόσμιας μεθόδου σε σχέση με το Feather

### Αξιολόγηση για Μη Δοκιμασμένες Τιμές Αραίωσης

Η μέθοδος δοκιμάστηκε και για ενδιάμεσες ή ακραίες τιμές στόχων αραίωσης. Η παγκόσμια συνάρτηση $\theta$ απέδωσε εξαιρετικά, συχνά ξεπερνώντας το Feather ακόμη και κατά 4% σε ακρίβεια σε εξαιρετικά υψηλές αραιώσεις.

| Ratio | 92% | 94% | 97% |
|---|---|---|---|
| ResNet-20 (1.096M Params): 73.59 | | | |
| Feather | 73.07 $\pm$0.40 | 72.73 $\pm$0.05 | 71.18 $\pm$0.43 |
| Global $\theta$ | **73.25** $\pm$0.08 | **72.91** $\pm$0.12 | **71.29** $\pm$0.06 |
| MobileNetV1 (3.315M Params): 71.15 | | | |
| Feather | 71.45 $\pm$0.09 | 71.01 $\pm$0.12 | 70.14 $\pm$0.46 |
| Global $\theta$ | **71.51** $\pm$0.24 | **71.26** $\pm$0.08 | **70.85** $\pm$0.29 |
| DenseNet40-24 (0.714M Params): 74.7 | | | |
| Feather | 73.28 $\pm$0.17 | 72.84 $\pm$0.16 | 70.85 $\pm$0.10 |
| Global $\theta$ | **73.38** $\pm$0.37 | **73.07** $\pm$0.19 | **70.89** $\pm$0.30 |

Table 2: Σύγκριση της ακρίβειας μη δοκιμασμένων ακριβειών στο CIFAR-100

| Ratio | 99.2% | 99.5% | 99.8% |
|---|---|---|---|
| ResNet-20 (1.096M Params): 73.59 | | | |
| Feather | 63.13$_{\pm 0.34}$ | 57.52$_{\pm 0.34}$ | 39.80$_{\pm 0.50}$ |
| Global $\theta$ | **63.58**$_{\pm 0.32}$ | **58.49**$_{\pm 0.14}$ | **43.24**$_{\pm 0.57}$ |
| MobileNetV1 (3.315M Params): 71.15 | | | |
| Feather | **66.62**$_{\pm 0.22}$ | 62.96$_{\pm 0.42}$ | 50.71$_{\pm 0.09}$ |
| Global $\theta$ | 66.41$_{\pm 0.23}$ | **63.04**$_{\pm 0.28}$ | **52.47**$_{\pm 0.29}$ |
| DenseNet40-24 (0.714M Params): 74.7 | | | |
| Feather | 60.53$_{\pm 0.31}$ | 53.52$_{\pm 0.24}$ | 34.86$_{\pm 0.54}$ |
| Global $\theta$ | **61.01**$_{\pm 0.38}$ | **55.20**$_{\pm 0.16}$ | **38.34**$_{\pm 0.14}$ |

Table 3: Σύγκριση της ακρίβειας μη δοκιμασμένων πολύ υψηλών ακριβειών στο CIFAR-100

Η δυναμική συνάρτηση βαθμωτής παραμέτρου $\theta$ αποτελεί ουσιαστική βελτίωση σε σχέση με τη στατική υλοποίηση του Feather. Η εξαρτημένη από την πυκνότητα λογαριθμική μορφή της επιτρέπει μια φυσική προσαρμογή κατά τη διάρκεια της εκπαίδευσης, που ενισχύει τη σταθερότητα και μειώνει τον κίνδυνο υποεκπαίδευσης. Η παγκόσμια εφαρμογή της συνάρτησης αποδείχθηκε η πιο σταθερή λύση σε ακραίες συνθήκες αραιότητας, ενώ η ανά επίπεδο μπορεί να αποδώσει καλύτερα σε βαθύτερα μοντέλα με πιο ανομοιογενή δομή.

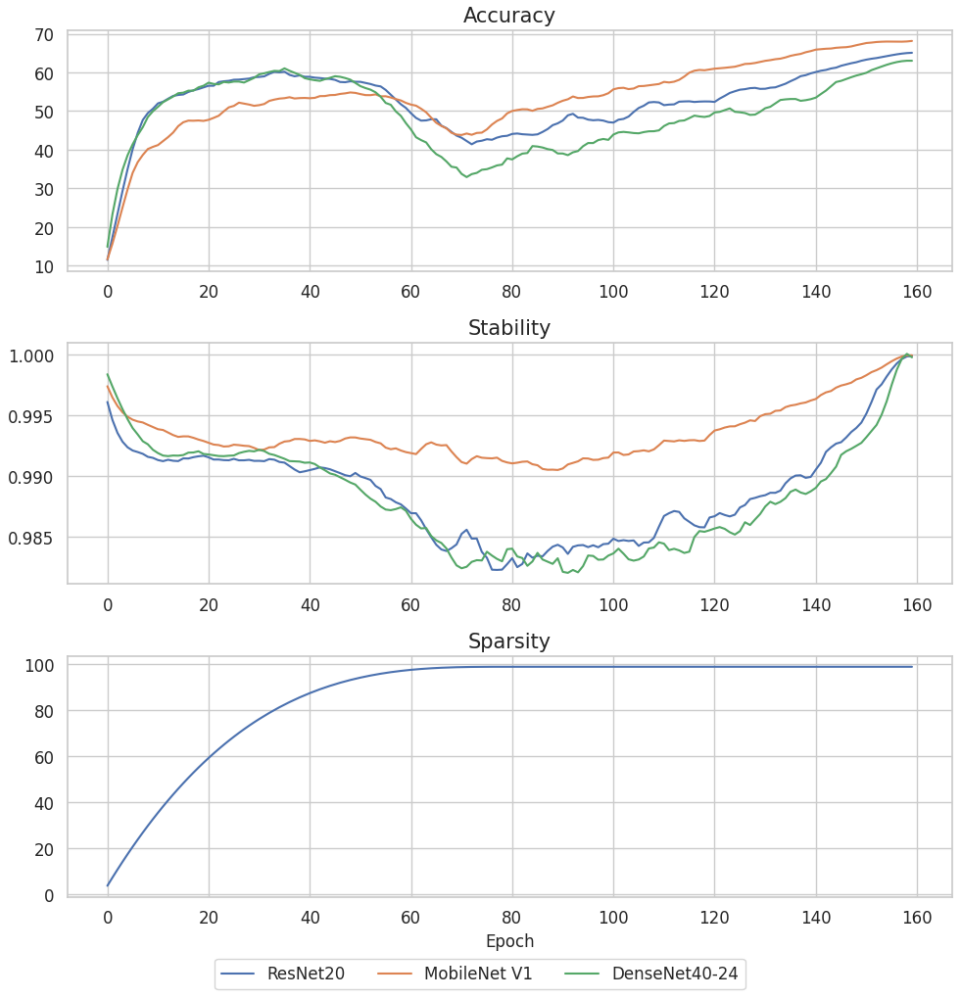
# 5 Προσαρμοστικός Προγραμματισμός Κλαδέματος

Κατά την εκπαίδευση με το Feather, η διαδικασία κλαδέματος εφαρμόζεται με τη χρήση δυαδικών μασκών $M$ σε κάθε στρώση του δικτύου. Καθώς αυξάνεται ο στόχος αραίωσης σε ακραίες τιμές όπως 98% ή 99%, η σταθερότητα των μασκών μειώνεται, προκαλώντας σοβαρές απώλειες στην απόδοση του μοντέλου. Σκοπός του κεφαλαίου είναι να ερευνήσει αυτή τη σύνδεση μεταξύ της σταθερότητας των μασκών, της απόδοσης και του στόχου αραίωσης, και να παρουσιάσει έναν απλό αλλά αποτελεσματικό προσαρμοστικό χρονοπρογραμματιστή κλαδέματος, ο οποίος μετριάζει τις απώλειες ακρίβειας.

## 5.1 Σταθερότητα Μάσκας και Απόδοση

Για να μετρηθεί η σταθερότητα μεταξύ δύο μασκών σε διαδοχικές επαναλήψεις της εκπαίδευσης χρησιμοποιούμε τον δείκτη ομοιότητας Jaccard, ο οποίος ορίζεται ως το μέγεθος της τομής δύο συνόλων προς το μέγεθος της ένωσής τους. Καθώς οι μάσκες είναι δυαδικοί πίνακες (με 0 και 1), αυτή η μετρική είναι κατάλληλη για την εκτίμηση των μεταβολών στο ποιοι σύνδεσμοι παραμένουν ενεργοί ανά επανάληψη.

Η σταθερότητα των μασκών μετράται ανά εποχή στα τρία μοντέλα ResNet20, MobileNet V1 και DenseNet40-24. Παρατηρήθηκε ότι όταν επιτυγχάνονται πολύ μεγάλες αραιότητες, όπως 98% ή 99%, η σταθερότητα μειώνεται απότομα, οδηγώντας σε πτώση της ακρίβειας, ιδιαίτερα στα ResNet20 και DenseNet40-24 που είναι μικρότερα μοντέλα.




Σχήμα 14: Ακρίβεια, μέση σταθερότητα μάσκας και αραιότητα μοντέλου που επιτυγχάνονται ανά εποχή εκπαίδευσης με στόχο αραιότητας 99% για ResNet20 (μπλε), MobileNet V1 (πορτοκαλί) και DenseNet40-24 (πράσινο)

## 5.2 Προτεινόμενη Μέθοδος

### Διαφορά Σταθερότητας Μάσκας

Η βασική ιδέα του χρονοπρογραμματιστή είναι να παρακολουθείται η διαφορά της σταθερότητας ανάμεσα σε δύο διαδοχικές εποχές $\mu_{k+1} - \mu_k$. Αν η σταθερότητα αυξάνεται, τότε μπορούμε να προχωρήσουμε σε αύξηση της συνολικής αραιότητας του μοντέλου. Αν όχι, παγώνουμε την τρέχουσα τιμή της αραιότητας. Η χρήση της διαφοράς και όχι της απόλυτης τιμής δίνει έμφαση στην τάση του συστήματος να σταθεροποιείται. Ο χρονοπρογραμματιστής ουσιαστικά δίνεται από την συνάρτηση:

$$S_{k+1} = S_k + \lambda \cdot (\mu_{k+1} - \mu_k)$$

Όπου $S_{k+1}$ είναι η παραγώμενη αραιότητα, $S_k$ η αραιότητα που επιτεύχθηκε στην προηγούμενη επανάληψη εκπαίδευσης και $\lambda$ ένας κατάλληλος συντελεστής που θα αναλυθεί παρακάτω.

### Συνάρτηση Κλιμάκωσης

Για να προσαρμόζεται η κλιμάκωση της διαφοράς των μασκών σταθερότητας όταν αυτή η διαφορά είναι θετική, χρησιμοποιείται μια σιγμοειδής συνάρτηση του τύπου:

$$\lambda(S_k) = \frac{A}{1 + e^{-C(B-S_k)}} + D$$

Το $A$ είναι η σταθερά που καθορίζει την κλιμάκωση της διαφοράς σταθερότητας για χαμηλές αραιότητες, το $B$ καθορίζει σε ποια αραιότητα η διαφορά θα είναι στο μέσο της πτώσης, το $C$ είναι η κλίση και το $D$ ένας όρος μεροληψίας για να βεβαιωθεί ότι η τιμή $\lambda$ δεν πέφτει κάτω από το 0 καθώς και προσαρμόζοντας την καμπύλη στις συμπεριφορές του καθενός σταθερότητα του μοντέλου. Αυτό είναι απαραίτητο αφού κάθε μοντέλο παρουσιάζει διαφορετικές πτώσεις και αυξήσεις στη σταθερότητα της μάσκας, όπως φαίνεται στο Σχήμα 14.

Όταν η διαφορά είναι αρνητική, τότε θέτουμε $\lambda(S_k) = 0$ ώστε η αραιότητα να παγώνει σε μία σταθερή τιμή.

### Κλιμάκωση ως Προς τον Αριθμό των Εποχών

Καθώς τα πειράματα για την εξαγωγή της συνάρτησης έγιναν για 160 εποχές, για να γίνει η μέθοδος ανεξάρτητη από το συνολικό πλήθος επαναλήψεων εισάγεται ο συντελεστής:

$$l = \frac{160}{|\text{epochs}|}$$

### Τελικές Συναρτήσεις

Η τελική μορφή του δυναμικού χρονοπρογραμματιστή είναι:

$$S_{k+1} = \begin{cases} S_k + l \cdot \lambda(S_k) \cdot (\mu_{k+1} - \mu_k) & \mu_{k+1} > \mu_k \\ S_k & \mu_{k+1} \leq \mu_k \end{cases}$$

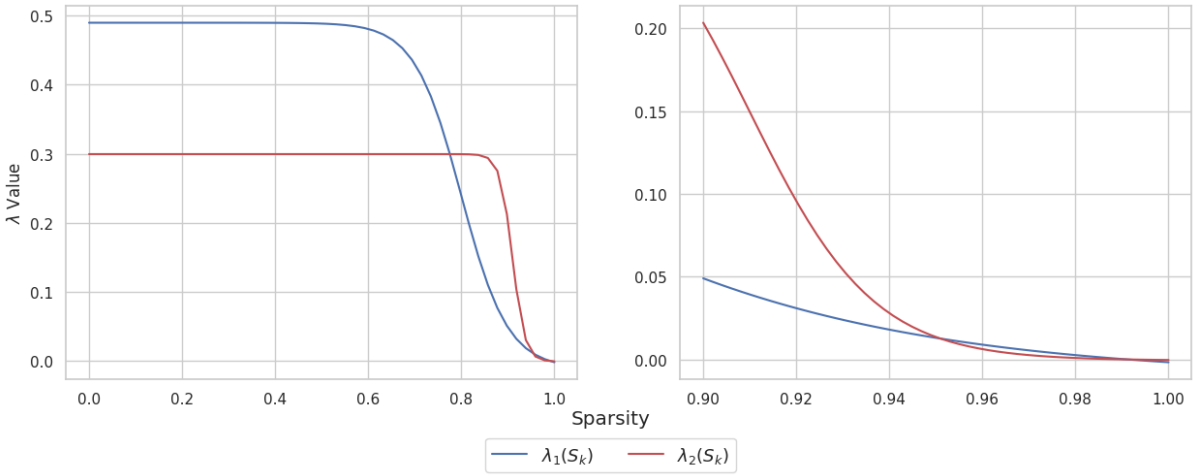
Δοκιμάζονται δύο παραλλαγές της συνάρτησης έτσι ώστε να μελετηθούν οι διαφορετικές ιδιότητες και συμπεριφορές.

$$\lambda_1(S_k) = \frac{0.5}{1 + e^{-20(0.80-S_k)}} + D$$

$$\lambda_2(S_k) = \frac{0.3}{1 + e^{-75(0.91-S_k)}} + D$$

Ο όρος μεροληψίας $D$ διαφέρει στην απόδοση διαφορετικών μοντέλων, κάτι που μπορεί να αποδοθεί στη διαφορετική κλιμάκωση της μέσης σταθερότητας της μάσκας. Για παράδειγμα, όπως φαίνεται στην Εικόνα 14, το MobileNet V1 είναι ένα πολύ πιο σταθερό μοντέλο, παρουσιάζοντας πολύ μικρότερες πτώσεις και αυξήσεις σε σύγκριση με το άλλα δύο. Με τον ίδιο όρο μεροληψίας είτε θα έφτανε στον στόχο αραιότητας πολύ αργά στην εκπαίδευση ή και καθόλου. Για $\lambda_1(S_k)$ έχουμε $D = -0.01$ για ResNet20 και DenseNet40-24 και $D = -0.007$ για MobileNet V1. Οι αντίστοιχες τιμές για το $\lambda_2(S_k)$ είναι $D = 0$ και $D = 0.003$ αντίστοιχα.

Στην Εικόνα 15 έχουμε τις δύο συναρτήσεις για το $\lambda(S_k)$

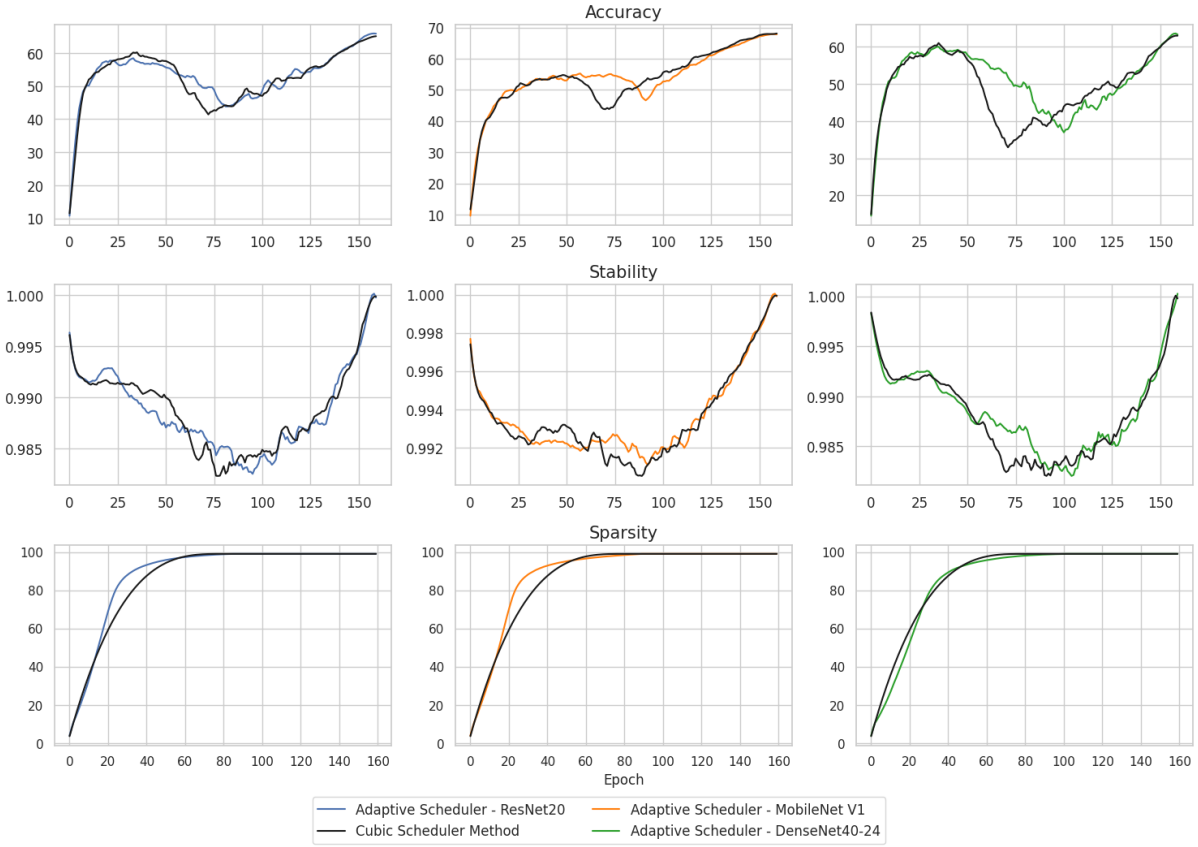



Σχήμα 15: Τιμές $\lambda$ για όλα τα επίπεδα αραιότητας (α) και για πολύ υψηλά (β). Το δεύτερο γράφημα παρέχεται για σαφήνεια, καθώς το πρώτο δεν δείχνει τις διαφορές για υψηλές αραιότητες

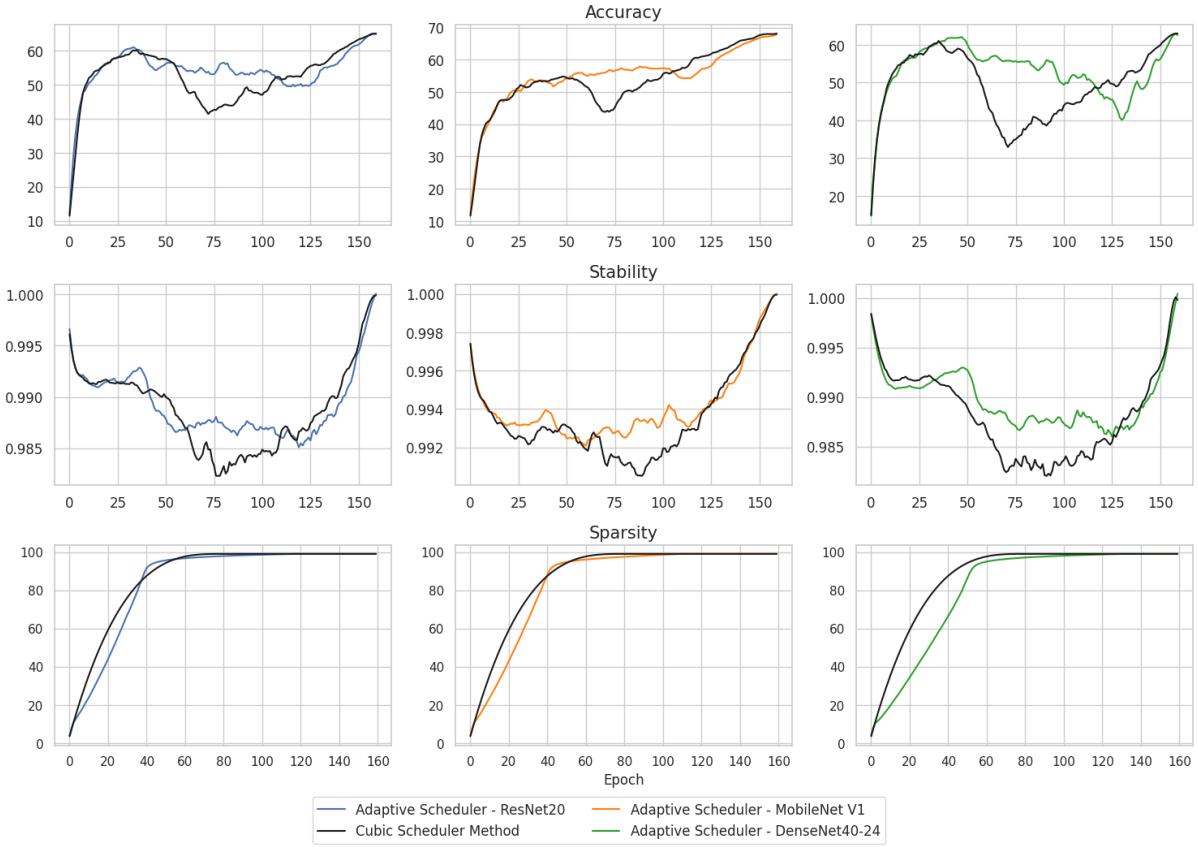
## 5.3 Αξιολόγηση

Οι δύο χρονοπρογραμματιστές αξιολογούνται σε σύγκριση με τον αρχικό κυβικό. Ο πρώτος δυναμικός χρονοπρογραμματιστής, ο οποίος βασίζεται στη συνάρτηση κλιμάκωσης $\lambda_1(S_k)$, απέδωσε καλύτερα όσον αφορά την τελική ακρίβεια σε σχέση με τον κυβικό. Αυτό οφείλεται στο ότι επιτρέπει ταχύτερη και πιο επιθετική προσέγγιση των υψηλών ποσοστών αραιότητας όταν ανιχνεύεται αυξανόμενη σταθερότητα στη μάσκα. Ουσιαστικά, η στρατηγική αυτή επιβραβεύει την σταθερότητα, επιτρέποντας στο μοντέλο να συνεχίσει το κλάδεμα όταν έχει ήδη επιτύχει μια σχετική ομοιομορφία στους ενεργούς συνδέσμους. Το αποτέλεσμα είναι ότι το μοντέλο μπορεί να φτάσει στον στόχο πολύ νωρίτερα, διαθέτοντας περισσότερο χρόνο για να προσαρμόσει τα βάρη του σε αυτό το επίπεδο, με θετικό αντίκτυπο στην απόδοση.

Αντιθέτως, ο δεύτερος δυναμικός χρονοπρογραμματιστής που χρησιμοποιεί τη συνάρτηση $\lambda_2(S_k)$, έχει πιο συντηρητική συμπεριφορά. Επιτρέπει μικρότερες αυξήσεις της αραιότητας ακόμη και όταν παρατηρείται αύξηση της σταθερότητας, με αποτέλεσμα το μοντέλο να φτάνει στον επιθυμητό στόχο αραίωσης με βραδύτερο ρυθμό. Αυτή η προσέγγιση εξασφαλίζει υψηλότερη μέση σταθερότητα μάσκας καθ' όλη τη διάρκεια της εκπαίδευσης, καθώς αποφεύγονται ξαφνικές αλλαγές, όμως περιορίζει την απόδοση του μοντέλου σε συνθήκες όπου απαιτείται ταχεία προσαρμογή. Σε πολλές περιπτώσεις, το μοντέλο δεν διαθέτει επαρκές χρονικό περιθώριο για προσαρμογή και έτσι η τελική ακρίβεια παραμένει ελαφρώς υποδεέστερη.




Σχήμα 16: Απόδοση και των τριών μοντέλων με στόχο αραιότητας 99% με κυβικό χρονοπρογραμματιστή (μαύρο) και τον προτεινόμενο προσαρμοστικό (μπλε, πορτοκαλί και πράσινο) χρησιμοποιώντας $\lambda_1(S_k)$ για κλιμάκωση




Σχήμα 17: Απόδοση και των τριών μοντέλων με στόχο αραιότητας 99% με κυβικό χρονοπρογραμματιστή (μαύρο) και τον προτεινόμενο προσαρμοστικό (μπλε, πορτοκαλί και πράσινο) χρησιμοποιώντας $\lambda_2(S_k)$ για κλιμάκωση

| Ratio | 90% | 95% | 98% | 99% |
|---|---|---|---|---|
| ResNet-20 (1.096M Params): 73.59 $_{\pm 0.44}$ | | | | |
| ST-3 | 72.81±0.13 | 71.72$_{\pm 0.20}$ | 67.53$_{\pm 0.53}$ | 58.32$_{\pm 0.17}$ |
| Spartan | 72.56$_{\pm 0.35}$ | 71.60$_{\pm 0.40}$ | 67.27$_{\pm 0.14}$ | 61.70$_{\pm 0.21}$ |
| Feather | **73.74**$_{\pm 0.17}$ | 72.53$_{\pm 0.32}$ | 69.83$_{\pm 0.14}$ | 65.55$_{\pm 0.25}$ |
| Feather with different schedulers | | | | |
| Adaptive Scheduler 1 | 73.48$_{\pm 0.32}$ | 72.56$_{\pm 0.12}$ | **70.03**$_{\pm 0.09}$ | **65.72**$_{\pm 0.17}$ |
| Adaptive Scheduler 2 | 73.70$_{\pm 0.26}$ | **72.69**$_{\pm 0.17}$ | 69.92$_{\pm 0.38}$ | 64.93$_{\pm 0.21}$ |
| MobileNetV1 (3.315M Params): 71.15 $_{\pm 0.17}$ | | | | |
| ST-3 | 70.94$_{\pm 0.25}$ | 70.44$_{\pm 0.23}$ | 69.40$_{\pm 0.06}$ | 66.63$_{\pm 0.15}$ |
| Spartan | 70.52$_{\pm 0.51}$ | 69.01$_{\pm 0.11}$ | 65.52$_{\pm 0.24}$ | 60.65$_{\pm 0.22}$ |
| Feather | 71.55$_{\pm 0.30}$ | 71.03$_{\pm 0.20}$ | 69.44$_{\pm 0.29}$ | 67.64$_{\pm 0.45}$ |
| Feather with different schedulers | | | | |
| Adaptive Scheduler 1 | **71.56**$_{\pm 0.20}$ | **71.23**$_{\pm 0.29}$ | 69.75$_{\pm 0.03}$ | **68.04**$_{\pm 0.19}$ |
| Adaptive Scheduler 2 | 71.20$_{\pm 0.07}$ | 71.06$_{\pm 0.19}$ | **70.02**$_{\pm 0.30}$ | 67.77$_{\pm 0.13}$ |
| DenseNet40-24 (0.714M Params): 74.70 $_{\pm 0.51}$ | | | | |
| ST-3 | 72.56$_{\pm 0.31}$ | 71.21$_{\pm 0.35}$ | 65.48$_{\pm 0.18}$ | 56.18$_{\pm 0.60}$ |
| Spartan | 73.13$_{\pm 0.25}$ | 71.61$_{\pm 0.04}$ | 65.94$_{\pm 0.07}$ | 58.64$_{\pm 0.18}$ |
| Feather | 73.75$_{\pm 0.36}$ | 72.36$_{\pm 0.21}$ | 69.06$_{\pm 0.23}$ | 63.40$_{\pm 0.44}$ |
| Feather with different schedulers | | | | |
| Adaptive Scheduler 1 | 73.32$_{\pm 0.12}$ | 72.13$_{\pm 0.12}$ | **69.14**$_{\pm 0.14}$ | **63.68**$_{\pm 0.37}$ |
| Adaptive Scheduler 2 | **73.80**$_{\pm 0.17}$ | **72.47**$_{\pm 0.20}$ | 69.10$_{\pm 0.09}$ | 62.73$_{\pm 0.15}$ |

Table 4: Απόδοση χρησιμοποιώντας τον προσαρμοστικό χρονοπρογραμματιστή για εκπαίδευση 160 εποχών

| Ratio | 90% | 95% | 98% | 99% |
|---|---|---|---|---|
| ResNet-20 (1.096M Params) | | | | |
| Cubic Scheduler | **69.77**$_{\pm 0.29}$ | 67.87$_{\pm 0.27}$ | 63.78$_{\pm 0.30}$ | 58.42$_{\pm 0.14}$ |
| Adaptive Scheduler 1 | 69.29$_{\pm 0.09}$ | **67.96**$_{\pm 0.30}$ | **63.94**$_{\pm 0.36}$ | 58.48$_{\pm 0.08}$ |
| Adaptive Scheduler 2 | 69.36$_{\pm 0.18}$ | 67.60$_{\pm 0.30}$ | 63.71$_{\pm 0.23}$ | **58.67**$_{\pm 0.44}$ |
| MobileNetV1 (3.315M Params) | | | | |
| Cubic Scheduler | 64.96$_{\pm 0.19}$ | 64.41$_{\pm 0.25}$ | 61.47$_{\pm 0.31}$ | 57.73$_{\pm 0.49}$ |
| Adaptive Scheduler 1 | 64.98$_{\pm 0.33}$ | 64.21$_{\pm 0.08}$ | 61.76$_{\pm 0.31}$ | **57.87**$_{\pm 0.39}$ |
| Adaptive Scheduler 2 | **65.01**$_{\pm 0.26}$ | **64.47**$_{\pm 0.26}$ | **62.14**$_{\pm 0.71}$ | 57.79$_{\pm 0.47}$ |
| DenseNet40-24 (0.714M Params) | | | | |
| Cubic Scheduler | 69.49$_{\pm 0.14}$ | **67.04**$_{\pm 0.40}$ | 62.07$_{\pm 0.25}$ | **56.37**$_{\pm 0.12}$ |
| Adaptive Scheduler 1 | 69.56$_{\pm 0.03}$ | 66.96$_{\pm 0.20}$ | 62.27$_{\pm 0.14}$ | 56.35$_{\pm 0.20}$ |
| Adaptive Scheduler 2 | **69.59**$_{\pm 0.11}$ | 66.49$_{\pm 0.25}$ | **62.41**$_{\pm 0.15}$ | 55.19$_{\pm 0.08}$ |

Table 5: Απόδοση χρησιμοποιώντας τον προσαρμοστικό χρονοπρογραμματιστή για εκπαίδευση 30 εποχών

Ενώ ο πρώτος προγραμματιστής έδειξε βελτιωμένη ακρίβεια σε πολλές περιπτώσεις, ο δεύτερος ήταν πιο σταθερός αλλά όχι πάντα αποτελεσματικός. Η μέθοδος αποτελεί ένα βήμα προς πιο σταθερή εκπαίδευση υπό ακραία κλάδεμα, με δυνατότητα περαιτέρω βελτιώσεων σε μελλοντική εργασία, ειδικά όσον αφορά την αυτόματη εξαγωγή των σταθερών της συνάρτησης.

# 6 Μελέτη για την Σημασία των Κλαδεμένων Βαρών

Η ενότητα αυτή διερευνά τη σημασία των βαρών που έχουν κλαδευτεί σε πλαίσιο εκπαίδευσης με αραιότητα, εστιάζοντας στον ρόλο του Straight-Through Estimator (STE) [7]. Το STE επιτρέπει στο δίκτυο να εκμεταλλευτεί τα πλεονεκτήματα της πλήρους εκπαίδευσης, επιτρέποντας τη ροή παραγώγων μέσα από τα κλαδεμένα βάρη. Πειράματα δείχνουν πως η χρήση του STE προσφέρει σημαντική βελτίωση στην ακρίβεια σε σχέση με την πλήρη απομάκρυνση των κλαδεμένων βαρών κατά την οπισθοδιάδοση. Ο στόχος του κεφαλαίου είναι να εντοπίσει πόσα από τα βάρη που κλαδεύονται τελικά διατηρούν σημασία και πώς αυτό εξαρτάται από το πότε αφαιρούνται κατά τη διάρκεια της εκπαίδευσης.

## 6.1 Μέθοδοι Αξιολόγησης

Για να εκτιμηθεί η πραγματική σημασία των κλαδεμένων βαρών, σχεδιάστηκαν συγκεκριμένα πειράματα που εξετάζουν διαφορετικές στρατηγικές αφαίρεσής τους. Οι μέθοδοι αξιολόγησης χωρίζονται σε δύο βασικούς άξονες: το κατώφλι κάτω από το οποίο τα βάρη σταματάνε πλήρως να ανανεώνονται και το χρονικό σημείο της εκπαίδευσης στο οποίο γίνεται αυτή η αφαίρεση. Κάθε άξονας χωρίζεται σε δύο υποπεριπτώσεις, που συνολικά προκύπτουν τέσσερις συνδυασμοί αξιολόγησης.

