Comparative Study of Pruning Techniques for Deep Neural Networks: From Small to Large Architectures

A Thesis

Submitted by

Renuka Kolusu (BS19B018)

For the award of the degree

Of

BS+MS Biological Sciences

DEPARTMENT OF BIOTECHNOLOGY
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
CHENNAI – 600036

MAY 2024

THESIS CERTIFICATE

This is to certify that the thesis entitled "**Comparative Study of Pruning Techniques for Deep Neural Networks: From Small to Large Architectures**" submitted by **Renuka Kolusu (BS19B018)** to the Indian Institute of Technology Madras, Chennai for the award of the degree of **BS+MS (Biological Sciences)** is a bona fide record of research work carried out by her under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Ganapathy Krishnamurthi
Department of Engineering Design
Indian Institute of Technology Madras
Chennai – 600036

Place: Chennai
Date: May 2024

ACKNOWLEDGEMENT

First and foremost, I am grateful to the Almighty for bestowing me an opportunity to pursue a Bachelor’s and Master’s degree at IITM. The success and final outcome of this project required a lot of guidance and assistance from many people and I wish to thank them all in this great opportunity.
I would like to express my heartfelt gratitude to my project advisor **Dr. Ganapathy Krishnamurthi** for his constant support, encouragement, and invaluable guidance during the various phases of this project.
I would like to acknowledge Research Scholars **Abjasree and Sidharth** who guided me through the process. Their feedback and continuous support are invaluable
I am confident that the skills and knowledge that I gained during my Dual Degree Project will be valuable in my future career.

Thank you again for the opportunity.

TABLE OF CONTENTS

ABSTRACT

Deep Learning models have witnessed unprecedented success in various domains, fueled by the availability of vast amounts of data and computational resources. However, the growing complexity of these models demands efficient strategies for model compression without compromising performance.

This thesis explores the application of pruning techniques for improving the efficiency of deep learning models, particularly focussing on large architectures for medical image classification. The research initially examines various compression methods and pruning strategies, including formula types and implementation on a convolutional neural network (CNN). Subsequently, the study investigates weight and unit pruning for a small architecture. The focus then shifts to large architectures, analyzing pruning techniques on the Vision Transformer (ViT) model. Finally, the research applies weight and unit pruning to both ViT and EfficientNet architectures for chest X-ray image classification. The achieved accuracy and saturation points for each configuration are compared and analyzed. By addressing pruning from both theoretical and practical standpoints, this research contributes to the understanding of optimization strategies for DNNs in various application domains.

The concluding section outlines potential avenues for future research in this domain.

KEYWORDS: Pruning, Sparse, Iterative pruning, Structured, Unstructured, Unit pruning, Weight Pruning, Tensor flow, pruning Weights, pruning Layers, Raspberry Pi, Digital Health

1.INTRODUCTION

The ever-increasing complexity of deep learning models has led to a growing demand for efficient solutions. These models often require significant computational resources, hindering their deployment on resource-constrained devices. Pruning techniques offer a promising approach to address this challenge by removing redundant or less-important weights and connections within a model.

This thesis investigates the effectiveness of pruning strategies for improving the efficiency of deep learning models, particularly focussing on large architectures for medical image classification. The research begins by exploring different compression methods and pruning techniques on a small-scale CNN architecture. It then delves into the application of weight and unit pruning for a small model. Following this, the study examines the application of pruning to a large architecture, the ViT model. Finally, the research applies weight and unit pruning to both ViT and EfficientNet architectures for a real-world medical image classification task involving chest X-ray images. By comparing the achieved accuracy and saturation points for each configuration, the research aims to provide insights into the efficacy of pruning strategies for large-scale models in the context of medical image classification.

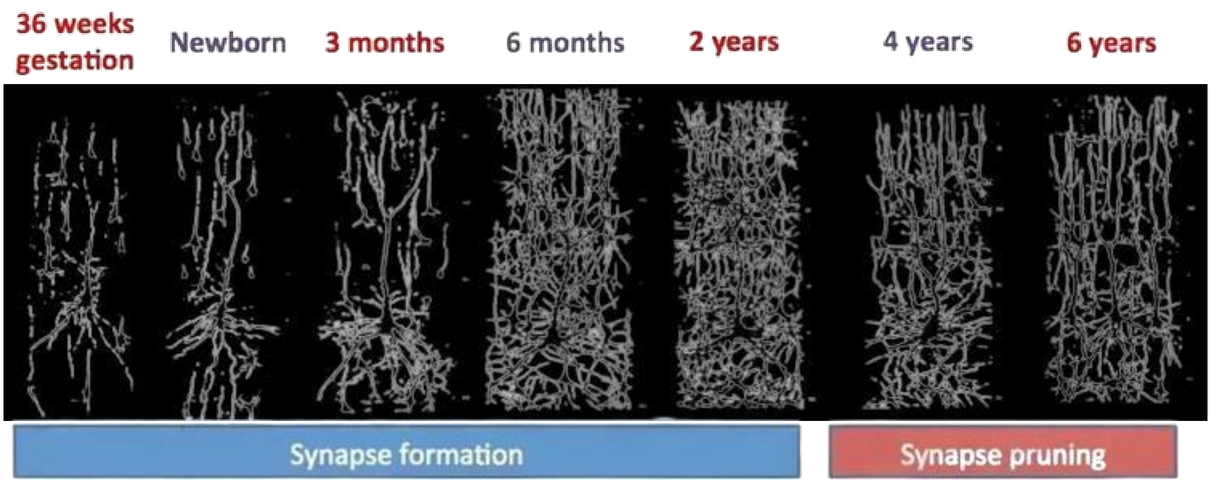
Model compression involves the method of implementing cutting-edge deep networks on devices with limited power and resources, all while maintaining the accuracy of the model. By compressing or reducing both size and latency, this approach results in a model with fewer and more compact parameters, thereby demanding less RAM.
Pruning can significantly reduce the size and complexity of deep learning models without compromising their accuracy. However, it is important to choose the right pruning method and parameters to avoid over-pruning, which can lead to a decrease in accuracy.
Other techniques include:

- Knowledge distillation: This technique involves training a smaller model to mimic the behaviour of a larger, more accurate model.
- Quantization: This technique involves reducing the precision of the parameters and activations in a deep learning model.
- Low-rank approximation: This technique involves approximating the weight matrices in a deep learning model with lower-rank matrices.
- Distributed training: This technique involves training a deep learning model on multiple GPUs or cloud computing resources.
- Quantization-aware training: This technique involves training a deep learning model in a lower-precision format while maintaining its accuracy.
- Neuromorphic computing: This is a new approach to computing that mimics the structure and function of the human brain.
- Edge computing: This involves performing deep learning computations closer to the data source.

