DSG

DSG BYOP Reproducibility Track , 2024

**Reproducibility report: [Re] UPop: Unified and Progressive Pruning for Compressing Vision-Language Transformers**

By Swadesh Swain

Indian Institute of Technology Roorkee

Reproducibility Summary

# Introduction

The paper introduces a novel framework called Unified and Progressive Pruning (UPop) aimed at efficiently compressing multimodal models, specifically those that integrate vision and language using Transformers. It presents a unique approach by 1) allowing for the unified search of multimodal subnetworks within a continuous optimization space, which automatically determines the pruning ratios across different compressible components and structures; and 2) employing a progressive method of searching and retraining these subnetworks to ensure stable convergence and achieve greater compression efficiencies. This area of research, particularly the compression of vision-language Transformers, is relatively unexplored, making the UPop framework a significant contribution.

The number of parameters and FLOPs of deep learning models have proliferated in recent years, which makes compression exceedingly critical for deploying the increasingly heavier models on edge devices. The paper examines several techniques for model compression, including factorization , quantization, parameter bootstrapping, knowledge distillation, and pruning. It particularly emphasizes pruning, highlighting its advantages such as leveraging the optimized parameters from the original model and offering a versatile framework for adapting to different architectures.

The paper offers a novel pruning algorithm for multimodal and unimodal vision language Transformer models. It does this by the 2 pillars of it’s algorithm, Unified Pruning and Progressive Pruning:

1. Unified Pruning:

The core idea of *Unified Pruning* is to unifiedly instead of separately search on different modalities and structures, which enables adaptively instead of manually assigning the appropriate pruning ratio to each compressible component.

It is noticed that simply uniting different structures degrades performance, and the reason is that the magnitude of the learned masks used for different structures vary greatly. This can solved by applying normalization (z-score in this case) to bring the values within an expected range.

1. Progressive Pruning:

Retrain the pruned model after the search is a traditional two- stage paradigm. However, this paradigm fails when it comes to high compression ratios, because there is no guarantee that the magnitude of searched mask corresponding to the eliminated neurons in compressible components will converge to 0, which makes the pruned subnet with the parameters sliced fromdifficult to converge.

To address the above issue, the paper proposes *Progressive Pruning*, whose core idea is to ensure each magnitude of the trainable mask corresponding to the eliminated neurons in compressible components converges to 0. This is achieved by updating trainable mask with a customed optimizer that is a function of the current iteration number , instead of updating the trainable masks with the sane optimizer.

Progressive Pruning can still enable the compressed model to converge successfully, while both Mask-based Pruning and Unified Pruning fail.

# Scope of reproducibility

**Claim 1: The** paper claims at various compression ratios, compared to the Mask-based Pruning, Unified Pruning gains accuracy improvement.

**Claim 2:** Furthermore, Unified Pruning converges successfully at higher compression ratios, while Mask-based Pruning does not.

**Claim 3:** Unified Pruning also outperforms Magnitude-based Pruning under the same setting of the compression ratio and granularity.

Claim 4: Progressive pruning is claimed to ensure compressibility while preserving accuracy at higher compression ratios , without the use of extensive retraining or fine tuning.

**Claim 5:** Applying normalization ensures proper working of the Unified Search algorithm

**Claim 6:** Inclusion of sparsity loss pushes the weights to be pruned earlier and effectively

This reproducibility study will aim to examine the following claims and experiments:

* Examine the feasibility of compression offered by Progressive Pruning at higher compression ratios
* Aim to replicate the automatic adaptability of the Unified Search and Pruning algorithm over different structures of a model simultaneously
* The trade-off from the baseline accuracy/performance by applying both aspects of the algorithm separately and simultaneously.
* Experiment with linear and non-linear scheduling as a function of the iterations in Progressive Pruning.
* Experiments with normalisation and sparsity loss inclusion
* Threshold fixing while binarizing
* Open ended final pruning ratio

This study will primarily focus on the implementation of the UPop algorithm on unimodal transformer models, and examining the above aims.

# Methodology

Since the author’s code proved to be intangible and unsuitable for reproduction, upon consultation with DSG mentors and member , I proceed to code and implement the entire algorithm and the models from scratch from the description provided in the paper.