### Κατωφλίωση

Το κατώφλι κάτω από το οποίο τα βάρη σταματάνε πλήρως να ανανεώνονται υπολογίζεται με βάση το επιθυμητό ποσοστό απομάκρυνσης. Αυτό το κατώφλι υπολογίζεται σε σχέση με το ποσοστό αραίωσης που έχει επιτευχθεί είτε από το δίκτυο είτε από κάθε επίπεδο ξεχωριστά και εφαρμόζεται είτε σε καθολική κλίμακα, όπου συγκρίνονται όλα τα βάρη του δικτύου μαζί, είτε κατά επίπεδο, όπου κάθε επίπεδο έχει το δικό του τοπικό κατώφλι. Η σύγκριση αυτών των δύο προσεγγίσεων προσδίδει την σημασία της τοπικής πληροφορίας στην απόφαση για αφαίρεση.

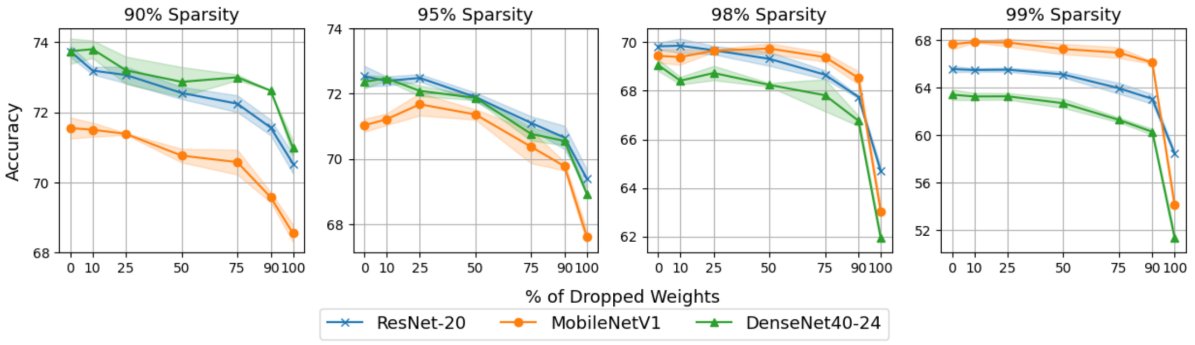
### Χρονικό Πλαίσιο Αφαίρεσης

Η δεύτερη διάσταση αξιολόγησης αφορά το πότε εφαρμόζεται η αφαίρεση. Εξετάζεται αρχικά η επίδραση της μόνιμης απομάκρυνσης ενός ποσοστού βαρών με το που ξεκινήσει η εκπαίδευση μέχρι το τέλος της. Αυτή η προσέγγιση είναι αυστηρότερη, καθώς το δίκτυο δεν έχει την ευκαιρία να αξιοποιήσει κάποιο ποσοστό βαρών. Η δεύτερη προσέγγιση περιλαμβάνει τα κλαδεμένα βάρη να συμμετέχουν πλήρως στην εκπαίδευση έως ότου επιτευχθεί το προκαθορισμένο επίπεδο αραιότητας, και τότε να σταματήσει το επιθυμητό ποσοστό να ανανεώνεται. Έτσι δίνεται η ευκαιρία στο δίκτυο να αναπτύξει πιθανές εξαρτήσεις, πριν αξιολογηθεί η αναγκαιότητα διατήρησης αυτών των βαρών.

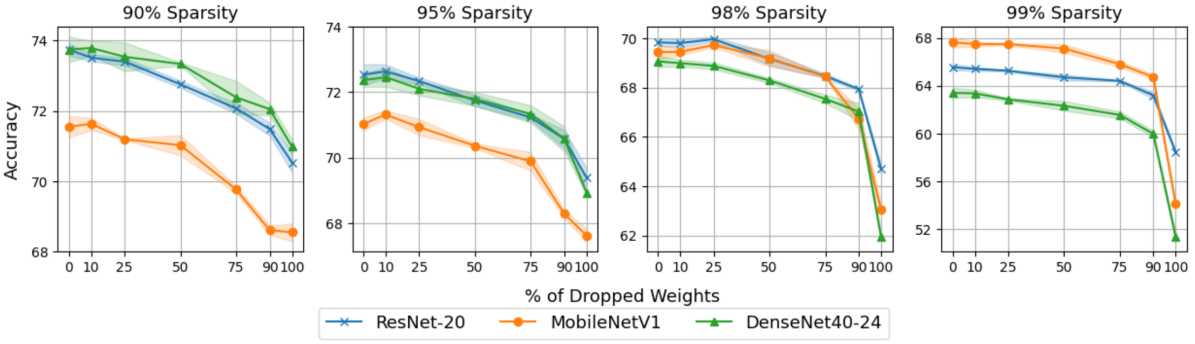
## 6.2 Πειραματική Αξιολόγηση

### Σύγκριση

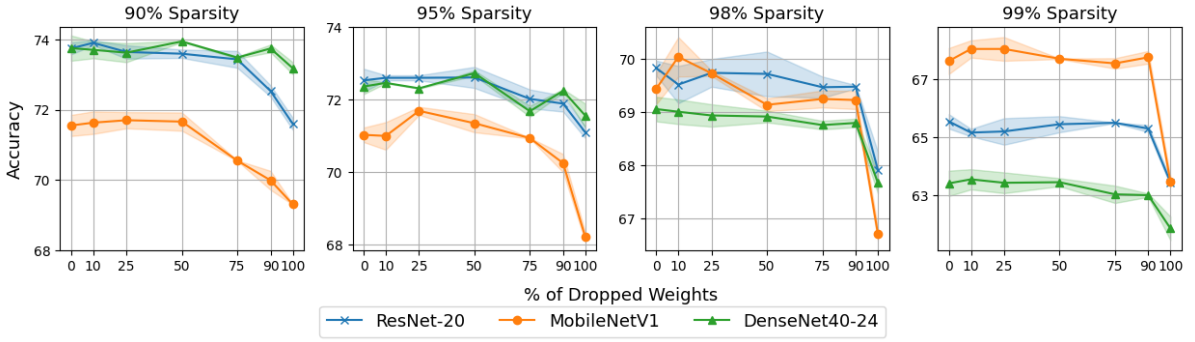
Πραγματοποιούνται εκτενείς μελέτες σε διάφορες αρχιτεκτονικές. Οι πειραματικές παραλλαγές περιλαμβάνουν αλλαγές στην τιμή του κατωφλίου, στον τρόπο εφαρμογής του και στο χρονικό πλαίσιο.



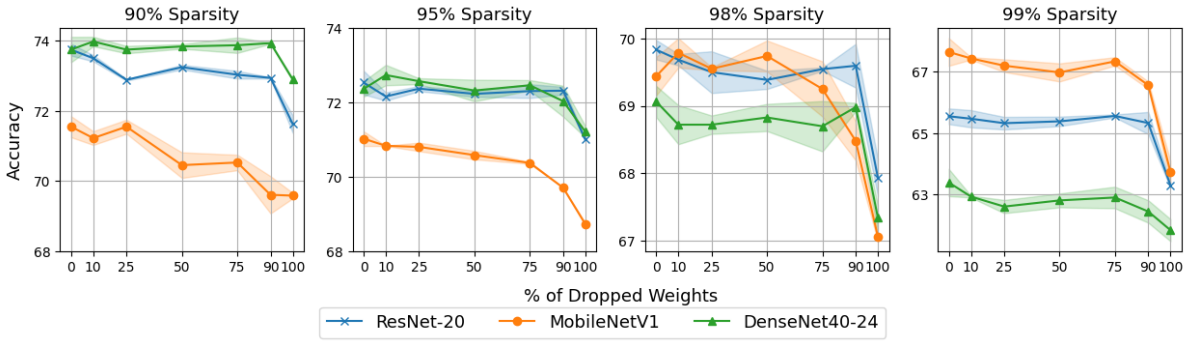

Σχήμα 18: Μελέτη για την επίδραση της πτώσης ποσοστού βαρών με καθολικό κατώφλι από την αρχή της εκπαίδευσης



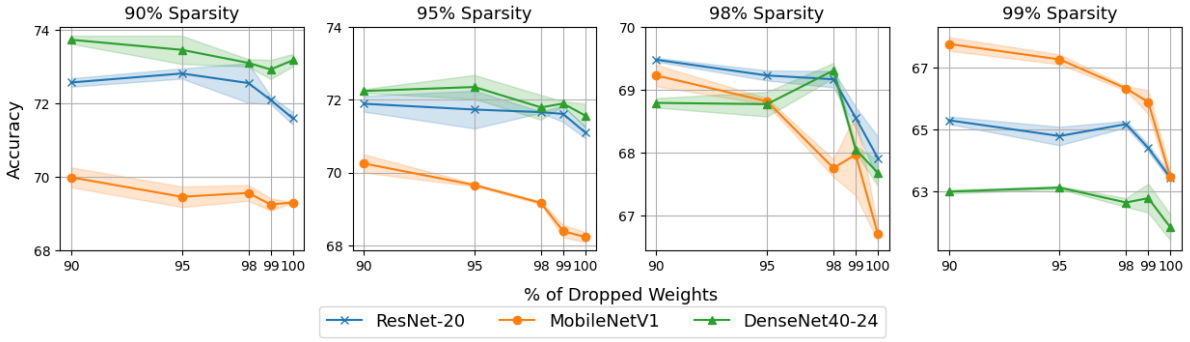

Σχήμα 19: Μελέτη για την επίδραση της πτώσης ποσοστού βαρών με κατώφλι ανά επίπεδο από την αρχή της εκπαίδευσης



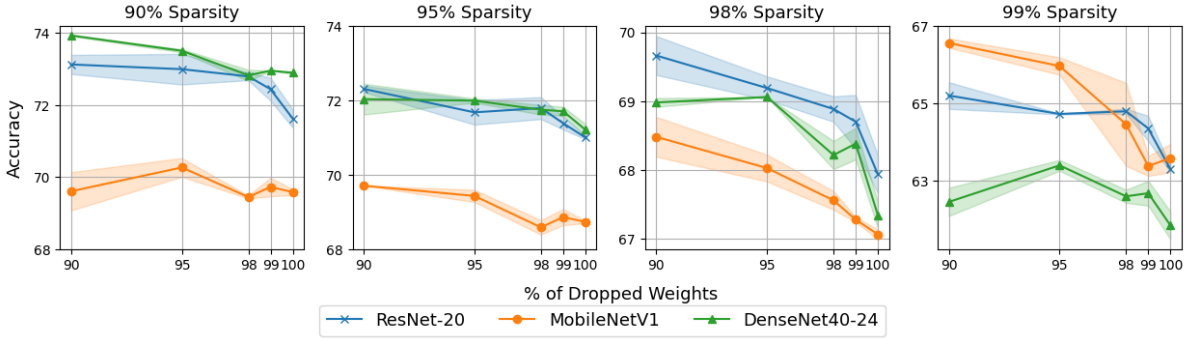

Σχήμα 20: Μελέτη για την επίδραση της πτώσης ποσοστού βαρών με καθολικό κατώφλι μετά την επίτευξη του στόχου αραίωσης




Σχήμα 21: Μελέτη για την επίδραση της πτώσης ποσοστού βαρών με κατώφλι ανά επίπεδο μετά την επίτευξη του στόχου αραίωσης




Σχήμα 22: Περαιτέρω μελέτη σχετικά με την πτώση ακραίων ποσοστών βαρών με καθολικό κατώφλι μετά την επίτευξη του στόχου αραίωσης




Σχήμα 23: Περαιτέρω μελέτη σχετικά με την πτώση ακραίων ποσοστών βαρών με κατώφλι ανά επίπεδο μετά την επίτευξη του στόχου αραίωσης

Τα αποτελέσματα δείχνουν ότι η εφαρμογή κατωφλιών μεγαλύτερων του 0.25 οδηγεί σε απότομη μείωση απόδοσης όταν το κλάδεμα εφαρμόζεται από την αρχή, ενώ η εφαρμογή μετά την επίτευξη αραίωσης οδηγεί σε σχεδόν παρόμοια απόδοση με το STE. Αυτά τα ευρήματα δείχνουν ότι τα μικρής τιμής βάρη μπορούν πράγματι να αφαιρεθούν με ασφάλεια μετά από κάποιο σημείο της εκπαίδευσης. Στον Πίνακα παρουσιάζονται τα καλύτερα αποτελέσματα που επιτεύχθηκαν όταν διαγράφηκαν βάρη σε αντίστοιχα ποσοστά, για όλους τους δυνατούς συνδυασμούς μεθόδων και χρονικών στιγμών εφαρμογής. Ο Πίνακας παρουσιάζει τα ποσοστά διαγραμμένων βαρών στα οποία η απόδοση μειώνεται σημαντικά. Η σημαντική πτώση ορίζεται ως το

σημείο στο οποίο η μέση ακρίβεια, μαζί με τα όρια που καθορίζονται από την τυπική απόκλιση, παύουν να επικαλύπτονται με αυτά του μοντέλου χωρίς καθόλου αφαίρεση βαρών.

Όταν τα βάρη αποκλείονται από τη διαδικασία βελτιστοποίησης από την αρχή της εκπαίδευσης, για μικρά ποσοστά απομακρυσμένων βαρών (π.χ. 10% ή 25%), η επίδραση στην απόδοση είναι συνήθως είτε αμελητέα είτε θετική. Αυτό υποδηλώνει ότι η πρώιμη απομάκρυνση ασήμαντων συνδέσεων μπορεί να βελτιώσει τη γενίκευση, ιδίως σε υπερπαραμετροποιημένα μοντέλα. Ωστόσο, καθώς το ποσοστό των διαγραμμένων βαρών αυξάνεται πέραν του 50%, η απόδοση αρχίζει να επιδεινώνεται. Αυτό συμβαίνει επειδή πολλά από τα βάρη που θα μπορούσαν ενδεχομένως να εξελιχθούν σε σημαντικά αφαιρούνται πρόωρα, εμποδίζοντας τον σχηματισμό κρίσιμων συνδέσεων και πιθανώς την επαναφορά τους σε μεταγενέστερα στάδια εκπαίδευσης.

Όταν τα βάρη απομακρύνονται μόνο αφού το μοντέλο έχει φτάσει στο επιθυμητό επίπεδο αραιότητας, τα αποτελέσματα είναι σαφώς πιο σταθερά. Σε πολλές περιπτώσεις, η απόδοση παραμένει σχεδόν ίδια με εκείνη της κανονικής μεθόδου STE, συνήθως μέχρι ένα πολύ υψηλό κατώφλι απομάκρυνσης βαρών (90–95%). Αυτό υποδηλώνει ότι τα περισσότερα από τα κλαδεμένα βάρη όντως δεν έχουν ιδιαίτερη σημασία μετά τη σύγκλιση της διαδικασίας αραιότητας. Η πτώση στην απόδοση μεταξύ 90%–100% ποσοστών αφαίρεσης υποδεικνύει ότι ένα μικρό μέρος αυτών των βαρών, κυρίως αυτά που βρίσκονται κοντά στο κατώφλι κλαδέματος, εξακολουθούν να συμβάλλουν στην απόδοση του μοντέλου όταν τους επιτρέπεται να ενημερώνονται σε μεταγενέστερα στάδια της εκπαίδευσης.

Ως προς τις δύο μεθόδους κατωφλιοποίησης, δεν υπάρχει σαφής νικητής. Η παγκόσμια μέθοδος έχει ελαφρώς καλύτερη απόδοση όταν απομακρύνονται τα βάρη αφού φτάσουν σε αραιότητα, ενώ η ανά επίπεδο όταν η αφαίρεση ξεκινά στην αρχή.

| Ratio | 90% | 95% | 98% | 99% |
|---|---|---|---|---|
| ResNet-20 (1.096M Params): 73.59 $\pm 0.44$ | | | | |
| Feather (0%) | $73.74_{\pm 0.17}$ | $72.53_{\pm 0.32}$ | $69.83_{\pm 0.14}$ | $65.55_{\pm 0.25}$ |
| From the Beginning of Training | | | | |
| Global | $73.19_{\pm 0.11}(10\%)$ | $72.48_{\pm 0.12}(25\%)$ | $69.86_{\pm 0.30}(10\%)$ | $65.50_{\pm 0.17}(25\%)$ |
| Layer-wise | $73.51_{\pm 0.10}(10\%)$ | $\mathbf{72.63}_{\pm 0.21}(10\%)$ | $\mathbf{69.96}_{\pm 0.10}(25\%)$ | $65.40_{\pm 0.16}(10\%)$ |
| After Reaching Sparsity | | | | |
| Global | $\mathbf{73.90}_{\pm 0.09}(10\%)$ | $72.61_{\pm 0.29}(50\%)$ | $69.75_{\pm 0.25}(25\%)$ | $65.50_{\pm 0.01}(75\%)$ |
| Layer-wise | $73.50_{\pm 0.09}(10\%)$ | $72.37_{\pm 0.07}(25\%)$ | $69.68_{\pm 0.09}(10\%)$ | $\mathbf{65.56}_{\pm 0.05}(75\%)$ |
| MobileNetV1 (3.315M Params): 71.15 $\pm 0.17$ | | | | |
| Feather (0%) | $71.55_{\pm 0.30}$ | $71.03_{\pm 0.20}$ | $69.44_{\pm 0.29}$ | $67.64_{\pm 0.45}$ |
| From the Beginning of Training | | | | |
| Global | $71.50_{\pm 0.20}(10\%)$ | $71.67_{\pm 0.34}(25\%)$ | $69.74_{\pm 0.21}(50\%)$ | $67.83_{\pm 0.03}(10\%)$ |
| Layer-wise | $71.63_{\pm 0.17}(10\%)$ | $71.31_{\pm 0.13}(10\%)$ | $69.72_{\pm 0.08}(25\%)$ | $67.48_{\pm 0.18}(10\%)$ |
| After Reaching Sparsity | | | | |
| Global | $\mathbf{71.70}_{\pm 0.23}(25\%)$ | $\mathbf{71.69}_{\pm 0.11}(25\%)$ | $\mathbf{70.04}_{\pm 0.37}(10\%)$ | $\mathbf{68.06}_{\pm 0.41}(25\%)$ |
| Layer-wise | $71.55_{\pm 0.20}(25\%)$ | $70.84_{\pm 0.01}(10\%)$ | $69.79_{\pm 0.22}(10\%)$ | $67.43_{\pm 0.04}(10\%)$ |
| DenseNet40-24 (0.714M Params): 74.70 $\pm 0.51$ | | | | |
| Feather (0%) | 73.75 $\pm 0.36$ | 72.36 $\pm 0.21$ | **69.06** $\pm 0.23$ | 63.40 $\pm 0.44$ |
| From the Beginning of Training | | | | |
| Global | $73.80_{\pm 0.25}(10\%)$ | $72.47_{\pm 0.09}(10\%)$ | $68.73_{\pm 0.29}(25\%)$ | $63.25_{\pm 0.28}(25\%)$ |
| Layer-wise | $73.79_{\pm 0.21}(10\%)$ | $72.45_{\pm 0.29}(10\%)$ | $68.99_{\pm 0.14}(10\%)$ | $63.35_{\pm 0.30}(10\%)$ |
| After Reaching Sparsity | | | | |
| Global | $73.94_{\pm 0.02}(50\%)$ | $72.73_{\pm 0.07}(50\%)$ | $69.01_{\pm 0.23}(10\%)$ | $\mathbf{63.54}_{\pm 0.35}(10\%)$ |
| Layer-wise | $\mathbf{73.97}_{\pm 0.13}(10\%)$ | $\mathbf{72.74}_{\pm 0.28}(10\%)$ | $68.79_{\pm 0.22}(10\%)$ | $62.94_{\pm 0.03}(10\%)$ |

Table 6: Οι παρουσιαζόμενες ακρίβειες είναι οι κορυφαίες που επιτεύχθηκαν μαζί με τα αντίστοιχα ποσοστά πτώσης βαρών

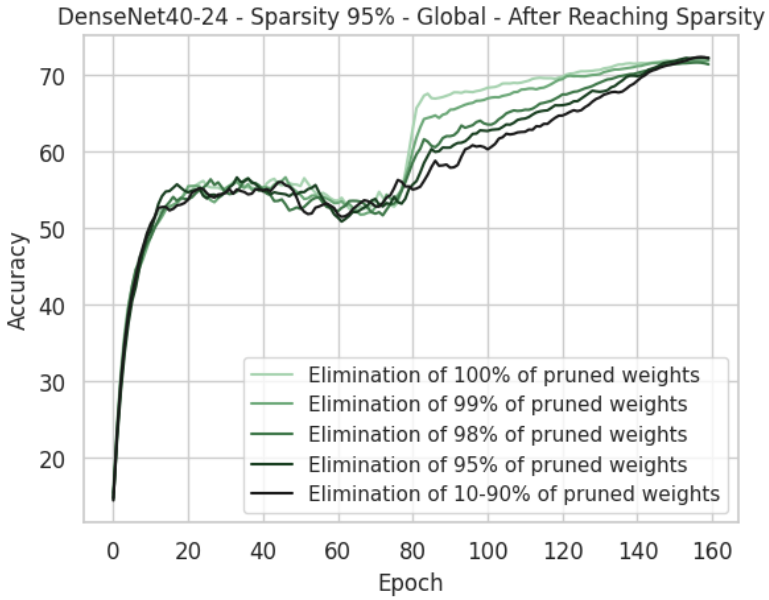
| Ratio | 90% | 95% | 98% | 99% |
|---|---|---|---|---|
| ResNet-20 (1.096M Params): 73.59 $\pm 0.44$ | | | | |
| From the Beginning of Training | | | | |
| Global | 10% | 50% | 50% | 75% |
| Layer-wise | 25% | 50% | 50% | 50% |
| After Reaching Sparsity | | | | |
| Global | 90% | 90% | 90% | 95% |
| Layer-wise | 25% | 95% | 95% | 95% |
| MobileNetV1 (3.315M Params): 71.15 $\pm 0.17$ | | | | |
| From the Beginning of Training | | | | |
| Global | 50% | 75% | 90% | 90% |
| Layer-wise | 25% | 50% | 75% | 75% |
| After Reaching Sparsity | | | | |
| Global | 75% | 90% | 95% | 98% |
| Layer-wise | 50% | 50% | 90% | 90% |
| DenseNet40-24 (0.714M Params): 74.70 $\pm 0.51$ | | | | |
| From the Beginning of Training | | | | |
| Global | 50% | 50% | 10% | 50% |
| Layer-wise | 75% | 50% | 50% | 50% |
| After Reaching Sparsity | | | | |
| Global | 98% | 98% | 99% | 98% |
| Layer-wise | 98% | 98% | 98% | 90% |

Table 7: Πίνακας που παρουσιάζει σε ποιο ποσοστό η απόδοση μειώνεται σημαντικά μαζί με την πτώση

### Σύγκλιση Απόδοσης

Εξετάζουμε τα πρότυπα ανάκτησης της ακρίβειας και τον ρυθμό σύγκλισης μετά από επιθετική μόνιμη απομάκρυνση βαρών. Τα δύο μικρότερα μοντέλα (ΡεσΝετ-20 και ΔενσεΝετ40-24) παρουσίασαν μία ενδιαφέρουσα συμπεριφορά όταν περισσότερο από το 95% των βαρών τους αφαιρέθηκε μόνιμα από τη διαδικασία βελτιστοποίησης. Μόλις επιτευχθεί το επιθυμητό ποσοστό αραιότητας και αφαιρεθεί το καθορισμένο ποσοστό των βαρών, και τα δύο δίκτυα παρουσίασαν μια έντονη άνοδο στην ακρίβεια, ακολουθούμενη από μια πιο αργή και σταδιακή αύξηση, που τελικά συγκλίνει περίπου στην εποχή 140. Αυτό έρχεται σε αντίθεση με την τυπική συμπεριφορά, δηλαδή σύγκλιση περίπου στην εποχή 155, γεγονός που δείχνει ότι η εκπαίδευση ουσιαστικά ολοκληρώθηκε νωρίτερα. Αυτό το μοτίβο μπορεί να αποδοθεί στη ξαφνική αύξηση της σταθερότητας της μάσκας κλαδέματος $\mu$.

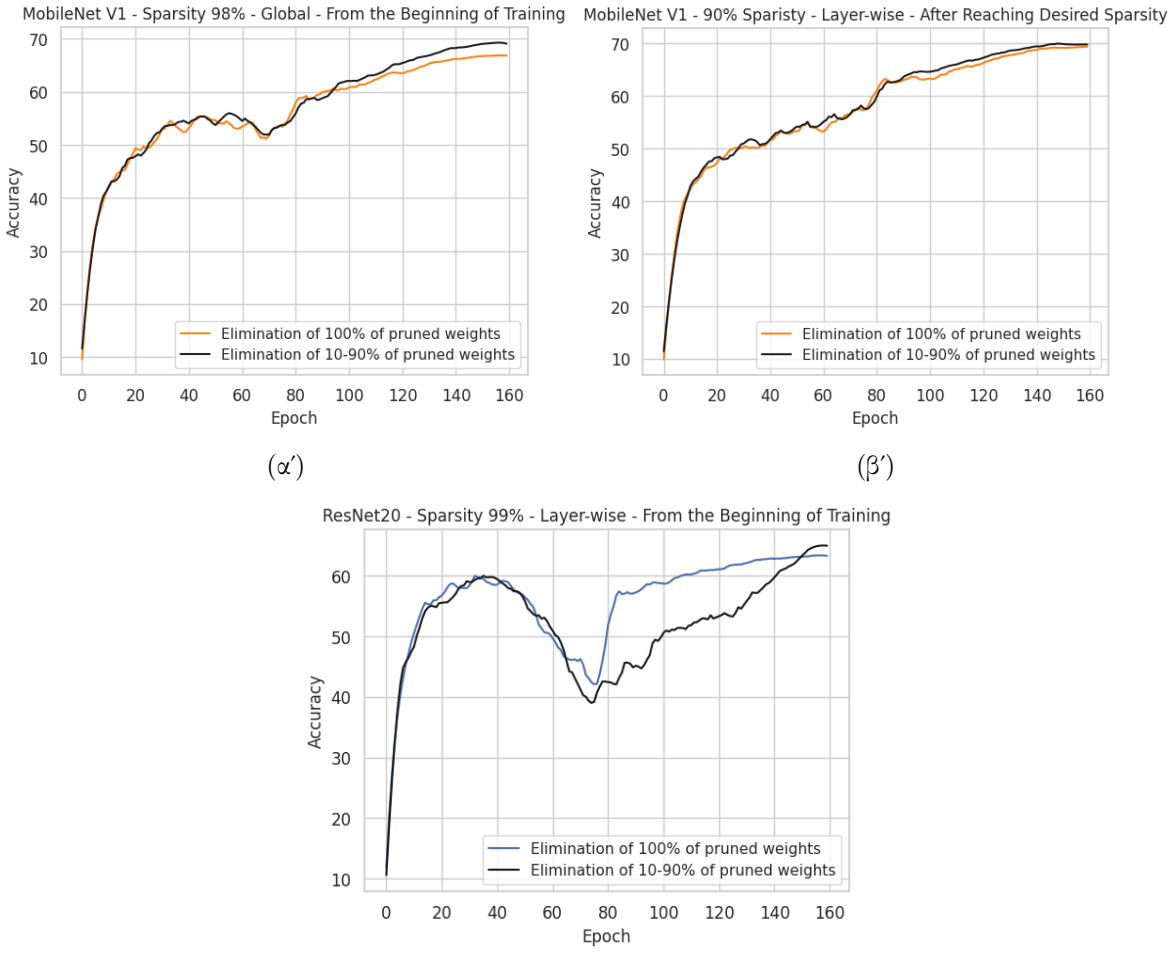
Το Σχήμα 24 απεικονίζει αυτό το φαινόμενο για το ΔενσεΝετ40-24, παρουσιάζοντας τις καμπύλες εκπαίδευσης για τέσσερα ποσοστά απομάκρυνσης βαρών μετά την επίτευξη της αραιότητας, με τη χρήση καθολικού κατωφλίου (95%, 98%, 99% και 100%).




Σχήμα 24: Ακρίβεια του DenseNet40-24 με στόχο αραιότητας 95% και απομάκρυνση βαρών μετά την επίτευξη της αραιότητας χρησιμοποιώντας την καθολική μέθοδο. Κάθε χρώμα αντιστοιχεί σε διαφορετικό ποσοστό απομάκρυνσης, ενώ η μαύρη γραμμή δείχνει την απόδοση για πιο συντηρητικές τιμές (10-90%).

Το Σχήμα 25 παρουσιάζει τρία ακόμη παραδείγματα αυτού του φαινομένου, δείχνοντας τις καμπύλες για απομάκρυνση 100% και τις πιο συντηρητικές τιμές 10-90%. Το ResNet-20 εμφανίζει παρόμοια συμπεριφορά με το DenseNet40-24.

Αντιθέτως, το μεγαλύτερο μοντέλο, το MobileNet V1, δεν παρουσίασε την ίδια έντονη άνοδο στην ακρίβεια ούτε πρώιμη σύγκλιση μετά την επίτευξη της αραιότητας, ανεξαρτήτως του πότε έγινε η απομάκρυνση των βαρών (είτε από την αρχή της εκπαίδευσης είτε μετά την επίτευξη της αραιότητας).



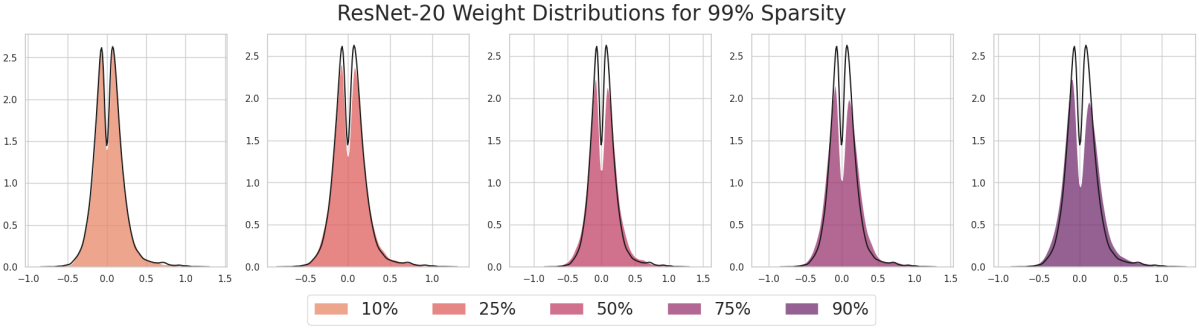

(α′) (β′)

(γ′)

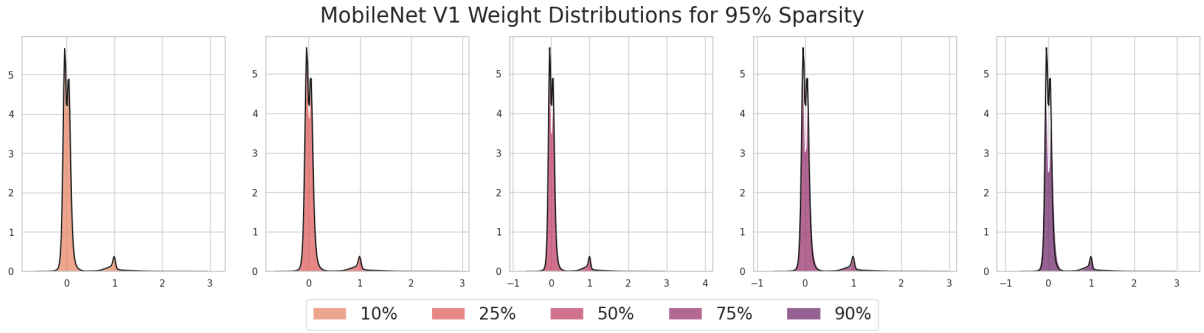
Σχήμα 25: Σύγκριση της συμπεριφοράς σύγκλισης μεταξύ διαφορετικών μοντέλων και ρυθμίσεων απομάκρυνσης βαρών.

## Κατανομές Βαρών

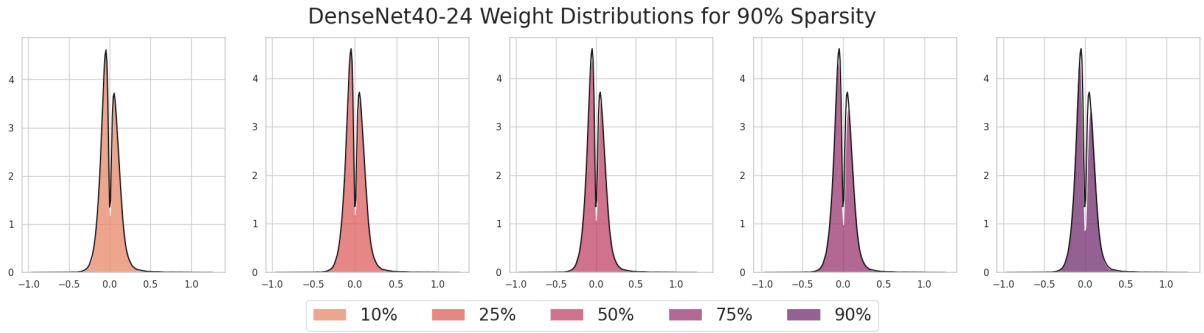
Παρουσιάζονται ενδεικτικά κατανομές βαρών για διαφορετικά μοντέλα και ποσοστά απομάκρυνσης. Παρατηρούμε και εδώ ο γεγονός ότι όσο περισσότερα βάρη εκτοπίζουμε από την διαδικασία της βελτιστοποίησης τόσο περισσότερες συνδέσεις χάνονται, με αποτέλεσμα να έχουμε πολύ διαφορετικές κατανομές.




Σχήμα 26: Κατανομές βάρους ResNet20 με στόχο αραίωσης 99% για όλα τα ποσοστά πτώσης βαρών με την καθολική μέθοδο που διαγράφει τα βάρη από την αρχή της εκπαίδευσης.




