TOWARDS META-PRUNING VIA OPTIMAL TRANSPORT

Type	Research Paper
Comments	optimal transport
Date	@June 30, 2025

Motivation

- **Model size vs. deployment feasibility:** Modern networks have overgrown in parameters and memory footprint, making on-device or large-model deployment costly. Traditional compression (pruning, quantization, distillation) recovers accuracy via expensive fine-tuning—often infeasible for very large models.
- **Fine-tuning-free compression:** Data-free methods eliminate retraining on the original dataset—vital for privacy-sensitive or resource-constrained settings—but conventional pruning still discards low-importance neurons, causing accuracy drops that almost always require fine-tuning.
- **Intra-Fusion (This work):** Inspired by federated learning and Optimal Transport–based model fusion (OTFusion), this meta-pruning framework recycles pruned neurons' information into survivors, unifying pruning and fusion into a single data-free pipeline without accuracy loss.

Optimal Transport (OT)

A methematical framework to compare probability distributions.

- **Two discrete distributions**
  - We have a "source" distribution

    $\mu = \sum_{i=1}^{n} \alpha_i \, \delta(x_i)$

    supported on points $x_1, \ldots, x_n$ with masses α_i (this could be importance scores).

- And a “target” distribution

  $\nu = \sum_{j=1}^{m} \beta_j \, \delta(y_j)$

  on points $y_1, \ldots, y_m$ with masses β_j.
- Both $\sum_i \alpha_i = 1$ and $\sum_j \beta_j = 1$.

- **Cost matrix** C
  - $C \in \mathbb{R}_+^{n \times m}$ has entries

    $C_{ij} = c(x_i, y_j)$, e.g. the squared Euclidean distance

    $\|x_i - y_j\|^2$.
- **Transport plan** T
  - $T \in \mathbb{R}_+^{n \times m}$ is a nonnegative matrix where

    T_{ij} represents how much mass you move from x_i to y_j.
- **Marginal constraints**
  - **Row sums equal source masses**:

    $T\,\mathbf{1}_m = \alpha$, i.e. $\sum_{j=1}^{m} T_{ij} = \alpha_i$.
  - **Column sums equal target masses**:

    $T^\top \mathbf{1}_n = \beta$, i.e. $\sum_{i=1}^{n} T_{ij} = \beta_j$.
- **Objective**

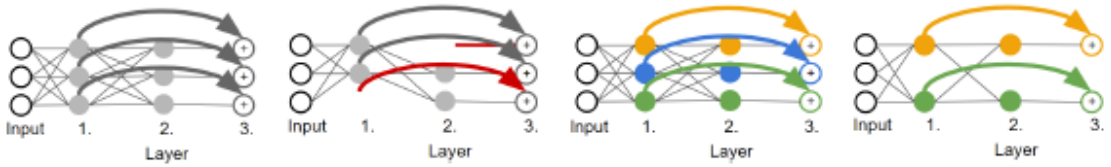
  $\min_{T \geq 0} \langle T, \, C \rangle_F \;=\; \sum_{i=1}^{n} \sum_{j=1}^{m} T_{ij} \, C_{ij}$,

  i.e. find the cheapest way to move the entire mass of μ to match ν.

Think of α_i as piles of “earth” at locations x_i and β_j as holes at y_j. We pay cost C_{ij} per unit of earth we move from pile i to hole j. OT finds the cheapest assignment of earth to holes that exactly fills each hole and empties each pile.

Algorithms and Methodologies

Structured Pruning: Group-by-Group.



(a) The beginning of an example network (b) Broken network: individually pruned layers (c) The network as a set of neuron pairings (d) Structurally pruning complete group

Data: IMPORTANCE $\boldsymbol{i}_{1\times n}$, group $\mathbb{G}$ with group cardinality n, and target group cardinality m.
Result: group $\mathbb{G}_{new}$ with group cardinality m

Algorithm 1: Conventional Pruning

$t \leftarrow m^{\text{th}}$ highest scalar in $\boldsymbol{i}$;
$\mathbb{G}_{new} \leftarrow \emptyset$;
for $layer \in \mathbb{G}$ **do**
  $layer_{new} \leftarrow layer$ without neurons j where $\boldsymbol{i}[j] < t$;
  $\mathbb{G}_{new} = \mathbb{G}_{new} \cup \{layer_{new}\}$;
end