I extensively referred PyTorch documentation for the code, which is the reason for the amount of time I took to code the model implementation. I used a Kaggle notebook environment with a P100 GPU Accelerator. I experimented with XLA implementation of PyTorch but the code proved to be unsuccessful to have a complete run. I did not use ParamGanga as my primary environment as it was facing problems as stated by many member and ICC , also given the state of the internet connections I thought to better not risk time in hit and trial instead proceeded to create multiple accounts on Kaggle ( 5 ) to use multiple GPU runtimes. Because I had to code multiple versions with several modifications in them and I couldn’t hope to do that in a single pipeline. At times I had had to train 10-12 models simultaneously and thought this was the best approach to proceed on.

I used various methods, functions and classes in my code to include each and every nook of the algorithm and it’s complexities, though the manual implementations of these slowed the runs down significantly. Due to time and resource constraints I could only implement the algorithm comprehensively on a Data Efficient Image Transformer (DeiT) model (also used in the original paper) performing image classification on the CIFAR-10 dataset. I use extensive code for precise debugging and step tracking throughout the run of the model. Each weight and mask element is separately and accurately tracked throughout the run. I hoped to recreate similar implementation on MNIST fashion and tiny-imagenet, and image segmentation tasks, but time constraints were exhausted in the meantime.

For my Ablation Study I experimented with the following parameters of the algorithm and even went beyond the scope of the paper’s experiments:

* Exclusion of Unified Pruning
* Exclusion of Progressive pruning
* Fixed threshold exclusion of z-score normalisation
* Fixed threshold exclusion of Sparsity Loss integration
* Variable pruning threshold with exclusion of sparsity loss calculation
* Non-linear scheduling of progressive pruning
  + Exponential Decay
  + Sigmoid
* Pre-Defined threshold for binarization (topk)

## Model descriptions

As previously stated, the study primarily focused on the implementation of the UPop algorithm on the DeiT model, specifically the ‘ facebook/deit-base-distilled-patch16-224 ’ (pretrained) model imported from HuggingFace. The specifics of the model is described as follows:

1. **facebook**: This part indicates the organization or group that developed or released the model. In this case, it signifies that the model was developed or made available by researchers affiliated with Facebook, which is now part of Meta.
2. **deit**: This stands for Data-efficient image Transformers. DeiT models are a family of Transformer-based neural networks specifically designed for image classification tasks. Unlike traditional Convolutional Neural Networks (CNNs), DeiT leverages the Transformer architecture, originally developed for natural language processing tasks, for processing images. The "data-efficient" part of the name highlights the model's ability to achieve high performance with relatively fewer data compared to similar models.
3. **base**: This indicates the size of the model. Transformer models typically come in various sizes (e.g., base, small, large) to balance between computational efficiency and performance. The "base" variant is a middle-ground option, offering a good trade-off between model complexity and resource usage.
4. **distilled**: This suggests that the model has undergone knowledge distillation during training.   
   ( Knowledge distillation is a technique where a smaller model (the "student") is trained to mimic the behavior of a larger, more complex model (the "teacher") or an ensemble of models. This process can improve the performance of the smaller model beyond what could be achieved through traditional training methods.) In the context of DeiT, distillation is used to enhance the model's accuracy without significantly increasing its complexity.
5. **patch16**: This part of the name describes how the input images are processed by the Transformer. Specifically, it means that the images are divided into patches of 16x16 pixels before being fed into the model.
6. **224**: This number specifies the resolution of the input images, which is 224x224 pixels. allows the model to balance between detail capture and computational efficiency.
7. This model was also modified further for the CIFAR-10 classification to have 10 output classes instead of 1000 as in ImageNet on which DeiT is originally trained.

* Key features of DeiT built upon a Vision transformer backbone are:
  + **Distillation Token**: One of the key innovations in DeiT is the use of knowledge distillation at training time to improve data efficiency. DeiT introduces a "distillation" token, analogous to the class token but used for distillation. This token is designed to learn from the output of a pre-trained teacher model (typically a CNN), enabling DeiT to benefit from the inductive biases of CNNs without directly incorporating CNN architecture elements.
  + **Training Strategy**: DeiT employs a tailored training strategy that includes extensive data augmentation and regularization techniques, allowing it to achieve high performance on datasets like ImageNet without requiring an immense dataset for pre-training.

An implementation on SwinTransformer model was also attempted for this study, though a successful run of the code couldn’t be done due to minor bug issues. A description of the Swin Transformer model is provided below:

This model introduces a hierarchical structure that allows for scalability across different image sizes and a more efficient computation compared to standard Vision Transformer (ViT) models. The key features include:

* **Shifted Windows**: Swin Transformer processes images through a series of non-overlapping windows. It alternates the window partitioning between layers, which enables cross-window connections and reduces computational complexity.
* **Hierarchical Representation**: It constructs feature maps of different resolutions by progressively merging neighboring windows, enhancing its capability for multi-scale representation.
* **Self-attention within Windows**: It computes self-attention within each window, significantly reducing the complexity of self-attention computation, which is quadratic in standard transformers.