Σχήμα 27: Κατανομές βάρους MobileNet V1 με στόχο αραίωσης 95% για όλα τα ποσοστά πτώσης βαρών με την καθολική μέθοδο που διαγράφει τα βάρη από την αρχή της εκπαίδευσης.




Σχήμα 28: Κατανομές βάρους DenseNet40-24 με στόχο αραίωσης 90% για όλα τα ποσοστά πτώσης βαρών με την καθολική μέθοδο που διαγράφει τα βάρη από την αρχή της εκπαίδευσης.

# 7 Επίλογος και Μελλοντικές Κατευθύνσεις

Η εργασία αυτή επικεντρώνεται στη βελτίωση της αποτελεσματικότητας και της σταθερότητας του κλαδέματος σε νευρωνικά δίκτυα μέσω επεκτάσεων στο Feather. Προτείνονται τρεις κύριες συνεισφορές: (1) μελέτη της σημασίας των κλαδεμένων βαρών, δείχνοντας ότι ορισμένα μπορούν να αγνοηθούν μόνιμα χωρίς απώλεια ακρίβειας, (2) δυναμική συνάρτηση κλιμάκωσης παραγώγων που προσαρμόζεται βάσει της αραιότητας, και (3) προσαρμοστικός προγραμματισμός κλαδέματος με βάση τη σταθερότητα των μασκών αραίωσης. Τα αποτελέσματα δείχνουν βελτιώσεις στην ακρίβεια και τη σύγκλιση, ενώ ανοίγουν δρόμους για περαιτέρω βελτιστοποίηση και συνδυασμό με άλλες τεχνικές.

# Chapter 1

# Introduction

Contents

## 1.1 Motivation

In recent years, deep learning has become the dominant approach in a wide range of machine learning applications, including computer vision [49], natural language processing [21], and multi-modal applications [76] to name a few. The success of deep neural networks in these domains is attributed to the fact that modern architectures can be designed with as many parameters as necessary to yield such results, which can be in the millions or even billions [19].

Such DNNs require substantial computational resources, both during training and inference. This creates several practical challenges, particularly when discussing deployment in resource constrained devices, such as mobile devices, embedded systems, wearables, and robots. The growing demand for these models in real-time and low-power applications has made crucial the need for more efficient models that can deliver high performance with minimal resource consumption.

Another concern arising from the large resource demands of DNNs is the environmental impact of training and deploying them. Studies have shown that the carbon footprint of training a single SoA neural network can be substantial, prompting the machine learning community to explore more sustainable alternatives [99].

In response to the aforementioned issues, extensive research is done on model compression [16], [19], [59]. Among several compression techniques, neural network pruning has received much attention, providing various modules to sparsify and therefore lighten large DNNs [9], [15]. Pruning removes redundant parameters from large models, such as weights, neurons, filters or even entire layers. Interestingly, research has shown that large models that are pruned up to a moderate sparsity ratio outperform smaller dense models with the same number of parameters [24], [27], [105].

The Feather [27] framework was developed as a highly effective approach utilizing an enhanced version of the STE [7], which prunes weights during the forward pass but allows gradients to flow freely and be updated during the backward. Feather combines magnitude-based unstructured pruning with a cubic sparsity scheduler [105] and a gradient scaling mechanism to enable training of extremely sparse networks. Feather was a primary motivation for this thesis, and as such all experiments will be run as additions to it. However, all three contributions can be run with different sparsification modules, which could prove their versatility.

The gradient scaling factor in Feather is fixed throughout training, which may not reflect the varying sensitivity of different layers or sparsity levels. In addition, most unstructured dense-to-sparse sparsification modules use a cubic pruning schedule, which is predetermined and static, offering no adaptability based on the stability of the pruning mask during training in extreme sparsity ratios. The rigidity of these parameters is what motivated this thesis, which will explore more dynamic approaches to realize whether adaptability improves performance.

## 1.2 Contributions

As mentioned, this thesis is motivated by ideas of developing more effective, principled, and adaptive pruning methods in place of static ones. Specifically, it introduces two core contributions: a gradient scaling function that dynamically adjusts learning behavior during training, and an adaptive pruning scheduler that adjusts the pruning rate based on the stability of the pruning mask. Together, these techniques aim to improve the reliability of pruning at extreme sparsity, enhance model performance, and offer insights into the actual significance of pruned weights.

- **Gradient Scaling Function:** A dynamic scaling mechanism is proposed for the Feather pruning framework, replacing its static hyperparameter with a function that adjusts during training. This new approach enhances learning efficiency in early training stages by facilitating greater gradient flow, thereby mitigating early-stage over-pruning.
- **Adaptive Pruning Scheduler:** An adaptive scheduler is developed to control the progression of sparsity throughout training. By quantifying the stability of pruning masks using metrics like the

Jaccard similarity index, this method dynamically adjusts pruning aggressiveness to avoid destabilizing the model. Experimental results demonstrate that the scheduler improves both accuracy and mask stability, particularly under very high sparsity ratios.

- A quantitative study of pruned weights' significance, identifying when certain weights can be permanently removed without negatively impacting model performance. This exploration can guide the development of more efficient pruning techniques and inform decisions about optimization strategies in sparse training.

## 1.3 Thesis Outline

This thesis is made up of seven chapters, including the Introduction. The topics discussed are as follows:

- Chapter 2 introduces the basic concepts of machine and deep learning crucial for the understanding of the work of this thesis.
- Chapter 3 provides an overview of the most important deep neural network compression methods, with an emphasis to neural network pruning. The chapter concludes with a summary of Feather [27], which is the module used for all further experiments.
- Chapter 4 introduces the proposed gradient scaling function, explains its formulation, and presents experimental results demonstrating its advantages over the simple static scaling approach.
- Chapter 5 presents the adaptive pruning scheduler, including its theoretical foundation, implementation, and performance evaluation.
- Chapter 6 investigates the importance of pruned weights, analyzing the difference between full STE-based training and permanent elimination of low-magnitude weights.
- Chapter 7 summarizes all previous three chapters and offers ideas for future work in each presented application.

# Chapter 2

# Theoretical Background

Contents

## 2.1 Machine Learning

This section aims to present an overview of the principles of machine learning [8], [28], to a better understanding of the following work.

Machine learning is a domain of Artificial Intelligence that studies the principles, methods, and algorithms that enable computer systems to learn and make predictions based on past evidence and experience rather than relying on explicitly coded instructions. This allows systems to become autonomous, adapting to changing environments and generalizing their knowledge to new, unseen scenarios. Machine learning not only improves a system's adaptability and generalization capabilities but also serves as an attempt to understand the mechanisms behind human learning better. Its significance spans various applications, from autonomous vehicles to personalized recommendations, making it an essential field in modern artificial intelligence.

### 2.1.1 Types of Machine Learning

To develop an optimal prediction algorithm, it is necessary to fully define the problem and find an appropriate dataset that will be used to train the algorithm. This dataset will determine the feedback given to the model during training, so we can divide machine learning into three categories depending on the nature of the data: Supervised, Unsupervised, and Reinforced Learning.

#### Supervised Learning

During supervised learning the model is trained on labeled datasets, meaning that each input sample is paired with a corresponding correct output. The goal of supervised learning is to learn a mapping function that can accurately predict outputs for new inputs based on that exact function it managed to learn during training. Some common supervised learning algorithms are decision trees [70], support vector machines [18] and k-nearest neighbors.

#### Unsupervised Learning

In unsupervised learning the model is trained on unlabeled data. This means that the algorithm must identify patterns and relationships within the data without predefined output labels. The trained model creates its classes by grouping similar samples. Unsupervised learning is commonly used for tasks such as clustering, where data points are grouped based on similarity. Popular algorithms for unsupervised learning include k-means clustering [84], hierarchical clustering [71], and autoencoders [6].

#### Reinforced Learning

In reinforced learning the model learns to make decisions by interacting with an environment and receiving feedback in the form of rewards or penalties. Unlike supervised learning, where the model learns from labeled data, reinforcement learning relies on trial and error to discover optimal strategies. Reinforced learning is widely used in applications such as robotics, game playing, and autonomous systems, where an agent interacts with the provided environment and is either rewarded or punished for each action. Some reinforced learning algorithms include Q-learning [97], policy gradients, and deep reinforcement learning methods such as Deep Q-Networks [69].

### 2.1.2 Supervised Machine Learning Tasks

The two most common tasks in supervised machine learning are regression and classification. Both involve predicting outcomes based on input data with their corresponding labels, but they differ in what they predict and how.

#### Regression

During the training of a regression algorithm, the relationship between the samples and their corresponding target variable is fitted into the best possible curve. By analyzing patterns in the training

data, regression identifies trends and correlations, making it particularly effective for tasks like forecasting, trend analysis, and numerical prediction.

#### Classification

Classification predicts a discrete value which is one of a finite set of available categories or classes. Common applications of classification include spam email detection, image recognition, sentiment analysis, and medical diagnosis, where the goal is to accurately determine the class or label for each data point based on its features rather than a continuous value.

The differences between the two tasks are highlighted in n Figure 2.1, where it becomes evident that in a classification task the resulting curve divides the data into distinct classes and the regression task fits a curve as well as possible to all data.

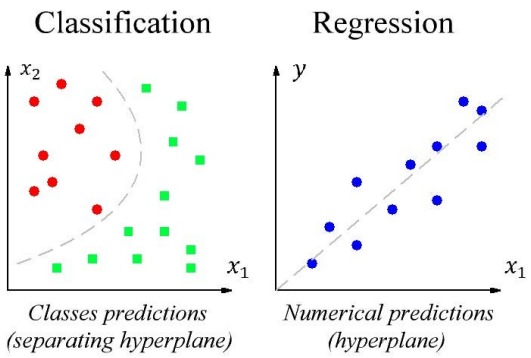



Figure 2.1: Difference between classification and regression. From [72]

### 2.1.3 Train, Test and Validation Sets

In machine learning, the dataset provided for the training of the model is usually split into three subsets. These are the training set, the validation set, and the test set. The training set is used to train the model by using all the samples with their corresponding labels, allowing the algorithm to form relationships between data and targets. The validation set is used during the training process to fine-tune the model's hyperparameters and evaluate its performance on further data, something that ensures the model generalizes well. Finally, the test set is used after training is complete to evaluate the model and prove that it can generalize and has learned efficiently, rather than adapting perfectly to only the training dataset.

### 2.1.4 Evaluation Metrics

In supervised classification tasks, there is a need to evaluate the model's performance.

After training and testing, all samples will be assigned with a label predicting the class in which they must belong. These samples then can belong to one of the four categories:

- True Positives (TP): The sample was correctly predicted to belong to the class
- False Positives (FP): The sample was falsely predicted to belong to the class
- True Negatives (TN): The sample was correctly predicted not to belong to the class
- False Negatives (FN): The sample was falsely predicted not to belong to the class

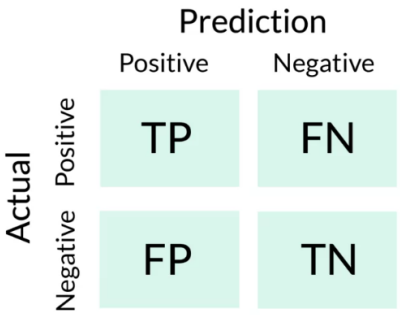



Figure 2.2: Visualization of all four categories for one class. From [43]

### Accuracy

Accuracy is one of the most common evaluation metrics for a model's performance and it represents the percentage of overall correct predictions.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \cdot 100\%$$

### Precision

Precision is a metric used for a single class, and it calculates the ratio of all the samples correctly identified as belonging to that class over the total number of samples predicted to belong to the class, whether they were accurately predicted or not.

$$precision = \frac{TP}{TP + FP} \cdot 100\%$$

### Recall

Recall (or sensitivity) is also used for each class and it represents the ratio of all the samples correctly identified as belonging to that class over the total number of samples that belong to that class.

$$recall = \frac{TP}{TP + FN} \cdot 100\%$$

### F1-Score

F1-Score understands the important information represented in both precision and recall and combines them with their harmonic mean.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \cdot 100\%$$

## 2.1.5 Common Loss Functions

Another way to evaluate the performance of a machine learning model is through loss functions, which are especially useful in regression tasks where the previous evaluation metrics cannot be used. They are also differentiable, which means that they can be utilized during training for gradient-based updating of the parameters (denoted as $\theta$) [80], in contrast to accuracy or F1-score. These algorithms aim to minimize the loss function $L(\theta)$, which usually represents the distance between the actual labels $y$ and the predicted ones $\hat{y}$. In this section we will list the four most common loss functions. The first two are used for regression tasks whereas the third and fourth are used for classification.

### Mean Square Error (MSE)

This loss function calculates the average squared difference between the actual labels and the predicted ones. By squaring the errors, the function makes sure that large errors are given more attention by the model to correct.

$$L_{MSE}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2$$

### Mean Absolute Error (MAE)

MAE measures the average absolute error between the actual labels and the predicted ones. In contrast to MSE, it gives equal attention to all errors, making it a preferable metric when the training dataset has large variation and outliers in its samples.

$$L_{MAE}(\theta) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y_i}|$$

### Cross-Entropy Loss

Cross-Entropy loss is the most widely used loss function for classification tasks in machine learning. To better understand this metric, it is import to first define cross-entropy. Cross-entropy between two discrete probability distributions $p$ and $q$ over the same underlying set of events (class) $X$ is a measure of how different the two distributions are and is mathematically defined as:

$$H(p, q) = -\sum_{x \in X} p(x) \cdot \log q(x)$$

For binary classification tasks where the label $y$ can either be 1 or 0, the binary cross-entropy loss over N samples is as follows:

$$L_{BCE}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y_i}) + (1 - y_i)\log(1 - \hat{y_i})]$$

This loss increases the more the predicted labes differ from the correct ones, penalizing greatly incorrect assumption. It can be extended for multiclass classification as such:

$$L_{CCE}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{M} y_{i,k} \cdot \log(\hat{y_{i,k}})$$

In the above function, M denotes the number of classes in the classification problem, and as such it calculates the cross entropy loss for each class. It represents the probability that the $i$-th sample belongs to class $k$.

### Kullback-Leibler Divergence Loss

Kullback-Leibler (KL) loss is similar to the cross-entropy one. Specifically, it measures how much information is lost when one distribution $q$ is used to approximate another distribution $p$ (with the same support $\mathcal{X}$) by utilizing the Kullback-Leibler divergence metric between two distributions:

$$D_{KL}(p||q) = \sum_{x \in \mathcal{X}} p(x) \cdot \log(\frac{p(x)}{q(x)})$$

## 2.1.6 Overfitting and Underfitting

Two very common issues that arise with machine learning models are overfitting and underfitting.

Overfitting in machine learning happens when a model learns the training data exceptionally well, understanding and learning both the desired patterns and the noise and randomness of some data. This results in the model achieving a very high performance on the training data, giving the illusion of a very well trained model. However, when it is evaluated on the test set it shows very poor results, as it has failed to generalize effectively. Overfitting can happen when the model is too complex for the amount of data it is trained on. For example, having too many parameters or features allows it to simply learn the exact training data rather than noticing patterns. It can also happen when the dataset is too small, allowing the model to learn only a few test cases without the need to generalize.

Underfitting on the other hand occurs when a model is too simple to identify the desired patterns in the training data, which yields a poor performance both on the training set and on the testing set. It most commonly happens when the model is too simple, having too little trainable parameters to effectively learn the data provided. It can also occur if the model is not trained for enough iterations or if the data is noisy, having too many outliers, therefore making it difficult for the model to learn.

Finding the right fitted model for the desired task involves figuring out a balance between underfitting and overfitting. This means designing a model which has sufficient parameters in accordance to the given data, but not too many, to ensure it generalizes well to both the training and test data. A correctly fitted model captures the underlying patterns in the data without being too simplistic (which leads to underfitting) or too complex (which leads to overfitting).

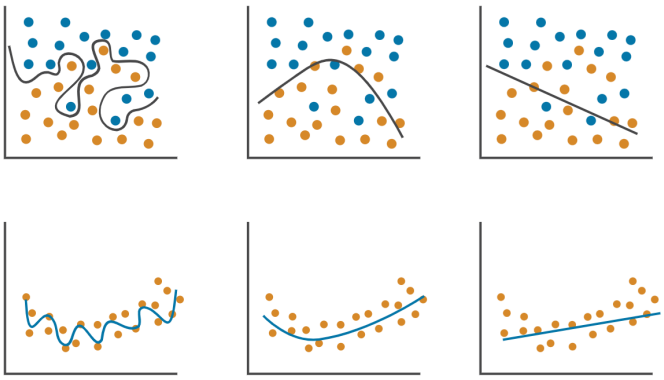


Figure 2.3: Examples of Overfitting, Right Fit, and Underfitting for a classification (up) and a regression task (down). From [67]

### 2.1.7 Regularization

Regularization in machine learning is a set of techniques used to prevent overfitting by adding constraints or penalties to the learning algorithm. This encourages the model to generalize unseen data better.

The first two regularization techniques presented here typically involve modifying the loss function by introducing a regularization term denoted as $R(\theta)$, which is scaled by a suitable parameter $\lambda$. The loss function is then as follows:

$$L_r(\theta) = L(\theta) + \lambda \cdot R(\theta)$$

The third method is commonly used in neural networks, which will be discussed further in this chapter.

#### $L_1$ Regularization

Commonly referred to as LASSO Regularization, it utilizes the $L_1$-norm of the parameters and promotes sparsity by encouraging some weights to become zero.

$$R_1(\theta) = ||\theta||_1 = \sum_i |\theta_i|$$

The result is a sparser model, which improves generalization by ignoring certain features deemed too unimportant.

#### $L_2$ Regularization

Also known as Ridge Regression or Weight Decay, it uses the $L_2$-norm and is used to discourage large weights by applying a squared penalty.

$$R_2(\theta) = ||\theta||_2 = \sum_i \theta_i^2$$

By decaying the larger weights, the model encourages more conservative solutions, which can prevent overfitting very efficiently.

#### Dropout

Dropout is an effective regularization technique for neural networks that enhances generalization and mitigates overfitting. During each training epoch, each neuron has a small probability $p$ of having its weight set to zero. This process dynamically alters the network's architecture throughout training, promoting robustness and reducing reliance on specific neurons. The concept is inspired by random forest algorithms [10], where multiple decision trees contribute to the final prediction.

## 2.2 Deep Learning and Neural Networks

After providing the basics of machine learning, we introduce deep learning [1], [52], which is a subset of machine learning. The core difference between the two is that deep learning utilizes neural networks with multiple layers to manage large, raw data, such as images, text or sound. Where machine learning algorithms need the data to be manually and appropriately transformed, deep learning neural networks receive them as they are, performing the necessary transformations at some of their layers. The way neural networks are constructed aims to model the function of the human brain, where pattern recognition and feature extraction are much more abstract. This is why neural networks are preferred for tasks such as object detection in images, text and speech recognition, and image classification, which are performed over too complex data for traditional machine learning algorithms to handle.

Deep learning networks have millions or more trainable parameters (weights), which is why it is crucial for them to be properly trained, using loss functions as evaluation metrics. In this section, we will provide an overview of the ways the weights are updated during training, using optimizers and backpropagation.

In this thesis we discuss the compression of neural networks, which is why we will now offer an overview over their basic principles and the most common architectures of convolutional neural networks, which are the subject of our pruning module.

### 2.2.1 Perceptron and Feed Forward Networks

The perceptron, visualized in Figure 2.4, is the fundamental structural unit of neural network architectures and resembles structurally and fundamentally a neuron. In its own right, it functions as a binary classifier designed to predict whether a given input belongs to one of two classes by learning a linear decision boundary. A perceptron is comprised of input features, weights that represent the importance of each feature, a bias term that allows the model to shift the decision boundary, and an activation function that determines the output based on the weighted sum of inputs. The perceptron updates its weights during the training process to minimize classification errors, in a way that will be analyzed in further sections.

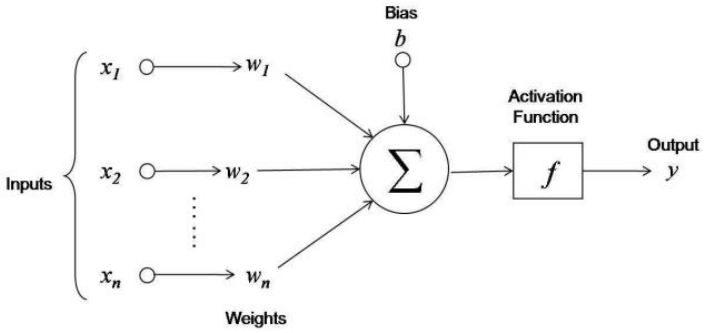



Figure 2.4: A single neuron used in neural networks. From [92]

Mathematically, the output $y$ of a single neuron is as follows, in which equation N is the total number of input features, $x_i$ each of those features, $w_i$ its corresponding weight, $b$ the bias term and $f$ the activation function.

$$y = f(\sum_{i=0}^{N}[w_i \cdot x_i + b])$$

According to the specific requirements of each classification problem, different activation functions can be employed to enhance the performance of neural networks. Here, we present the five most commonly used activation functions:

- Identity: $f(x) = x$
- ReLU: $f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$
- Binary Step: $f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$
- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
- Tanh: $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

It is interesting to note that for multiclass classification problems a popular activation function is the Softmax function, which converts raw output scores into probabilities that sum to 1, allowing the model to predict the likelihood of the input to belong to each class.

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{N} e^{x_k}}, \ for \ i = 1, ..., N$$

By using such neurons as building blocks, more complex neural network architectures can be designed by creating layers with multiple perceptrons. This modifies the network to be able to handle multiclass classification problems, extending beyond the binary classification capability of a single perceptron. In these architectures, called multi layer perceptrons (MLPs) or deep feedforward neural networks (FNNs), neurons are organized into several layers: an input layer, one or more hidden layers, and an output layer. In the case that each neuron in a layer is typically connected to all neurons in the subsequent layer, as shown in Figure 2.5, the network is called a fully connected neural network (FCNN) and it is a subcategory of FNNs.

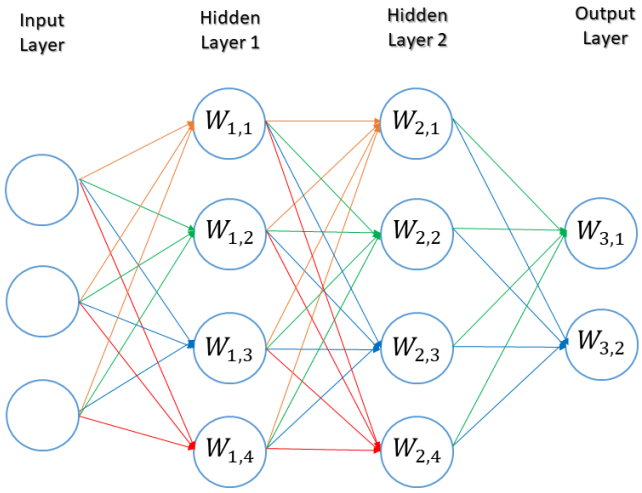



Figure 2.5: Fully connected neural network model architecture. From [55]

Another wide subcategory of FFNs are the convolutional neural networks (CNNs), which are specifically optimized for processing and classifying images. Similarly to FCNNs, CNNs are designed by organizing multiple neurons into multiple layers, using unique architectures designed to extract and process the desired features from the input. These architectures will be thoroughly discussed further in this chapter, after first providing an overview of the most important concepts of training FFNs.

### 2.2.2 Backpropagation

Most neural networks are trained using optimization algorithms, which will be discussed in the following section, that rely on the calculation of gradients to update the model's weights. These gradients decide the weight adjustments needed during training, since they are based on the loss function's gradient with respect to the weights. The primary method used to compute these gradients is backpropagation, which is based on the application of the chain rule [14]:

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

The process consists of two main steps which are discussed below, the forward pass and the backward pass.

#### Forward Pass

During the forward pass, the model performs the necessary calculations in all successive layers using the current weight values for any given input in order to provide the final output. It starts with the input layer, then calculates the hidden layers and finally it calculates the output layer, storing all results in the process.

#### Backward Pass

After the output is calculated, the flow of the data is reversed, going now from output to input. This is called the backward pass, and it distributes the loss of the output to the entire network by utilizing gradients. During this process, the partial derivatives of the loss function with respect to the model's weights are calculated. The chain rule is used to avoid redundant computations, and it efficiently calculates the derivative of the loss function with respect to the weights, the inputs and all intermediate layers, helping the model to learn how to minimize the loss for each of these cases.

### 2.2.3 Optimizers

Optimizers are algorithms designed to minimize the loss function during training by adjust the network's parameters (weights and biases). In this section we will discuss the most common optimizers [102] used in training of convolutional neural networks, which are the object of this thesis. The first two discussed are variations of gradient descent, the third introduces momentum, then we present two adaptive gradient based learning algorithms and finally one that combines momentum and adaptive gradient.

#### Vanilla Gradient Descent

Vanilla gradient descent is the most basic form of the gradient descent optimization algorithm. It iteratively updates model weights by computing the gradient of the loss function with respect to these weights and moving in the opposite direction of the gradient to minimize the loss. Mathematically it can be expressed as:

$$\theta_k = \theta_{k-1} - \eta \cdot \nabla_\theta \mathcal{L}(\theta_{k-1};\ \mathcal{T})$$

where $\theta$ are the model's updateable parameters, $\mathcal{L}(\tilde{\boldsymbol{w}}_k;\ \mathcal{T})$ the loss function with respect to the previous weights and the training set $\mathcal{T}$ and $\eta$ the learning rate.

The learning rate is a model hyperparameter that defines the step size at which the model updates its weights, determining how much the model learns from the gradient at each iteration. A small learning rate results in slow convergence, which can require many iterations to reach the optimal solution. However, a high learning rate can cause the model to overshoot the minimum or even diverge, preventing convergence altogether. There has been lately a technique known as learning rate scheduling, which is used to adjust the learning rate during training instead of keeping it a constant. Common strategies include step decay, where the learning rate drops at fixed intervals, exponential decay, where it decreases continuously over time, and adaptive methods like Adam, which will be discussed further in this section. Other approaches, such as cyclical learning rates, oscillate between bounds to help escape local minima, while warm-up and cool-down schedules gradually increase or decrease the learning rate for better stability. A well defined learning rate scheduler can lead to faster training, better generalization, and improved optimization efficiency.

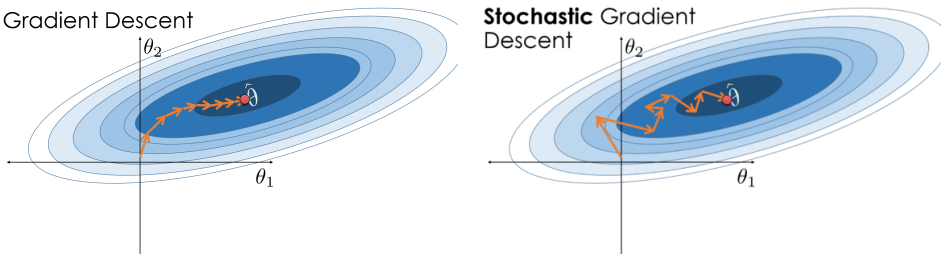
While vanilla gradient descent is one of the most simple and effective optimization algorithms, it still has limitations, such as slow convergence and sensitivity to the learning rate, making it less practical for large datasets. Furthermore, this algorithm can falsely converge to a local minimum instead of the global one. This is why several variants were developed, which will be further discussed.

#### Stochastic Gradient Descent

Vanilla gradient descent computes the necessary gradients with the entire training set samples, which as mentioned can be inefficient for large datasets. Stochastic gradient descent (SGD) tackles this problem by computing the gradients per sample, which can be very fast but may also cause great variations to the loss function, due to the fact that a single sample may not be representative of the entire dataset. Figure 2.6 visualizes these fluctuations, comparing to vanilla gradient descent which is smoother but slower. Although the learning process can be highly variable, this randomness also allows it to escape local minima.

For the individual sample pair $(x_i, y_i)$ the updating equation is now:

$$\theta_k = \theta_{k-1} - \eta \cdot \nabla_\theta \mathcal{L}(\theta_{k-1};\ x_i, y_i)$$




(a) Vanilla Gradient Descent (b) Stochastic Gradient Descent

Figure 2.6: Vanilla and Stochastic Gradient Descent visualization. From [75]

### Mini-Batch Gradient Descent

Mini-batch gradient descent manages a balance between the vanilla and stochastic variants by updating model parameters using a small batch of samples rather than the entire dataset or a single one. This approach reduces the high variance of stochastic gradient descent while being more computationally efficient than vanilla gradient descent. If we denote as $\mathcal{B}$ a batch of samples, the updating equation is now:

$$\theta_k = \theta_{k-1} - \eta \cdot \nabla_\theta \mathcal{L}(\theta_{k-1};\ \mathcal{B})$$

In most deep learning tasks, mini-batch gradient descent is referred to simply as SGD and the batch size is provided as a hyperparameter before training, having values of 64, 128, 256 etc.

### SGD with Momentum

Momentum is introduced into the SGD algorithms as a way to smooth fluctuations and speed up the learning process, while avoiding convergence to local minima. The following formulas show the parameter updating with momentum:

$$\Delta v_k = \rho \cdot \Delta v_{k-1} + (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta_{k-1};\ \mathcal{B})$$

$$\theta_k = \theta_{k-1} - \eta \cdot \Delta v_k$$

$v_k$ is the momentum term introduced as a way to keep track of all gradients during the entire training cycle with $\rho \in [0, 1]$ denoting the weight of momentum. This formulation incorporates an exponentially decaying moving average of past gradients into the current update term, which causes a momentum-like effect that enhances both convergence stability and speed.

### Adaptive Gradient Algorithms: Adagrad and RMSprop

Some optimizers are adaptive gradient based, which means that the variables are updated with adaptive learning rates rather than a stable or scheduled one.

Adagrad (Adaptive Gradient) is a gradient method that scales each learning rate of a parameter based on a cumulative sum of squared gradients. The approach allows sparsely updating parameters to be assigned more weight in their updates over frequently updating parameters, making it very useful in dealing with sparse data. One of the shortcomings of Adagrad is that it allows the sum of squared gradients to become unbounded, causing a learning rate that is constantly decreasing to lead to premature stalling of learning.

RMSprop (Root Mean Square Propagation) mitigates Adagrad's issue with earning rate diminishing by introducing a moving squared gradient average. This average manages to suitably decay the weights of historical accumulated gradient instead of accumulating all previously computed. This prevents the learning rate to rapidly drop in account of large accumulative gradients.

### Adaptive Moment Estimation (Adam)

The Adam optimization algorithm combines the advantages of momentum and adaptive gradient algorithms. This combination along with its computational efficiency and low memory requirements makes Adam one of the most popular optimizers for most deep learning tasks.

Adam maintains an exponentially decaying average of past gradients (first moment estimate) and an exponentially decaying average of past squared gradients (second moment estimate). This allows the optimizer to maintain learning rates suitable for each specific parameter, improving performance on problems with sparse gradients and noisy data.

## 2.2.4 Convolutional Neural Network (CNN) Architecture

After providing an overview of the most crucial paradigms in deep learning and neural networks, we now analyze the architecture of CNNs, which are the backbone of all further sections presented in this work.

As previously mentioned, CNNs are are designed and optimized for processing and classifying images. They receive raw images with dimensions $W \times H \times D$ (width, height and depth) as the input data and provide prediction scores for each possible class as output. Since they are designed exclusively for images, they are able to extract and process important features from the provided pixels, allowing them to effectively learn patterns. In CNNs, each layer has a specific task to perform, and can therefore be categorized as a a convolutional layer, a pooling layer, a batch normalization layer and a fully connected one. Figure 2.7 visualizes a common CNN architecture with such layers.

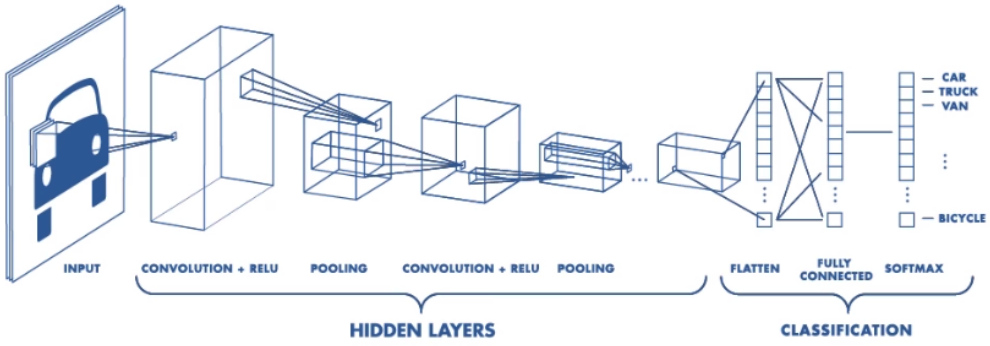



Figure 2.7: CNN model architecture. From [100]

### Convolutional Layer

Convolutional layers are the most fundamental layers in CNNs and perform the most important tasks of feature extraction, yielding outputs that are called feature maps. They apply a set of $K$ filters with dimensions $F \times F$ with learnable parameters that slide over the input performing convolutions, thus producing feature maps which contain the most important aspects of the image, such as edges, textures, shapes etc. These filters are usually small with common dimensions being $F = 3$ or $F = 5$ and manage to be applied to the entire input. Each filter is responsible for capturing specific features, allowing the network to learn hierarchical representations of the data.

The input of each convolutional layer will have dimensions of $W_i \times H_i \times D_i$. To obtain the layer's output dimensions we must also provide two additional parameters, stride and padding.

**Stride** $S$ is the step size by which the kernel is slid on the input. For $S = 1$, the filter performs the necessary convolutions by sliding one pixel each time, for $S = 2$ two pixels, etc.

**Padding** $P$ is the number of zeros to be appended to the borders of the output of the layer, aiming to match the output dimensions with the inputs.

As such, the dimensions of the feature map will be $\frac{W_i-F+2P}{S+1} \times \frac{H_i-F+2P}{S+1} \times K$.

