# Pruning-Aware Merging for Efficient Multitask Inference

Xiaoxi He
ETH Zürich
Zürich, Switzerland
hex@ethz.ch

Dawei Gao
SKLSDE & BDBC, Beihang University
Beijing, China
david_gao@buaa.edu.cn

Zimu Zhou
Singapore Management University
Singapore, Singapore
zimuzhou@smu.edu.sg

Yongxin Tong
SKLSDE & BDBC, Beihang University
Beijing, China
yxtong@buaa.edu.cn

Lothar Thiele
ETH Zürich
Zürich, Switzerland
thiele@ethz.ch

## ABSTRACT

Many mobile applications demand selective execution of multiple correlated deep learning inference tasks on resource-constrained platforms. Given a set of deep neural networks, each pre-trained for a single task, it is desired that executing arbitrary combinations of tasks yields minimal computation cost. Pruning each network separately yields suboptimal computation cost due to task relatedness. A promising remedy is to merge the networks into a multitask network to eliminate redundancy across tasks before network pruning. However, pruning a multitask network combined by existing network merging schemes cannot minimise the computation cost of every task combination because they do not consider such a future pruning. To this end, we theoretically identify the conditions such that pruning a multitask network minimises the computation of all task combinations. On this basis, we propose Pruning-Aware Merging (PAM), a heuristic network merging scheme to construct a multitask network that approximates these conditions. The merged network is then ready to be further pruned by existing network pruning methods. Evaluations with different pruning schemes, datasets, and network architectures show that PAM achieves up to 4.87× less computation against the baseline without network merging, and up to 2.01× less computation against the baseline with a state-of-the-art network merging scheme.

## CCS CONCEPTS

• **Computing methodologies** → *Neural networks.*

## KEYWORDS

Deep Learning; Network Pruning; Multitask Inference

**ACM Reference Format:**
Xiaoxi He, Dawei Gao, Zimu Zhou, Yongxin Tong, and Lothar Thiele. 2021. Pruning-Aware Merging for Efficient Multitask Inference. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467271


*KDD '21, August 14–18, 2021, Virtual Event, Singapore*

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467271

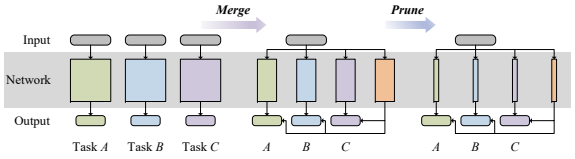


**Figure 1: Efficient multitask inference by "merge & prune". Three networks pre-trained for tasks $A$, $B$ and $C$ are first merged into a multitask network and then pruned.**

## 1 INTRODUCTION

Deep neural networks that can run locally on resource-constrained devices hold potential for various emerging applications such as autonomous drones and social robots [7, 18]. These applications often simultaneously perform a set of correlated inference tasks based on the current context to deliver accurate and adaptive services. Although deep neural networks pre-trained for individual tasks are readily available [17, 24], deploying multiple such networks easily overwhelms the resource budget.

To support these applications on low-resource platforms, we investigate *efficient multitask inference*. Given a set of correlated inference tasks and deep neural networks (each network pre-trained for an individual task), we aim to minimise the computation cost when **any subset of tasks** is performed at inference time.

One naive solution to efficient multitask inference is to *prune* each network for individual tasks *separately*. A deep neural network is typically over-parameterised [5]. Network pruning [3, 4, 8, 21, 25] can radically reduce the number of operations within a network without accuracy loss in the inference task. This solution, however, is only optimal if *a single task* is executed at a time. When multiple correlated tasks are running concurrently, this solution is unable to save computation cost by exploiting tasks relatedness and sharing intermediate results among networks.

A more promising solution framework is *"merge & prune"*, which merges multiple networks into a *multitask network*, before pruning it (Fig. 1). A few pioneer studies [2, 13] have explored network merging schemes to eliminate the redundancy among multiple networks pre-trained for correlated tasks. However, pruning a multitask network merged via these schemes can only minimise computation cost when *all tasks* are executed at the same time.

In this paper, we propose Pruning-Aware Merging (PAM), a new network merging scheme for efficient multitask inference. By applying existing network pruning methods on the multitask network merged by PAM, the computation cost when performing *any subset* of tasks can be reduced. Extensive experiments show that "PAM & Prune" consistently achieves solid advantages over the state-of-the-art network merging scheme across tasks, datasets, network architectures and pruning methods.

Our main contributions and results are as follows:

- We theoretically show that pruning a multitask network may not simultaneously minimise the computation cost of all task combinations in the network. We then identify conditions such that minimising the computation of all task combinations via network pruning becomes feasible. To the best of our knowledge, this is the first explicit analysis on the applicability of network pruning in multitask networks.
- We propose Pruning-Aware Merging (PAM), a heuristic network merging scheme to construct a multitask network that approximately meets the conditions in our analysis and enables "merge & prune" for efficient multitask inference.
- We evaluate PAM with various pruning schemes, datasets and architectures. PAM achieves up to 4.87× less computation cost against the baseline without network merging, and up to 2.01× less computation cost against the baseline with the state-of-the-art network merging scheme [13].

In the rest of this paper, we review related work in Sec. 2, introduce our problem statement in Sec. 3, theoretical analysis in Sec. 4 and our solution in Sec. 5. We present the evaluations of our methods in Sec. 6 and finally conclude in Sec. 7.

## 2 RELATED WORK

Our work is related to the following categories of research.

**Network Pruning.** Network pruning reduces the number of operations in a deep neural network without loss in accuracy [4, 25]. Unstructured pruning removes unimportant weights [6, 8, 9]. However, customised hardware [10] is compulsory to exploit such irregular sparse connections for acceleration. Structured pruning enforces sparsity at the granularity of channels/filters/neurons [3, 19, 21, 27]. The resulting sparsity is fit for acceleration on general-purpose processors. Prior pruning proposals implicitly assume a single task in the given network. We identify the challenges to prune a multitask network and propose a network merging scheme such that pruning the merged multitask network minimises computation cost of all task combinations in the network.

**Multitask Networks.** A multitask network can be either constructed from scratch via Multi-Task Learning (MTL) or merged from multiple networks pre-trained for individual tasks. MTL joint trains multiple tasks for better generalisation [29], while we focus on the computation cost of running multiple tasks at inference time. Network merging schemes [2, 13] aim to construct a compact multitask network from networks pre-trained for individual tasks. Both MTZ [13] and NeuralMerger [2] enforce weight sharing among networks to reduce their overall storage. In contrast, we account for the computation cost of a multitask network. Although constructing a multitask network using these schemes [2, 13] and pruning it via existing pruning methods can reduce the computation when *all tasks* are concurrently executed, they cannot minimise the computation cost for *every combination of tasks*.

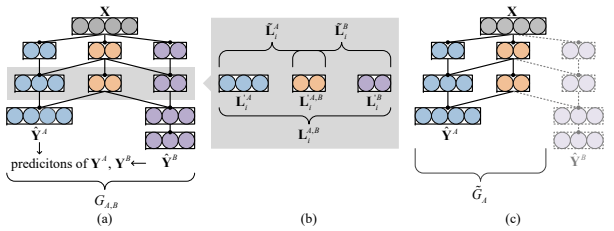


**Figure 2: Important notations: (a) graph representation $G_{A,B}$ of a multitask network for tasks $A$ and $B$, with $N_A = 2$ hidden layers for task $A$ and $N_B = 3$ hidden layers for task $B$; (b) layer outputs for the $i$-th layer; (c) subgraph $\widetilde{G}_A$ for task $A$.**

## 3 PROBLEM STATEMENT

We define and analyse our problem based on the graph representation of neural networks. The graph representation reflects the computation cost of neural networks (see below) and facilitates an information theoretical understanding on network pruning (see Sec. 4). Fig. 2 shows important notations used throughout this paper. For ease of illustration, we explain our analysis using two tasks. Extensions to more than two tasks are in Sec. 5.4.

### 3.1 Graph Representation of Neural Networks

**Task.** Consider three sets of random variable $\mathbf{X} \in \mathcal{X}$, $\mathbf{Y}^A \in \mathcal{Y}^A$, and $\mathbf{Y}^B \in \mathcal{Y}^B$. Task $A$ outputs $\widehat{\mathbf{Y}}^A$, a prediction of $\mathbf{Y}^A$, by learning the conditional distribution $\Pr\{\mathbf{Y}^A = \mathbf{y}|\mathbf{X} = \mathbf{x}\}$. Task $B$ outputs $\widehat{\mathbf{Y}}^B$, a prediction of $\mathbf{Y}^B$, by learning $\Pr\{\mathbf{Y}^B = \mathbf{y}|\mathbf{X} = \mathbf{x}\}$.

**Single-Task Network.** For task $A$, a neural network without feedback loops can be represented by an acyclic directed graph $G_A = \{V^A, E^A\}$. Each vertex represents a neuron. There is an edge between two vertices if two neurons are connected. The vertex set $V_A$ can be categorised into three types of nodes: source, internal and sink node. $\deg^-(v)/\deg^+(v)$ is the indegree/outdegree of a vertex $v$.

- Source node set $\mathbf{v}_X^A = \{v|v \in V^A \wedge \deg^-(v) = 0\}$ represents the *input layer*. Each source node represents an *input neuron* and outputs a random variable $X_i \in \mathbf{X}$. The output of the input layer is the input random variable set $\mathbf{X}$.
- Internal nodes $v_i \in \{v|v \in V \wedge \deg^-(v) \neq 0 \wedge \deg^+(v) \neq 0\}$ represents the *hidden neurons*. The output of each hidden neuron is generated by calculating the weighted sum of its inputs and then applying an activation function.
- Sink node set $\mathbf{v}_Y^A = \{v|v \in V \wedge \deg^+(v) = 0\}$ represents the *output layer*. Each sink node represents an *output neuron* and the output is calculated in the same way as the hidden neurons. The output of the output layer is the prediction $\widehat{\mathbf{Y}}^A$ of ground-truth labels $\mathbf{Y}^A$.