## Datasets

The study primarily focused on Image classification on the CIFAR-10 dataset, a detai;ed description of which , and the preprocessing steps and transformations involved is given below:

**Statistics and Label Distribution**

* **Total number of examples:** 60,000 images
* **Number of classes:** 10
* **Examples per class:** 6,000 images
* **Class labels:** Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, Truck
* **Image size:** 32x32 pixels, color (RGB channels)

**Train / Dev (Validation) / Test Splits**

* **Training set:** 50,000 images
* **Test set:** 10,000 images
* Note: The CIFAR-10 dataset does not come with a predefined development (validation) set. Researchers often partition the training set to create a validation set for hyperparameter tuning and model evaluation during development. A common practice is to use 45,000 images for training and set aside 5,000 for validation.

**Preprocessing Steps**

1. **Resizing:** The images were resized from 32x32 to 224x224 pixels to match the input size expected by the DeiT model.
2. **Normalization:** Images were normalized using the mean and standard deviation of the ImageNet dataset: mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225]. This step is crucial for models pretrained on ImageNet, aligning the CIFAR-10 images' distribution closer to what the model expects.
3. **ToTensor Conversion:** Images were converted to PyTorch tensors to facilitate computations on the GPU.

Other Datasets aimed to be tested were:

**tiny-imagenet -**

Tiny ImageNet is a scaled-down version of the larger ImageNet dataset, which is widely used in image classification and computer vision research. It's designed to provide a more manageable framework for developing and testing machine learning models, especially for those with limited computational resources.

**Statistics and Label Distribution**

* **Total number of examples:** Approximately 110,000 images.
* **Number of classes:** 200 (reduced from the 1,000 classes in the standard ImageNet).
* **Examples per class:** 500 training images, 50 validation images, and 50 test images per class.
* **Image size:** 64x64 pixels, color (RGB channels).
* **Class labels:** A subset of the ImageNet hierarchy, including a wide range of objects, animals, and scenes.

**Train / Dev (Validation) / Test Splits**

* **Training set:** 100,000 images (500 per class).
* **Validation set:** 10,000 images (50 per class).
* **Test set:** Not explicitly defined with labels in many versions of Tiny ImageNet; however, some versions may include 10,000 images for testing without provided labels.

**Preprocessing Steps:**

Preprocessing steps for Tiny ImageNet would typically be similar to those for the specific characteristics of Tiny ImageNet, such as image size and label distribution:

1. **Resizing:** Images might be resized for specific model inputs, though they're already at a lower resolution (64x64) compared to standard ImageNet images.
2. **Normalization:** Similar to CIFAR-10 and standard ImageNet, images are often normalized. The exact mean and standard deviation values for normalization would depend on whether the model being used was pretrained on the full ImageNet dataset.
3. **Data Augmentation:** Techniques such as random cropping, flipping, and rotation might be used to increase the diversity of the training data, helping to improve model robustness.

**Tests on the tiny-imagenet dataset could not be completed in time.**

## Hyperparameters

The list of hyperparameters is as follows:

initial\_pruning\_rate = 0.1 # Example initial pruning rate

final\_pruning\_rate = 0.5 # Example final pruning rate for unified pruning

performance\_threshold = 0.02 # Example performance threshold for progressive pruning

lambda\_sparsity = 1e-4 # Sparsity weighting factor for combined loss

optimizer = optim.Adam(pruned\_model.parameters(), lr=0.001)

max\_epoch = 16

The hyperparameter search was done manually referring to common values set according to the studies conducted in the paper. The parameter max\_epoch was set in light of a smaller dataset but keeping in mind the range of pruning and complexity of the model layers involved.

* The performance threshold was kept low in the range of 0.01 to 0.05 as the pruning rate variation becomes too high and unpredictable otherwise.
* Regularization constant for sparsity loss (lambda\_sparsity ) is kept in the range of 1e-2 – 1e-4 otherwise the sparsity loss pushes the weights too quickly to tend to 0, nullifying the gradual effect of progressive pruning.
* Initial and Final pruning rates were set according to the required amount of pruning.
* Optimizer (Adam) and Learning Rate were set according to best measures from the common practices adopted and experimented specific to the model at hand

The best hyperparameter values after the thorough and repetitive experimentation with the model are displayed as example values above.