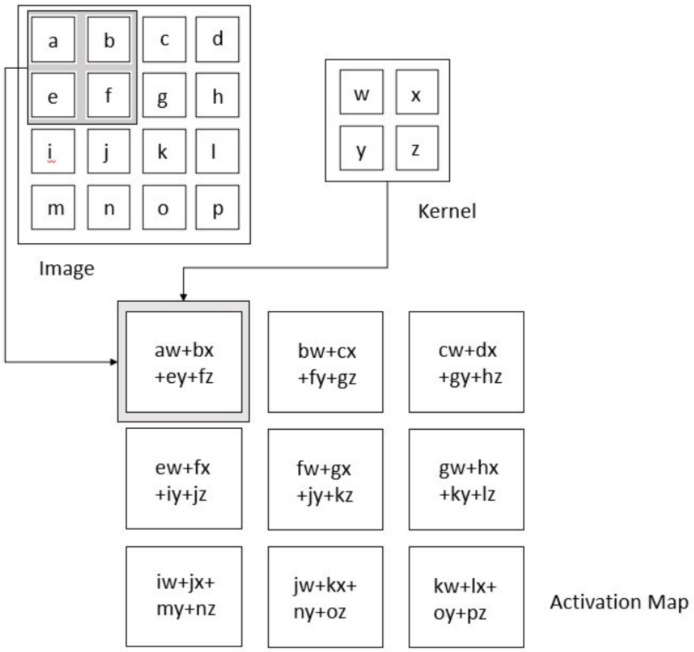



Figure 2.8: Example of a convolutional operation with $S = 1$ and $P = 0$. The input has dimensions $4 \times 4 \times 1$, the one filter $2 \times 2$ and the output $3 \times 3 \times 1$. From [28]

### Pooling Layer

A pooling layer is usually between two subsequent convolutional layers and is used to reduce the spatial dimensions of the feature map, therefore reducing the number of learnable parameters for each layer. This helps the model become faster and less computationally efficient, while also improving the CNN's ability to generalize, since fewer parameters mean a smaller chance to overfit to the training data. Furthermore, downsampling the activation map helps the network to become invariant to small translations or distortions in the input, making it more robust.

The pooling layer, like the convolutional, slides a filter across the input. This filter can either be a max pooling one or an average pooling one. Max pooling provides the maximum element of the feature map's selected area, preserving the most important features. Average pooling calculates the average of all the elements in that area, which has a smoothing effect on the feature map.

While pooling has several advantages to overall performance, it is important to note that the dimensionality reduction can result to the loss of fine details in the image, while also smoothing out important features. This makes the selection of the filter's hyperparameters, such as filter size, type, and stride, crucial.

### Batch Normalization Layer

During CNN training, as the parameters of the layers are updated, the distributions of said parameters also change. This means that at each layer, the inputs belong to different distributions making the layers more disconnected with each other, which makes it difficult for the network to learn and converge efficiently. This problem is called the covariate shift problem.

The batch normalization layer [44] alleviates this problem by normalizing the input, as the name suggests, stabilizing the model and speeding up the training process. The normalization process involves the calculation of the mean and the variance of every batch and shifting it so that the outputs obtain a mean

close to 0 and a standard deviation close to 1. This ensures that the data are always from the same distribution, which solves the internal covariate shift problem, thus allowing for larger learning rates with faster convergence, something previously difficult to achieve.

### Fully Connected Layer

The final layer in every CNN is a fully connected one (FC), which provides the final outputs which are the probability scores for each available class. It is basically a feed-forward network. Since these networks require a 1D input rather than the 3D produced by the rest of the CNN, the final feature map must be flattened.

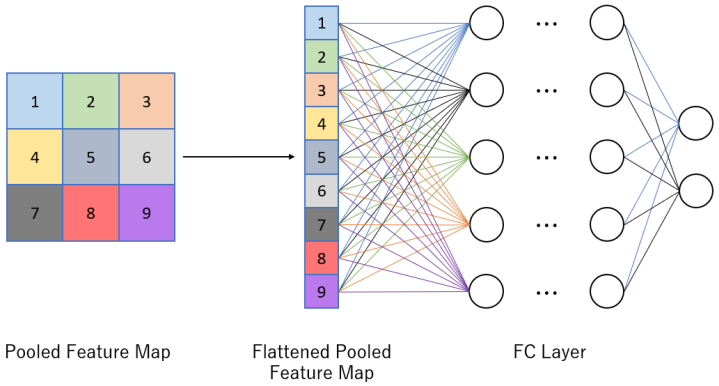



Figure 2.9: Flattening of a feature mask and its subsequent feeding as input to an FC layer in a CNN. From [3]

# Chapter 3

# Literature Review

## Contents

## 3.1 Introduction

In the last few years, with the continuous development of deep neural networks, artificial intelligence has made large leaps in various areas. Such areas are Computer Vision (CV) [34], [49], Natural Language Processing (NLP) [21], Audio and Speech Recognition (ASR) [37] and multi-modal applications [60], [76]. However, to achieve excellent performance in these applications, the DNNs rely from a few million to several billion parameters, making the computational cost massive [19]. They require more memory, processing power, and energy consumption, presenting challenges for deployment in resource-constrained environments like mobile devices, embedded systems, wearables, and robots. Another issue with modern DNNs is that the existence of so many trainable parameters leads to several of them being rendered redundant, compromising their robustness and making them vulnerable to adversarial attacks [82]. Finally, in recent years there have been environmental concerns about the carbon footprint produced by the training and usage of large DNNs, which call for a more sustainable approach in designing and deploying such models [99].

Since DNNs' increasingly large volume presents so many challenges, to tackle them there has been extensive research on compression and acceleration of neural networks [15], [16], [19], [59]. Most techniques aim to compress and accelerate the model as much as possible with a minimal trade-off in performance. This leads to the ability of those models to be deployed in resource-constrained environments without any significant losses, while in some cases enhancing robustness and generalization ability. Moreover, the computational cost is greatly lessened as well as the energy consumption.

In this chapter, we provide an overview of the most common compression methods for DNNs, with the biggest focus being on network pruning. Section 3.2 briefly analyzes the other methods, with section 3.3 focusing exclusively on network pruning, which is the main focus of this thesis. Finally, in section 3.4 the SoA Feather [27] sparse training module is presented, which is the module that all subsequent experiments are run on.

## 3.2 Relevant Compression Methods

In this section, the most common compression methods are presented (except network pruning) along with some influential applications. These methods are quantization [59], low-rank factorization, knowledge distillation [29], and neural architecture search.

### 3.2.1 Quantization

Quantization in general refers to the process of mapping a continuous signal with infinite values to a set of discrete, finite representations, usually integers. As a method, it is commonly used in signal processing as a way to compress raw data, reducing computational demands and processing power, while retaining important information. This idea was adopted in neural network compression, where the quantization of DNNs led to the same advantages while retaining good performance [59]. In this section, we present the two most common quantization methods.

#### Numerical Low-Bit Quantization

Numerical low-bit quantization is the most straightforward quantization method since it involves the quantization of the model's weights and biases in the strict definition. In typical DNNs, the parameter values are stored using 32-bit or 16-bit floating point precision, since those are typically used on GPUs and CPUs [88]. Quantization aims to store the values as INT-8, INT-4, and INT-2, thus decreasing complexity and computational cost. There is also a method that stores the values as INT-1 producing binarized neural networks [42]. Generally, there are two ways to quantize a neural network: by quantizing it after training (post-training quantization) and by training the model with quantized values (quantization-aware training) [59].

Post-training quantization [87] is the process of quantizing a neural network after it has been fully trained with 32-bit or 16-bit precision parameters. This results in a more compact model while also achieving acceleration in inference time, with the cost being reduced accuracy and performance due to

approximation errors, especially when quantizing in an INT-2 or INT-1 format [59]. The acceleration is determined by the hardware and typically ranges from 1-4x.

The issue with approximation errors introduced in post-training quantization is mitigated in quantization-aware training [46], another popular method of compressing a model. It involves training with quantized weights along the entire process, while not quantizing the gradients for the backward pass. This is known as the Straight-Through Estimator (STE) [7], in 2013. The STE treats the quantization operation as an identity function when computing gradients. As a result, the gradients flow through the quantized weights as if no quantization had occurred in the training iteration. This method prevents the model from updating the weights with zeroed gradients, which would be the case if the gradients were quantized. An illustration visualizing the use of the STE during the training process is shown in Figure 3.1

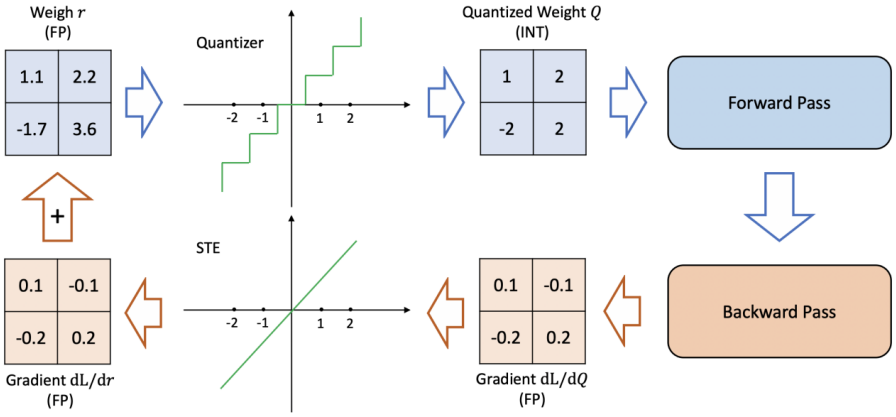



Figure 3.1: Quantization aware training process. From [26]

### Clustering and Parameter Sharing

A different approach in DNN quantization is weight sharing, in which different weights share the same numerical value, thus decreasing the number of unique values stored and reducing the storage space needed for the model. This is typically achieved through clustering algorithms such as k-means [84], which group similar weights into clusters based on some distance metric (eg. Euclidean, Manhattan). Each weight within a cluster is then assigned a shared value, typically the cluster's centroid. During inference, instead of storing the full precision weights individually, only small indices pointing to the corresponding centroids need to be stored, leading to significant compression. However, since the lookup table used for weight reconstruction still operates with the original floating-point precision, the inference speed remains largely unaffected. It still offers considerable advantages, especially for hardware-limited applications, such as FPGA-based AI accelerators, yielding the same performance for smaller memory demands.

## 3.2.2 Tensor Decomposition

Tensor Decomposition or Low-rank factorization is a family of network compression methods that operate by decomposing large-weight matrices into products of lower-rank ones [83]. This approach is particularly useful for compressing fully connected and convolutional layers, where the weights are stored in very large matrices, thus significantly reducing the number of parameters and floating-point operations (FLOPs), enabling efficient inference on resource-constrained devices. Common approaches for CNNs include Singular Value Decomposition (SVD) [20], [73], Tucker Decomposition [47], [57], Canonical Polyadic (CP) Decomposition [51], and Dictionary Learning [79].

### Singular Value Decomposition

SVD is a classical matrix factorization technique widely used for compressing fully connected layers in neural networks. For a weight matrix $W \in \mathbb{R}^{m \times n}$, SVD decomposes it as:

$$W = U \Sigma V^\top$$

Where $U \in \mathbb{R}^{m \times r}$, $\Sigma \in \mathbb{R}^{r \times r}$, and $V \in \mathbb{R}^{n \times r}$ with $r \leq \min(m, n)$ denoting the rank. By truncating the decomposition and retaining only the top $k < r$ singular values, a low-rank approximation of the original weight matrix is obtained. This reduces memory and computation while maintaining representational capacity [20], [73].

### Tucker Decomposition

Tucker decomposition generalizes SVD to higher-order tensors by factorizing a tensor into a smaller core tensor multiplied by a matrix along each mode. For a 3D convolutional kernel tensor $\mathcal{W} \in \mathbb{R}^{I \times J \times K}$, the Tucker decomposition approximates it as:

$$\mathcal{W} \approx \mathcal{G} \times_1 A \times_2 B \times_3 C$$

where $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ is the core tensor, $A \in \mathbb{R}^{I \times R_1}$, $B \in \mathbb{R}^{J \times R_2}$, and $C \in \mathbb{R}^{K \times R_3}$ are factor matrices along each mode, and $\times_n$ denotes the mode-$n$ tensor-matrix product.

Tucker decomposition is particularly effective for compressing convolutional layers in deep neural networks by preserving the core structure of the tensor while discarding redundancy [47], [57].

### Canonical Polyadic Decomposition

Canonical Polyadic decomposition factorizes a tensor into a sum of tensors with a rank of one. For a 3D weight tensor $\mathcal{W} \in \mathbb{R}^{I \times J \times K}$, it approximates:

$$\mathcal{W} \approx \sum_{r=1}^{R} a_r \circ b_r \circ c_r$$

Where $a_r \in \mathbb{R}^I$, $b_r \in \mathbb{R}^J$, and $c_r \in \mathbb{R}^K$, and $\circ$ denotes the vector outer product. This method is particularly suitable for compressing convolutional kernels due to its effectiveness in reducing parameters and operations [51].

### Dictionary Learning

Dictionary learning is an unsupervised technique that represents data as a sparse linear combination of basis elements (atoms) from a learned dictionary. For weight approximation:

$$W \approx DX$$

Where $D$ is the dictionary and $X$ contains the sparse codes. This approach supports efficient storage and computation, especially when $X$ is highly sparse. Dictionary learning can be applied post-training or as a training regularizer, and has shown success in compressing both convolutional and dense layers [79].

## 3.2.3 Knowledge Distillation

Knowledge distillation is a model compression technique where a small and compact network called the student is trained to mimic the behavior of a larger one, the teacher, which is usually a pre-trained large model [29], [38]. Instead of training the student with the standard method using the ground-truth labels, it is trained to match the outputs (soft probabilities) produced by the teacher. These outputs distill the student with information about class relationships more effectively than hard labels alone, enabling the student to generalize better with fewer parameters and little computational cost. Bucilua et al. [11] introduced the concept in 2006, which then got popularized by Hinton et al. [38] in 2015, who proved

that the same compact model performed notably better when using knowledge distillation rather than being trained from scratch.

The knowledge distillation method proposed by Hinton at al. [38] is response-based knowledge, since it operates on the predictions made by the teacher. In the literature there exist several types of knowledge that can be extracted from the teacher, such as feature-based knowledge and relation-based knowledge, where the knowledge extracted is provided from intermediate layers and feature maps respectively [29].

During response-based knowledge distillation, the student network is trained to minimize a combination of the standard task loss and a distillation loss that encourages the student to replicate the teacher's behavior. A typical formulation of the total loss is:

$$\mathcal{L} = (1 - a)\mathcal{L}_{student} + a\mathcal{L}_{distillation}$$

Where $\mathcal{L}_{student}$ is the cross-entropy loss between the ground-truth labels student's predictions, $\mathcal{L}_{\lceil\rangle\int\sqcup\rangle\updownarrow\updownarrow\dashv\sqcup\rangle\wr\backslash}$ is the Kullback-Leibler divergence between the softened student and teacher outputs, and $a$ is a balancing hyperparameter. The softened outputs (logits) for both student $s$ and teacher $t$ for the $i$-th class are computed as:

$$p_i^{t \text{ or } s} = \frac{exp(\frac{z_i}{T})}{\sum_j exp(\frac{z_j}{T})}$$

Where $T > 0$ is a temperature parameter which when high enough smooths the output probabilities, making it easier for the student to capture dark knowledge about the relationships between classes [29].

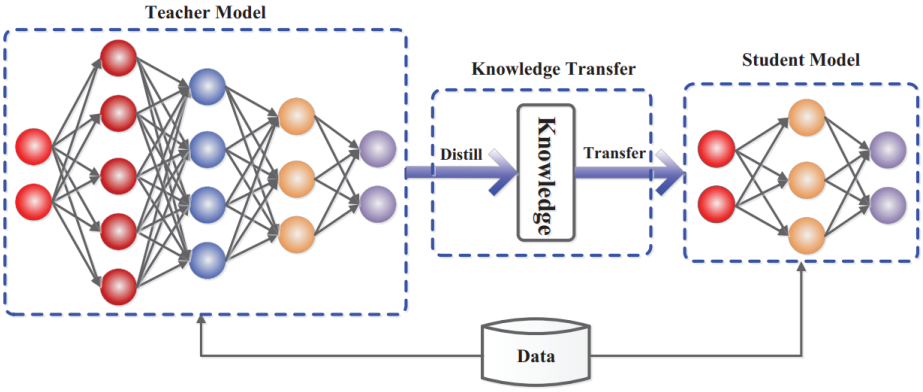



Figure 3.2: Visualization of the teacher and student models and the knowledge distillation process. From [29]

### 3.2.4 Compact Model Design and Neural Architecture Search

Instead of compressing large, over-parameterized neural networks, an alternative approach is to design compact architectures with the explicit goal to deploy it in resource-constrained environments such as mobile devices, embedded systems, or edge computing platforms. This design process can be carried out manually or automatically through Neural Architecture Search [77].

Two of the most prominent models in compact model design are MobileNets and DenseNets. MobileNet V1 [40] was proposed by Howard et al. and introduced the use of depthwise separable convolutions, a factorized version of the standard convolution, to significantly reduce computational complexity while preserving accuracy. MobileNet V2 [40] further improved performance by incorporating inverted residual blocks and linear bottlenecks.

DenseNets [41] was developed by Huang et al. as CNNs for object recognition tasks, achieving high accuracy with much less parameters than other CNNs. This was achieved by employing dense connectivity between layers to improve feature reuse and gradient flow, where each layer receives as input the

concatenation of all preceding layers' outputs. Despite its deeper structure, DenseNet requires fewer parameters and less computation than comparable architectures due to its high feature reuse and the ability to maintain compact growth rates.

In contrast to compact model design by hand, Neural Architecture Search (NAS) is an automated method for designing neural networks for tasks and deployment constraints that has lately gained increased research interest. It was initially introduced to improve the accuracy of the model and the design process, while also addressing the limitations of manual architecture design by exploring vast search spaces for the optimal design [77].

In the context of compression, NAS is used to discover architectures that inherently require fewer parameters and operations. This is distinct from classic compression techniques such as pruning or quantization, as NAS generates architectures from scratch or through fine-tuning. For example MobileNet V3 [39], which improved the Mobilenet family presented above by using NAS techniques to maximize efficiency, and EfficientNet [90] were partially discovered or refined using NAS approaches focused on optimizing the trade-off between accuracy and resource usage on mobile devices.

## 3.3 Neural Network Pruning

Neural network pruning is a model compression technique used to reduce the size and complexity of a neural network by eliminating parts that are considered less important or redundant. The main objective is to make the model smaller and more efficient, ideally without significantly affecting its performance. It is one of the most popular compression methods, with research starting as early as 1989 with "Optimal Brain Damage" by LeCun et al [53] and 1992 with "Optimal Brain Surgeon" by Hassibi and Stork [33]. In later years, with SoA neural networks yielding unprecedented performances, neural network pruning is at the forefront of research as a means to contain the ever growing amount of trainable parameters [15].

Pruning works by identifying and removing redundant or non-essential components of a network, such as individual weights, neurons, channels, or even entire layers. Depending on the level of granularity, pruning methods can be categorized into unstructured pruning, which removes individual weights leading to sparse weight matrices, and structured pruning, which eliminates groups of parameters in a way that preserves the network's original structure and enables efficient hardware acceleration. These will be discussed in the Section 3.3.1, the first of this part, along with semi-structured pruning, a later development in the literature.

Beyond the basic distinction in structure, pruning strategies also differ in terms of criteria and timing, discussed in Sections 3.3.2 and 3.3.3 respectively. Neural network pruning has also lately been enhanced by combining it with the other compression techniques discussed before in this Chapter, in order to complement the methods strenghts and weaknesses. Although these methods are beyond the scope of this thesis, they will be briefly discussed in Section 3.3.4 for the sake of continuity.

### 3.3.1 Pruning Methods

Pruning can be categorized by the granularity of sparsified elements into unstructured, structured, and semi-structured [15]. Unstructured pruning allows the pruning algorithm to freely remove any specific parameters it deems necessary, while structured pruning aims at removing entire groups of parameters, like vectors or entire filters. Semi-structured pruning is a recently developed method that combines the advantages of both structured and unstructured pruning, in ways further analyzed later in this section. Figure 3.3 provides a visualization of these pruning methods for further clarity.

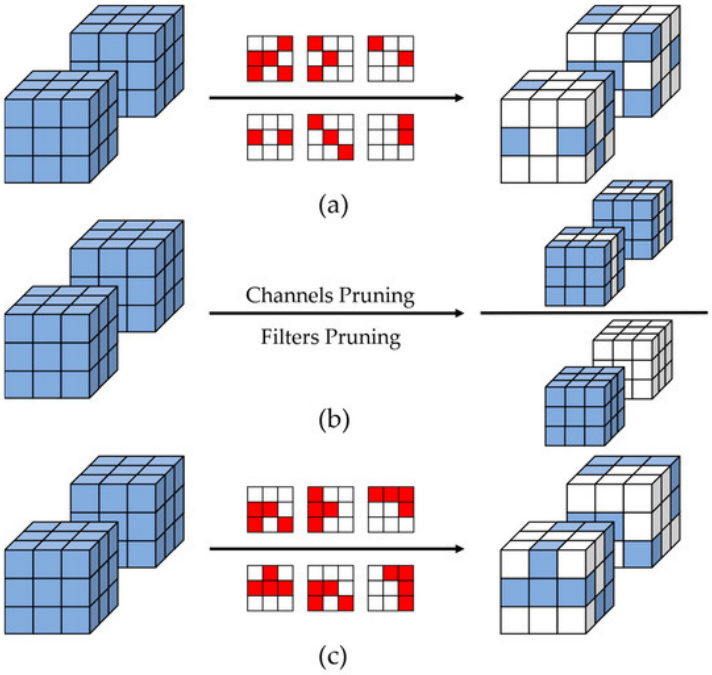



Figure 3.3: A visualization of pruning methods in a CNN where the blue weights are the ones retained, the white the ones pruned and the red are the pruning masks: **(a)** Unstructured Pruning **(b)** Structured Pruning **(c)** Semi-Structured Pruning. From [86]

### Unstructured Pruning

Unstructured pruning is the finest-grained type of the three methods, removing specific weights from any part in the model without taking into account the relative position in which they are in [15]. This approach leads to sparse weight matrices, which means that the distributions of the non-zero parameters are irregular, therefore leading to unstructured sparsity patterns. This irregularity allows the model to reach very high sparsity ratios with a minimal negative impact on performance [25].

However, for the same reason, it can be challenging to fully leverage the efficiency gains on standard hardware, requiring specialized libraries or hardware accelerators to observe a substantial speed-up [96]. There has been extensive research to develop the suitable hardware to support speed-up in unstructured pruning, which involves acceleration in sparse matrix operations [45].

### Structured Pruning

Structured pruning is a model optimization technique that removes entire structures within a neural network in various granularities [2]. This includes neurons [64], which is the finest-grained type, filters [101], or channels [62], rather than individual weights. By eliminating larger, well-defined components, structured pruning creates a more compact and efficient model that maintains a regular architecture, making it easier to accelerate on standard hardware like CPUs, GPUs, or FPGAs [36]. This approach reduces both the number of parameters and the computational load, leading to faster inference times without requiring specialized sparse matrix operations, unlike the unstructured counterpart. However, the trade-off between structural integrity and accuracy is large, which means that with structured pruning the pruning target must remain conservative or else there will be a significant drop in performance [25].

### Semi-Structured Pruning

Also known as pattern pruning [96], semi-structured pruning has recently developed as a way to leverage the advantages of both the structured and unstructured variants, while also discarding as many disadvantages as possible [81]. N:M structured sparsity [103] was the first instance of semi-structured sparsity

that balanced between the two aforementioned types, providing fine-grained sparsity and requiring specialized hardware while also maintaining structure. Meng et al. in [68] (2020) propose pruning stripes from filters in CNNs rather than entire filters or individual weights, resulting in both heightened accuracy and regularity in sparsity patterns, combining the performance and the speedup in most available hardware. This combination of advantages can be ideal for application in autonomous vehicles (AVs), where speed and accuracy are of the essence. R-TOSS by Balasubramaniam et al. [5] (2023) utilizes semi-structured pruning in such a way that proves object detection can be both fast and accurate in AVs and their hardware.

### 3.3.2 Pruning Criteria

While we explored the methods by which pruning can be applied, it is also important to discuss the criteria by which the weights, kernels, layers etc are eliminated. [15], [96]

#### Magnitude-based Pruning

Magnitude-based pruning is one of the most widely used and intuitive criteria for pruning neural networks. It is based on the idea that weights with smaller absolute values contribute less to the model's output and can therefore be removed with minimal loss in performance. Han et al. (2015) [32] in their deep compression work popularized magnitude-based pruning by proving it could significantly reduce model size without noticeable drops in accuracy. In practice, after or during training, a threshold $T$ is applied, and all weights below that threshold are pruned away while the rest keep a non-zero value as follows:

$$\tilde{w} = \mathcal{P}_T(w) = \begin{cases} f(w), & \text{if } |w| \geq T \\ 0, & \text{if } |w| < T \end{cases}$$

The $f(w)$ function provides the model with the non-zero value of the remaining weights and can be implemented through hard or soft thresholding [23]. In hard thresholding, the weights keep their initial value:

$$f_h(w) = w$$

This creates a sharply sparse model but introduces non-differentiability at the pruning operation, complicating training in the backward pass. Soft thresholding offers a smoother alternative, shrinking the surviving weights toward zero by

$$f_s(w) = \begin{cases} w - T, & \text{if } w > T \\ w + T, & \text{if } w < T \end{cases}$$

This allows the pruning effect to be integrated during optimization, gradually encouraging sparsity rather than enforcing it abruptly. However, in the case where some weights are set exactly to zero, soft thresholding still has non-differentiable operations.

To address this non-differentiability in both hard and soft thresholding operators, the Straight-Through Estimator (STE) [7] is often employed, approximating the backward gradient of the pruning function as the identity function. This allows gradients to flow through and enables continued training, where the pruned weights are still present in the optimization process. Research showed that allowing pruned weights to be optimized rather than completely eliminated significantly improved performance, since important connections were allowed to develop instead of being completely pruned from the beginning [7], [27].

These techniques can be applied in an unstructured manner, removing individual weights resulting in irregular sparsity, or in a structured manner, where entire neurons, filters, or channels are pruned based on aggregate magnitude metrics like $L_1$ or $L_2$ norms. For example, Li et al. [56] prune entire CNN filters by scoring their average magnitudes using the $L_1$ norm.

Although magnitude pruning is simple and quite effective, since it does not consider the sensitivity of the network to individual weights or structures, it may struggle to maintain performance at extremely high sparsity levels [9], [24].

### Sensitivity-based Pruning

Sensitivity-based pruning techniques aim to remove weights from a neural network based on their impact on the model's loss function, rather than simply their magnitude. These methods evaluate how sensitive the loss is to changes in each parameter, pruning those weights whose removal causes the smallest increase in error. Unlike magnitude-based pruning, which assumes that small weights are unimportant, sensitivity-based approaches directly estimate the importance of each weight in maintaining the model's predictive performance. As a result, they can produce more compact and accurate models, especially at high sparsity levels where magnitude-based pruning methods fall short [33], [53].

One of the earliest and most influential sensitivity-based methods is Optimal Brain Damage (OBD) by LeCun et al. [53] in 1989. In this method, the sensitivity of each weight is estimated using a second-order Taylor expansion of the loss function, incorporating information from the diagonal elements of the Hessian matrix, which are equivalent to the second derivatives with respect to each weight. Weights are ranked based on their estimated contribution to the loss, and those with the least impact are pruned first. Optimal Brain Surgeon (OBS) by Hassibi and Stork [33] in 1993 extended this idea by considering the full Hessian matrix rather than just its diagonal, allowing for even more precise weight removal by accounting for the interactions between different parameters.

Although these methods were proposed as a more accurate alternative to magnitude-based pruning, they were applied to the neural networks available back then, which were rather small compared to those available now. Computing and inverting the full Hessian for millions of parameters is computationally expensive [59]. Due to this computational overhead, sensitivity-based pruning is less common in modern literature but has inspired several methods utilizing low-cost approximation approaches, such as Woodfisher [85] and EigenDamage [94]. Another notable example is SNIP by Lee et al. [54], which proposes a saliency criterion called the connection sensitivity criterion. It identifies important connections before training by measuring how sensitive the loss function is to each individual weight at initialization, utilizing first-order gradients rather than second. This makes it rather computationally efficient, compared to OBD and OBS.

### Regularization-based Pruning

While the previous two methods directly set parameters to zero, regularization-based pruning techniques introduce sparsity in a more indirect way. Instead of pruning weights based on heuristic criteria like magnitude and loss change, these methods train the model in a way that it naturally sets some parameters close to zero during optimization. This is achieved by adding a regularization term ($L_0$, $L_1$, $L_2$ norms) to the loss function. These weights are then usually pruned using another criterion, such as magnitude [96].

While $L_0$ regularization directly penalizes the number of nonzero weights, making it ideal for pruning, it is non-differentiable. Louizos et al. [66] address this by introducing a differentiable approximation using the hard-concrete distribution. Their method enables sparse training and can be implemented for both structured and unstructured sparsity.

Wen et al. [98] introduced structured sparsity learning by applying $L_1$ and group LASSO regularization to architectural components such as neurons, filters, and channels. This regularization framework is ideal for structured pruning without requiring a separate pruning step, something that mitigates the computational overhead.

$L_2$ regularization (weight decay) is traditionally used to improve generalization as discussed in Section 2.1.7, but it can also assist in pruning since it shrinks (decays) less important weights. Han et al. [31] introduced this concept for unstructured pruning, which while it is not aggressive in driving weights to zero, it supports subsequent threshold-based pruning in unstructured pruning schemes. Another method is proposed by Chin et al. [17] which employs a dynamically increasing $L_2$ regularization

term during training. This growing penalty progressively encourages structured or unstructured sparsity, enabling the network to prune unimportant weights while preserving important structures.

#### Random Pruning

Random pruning is used as a control method in various works, such as [9], offering a way to measure the effectiveness of more intelligent pruning strategies. It involves removing weights, neurons, or structures from a neural network without using one of the aforementioned pruning criteria [63]. This results in significant performance degradation, especially in higher sparsity ratios, whereas works like [63] show that it can be quite effective for low targets. The Lottery Ticket Hypothesis by Frankle and Carbin [24] (2018) is a prime example of using random pruning to evaluate their methods. In this work, not all sparse subnetworks are equally capable of learning. Even when reset to their original initialization, they fail to match the performance of the intelligently picked winning tickets, which are sparse subnetworks that can train effectively from initialization. This comparison highlights that intelligent pruning criteria, rather than random removal, are essential.

### 3.3.3 Pruning Timeframe

The timeframe of pruning refers to when the pruning process is applied relative to the training of a neural network. Different strategies have emerged based on whether pruning occurs before training, along training, or after training, each with its own philosophy, advantages and challenges.

#### Pruning Before Training

Pruning before training, also called foresight pruning in [95] or pruning at initialization in [91] and [54], is a sparse-to-sparse training approach with the objective of reducing the inference time even during training [15].

One of the most influential works in this area is the Lottery Ticket Hypothesis (LTH) proposed by Frankle and Carbin in 2018 [24], in which they proved that within randomly initialized networks, there exist sparse subnetworks (winning tickets) that can be trained by themselves to achieve comparable accuracy to the original large and dense model. Lee et al. [54] introduced SNIP, a method that prunes a network before training begins by evaluating the sensitivity of the loss function to each weight at initialization, therefore pruning in a single shot without the need for computationally expensive iterative procedures.

Other techniques, such as GraSP [95] and SynFlow [91], both proposed in 2020, further refined SNIP's idea by proposing pruning criteria in order to stabilize the gradient signal throughout training, something that rendered SNIP unable to perform well in large sparsities. GraSP retains gradient flow throughout the network, whereas SynFlow avoids the pruning of entire layers (layer collapse) by only pruning weights with the lowest synaptic strengths.

#### Pruning Along Training

Pruning along training is a dense-to-sparse pruning approach that has recently gained popularity [4], [27], [89], [93], [104], [105]. In this approach, weights are pruned not before or after the training process but with it. The core idea is that pruning happens gradually during training, by updating a weight, filter or kernel mask $M$ (usually binary) which then produces the pruned weight matrix $\mathcal{W} = M \odot W$ [15]. This mask can be a thresholding function as discussed in Section 3.3.2, with the amount of weights to be eliminated (sparsity ratio) in each iteration predefined by a scheduler. The simplest scheduler is the ramp, which starts from 0% sparsity and then linearly reaches the target ratio at the desired epoch. In most cases, after reaching the target sparsity ratio, there exists a training phase where the final pruning rate is kept constant, to allow the network to adjust for the sparsity, instead of ending training when the desired sparsity is reached. Another popular scheduler is cubic, first used by Zhu and Gupta in [105]. They proposed this scheduler by noticing that a network can reach low levels of sparsity fairly fast without compromising performance, therefore it would be beneficial to prune aggressively in

the beginning of training and smoothly when reaching the target. Figure 3.4 shows both schedulers for a few different targets.