We organise the hidden neurons $v_i$ of $G^A$ into layers $\mathbf{v}_i^A$ by Algorithm 1. $N^+(\mathbf{v})$ represents the out-coming neighbours of the vertex set $\mathbf{v}$. Algorithm 1 can organise any acyclic single-task network into layers and the layer outputs satisfy the Markov property.

**Algorithm 1:** Organise vertices in the graph representation of a neural network into layers.

**Input:** A neural network graph $G^A$
**Output:** $N+1$ layers $\mathbf{v}_i^A$ with $i = 1, \cdots, N+1$.

$\mathbf{v}_0^A \leftarrow \mathbf{v}_X^A$;
$i \leftarrow 0$;
**while** $N^+(\mathbf{v}_i^A) \neq \mathbf{v}_Y^A$ **do**
  $\mathbf{v}_{i+1}^A \leftarrow \emptyset$;
  **for** *each node* $v_{i,j}^A \in \mathbf{v}_i^A$ **do**
    **if** $N^+(v_{i,j}^A) \cap \mathbf{v}_Y^A \neq \emptyset$ **then**
      $\mathbf{v}_{i+1}^A \leftarrow \mathbf{v}_{i+1}^A \cup \{v_{i,j}^A\}$;
    **end**
  **end**
  $\mathbf{v}_{i+1}^A \leftarrow \mathbf{v}_{i+1}^A \cup \left(N^+(\mathbf{v}_i^A) \setminus \mathbf{v}_i^A\right)$;
  $i \leftarrow i+1$;
**end**
$N \leftarrow i$;
$\mathbf{v}_{N+1}^A \leftarrow \mathbf{v}_Y^A$;

**Multitask Network.** For task $A$ and $B$, a multitask network without feedback loops can be represented by an acyclic directed graph $G_{A,B}$. All paths from the input neurons to the output neurons for task $A$ form a subgraph $\widetilde{G}_A$ (see Fig. 2(c)), which is in effect the same as a single-task network. When only task $A$ is performed, only $\widetilde{G}_A$ is activated. Subgraph $\widetilde{G}_B$ is defined similarly. We also organise vertices of $\widetilde{G}_A$ and $\widetilde{G}_B$ into layers with Algorithm 1. Layer outputs of $\widetilde{G}_A$ and $\widetilde{G}_B$ are denoted as $\widetilde{\mathbf{L}}_i^A$ and $\widetilde{\mathbf{L}}_i^B$. Suppose $\widetilde{G}_A$ and $\widetilde{G}_B$ have respectively $N_A$ and $N_B$ hidden layers. We assume $N_A \leq N_B$ w.l.o.g.. Then the $i$-th layer output of $G_{A,B}$ is defined as $\mathbf{L}_i^{A,B} = \widetilde{\mathbf{L}}_i^A \cup \widetilde{\mathbf{L}}_i^B$ with $i = 0, \cdots, N_A$. As shown in Fig. 2(b), $\mathbf{L}_i^{A,B}$ consists of three sets of neurons: $\mathbf{L}_i'^A$, $\mathbf{L}_i'^B$ and $\mathbf{L}_i'^{A,B}$.

**Remarks.** The above definitions have two benefits. *(i)* The computation cost of a neural network is an *increasing function* of the size of the graph, *i.e.*, the number of edges plus vertices. Reducing the computation cost of the network is transformed into removing edges or vertices in the graph. *(ii)* For a single-task network with $N_A$ hidden layers, its layer outputs form a Markov chain: $\mathbf{Y}^A \rightarrow \mathbf{L}_0^A \rightarrow \cdots \rightarrow \mathbf{L}_{N_A+1}^A$. All layer outputs $\mathbf{L}_i^{A,B}$ in a multitask network also form a Markov chain. The Markov property allows an information theoretical analysis on neural networks [23, 26].

## 3.2 Problem Definition

Given two single-task networks $G_A$ and $G_B$ pre-trained for task $A$ and $B$, we aim to construct a multitask network $G_{A,B}$ such that pruning on $G_{A,B}$ can minimise the number of vertices and edges in $G_{A,B}$, $\widetilde{G}_A$ and $\widetilde{G}_B$ while preserving inference accuracy on $A$ and $B$. To ensure minimal computation of *any subset* of tasks, we need to minimise the number of vertices and edges in *any subgraph*. For two tasks, $G_{A,B}$ corresponds to running task $A$ and $B$ concurrently; $\widetilde{G}_A$ ($\widetilde{G}_B$) corresponds to running task $A$ ($B$) only. Next, we show the difficulty to optimise all subgraphs simultaneously.

# 4 THEORETICAL UNDERSTANDING

This section presents a theoretical understanding on the challenges to prune a multitask network and identifies conditions such that minimising the computation cost of all task combinations via pruning becomes feasible (Theorem 1). Proofs are in Appendix A.

## 4.1 Why Pruning a Single-task Network Work

Pruning a single-task network reduces the computation cost of a neural network while retaining task inference accuracy by suppressing *redundancy* in the network [4, 25]. From the information theoretical perspective [23, 26], since the layer outputs form a Markov chain, the inference accuracy for a given task $A$ is positively correlated to the task related information transmitted through the network at each layer, measured by $I(\mathbf{L}_i^A; \mathbf{Y}^A)$. All other information is irrelevant for the task. Hence *the redundancy within a single-task network* can be defined as below.

**Definition 1.** For the $i$-th layer in the single-task neural network $G_A$, the redundancy of the layer is defined as $\mathcal{R}_A(\mathbf{L}_i^A) = \sum_{L_{i,j}^A \in \mathbf{L}_i^A} H(L_{i,j}^A) - I(\mathbf{L}_i^A; \mathbf{Y}^A)$.

$\sum_{L_{i,j}^A \in \mathbf{L}_i^A} H(L_{i,j}^A)$ measures *the maximal amount of information* the layer can express. $I(\mathbf{L}_i^A; \mathbf{Y}^A)$ measures *the amount of task A related information* in the layer output. By definition, $\mathcal{R}_A(\mathbf{L}_i^A) \geq 0$.

**Remarks.** $\sum_{L_{i,j}^A \in \mathbf{L}_i^A} H(L_{i,j}^A)$ is positively correlated to the number of vertices and incoming edges of the $i$-th layer. Therefore, in a well trained network where $I(\mathbf{L}_i^A; \mathbf{Y}^A)$ can no longer increase, the computation cost can be minimised by reducing $\mathcal{R}_A(\mathbf{L}_i^A)$.

Accordingly, pruning a single-task network can be formalised as an optimisation problem

$$\text{minimise} \sum_{i=1}^{N_A+1} \left(\mathcal{R}_A(\mathbf{L}_i^A) - \xi_i \cdot I(\mathbf{L}_i^A; \mathbf{Y}^A)\right) \tag{1}$$

where $\xi_i > 0$ controls the trade-off between inference accuracy and computation cost.

**Remarks.** Existing pruning methods implicitly assume a single-task network. That is, they are all designed to solve optimisation problem (1), even though the concrete strategies vary. We now show the problems that occur when these pruning methods are applied to a multitask network.

## 4.2 Why Pruning a Multitask Network Fail

As mentioned in Sec. 3.2, we aim to minimise the computation cost of any subset of tasks, which is a *multi-objective* optimisation problem. As we will show below, existing network pruning methods are unable to handle these objectives simultaneously.

We first define redundancy when performing two tasks at the same time, similarly as in Definition 1.

**Definition 2.** For a multitask network $G_{A,B}$, the redundancy of its $i$-th layer is $\mathcal{R}_{A,B}(\mathbf{L}_i^{A,B}) = \sum_{L_{i,j}^{A,B} \in \mathbf{L}_i^{A,B}} H(L_{i,j}^{A,B}) - I(\mathbf{L}_i^{A,B}; \mathbf{Y}^A, \mathbf{Y}^B)$.

Following the above definitions of redundancy, our objective in Sec. 3.2 is equivalent to minimising the redundancy in $G_{A,B}$ as well as in its two subgraphs $\widetilde{G}_A$ and $\widetilde{G}_B$, which leads to the following

three-objective optimisation (still, we assume $N_A \leq N_B$ w.l.o.g.):

$$\begin{aligned} &\text{minimise} \\ &\sum_{i=1}^{N_A+1} \left(\mathcal{R}_A(\widetilde{\mathbf{L}}_i^A) - \tilde{\xi}_i^A \cdot I(\widetilde{\mathbf{L}}_i^A; \mathbf{Y}^A)\right), \\ &\sum_{i=1}^{N_B+1} \left(\mathcal{R}_B(\widetilde{\mathbf{L}}_i^B) - \tilde{\xi}_i^B \cdot I(\widetilde{\mathbf{L}}_i^B; \mathbf{Y}^B)\right), \\ &\sum_{i=1}^{N_A} \left(\mathcal{R}_{A,B}(\mathbf{L}_i^{A,B}) - \xi_i^A \cdot I(\widetilde{\mathbf{L}}_i^A; \mathbf{Y}^A) - \xi_i^B \cdot I(\widetilde{\mathbf{L}}_i^B; \mathbf{Y}^B)\right) \end{aligned} \tag{2}$$

Reducing $\mathcal{R}_A(\widetilde{\mathbf{L}}_i^A)$, $\mathcal{R}_B(\widetilde{\mathbf{L}}_i^B)$ and $\mathcal{R}_{A,B}(\mathbf{L}_i^{A,B})$ decreases the number of vertices and edges in $\widetilde{G}_A$, $\widetilde{G}_B$ and $G_{A,B}$, respectively. $\xi_i^A, \xi_i^B, \tilde{\xi}_i^A, \tilde{\xi}_i^B > 0$ are parameters to control the trade-off between computation cost and inference accuracy, as well as to balance task $A$ and $B$.

To solve optimisation problem (2) with prior network pruning methods, we observe two problems.