## Experimental setup and code

* The libraries and dependencies to be installed were listed and imported directly into the Kaggle environment
* Enabled and initiated integration of GPU in code
* Appropriate transforms to the dataset and modifications
* Classes and functions included for implementing various aspects of the algorithm including but not limited to:
  + Unified Search
  + Progressive Pruning
  + Scheduling
  + Binarization of masks
  + Normalization of weights copied
  + Sparsity Loss calculation and inclusion
  + Freezing of pruned weights to prevent updation
  + Code for debugging and tracking every function call and step implemented.

Link to code: <https://github.com/Swadesh06/BYOP_Repro_UPop/>

## Computational requirements

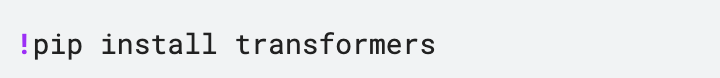
As stated previously a Kaggle environment with P100 GPU accelerator was used for the reproducibility study. The libraries to be imported and the dataset downloading , transformation and pre-processing steps are included below:

Libraries and installations:

Python code:

A screen shot of a computer code

Description automatically generated

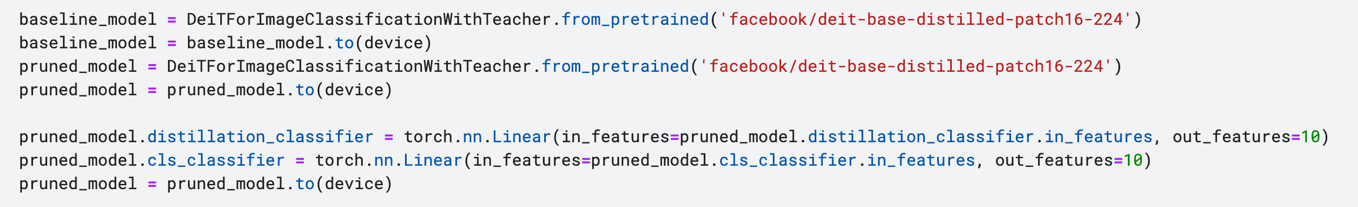


GPU initialization , pre-processing, data transformations and splitting:

A screenshot of a computer code

Description automatically generated

Model import, initiation and modification for CIFAR-10:



Average runtimes for training 16 epochs with UPop intergration on DeiT ~ 4 hours (with P100 GPU enabled)

Average prediction time on the test set ~ 5 mins (GPU enabled)

Total GPU hours including all experiments and model runs ~ 200 hours

# Results

The results produced aligned with the majority of the claims made by the paper. Progressive pruning turned out to be especially exceptional at preserving accuracy during higher compression runs (~90 percent ) and required lesser fine tuning as claimed.

Unified Pruning implementation in the algorithm proved to be significant but not as much of an effect as Progressive pruning entailed, normalizing the weight values proved to be a significant step in the working of the Unified Search part correctly, the exclusion of which led to random and exceptionally accelerated pruning due to high Sparsity Loss.

Compared to Masked Pruning the algorithm proved to be an improvement as stated, especially if the model is to be trained for a new type of dataset.

## Results reproducing original paper

**Examining the feasibility of compression offered by Progressive Pruning at higher compression ratios and Experiment with linear and non-linear scheduling as a function of the iterations in Progressive Pruning:**

* Runs with the implemented DeiT model resulted in the following:

| **Compression Ratio** | **Δ Accuracy (from baseline)** | **Scheduling** |
| --- | --- | --- |
| **0.5** | **~ -3%** | **Linear** |
| **0.8** | **~ -6%** | **Linear** |
| **0.9** | **~ -7%** | **Linear** |
| **0.5** | **~ -3%** | **Sigmoid** |
| **0.8** | **~ -6%** | **Sigmoid** |
| **0.9** | **~ -7 %** | **Sigmoid** |
| **0.5** | **~ -4.5%** | **Exponential Decay** |
| **0.8** | **~ -5%** | **Exponential Decay** |
| **0.9** | **~ -5.5%** | **Exponential Decay** |

* Improvement from mask-based pruning
  + **~ +3% (for 0.5 compression ratio)**
  + **~ +5% (for 0.8)**
  + **~ +5.5 (for 0.9)**
* Experiments with mask based pruning resulted in very high drops in accuracy with higher compression ratios and effectiveness of the pruning process declined after testing for a pruning ratio of more than 0.6 .
* **Claim 2 and 4 supported by experimental results**