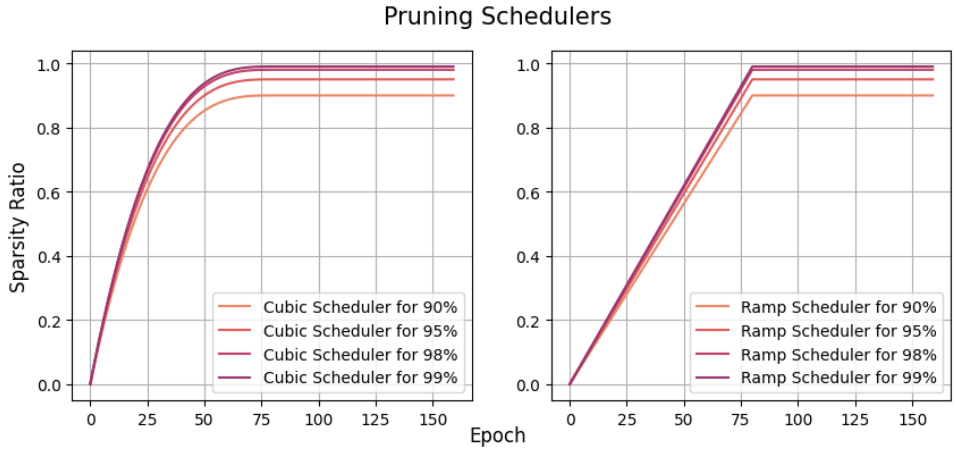



Figure 3.4: Cubic and ramp pruning schedulers for reaching various sparsity ratio targets in 80 epochs out of the 160 total

A large discussion in pruning along training concerns the handling of backpropagation through the thresholding operator that determines which weights are pruned. The most straightforward and simple approach used in earlier works is to apply the thresholding operation during both the forward and backward passes, removing pruned weights from the optimization process entirely [105]. This however has since proven to drop performance significantly, since it can result in premature decay of important connections, especially during the very early stages of training. For this reason, many methods employ the Straight-Through Estimator (STE) [7] discussed in Section 3.3.2, which approximates the gradient of the non-differentiable thresholding operation using the identity function. This allows all gradients to flow through during backpropagation, allowing them into the optimization process to be updated and potentially recovered from premature pruning in future steps. Modern pruning approaches such as Feather [27] and ST-3 [93] utilize the STE by combining it with advanced thresholding or soft-thresholding operations, aiming for the stability of gradients. Feather interestingly compared their method with and without utilizing the STE and noted a difference of more than 10% in accuracy for large sparsity ratios and 3% for low. The way Feather operates will be further discussed in Section 3.4, since it is the core of the experiments for this thesis.

### Pruning After Training

Pruning after training is the most intuitive dense-to-sparse pruning approach in neural network pruning with earlier pruning modules, such as OBD [53] and OBS [33], as well as some more recent ones [32], [56], [65] employing it. The intuition is that since the model has already been trained, then it has learned effectively how to perform its objective task, and therefore it will be easier to identify redundant or less significant components such as weights, filters or layers that can be safely pruned.

Deep Compression by Han et al. [32] in 2015 popularized the training pipeline used in modern applications. This pipeline is comprised by three steps: train a dense network to full accuracy, prune weights with magnitudes below a threshold, and then retrain (fine-tune) the sparse model to regain any lost accuracy. The magnitude-based criterion assumes that smaller weights contribute less to the final output and thus can be removed without severely impacting performance. Pruning introduces a sudden drop in model capacity, which is why fine-tuning the model after pruning is crucial. It is important to note that the two final steps can be iterated more than once, up until the performance is acceptable. Because of this, such methods can require a longer training time than usual, since they could need many cycles of pruning and fine-tuning [65].

### 3.3.4 Fusion of Pruning with Other Compression Methods

While pruning is one of the most popular compression techniques, in later years there has been extensive research in enhancing it by combining it with the other model compression techniques. This includes the methods discussed in Section 3.2, meaning quantization, tensor decomposition, knowledge distillation, and neural architecture search (NAS). The fusion explores different forms of redundancy in deep neural networks, broadening the range of available strategies, as well as enabling distinct methods to complement each other, providing their joint advantages and covering each others gaps [15], [36].

**Pruning and Quantization** is the most common combination in literature, used to reduce both the parameter count and the precision of weights. While pruning removes unimportant connections or structures, quantization compresses the remaining weights to lower bit-width representations. Han et al. [31] demonstrated that combining pruning with 8-bit quantization could achieve compression rates of up to 49× without significant loss in accuracy on models such as AlexNet [49].

**Pruning and Tensor Decomposition** explore the redundancy in both weights and structure. Pruning eliminates unnecessary connections, while tensor decomposition methods reduce the dimensionality of weight tensors. For example, Li et al. [57] propose CC (Compressible Convolution), in which channels are pruned according to their relative importance, while the weight tensors are decomposed using Tucker decomposition. This encourages both sparsity and low-rank, leading to models that are not only more compact but also more computationally efficient without compromising performance.

**Pruning and Knowledge Distillation (KD)** are often combined to preserve accuracy during compression. Pruning removes redundant parameters to reduce model size and computation, but in extreme sparsity ratios can drop performance significantly. KD compensates for this loss by transferring knowledge from a large, unpruned teacher model to the pruned student. Chen et al. [13] proved that this method allows the model to regain lost performance much better than simply fine-tuning it after training. Park and No [74] propose the opposite by pruning the teacher model rather than the student, in order to make it more transferable and then distill its knowledge onto the unpruned but significantly smaller student.

**Pruning and Neural Architecture Search (NAS)** can also be combined to create inherently compact networks. Rather than starting with a large model and pruning it, NAS explores architectures under sparsity constraints to discover efficient subnetworks directly. Some approaches, such as Once-for-All (OFA) [12], jointly search and prune across multiple architectures to deploy lightweight models for specific hardware constraints. A more straightforward approach is NPAS [58], which adds pruning algorithms to the search space to find the best possible combination of CNN architecture and pruning method. Similarly TAS [22] also adds pruning into the search space to obtain an already pruned network, with the addition that this network will then benefit from knowledge distillation techniques.

## 3.4 Feather and Relevant Work

### 3.4.1 Unstructured Magnitude-based Dense-to-Sparse Modules

One of the most influential and intuitive works in this area is Gradual Magnitude Pruning (GMP), introduced by Zhu and Gupta in 2017 [105]. GMP progressively removes the weights with the lowest magnitude by using a cubic sparsity scheduler, enabling a smooth dense-to-sparse transition throughout training. Its simplicity and generality have made it a reference point for many subsequent pruning strategies.

GMP removes weights with the smallest magnitudes by employing a hard-thresholding operator. This process is non-differentiable and relies on a fixed pruning schedule and threshold, which may not adapt optimally to the training dynamics. To combat this, STR [50] was proposed, introducing a differentiable soft-thresholding function applied via reparameterization, allowing the sparsity threshold and the weights themselves to be learned jointly through gradient-based optimization. STR encourages weights to shrink smoothly toward zero during training rather than abruptly removing them, thus enabling the network

to adaptively discover which weights to prune. As a result, STR often leads to more stable training and better retention of model accuracy compared to the stepwise, magnitude-based hard pruning strategy of GMP.

ST-3 [93] improves the STR by addressing the gradient flow issues around the thresholding operation more effectively. While the STR uses a soft-thresholding function that is differentiable, it can still suffer from vanishing or very small gradients near the threshold, which can limit how well the network learns under sparsity constraints, thus reducing model accuracy. ST-3 incorporates a Straight-Through Estimator (STE) [7], which allows gradients to flow freely during backpropagation. Additionally, the ST-3 employs a rescaling mechanism that dynamically adjusts weight magnitudes to maintain model efficiency even as pruning progresses. This combination results in more stable and efficient training, learning sparser models with less accuracy loss compared to STR.

The Feather [27] module, studied in this thesis and further discussed in Subsection 3.4.2, also follows the dense-to-sparse paradigm. It applies an STE-compatible pruning mechanism with custom gradient scaling and thresholding. While based on GMP's magnitude principle, Feather introduces innovations in scheduling and differentiability that improve training stability under extreme sparsity.

Spartan, introduced by Tai et al. [89], offers a different approach by proposing a differentiable sparsity mechanism based on optimal transport theory. Spartan defines a sparse mask through a regularized optimal transport problem and employs dual averaging to stabilize learning under sparsity constraints. Unlike dynamic methods, Spartan uses soft top-k projection for differentiability while enforcing hard sparsity in the forward pass. Compared to STR and ST-3, which focus on soft-thresholding mechanisms to induce sparsity, Spartan's optimal transport framework provides a more global and theoretically proven approach to sparse training.

Magnitude Attention-based Dynamic Pruning (MAP) is one of the latest developments in this area, proposed in 2025 [4]. It comprises of two phases, the exploration and the exploitation. The exploration phase utilizes dynamic sparse training by introducing an attention mechanism based on the weight magnitude. Most pruning methods use magnitude as a means to create a binary pruning mask $M$, but MAP creates a continuous-valued magnitude attention mask to train with important weights. This method still allows updates to unimportant weights without using the STE. In the middle of training, the exploitation phase is initiated, in which the sparse structure freezes and the method focuses on optimizing only the most relevant connections. This combination offers enhanced results compared to other methods.

### 3.4.2 Feather

Feather [27] is a magnitude-based, unstructured sparsification module that efficiently prunes weights during training. As shown in Figure 3.5, Feather operates by employing an improved version of the straight-through estimator (STE) [7] that incorporates a novel thresholding operator and gradient scaling mechanism to enhance model sparsification. Feather's pruning process is dense-to-sparse, gradually introducing sparsity during training using a cubic scheduler [105].

Feather achieves SoA results compared to similar modules throughout different training models, especially in extreme sparsity ratios such as 98% and 99% while using both a global magnitude pruning framework [105] and a layer-wise one [78].

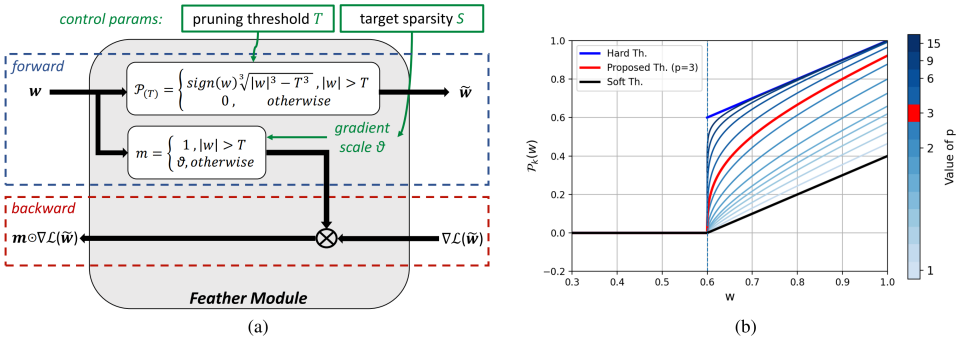



Figure 3.5: Feather Module. From [27]

### STE Thresholding Operator

Using the STE Feather as well as several SoA modules in unstructured pruning achieve high performance in sparse training. Sparse training with the STE involves applying the pruned version of the weights during the forward pass, and the unpruned during the backward pass. This is achieved by treating the thresholding function $\mathcal{P}_{(T)}$ (where $T$ denotes the pruning threshold), which performs the pruning, as an identity function when computing gradients. As a result, the gradients flow through the pruned weights as if no pruning had occurred. This allows the model to successfully sparsify the model while still keeping the advantages gained from a fully dense optimization process.

$$\tilde{\boldsymbol{w}}_k = \mathcal{P}_{(T)}(\boldsymbol{w}_k)$$

$$\mathcal{P}_{(T)} = \begin{cases} sign(w) \cdot \sqrt[p]{|w|^p - T^p} & |w| > T \\ 0 & otherwise \end{cases}$$

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \cdot \nabla \mathcal{L}(\tilde{\boldsymbol{w}}_k)$$

Instead of employing soft or hard thresholding, Feather chooses the approach of balancing between the two as proposed by several works [30], [61]. To achieve this, the operator $\mathcal{P}_{(T)}$ uses $p = 3$. When $p = 1$ we have the soft version and the higher the value, the harder the thresholding operator becomes, as is evident in Figure 3.5.

### STE and Gradient Scaling

During each training iteration, Feather computes a binary pruning mask applied to the forward pass for each layer. As the sparsity target ratio increases to more demanding values (98% or 99%) the mask destabilizes, resulting in lower performance. Feather proposes gradient scaling as a way to stabilize these pruning masks in the latter training epochs. During the backward pass, the gradients of the pruned weights are scaled by a constant $\theta \in (0, 1)$. It is proposed that $\theta = 1$ for targets less than 95% and $\theta = 0.5$ otherwise, a selection which seems to work mostly well. The gradient scaling parameter is kept constant during training and is established at the beginning of training based solely on the final target. The same value is maintained for all layers of the training model throughout the training process, while the weights are now updated using the following equations:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \cdot \boldsymbol{m}_k \odot \nabla \mathcal{L}(\tilde{\boldsymbol{w}}_k)$$

Where $\boldsymbol{m}_k \in \{\theta, 1\}^N$ such that $m_{i,k} = 1$ if $w_{i,k} > T$ and $m_{i,k} = \theta$ if $w_{i,k} \leq T$, with $\odot$ denoting element-wise product.

# Chapter 4

# Gradient Scaling Function

Contents

## 4.1 Introduction

In Feather [27], gradient scaling is proposed as an alteration to the backward pass of the Straight-Through Estimator (STE) [7] to stabilize the pruning masks in the latter training epochs. During the backward pass, the gradients of the pruned weights are scaled by a constant $\theta \in (0, 1)$. It is proposed that $\theta = 1$ for targets less than 95% and $\theta = 0.5$ otherwise, a selection which seems to work mostly well. The gradient scaling parameter is kept constant during training and is established at the beginning of training based solely on the final target. The same value is maintained for all layers of the training model throughout the training process, while the weights are now updated using the following equations:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \cdot \boldsymbol{m}_k \odot \nabla \mathcal{L}(\tilde{\boldsymbol{w}}_k)$$

Where $\boldsymbol{m}_k \in \{\theta, 1\}^N$ such that $m_{i,k} = 1$ if $w_{i,k} > T$ and $m_{i,k} = \theta$ if $w_{i,k} \leq T$, with $\odot$ denoting element-wise product.

In several cases, as shown in Figure 4.1, the best scaling is neither 0.5 nor 1, but 0.75. This points towards an existing function that can calculate the optimal gradient scaling parameter and reach the best possible final accuracy. This chapter aims to identify this function, which will calculate the best possible $\theta$ throughout the training process.

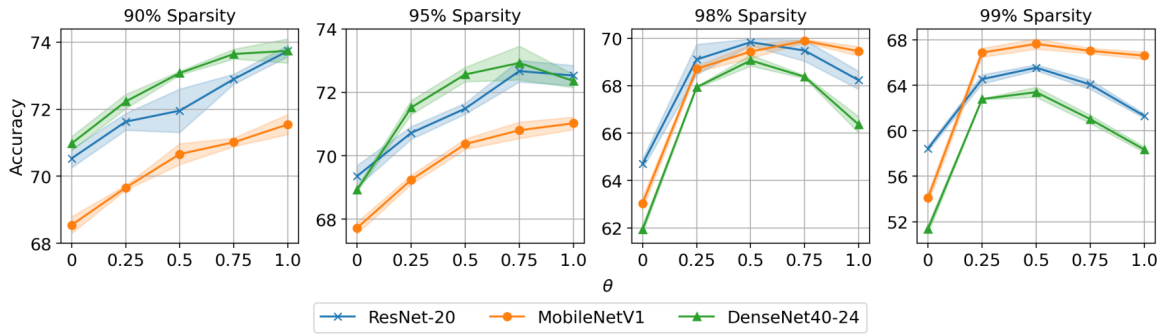



Figure 4.1: Effect of gradient scaling in Feather [27]

Instead of using a single static parameter, we propose a dynamic scaling function that adapts the gradient flow depending on the training stage. The proposed method changes the scaling value over time, allowing larger gradients to flow early in training and gradually scaling with a lower value the gradients from pruned weights as sparsity increases and the network stabilizes.

The aim of this dynamic scaling function is that it can both help stabilize the pruning mask during later epochs while also allowing a greater gradient flow during early training. This theoretically could improve final model accuracy, since the gradients will be updated as needed throughout training. Additionally, we explore both global and layer-wise applications of the dynamic scaling function to investigate whether different layers may benefit from varying gradient sensitivities. The function's adaptability provides a principled way to incorporate training dynamics into pruning-aware optimization, filling the gap left by Feather's heuristic-based scaling.

The effectiveness of the proposed method is evaluated through experiments on ResNet20 [35], MobileNet V1 [40] and DenseNet40-24 [41], all trained on the CIFAR-100 [48] dataset. The results indicate that our dynamic gradient scaling approach consistently outperforms the original Feather configuration especially at high sparsity levels. It also becomes evident that the final $\theta$ value is close to the ones shown in the ablation study of Figure 4.1.

## 4.2 Proposed Method

### 4.2.1 Function Selection

We propose a method for calculating the optimal $\theta$ during training based on the density of each layer at each training iteration. The module is modified to be able to set a different $\theta$ for each layer, which is calculated in each training iteration based on a suitable function. Specifically, we chose the function family of $f(x) = 1 + \alpha \cdot \ln(x)$, where $x$ is the layer density.

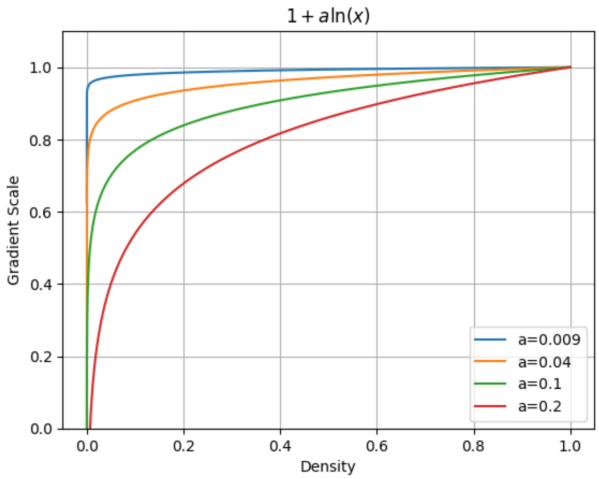



Figure 4.2: Function $1 + a \cdot \ln(density)$ for different $a$

The function form was selected based on observations during training that for low sparsities, the gradients do not need to be scaled, making $\theta = 1$ preferable. When large sparsities are reached, $\theta$ must drop significantly the larger the target sparsity, which was observed with the original Feather module. The issue was that this crude selection of the scaling parameter based on the target resulted in some models reaching lower accuracy than possible since some cases needed a scaling of 0.75 rather than 1 or 0.5.

The proposed function aims to calculate the best possible gradient scaling parameter during training to achieve the best possible accuracy in each case.

### 4.2.2 Selection of α

In preliminary experiments, the objective was to identify an ideal function that would yield the best possible result for any given sparsity target ratio. Nevertheless, it became evident that different rates gave optimal results for different target ratios. More specifically, the higher the target, the larger $\alpha$ the function required. This led to the conclusion that higher targets required a much steeper decline in $\theta$ earlier in the process to stabilize the sparsity mask efficiently, whereas lower targets performed much better when the gradient scaling parameter remained closer to 1, meaning that the drop should happen in sparsities well higher than the targeted one.

To determine the optimal $\alpha$ parameter, several experiments were run for 30 epochs instead of the standard of 160. In Figure 4.3, the experiments for ResNet-20 are shown for two targets of 95% and 99%. Accuracy alone, however, proved to be an inconclusive performance metric, especially for the 95% target, where no discernible pattern emerged to identify the optimal parameter range. For this reason, the results in Figure 3.5 were utilized, and as such the combination of accuracy and final $\theta$ yielded the ideal range for the optimal $\alpha$. By identifying the range where accuracy was high enough while the final average $\theta$ was closely aligned to the one provided in Feather's ablation studies, a few key $\alpha$ values were selected for further testing at 160 epochs.

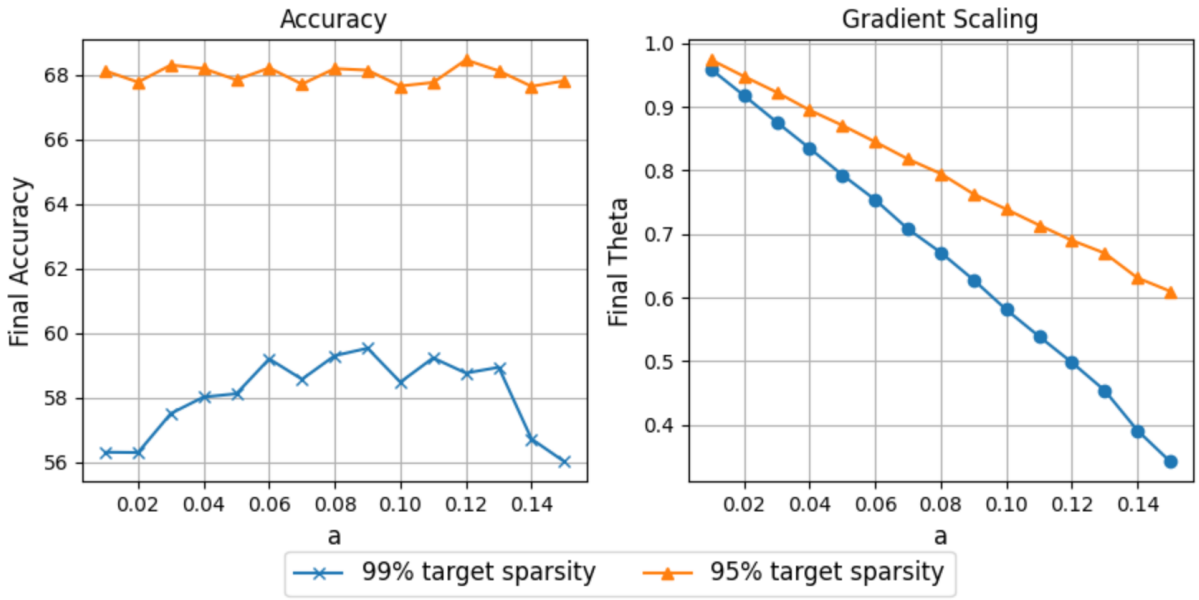



Figure 4.3: Effect of different functions at ResNet-20 for a 30 epoch training in final accuracy and $\theta$

As such, by finding the best possible $\alpha$ for each target sparsity analyzed in Feather (95%, 95%, 98%, and 99%), the results were mapped to create a function for calculating $\alpha$ based on the target sparsity $S$, which is set before the training even begins. It is interesting to note that for a target of 90% sparsity, there was no suitable parameter $\alpha$, suggesting that the initial $\theta = 1$ is the optimal "function". For that reason, it was set as $\alpha = 0$, which yields the constant $\theta = 1$ throughout training.

By the four defined sparsity target ratios, the following tangent function was calculated:

$$a(S) = 0.026 \cdot \tan(23.09 \cdot S + 22.08) + 0.093$$

To encourage generalization and consistency across all possible target sparsity ratios over 90%, including those not explicitly tested during experimentation, $\alpha$ is rounded to two decimal places. This minimizes the risk of overfitting to specific targets and ensures that negligible differences between targets do not lead to large variations in the function's behavior. This standardization maintains robustness and scalability for applications where the model may encounter sparsities not tested either on the original Feather module or in this study.

In Figure 4.4, the function for calculating the best $\alpha$ is shown, with the tested ratios highlighted. As expected, there are target sparsity ratio families, where for a group of different ratios the same parameter is selected. For the four targets tested the resulting gradient scaling function is also shown, providing visualization of the rate at which $\theta$ ought to drop.

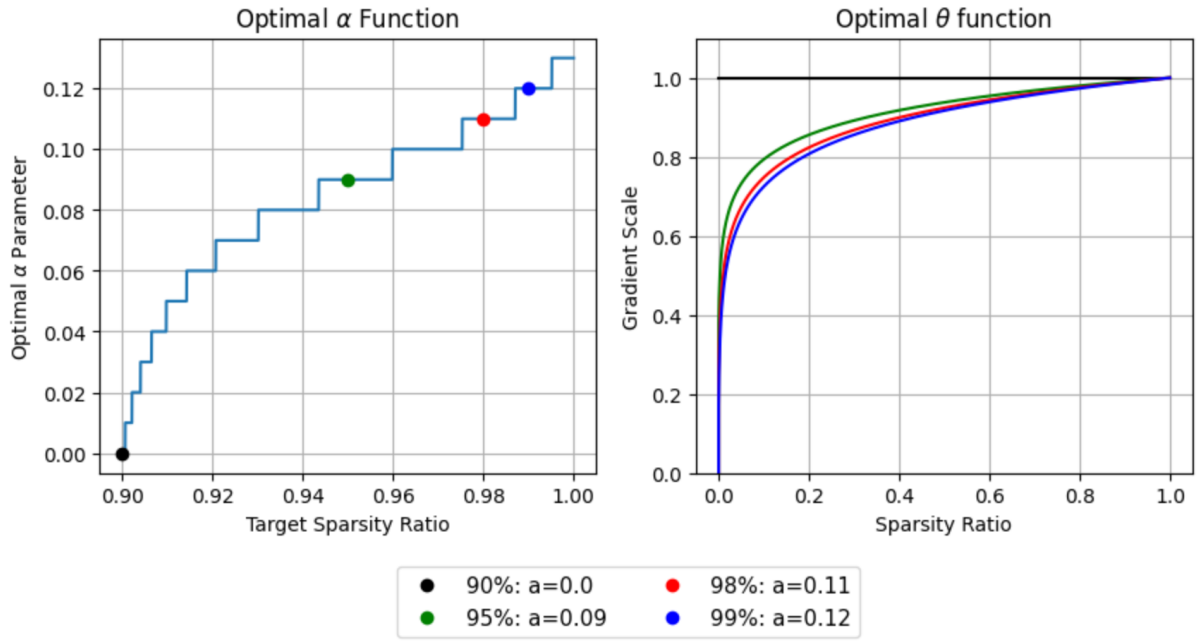



Figure 4.4: Opimal $\alpha$ function with the tested sparsity targets highlighted alongside the produced optimal gradient scaling function

### 4.2.3 Global and Layer-wise Approaches

The function was tested using two approaches, a layer-wise one and a global one. In the layer-wise approach, $\theta$ is calculated for each layer based on the calculated optimal function and updated independently for each layer. In the global approach, there is one further step in which the average is calculated across all layers, which is then applied at every layer.

## 4.3 Experimental Evaluation

For the evaluation of the modified module, we experiment on the three architectures tested on Feather, ResNet20 [35], MobileNet V1 [40] and DenseNet40-24 [41] on CIFAR-100 [48]. The training hyperparameters used in all experiments are presented in Table 4.1

| | |
|---|---|
| Epochs | 160 |
| Batch Size | 128 |
| Weight Decay | $5 \cdot 10^{-4}$ |
| Learning Rate | 0.1 |
| Optimizer | SGD |
| LR Scheduler | Cosine |
| Momentum | 0.9 |

Table 4.1: Training Hyperparameters

### 4.3.1 Global and Layer-wise Approaches

The layer-wise and global methods performed similarly well at lower sparsity levels, either exceeding or keeping up with the original Feather results. However, a disparity in accuracy was observed for high target sparsity ratios. The global approach showed superior performance. This difference is caused by the fact that when a layer reaches a sufficiently high sparsity, the gradient scaling parameter drops

significantly because of the nature of the selected function. With $\theta$ reaching values as low as 0.05, as shown in Figure 4.5, pruned weights decay significantly, offering great stability to the mask, but hindering training in the process. This results in the model having under-trained layers, which, even if they are few in number, cause a drop in accuracy. The effect of this is reduced global accuracy at the end of training, yielding results worse than originally achieved.

This is more prevalent in DenseNet40-24, which has the fewest trainable parameters by a wide margin. Under-trained layers can prove catastrophic, yielding much lower results than Feather. It is interesting to note that while MobileNet also excibits layers with a very low $\theta$ value just as the other models, the effect is minimized, giving good results within the margins of Feather and, in some cases, even better. That might be due to the fact that those layers have one hundred times more parameters than the equivalent layers in other models, resulting in a much larger amount of trainable parameters.

However, this issue with the layer-wise approach could prove to be a limitation of the proposed function rather than a general problem, and a more carefully designed one could bypass this collapse issue, something that remains an open research question for future work.

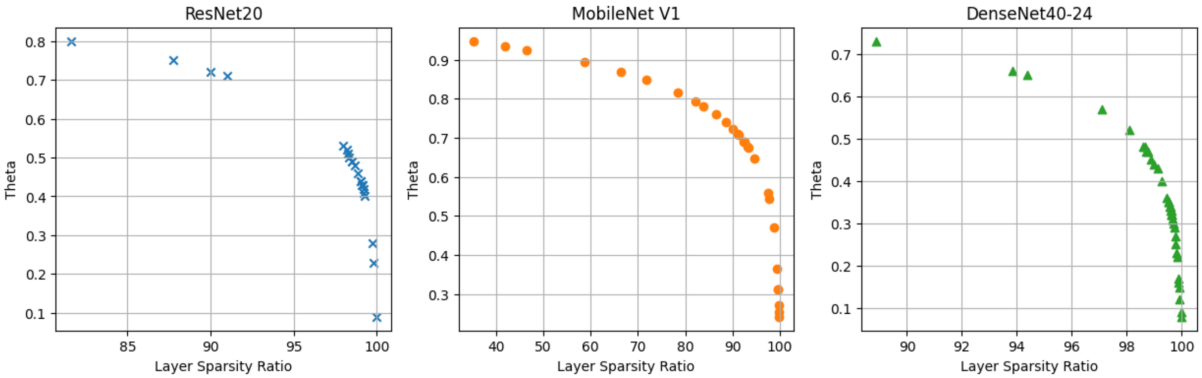



Figure 4.5: Final achieved $\theta$ and sparsity for each layer in each model for a target of 99% sparsity with the layer-wise updating approach

The global update of the parameter minimizes this problem, as the average value of $\theta$ takes into account the density variations between all layers, ensuring better performance and generalization. With this method, no layer has a $\theta$ with an extremely low value, resulting in the layers being pruned more efficiently achieving acceptable ratios without hindering parameter training. In Figure 4.6, the difference between final $\theta$ with each approach is highlighted. For the layer-wise approach, the average of all layers was calculated and was shown to generally reach a lower value, pointing toward the fact that some layers, more so in smaller models, get over-pruned when the gradient scaling parameter is too low.

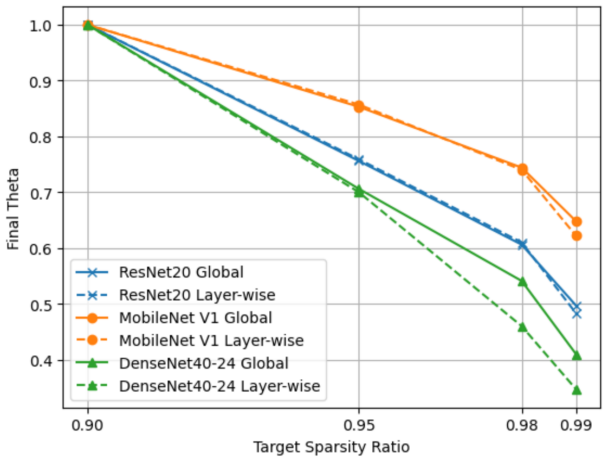



Figure 4.6: Average $\theta$ for both approaches for all tested models and target sparsity ratios

Figure 4.7 visualizes the importance of a different optimal global $\theta$ for different models while targeting the same final sparsity ratio of 99%. In MobileNet, many layers do not reach very high sparsity levels, resulting in a better performance with a higher average $\theta$. On the other hand, DenseNet achieves the desired sparsity level more consistently across its layers, with even the most dense layers having little deviation from the target, therefore requiring a lower $\theta$ to ensure effective training. ResNet also achieves the desired sparsity in most layers, having a few with a more significant difference than DenseNet, but still at acceptable levels. This results in an average $\theta$ between the two other models, but closer to the lower end. This behavior aligns with what the ablation studies concerning gradient scaling presented in Figure 3.5, where MobileNet demonstrated its top performance when using a higher $\theta$ at sparsity targets of 95% and 98% compared to other models. Figure 4.8 shows the different gradient scaling parameters achieved throughout training with a global $\theta$ updating approach for all three models for all three target sparsities. It is evident that after reaching the target sparsity ratio at the 80th epoch, $\theta$ converges to its final value.

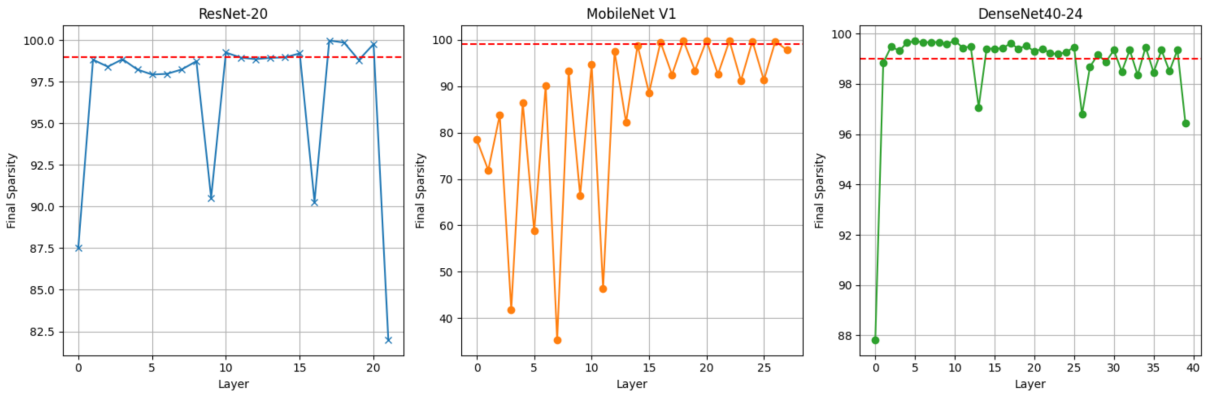
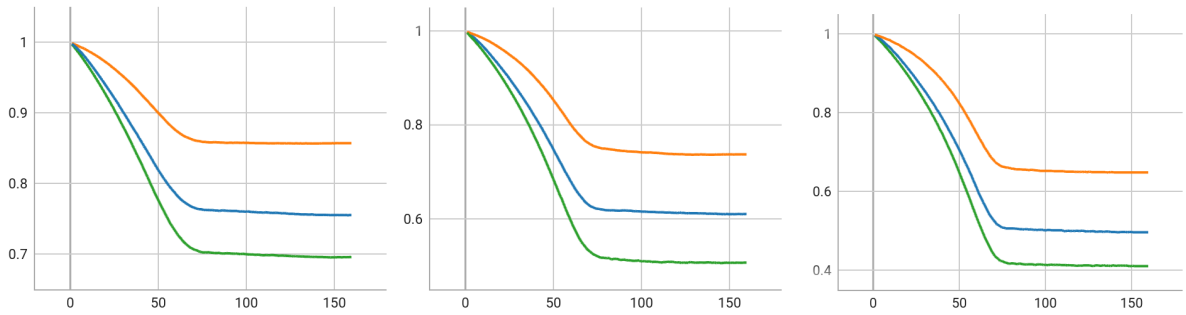



Figure 4.7: Achieved sparsities for a target of 99% per layer for each tested model with the global updating approach




(a) 95% (b) 98% (c) 99%

Figure 4.8: Gradient scaling evolution per epoch for each tested model with the global updating approach

### 4.3.2 Comparison with Feather

The results suggest that gradient scaling is insignificant for low sparsities, causing a significant change only after reaching a certain point. The optimal $\theta$ is not needed from the beginning of training and pruning, but can rather be computed while training and pruning the model. The optimal value is reached when the model reaches the target sparsity ratio.