**Problem 1: The first two objectives in** (2) **may conflict.** This is because reducing $\mathcal{R}_B(\widetilde{\mathbf{L}}_i^B)$ may decrease $I(\widetilde{\mathbf{L}}_i^A; \mathbf{Y}^A)$ (proofs in Appendix A.1). In other words, when pruning subgraph $\widetilde{G}_B$, it is possible that some information related to task A is removed from the shared vertices between $\widetilde{G}_A$ and $\widetilde{G}_B$. Hence $I(\widetilde{\mathbf{L}}_i^A; \mathbf{Y}^A)$ decreases and the inference accuracy of task $A$ deteriorates.

**Problem 2: It is unclear how to minimise the third objective in** (2)**.** As mentioned in Sec. 4.1, most pruning methods are designed with a single-task network in mind. It is unknown how to apply them to a multitask network $G_{A,B}$ with architecture in Fig. 2 (a).

### 4.3 When Pruning a Multitask Network Work

The two problems in Sec. 4.2 show that not all multitask networks can be pruned for efficient multitask inference. However, a multitask network can be effectively pruned if it meets the conditions stated by the following theorem.

**Theorem 1.** If $\forall\, 1 \leq i \leq N_A$, the conditions below are satisfied:

$$\begin{aligned} I(\mathbf{L}_i'^A; \mathbf{L}_i'^B; \mathbf{Y}^A; \mathbf{Y}^B) &= 0 \\ I(\mathbf{L}_i'^{A,B}; \mathbf{Y}^A | \mathbf{L}_i'^A, \mathbf{Y}^B) &= 0 \\ I(\mathbf{L}_i'^{A,B}; \mathbf{Y}^B | \mathbf{L}_i'^B, \mathbf{Y}^A) &= 0 \end{aligned} \tag{3}$$

where $I(\mathbf{L}_i'^A; \mathbf{L}_i'^B; \mathbf{Y}^A; \mathbf{Y}^B)$ is the *co-information* [1], then the *three-objective* optimisation problem (2) can be reduced to *two* non-conflicting optimisation problems that can be solved *independently*:

$$\begin{aligned} &\text{minimise} \sum_{i=1}^{N_A+1} \mathcal{R}_A(\widetilde{\mathbf{L}}_i^A) - \tilde{\xi}_i^A \cdot I(\widetilde{\mathbf{L}}_i^A; \mathbf{Y}^A), \\ &\text{minimise} \sum_{i=1}^{N_B+1} \mathcal{R}_B(\widetilde{\mathbf{L}}_i^B) - \tilde{\xi}_i^B \cdot I(\widetilde{\mathbf{L}}_i^B; \mathbf{Y}^B) \end{aligned} \tag{4}$$

Each of the two optimisation problems (4) are in effect single-task pruning problem like optimisation problem (1), which can be effectively solved by prior pruning proposals.

**Remarks.** Theorem 1 provides important guidelines to design the network merging scheme for our problem in Sec. 3.2. Specifically, if $G_A$ and $G_B$ can be merged into a a multitask network $G_{A,B}$ such that conditions (3) are satisfied, we can simply apply existing network pruning on the two subgraphs $\widetilde{G}_A$ and $\widetilde{G}_B$ to minimise the computation cost when performing any subset of tasks.

## 5 PRUNING-AWARE MERGING

Based on the above analysis, we propose Pruning-Aware Merging (PAM), a novel network merging scheme that constructs a multitask network from pre-trained single task networks. PAM approximately meets the conditions in Theorem 1 such that the merged multitask network can be effectively pruned for efficient multitask inference.

### 5.1 PAM Workflow

Given two single-task networks $G_A$ and $G_B$ pre-trained for task $A$ and $B$ ($N_A \leq N_B$), PAM constructs a multitask network $G_{A,B}$ with the steps below (see Fig. 3).

(1) Assign $\mathbf{L}_0^{A,B} = \mathbf{X}$, as $G_{A,B}$, $G_A$ and $G_B$ use the same inputs.
(2) For $i = 1, \cdots, N_A$, regroup the neurons from $\mathbf{L}_i^A$ and $\mathbf{L}_i^B$ into $\mathbf{L}_i'^A$, $\mathbf{L}_i'^B$ and $\mathbf{L}_i'^{A,B}$ by the regrouping algorithm in Sec. 5.2.
(3) Take over the output layer for task $A$: $\widetilde{\mathbf{L}}_{N_A+1}^A = \mathbf{L}_{N_A+1}^A$. For $i = N_A + 1, \cdots, N_B + 1$, take over the remaining layers from $G_B$: $\widetilde{\mathbf{L}}_i^B = \mathbf{L}_i^B$.
(4) Reconnect the neurons as in Fig. 3. If a connection exist before merging, it preserves its original weight. Otherwise it is initialised with a zero.
(5) Finetune $G_{A,B}$ on $A$ and $B$ to learn the newly added connections. For the shared connections, $\mathbf{L}_{i-1}'^{A,B} \rightarrow \mathbf{L}_i'^{A,B}$. The gradients are first calculated separately on $A$ and $B$, and then averaged before weight updating.

Now the multitask network $G_{A,B}$ is ready to be pruned. From Theorem 1, we can apply network pruning on the two subgraphs $\widetilde{G}_A$ and $\widetilde{G}_B$ independently and achieve a minimal computation cost for all combinations of tasks. However, since we only approximate the conditions in (3), pruning $\widetilde{G}_A$ and $\widetilde{G}_B$ is not perfectly independent in practice. Hence we prune $\widetilde{G}_A$ and $\widetilde{G}_B$ *in an alternating manner* to balance between task $A$ and $B$.

### 5.2 Regrouping Algorithm

The core of PAM is the regrouping algorithm in the second step in Sec. 5.1. It regroups the neurons from $\mathbf{L}_i^A$ and $\mathbf{L}_i^B$ into three sets: $\mathbf{L}_i'^A$, $\mathbf{L}_i'^B$ and $\mathbf{L}_i'^{A,B}$, such that the conditions (3) in Theorem 1 are satisfied. However, it is computation-intensive to estimate the co-information and conditional mutual information in (3) precisely. We rely on the following theorem to approximate the conditions.

**Theorem 2.** The conditions in (3) can be achieved by minimising $I(\mathbf{L}_i'^A; \mathbf{Y}^B)$, $I(\mathbf{L}_i'^B; \mathbf{Y}^A)$, and maximising $I(\mathbf{L}_i'^A; \mathbf{Y}^A)$, $I(\mathbf{L}_i'^B; \mathbf{Y}^B)$.

**Remarks.** $I(\mathbf{L}_i'^A; \mathbf{Y}^B)$ and $I(\mathbf{L}_i'^B; \mathbf{Y}^A)$ describe the "misplaced" information, i.e., the information that is useful for one task, but contained in neurons that are not connected to the outputs of this task. Therefore such information is redundant and needs to be minimised. $I(\mathbf{L}_i'^A; \mathbf{Y}^A)$ and $I(\mathbf{L}_i'^B; \mathbf{Y}^B)$ measure the "relevant" information, i.e., the information useful for one task and contained in neurons connected to this task. Note that this information may not be simply maximised, because it includes the information that is useful for

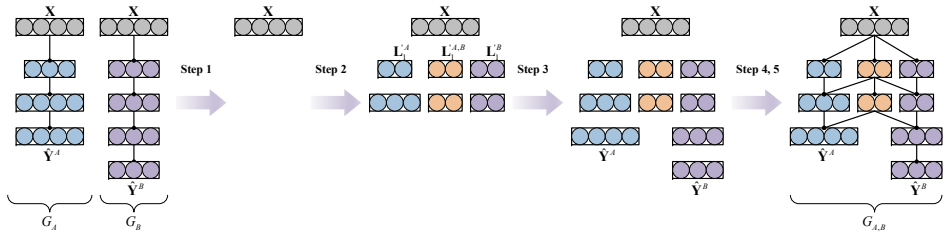


**Figure 3: PAM workflow to construct a multitask network ($G_{A,B}$) from two single-task networks ($G_A$ and $G_B$).**

**Algorithm 2:** Regroup algorithm.

**Input:** $\mathbf{L}_i^A$, $\mathbf{L}_i^B$, $\mathbf{X}$, $\mathbf{Y}^A$, $\mathbf{Y}^B$, $\alpha$
**Output:** $\mathbf{L}_i'^A$, $\mathbf{L}_i'^B$, $\mathbf{L}_i'^{A,B}$

1 $N = \min\{N^A, N^B\}$;
2 **for** $i \leftarrow 1$ **to** $N$ **do**
3 $\quad \mathbf{F}^A \leftarrow \mathbf{F}^B \leftarrow \mathbf{L}_i^A \cup \mathbf{L}_i^B$;
4 $\quad \mathbf{L}_i'^A \leftarrow \emptyset$;
5 $\quad$ **while** $I(\mathbf{L}_i'^A; \mathbf{Y}^B) \leq \alpha$ **do**
6 $\quad\quad L_{i,\cdot} \leftarrow \arg\min_{L_{i,j} \in \mathbf{F}_i^A} I(\{L_{i,j}\} \cup \mathbf{L}_i'^A; \mathbf{Y}^B)$;
7 $\quad\quad$ move the neuron $L_{i,\cdot}$ from $\mathbf{F}^A$ to $\mathbf{L}_i'^A$
8 $\quad$ **end**
9 $\quad \mathbf{L}_i'^B \leftarrow \emptyset$ ;
10 $\quad$ **while** $I(\mathbf{L}_i'^B; \mathbf{Y}^A) \leq \alpha$ **do**
11 $\quad\quad L_{i,\cdot} \leftarrow \arg\min_{L_{i,j} \in \mathbf{F}_i^B} I(\{L_{i,j}\} \cup \mathbf{L}_i'^B; \mathbf{Y}^A)$;
12 $\quad\quad$ move the neuron $L_{i,\cdot}$ from $\mathbf{F}^B$ to $\mathbf{L}_i'^B$
13 $\quad$ **end**
14 $\quad$ The remaining neurons join $\mathbf{L}_i'^{A,B}$:
$\quad\quad \mathbf{L}_i'^{A,B} \leftarrow \mathbf{L}_i^A \cup \mathbf{L}_i^B \setminus \left(\mathbf{L}_i'^A \cup \mathbf{L}_i'^B\right)$;
15 $\quad$ If a neuron exists in both $\mathbf{L}_i'^A$ and $\mathbf{L}_i'^B$, remove the neuron from them both.
16 **end**

both tasks. It requires simultaneously minimising the "misplaced" information and maximising the "correct" information to achieve the conditions in (3). The proof of Theorem 2 is in Sec. A.3.