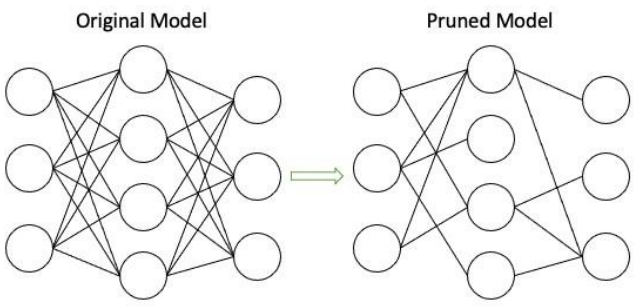
Choosing Pruning method as main focus for the project

1.1 Pruning

Pruning removes weights with minimal impact on the final model performance, resulting in a sparser model with the same architecture but fewer non-zero weights. Neural network pruning draws inspiration from the remarkable evolution of our own brains. During childhood, our brains experience rapid growth, forming a vast network of connections called synapses. As we mature, however, a fascinating "use it or lose it" process takes over. Unused connections, or synapses, are pruned away, leaving behind a more efficient and optimized network tailored to our individual needs. This process of synaptic pruning serves as the foundation for neural network pruning, which aims to achieve similar benefits in artificial intelligence by removing unimportant weights and simplifying the model structure.



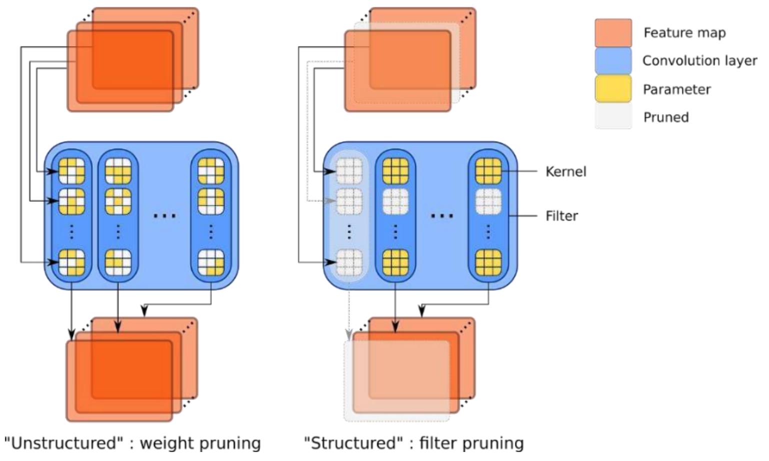
The core concept of neural network pruning involves eliminating connections deemed insignificant, maintaining the network's performance while streamlining its structure. Though not a new idea, akin to many deep learning concepts, pruning remains a vibrant research area with ongoing advancements and exploration.



1.2 Types of Pruning

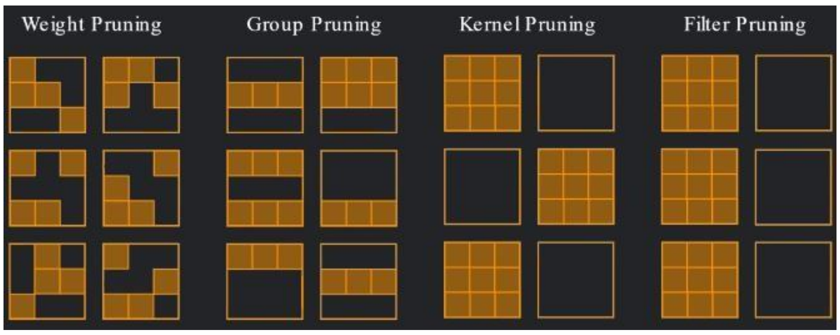
Weight pruning is the most common pruning technique used in brain networks. It involves setting some of the weights in the network to zero or eliminating them. This results in a sparser network that is faster and more effective than the first network. Weight pruning can be done in more than one way, including magnitude-based pruning, which removes the smallest magnitude weights, and iterative pruning, which removes weights during training.

Neuronal pruning involves eliminating whole neurons from the network. This can be useful for diminishing the size of the network and working on its speed and effectiveness. Neuron pruning can be done in more ways than one, including threshold-based pruning, which removes neurons with small activation values, and sensitivity-based pruning, which removes neurons that only slightly affect the result.


Feature map
Convolution layer
Parameter
Pruned
Kernel
Filter
"Unstructured" : weight pruning
"Structured" : filter pruning

Channel pruning is a technique used in convolutional brain networks (CNNs) that involves eliminating whole channels from the network. A channel in a CNN corresponds to a gathering of filters that figure out how to distinguish a specific element. Eliminating unnecessary channels can decrease the size of the network and work on its speed and effectiveness without sacrificing accuracy.

Filter pruning involves eliminating whole filters from the network. A filter in a CNN corresponds to a set of weights that figure out how to identify a specific element. Eliminating unnecessary filters can decrease the size of the network and improve its speed and effectiveness without sacrificing accuracy.



2.LITERATURE REVIEW

Large architectures like Vision Transformer (ViT) and EfficientNet are pushing the boundaries of medical image classification, leading to advancements in digital health. Here's a breakdown of their impact and how pruning these models benefits the field:

Benefits of Large Architectures in Medical Image Classification:

- **Handling Complexities:** Traditional Convolutional Neural Networks (CNNs) struggle with intricate details in high-resolution medical images (e.g., gigapixel histopathology slides). Large architectures like ViT can handle these complexities more effectively due to their inherent design.
- **Joint Analysis:** ViT goes beyond just analyzing images. It can also process textual data like radiology reports simultaneously. This combined analysis provides a richer understanding of the medical case.
- **Less Training Data:** Medical datasets are often limited due to privacy concerns and labeling costs. Large architectures like ViT often require less training data compared to traditional CNNs, making them particularly useful in this context.
- **Improved Generalizability:** The ability to learn effectively with less data can lead to better generalizability. This means the model performs well on unseen medical cases, improving its real-world applicability.

EfficientNet specifically offers:

- **Accuracy with Efficiency:** This architecture achieves a remarkable balance - high classification accuracy for medical images along with computational efficiency. This is crucial for resource-constrained healthcare settings.
- **Scalability:** EfficientNet allows for scaling the model size (depth, width, resolution) based on the desired accuracy-efficiency trade-off. This flexibility caters to diverse deployment scenarios, from powerful workstations to mobile devices.
- **Transfer Learning Advantage:** EfficientNets pre-trained on vast image datasets can be fine-tuned for specific medical image classification tasks. This significantly reduces training time and effort for medical applications.

Impact on Digital Health:

These advancements in medical image classification using large architectures contribute to:

- **Improved Medical Diagnosis:** More accurate classification of medical images like X-rays, CT scans, and mammograms can aid healthcare professionals in early and precise diagnoses. This leads to better patient outcomes.
- **Automating Workflows:** Automating tasks like image analysis through deep learning models frees up valuable time for doctors and nurses, allowing them to focus on more complex patient care aspects.
- **Personalized Medicine:** Advanced image analysis using large architectures paves the way for personalized treatment plans based on individual patient characteristics gleaned from medical images. This can lead to more targeted and effective therapies.