The results for the global updating method also show a small but consistent improvement compared to the original performance, even if the final $\theta$ is equal to the module's set. This suggests that calculating the gradient scaling parameter during training results in having a sufficiently high $\theta$ value during early epochs with lower sparsities, allowing the model to learn more efficiently before being pruned to extreme ratios.

It is interesting to note that the larger the model, the smaller the performance gap between the two approaches. Notably, for MobileNet V1 the layer-wise approach surpasses the global one for targets of 95% and 98%, while being very close for the 99% sparsity target ratio. In contrast, for the smallest of the three models, DenseNet40-24, the gap in all cases is significant. This implies that for even larger models, a layer-wise approach will be preferable.

Table 4.2 provides the results obtained during our experiments in CIFAR-100 compared to ST-3 [93], Spartan [89] Feather. The experiments were run with the exact same training parameters for 160 epochs. The results, as with Feather, are the averages of three runs with the corresponding standard deviations. It is important to note that calculating the $\theta$ value at each iteration did not impact training time.

| Ratio | 90% | 95% | 98% | 99% |
|---|---|---|---|---|
| ResNet-20 (1.096M Params): 73.59 $\pm$0.44 | | | | |
| ST-3 | 72.81$_{\pm 0.13}$ | 71.72$_{\pm 0.20}$ | 67.53$_{\pm 0.53}$ | 58.32$_{\pm 0.17}$ |
| Spartan | 72.56$_{\pm 0.35}$ | 71.60$_{\pm 0.40}$ | 67.27$_{\pm 0.14}$ | 61.70$_{\pm 0.21}$ |
| Feather-Global | **73.74** $\pm$0.17 | 72.53 $\pm$0.32 | 69.83 $\pm$0.14 | 65.55 $\pm$0.25 |
| Feather with best $\theta$ | **73.74** $\pm$0.17 | 72.67 $\pm$0.34 | 69.83 $\pm$0.14 | 65.55 $\pm$0.25 |
| Global Theta | **73.74** $\pm$0.17 | **72.77** $\pm$0.30 | **70.04** $\pm$0.13 | **65.75** $\pm$0.28 |
| Theta per Layer | **73.74** $\pm$0.17 | 72.66 $\pm$0.11 | 69.54 $\pm$0.20 | 64.94 $\pm$0.34 |
| MobileNetV1 (3.315M Params): 71.15 $\pm$0.17 | | | | |
| ST-3 | 70.94$_{\pm 0.25}$ | 70.44$_{\pm 0.23}$ | 69.40$_{\pm 0.06}$ | 66.63$_{\pm 0.15}$ |
| Spartan | 70.52$_{\pm 0.51}$ | 69.01$_{\pm 0.11}$ | 65.52$_{\pm 0.24}$ | 60.65$_{\pm 0.22}$ |
| Feather-Global | **71.55** $\pm$0.30 | 71.03 $\pm$0.20 | 69.44 $\pm$0.29 | 67.64 $\pm$0.45 |
| Feather with best $\theta$ | **71.55** $\pm$0.30 | 71.03 $\pm$0.20 | 69.89 $\pm$0.07 | 67.64 $\pm$0.45 |
| Global Theta | **71.55** $\pm$0.30 | 71.22 $\pm$0.20 | 70.04 $\pm$0.24 | **67.73** $\pm$0.18 |
| Theta per Layer | **71.55** $\pm$0.30 | **71.33** $\pm$0.12 | **70.29** $\pm$0.58 | 67.64 $\pm$0.31 |
| DenseNet40-24 (0.714M Params): 74.70 $\pm$0.51 | | | | |
| ST-3 | 72.56$_{\pm 0.31}$ | 71.21$_{\pm 0.35}$ | 65.48$_{\pm 0.18}$ | 56.18$_{\pm 0.60}$ |
| Spartan | 73.13$_{\pm 0.25}$ | 71.61$_{\pm 0.04}$ | 65.94$_{\pm 0.07}$ | 58.64$_{\pm 0.18}$ |
| Feather-Global | **73.75** $\pm$0.36 | 72.36 $\pm$0.21 | 69.06 $\pm$0.23 | 63.40 $\pm$0.44 |
| Feather with best $\theta$ | **73.75** $\pm$0.36 | **72.93** $\pm$**0.53** | 69.06 $\pm$0.23 | 63.40 $\pm$0.44 |
| Global Theta | **73.75** $\pm$0.36 | 72.92 $\pm$0.04 | **69.13** $\pm$0.10 | **63.54** $\pm$0.32 |
| Theta per Layer | **73.75** $\pm$0.36 | 72.53 $\pm$0.09 | 68.72 $\pm$0.31 | 61.93 $\pm$0.35 |

Table 4.2: Top-1 accuracy in CIFAR-100

For further clarity on the optimal values $\theta$ and the evaluation of the results summarized in Table 4.2, a visualization is provided. In Figure 4.9 the experiments with the proposed function updating $\theta$ globally, highlighted in red, are presented in comparison to Feathers' previous ablation studies in regards to the gradient scaling effect, while also giving the optimal $\theta$ value. As intended, the optimal $\theta$ falls close to the value identified in the previous study consistently.

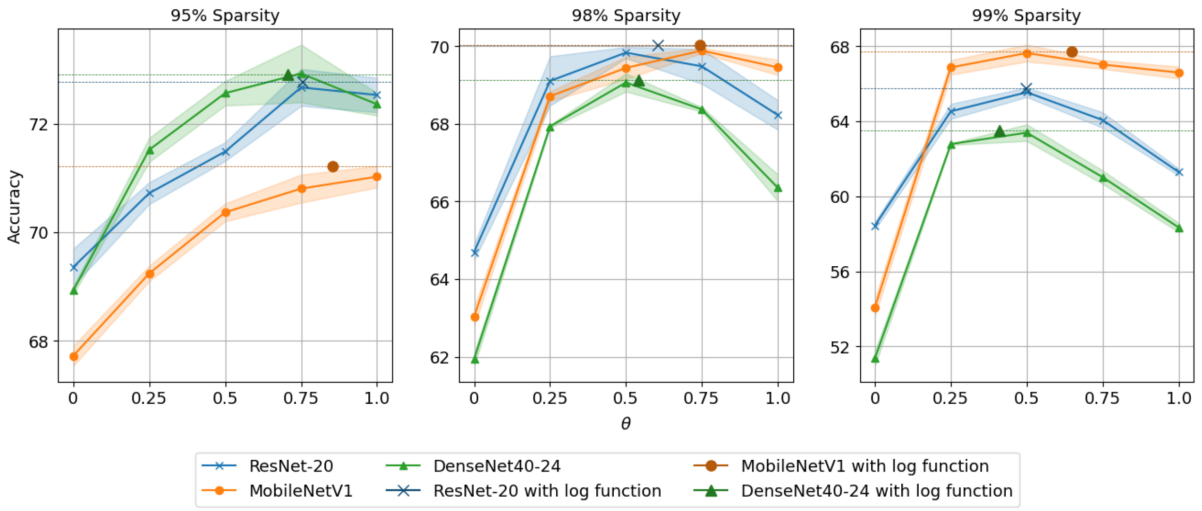



Figure 4.9: Final results of the global approach over Feather

### 4.3.3 Experimental Evaluation for Untested Sparsities

Since optimal functions were derived from four sparsity target ratios but built to work universally, it was necessary to test whether they worked for different targets. The tests were run only for the global method, since it had the best results overall and it was needed to see whether the function operates as expected throughout all targets and models. The results shown in Tables 4.3 and 4.4 validated this assumption, as it is shown that for every possible combination, the globally updated $\theta$ achieves the highest results. When reaching very extreme sparsity ratios, the difference becomes even more evident, with accuracy exceeding the original one by even almost 4%.

In Figure 4.10, we provide all the final global $\theta$ values calculated for each combination of model and target sparsity, showcasing the optimal gradient scaling for each.

| Ratio | 92% | 94% | 97% |
|---|---|---|---|
| ResNet-20 (1.096M Params): 73.59 | | | |
| Feather | 73.07 ±0.40 | 72.73 ±0.05 | 71.18 ±0.43 |
| Global Theta | **73.25** ±0.08 | **72.91** ±0.12 | **71.29** ±0.06 |
| MobileNetV1 (3.315M Params): 71.15 | | | |
| Feather | 71.45 ±0.09 | 71.01 ±0.12 | 70.14 ±0.46 |
| Global Theta | **71.51** ±0.24 | **71.26** ±0.08 | **70.85** ±0.29 |
| DenseNet40-24 (0.714M Params): 74.7 | | | |
| Feather | 73.28 ±0.17 | 72.84 ±0.16 | 70.85 ±0.10 |
| Global Theta | **73.38** ±0.37 | **73.07** ±0.19 | **70.89** ±0.30 |

Table 4.3: Comparison for previously untested target sparsity ratios

| Ratio | 99.2% | 99.5% | 99.8% |
|---|---|---|---|
| ResNet-20 (1.096M Params): 73.59 | | | |
| Feather | 63.13±0.34 | 57.52±0.34 | 39.80±0.50 |
| Global Theta | **63.58**±0.32 | **58.49**±0.14 | **43.24**±0.57 |
| MobileNetV1 (3.315M Params): 71.15 | | | |
| Feather | **66.62**±0.22 | 62.96±0.42 | 50.71±0.09 |
| Global Theta | 66.41±0.23 | **63.04**±0.28 | **52.47**±0.29 |
| DenseNet40-24 (0.714M Params): 74.7 | | | |
| Feather | 60.53±0.31 | 53.52±0.24 | 34.86±0.54 |
| Global Theta | **61.01**±0.38 | **55.20**±0.16 | **38.34**±0.14 |

Table 4.4: Comparison for very large target sparsity ratios

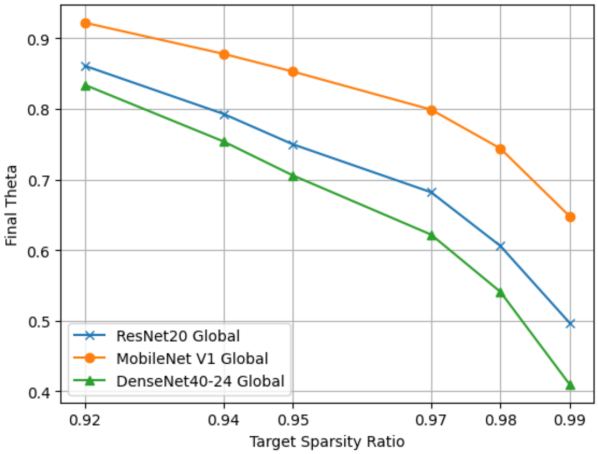



Figure 4.10: Final globally updated $\theta$ for all models and target sparsity ratios

## 4.4 Conclusions

In this chapter, we introduced a dynamic gradient scaling function designed to improve upon the fixed scaling parameter used in Feather. By employing a logarithmic function dependent on layer density and adapting the function's slope on the target sparsity, the scaling parameter $\theta$ can now be computed continuously during training. This approach addresses the limitations of Feather's static scaling, providing greater flexibility and adaptability to different sparsity targets.

Two approaches where evaluated, a layer-wise one and a global one. While they both perform well at lower sparsity levels, the global approach proves to be better and more robust at higher sparsity targets. By averaging $\theta$ across layers, the global method prevents the over-pruning and under-training observed in the layer-wise method, especially in smaller models like DenseNet40-24.

Compared to the original Feather module, the proposed method demonstrated small but consistent improvements in performance. These improvements are based on the fact that a higher $\theta$ value early in training allows higher gradient flow with the STE operator, enabling more effective learning before entering the aggressive pruning phases.

# Chapter 5

# Adaptive Pruning Scheduler

Contents

## 5.1 Introduction

As DNN pruning reaches extreme sparsity levels, such as 98% or 99%, performance presents significant drops in accuracy and stability (Figure 5.1). This could be dependent on the variability of the pruning masks $M$, which are binary matrices that determine which weights shall be kept (1) and which pruned (0).

In many pruning frameworks, including Feather [27], the sparsity level increases according to a predefined schedule, such as the cubic one proposed in [105], regardless of the network's training state or the stability of its sparsity mask. While this scheduler is more than effective for moderate pruning levels, when concerning extreme sparsity ratios they seem to reach the target too fast too soon. As a result, pruning masks may differ significantly between consecutive training iterations, suggesting that the network has not settled into a stable sparse pattern.

This chapter introduces an adaptive pruning scheduler function family, by which each possible configuration dynamically adjusts the pruning rate based on the observed stability of the pruning masks. This means that at each training iteration the scheduler measures how much the pruning masks change. To quantify mask stability we use the Jaccard Similarity Index, which measures the overlap between consecutive pruning masks. When the stability of the mask appears rises, this means that an acceptable stability is reached and the scheduler allows pruning to proceed more confidently. When drops in stability are detected, pruning is paused to give the model more time to stabilize.

Through experiments on architectures such as ResNet20 [35], DenseNet40-24 [41], and MobileNet V1 [40] trained on CIFAR-100 [48] we evaluate the proposed scheduler's effect on accuracy, mask stability, and overall pruning efficiency. The results demonstrate that incorporating a feedback-driven adjustment mechanism can help maintain model quality even under extremely high sparsity ratios, as long as the schedulers hyperparameters are chosen correctly.

## 5.2 Mask Stability and Performance

In this section we will quantify this destabilization from training iteration to training iteration, providing a more definite connection between drops in accuracy and mask stability. First, we present the metric used to calculate the similarity between subsequent masks. Then we define mask stability while presenting the correlation between this stability and the performance.

### 5.2.1 Jaccard Similarity Index and Mask Stability Definition

The Jaccard (or Tanimoto) similarity index $J(A, B)$ is a statistical measure of similarity between two sets $A$ and $B$. Mathematically it is defined as the size of the intersection of the sets divided by the size of their union.

The Jaccard index ranges from 0 to 1, where 0 indicates no similarity and 1 signifies that the sets are identical. Unlike other similarity metrics, Jaccard similarity does not account for element frequency, making it ideal for binary or categorical data, which is why it is widely used in tasks like text analysis and image recognition.

The sparsity mask of a layer is a matrix containing only binary elements (0 or 1), pointing towards which weights shall be retained (1) and which shall be pruned (0) at each training iteration. As such, we can define the mask stability $\mu_{l,k}$ of a layer $l$ in an iteration $k$ of two subsequent masks $M_{l,k-1}$ and $M_{l,k}$ as their Jaccard index:

$$\mu_{l,k} = J(M_{l,k-1}, M_{l,k}) = \frac{|M_{l,k-1} \cap M_{l,k}|}{|M_{l,k-1} \cup M_{l,k}|}$$

### 5.2.2 Performance Evaluation

Feather employs a cubic pruning scheduler [105], which prunes the model aggressively at the beginning where the targets are low enough, and smooth when reaching the final target.

We train ResNet20 [35], MobileNet V1 [40], and DenseNet40-24 [41] for 160 epochs along with the Feather sparsity module with a target sparsity ratio of 99%, which is the most demanding one and the one presenting the biggest drops in performance. Along with training, we calculate the mask stability $\mu_{l,k}$ for each layer $l$ at each iteration $k$ and then receive the average $\mu_k$. Figure 5.1 presents the accuracy, the average mask stability $\mu_k$ as well as the sparsity ratio at the end of each epoch for all three models.

It is evident that when very large sparsities are achieved, the mask stability drops significantly, which means that the pruning masks get too destabilized. This results in a significant drop in accuracy (in the cases of ResNet20 and DenseNet40-24), creating a local minimum close to the epoch where 99% sparsity is reached.

MobileNet V1 in general retains a high enough stability, which results in a small drop and by far the best performance, with an average accuracy of 67.64%. This is attributed to the fact that MobileNet, having by far the most parameters, retains a large amount of weights even after pruning excessively, which can then be trained. DenseNet40-24 has the least amount of parameters, which means that after pruning there are too little of them, resulting in a large drop in stability and an average accuracy of 63.40%, with ResNet20 having similar issus and achieving 65.55%.

This correlation between mask stability and performance points to the possibility of designing an adaptive pruning scheduler, which provides the model with the target sparsity ratio to achieve based on the average mask stability.

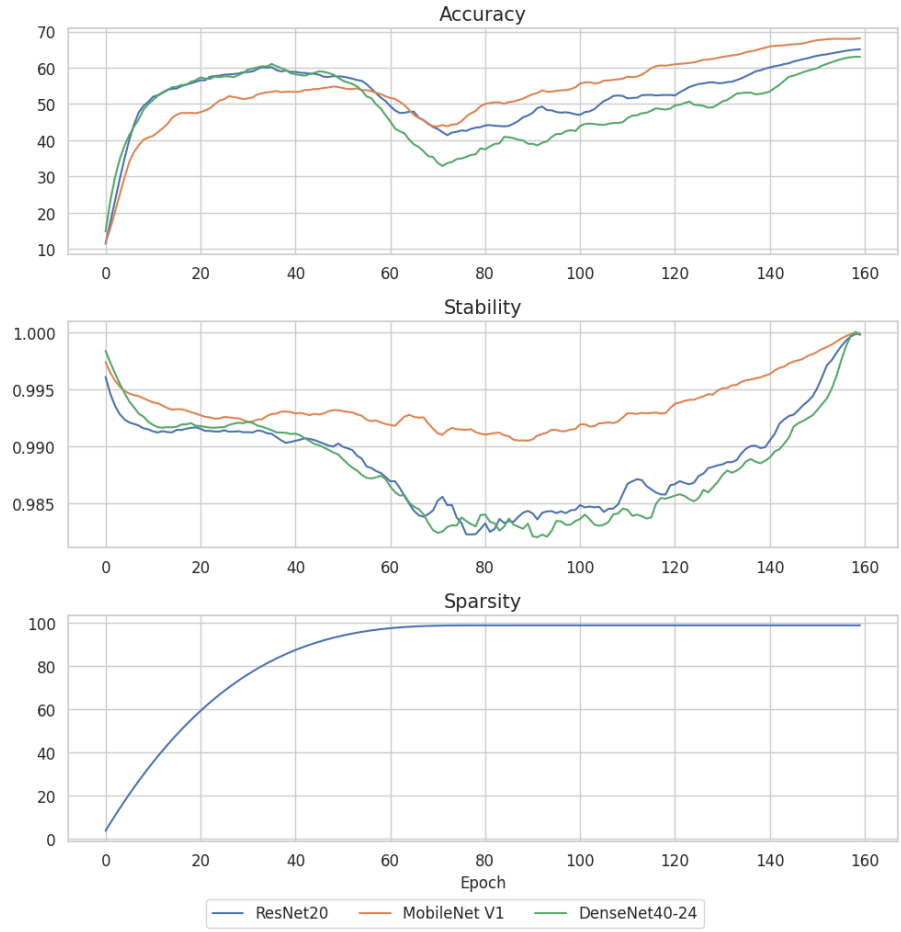



Figure 5.1: Accuracy, average mask stability, and model sparsity achieved per training epoch with a target sparsity ratio of 99% for ResNet20 (blue), MobileNet V1 (orange) and DenseNet40-24 (green)

## 5.3 Proposed Method

Since it was made evident that sparsity, performance, and mask stability are closely related, in this section, a simple, experimentally derived adaptive pruning scheduler is proposed. The scheduler is based on the average mask stability difference between two consequent training iterations.

### 5.3.1 Mask Stability Difference

While the average mask stability value $\mu_k$ of a single iteration $k$ can offer important information about the pruning and accuracy state, it requires previously set limits. A predefined limit must be set, by which if the stability falls under it indicates drops in performance. This is why for this work we chose to employ the difference between two consequent iterations, something that shows whether overall stability drops, rises or remains steady. Usually large drops indicate that a high enough sparsity is reached, which translate to big drops in overall performance. On the other hand, a large increase translates to the model adapting to its current sparsity, and indicates that it is able to reach even higher values.

This difference between two iterations, denoted as $\mu_{k+1} - \mu_k$, can be used with proper scaling to calculate the $k+1$ iteration's desired sparsity. The proposed function is

$$S_{k+1} = S_k + \lambda \cdot (\mu_{k+1} - \mu_k)$$

where $S_{k+1}$ is the calculated sparsity desired in this iteration, $S_k$ is the sparsity previously achieved and $\lambda$ a scaling factor which will be discussed further in the next section.

This function ensures that when stability rises, the target sparsity rises accordingly, as the model is now able to handle such change. When stability drops or remains steady, the model has not yet adapted to the current sparsity level, which is why the scheduler keeps the pruning ratio to a steady state.

Figure 5.2 visualizes the mask stability differences for 1000 iterations based on the training task presented in 5.1.

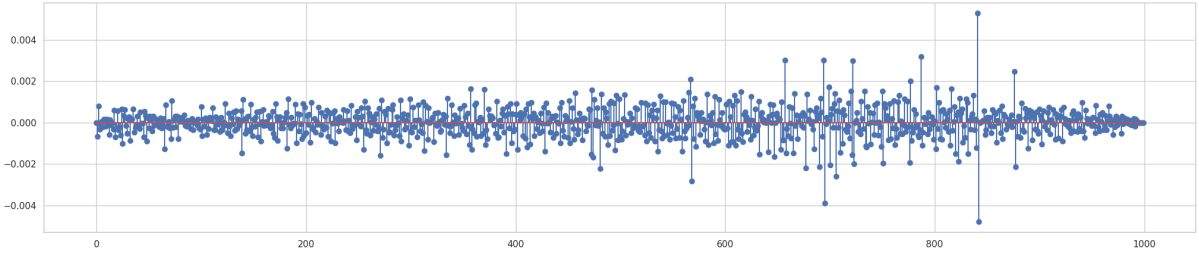



Figure 5.2: Difference between consecutive average mask stabilities per iteration

### 5.3.2 Scaling Factor

After experiments it became evident that when $\mu_{k+1} \leq \mu_k$, the best approach is to completely freeze the pruning process, giving the model time to fully adapt to the current sparsity level. This means that when the average mask stability drops or remains the same, the scaling factor will be set to 0.

In the case of mask stability rising we add this difference to the previous ratio. There are two factors that must be taken into account when choosing the proper scaling function. First, it must scale the difference accordingly to the sparsity's order of magnitude to ensure that the sparsity rises smoothly during training, neither too fast nor too slow. The second factor is that for very high and therefore challenging sparsities the scheduler must be able to slow down, diminishing the stability difference's effect considerably. This role falls on the scaling factor because as seen in Figure 5.2 the differences in mask stability are relatively close in absolute values throughout training. Keeping the same magnitude for large sparsities can cause even larger drops in performance, which is why it is crucial to scale the difference accordingly.

Modeling the properties described above with a function leads to the selection of a sigmoid function. The sigmoid function family has the ability to keep a steady scaling value for a wide range of lower sparsities, and then smoothly drop the value the larger they become, rendering it ideal for this task. The general form of the proposed function is as follows:

$$\lambda(S_k) = \frac{A}{1 + e^{-C(B-S_k)}} + D$$

$A$ is the constant that determines the scaling of the stability difference for low sparsities, $B$ determines at which sparsity the difference will be at the midpoint of the drop, $C$ is the slope and $D$ a bias term to make sure the $\lambda$ value does not fall below 0 as well as adjusting the curve to the behaviors of each model's stability. This is necessary since each model presents different drops and rises in mask stability, as shown in Figure 5.1.

### 5.3.3 Total Iterations Scaling

The experiments to determine the proper scaling function and its constants were conducted for a total training of 160 epochs, something which means that the scaling factor is proportional to the iterations of this training time. The order of magnitude of the sparsity added during each iteration is determined directly from the fact that it must happen at most in 160 epochs. Nevertheless, the scheduler should adapt to all possible amounts of training epochs, which is why we introduce one more scaling value $l$:

$$l = \frac{160}{|\text{epochs}|}$$

This constant manages to compress or expand the adaptive scheduler's form in such way that it performs as expected in all cases, adding a proportional value of sparsity during each training iteration.

### 5.3.4 Proposed Scheduler Constants

The final experimentally derived scheduler is described as follows:

$$S_{k+1} = \begin{cases} S_k + l \cdot \lambda(S_k) \cdot (\mu_{k+1} - \mu_k) & \mu_{k+1} > \mu_k \\ S_k & \mu_{k+1} \leq \mu_k \end{cases}$$

For the experiments conducted to evaluate the performance of the scheduler we choose two vastly different $a(S_k)$ functions, to showcase the different behaviors that can be achieved and the different effects they present in both accuracy and stability. The two selected functions are:

$$\lambda_1(S_k) = \frac{0.5}{1 + e^{-20(0.80-S_k)}} + D$$

$$\lambda_2(S_k) = \frac{0.3}{1 + e^{-75(0.91-S_k)}} + D$$

As mentioned in a previous section, the bias term $D$ differs between different models, something that can be attributed to the different scaling of the average mask stability. For example, as shown in Figure 5.1, MobileNet V1 is a much more stable model, presenting much smaller drops and rises compared to the other two models. With the same bias term it would either reach the target sparsity too late in training or never at all. The bias term aims to adjust the scheduler to the model's overall stability behavior. For $\lambda_1(S_k)$ the bias term is $D = -0.01$ for ResNet20 and DenseNet40-24 and $D = -0.007$ for MobileNet V1. The corresponding values for $\lambda_2(S_k)$ are $D = 0$ and $D = 0.003$. For future discussion, it would be interesting to explore how to derive the $D$ constant from the properties of the training model rather than selecting a specific value for each model.

Figure 5.3 graphs the functions with the bias terms set to those of ResNet20 and DenseNet40-24.

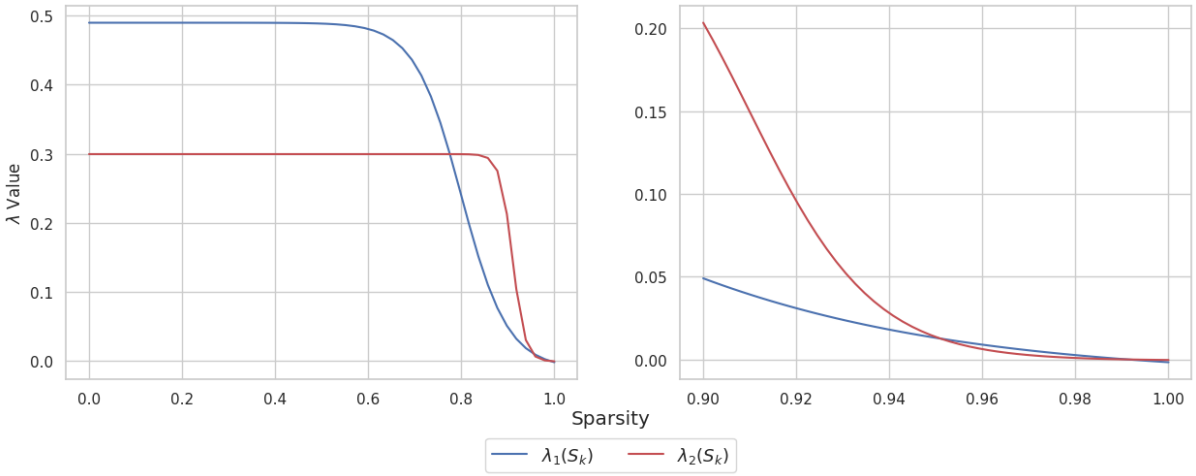



Figure 5.3: $\lambda$ values for all sparsity levels (a) and for very high ones (b). The second graph is provided for clarity, as the first one does not show the differences for high sparsities

## 5.4 Evaluation

Figure 5.4 shows the performance of a ResNet20 model with a target sparsity of 99% both with the cubic scheduler (black) and our proposed adaptive one (blue). From both the accuracy and stability graphs it is evident that with the adaptive scheduler while the drops in performance are still prominent they are less steep than before, giving a smoother appearance. This helps the model to become more stable and therefore achieve a slightly higher accuracy, as shown both in the graph and in Table 5.1. However, the overall stability seems to follow a similar pattern as the original one, showing no improvement.

It is also interesting to note that the model is able to reach certain sparsity levels earlier than previously thought. For example, a sparsity of 90% is now reached at epoch 34 rather than 45. This causes a minimal decrease in performance at that time during training and a small increase in stability, which contribute to the overall smoothing of performance. The target of 99% sparsity is reached now at the 86th epoch rather than the 80th, which in combination with the earlier reaching of 90% results in the model having more available iterations to train with very high sparsity ratios ($\geq 90\%$).

This method of adaptive scheduling appears to be more beneficial for very extreme sparsity ratios, such as 98% and 99% where the drop in performance during training was already quite significant. For lower targets, performance is usually close to the original, with the exception of ResNet20 that reveals a drop in 90% sparsity compared to cubic scheduling.

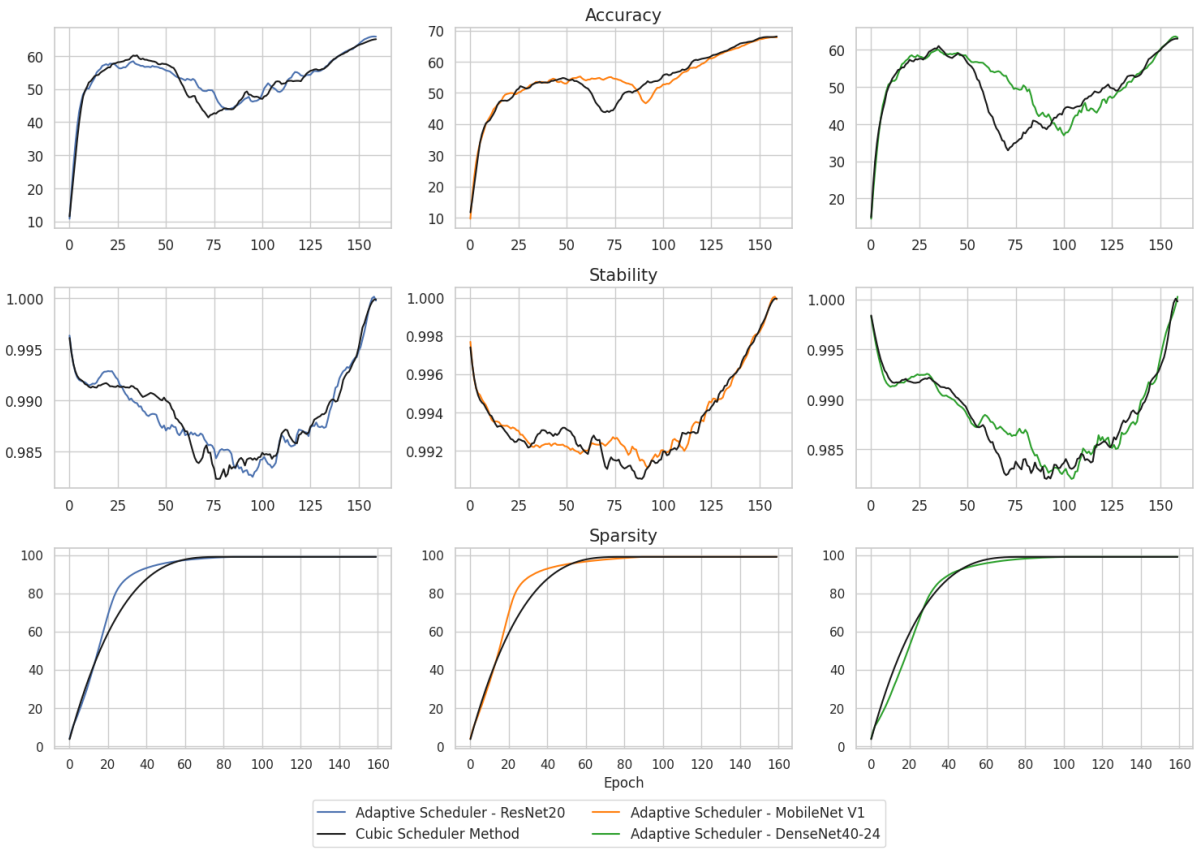



Figure 5.4: Performance of all three models with a target sparsity of 99% with a cubic scheduler (black) and our proposed adaptive one (blue, orange and green) using $\lambda_1(S_k)$ for scaling

Figure 5.5 shows the performance of the same model with the second adaptive scheduler, as a response to the fact that the stability with the first one shows little improvement. In this case, both the accuracy and the stability keep a remarkably high value throughout training, only slightly dropping when the target of 99% is reached at the 117th epoch. This drop however, despite being considerably smaller than usual, happens too late into the training process which results in the performance never recovering, rendering a final accuracy lower than the one achieved with a cubic scheduler.

Since this scheduler reaches the target further into the training process than before, it seems to perform better for lower targets, such as 90% and 95%. In the case of 98% sparsity it also appears to barely surpass the performance of the cubic scheduler in all three models. However, in 99% there is a dramatic drop in performance in ResNet-24 and DenseNet40-24, which happens due to the very late sparsity target achievement. MobileNet V1 once again shows increased stability, which is why it is the only model to surpass Feather with both schedulers.