Based on Theorem 2, we propose an algorithm to regroup the neurons such that conditions (3) are approximately met. It constructs the largest possible set $\mathbf{L}_i'^A$ and $\mathbf{L}_i'^B$ from all the neurons in $\mathbf{L}_i^A$ and $\mathbf{L}_i^B$ while $I(\mathbf{L}_i'^A; \mathbf{Y}^B)$ and $I(\mathbf{L}_i'^B; \mathbf{Y}^A)$ remain close to zero, such that $I(\mathbf{L}_i'^A; \mathbf{Y}^A)$ and $I(\mathbf{L}_i'^B; \mathbf{Y}^B)$ are approximately maximised. To estimate $I(\mathbf{L}_i'^A; \mathbf{Y}^B)$ and $I(\mathbf{L}_i'^B; \mathbf{Y}^A)$, we use a Kullback–Leibler-based mutual information upper bound estimator from [15].

Algorithm 2 illustrates the pseudocode to regroup the neurons such that the conditions in Theorem 1 are approximated met. Central in Algorithm 2 is a greedy search in Lines 5-8 and 10-13. In Lines 5-8, we search for the largest possible set of neuron $\mathbf{L}_i'^A$ while $I(\mathbf{L}_i'^A; \mathbf{Y}^B)$ remains approximately zero (smaller than a pre-defined threshold $\alpha$), such that $I(\mathbf{L}_i'^A; \mathbf{Y}^A)$ is approximately maximised. Similarly, in Lines 10-13, we approximately maximise $I(\mathbf{L}_i'^B; \mathbf{Y}^B)$ while keeping $I(\mathbf{L}_i'^B; \mathbf{Y}^A)$ close to zero. According to Theorem 2, the conditions in Theorem 1 are approximately met.

**Practical Issue: How to Estimate Mutual Information.** We use a Kullback–Leibler-based mutual information upper bound estimator from [15] to estimate the upper bounds of $I(\mathbf{L}_i'^A; \mathbf{Y}^B)$ and $I(\mathbf{L}_i'^B; \mathbf{Y}^A)$. Since the upper bounds are approximate, it is impossible to request them to be exactly zero. Hence, we use a threshold parameter $\alpha$ to keep $I(\mathbf{L}_i'^A; \mathbf{Y}^B)$ and $I(\mathbf{L}_i'^B; \mathbf{Y}^A)$ close to zero.

**Practical Issue: How to Tune Threshold $\alpha$.** The parameter $\alpha$ affects the performance of "PAM & prune". A larger $\alpha$ results in more neurons in $\mathbf{L}_i'^A$ and $\mathbf{L}_i'^B$ and fewer shared neurons in $\mathbf{L}_i'^{A,B}$. In this case, the multitask network after "PAM & prune" performs worse in terms of efficiency when both tasks are executed concurrently, but better when only one task is executed (similar to "baseline 1 & prune"). Conversely, a smaller $\alpha$ results in more shared neurons. In this case, the multitask network after "PAM & prune" performs worse when only one task is executed, but better when both tasks are executed concurrently, (similar to "baseline 2 & prune").

The parameter $\alpha$ can be empirically tuned as follows:

(1) Execute Algorithm 2 with a small $\alpha$.
(2) Increase the value of $\alpha$ slightly and rerun Algorithm 2. Since Lines 5-8 and 10-13 are greedy search, the results for the smaller $\alpha$ in Step 1 (*i.e.*, the already constructed neuron sets $\mathbf{L}_i'^A$ and $\mathbf{L}_i'^B$ ) can be reused, instead of starting with empty sets as in Line 4 and 9.
(3) Iterate Step 2 till a satisfying balance among task combinations. In each iteration of Step 2, we can reuse the neuron sets $\mathbf{L}_i'^A$ and $\mathbf{L}_i'^B$ from the last iteration.

The impact of $\alpha$ is shown in Appendix C.

## 5.3 Extensions to ResNets

In order to support merging Residual Networks [11], PAM needs to be slightly modified. As illustrated in Fig. 4, the regrouping of the last layer in each residual block happens not directly after the weighted summation, but after the superposition with the shortcut connection and just before the vector is passed as inputs to the first layer in the next block. This input vector of the first layer in each block is also regrouped using Algorithm 2 and then pruned at a later stage. This special treatment for the last layer in each residual block is consistent with ResNet compatible pruning methods such as [21], which can also prune the block outputs just before it is fed into the first layer in the next block.

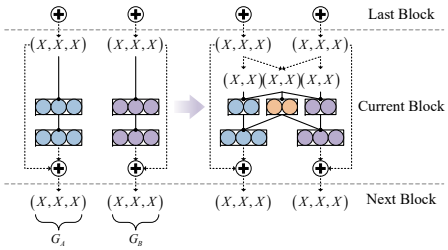


**Figure 4: Applying PAM on residual blocks. Vectors are denoted as $(X, \cdots, X)$. Dotted lines are identical connections, and firm lines represent weighted connections for neurons.**

## 5.4 Extension to Three or More Tasks

When there are $K \geq 3$ tasks, we define the set of all the task as $\upsilon = \{t_1, \cdots, t_K\}$. The merged multitask network can be divided into subgraphs $\widetilde{G}_\tau$, where $\tau \subseteq \upsilon$ and $\tau \neq \emptyset$ is a nonempty subset of tasks. Each vertex in $\widetilde{G}_\tau$ has paths to all the outputs $\widehat{\mathbf{Y}}^t$ with $t \in \tau$. When a task combination (*i.e.*, a subset of tasks) $\tau$ is executed, only subgraph $\widetilde{G}_\tau$ is activated. Layers in $\widetilde{G}_\tau$ is denoted as $\widetilde{\mathbf{L}}_i^\tau$. The output layer for task combination $\tau$ is denoted as $\widehat{\mathbf{Y}}^\tau = \bigcup_{t\in\tau} \widehat{\mathbf{Y}}^t$, which is the prediction of ground-truth labels $\mathbf{Y}^\tau = \bigcup_{t\in\tau} \mathbf{Y}^t$.

**Extension of Theorem 1.** For any pair of non-overlapped nonempty subsets of task $\tau_A$ and $\tau_B$ ($\tau_A \cap \tau_B = \emptyset$), define:

$$\mathbf{A}_i = \widetilde{\mathbf{L}}_i^{\tau_A} \setminus \widetilde{\mathbf{L}}_i^{\tau_B} \tag{5}$$

$$\mathbf{B}_i = \widetilde{\mathbf{L}}_i^{\tau_B} \setminus \widetilde{\mathbf{L}}_i^{\tau_A} \tag{6}$$

$$\mathbf{M}_i = \widetilde{\mathbf{L}}_i^{\tau_A} \cap \widetilde{\mathbf{L}}_i^{\tau_B} \tag{7}$$

Then Theorem 1 is extended into:

**Theorem 3.** If for all $i = 1, \cdots, N$ with $N = \min_{t\in\upsilon} N_t$, and for any pair of non-overlapped nonempty subsets of task $\tau_A$ and $\tau_B$, the following conditions are satisfied:

$$\begin{aligned} I(\mathbf{A}_i; \mathbf{B}_i; \mathbf{Y}^{\tau_A}; \mathbf{Y}^{\tau_B}) &= 0 \\ I(\mathbf{M}_i; \mathbf{Y}^{\tau_A} | \mathbf{A}_i, \mathbf{Y}^{\tau_B}) &= 0 \\ I(\mathbf{M}_i; \mathbf{Y}^{\tau_B} | \mathbf{B}_i, \mathbf{Y}^{\tau_A}) &= 0 \end{aligned} \tag{8}$$

then the computation cost of executing all task combinations can be minimised by the following $K$ non-conflicting optimisation problems that can be solved independently:

$$\text{For every } t \in \upsilon\text{: minimise } \sum_{i=1}^{N+1} \mathcal{R}_t(\widetilde{\mathbf{L}}_i^t) - \tilde{\xi}_i^t \cdot I(\widetilde{\mathbf{L}}_i^t; \mathbf{Y}^t) \tag{9}$$

Theorem 3 can be proven by recursively applying Theorem 1.

**Extension of PAM.** The neuron sets $\mathbf{L}_i'^A$, $\mathbf{L}_i'^B$ and $\mathbf{L}_i'^{A,B}$ are extended to:

$$\mathbf{L}_i'^\tau = \bigcap_{t\in\tau} \widetilde{\mathbf{L}}_i^t \setminus \bigcup_{t\notin\tau} \widetilde{\mathbf{L}}_i^t \tag{10}$$

Note that neurons in $\mathbf{L}_i'^\tau$ are activated iff any task $t \in \tau$ is executed. Now Algorithm 2 is extended to Algorithm 3. And at step 5 of the PAM workflow in Sec. 5.1, we connect $\mathbf{L}_{i-1}'^{\tau_1} \to \mathbf{L}_i'^{\tau_2}$ iff $\tau_2 \subseteq \tau_1$.

It is worth mentioning that when tasks are highly related, the numbers of neurons in $\mathbf{L}_i^\tau$ with $1 < |\tau| < K$ can be extremely small (as in our experiment on the LFW dataset in Appendix B).