Benefits of Pruning Large Architectures

While large architectures offer significant advantages, their computational demands and model size can pose challenges. Pruning techniques address these concerns:

- **Reduced Model Size:** Pruning removes redundant or unimportant connections and weights within the model, resulting in a smaller model footprint. This translates to lower storage requirements, faster deployment on devices with limited storage, and potentially lower cloud storage costs.
- **Faster Inference:** Smaller models require less computational power to process medical images, enabling faster image classification. This is crucial for real-time applications in digital health, such as image-guided surgery.
- **Lower Memory Consumption:** Reduced model size translates to lower memory requirements. This makes it possible to deploy these models on edge devices with limited memory, such as mobile phones or embedded medical devices used in remote healthcare settings.
- **Potential for Improved Generalizability:** Pruning can sometimes lead to better generalization by removing noisy or overfitting weights within the model. However, careful implementation is essential to avoid sacrificing accuracy for efficiency.

2.1 Evaluating Neural Network Pruning Techniques on Visual Transformers

This research investigates the effectiveness of pruning techniques on Visual Transformers (ViTs) for model compression. Pruning reduces a model's size and computational cost by removing unimportant elements. The study compares two main pruning categories: unstructured and structured. Unstructured pruning removes individual weights, while structured pruning removes entire sub-structures like attention heads.

Considered Model : Considered Open-source unpruned ViT implemented in PyTorch 6.3-million-parameter network, 7 layers, 12 attention heads and 384 embedding dimensions Adam optimizer, batch size of 128, approx 4 hours for 200 epochs on NVIDIA T4 GPU

Dataset: CIFAR-10 consisting of 60,000 colored images
The unpruned ViT achieves 90% test set accuracy on CIFAR-10 after 200 epochs,

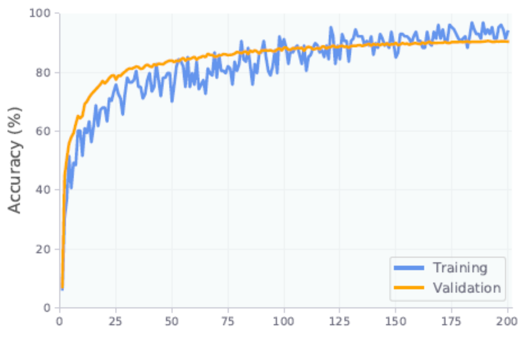


Figure: Accuracy curve for the unpruned ViT. achieved a final accuracy of 90.44%

Pruning Methodology: 3 pruning Methods - Unstructured, structured per-row, structured per-column
For each method considering two pruning schedules 1) one-shot pruning with fine tuning
and 2) iterative pruning

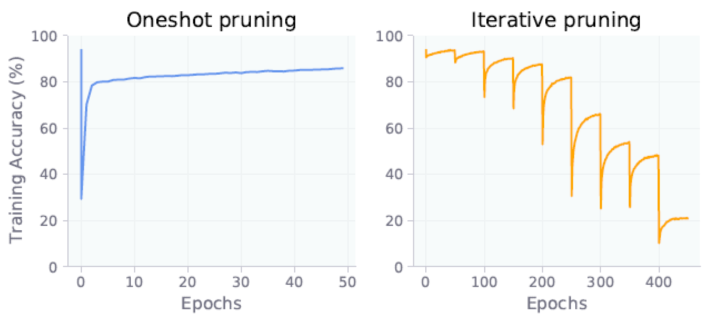


Figure: Schematic of training accuracy for the trained model after one-shot pruning with fine-tuning vs. iterative pruning.

The three main experimental setups:

1.Unstructured one-shot magnitude-based pruning removes the p% of the weights with smallest magnitude after training.

More formally, for global pruning, for a network $f(x;\theta)$ with parameters $\theta \in \mathbb{R}^{n_\theta}$ we derive a mask $\mu \in \{0,1\}^{n_\theta}$ on the parameters such that

$$\mu = \operatorname{argmin} \|(1-\mu)\odot\theta\|_1 \quad \text{s.t.} \quad \frac{\|1-\mu\|_1}{n_\theta} = r,$$

where r is the desired sparsity ratio and ⨀ is the Hadamard product. The network then continues to fine-tuning as f(x; µ ⨀ θ).

Similarly, for layerwise pruning on a layer with weight matrix $W \in \mathbb{R}^{m\times n}$, we derive a mask $\mu_W \in \{0,1\}^{m\times n}$ such that

$$\mu_W = \operatorname{argmin} \|(1-\mu_W)\odot W\|_1$$
$$\text{s.t.} \quad \frac{\|1-\mu_W\|_1}{mn} = r.$$

2.Structured one-shot magnitude-based pruning removes entire rows or columns of a weight matrix based on their l1 norm.

Formally, considering a weight matrix $W \in \mathbb{R}^{m\times n}$, for per-row structured pruning we consider a mask $\mu_W \in \{0,1\}^m$ such that

$$\mu_W = \operatorname{argmin} (1-\mu_W \mathbf{1}_n^\top)\odot W$$
$$\text{s.t.} \quad \frac{\|1-\mu_W\|_1}{m} = r,$$

where $\mathbf{1}_n$ is an n-dimensional vector of ones. Similarly, for per-column pruning we have $\mu_W \in \{0,1\}^n$ such that

$$\mu_W = \operatorname{argmin} (1-(\mu_W \mathbf{1}_m^\top))\odot W^\top$$
$$\text{s.t.} \quad \frac{\|1-\mu_W\|_1}{n} = r.$$

3.evaluate iterative versions of the above pruning methods by utilizing an iterative pruning schedule, which entails pruning in smaller steps until the desired sparsity ratio and fine-tuning between consecutive steps to minimize performance loss

The key findings are:

Unstructured Pruning

1.Global unstructured pruning achieves high sparsity -95% sparsity, validation accuracy- from 90.44% to 88.90%,

2.random pruning drops to 59.61%. 3.Layerwise unstructured pruning maintains similarly strong performance until 97.5% sparsity.

Structured Pruning

1.ViTs are less conducive to structured pruning. accuracy drops below 85% upon reaching 90% sparsity for both per-column and per-row, and it only outperforms the random baseline accuracy by approximately 10% upon reaching 98.5% sparsity.

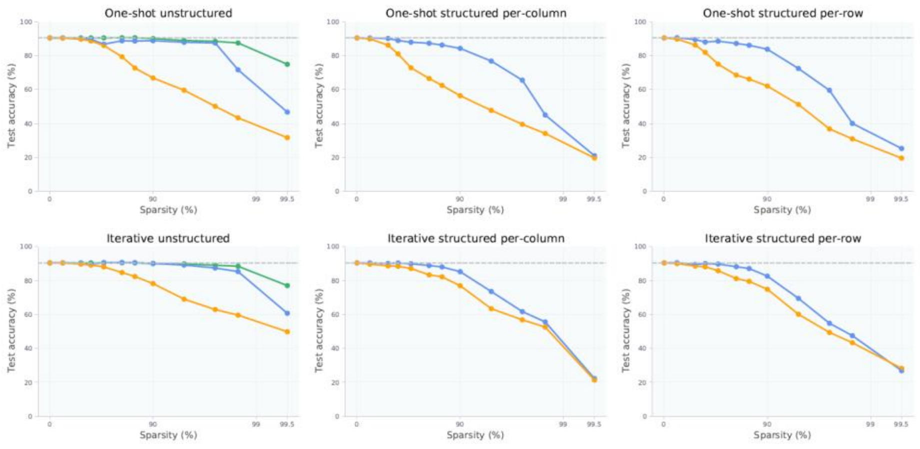


Figure: Model performance after one-shot pruning with fine-tuning (top) and iterative pruning (bottom) .Each column is the pruning method: unstructured, structured per-row ,and structured per-column. On each subplot, all pruning is ℓ1 magnitude based with distribution variants: global and layer wise, as well as a random baseline. We find that the magnitude-based methods consistently outperform their respective random baselines. Iterative varieties on average tend to perform better than their one-shot counterparts ,yet all methods show high resilience to pruning.