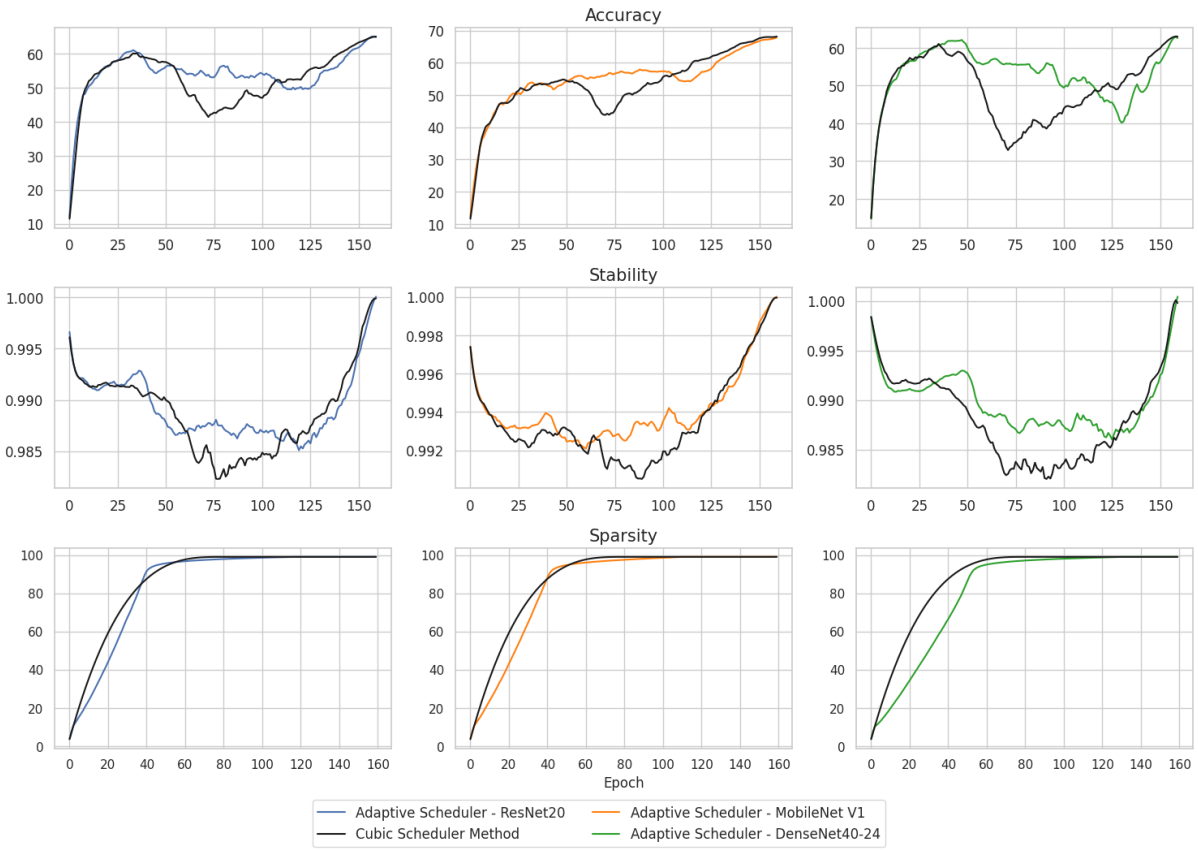



Figure 5.5: Performance of all three models with a target sparsity of 99% with a cubic scheduler (black) and our proposed adaptive one (blue, orange and green) using $\lambda_2(S_k)$ for scaling

Table 5.1 presents all the available results. The first adaptive scheduler, using the scaling function $\lambda_1(S_k)$, has improved final accuracy over the cubic one. The scheduler increases sparsity only when a positive trend in mask stability is observed, greatly rewarding the model for stabilizing its pruning mask. As a result, the target sparsity is achieved earlier in the training schedule, giving the model more time to fine-tune its weights under the final sparse configuration, thereby leading to better overall performance.

In contrast, the second adaptive scheduler with the $\lambda_2(S_k)$ function, adopts a more conservative approach. It allows smaller sparsity steps, even when the model demonstrates stable pruning behavior. This strategy ensures a higher average mask stability throughout training, but it often fails to reach the target sparsity early enough. For higher sparsity targets the model lacks the time to fine-tune, which results in slightly inferior accuracy. This highlights the fundamental trade-off between accuracy and stability in extreme sparsities.

The findings confirm the necessity of adaptivity into pruning schedulers. By making pruning rate dependent on real-time mask stability, the adaptive scheduler aligns more closely with the learning dynamics of the model. The first adaptive scheduler is shown to be more effective when high accuracy and fast convergence to sparsity are needed, while the second is better suited for applications where mask consistency is crucial.

| Ratio | 90% | 95% | 98% | 99% |
|---|---|---|---|---|
| ResNet-20 (1.096M Params): 73.59 $_{\pm 0.44}$ | | | | |
| ST-3 | 72.81±0.13 | 71.72$_{\pm 0.20}$ | 67.53$_{\pm 0.53}$ | 58.32$_{\pm 0.17}$ |
| Spartan | 72.56$_{\pm 0.35}$ | 71.60$_{\pm 0.40}$ | 67.27$_{\pm 0.14}$ | 61.70$_{\pm 0.21}$ |
| Feather | **73.74**$_{\pm 0.17}$ | 72.53$_{\pm 0.32}$ | 69.83$_{\pm 0.14}$ | 65.55$_{\pm 0.25}$ |
| Feather with different schedulers | | | | |
| Adaptive Scheduler 1 | 73.48$_{\pm 0.32}$ | 72.56$_{\pm 0.12}$ | **70.03**$_{\pm 0.09}$ | **65.72**$_{\pm 0.17}$ |
| Adaptive Scheduler 2 | 73.70$_{\pm 0.26}$ | **72.69**$_{\pm 0.17}$ | 69.92$_{\pm 0.38}$ | 64.93$_{\pm 0.21}$ |
| MobileNetV1 (3.315M Params): 71.15 $_{\pm 0.17}$ | | | | |
| ST-3 | 70.94$_{\pm 0.25}$ | 70.44$_{\pm 0.23}$ | 69.40$_{\pm 0.06}$ | 66.63$_{\pm 0.15}$ |
| Spartan | 70.52$_{\pm 0.51}$ | 69.01$_{\pm 0.11}$ | 65.52$_{\pm 0.24}$ | 60.65$_{\pm 0.22}$ |
| Feather | 71.55$_{\pm 0.30}$ | 71.03$_{\pm 0.20}$ | 69.44$_{\pm 0.29}$ | 67.64$_{\pm 0.45}$ |
| Feather with different schedulers | | | | |
| Adaptive Scheduler 1 | **71.56**$_{\pm 0.20}$ | **71.23**$_{\pm 0.29}$ | 69.75$_{\pm 0.03}$ | **68.04**$_{\pm 0.19}$ |
| Adaptive Scheduler 2 | 71.20$_{\pm 0.07}$ | 71.06$_{\pm 0.19}$ | **70.02**$_{\pm 0.30}$ | 67.77$_{\pm 0.13}$ |
| DenseNet40-24 (0.714M Params): 74.70 $_{\pm 0.51}$ | | | | |
| ST-3 | 72.56$_{\pm 0.31}$ | 71.21$_{\pm 0.35}$ | 65.48$_{\pm 0.18}$ | 56.18$_{\pm 0.60}$ |
| Spartan | 73.13$_{\pm 0.25}$ | 71.61$_{\pm 0.04}$ | 65.94$_{\pm 0.07}$ | 58.64$_{\pm 0.18}$ |
| Feather | 73.75$_{\pm 0.36}$ | 72.36$_{\pm 0.21}$ | 69.06$_{\pm 0.23}$ | 63.40$_{\pm 0.44}$ |
| Feather with different schedulers | | | | |
| Adaptive Scheduler 1 | 73.32$_{\pm 0.12}$ | 72.13$_{\pm 0.12}$ | **69.14**$_{\pm 0.14}$ | **63.68**$_{\pm 0.37}$ |
| Adaptive Scheduler 2 | **73.80**$_{\pm 0.17}$ | **72.47**$_{\pm 0.20}$ | 69.10$_{\pm 0.09}$ | 62.73$_{\pm 0.15}$ |

Table 5.1: Performance using an Adaptive Scheduler for 160 epochs

Table 5.2 presents the results of all experiments with the first adaptive scheduler for 30 epochs rather than 160, to evaluate the $l$ constant.

| Ratio | 90% | 95% | 98% | 99% |
|---|---|---|---|---|
| ResNet-20 (1.096M Params) | | | | |
| Cubic Scheduler | **69.77**$_{\pm 0.29}$ | 67.87$_{\pm 0.27}$ | 63.78$_{\pm 0.30}$ | 58.42$_{\pm 0.14}$ |
| Adaptive Scheduler 1 | 69.29$_{\pm 0.09}$ | **67.96**$_{\pm 0.30}$ | **63.94**$_{\pm 0.36}$ | 58.48$_{\pm 0.08}$ |
| Adaptive Scheduler 2 | 69.36$_{\pm 0.18}$ | 67.60$_{\pm 0.30}$ | 63.71$_{\pm 0.23}$ | **58.67**$_{\pm 0.44}$ |
| MobileNetV1 (3.315M Params) | | | | |
| Cubic Scheduler | 64.96$_{\pm 0.19}$ | 64.41$_{\pm 0.25}$ | 61.47$_{\pm 0.31}$ | 57.73$_{\pm 0.49}$ |
| Adaptive Scheduler 1 | 64.98$_{\pm 0.33}$ | 64.21$_{\pm 0.08}$ | 61.76$_{\pm 0.31}$ | **57.87**$_{\pm 0.39}$ |
| Adaptive Scheduler 2 | **65.01**$_{\pm 0.26}$ | **64.47**$_{\pm 0.26}$ | **62.14**$_{\pm 0.71}$ | 57.79$_{\pm 0.47}$ |
| DenseNet40-24 (0.714M Params) | | | | |
| Cubic Scheduler | 69.49$_{\pm 0.14}$ | **67.04**$_{\pm 0.40}$ | 62.07$_{\pm 0.25}$ | **56.37**$_{\pm 0.12}$ |
| Adaptive Scheduler 1 | 69.56$_{\pm 0.03}$ | 66.96$_{\pm 0.20}$ | 62.27$_{\pm 0.14}$ | 56.35$_{\pm 0.20}$ |
| Adaptive Scheduler 2 | **69.59**$_{\pm 0.11}$ | 66.49$_{\pm 0.25}$ | **62.41**$_{\pm 0.15}$ | 55.19$_{\pm 0.08}$ |

Table 5.2: Performance using an Adaptive Scheduler for 30 epochs

## 5.5 Conclusions

In this chapter, we explored the relationship between pruning mask stability, target sparsity, and model performance. We showed that as the sparsity target increases to very high levels (such as 98% and 99%),

the pruning masks become destabilized, leading to significant drops in accuracy. This destabilization could be quantitatively measured using the Jaccard similarity index, which allowed us to identify the correlation between mask stability and performance.

Based on these observations, we proposed an adaptive pruning scheduler that adjusts the sparsity target dynamically during training by using the difference in mask stability between consecutive training iterations. We experimented with two schedulers from the same function family with very different constants. The first adaptive scheduler demonstrated significant improvements for very high sparsity targets, such as 99%, by reaching these levels more gradually and providing the model with more time to stabilize at each stage. It did not manage though to keep the average stability high enough, which is why we evaluated another scheduler. However, the second adaptive scheduler, while keeping the overall mask stability high enough, presented drops in accuracy since in its task of keeping a high stability and a small drop in accuracy it reached the target sparsity ratio too late in training.

The adaptive pruning scheduler provides a small solution for training in extreme sparsity levels. While the first version of the scheduler showed improved accuracy in many cases, further research, particularly for the scaling factor and bias terms, could lead to even better results, while also maintaining a high enough stability and a smaller drop. Future work could focus on optimizing these constants and exploring methods to derive the bias term without setting a new one for each model, while also testing the adaptive scheduler with other pruning modules like ST-3 and Spartan.

# Chapter 6

# A Study in Pruned Weights' Significance

Contents

## 6.1 Introduction

Using the Straight-Through Estimator (STE) [7], Feather [27] as well as several SoA modules in unstructured pruning achieve high performance in sparse training. Sparse training with the STE involves applying the pruned version of the weights during the forward pass, and the unpruned during the backward pass. This is achieved by treating the thresholding function $\mathcal{P}_{(T)}$, which performs the pruning, as an identity function when computing gradients. As a result, the gradients flow through the pruned weights as if no pruning had occurred. This allows the model to successfully sparsify the model while still keeping the advantages gained from a fully dense optimization process. This results in much better performance than discarding the pruned weights during backward passes. The weight updating and thresholding equations are now as follows:

$$\tilde{\boldsymbol{w}}_k = \mathcal{P}_{(T)}(\boldsymbol{w}_k)$$

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \cdot \nabla \mathcal{L}(\tilde{\boldsymbol{w}}_k)$$

In Feather's ablation studies it becomes evident that the use of the STE accounts for a difference in final accuracy from 4% to more than 10% in contrast to not using it at all. This large success of the STE points towards the fact that out of all the pruned weights, at least a small percentage of the largest ones hold some significance.

This chapter aims to investigate this significance of the pruned weights. Beyond finding which pruned weights help with accuracy, the potential computational benefit from eliminating a percentage of weights completely from the training process is a central motivation. Providing the model with sparse gradients during training could offer large reductions in training time and memory usage. If a subset of weights can be entirely removed from the optimization process, this could mean fewer operations per iteration, lower memory bandwidth usage, and reduced parameter updates.

The former possibility along with the success of the STE in enhanced performance motivates us to understand whether the high accuracy enabled by the STE is uniformly necessary across all sparsity levels or whether some portion of the weights is indeed redundant during both the forward and backward passes. If such a threshold exists, it would imply that sparse training can be further optimized not just for inference, but for training efficiency as well.

To prove the existence of this threshold and derive it, we design a set of ablation experiments on various architectures and sparsity targets, assessing both global and layer-wise thresholding strategies. The experiments involve dropping from the optimization process a fixed percentage of weights either from the beginning of training or after the desired sparsity level is reached. The question is whether a threshold exists under which the non-STE method can be safely applied and above which the STE is necessary. This would suggest that a portion of weights is truly redundant and does not significantly influence the model's ability to predict classes accurately. To explore this, we analyze the significance of weights by entirely removing varying percentages of them from the optimization process either from the beginning of training or after the model reaches target sparsity.

## 6.2 Method of Evaluation

Feather determines the threshold $T$ globally in each training iteration by receiving the target sparsity from the pruning scheduler and sorting all prunable parameters across all layers. The threshold is then calculated to meet the global target sparsity, uniformly distributing the desired sparsity throughout the network. For the experiments conducted in this section, a new threshold $t$ must be calculated. This can be achieved with two methods, where the threshold is either the same for each layer or unique, both analyzed in Section 6.2.1. There is also the question of when the decaying will begin, as described in Section 3.3.3. We consider two different timeframes, one where the decay starts straight away in the beginning of training along the sparsification, and one where all weights are allowed to be updated up until the desired sparsity is reached, after which the weights with the smallest magnitude are completely excluded from the updating process.

### 6.2.1 Thresholding

#### Global Threshold

The global pruning ratio in Feather is calculated at each iteration based on the desired sparsity ratio. For these experiments, a further step is added, in which the desired global sparsity $S_k \in [0, 1]$ is multiplied by a parameter $\rho \in [0, 1]$ that denotes the decimal fraction of weights dropped.

$$t = \rho \cdot S_k$$

As such, Feather now calculates two global thresholds using the same function, one for sparsity $S_k$ and one for $\rho \cdot S_k$. Gradients corresponding to values below the second threshold are not updated, effectively decaying them and eliminating them from optimization. The backward pass is modified as shown:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \cdot \boldsymbol{m}_k \odot \nabla \mathcal{L}(\tilde{\boldsymbol{w}}_k)$$

Where $\boldsymbol{m}_k \in \{0, \theta, 1\}^N$ such that $m_{i,k} = 1$ if $w_{i,k} > T$, $m_{i,k} = \theta$ if $t \leq w_{i,k} \leq T$ and $m_{i,k} = 0$ if $w_{i,k} < t$ with $\odot$ denoting element-wise product.

Throughout our experients, the global elimination threshold always converged to a specific value after reaching the desired sparsity along with the pruning one.

#### Layer Specific Threshold

This threshold is based on the sparsity achieved by each layer $S_{layer,k}$ after calculating the global threshold, rather than the global target. With respect to the global threshold, all weights with a smaller magnitude get pruned. Since each layer has a different weight distribution, it achieves a different sparsity than the global one. The layer-specific threshold $t_k$ is now calculated by multiplying the $l$-th layer's achieved sparsity by the desired scale $\rho \in [0, 1]$ of the weights to be dropped.

$$t_l = \rho \cdot S_{l,k}$$

Then, as the Feather module operates, the algorithm calculates the corresponding threshold $t_l$ after sorting all the layer weights. Thus, the backward pass is now represented as:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \cdot \boldsymbol{m}_k \odot \nabla \mathcal{L}(\tilde{\boldsymbol{w}}_k)$$

Where $\boldsymbol{m}_k \in \{0, \theta, 1\}^N$ such that $m_{i,k} = 1$ if $w_{i,k} > T$, $m_{i,k} = \theta$ if $t_l \leq w_{i,k} \leq T$ and $m_{i,k} = 0$ if $w_{i,k} < t_l$ with $\odot$ denoting element-wise product.

Throughout our experiments and across all layers, the elimination thresholds always converged to specific values after reaching the desired sparsity, along with the pruning one.

### 6.2.2 Timeframe of Weight Elimination

#### From the Beginning of Training

Initiating the weight elimination process at the beginning of training removes the smallest weights from the optimization process at each iteration throughout the entire training period. However, this approach risks premature loss of significant connections, as they are cut off before developing. This can limit the model's ability to learn optimal representations when considering large enough percentages of decaying weights. On the other hand, for small percentages it could be beneficial, aiding with the generalization ability of the model since it can prevent unnecessary connections from even being considered.

#### After Reaching the Desired Sparsity

This approach aims to mitigate the issue described above. It will allow the model to first develop all important connections, whether pruned or not, before completely eliminating the most insignificant ones. This will also improve training speed, since calculating thresholds at each iteration is computationally heavy, therefore cutting the amount of calculations by half could mitigate the issue. By gradually decaying these weights after reaching the target sparsity ratio, the model continues to update the important parameters during the remaining training iterations while removing the least significant ones.

## 6.3 Experimental Evaluation

The percentages of dropped weights that we evaluate in our experiments are 10%, 25%, 50%, 75%, and 90%. This percentage does not refer to the absolute amount of weights dropped, as it is relative to the overall sparsity. For example, in the global threshold approach, if the sparsity is 95% and we drop 25% of weights, 23.75% of all available weights is decayed instead of 25%. We also note the extreme cases of 0%, in which the STE is not affected, and 100%, which is essentially the non-STE approach when experimenting with the elimination of the weights from the beginning of training.

The training hyperparameters are those provided in Table 4.1 and run with ResNet20 [35], MobileNet V1 [40] and DenseNet40-24 [41] with the CIFAR-100 dataset [48]. All results are the averages of three runs with their corresponding standard deviations. We note that training time was larger than usual, since the module calculates now more thresholds. However, the results hint to the possibility of introducing sparse gradients to the backward pass, which could mean acceleration with suitable hardware and libraries. The proof , however, of this is not in the scope of this study.

### 6.3.1 Ablation Studies

Figures 6.1 and 6.2 show the results for all percentages of weights dropped for all models and for each target sparsity with decay occurring from the beginning of training. Figures 6.3 and 6.4 show the same results with decay occurring after the model reaches the desired sparsity ratio.

#### Decaying Weights from the Beginning of Training

In some cases for both thresholding methods with the elimination happening from the beginning of training, while it mostly does not affect the performance for low percentages, decaying some of the weights can even be beneficial, achieving higher performance than the STE approach. This points to the fact that many parameters are actually harming the model's ability to generalize effectively, and by decaying them from the training process the model is able to form more meaningful connections. Most prominently, it benefits MobileNet V1, which is by far the largest network, something that suggests it overfits.

For large percentages such as 75% and 90%, performance appears to drop significantly, as expected since some significant connections that could develop in the course of training are prematurely cut. It is interesting to note that for a target sparsity of 99% the accuracy drop is less significant, whereas for 90% the drop happens immediately.

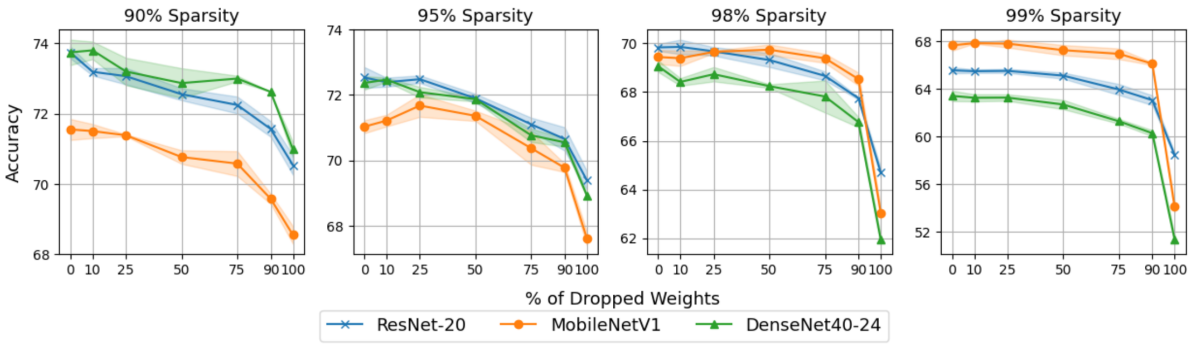



Figure 6.1: Ablation study on the effect of dropping a percentage of weights with a global threshold from the beginning of training

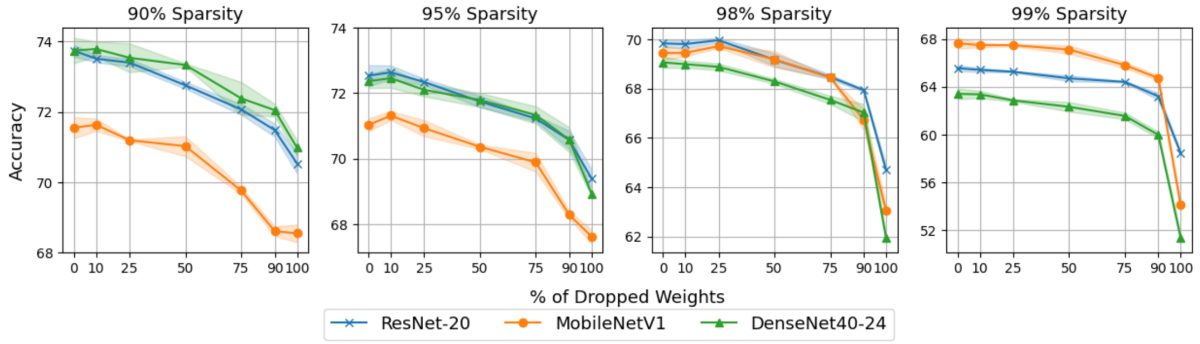



Figure 6.2: Ablation study on the effect of dropping a percentage of weights with a layer-wise threshold from the beginning of training

### Decaying Weights after Reaching Target Sparsity

When the least significant weights are eliminated from the training process after reaching the target sparsity, the results change drastically as is obvious in Figures 6.3 and 6.4. In this case, the performance is relatively steady throughout all dropped weights percentages up until the 100% mark, neither improving nor worsening the final accuracy. This come in contrast with the 10%-25% threshold in previous experiments, where a rise in accuracy was observed and then an immediate drop. This is because now the most important connections are allowed to develop during training, up until the target sparsity ratio is reached. When the elimination happens, the eliminated pruned weights are truly the least significant.

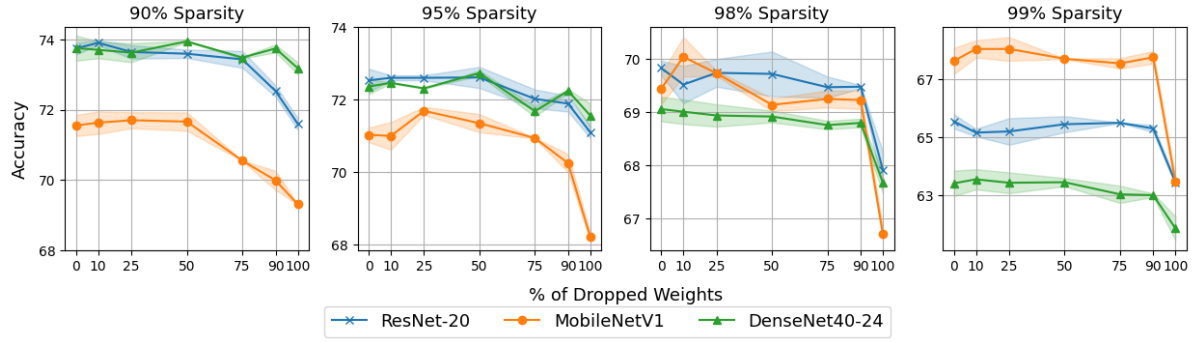



Figure 6.3: Ablation study on the effect of dropping a percentage of weights with a global threshold after reaching the desired sparsity ratio

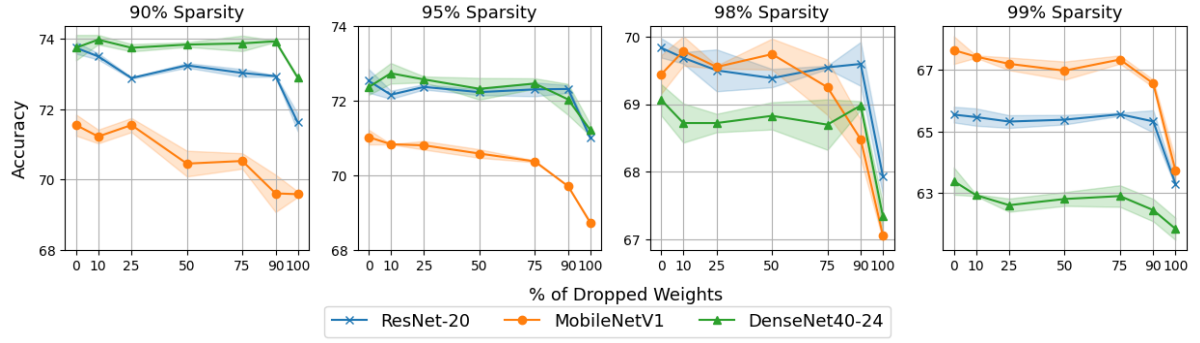



Figure 6.4: Ablation study on the effect of dropping a percentage of weights with a layer-wise threshold after reaching the desired sparsity ratio

In the extreme case of excluding all the weights below the pruning ratio (the case of 100%) from the optimization process a drop in accuracy is observed, which points to the fact that there are at least a few weights (a percentage between 0% and 10%) with a magnitude near the threshold that hold some significance in late training. Figures 6.5 and 6.6 show these experiments to figure out the threshold where dropping those weights from the training process after reaching the desired sparsity negatively affects performance.

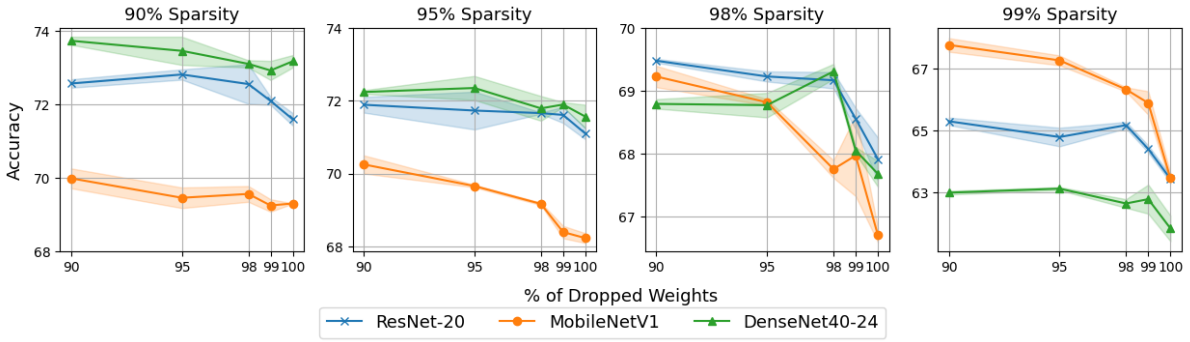



Figure 6.5: Further study on the dropping a extreme percentages of weights with a global threshold after reaching the desired sparsity ratio

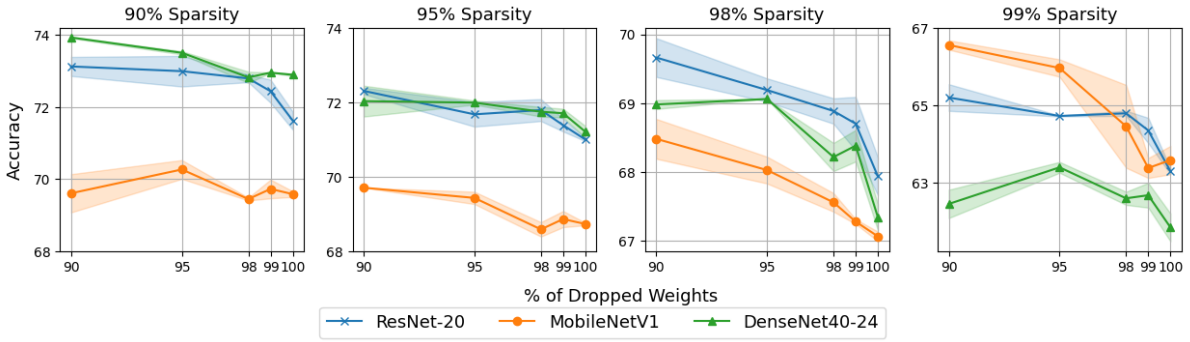



Figure 6.6: Further study on the dropping a extreme percentages of weights with a layer-wise threshold after reaching the desired sparsity ratio

### 6.3.2 Comparison

In Table 6.1 we present the best results achieved with dropping weights with the corresponding percentage for all four possible combinations of methods and timeframes. Table 6.2 presents the percentage of dropped weights were performance drops significantly. The significant drop is defined as the point at which the average accuracy, along with the bounds determined by the standard deviation, no longer overlaps with those of the model without any weight pruning. Together with the figures of the ablation studies, these experiments yield interesting results for discussion.

#### Comparison of Timeframes

When weights are excluded from the optimization process right from the start, for small percentages of decayed weights (e.g., 10% or 25%), the impact on performance is often minimal and in some cases even beneficial. This suggests that early elimination of clearly unimportant connections might improve generalization, particularly in overparameterized models like MobileNetV1. However, as the percentage of decayed weights increases beyond 50%, the performance starts to degrade significantly. This is because many of the weights that would eventually grow to be important are instead removed too early, preventing them from forming important connections an maybe later being unpruned.

When weights are decayed only after the model has reached its target sparsity, the results are far more stable across different decay percentages. In many cases, the performance remains nearly identical to that of the normal STE method, usually up until a very high threshold of weight decay (90-95%). This suggests that most of the pruned weights truly hold little to no significance after sparsification has converged. The drop in performance between 90-100% decay, however, indicates that a small number of these pruned weights, mainly those near the pruning threshold, still contribute to the model's performance when allowed to be updated late in training. DenseNet40-24 presents a curious case, as it seems to have very few pruned weights that hold any significance, with most cases presenting a drop in performance only when more than 98% of the pruned weights are eliminated. This could mean that all of the important weights were correctly identified during the training process. In the case of 90% and 95% sparsity and the global threshold method, accuracy presents a minimal and insignificant drop, furthering the point that the unpruned weights are truly the most significant.

Overall, this comparison strongly supports the conclusion that the STE's advantage comes primarily from allowing small, pruned weights to mature before being discarded. Weight significance is not only determined by the magnitude. Therefore, aggressively discarding weights from the start risks hindering the model's ability to learn, while doing so after sparsity is achieved allows for a more reliable identification of truly redundant weights. Even in that case though there are always some pruned weights that hold importance. This could point to the existence of a more sophisticated function than the identity used in the STE for the backward pass, possibly giving a different weight term to the gradients of each pruned weight based on its significance.