**Algorithm 3:** Extending Algorithm 2 to over two tasks

**Input:** $\mathbf{X}$, $\alpha$, $\mathbf{L}_i^t$, and $\mathbf{Y}^t$ for all $t \in \upsilon$
**Output:** $\mathbf{L}_i'^\tau$ for all $\tau \subseteq \upsilon$ and $\tau \neq \emptyset$

```
N ← min_{t∈υ} N_t;
K ← |υ|;
for i ← 1 to N do
    S ← ⋃_{t∈υ} L_i^t;
    for n ← 1 to K − 1 do
        for any τ with |τ| = n do
            F ← S;
            L_i'^τ ← ∅;
            Y^{∉τ} ← ⋃_{t∉τ} Y^t
            while I(L_i'^τ; Y^{∉τ}) ≤ α do
                L_{i,·} ← arg min_{L_{i,j}∈F} I({L_{i,j}} ∪ L_i'^τ; Y^{∉τ})
                move the neuron L_{i,·} from F to L_i'^τ
            end
        end
        Remove all selected neurons from S:
          S ← S \ ⋃_{|τ|=n} L_i'^τ
        Among all L_i'^τ, if a neuron exists in more than one
          set, remove the neuron from them all
    end
    L_i'^υ ← S
end
```

Therefore we can simplify Algorithm 3 by fixing $n = 1$ and skip the remaining loops. Every layer in the multitask network merged by the simplified PAM contains only neuron sets $\mathbf{L}_i^t$ with $t \in \upsilon$ and one shared neuron set $\mathbf{L}_i^\upsilon$. Shared neurons in $\mathbf{L}_i^\upsilon$ are always activated, while non-shared neurons in $\mathbf{L}_i^t$ are activated iff task $t$ is executed.

# 6 EXPERIMENTS

We compare different network merging schemes on whether lower computation is achieved when performing *any subset* of tasks.

## 6.1 Experiment Settings

**Baselines for Network Merging.** We compare PAM with two merging schemes.

- **Baseline 1**. It simply skips network merging in the "merge & prune" framework. Therefore, no multitask network is constructed. As mentioned in Sec. 1, this scheme optimises the pruning of *single-task* networks.
- **Baseline 2**. Pre-trained single-task networks are merged as a multitask network by MTZ [13], a state-of-the-art network merging scheme. Applying MTZ in "merge & prune" can minimise the computation cost of a multitask network when *all tasks* are executed.

**Methods for Network Pruning.** Since we aim to compare different network merging schemes in the "merge & prune" framework, we apply the same network pruning method on the neural network(s) constructed by different merging schemes. To show that PAM works with different pruning methods, we choose two

state-of-the-art structured network pruning methods: one [3] uses information theory based metrics (denoted as P1), and the other [21] uses sensitivity based metrics (denoted as P2).

The pruning methods are applied to the neural network(s) constructed by different merging schemes as follows. For Baseline 1, each single-task network is pruned independently. For the multi-task network constructed with Baseline 2 and PAM, we prune every subgraph for each individual task in an alternating manner (*e.g.*, task $A \rightarrow B \rightarrow C \rightarrow A \rightarrow B \rightarrow \cdots$) in order to balance between tasks. However, only P2 is originally designed to prune a ResNet. Hence we only experiment ResNets with P2.

**Datasets and Single-Task Networks.** We define tasks from three datasets: Fashion-MNIST [28], CelebA [20], and LFW [14]. Fashion-MNIST and CelebA each contains *two* tasks. LFW contains *five* tasks. We use LeNet-5 [17] as pre-trained single-task networks for tasks derived from Fashion-MNIST, and VGG-16 [24] for tasks from CelebA and LFW. We also use ResNet-18 and ResNet-34 [11] as pre-trained single-task networks for CelebA. See Appendix B for more details of dataset setup and the inference accuracy and FLOPs of the *pre-trained* single-task networks.

**Evaluation Metrics.** For a given set of tasks, we aim to minimise the computation cost of all task combinations. To assess computation cost independent of hardware, we use the number of floating point operations (FLOP) as the metric. For fair comparison, the network(s) constructed by different merging schemes are pruned while preserving almost the same inference accuracy. To quantify the performance advantage of PAM *over baselines* over all task combinations, we adopt the following two single-valued criteria:

- **Average Gain**. This metric measures the averaged computation cost reduction of "PAM & prune" over "baseline & prune" across *all task combinations*. For example, given two tasks $A$ and $B$, there are three task combinations: $A$, $B$ and $A\&B$. When executing these task combinations, the FLOPs of the network after "PAM & prune" are $c^P_A$, $c^P_B$ and $c^P_{A,B}$, respectively. After "baseline 1 & prune", the FLOPs are $c^{B1}_A$, $c^{B1}_B$ and $c^{B1}_{A,B}$, respectively. The average gain over baseline 1 is calculated as $\frac{1}{3}(c^{B1}_A/c^P_A + c^{B1}_B/c^P_B + c^{B1}_{A,B}/c^P_{A,B})$.
- **Peak Gain**. This metric measures the maximal computation cost reduction across *all task combinations*. Using the same example and notations as above, the peak gain over baseline 1 is calculated as $\max\{c^{B1}_A/c^P_A, c^{B1}_B/c^P_B, c^{B1}_{A,B}/c^P_{A,B}\}$.

All experiments are implemented with TensorFlow and conducted on a workstation with Nvidia RTX 2080 Ti GPU.

## 6.2 Main Experiment Results

**Overall Performance Gain.** Fig. 5 shows the average and peak gains of PAM over the two baselines with different models (LeNet-5, VGG-16, ResNet-18, RestNet-34), datasets (Fashion-MNIST, CelebA, LFW), and pruning methods (P1, P2). The detailed FLOPs and inference accuracy on two-task merging (Fashion-MNIST and CelebA) are listed in Table 1, Table 2 and Table 3. Due to limited space, the results of five-task merging (LFW) are in our technical report [12].

Compared with baseline 1, PAM achieves 1.07× to 1.64× average gain and 1.16× to 4.87× peak gain. Compared with baseline 2, PAM achieves 1.51× to 1.69× average gain and 1.56× to 2.01× peak gain. In general, PAM has significant performance advantage over both baselines across datasets and network architectures.

**Table 1: Test accuracy and computation cost of all tasks combinations with LeNet-5 on Fashion-MNIST pruned by P1/P2.**

| Pruning | Tasks | Accuracy | | | FLOPs ($\times 10^6$) | | |
|---|---|---|---|---|---|---|---|
| | | B1 | B2 | PAM | B1 | B2 | PAM |
| P1 | A | 95.42% | 95.30% | 94.67% | 28.34 | 52.58 | 28.49 |
| | B | 96.30% | 96.40% | 95.70% | 28.34 | 52.58 | 26.16 |
| | A&B | 95.86% | 95.85% | 95.19% | 56.69 | 52.58 | 48.68 |
| P2 | A | 95.82% | 95.73% | 95.70% | 18.64 | 31.19 | 18.65 |
| | B | 96.46% | 96.72% | 96.38% | 18.64 | 31.19 | 18.65 |
| | A&B | 96.14% | 96.22% | 96.04% | 37.27 | 31.19 | 26.48 |

**Table 2: Test accuracy and computation cost of all tasks combinations with VGG-16 on CelebA pruned by P1/P2.**

| Pruning | Tasks | Accuracy | | | FLOPs ($\times 10^6$) | | |
|---|---|---|---|---|---|---|---|
| | | B1 | B2 | PAM | B1 | B2 | PAM |
| P1 | A | 89.45% | 89.09% | 89.60% | 4.52 | 7.3 | 4.48 |
| | B | 87.81% | 87.69% | 88.00% | 4.32 | 7.3 | 4.49 |
| | A&B | 88.63% | 88.39% | 88.80% | 8.85 | 7.3 | 4.70 |
| P2 | A | 90.34% | 90.27% | 90.36% | 153.13 | 243.20 | 155.82 |
| | B | 88.84% | 88.74% | 88.76% | 152.65 | 243.20 | 155.84 |
| | A&B | 89.59% | 89.51% | 89.56% | 305.78 | 243.20 | 156.74 |

**Table 3: Test accuracy and computation cost with ResNet-18/ResNet-34 on CelebA pruned by P1.**

| Model | Tasks | Accuracy | | | FLOPs ($\times 10^6$) | | |
|---|---|---|---|---|---|---|---|
| | | B1 | B2 | PAM | B1 | B2 | PAM |
| ResNet-18 | A | 89.83% | 89.30% | 89.93% | 5.72 | 8.84 | 4.78 |
| | B | 88.25% | 88.20% | 88.36% | 5.72 | 8.84 | 4.83 |
| | A&B | 89.04% | 88.75% | 89.15% | 11.44 | 8.84 | 6.40 |
| ResNet-34 | A | 89.99% | 89.70% | 90.05% | 8.43 | 12.11 | 6.94 |
| | B | 88.44% | 88.98% | 88.42% | 8.43 | 12.11 | 6.94 |
| | A&B | 89.22% | 89.34% | 89.24% | 16.86 | 12.11 | 10.29 |

**Effectiveness of PAM.** From Fig. 5, the performance gain of PAM varies across baselines and datasets. Such variations in average and peak gains are influenced by *how many neurons are shared* and *how many networks are merged*. Fig. 6 shows how many neurons (kernels) are shared after "PAM & prune" on LeNet-5 and VGG-16.

- **The more neurons shared, the higher gain PAM has over baseline 1.** "Baseline 1 & prune" can effectively reduce the computation cost when *only one* task is performed. However, when many neurons can be shared (see Fig. 6(b), (c), (e), and (f)), baseline 1 is sub-optimal when multiple tasks are executed simultaneously, as it is unable to reduce computation by sharing neurons. This is why PAM outperforms baseline 1 more on CelebA and LFW.
- **The fewer neurons shared, the higher gain PAM has over baseline 2.** "Baseline 2 & prune" can effectively reduce the computation cost via neuron sharing when *all* tasks are performed simultaneously. However, when only few neurons

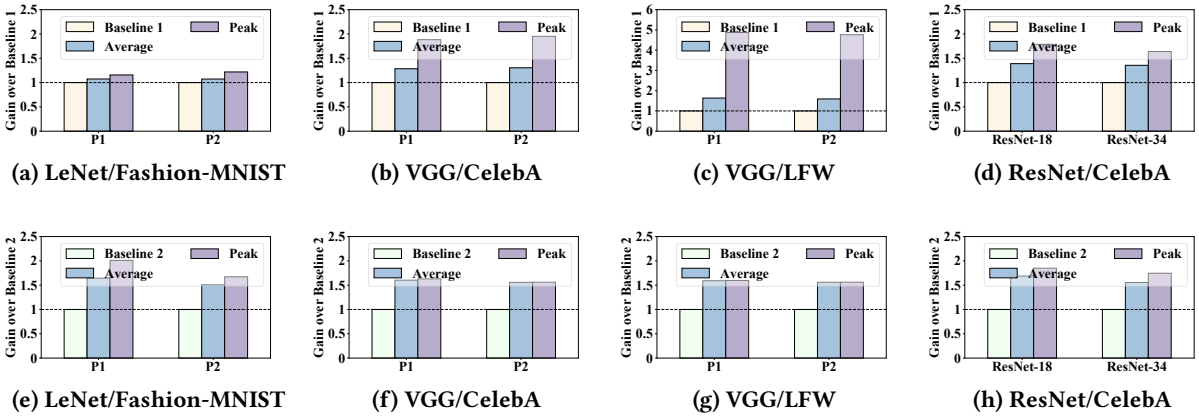


**Figure 5: Average and peak gain of PAM over baselines in different combinations of models, datasets, and pruning methods. The upper row (a)-(d) shows the gain of PAM over baseline 1. The lower row (e)-(h) shows the gain of PAM over baseline 2. Note that the average and peak gain of each baseline is 1 by definition.**

can be shared (see Fig. 6(a) and (d)), the multitask network merged by baseline 2 cannot shut down the unnecessary neurons when not all tasks are executed, and hence yields sub-optimal computation cost. This is why PAM outperforms baseline 2 more on Fashion-MNIST.