Analysis of Pruning on Network Modules

1.For global unstructured pruning, we see that different layers consistently get pruned at different ratios: specifically, the fully connected and embedding layers retain more parameters than other layers; earlier layers benefit from a higher density of weights in the feed-forward layers, while later layers prioritize the attention layers, with the MLP layers 4, 5 being almost entirely removed.

Transformer-specific Pruning relative importance of feed-forward layers and attention layers, we compare pruning only MLP layers and pruning only from attention layers. even when pruning all the MLP layers, relying only on residual connections between attention the network is able to maintain reasonable accuracy, whereas if we prune all the attention layers, accuracy drops down to random guessing. This highlights the fact that attention is paramount in facilitating the performance of the Transformer architecture, in line with other studies

Modelperformancewhenone-shotpruningisappliedtoMLPlayersonly,attentionlayersonly,andalllayers.For everyexperiment,weuseasetofsparsityratiosupto100%. Per-layer weight matrix sparsities generated by one-shot global unstructured pruning. For each pruning ratio, the observed per-layer sparsity varies across layer numbers and types.

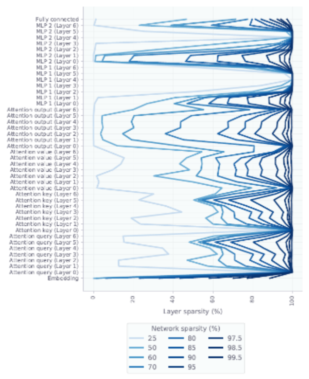


Figure: Per-layer weight matrix sparsities generated by one-shot global unstructured pruning. For each pruning ratio, the observed per-layer sparsity varies across layer numbers and types

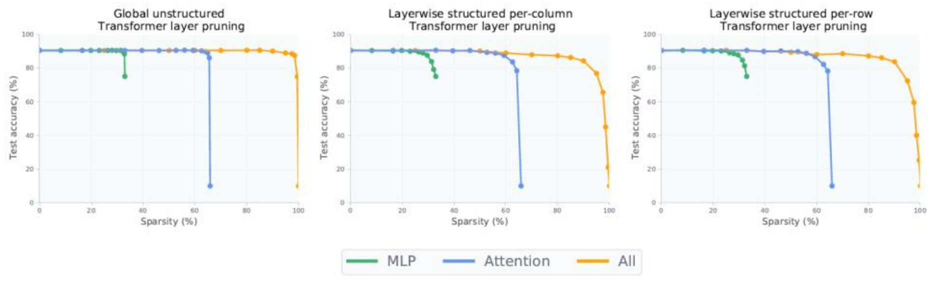


Figure: Model performance when one-shot pruning is applied to MLP layers only, attention layers only ,and all layers. For every experiment, we use a set of sparsity ratios upto 100

3.PROBLEM STATEMENT

Deep learning models, particularly large architectures such as Vision Transformer (ViT) and EfficientNet, have demonstrated remarkable performance in medical image analysis tasks, especially in the interpretation of chest X-ray images. However, the computational demands of these models pose challenges for real-world deployment, particularly in resource-constrained environments. Pruning techniques offer a promising avenue for reducing model complexity and computational overhead while preserving predictive accuracy.

This study aims to investigate the efficacy of weight pruning and unit pruning techniques in ViT and EfficientNet architectures for chest X-ray image analysis tasks.

4.OBJECTIVE OF THE WORK

This research aims to address the challenge of deploying large deep learning models for medical image analysis on edge devices. We will focus on investigating the effectiveness of pruning techniques, specifically weight and unit pruning, on ViT and EfficientNet architectures in the context of chest X-ray classification. Our goal is to:

Optimize these large architectures for edge deployment by reducing their size and computational complexity through pruning.

Evaluate the impact of different pruning strategies (weight vs. unit pruning) on model size, computational efficiency, and classification accuracy on chest X-ray datasets.

Compare the performance and saturation points of pruning applied to ViT and EfficientNet.

Identify the most suitable pruning strategy and architecture combination for efficient medical image analysis on edge devices.

5.METHODOLOGY

5.1 Pruning Implementation

Given a dataset D represented by the function f(x, y) for data points i ranging from 1 to n, and a desired sparsity level k (representing the number of unimportant weights to remove), the task of pruning can be formulated as an optimization problem:

$$\min L(w;D)= \min (1/n)\text{Summation } (w;(x,y)) \text{ s.t } w<k$$

An optimization algorithm (e.g., Adam, SGD) utilizes the gradients from the backward phase to update the weights in a direction that minimizes the loss function.
This objective function attempts to achieve two crucial goals:

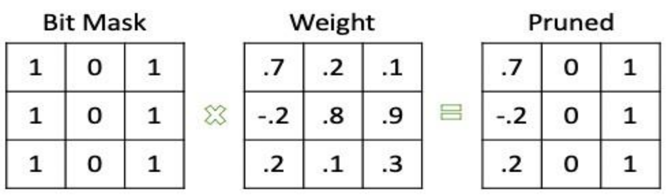
1. <u>Minimizing the Loss Function (L(w;D)):</u>
   - L(w;D) represents the model's loss function, which measures the difference between its predictions and the actual labels for a given dataset (D) and model parameters (w).
2. <u>Minimizing the Average Weight Magnitude (Σ w(x, y)):</u>
   - The sum of the absolute values of all weights (Σ |w|) represents the overall size and complexity of the model.
   - Minimizing this term encourages sparsity, meaning the model has fewer non-zero weights, leading to a smaller and more efficient representation.
3. <u>Sparsity Constraint (w < k):</u>
   - This constraint imposes a limit on the overall weight magnitude, ensuring that the model remains sufficiently complex to represent the data effectively and avoids significant performance degradation due to excessive pruning.
   - The value of k (sparsity level) acts as a control parameter, allowing you to trade-off model size and accuracy.

The formula guides the optimization process to update weights in a way that minimizes the loss while encouraging sparsity. Based on the current weight magnitudes and the sparsity constraint, the pruning strategy determines which weights to remove using the bit mask.