**Replication the automatic adaptability of the Unified Search and Pruning algorithm over different structures of a model simultaneously resulted in:**

| Inclusion/Exclusion of automatic adapting of pruning rate | Accuracy change from baseline accuracy (for compression ratio of 0.8 with linear scheduling ) |
| --- | --- |
| Inclusion | ~ -6% |
| Exclusion | ~ 7.5% |

* Progressive pruning handles fine tuning and dampens effect of exclusion of pruning rate adaptation algorithm
* Adaptation of the pruning rate turns out to be significant improvement included in the unified search algorithm

**Experiment to examine the trade-off from the baseline accuracy/performance by applying both aspects of the algorithm separately and simultaneously:**

| **Inclusion of:** | Change in Accuracy (linear scheduling to 0.5 compression ration ) | Improvement on mask based pruning (for 0.5 compression) |
| --- | --- | --- |
| **Unified Pruning Only** | **N/A** | **N/A** |
| **Progressive Pruning** | **-5%** | **~ +4%** |
| **Unified and Progressive Pruning** | **-4%** | **~ +3%** |

* Experimental results excluding unified pruning couldn’t be performed due to computational constraints (needed to access each layers separately and iterate over it for Progressive Pruning application)
* **Supports Clam 1 from the paper**

**Experiments with normalisation and sparsity loss inclusion:**

* **Exclusion of normalization from the unified search algorithm resulted in the following:**
  + Very high Sparsity Loss – as weight values weren’t in expected range
  + Exceptionally low values of weights and therefore very similar output (SoftMax) probabilities keeping the final pruning ratio fixed.
  + With unbounded final pruning ratio , sparsity loss being high, the model converges to exceptionally high ratios (~0.995)
* This experiment ensured that normalization was essential in proper function of the algorithm supporting **Claim 5.**
* **Exclusion of Sparsity loss calculation resulted in the following:**
  + For fixed threshold binarization (without top k )and unbounded final pruning ratios, significant percentage of the weights retained higher values than the threshold, and were pruned, **effectively decreasing the magnitude of pruning**.
  + This meant that when working to compress the model as much as possible , sparsity loss inclusion will help regularize the value of weights and will be an essential component to prune the model to the highest feasible extent.
  + This supports Claim 6 of the original paper.

**Experimenting with Fixed Threshold Binarization of the masks along with open ended final pruning ratio:**

* Replacing topk pruning with fixed threshold masks performed inferiorly than the original algorithmwith respect to accuracy, but that may well be optimized by searching adaptively for a suitable pruning threshold inside the masks.
* Fixed threshold along with open ended final pruning ratio consistently resulted in the model being more pruned than expected , which might be of help if the goal is to prune the model to the highest extent without performing manual experiments with final pruning ratios

## Results beyond original paper

Additional Results:

— Application of Non-Linear Scheduling for progressive Pruning:

* Experiments with Sigmoid and Exponential Decay Scheduling resulted the following:
  + Sigmoid outperformed other schedules among moderate compression ratios (0.4 – 0.7)
  + At very high compression ratios (~0.9) Exponential Decay Schedules prevailed
  + Linear Schedules proved to be best at low to moderate Pruning ratios (<0.5)

\*\*\* **Application of Adaptive pruning based on current epoch validation accuracy:**

* On adding features to adapt the pruning rate on seeing how the model is performing currently on the validation set, by the hyperparamter performance\_threshold which measures change in validation accuracy from the previous epoch to the current , the effectiveness of the UPop algorithm was enhanced , especially at higher compression ratios , during which the trade-off in accuracy and size of the model became less pronounced
* At a 0.9 compression ratio, the Adaptive Pruning method recorded an increase 2 percent of what was without inclusion of it
* This algorithm is included into my final code due to it’s effectiveness.

# Discussion

## What was easy

* After implementing the entire code by myself , the ablation study proved to be relatively simple , only needing slight modifications to the code (except exclusion of unified pruning)
* Having multiple GPU runtime accesses made it easy to train several different codes simultaneously
* DSG mentors and members were accommodating and informative , helping rightly throughout the process

## What was difficult

* Being unable to utilize the author’s source code, the majority of the difficulties presented themselves in self implementing the algorithm form scratch into a model, requiring extensive debugging and experimentation and careful integration into each part of the model.
* Due to self implementation requirements of the project, the time constraints proved to be very difficult to follow, and experiments with several other model =s and datasets couldn’t be completed as planned
* Extensive ablation studies required proper documentation of each model, and careful debugging of the modifications done, which were the cause of much confusion.