- **The more networks merged, the higher gain PAM has over both baselines.** As the number of single-task networks (tasks) increases, "PAM & prune" can either share more neurons and yield lower computation than "baseline 1 & prune", or shut down more unnecessary neurons and yield lower computation than "baseline 2 & prune". Therefore the performance gain of PAM over baseline 1 on LFW is such significantly higher than on CelebA. This is also the reason why the performance gain of PAM over baseline 2 on LFW is not much lower than on CelebA, although on LFW we have the highest degree of sharing.

**Takeaways.** Although the performance of PAM varies across tasks, it achieves consistently solid advantages over both baselines. We may conclude that it is always preferable to use PAM for efficient multitask inference, regardless of the amount of shareable neurons, of the probability of executing each task combination, of the network architecture, or of the pruning method used after merging.

## 6.3 Ablation Study

This subsection presents experiments to further understand the effectiveness of PAM.

*6.3.1 Impact of Task Relatedness.* This study aims to show the impact of task relatedness on the performance gain PAM can achieve. The number of neurons that can be shared among pre-trained networks is related to the relatedness among tasks. An effective network merging scheme should enforce increasing numbers of shared neurons between tasks with the increase of task relatedness.

**Settings.** We consider the 73 labels in LFW as 73 binary classification tasks, and measure the relatedness between each task pair by

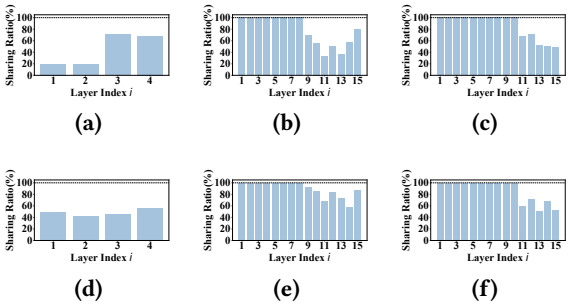


**Figure 6: Sharing ratio of each layer after "PAM & prune (P1 or P2)" on (a) LeNet/Fashion-MNIST with P1, (b) VGG/CelebA with P1, (c) VGG/LFW with P1, (d) LeNet/Fashion-MNIST with P2, (e) VGG/CelebA with P2, and (f) VGG/LFW with P2. In each layer, the sharing ratio is calculated as the number of shared neurons in $\mathrm{L}_i'^{A,B}$, divided by all neurons in $\mathrm{L}_i^{A,B}$. It ranges from 0% to 100%.**

$I(\mathrm{Y}^A;\mathrm{Y}^B)$. We then pick four pairs of tasks with $I(\mathrm{Y}^A;\mathrm{Y}^B) \approx 0$, 0.1, 0.2 and 0.5 bits, train four pairs of single-task VGG-16's on them, and construct four multitask networks using PAM.

**Results.** Fig. 7a plots the number of shared neurons in layer `f7` of these four multitask networks with different tuning threshold $\alpha$. The multitask networks for tasks pairs with higher correlation always share neurons. Hence, PAM can share an increasing number of neurons between tasks with the increase of task relatedness.

*6.3.2 Case Study: Task Inclusion.* This study aims to validate the effectiveness of PAM in an extreme yet common case of task relatedness where task $B$ is a sub-task of task $A$. Ideally, when the mutual information is precisely estimated and true largest sets of task-exclusive neurons are selected, PAM should effectively pick out only task-$A$-exclusive neurons.

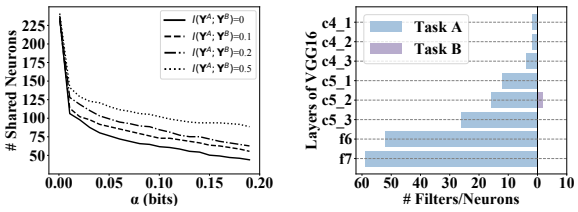


**Figure 7: Ablation studies: (a) Number of shared neurons in layer f7 of the four multitask networks constructed with PAM for different task pairs on LFW dataset, with different tuning parameter $\alpha$. (b) The number of non-shared neurons in $\mathbf{L}_i'^A$ and $\mathbf{L}_i'^B$ in the last eight layers when task $B$ is a subtask of task $A$. The networks are trained and merged on LFW.**

**Settings.** We pick 30 labels in LFW as task $A$ and 15 of them as task $B$. Hence task $A$ includes task $B$. We train two single-task VGG-16's on these two tasks separately and then merge them by PAM.

**Results.** Fig. 7b shows the number of non-shared neurons in $\mathbf{L}_i'^A$ and $\mathbf{L}_i'^B$ in the last eight layers of the merged network (the previous layers have exclusively shared neurons). Almost no neurons are selected for $\mathbf{L}_i'^B$ by Algorithm 2, validating its effectiveness.

## 7 CONCLUSION

In this paper, we investigate network merging schemes for efficient multitask inference. Given a set of single-task networks pre-trained for individual tasks, we aim to construct a multitask network such that applying existing network pruning methods on it can minimise the computation cost when performing any subset of tasks. We theoretically identify the conditions on the multitask network, and design Pruning-Aware Merging (PAM), a heuristic network merging scheme to construct such a multitask network. The merged multitask network can then be effectively pruned by existing network pruning methods. Extensive evaluations show that pruning a multitask network constructed by PAM achieves low computation cost when performing any subset of tasks in the network.

## ACKNOWLEDGEMENT

Part of Xiaoxi He and Lothar Thiele's work was supported by the Swiss National Science Foundation in the context of the NCCR Automation. Dawei Gao and Yongxin Tong's work is partially supported by the National Key Research and Development Program of China under Grant No. 2018AAA0101100, the National Science Foundation of China (NSFC) under Grant Nos. 61822201 and 62076017, the CAAI Huawei MindSpore Open Fund No. CAAIXSJLJJ-2020-020-A. Zimu Zhou's research was supported by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant. Zimu Zhou is the corresponding author.

## REFERENCES

[1] Anthony J Bell. 2003. The co-information lattice. In *International Workshop on Independent Component Analysis and Blind Signal Separation: ICA*. IEEE Press, Piscataway, NJ, USA, 921–926.
[2] Yi-Min Chou, Yi-Ming Chan, Jia-Hong Lee, Chih-Yi Chiu, and Chu-Song Chen. 2018. Unifying and merging well-trained deep neural networks for inference stage. In *IJCAI*. Morgan Kaufmann, Burlington, MA, USA, 2049–2056.
[3] Bin Dai, Chen Zhu, Baining Guo, and David Wipf. 2018. Compressing neural networks using the variational information bottleneck. In *ICML*. ACM, New York, NY, USA, 1143–1152.
[4] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model compression and hardware acceleration for neural networks: a comprehensive survey. *Proc. IEEE* 108, 4 (2020), 485–532.
[5] Misha Denil, Babak Shakibi, Laurent Dinh, Nando De Freitas, et al. 2013. Predicting parameters in deep learning. In *NeurIPS*. Curran Associates Inc., Red Hook, NY, USA, 2148–2156.
[6] Xin Dong, Shangyu Chen, and Sinno Pan. 2017. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *NeurIPS*. Curran Associates Inc., Red Hook, NY, USA, 4860–4874.
[7] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. NestDNN: resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *MobiCom*. ACM, New York, NY, USA, 115–127.
[8] Dawei Gao, Xiaoxi He, Zimu Zhou, Yongxin Tong, Ke Xu, and Lothar Thiele. 2020. Rethinking Pruning for Accelerating Deep Inference At the Edge. In *KDD*. ACM, New York, NY, USA, 155—-164.
[9] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic network surgery for efficient dnns. In *NeurIPS*. Curran Associates Inc., Red Hook, NY, USA, 1379–1387.
[10] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *ISCA*. ACM, New York, NY, USA, 243–254.
[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*. IEEE Press, Piscataway, NJ, USA, 770–778.
[12] Xiaoxi He, Dawei Gao, Zimu Zhou, Yongxin Tong, and Lothar Thiele. 2019. Pruning-aware xxx merging for efficient multitask inference. arXiv:1905.09676
[13] Xiaoxi He, Zimu Zhou, and Lothar Thiele. 2018. Multi-task zipping via layer-wise neuron sharing. In *NeurIPS*. Curran Associates Inc., Red Hook, NY, USA, 6016–6026.
[14] Gary Huang, Marwan Mattar, Honglak Lee, and Erik G Learned-Miller. 2012. Learning to align from scratch. In *NeurIPS*. Curran Associates Inc., Red Hook, NY, USA, 764–772.
[15] Artemy Kolchinsky and Brendan Tracey. 2017. Estimating mixture entropy with pairwise distances. *Entropy* 19, 7 (2017), 361.
[16] Neeraj Kumar, Alexander C Berg, Peter N Belhumeur, and Shree K Nayar. 2009. Attribute and simile classifiers for face verification. In *ICCV*. IEEE Press, Piscataway, NJ, USA, 365–372.
[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[18] Seulki Lee and Shahriar Nirjon. 2020. Fast and scalable in-memory deep multitask learning via neural weight virtualization. In *MobiSys*. ACM, New York, NY, USA, 175–190.
[19] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning filters for efficient convnets. In *ICLR*.
[20] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep learning face attributes in the wild. In *ICCV*. IEEE Press, Piscataway, NJ, USA, 3730–3738.
[21] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance estimation for neural network pruning. In *CVPR*. IEEE Press, Piscataway, NJ, USA, 11264–11272.
[22] Rasmus Rothe, Radu Timofte, and Luc Van Gool. 2018. Deep expectation of real and apparent age from a single image without facial landmarks. *International Journal of Computer Vision* 126, 2-4 (2018), 144–157.
[23] Andrew Michael Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan Daniel Tracey, and David Daniel Cox. 2018. On the information bottleneck theory of deep learning. In *ICLR*.
[24] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556
[25] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.
[26] Naftali Tishby and Noga Zaslavsky. 2015. Deep learning and the information bottleneck principle. In *Information Theory Workshop*. IEEE Press, Piscataway, NJ, USA, 1–5.
[27] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *NeurIPS*. Curran Associates Inc., Red Hook, NY, USA, 2074–2082.
[28] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747
[29] Yu Zhang and Qiang Yang. 2017. An overview of multi-task learning. *National Science Review* 5, 1 (2017), 30–43.