Pruning utilizes a binary bitmask of the same size and shape as the weight layer. This mask determines which weights contribute to training the model. The bitmask can be computed based on different pruning strategies. One such strategy prunes every even column of the weight matrix. This binary mask is continuously updated during training to achieve the desired level of sparsity. Notably, many pruning strategies focus on removing weights with the smallest magnitudes.

$$w\_new = w\_old * mask$$

- ***w_old*** is the original weight
- ***w_new*** is the pruned weight
- ***mask*** is a binary mask that indicates which weights should be kept (1) and which weights should be removed (0)



$$\begin{array}{c}\text{Bit Mask}\\ \begin{bmatrix}1&0&1\\1&0&1\\1&0&1\end{bmatrix}\end{array} \times \begin{array}{c}\text{Weight}\\ \begin{bmatrix}.7&.2&.1\\-.2&.8&.9\\.2&.1&.3\end{bmatrix}\end{array} = \begin{array}{c}\text{Pruned}\\ \begin{bmatrix}.7&0&1\\-.2&0&1\\.2&0&1\end{bmatrix}\end{array}$$

5.2 Structured Vs. Unstructured Pruning Implementation

5.2.1 Weight Pruning

```
def weight_prune (model, pruning_percentage):
model1 = copy.deepcopy (model)
length = len(list(model1.parameters()))
for i, param in enumerate(model1.parameters()):
if len(param.size())!=1 and i<length-2:
weight = param.detach().cpu().numpy()
weight[np.abs(weight)<np.percentile(np.abs(weight), pruning_percentage)] = 0
weight = torch.from_numpy(weight).to(device)
param.data = weight
return model1
```

The algorithm iterates through all layers except the last, calculating the absolute values of the weights and identifying those below a specific percentile threshold. These weights are then set to zero, effectively "pruning" them from the network. The pruned weights for each layer are stored in a list, and the original weights of the final layer are appended without any modification. This process enables efficient implementation of sparse models while preserving the essential functionality of the unpruned connections.

5.2.2 Unit Pruning

```
def neuron_pruning(model, pruning_percentage):
model1 = copy.deepcopy(model) # Create a deep copy of the model
length = len(list(model1.parameters())) # Get the number of parameters
for i, param in enumerate(model1.parameters()):
if len(param.size()) > 1 and i < length - 2:
weight = param.detach().cpu().numpy() # Convert to numpy
norm = np.linalg.norm(weight, axis=0) # Calculate norm along axis 0
if weight.shape[0] == 1 or weight.shape[1] == 1:
continue # Skip if indexing would cause out-of-bounds error
indices = np.argwhere(norm < np.percentile(norm,pruning_percentage )).flatten()
if indices.size > 0 and np.any(indices >= weight.shape[1]):
continue # Skip if indices are out of bounds
weight[:, indices] = 0 # Apply pruning by setting weights to zero
weight = torch.from_numpy(weight).to(device)
param.data = weight # Assign updated weights back to parameter
return model1 # Return the pruned model
```

This function implements unit pruning, a technique for shrinking neural networks. It iterates through each layer, except the last, analysing the L2-norm of each neuron (column) in the weight matrix. Neurons with L2-norms below a threshold determined by the percentile parameter are considered redundant and eliminated by setting their corresponding entries to zero. The active neurons for the next layer are based on the surviving neurons in the current layer, ensuring connectivity is maintained. The function tracks the number of remaining neurons (units) per layer and returns both the pruned weight matrices and the layer unit counts.

6.EXPERIMENTS AND RESULTS

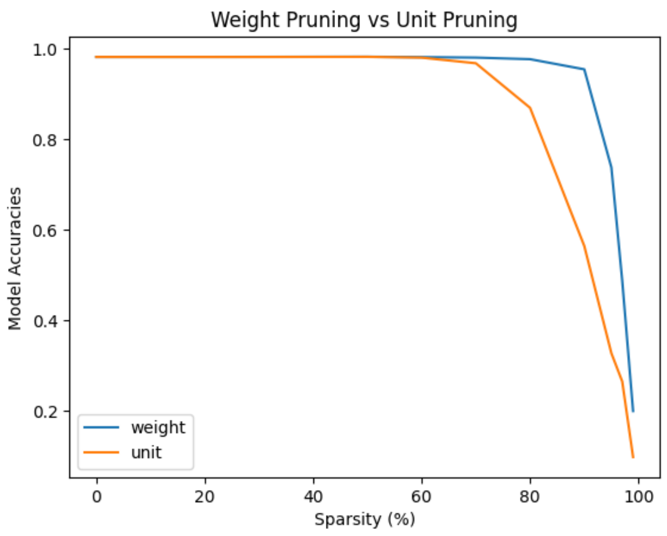
6.1 Model 1 – CNN

Dataset used is MNIST

Model Architecture -

```
Model: "sequential"
_________________________________
Layer (type) Output Shape Param #
=================================
linear (Linear) (None, 1000) 784000
linear_1 (Linear) (None, 1000) 1000000
linear_2 (Linear) (None, 500) 500000
linear_3 (Linear) (None, 200) 100000
linear_4 (Linear) (None, 10) 2000
=================================
Total params: 2,386,000
Trainable params: 2,386,000
Non-trainable params: 0
```

Pruning Method - Weight and Unit Pruning Implementation on above architecture



Results – from the above diagram its concluded that weight pruning outperformed unit pruning

6.1 Model 2 – VIT and EfficientNet

Dataset used is chest X ray Dataset. There are 5,863 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal).

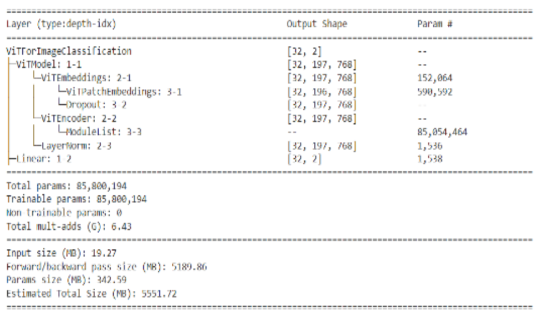
Models used Vision Transformer - VIT_B, VIT_L, VIT_H
EfficientNet - efficientnet_b1, efficientnet_b2, efficientnet_b3

Pruning Method - Weight and Unit Pruning Implementation

Model Architectures –

VIT_B on chest Xray Pneumonia Classification

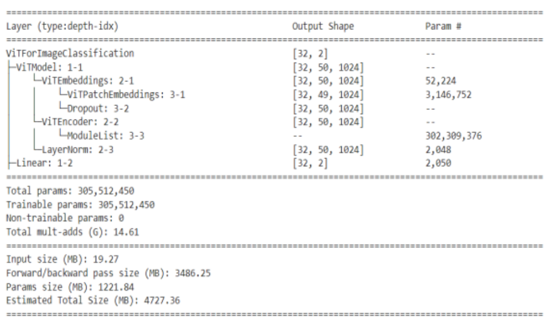
test_accuracy	88.301



```
====================================================================================================
Layer (type:depth-idx)                             Output Shape              Param #
====================================================================================================
ViTForImageClassification                          [32, 2]                   --
├─ViTModel: 1-1                                    [32, 197, 768]            --
│    └─ViTEmbeddings: 2-1                          [32, 197, 768]            152,064
│    │    └─ViTPatchEmbeddings: 3-1                [32, 196, 768]            590,592
│    │    └─Dropout: 3-2                           [32, 197, 768]            --
│    └─ViTEncoder: 2-2                             [32, 197, 768]            --
│    │    └─ModuleList: 3-3                        --                        85,054,464
│    └─LayerNorm: 2-3                              [32, 197, 768]            1,536
├─Linear: 1-2                                      [32, 2]                   1,538
====================================================================================================
Total params: 85,800,194
Trainable params: 85,800,194
Non trainable params: 0
Total mult-adds (G): 6.43
====================================================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 5189.86
Params size (MB): 342.59
Estimated Total Size (MB): 5551.72
====================================================================================================
```