### Comparison of Global and Layer-Wise Thresholds

There is no consistent winner between global and layer-wise approaches. The global method seems to outperform slightly when weights are dropped after reaching sparsity, while the layer-wise method does better when pruning starts at the beginning. This might be due to the global threshold's stricter enforcement across all layers after reaching the desired sparsity, and the layer-wise method's adaptability to per layer sensitivity during early training.

| Ratio | 90% | 95% | 98% | 99% |
|---|---|---|---|---|
| ResNet-20 (1.096M Params): 73.59 $_{\pm 0.44}$ | | | | |
| Feather (0%) | 73.74$_{\pm 0.17}$ | 72.53$_{\pm 0.32}$ | 69.83$_{\pm 0.14}$ | 65.55$_{\pm 0.25}$ |
| From the Beginning of Training | | | | |
| Global | 73.19$_{\pm 0.11}$(10%) | 72.48$_{\pm 0.12}$(25%) | 69.86$_{\pm 0.30}$(10%) | 65.50$_{\pm 0.17}$(25%) |
| Layer-wise | 73.51$_{\pm 0.10}$(10%) | **72.63**$_{\pm 0.21}$(10%) | **69.96**$_{\pm 0.10}$(25%) | 65.40$_{\pm 0.16}$(10%) |
| After Reaching Sparsity | | | | |
| Global | **73.90**$_{\pm 0.09}$(10%) | 72.61$_{\pm 0.29}$(50%) | 69.75$_{\pm 0.25}$(25%) | 65.50$_{\pm 0.01}$(75%) |
| Layer-wise | 73.50$_{\pm 0.09}$(10%) | 72.37$_{\pm 0.07}$(25%) | 69.68$_{\pm 0.09}$(10%) | **65.56**$_{\pm 0.05}$(75%) |
| MobileNetV1 (3.315M Params): 71.15 $_{\pm 0.17}$ | | | | |
| Feather (0%) | 71.55$_{\pm 0.30}$ | 71.03$_{\pm 0.20}$ | 69.44$_{\pm 0.29}$ | 67.64$_{\pm 0.45}$ |
| From the Beginning of Training | | | | |
| Global | 71.50$_{\pm 0.20}$(10%) | 71.67$_{\pm 0.34}$(25%) | 69.74$_{\pm 0.21}$(50%) | 67.83$_{\pm 0.03}$(10%) |
| Layer-wise | 71.63$_{\pm 0.17}$(10%) | 71.31$_{\pm 0.13}$(10%) | 69.72$_{\pm 0.08}$(25%) | 67.48$_{\pm 0.18}$(10%) |
| After Reaching Sparsity | | | | |
| Global | **71.70**$_{\pm 0.23}$(25%) | **71.69**$_{\pm 0.11}$(25%) | **70.04**$_{\pm 0.37}$(10%) | **68.06**$_{\pm 0.41}$(25%) |
| Layer-wise | 71.55$_{\pm 0.20}$(25%) | 70.84$_{\pm 0.01}$(10%) | 69.79$_{\pm 0.22}$(10%) | 67.43$_{\pm 0.04}$(10%) |
| DenseNet40-24 (0.714M Params): 74.70 $_{\pm 0.51}$ | | | | |
| Feather (0%) | 73.75 $_{\pm 0.36}$ | 72.36 $_{\pm 0.21}$ | **69.06** $_{\pm 0.23}$ | 63.40 $_{\pm 0.44}$ |
| From the Beginning of Training | | | | |
| Global | 73.80$_{\pm 0.25}$(10%) | 72.47$_{\pm 0.09}$(10%) | 68.73$_{\pm 0.29}$(25%) | 63.25$_{\pm 0.28}$(25%) |
| Layer-wise | 73.79$_{\pm 0.21}$(10%) | 72.45$_{\pm 0.29}$(10%) | 68.99$_{\pm 0.14}$(10%) | 63.35$_{\pm 0.30}$(10%) |
| After Reaching Sparsity | | | | |
| Global | 73.94$_{\pm 0.02}$(50%) | 72.73$_{\pm 0.07}$(50%) | 69.01$_{\pm 0.23}$(10%) | **63.54**$_{\pm 0.35}$(10%) |
| Layer-wise | **73.97**$_{\pm 0.13}$(10%) | **72.74**$_{\pm 0.28}$(10%) | 68.79$_{\pm 0.22}$(10%) | 62.94$_{\pm 0.03}$(10%) |

Table 6.1: The presented accuracies are the top achieved along with their corresponding percentages of dropped weights

| Ratio | 90% | 95% | 98% | 99% |
|---|---|---|---|---|
| ResNet-20 (1.096M Params): 73.59 $\pm 0.44$ | | | | |
| From the Beginning of Training | | | | |
| Global | 10% | 50% | 50% | 75% |
| Layer-wise | 25% | 50% | 50% | 50% |
| After Reaching Sparsity | | | | |
| Global | 90% | 90% | 90% | 95% |
| Layer-wise | 25% | 95% | 95% | 95% |
| MobileNetV1 (3.315M Params): 71.15 $\pm 0.17$ | | | | |
| From the Beginning of Training | | | | |
| Global | 50% | 75% | 90% | 90% |
| Layer-wise | 25% | 50% | 75% | 75% |
| After Reaching Sparsity | | | | |
| Global | 75% | 90% | 95% | 98% |
| Layer-wise | 50% | 50% | 90% | 90% |
| DenseNet40-24 (0.714M Params): 74.70 $\pm 0.51$ | | | | |
| From the Beginning of Training | | | | |
| Global | 50% | 50% | 10% | 50% |
| Layer-wise | 75% | 50% | 50% | 50% |
| After Reaching Sparsity | | | | |
| Global | 98% | 98% | 99% | 98% |
| Layer-wise | 98% | 98% | 98% | 90% |

Table 6.2: Table that presents at which percentage performance drops significantly along with the drop

### 6.3.3 Performance Convergence

In this subsection, we explore the patterns in accuracy recovery and convergence timing following aggressive permanent weight elimination. The two smaller models (ResNet-20 and DenseNet40-24) exhibited an interesting behavior when more than 95% of their weights were permanently removed from the optimization process. This behavior was consistent across all combinations of pruning schedules and across four sparsity targets (95%, 98%, 99%, and 100%). Specifically, once the desired sparsity level was reached and the designated fraction of weights was eliminated, both networks exhibited a large jump in accuracy followed by a more gradual, slower rise that converged around epoch 140. This is in contrast with the default, meaning converging at around 155 epochs, indicating that training actually terminated early.

This pattern can be attributed to the abrupt increase in pruning mask stability $\mu$ (Section 5.2). When weights are pruned and then permanently excluded from optimization, the set of active weights becomes fixed, and no further structural changes occur within the network. This results in the pruning masks becoming fully stable, meaning that subsequent iterations are dedicated solely to refining the surviving parameters with no chance of regrowing pruned ones. As a consequence, the gradient flow becomes more consistent and less noisy, which accelerates convergence but also reduces the network's capacity to explore different connections.

Figure 6.7 illustrates this phenomenon for DenseNet40-24, displaying training curves for four levels of post-sparsity weight elimination with a global threshold (95%, 98%, 99%, and 100%). As the percentage of eliminated weights increases, the observed accuracy jump becomes larger, and convergence is achieved earlier. This confirms the hypothesis that mask stability is correlated with convergence speed.

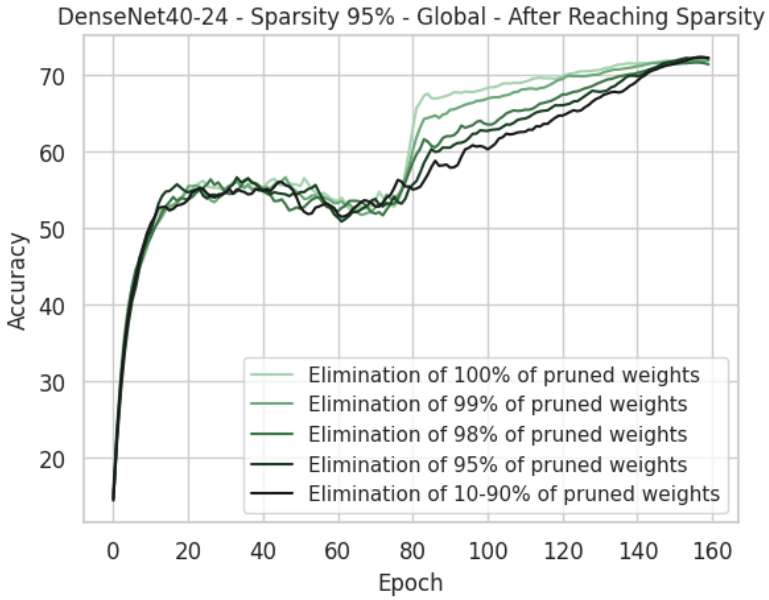



Figure 6.7: DenseNet40-24 accuracy with a target of 95% sparsity with the global method eliminating weights after reaching desired sparsity. Each color represents a different percentage while the black one shows performance for more conservative percentages (10-90%.

Figure 6.8 gives three more examples of this phenomenon by providing the curves of 100% elimination and 10-90% for simplicity. ResNet-20 shows the same behavior as DenseNet even for different configuration in elimination timeframes and target sparsity.

Unlike the smaller models, MobileNet V1 did not exhibit the same pronounced accuracy jump or early convergence behavior after reaching the target sparsity in both timeframes (elimination from the beginning of training and after reaching target sparsity). Even when high percentages of weights were permanently removed from the optimization process, MobileNet maintained a relatively smooth and gradual training curve without the sharp recovery phase observed in ResNet-20 and DenseNet40-24. Even after aggressive pruning, MobileNet still retains a very large amount of trainable parameters, thus making it more resilient to early convergence.

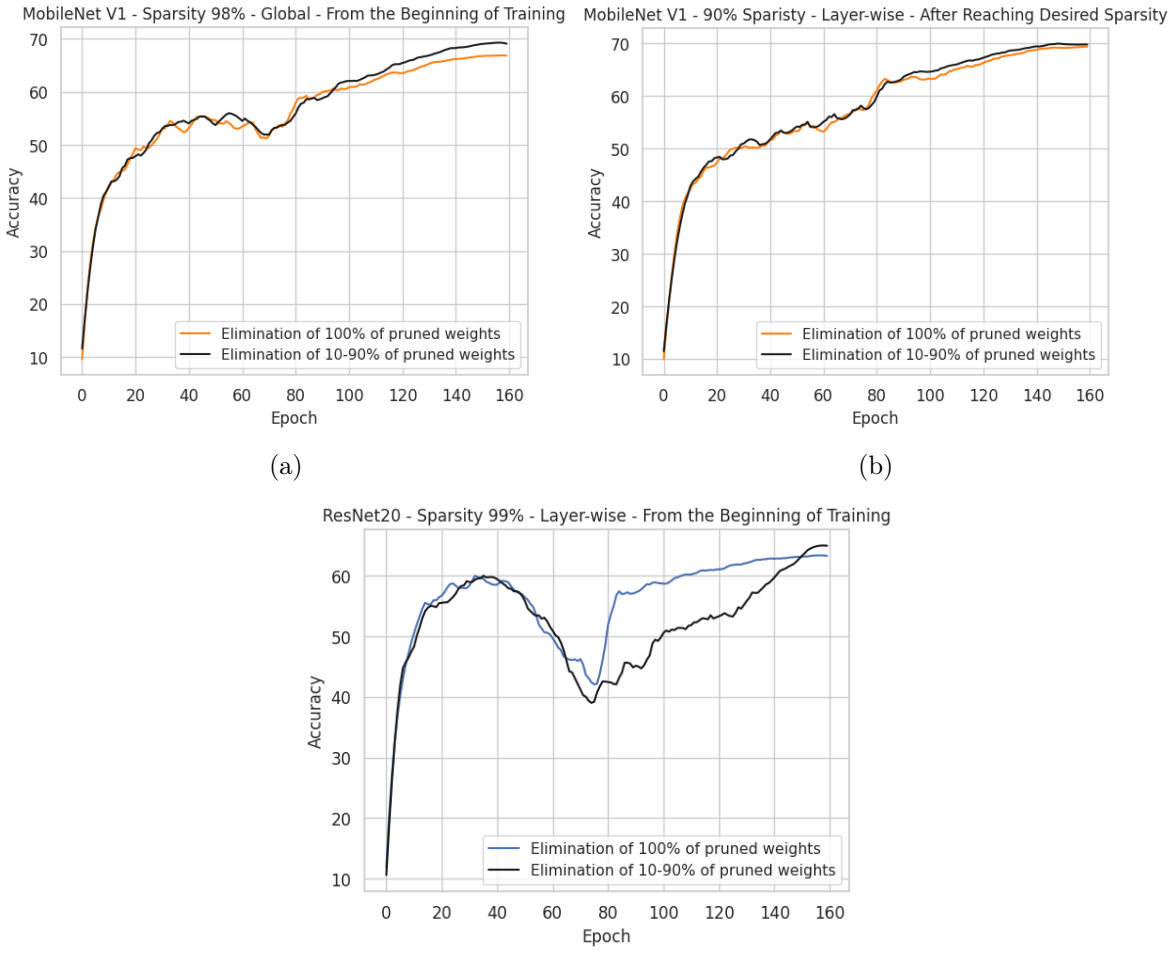



Figure 6.8: Comparison of convergence behavior across different models and weight elimination settings

### 6.3.4 Weight Distributions

In this section, we visually compare distributions for the global threshold method in order to better understand the results provided in Table 6.1 regarding the elimination of weights from the optimization process from the beginning of training. In the figures below, the original weight distribution is represented with a black line. By eliminating weights from the beginning of training it was evident that many unimportant connections never managed to fully develop, which is why we chose to only visualize the distributions of only this timeframe approach. It is important to truly see that the connections are hindered with this method, and as such this visualization was deemed more important.

It is evident that for large percentages of dropped weights the weights change significantly because important connections never formed, resulting in the accuracy drops. In contrast, for smaller percentages the distributions nearly overlap, aligning with the results in Table 6.1.

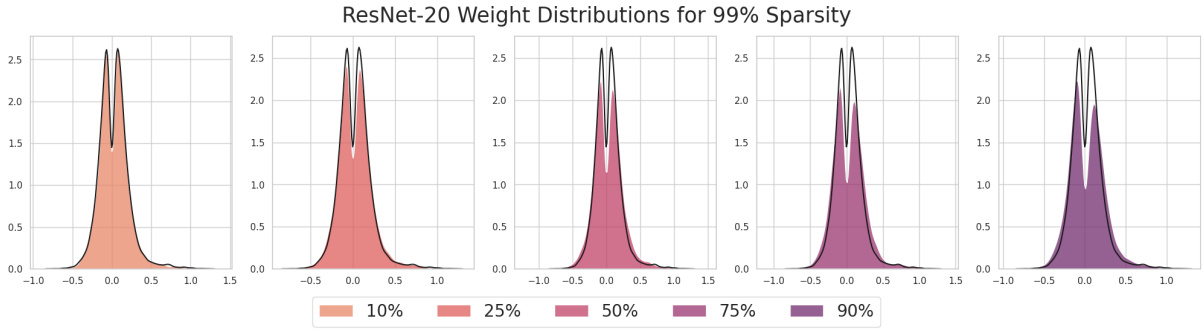



Figure 6.9: ResNet20 weight distributions with a target of 99% sparsity for all evaluated dropped weights percentages with the layer-wise method eliminating weights from the beginning of training. Each color represents a different percentage while the black line represents the original distribution.

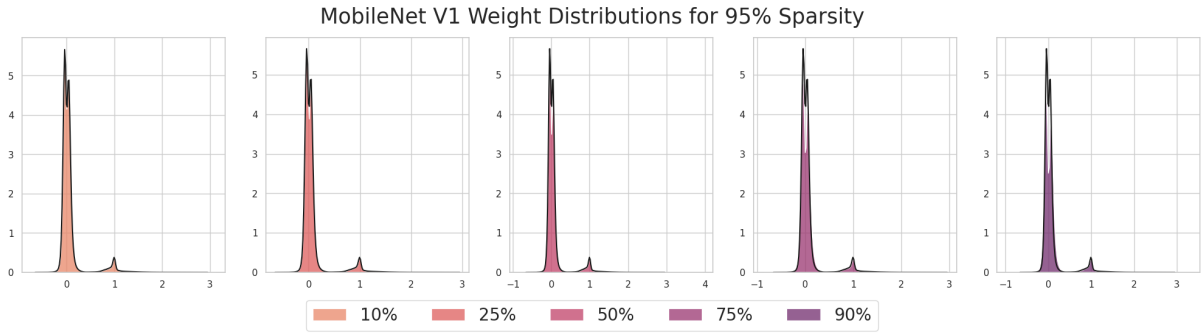



Figure 6.10: MobileNet V1 distributions with a target of 95% sparsity for all evaluated dropped weights percentages with the global method eliminating weights from the beginning of training.

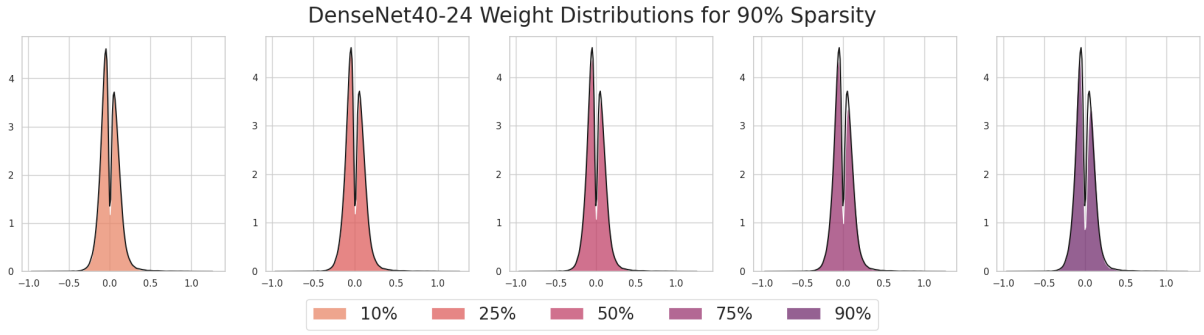



Figure 6.11: DenseNet40-24 weight distributions with a target of 90% sparsity for all evaluated dropped weights percentages with the global method eliminating weights from the beginning of training.

For clarity, in Figure 6.12 we provide an example of distributions with the other timeframe, the one where the elimination happens after reaching the desired sparsity. As mentioned, the distributions are all nearly identical with each other, even if they slightly differ from the original. This points to the fact that the most important connections where salvaged in each case.

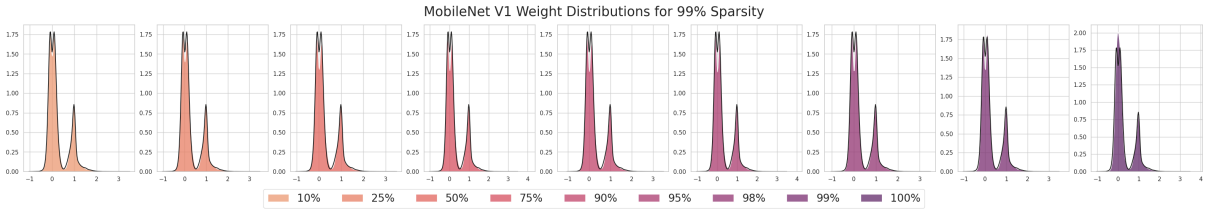



Figure 6.12: MobileNet V1 weight distributions with a target of 99% sparsity for all evaluated dropped weights percentages with the global method eliminating weights after reaching the desired sparsity.

## 6.4 Conclusions

Through these experiments, we proved that not all pruned weights are equally important. A significant portion of the pruned weights with very small magnitudes can be excluded from the optimization process without worsening performance. In some cases, eliminating a small amount (up to 25%) of the least significant weights can even be beneficial to the model's generalization ability, especially for larger models such as MobileNet V1.

Changing the timeframe of the decaying of weights until after reaching the target sparsity proves to be a more robust method, allowing important connections to develop first fully. This strategy consistently yields performance similar to that of the standard STE approach, with the exception of very high percentages. This proved that there is a small amount of pruned weights that hold some significance in late training.

These findings suggest that an approach to sparse training in which pruned weights below a second threshold are permanently excluded can yield the same results in sparsity and accuracy, while providing a better computational efficiency. While training speed is slightly larger, the introduction of sparse gradients could be more beneficial with suitable hardware and libraries. A future direction of this study could be proving this assessment. Another possible research venture could be using these observations to find of more sophisticated, differentiable functions for the backward pass than the STE.

# Chapter 7

# Conclusion and Future Work

**Contents**

## 7.1 Conclusion

This thesis focused on enhancing the effectiveness and stability of neural network pruning in the Feather module through two contributions and one quantitative strudy. These were developed as improvements upon the Feather pruning framework, which, despite its strong performance, suffers from fixed scheduling and static gradient scaling limitations.

Chapter 4 introduced a dynamic gradient scaling function that adjusts its value throughout training based on layer sparsity and the global sparsity target. This method expanded on Feather's observation in its ablation studies that the scaling of the gradients improved performance with a different value at each combination of model and target, even if they opted to a more fixed solution.

Chapter 5 proposed a function family for adaptive pruning schedulers designed to adjust the rate of pruning based on the stability of pruning masks. Two variations of the scheduler were evaluated, in which one favored gradual sparsity increases and improved accuracy at extreme sparsity, while the other maintained high pruning mask stability but often converged too late, causing slight drops in final accuracy.

In Chapter 6, we explored the significance of pruned weights in sparse training using the Straight-Through Estimator (STE). Through a series of ablation experiments, we demonstrated that not all pruned weights contribute equally to the learning process. Specifically, we found that a considerable portion of low-magnitude weights can be permanently excluded from training without significant loss in performance, especially when this elimination happens after the model has reached the desired sparsity level. This effectively allows the introduction of sparse gradients along with sparse weights in training

## 7.2 Future Work

The proposed methods achieved encouraging results, while also offering openings for further research.

- **The dynamic gradient scaling function** currently uses a logarithmic formulation based on achieved sparsity that could be further optimized. It could be possible that a more optimal function exists, using different mathematical formulations as well as different metrics. For example, using mask stability as done in the pruning scheduler rather than sparsity could yield better results.
- **The adaptive pruning scheduler** as it is now is not the optimal, but rather proof that the optimal can be found. Future research could focus on optimizing the functions' constants to find the best balance between high accuracy and high mask stability and exploring methods to derive the bias term without setting a new one for each model.
- **The study on pruned weights' significance** offered an insight to the weights close to the pruning threshold, introducing sparse gradients with a different threshold. One possible endeavor will be proving that these sparse gradients can accelerate training and testing when using suitable hardware. This insight could also be leveraged for designing more refined, differentiable functions for the backward pass than the STE.
- Finally, all these results could be replicated for different datasets and for different pruning modules to prove the versatility of those additions

# Bibliography


[1] C. C. Aggarwal, *Neural Networks and Deep Learning*, en. Springer International Publishing, 2023.

[2] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks", *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, pp. 1–18, 2017.

[3] A. I. Aramendia, *Convolutional neural networks (cnns) : A complete guide*, https://medium.com/@alejandro.itoaramendia/convolutional-neural-networks-cnns-a-complete-guide-a803534a1930.

[4] J. Back, N. Ahn, and J. Kim, "Magnitude attention-based dynamic pruning", *Expert Systems with Applications*, vol. 276, p. 126 957, 2025.

[5] A. Balasubramaniam, F. Sunny, and S. Pasricha, "R-toss: A framework for real-time object detection using semi-structured pruning", in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2023, pp. 1–6.

[6] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures", in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver, Eds., ser. Proceedings of Machine Learning Research, vol. 27, Bellevue, Washington, USA: PMLR, Jul. 2012, pp. 37–49.

[7] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation", *arXiv preprint arXiv:1308.3432*, 2013.

[8] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[9] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?", *Proceedings of Machine Learning and Systems*, vol. 2, pp. 129–146, 2020.

[10] L. Breiman, "Random forests", *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001, ISSN: 0885-6125.

[11] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression", in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006, pp. 535–541.

[12] H. Cai, C. Gan, and S. Han, "Once for all: Train one network and specialize it for efficient deployment", in *International Conference on Learning Representations (ICLR)*, 2020.

[13] L. Chen, Y. Chen, J. Xi, and X. Le, "Knowledge from the original network: Restore a better pruned network with knowledge distillation", *Complex & Intelligent Systems*, pp. 1–10, 2021.

[14] W. Cheney, "Analysis for applied mathematics", en, in Springer Science and Business Media, Jun. 21, 2001, ch. The Chain Rule and Mean Value Theorems, pp. 121–125, ISBN: 9780387952796.

[15] H. Cheng, M. Zhang, and J. Q. Shi, "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, pp. 10 558–10 578, 2024.

[16] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges", *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 126–136, 2018.

[17] T. Chin, Z. Wu, Q. Zhang, and Y. Liu, "Neural pruning via growing regularization", in *International Conference on Learning Representations (ICLR)*, 2020.

[18] C. Cortes and V. Vapnik, "Support-vector networks", en, *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep. 1995.

[19] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey", *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[20] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation", *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 27, 2014.

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding", in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[22] X. Dong and Y. Yang, "Network pruning via transformable architecture search", *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[23] D. L. Donoho, "De-noising by soft-thresholding", *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 613–627, 2002.

[24] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks", in *International Conference on Learning Representations (ICLR)*, 2018.

[25] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks", *arXiv preprint arXiv:1902.09574*, 2019.

[26] A. Gholami, S. Kim, D. Zhen, Z. Yao, M. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference", in Chapman and Hall/CRC, Jan. 2022, pp. 291–326.

[27] A. Glentis Georgoulakis, G. Retsinas, and P. Maragos, "Feather: An elegant solution to effective dnn sparsification", in *Proceedings of the British Machine Vision Conference (BMVC)*, 2023.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[29] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey", *International Journal of Computer Vision*, vol. 129, pp. 1789–1819, 2021.

[30] K. Hagiwara, "Bridging between soft and hard thresholding by scaling", *IEICE Transactions on Information and Systems*, vol. E105.D, pp. 1529–1536, Sep. 2022.

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

[32] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural networks", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[33] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon", in *Advances in Neural Information Processing Systems (NeurIPS)*, S. Hanson, J. Cowan, and C. Giles, Eds., vol. 5, Morgan-Kaufmann, 1992.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 770–778.

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

[36] Y. He and L. Xiao, "Structured pruning for deep convolutional neural networks: A survey", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 5, pp. 2900–2919, 2023.

[37] G. Hinton, l. Deng, D. Yu, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups", *Signal Processing Magazine, IEEE*, vol. 29, pp. 82–97, Nov. 2012.

[38] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network", *stat*, vol. 1050, p. 9, 2015.

[39] A. Howard, M. Sandler, G. Chu, *et al.*, "Searching for MobileNetV3", in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1314–1324.

[40] A. G. Howard, M. Zhu, B. Chen, *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications", *arXiv preprint arXiv:1704.04861*, 2017.

[41] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks", in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[42] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks", *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 29, 2016.

[43] iguazio, *What is the true positive rate in machine learning?*, https://www.iguazio.com/glossary/true-positive-rate/.

[44] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", in *Proceedings of the International Conference on Machine Learning (ICML)*, arXiv, 2015.

[45] V. Isaac–Chassande, A. Evans, Y. Durand, and F. Rousseau, "Dedicated hardware accelerators for processing of sparse matrices and vectors: A survey", *ACM Transactions on Architecture and Code Optimization*, vol. 21, no. 2, pp. 1–26, 2024.

[46] B. Jacob, S. Kligys, B. Chen, *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018.

[47] Y. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications", in *International Conference on Learning Representations (ICLR)*, 2016.

[48] A. Krizhevsky, "Learning multiple layers of features from tiny images", M.S. thesis, University of Toronto, 2009.

[49] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks", *Neural Information Processing Systems*, vol. 25, Jan. 2012.

[50] A. Kusupati, V. Ramanujan, R. Somani, *et al.*, "Soft threshold weight reparameterization for learnable sparsity", in *International Conference on Machine Learning*, PMLR, 2020, pp. 5544–5555.

[51] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned CP-decomposition", in *3rd International Conference on Learning Representations, ICLR 2015-Conference Track Proceedings*, 2015.

[52] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", en, *Nature*, vol. 521, no. 7553, pp. 436–444, May 27, 2015.

[53] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage", in *Advances in Neural Information Processing Systems (NeurIPS)*, D. Touretzky, Ed., vol. 2, Morgan-Kaufmann, 1989.

[54] N. Lee, T. Ajanthan, and P. Torr, "SNIP: Single-shot Network Pruning based on Connection Sensitivity", in *International Conference on Learning Representations (ICLR)*, 2018.

[55] A. Lheureux, *Feed-forward vs feedback neural networks*, https://www.digitalocean.com/community/tutorials/feed-forward-vs-feedback-neural-networks.

[56] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets", in *International Conference on Learning Representations (ICLR)*, 2017.

[57] Y. Li, S. Lin, J. Liu, *et al.*, "Towards compact cnns via collaborative compression", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 6438–6447.

[58] Z. Li, G. Yuan, W. Niu, *et al.*, "Npas: A compiler-aware framework of unified network pruning and architecture search for beyond real-time mobile acceleration", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 255–14 266.

[59] T. Liang, J. Glossner, L. Wang, and S. Shi, "Pruning and quantization for deep neural network acceleration: A survey", *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[60] J. Lin, H. Yin, W. Ping, P. Molchanov, M. Shoeybi, and S. Han, "Vila: On pre-training for visual language models", in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE Computer Society, Jun. 2024, pp. 26 679–26 689.

[61] H. Liu and R. Barber, "Between hard and soft thresholding: Optimal iterative thresholding algorithms", *Information and Inference: A Journal of the IMA*, vol. 9, pp. 899–933, Apr. 2018.

[62] L. Liu, S. Zhang, Z. Kuang, *et al.*, "Group fisher pruning for practical network compression", in *International Conference on Machine Learning (ICML)*, PMLR, 2021, pp. 7021–7032.

[63] S. Liu, T. Chen, X. Chen, *et al.*, "The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training", in *10th International Conference on Learning Representations, ICLR 2022*, OpenReview, 2022.

[64] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming", in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.

[65] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning", in *International Conference on Learning Representations (ICLR)*, 2018.

[66] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through $\ell_0$ regularization", in *International Conference on Learning Representations (ICLR)*, 2018.

[67] MathWorks, *What is overfitting?*, https://www.mathworks.com/discovery/overfitting.html.

[68] F. Meng, H. Cheng, K. Li, *et al.*, "Pruning filter in filter", in *Advances in Neural Information Processing Systems (NeurIPS)*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 17 629–17 640.

[69] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning", en, *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 25, 2015.

[70] A. Navada, A. N. Ansari, S. Patil, and B. A. Sonkamble, "Overview of use of decision tree algorithms in machine learning", in *2011 IEEE Control and System Graduate Research Colloquium*, 2011, pp. 37–42.

[71] F. Nielsen, "Hierarchical clustering", in Springer, Feb. 2016, pp. 195–211.

[72] D. Nikolaiev, *Supervised learning algorithms cheat sheet*, https://towardsdatascience.com/supervised-learning-algorithms-cheat-sheet-40009e7f29f5.

[73] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks", *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 28, 2015.

[74] J. Park and A. No, "Prune your model before distill it", in *European Conference on Computer Vision*, Springer, 2022, pp. 120–136.

[75] J. Price, A. Wong, J. M. Tiancheng Yuan, and T. Olorunniwo, *Stochastic gradient descent*, https://optimization.cbe.cornell.edu/index.php?title=Stochastic_gradient_descent.

[76] A. Radford, J. W. Kim, C. Hallacy, *et al.*, "Learning transferable visual models from natural language supervision", in *International conference on machine learning*, PmLR, 2021, pp. 8748–8763.

[77] P. Ren, Y. Xiao, X. Chang, *et al.*, "A comprehensive survey of neural architecture search: Challenges and solutions", *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–34, 2021.

[78] G. Retsinas, A. Elafrou, G. Goumas, and P. Maragos, "Weight pruning via adaptive sparsity loss", in *2021 IEEE international conference on image processing (ICIP)*, IEEE, 2021.

[79] R. Rigamonti, A. Sironi, V. Lepetit, and P. Fua, "Learning separable filters", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2754–2761.

[80] S. Ruder, "An overview of gradient descent optimization algorithms", *arXiv preprint arXiv:1609.04747*, 2016.

[81] H. Sahbi, "Designing semi-structured pruning of graph convolutional networks for skeleton-based recognition", *arXiv preprint arXiv:2412.11813*, 2024.

[82] V. Sehwag, S. Wang, P. Mittal, and S. Jana, "Hydra: Pruning adversarially robust neural networks", in *Advances in Neural Information Processing Systems (NeurIPS)*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 19 655–19 666.

[83] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning", *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582, 2017.

[84] K. P. Sinaga and M.-S. Yang, "Unsupervised k-means clustering algorithm", *IEEE Access*, vol. 8, pp. 80 716–80 727, 2020.

[85] S. P. Singh and D. Alistarh, "WoodFisher: Efficient second-order approximation for neural network compression", *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 18 098–18 109, 2020.

[86] X. Sui, Q. Lv, L. Zhi, *et al.*, "A hardware-friendly high-precision cnn pruning method and its fpga implementation", *Sensors*, vol. 23, no. 2, 2023, ISSN: 1424-8220.

[87] W. Sung, S. Shin, and K. Hwang, "Resiliency of deep neural networks under quantization", *arXiv preprint arXiv:1511.06488*, 2015.

[88] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey", *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[89] K. S. Tai, T. Tian, and S. N. Lim, "Spartan: Differentiable sparsity via regularized transportation", *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, pp. 4189–4202, 2022.

[90] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks", in *International Conference on Machine Learning*, PMLR, 2019, pp. 6105–6114.

[91] H. Tanaka, D. Kunin, D. L. K. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[92] C. University, *Neural networks and machine learning*, https://blogs.cornell.edu/info2040/2015/09/08/neural-networks-and-machine-learning/.

[93] A. Vanderschueren and C. De Vleeschouwer, "Are straight-through gradients and soft-thresholding all you need for sparse training?", in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2023, pp. 1125–1135.

[94] C. Wang, R. Grosse, S. Fidler, and G. Zhang, "EigenDamage: Structured pruning in the Kronecker-factored eigenbasis", in *International Conference on Machine Learning (ICML)*, PMLR, 2019, pp. 6566–6575.

[95] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow", in *International Conference on Learning Representations (ICLR)*, 2020.

[96] H. Wang, C. Qin, Y. Bai, and Y. Fu, "Why is the state of neural network pruning so confusing? on the fairness, comparison setup, and trainability in network pruning", *arXiv preprint arXiv:2301.05219*, 2023.

[97] C. J. C. H. Watkins and P. Dayan, "Q-learning", en, *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, May 1992.

[98] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[99] C.-J. Wu, R. Raghavendra, U. Gupta, *et al.*, "Sustainable AI: Environmental implications, challenges and opportunities", *Proceedings of Machine Learning and Systems*, vol. 4, pp. 795–813, 2022.

[100] C. Yeola, *Convolutional neural network (CNN) in deep learning*, https://python.plainenglish.io/convolution-neural-network-cnn-in-deep-learning-77f5ab457166.

[101] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang, "Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks", *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[102] J. Zhang, *Gradient descent based optimization algorithms for deep learning models training*, 2019. arXiv: 1903.03614 [cs.LG].

[103] A. Zhou, Y. Ma, J. Zhu, *et al.*, "Learning N:M fine-grained structured sparse neural networks from scratch", in *International Conference on Learning Representations (ICLR)*, 2021.

[104] X. Zhou, W. Zhang, H. Xu, and T. Zhang, "Effective sparsification of neural networks with global sparsity constraint", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3599–3608.

[105] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression", *arXiv preprint arXiv:1710.01878*, 2017.