# APPENDIX

## A PROOFS

### A.1 Proof of Problem 1 in Sec. 4.2

Problem 1 occurs because of the lemma below.

**Lemma 1.** Reducing $\mathcal{R}_B(\widetilde{\mathbf{L}}_i^B)$ may decrease $I(\widetilde{\mathbf{L}}_i^A;\mathbf{Y}^A)$.

Proof. We decompose $I(\widetilde{\mathbf{L}}_i^A;\mathbf{Y}^A)$:

$$\begin{aligned} &I(\widetilde{\mathbf{L}}_i^A;\mathbf{Y}^A) = I(\mathbf{L}_i'^A;\mathbf{Y}^A)+ \\ &I(\mathbf{L}_i'^{A,B};\mathbf{Y}^A|\mathbf{L}_i'^A,\mathbf{Y}^B) + I(\mathbf{L}_i'^{A,B};\mathbf{Y}^A;\mathbf{Y}^B|\mathbf{L}_i'^A) \end{aligned} \tag{11}$$

where $I(A;B;C) = I(A;B) - I(A;B|C)$ is the *co-information* [1]. From Definition 1, we have:

$$\begin{aligned} \mathcal{R}_B(\widetilde{\mathbf{L}}_i^B) &= \sum_{\widetilde{L}_{i,j}^B \in \widetilde{\mathbf{L}}_i^B} H(\widetilde{L}_{i,j}^B) - I(\widetilde{\mathbf{L}}_i^B;\mathbf{Y}^B) \\ &= \sum_{\widetilde{L}_{i,j}^B \in \widetilde{\mathbf{L}}_i^B} H(\widetilde{L}_{i,j}^B) - H(\widetilde{\mathbf{L}}_i^B) + H(\widetilde{\mathbf{L}}_i^B|\mathbf{Y}^B) \end{aligned} \tag{12}$$

For the last term, we have:

$$\begin{aligned} &H(\widetilde{\mathbf{L}}_i^B|\mathbf{Y}^B) \\ =&H(\mathbf{L}_i'^B,\mathbf{L}_i'^{A,B}|\mathbf{Y}^B) && (13)\\ =&H(\mathbf{L}_i'^{A,B}|\mathbf{Y}^B) + H(\mathbf{L}_i'^B|\mathbf{L}_i'^{A,B},\mathbf{Y}^B) && (14)\\ =&I(\mathbf{L}_i'^{A,B};\mathbf{Y}^A|\mathbf{Y}^B) + H(\mathbf{L}_i'^{A,B}|\mathbf{Y}^A,\mathbf{Y}^B) + H(\mathbf{L}_i'^B|\mathbf{L}_i'^{A,B},\mathbf{Y}^B) && (15)\\ =&I(\mathbf{L}_i'^{A,B};\mathbf{Y}^A|\mathbf{L}_i'^A,\mathbf{Y}^B) + I(\mathbf{L}_i'^{A,B};\mathbf{Y}^A;\mathbf{L}_i'^A|\mathbf{Y}^B)+ \\ &H(\mathbf{L}_i'^{A,B}|\mathbf{Y}^A,\mathbf{Y}^B) + H(\mathbf{L}_i'^B|\mathbf{L}_i'^{A,B},\mathbf{Y}^B) && (16) \end{aligned}$$

Hence, $H(\widetilde{\mathbf{L}}_i^B|\mathbf{Y}^B)$ includes $I(\mathbf{L}_i'^{A,B};\mathbf{Y}^A|\mathbf{L}_i'^A,\mathbf{Y}^B)$. Reducing $\mathcal{R}_B(\widetilde{\mathbf{L}}_i^B)$ may decrease $I(\widetilde{\mathbf{L}}_i^A;\mathbf{Y}^A)$. □

### A.2 Proof of Theorem 1

Proof. The proof shows the conditions in Theorem 1 solve *(i)* Problem 1 in Sec. 4.2 and *(ii)* Problem 2 in Sec. 4.2.

**Solving Problem 1 in Sec. 4.2.** From (11) we have the following if $I(\mathbf{L}_i'^{A,B};\mathbf{Y}^A|\mathbf{L}_i'^A,\mathbf{Y}^B) = 0$:

$$I(\widetilde{\mathbf{L}}_i^A;\mathbf{Y}^A) = I(\mathbf{L}_i'^A;\mathbf{Y}^A) + I(\mathbf{L}_i'^{A,B};\mathbf{Y}^A;\mathbf{Y}^B|\mathbf{L}_i'^A) \tag{17}$$

$\mathbf{L}_i'^A$ is not in $\widetilde{\mathbf{L}}_i^B$. Hence $I(\mathbf{L}_i'^A;\mathbf{Y}^A)$ is unaffected when $\mathcal{R}_B(\widetilde{\mathbf{L}}_i^B)$ is reduced. $I(\mathbf{L}_i'^{A,B};\mathbf{Y}^A;\mathbf{Y}^B|\mathbf{L}_i'^A)$ is included in $I(\widetilde{\mathbf{L}}_i^B;\mathbf{Y}^B)$. Thus minimising $\mathcal{R}_B(\widetilde{\mathbf{L}}_i^B) - \tilde{\xi}_i^B \cdot I(\widetilde{\mathbf{L}}_i^B;\mathbf{Y}^B)$ will not reduce $I(\mathbf{L}_i'^{A,B};\mathbf{Y}^A;\mathbf{Y}^B|\mathbf{L}_i'^A)$ with a proper $\tilde{\xi}_i^B$. All still hold if we swap $A$ and $B$ in the above equations. Consequently, if $I(\mathbf{L}_i'^{A,B};\mathbf{Y}^A|\mathbf{L}_i'^A,\mathbf{Y}^B) = I(\mathbf{L}_i'^{A,B};\mathbf{Y}^B|\mathbf{L}_i'^B,\mathbf{Y}^A) = 0$, the first two objectives in optimisation problem (2) become non-conflicting.

**Table 4: Decomposition of $\mathcal{R}_{A,B}(\mathbf{L}_i^{A,B})$.**

$$\begin{aligned} &\mathcal{R}_{A,B}(\mathbf{L}_i^{A,B}) \\ =&\sum_{L_{i,j}^{A,B} \in \mathbf{L}_i^{A,B}} H(L_{i,j}^{A,B}) - H(\widetilde{\mathbf{L}}_i^A,\widetilde{\mathbf{L}}_i^B) + H(\widetilde{\mathbf{L}}_i^A,\widetilde{\mathbf{L}}_i^B|\mathbf{Y}^A,\mathbf{Y}^B) && (27)\\ =&\sum_{L_{i,j}^A \in \widetilde{\mathbf{L}}_i^A} H(L_{i,j}^A) - I(\widetilde{\mathbf{L}}_i^A;\mathbf{Y}^A,\mathbf{Y}^B) + \sum_{L_{i,j}^B \in \widetilde{\mathbf{L}}_i^B} H(L_{i,j}^B) - I(\widetilde{\mathbf{L}}_i^B;\mathbf{Y}^A,\mathbf{Y}^B) \\ &+I(\widetilde{\mathbf{L}}_i^A,\widetilde{\mathbf{L}}_i^B;\mathbf{Y}^A,\mathbf{Y}^B) - \sum_{L_{i,j} \in \mathbf{L}_i'^{A,B}} H(L_{i,j}) && (28)\\ =&\sum_{L_{i,j}^A \in \widetilde{\mathbf{L}}_i^A} H(L_{i,j}^A) - I(\widetilde{\mathbf{L}}_i^A;\mathbf{Y}^A) - I(\widetilde{\mathbf{L}}_i^A;\mathbf{Y}^B|\mathbf{Y}^A) + \sum_{L_{i,j}^B \in \widetilde{\mathbf{L}}_i^B} H(L_{i,j}^B) - I(\widetilde{\mathbf{L}}_i^B;\mathbf{Y}^B) \\ &-I(\widetilde{\mathbf{L}}_i^B;\mathbf{Y}^A|\mathbf{Y}^B) + I(\widetilde{\mathbf{L}}_i^A;\widetilde{\mathbf{L}}_i^B;\mathbf{Y}^A,\mathbf{Y}^B) - \sum_{L_{i,j} \in \mathbf{L}_i'^{A,B}} H(L_{i,j}) && (29)\\ =&\mathcal{R}_A(\widetilde{\mathbf{L}}_i^A) + \mathcal{R}_B(\widetilde{\mathbf{L}}_i^B) - I(\widetilde{\mathbf{L}}_i^A;\mathbf{Y}^B|\mathbf{Y}^A) \\ &-I(\widetilde{\mathbf{L}}_i^B;\mathbf{Y}^A|\mathbf{Y}^B) + I(\widetilde{\mathbf{L}}_i^A;\widetilde{\mathbf{L}}_i^B;\{\mathbf{Y}^A,\mathbf{Y}^B\}) - \sum_{L_{i,j} \in \mathbf{L}_i'^{A,B}} H(L_{i,j}) && (30) \end{aligned}$$