VIT_L on chest Xray Pneumonia Classification

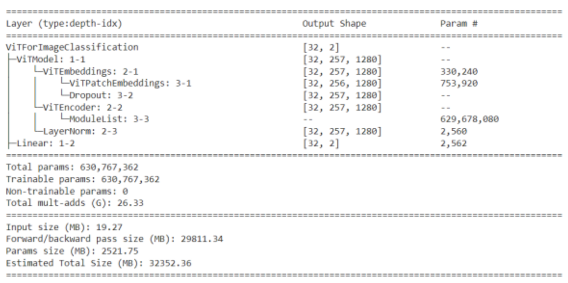
test_accuracy	84.775



```
====================================================================================================
Layer (type:depth-idx)                             Output Shape              Param #
====================================================================================================
ViTForImageClassification                          [32, 2]                   --
├─ViTModel: 1-1                                    [32, 50, 1024]            --
│    └─ViTEmbeddings: 2-1                          [32, 50, 1024]            52,224
│    │    └─ViTPatchEmbeddings: 3-1                [32, 49, 1024]            3,146,752
│    │    └─Dropout: 3-2                           [32, 50, 1024]            --
│    └─ViTEncoder: 2-2                             [32, 50, 1024]            --
│    │    └─ModuleList: 3-3                        --                        302,309,376
│    └─LayerNorm: 2-3                              [32, 50, 1024]            2,048
├─Linear: 1-2                                      [32, 2]                   2,050
====================================================================================================
Total params: 305,512,450
Trainable params: 305,512,450
Non-trainable params: 0
Total mult-adds (G): 14.61
====================================================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 3486.25
Params size (MB): 1221.84
Estimated Total Size (MB): 4727.36
====================================================================================================
```

VIT_H on chest Xray Pneumonia Classification

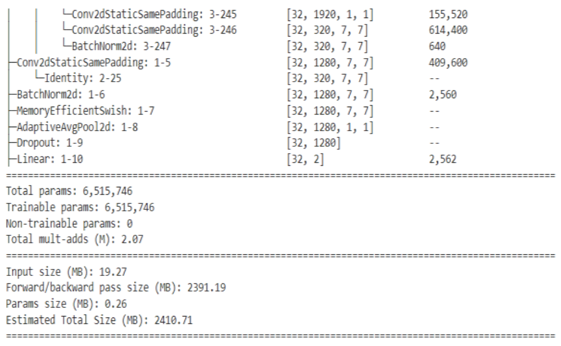
test_accuracy	83.814



```
====================================================================================================
Layer (type:depth-idx)                             Output Shape              Param #
====================================================================================================
ViTForImageClassification                          [32, 2]                   --
├─ViTModel: 1-1                                    [32, 257, 1280]           --
│    └─ViTEmbeddings: 2-1                          [32, 257, 1280]           330,240
│    │    └─ViTPatchEmbeddings: 3-1                [32, 256, 1280]           753,920
│    │    └─Dropout: 3-2                           [32, 257, 1280]           --
│    └─ViTEncoder: 2-2                             [32, 257, 1280]           --
│    │    └─ModuleList: 3-3                        --                        629,678,080
│    └─LayerNorm: 2-3                              [32, 257, 1280]           2,560
├─Linear: 1-2                                      [32, 2]                   2,562
====================================================================================================
Total params: 630,767,362
Trainable params: 630,767,362
Non-trainable params: 0
Total mult-adds (G): 26.33
====================================================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 29811.34
Params size (MB): 2521.75
Estimated Total Size (MB): 32352.36
====================================================================================================
```

EfficientNet_b1 on chest Xray Pneumonia Classification

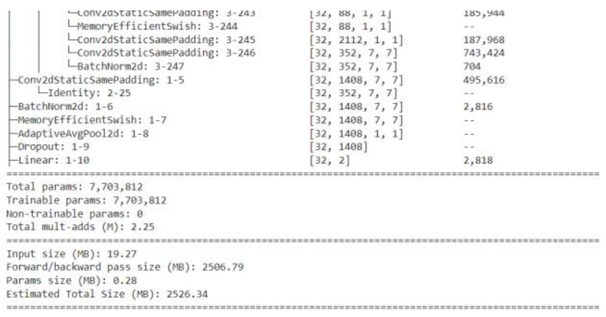
test_accuracy	85.10



```
│    │    └─Conv2dStaticSamePadding: 3-245         [32, 1920, 1, 1]          155,520
│    │    └─Conv2dStaticSamePadding: 3-246         [32, 320, 7, 7]           614,400
│    │    └─BatchNorm2d: 3-247                     [32, 320, 7, 7]           640
├─Conv2dStaticSamePadding: 1-5                     [32, 1280, 7, 7]          409,600
│    └─Identity: 2-25                              [32, 320, 7, 7]           --
├─BatchNorm2d: 1-6                                 [32, 1280, 7, 7]          2,560
├─MemoryEfficientSwish: 1-7                        [32, 1280, 7, 7]          --
├─AdaptiveAvgPool2d: 1-8                           [32, 1280, 1, 1]          --
├─Dropout: 1-9                                     [32, 1280]                --
├─Linear: 1-10                                     [32, 2]                   2,562
====================================================================================================
Total params: 6,515,746
Trainable params: 6,515,746
Non-trainable params: 0
Total mult-adds (M): 2.07
====================================================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 2391.19
Params size (MB): 0.26
Estimated Total Size (MB): 2410.71
====================================================================================================
```

EfficientNet_b2 on chest Xray Pneumonia Classification

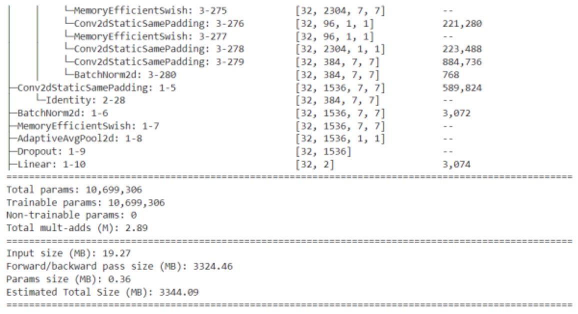
test_accuracy	87.66



```
│    │    └─Conv2dStaticSamePadding: 3-243         [32, 88, 1, 1]            185,944
│    │    └─MemoryEfficientSwish: 3-244            [32, 88, 1, 1]            --
│    │    └─Conv2dStaticSamePadding: 3-245         [32, 2112, 1, 1]          187,968
│    │    └─Conv2dStaticSamePadding: 3-246         [32, 352, 7, 7]           743,424
│    │    └─BatchNorm2d: 3-247                     [32, 352, 7, 7]           704
├─Conv2dStaticSamePadding: 1-5                     [32, 1408, 7, 7]          495,616
│    └─Identity: 2-25                              [32, 352, 7, 7]           --
├─BatchNorm2d: 1-6                                 [32, 1408, 7, 7]          2,816
├─MemoryEfficientSwish: 1-7                        [32, 1408, 7, 7]          --
├─AdaptiveAvgPool2d: 1-8                           [32, 1408, 1, 1]          --
├─Dropout: 1-9                                     [32, 1408]                --
├─Linear: 1-10                                     [32, 2]                   2,818
====================================================================================================
Total params: 7,703,812
Trainable params: 7,703,812
Non-trainable params: 0
Total mult-adds (M): 2.25
====================================================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 2506.79
Params size (MB): 0.28
Estimated Total Size (MB): 2526.34
====================================================================================================
```