Algorithm 2: Intra-Fusion

$\mathbb{Y} \leftarrow$ GETTARGET$(\mathbb{G})$; 3.2.1
$\mathbb{X} \leftarrow \mathbb{G}$; 3.2.1
$\boldsymbol{C}_{n\times m} \leftarrow$ COMPUTECOST$(\mathbb{X}, \mathbb{Y})$; 3.2.2
$\boldsymbol{\nu}_{1\times m} \leftarrow$ GETTARGETDISTR$(m, \boldsymbol{i})$; 3.2.3
$\boldsymbol{\mu}_{1\times n} \leftarrow$ GETSOURCEDISTR$(n, \boldsymbol{i})$; 3.2.3
$\boldsymbol{T}_{n\times m} \leftarrow \text{OT}(\boldsymbol{\mu}, \boldsymbol{\nu}, \boldsymbol{C})$; 3.2.4
$\boldsymbol{T}_{m\times n} \leftarrow \text{diag}(\frac{1}{\boldsymbol{\mu}}) \times \boldsymbol{T}^{\top}$; 3.2.4
$\mathbb{G}_{\text{new}} \leftarrow \emptyset$;
for $layer \in \mathbb{G}$ **do**
  $\mathbb{G}_{new} = \mathbb{G}_{new} \cup \{\boldsymbol{T} \times layer\}$; 3.2.4
end

Both **conventional pruning** and **Intra-Fusion** start from the same setup:

- A **group** $\mathbb{G}$ of n neuron-pairings (its **group cardinality** is n) → compress down to m pairings ($m \leq n$).
- An **importance vector** $i \in \mathbb{R}^{1\times n}$, which assigns each pairing an **agnostic** score (e.g. the ℓ_1-norm of its weights).

Conventional Pruning (Algorithm 1)

1. Find the mth largest entry in i and call it threshold t.
2. **Keep** the m pairings whose importance $\geq t$.
3. **Discard** the other $n - m$ pairings.
4. The resulting group $\mathbb{G}_{\text{new}}$ has exactly m pairings.

Most prior work has concentrated on crafting ever-better importance metrics i, but still follows this "keep-top-m, drop-the-rest" routine unchanged.

Intra-Fusion Meta-Pruning (Algorithm 2)

Rather than throwing away the less-important pairings, Intra-Fusion **fuses** their information into the survivors via an **Optimal Transport (OT)** plan:

1. **Source distribution μ**:

   Treat each of the n original pairings as a discrete "pile of mass" on the space of neuron-pairing weight vectors, with total mass proportional to its importance score.

2. **Target distribution ν**:

   Choose m "target" neurons—either the same top-m survivors or m cluster-centroids of the original n—and assign each a probability mass (uniformly or again by importance).

3. **Cost matrix $C \in \mathbb{R}^{n \times m}$**:

   Quantify how "far apart" each source pairing is from each target, e.g. via normalized ℓ_1-distance of their concatenated weight vectors.

4. **Solve OT**

   $$T = \arg\min_{T \geq 0} \langle T,\, C \rangle_F \quad \text{s.t.} \quad T\,\mathbf{1}_m = \mu, \;\; T^\top \mathbf{1}_n = \nu.$$

   This yields an $n \times m$ transport plan T telling how to move mass from each of the n sources into the m targets.

5. **Fuse pairings**

   Use the columns of T as weights: each new "fused" pairing is a weighted sum of all n original pairings, according to the corresponding column of T. The result is a compressed group $\mathbb{G}_{\text{new}}$ of cardinality m that **recycles** information from all n original pairings instead of discarding $n - m$ of them.

Transportation cost

Design a metric that measures how similar/dissimilar neurons are.

Batch Normalization

A technique to stabilize and accelerate the training of deep neural networks by normalizing layer inputs.

- Motivation

During training, as parameters in earlier layers change, the distribution of inputs to later layers shifts, forcing those layers to constantly adapt. Batch normalization (BatchNorm) reduces this shift by keeping inputs to each layer more stable.

- The Forward Pass

  Given a mini-batch of activations $\{x^{(1)}, \ldots, x^{(m)}\}$ for a single neuron (or feature channel) in one layer:

  - **Compute batch statistics**

    $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}, \quad \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \mu_B\right)^2.$

  - **Normalize** each activation:

    $\hat{x}^{(i)} = \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}},$

    where ϵ is a small constant for numerical stability.

  - **Scale and shift**: introduce two learned parameters, γ (scale) and β (shift), to allow the network to recover the identity transform if needed:

    $y^{(i)} = \gamma\,\hat{x}^{(i)} + \beta.$

  The output $y^{(i)}$ is then passed on to the next layer.

- Inference Mode

  At test time, we don't have meaningful batch statistics, so BatchNorm uses running averages of μ_B and σ_B^2 that were accumulated during training. Concretely, during training we update

  $\text{running\_mean} \leftarrow \alpha\,\text{running\_mean} + (1-\alpha)\,\mu_B$
  $\text{running\_var} \leftarrow \alpha\,\text{running\_var} + (1-\alpha)\,\sigma_B^2,$

  and at inference we normalize each activation x as

  $\hat{x} = \frac{x - \text{running\_mean}}{\sqrt{\text{running\_var} + \epsilon}}, \quad y = \gamma\,\hat{x} + \beta.$

- Key Properties and Variants

- **Per-feature/channel normalization**: In convolutional layers, statistics are computed over the batch *and* spatial dimensions, separately for each channel.
- **Learnable parameters** (γ, β) let the layer choose any desired output scale and bias.