EfficientNet_b3 on chest Xray Pneumonia Classification

test_accuracy	87.82



```
│    │    └─MemoryEfficientSwish: 3-275            [32, 2304, 7, 7]          --
│    │    └─Conv2dStaticSamePadding: 3-276         [32, 96, 1, 1]            221,280
│    │    └─MemoryEfficientSwish: 3-277            [32, 96, 1, 1]            --
│    │    └─Conv2dStaticSamePadding: 3-278         [32, 2304, 1, 1]          223,488
│    │    └─Conv2dStaticSamePadding: 3-279         [32, 384, 7, 7]           884,736
│    │    └─BatchNorm2d: 3-280                     [32, 384, 7, 7]           768
├─Conv2dStaticSamePadding: 1-5                     [32, 1536, 7, 7]          589,824
│    └─Identity: 2-28                              [32, 384, 7, 7]           --
├─BatchNorm2d: 1-6                                 [32, 1536, 7, 7]          3,072
├─MemoryEfficientSwish: 1-7                        [32, 1536, 7, 7]          --
├─AdaptiveAvgPool2d: 1-8                           [32, 1536, 1, 1]          --
├─Dropout: 1-9                                     [32, 1536]                --
├─Linear: 1-10                                     [32, 2]                   3,074
====================================================================================================
Total params: 10,699,306
Trainable params: 10,699,306
Non-trainable params: 0
Total mult-adds (M): 2.89
====================================================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 3324.46
Params size (MB): 0.36
Estimated Total Size (MB): 3344.09
====================================================================================================
```

Performed Weight pruning and Unit pruning on all the 6 models considered
Below is the comparison table of pruning accuracies of both pruning methods on all the 6 models and the saturation point where though the sparsity percentage increased there is no change in pruning accuracy of the particular model.

Models with original accuracy	Params	Sparsity %	10%	25%	30%	40%	50%	60%	70%	Saturation Point
VIT_B test_acc 88.301	85.8 M	Acc of Weight Pruning	87.820	75.801	72.275	67.147	62.660	62.5	62.5	60%
		Acc of Neuron Pruning	68.589	62.339	62.5	62.5	62.5	62.5	62.5	30%
VIT_L test_acc 84.775	305.5 M	Acc of Weight Pruning	84.775	80.448	78.044	68.429	64.102	62.5	62.5	60%
		Acc of Neuron Pruning	72.756	62.5	62.5	62.5	62.5	62.5	62.5	25%
VIT_H test_acc 83.814	630.7 M	Acc of Weight Pruning	83.974	83.974	83.974	83.493	80.769	72.43	62.5	70%
		Acc of Neuron Pruning	85.096	69.711	62.5	62.5	62.5	62.5	62.5	30%
EfficientNet_b1 test_acc 85.10	6.5 M	Acc of Weight Pruning	84.775	77.564	69.391	63.461	62.5	42.94	43.58	90%
		Acc of Neuron Pruning	62.339	37.5	37.5	37.5	37.5	37.5	37.5	25%
EfficientNet_b2 test_acc 87.66	7.7 M	Acc of Weight Pruning	87.660	84.134	79.487	72.275	62.5	62.5	62.5	50%
		Acc of Neuron Pruning	62.5	62.5	37.5	37.5	37.5	37.5	37.5	30%
EfficientNet_b3 test_acc 87.82	10.6 M	Acc of Weight Pruning	87.660	82.371	77.083	70.352	62.5	62.5	62.5	50%
		Acc of Neuron Pruning	62.5	37.5	37.5	37.5	37.5	37.5	37.5	25%

Results – As in the literature review mentioned the experiments also proved that unstructured weight pruning method outperformed structured neuron pruning. The table shows the accuracy of different pruning methods on various vision transformer (ViT) and EfficientNet models. The models are pruned by weight or neuron. Weight pruning removes unimportant weights from the model, while neuron pruning removes unimportant neurons. The accuracy is measured on a hold-out test set.

The table shows that the accuracy of the models generally decreases as the pruning rate increases. This is because pruning removes information from the model, which can make it less accurate. However, the amount of accuracy loss depends on the model and the pruning method.

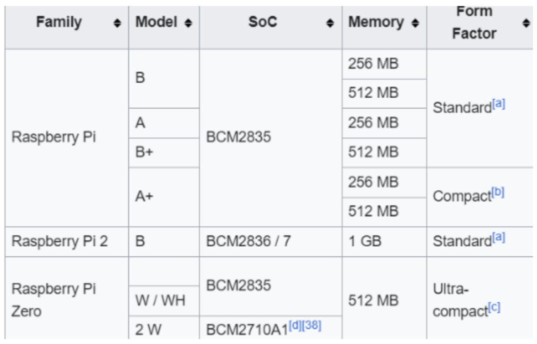
For example, the VIT_B model with weight pruning at a 10% pruning rate has an accuracy of 87.820%, which is only slightly lower than the original test accuracy of 88.301%. However, the VIT_B model with neuron pruning at a 10% pruning rate has a much lower accuracy of 68.589%. This suggests that weight pruning is a more effective pruning method for this model. Similarly, the EfficientNet_b3 model with weight pruning at a 40% pruning rate has an accuracy of 70.352%, which is still relatively accurate. However, the EfficientNet_b3 model with neuron pruning at a 40% pruning rate has a much lower accuracy than of 62.5%. This again suggests that weight pruning is a more effective pruning method for this model.

From the experiment, considering the model VIT_B and performing global unstructured pruning with 10% sparsity the size of the pruned model is 5224.7 that is 5% reduction in the size.