Use ℓ_1-distance between the neuron-pairings' weight vectors to define the cost C :

$C_{ij} = \frac{\| w^{(i)} - w^{(j)}_{\text{target}} \|_1}{(\text{normalizer})}.$

OT will preferentially fuse together neurons whose weight-patterns are more similar.

BatchNorm layers sit "in the middle" of the weights and activations, so if just grab the raw weight vectors and biases to build the cost matrix C_{ij}, we are ignoring the fact that every forward pass actually does

$z = w\,x + b, \quad y = \mathrm{BN}(z) \;=\; \gamma\,\frac{z-\mu}{\sqrt{\sigma+\varepsilon}} + \beta.$

The authors "fold" a BatchNorm layer into the preceding linear (or convolutional) layer so that they no longer need a separate BN at inference time. Concretely, suppose the layer originally computes

$z = w\,x + b,$

and then we apply BatchNorm:

$\mathrm{BN}(z) = \gamma\,\frac{z-\mu}{\sqrt{\sigma+\varepsilon}} + \beta,$

where

- w is the original weight vector,
- b is its bias,
- μ and σ are the running mean and variance learned by BatchNorm,
- γ and β are the BN's learned scale and shift,
- ε is a small constant for numerical stability.

We want a single "folded" layer

$z_{\text{new}} = w_{\text{new}}\,x + b_{\text{new}}$

that exactly replicates $\mathrm{BN}(z)$. Plugging in:

$$\mathrm{BN}(z) = \gamma\,\tfrac{w\,x+b-\mu}{\sqrt{\sigma+\varepsilon}} \;+\; \beta = \left(\tfrac{\gamma}{\sqrt{\sigma+\varepsilon}}\,w\right) x \;+\; \left(\tfrac{\gamma}{\sqrt{\sigma+\varepsilon}}(b-\mu) \;+\; \beta\right).$$

Thus we set

$$w_{\text{new}} \;=\; \tfrac{\gamma}{\sqrt{\sigma+\varepsilon}}\,w, \qquad b_{\text{new}} \;=\; \tfrac{\gamma}{\sqrt{\sigma+\varepsilon}}\,(b-\mu) \;+\; \beta.$$

Now BN is absorbed into the weights and bias.

The authors set $\varepsilon = 0$, then

$$w_{\text{new}} = \tfrac{w\times\gamma}{\sqrt{\sigma}}, \quad b_{\text{new}} = \tfrac{(b-\mu)\times\gamma}{\sqrt{\sigma}} + \beta$$

Probability Distribution

- **Uniform distribution.**
  - Every neuron pair receives exactly the same probability mass.
  - If there are N total pairs, each pair has probability $1/N$.
  - This approach makes no use of the underlying importance values: it treats all pairs as equally likely.
- **Importance-informed distribution.**
  - Each pair's probability is proportional to its importance score.
  - Intuitively, neuron pairs deemed more "important" are sampled more often.
  - To convert raw importance values I_k into probabilities p_k, we need a normalizing step:
    - **Simple normalization:**

      $p_k = \frac{I_k}{\sum_j I_j}$
    - **Softmax normalization:**

      $p_k = \frac{\exp(I_k)}{\sum_j \exp(I_j)}$.
    - Both ensure that all p_k sum to 1

There is no final accuracy difference however we choose the distribution type according to the authors.

Given the cost matrix C, and our probability distributions μ
and
ν, we can finally derive the optimal transport map T.

Consequently, we proceed to traverse each layer within the group denoted as $\mathbb{G}$, and rather than removing neurons of diminished significance, we opt to generate fused neurons through a process of matrix multiplication with the transport map T
.

Empirical Results

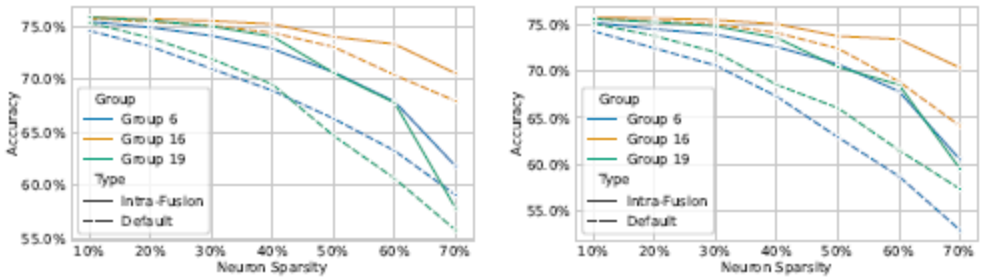


Figure 3: Data-Free Pruning: ResNet50 on ImageNet. ℓ_1 (left), Taylor (right).

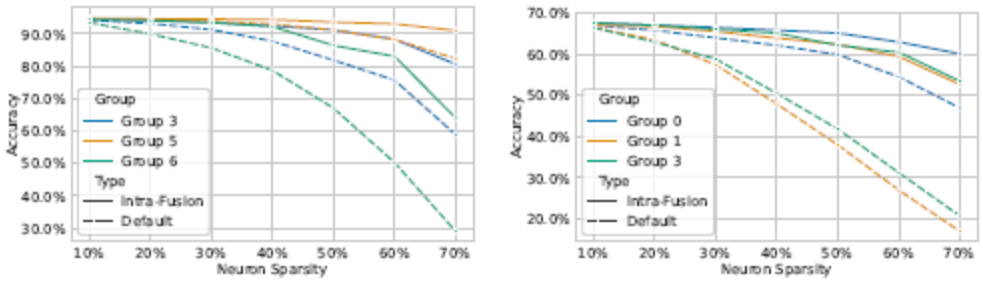


Figure 4: Data-Free Pruning: ResNet18 on CIFAR-10 and VGG11-BN on CIFAR-100, ℓ_1.

Figure 3: ResNet-50 on ImageNet

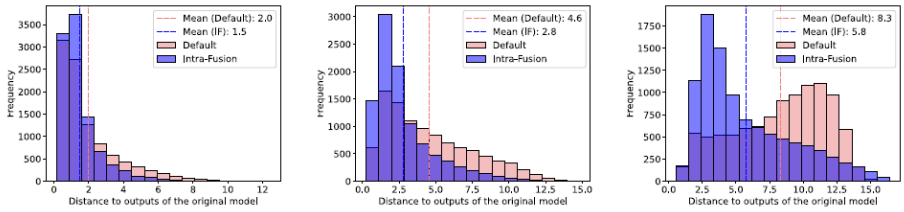
- **Left panel (ℓ_1 importance)** vs. **Right panel (Taylor importance)**.
- In both cases, as you increase sparsity, accuracy falls—but the **Intra-Fusion** curves always lie above their **Default** counterparts.

Figure 4: ResNet-18 on CIFAR-10 & VGG11-BN on CIFAR-100

- Both networks show the same qualitative trend:

Intra-Fusion (solid) preserves significantly more accuracy than **Default** (dashed) at every sparsity level.

5.1 OUTPUT PRESERVATION



(a) Sparsity: 10% (b) Sparsity: 20% (c) Sparsity: 30%

Figure 5: Output preservation comparison of the original model. Model: ResNet18. Dataset: CIFAR-10. Group: 0. Importance metric: ℓ_1.

Figure 5 directly measures how "close" each pruned model's outputs remain to the un-pruned ResNet-18 (on CIFAR-10), by computing the ℓ_2 distance between their final logits for a large batch of inputs. Each subplot is a histogram of those distances under two schemes—Default pruning (red) vs. Intra-Fusion (blue)—at different neuron-sparsity levels:

- **(a) 10 % sparsity**
  - Default: mean ℓ_2-distance ≈ 2.0
  - Intra-Fusion: mean ≈ 1.5
  - Blue bars are more tightly clustered near zero, so most outputs stay very close to the original.
- **(b) 20 % sparsity**
  - Default: mean ≈ 4.6
  - Intra-Fusion: mean ≈ 2.8
  - Again, Intra-Fusion's distribution is shifted significantly toward smaller deviations.
- **(c) 30 % sparsity**
  - Default: mean ≈ 8.3
  - Intra-Fusion: mean ≈ 5.8

- At higher sparsity the gap widens—the blue histogram remains far more concentrated at low distances, whereas default pruning spills out to much larger errors.

Factorizing Model Training

Split-Data Training Protocols

- **Data Partitioning:** The full training set is split into two (or k) disjoint subsets of equal size.
- **Independent Submodel Training:** Two (or k) network copies are trained **to convergence**—each on its own data split.
- **Prune-then-Fuse (PaF):**
  1. Each submodel is first pruned via Intra-Fusion (OT-based meta-pruning) on its split.
  2. The pruned weights are then merged via Optimal Transport into one final model.
- **Fuse-then-Prune (FaP):**
  1. Unpruned submodels are first OT-fused into a single full-size network.
  2. That fused network is then pruned (and optionally lightly fine-tuned).
- **One-Shot Aggregation:** Unlike data-parallelism's per-batch gradient synchronization, Split-Data only communicates final weight sets once, dramatically reducing communication overhead and latency sensitivity.