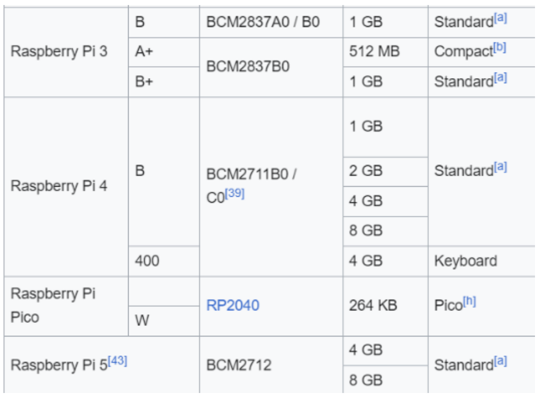
```
def prune_vit(model,percent):
 parameters_to_prune = []
 for name, module in model.named_modules():
  if isinstance(module, nn.Linear):
    parameters_to_prune.append((module, 'weight'))
 prune.global_unstructured( parameters_to_prune,
    pruning_method=prune.L1Unstructured,
    amount=percent,
 )
 return model
```

7.CONCLUSION

1.From the plots, we can clearly see Weight pruning outperforms Unit pruning when we are measuring model accuracy.
2.In case of Weight pruning the performance only starts decreasing after 50% sparsity, while in the case of Unit pruning the decline happens much earlier starting from 25% sparsity.
3.The VIT_B model with less sparsity fits in Raspberry Pi 4 model having storage limit 8 GB.



Family	Model	SoC	Memory	Form Factor
Raspberry Pi	B	BCM2835	256 MB	Standard[a]
			512 MB	
	A		256 MB	
	B+		512 MB	
	A+		256 MB	Compact[b]
			512 MB	
Raspberry Pi 2	B	BCM2836 / 7	1 GB	Standard[a]
Raspberry Pi Zero		BCM2835	512 MB	Ultra-compact[c]
	W / WH			
	2 W	BCM2710A1[d][38]		



Family	Model	SoC	Memory	Form Factor
Raspberry Pi 3	B	BCM2837A0 / B0	1 GB	Standard[a]
	A+	BCM2837B0	512 MB	Compact[b]
	B+		1 GB	Standard[a]
Raspberry Pi 4	B	BCM2711B0 / C0[39]	1 GB	Standard[a]
			2 GB	
			4 GB	
			8 GB	
	400		4 GB	Keyboard
Raspberry Pi Pico		RP2040	264 KB	Pico[h]
	W			
Raspberry Pi 5[43]		BCM2712	4 GB	Standard[a]
			8 GB	

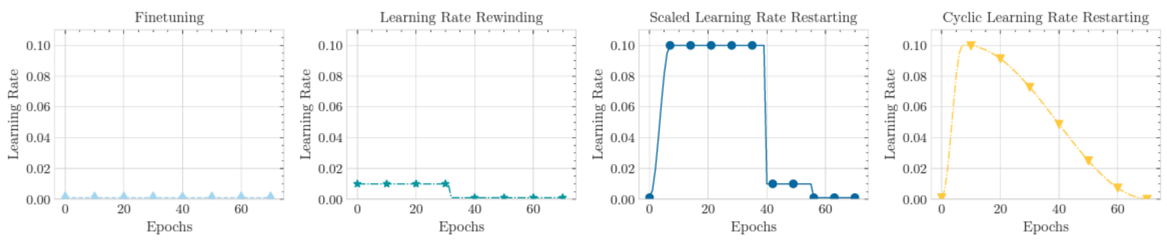
4.Synergy between Pruning and Efficient Architectures: This research demonstrates that combining weight pruning with efficient architectures like Vision Transformers (ViT) and EfficientNets offers a powerful approach for deploying medical image analysis on resource-constrained devices. This synergy allows for high accuracy models that fit on devices with limited storage and processing power.

5.Unlocking Edge-based Medical Imaging: By enabling medical image analysis on edge devices, this research paves the way for exciting possibilities in digital health. Imagine real-time disease screening in remote clinics, on-the-spot diagnostics during emergencies, or even integrating medical image analysis into mobile health apps - all facilitated by efficient, compact models.

8.SCOPE OF THE FUTURE WORK

From the above experiments it's clear that large parameter architecture can be pruned. But in the experiment performed the VIT_B with 10% pruned model gives 5.4 GB which is 5% of size reduction (very less) and it was shown that it can be fit in Raspberry Pi 4 model. But if we prune the models to large sparsity the accuracy of the model decreases drastically. So to retain the model accuracy with more size reduction it can be achieved by finetuning and that needs further exploration.

This can be achieved, as shown in the 'Network Pruning That Matters: A Case Study on Retraining Variants' paper. This study highlights that retraining strategies, particularly those employing large learning rates, play a crucial role in unlocking the potential for greater size reduction in pruned models. The paper demonstrates that even randomly pruned networks can outperform carefully pruned ones if a suitable retraining strategy is implemented. The paper focus on Learning with different schedules and demonstrates which works the best



(a) FT (b) LRW (c) SLR (d) CLR

Here are some promising directions for future work, building upon these findings:

- **Investigating More Sophisticated Fine-Tuning Techniques:** The study suggests that fine-tuning can be a viable approach to maintaining model accuracy after aggressive pruning. Future work can delve into more advanced fine-tuning techniques, such as architecture search or knowledge distillation, to achieve a better balance between model size reduction and accuracy preservation.
- **Exploring Pruning Strategies and Architectures:** The study employed random pruning, which yielded surprisingly good results when combined with a large learning rate retraining strategy. It would be beneficial to investigate more sophisticated pruning techniques, such as architecture search or channel pruning, to determine if they can achieve even greater compression ratios while maintaining accuracy.
- **Co-Pruning and Retraining:** This work focused on pruning pre-trained models. An interesting future direction would be to explore co-pruning and retraining, where pruning and retraining happen simultaneously during the training process. This co-adaptation might lead to more compact models with better accuracy.
- **Hardware-Aware Pruning:** Pruning strategies can be tailored to target specific hardware platforms. Future work can explore designing pruning algorithms that consider the hardware constraints of the target deployment platform, such as memory bandwidth or computational resources.

By delving deeper into these areas, researchers can develop more effective network pruning techniques that can significantly reduce model size while ensuring accuracy, paving the way for deploying these models on resource-constrained devices.

9.REFERENCES

1. GAURAV MENGHANI, Google Research, USA, "Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better"

2. https://nathanhubens.github.io/posts/deep%20learning/2020/05/22/pruning.html

3. https://medium.com/@souvik.paul01/pruning-in-deep-learning-models-1067a19acd89

4. https://morioh.com/a/afb5d52d3f18/tensorflow-model-optimization-toolkit-pruning-api

5. https://github.com/gowtham1997/Analysis_Of_Pruning_Techniques

6. https://towardsdatascience.com/pruning-neural-networks-1bb3ab5791f9

7. pruning-in-deep-learning-models- from medium.com

8. https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-forbeginners-a833c686ae6c

9. Sarah Chen, Victor Kolev, Kaien Yang (Stanford University), Jonathan Frankle Mosaic ML "Evaluating Neural Network Pruning Techniques on Visual Transformers"

10. Dataset https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia

11. Duong H. Le VinAI Research , Binh-Son Hua VinAI Research and VinUniversity, Vietnam "Network Pruning That Matters: A Case Study on Retraining Variants"

CODE IMPLEMENTATION LINK:

1. Pruning on CNN model

2. Pruning on VGG16 model

3. Analysis on different pruning techniques

4. Pruning VIT and EfficientNet trained on Medical Image data