**Solving Problem 2 in Sec. 4.2.** We first decompose $\mathcal{R}_{A,B}(\mathbf{L}_i^{A,B})$ as in Table 4. Then from (30), we have

$$\begin{aligned} &\mathcal{R}_{A,B}(\mathbf{L}^{A,B}) - \big(\mathcal{R}_A(\widetilde{\mathbf{L}}_i^A) + \mathcal{R}_B(\widetilde{\mathbf{L}}_i^B)\big) \\ \le&I(\widetilde{\mathbf{L}}_i^A;\widetilde{\mathbf{L}}_i^B;\{\mathbf{Y}^A,\mathbf{Y}^B\}) - \sum_{L_{i,j} \in \mathbf{L}_i'^{A,B}} H(L_{i,j}) && (18)\\ \le&I(\widetilde{\mathbf{L}}_i^A;\widetilde{\mathbf{L}}_i^B) - \sum_{L_{i,j} \in \mathbf{L}_i'^{A,B}} H(L_{i,j}) && (19)\\ =&I(\mathbf{L}_i'^A,\mathbf{L}_i'^{A,B};\mathbf{L}_i'^B,\mathbf{L}_i'^{A,B}) - \sum_{L_{i,j} \in \mathbf{L}_i'^{A,B}} H(L_{i,j}) && (20)\\ \le&I(\mathbf{L}_i'^A;\mathbf{L}_i'^B) + H(\mathbf{L}_i'^{A,B}) - \sum_{L_{i,j} \in \mathbf{L}_i'^{A,B}} H(L_{i,j}) && (21)\\ \le&I(\mathbf{L}_i'^A;\mathbf{L}_i'^B) && (22) \end{aligned}$$

Further,

$$\begin{aligned} &I(\mathbf{L}_i'^A;\mathbf{L}_i'^B) \\ =&I(\mathbf{L}_i'^A;\mathbf{L}_i'^B;\mathbf{Y}^A;\mathbf{Y}^B) + I(\mathbf{L}_i'^A;\mathbf{L}_i'^B;\mathbf{Y}^A|\mathbf{Y}^B) + I(\mathbf{L}_i'^A;\mathbf{L}_i'^B|\mathbf{Y}^A) && (23)\\ \le&I(\mathbf{L}_i'^A;\mathbf{L}_i'^B;\mathbf{Y}^A;\mathbf{Y}^B) + H(\mathbf{L}_i'^A|\mathbf{Y}^A) + H(\mathbf{L}_i'^B|\mathbf{Y}^B) && (24)\\ \le&I(\mathbf{L}_i'^A;\mathbf{L}_i'^B;\mathbf{Y}^A;\mathbf{Y}^B) + \mathcal{R}_A(\widetilde{\mathbf{L}}_i^A) + \mathcal{R}_B(\widetilde{\mathbf{L}}_i^B) && (25) \end{aligned}$$

This is a loose upper bound. However, since $\mathcal{R}_{A,B}(\mathbf{L}^{A,B})$, $\mathcal{R}_A(\widetilde{\mathbf{L}}_i^A)$ and $\mathcal{R}_B(\widetilde{\mathbf{L}}_i^B)$ are lower bounded by 0, it suffices to show that when $I(\mathbf{L}_i'^A;\mathbf{L}_i'^B;\mathbf{Y}^A;\mathbf{Y}^B) = 0$, minimising $\mathcal{R}_A(\widetilde{\mathbf{L}}_i^A)$ and $\mathcal{R}_B(\widetilde{\mathbf{L}}_i^B)$ will minimise $\mathcal{R}_{A,B}(\mathbf{L}^{A,B})$.

In summary, when

$$\begin{aligned} I(\mathbf{L}_i'^A;\mathbf{L}_i'^B;\mathbf{Y}^A;\mathbf{Y}^B) &= 0 \\ I(\mathbf{L}_i'^{A,B};\mathbf{Y}^A|\mathbf{L}_i'^A,\mathbf{Y}^B) &= 0 \\ I(\mathbf{L}_i'^{A,B};\mathbf{Y}^B|\mathbf{L}_i'^B,\mathbf{Y}^A) &= 0 \end{aligned} \tag{26}$$

the optimisation problem (2) is reduced to two non-conflicting optimisation problems (4). □

## A.3 Proof of Theorem 2

Proof. First, for co-information between four random variables, we have from [1]:

$$0 \leq I(\mathbf{L}_i'^A; \mathbf{L}_i'^B; \mathbf{Y}^A; \mathbf{Y}^B) \leq \min\{I(\mathbf{L}_i'^A; \mathbf{Y}^B), I(\mathbf{L}_i'^B; \mathbf{Y}^A)\} \quad (31)$$

Therefore, the first condition in Theorem 1, *i.e.*, $I(\mathbf{L}_i'^A; \mathbf{L}_i'^B; \mathbf{Y}^A; \mathbf{Y}^B) = 0$, is achieved by minimising $I(\mathbf{L}_i'^A; \mathbf{Y}^B)$ and $I(\mathbf{L}_i'^B; \mathbf{Y}^A)$ to 0.

For the second condition in Theorem 1, *i.e.*, $I(\mathbf{L}_i'^{A,B}; \mathbf{Y}^A | \mathbf{L}_i'^A, \mathbf{Y}^B) = 0$, we have:

$$\begin{aligned} & I(\mathbf{L}_i'^{A,B}; \mathbf{Y}^A | \mathbf{L}_i'^A, \mathbf{Y}^B) \\ \leq & H(\mathbf{Y}^A | \mathbf{L}_i'^A, \mathbf{Y}^B) & (32) \\ = & H(\mathbf{Y}^A | \mathbf{Y}^B) - I(\mathbf{Y}^A; \mathbf{L}_i'^A) + I(\mathbf{Y}^A; \mathbf{L}_i'^A; \mathbf{Y}^B) & (33) \\ \leq & H(\mathbf{Y}^A | \mathbf{Y}^B) - I(\mathbf{Y}^A; \mathbf{L}_i'^A) + I(\mathbf{L}_i'^A; \mathbf{Y}^B) & (34) \end{aligned}$$

Given $A$ and $B$, $H(\mathbf{Y}^A | \mathbf{Y}^B)$ is constant. The second condition in Theorem 1 is achieved by minimising $I(\mathbf{L}_i'^A; \mathbf{Y}^B)$ to 0 and maximising $I(\mathbf{Y}^A; \mathbf{L}_i'^A)$ to $H(\mathbf{Y}^A | \mathbf{Y}^B)$.

The same holds if we swap $A$ and $B$. The third condition in Theorem 1, *i.e.*, $I(\mathbf{L}_i'^{A,B}; \mathbf{Y}^B | \mathbf{L}_i'^B, \mathbf{Y}^A) = 0$, is achieved by minimising $I(\mathbf{L}_i'^B; \mathbf{Y}^A)$ and maximising $I(\mathbf{Y}^B; \mathbf{L}_i'^B)$. □

# B DETAILED DATASET SETUP

**Fashion-MNIST.** The Fashion-MNIST dataset[1] contains 8000 training images and 2000 test images with a resolution of $496 \times 124$. Each image has four fashion product images randomly selected from Fashion-MNIST [28]. The 10 categories of fashion products is considered as 10 binary classification problem, and we divide them into two groups (5/5) to form task $A$ and $B$. On each task we train a LeNet-5, a commonly used architecture for Fashion-MNIST.

**CelebA.** The CelebA dataset[2] contains over 200 thousand celebrity face images labelled with 40 attributes. The 40 attributes is divided into two groups (20/20) to form task $A$ and $B$. The dataset is divided into training and test sets containing 80% and 20% of the samples. The input picture resolution is resized to $72 \times 72$. On each task we train slightly modified VGG-16 models, a commonly used single-task network architecture on CelebA. The width of the fully connected layers in VGG-16 is changed to 512. The convolutional layers are initialised with weights pre-trained for imdb-wiki [22], and use the same pre-processing steps.

**LFW.** The Labeled Faces in the Wild (LFW) dataset[3] contains over 13,000 face photographs collected from the web. Each face photo is associated with 73 attributes [16]. We randomly split the 73 labels in the LFW dataset into four groups with 15 labels each and one group with 13 labels. Each group of labels forms a single task. The dataset is divided into training and test sets containing 80% and 20% of the samples. Same as in CelebA, the input picture resolution is resized to $72 \times 72$. On each task we train slightly modified VGG-16 models, a commonly used single-task network architecture on LFW. The width of the fully connected layers in VGG-16 is changed to 128. The convolutional layers are initialised with weights pre-trained for imdb-wiki [22], and use the same pre-processing steps.

[1] https://github.com/f-rumblefish/Multi-Label-Fashion-MNIST
[2] http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html
[3] http://vis-www.cs.umass.edu/lfw/

**Table 5: Test accuracy and computation cost of pre-trained single-task networks.**

| Model/Dataset | Task | Accuracy | FLOPs ($\times 10^6$) |
|---|---|---|---|
| LeNet-5/Fashion-MNIST | A | 96.05% | 106.42 |
| | B | 96.37% | 106.42 |
| VGG-16/CelebA | A | 90.28% | 3112.20 |
| | B | 89.03% | 3112.20 |
| VGG-16/LFW | A | 90.23% | 3110.12 |
| | B | 84.15% | 3110.12 |
| | C | 85.03% | 3110.12 |
| | D | 86.62% | 3110.12 |
| | E | 87.44% | 3110.12 |
| ResNet-18/CelebA | A | 90.56% | 994.00 |
| | B | 88.91% | 994.00 |
| ResNet-34/CelebA | A | 90.42% | 1115.06 |
| | B | 88.70% | 1115.06 |

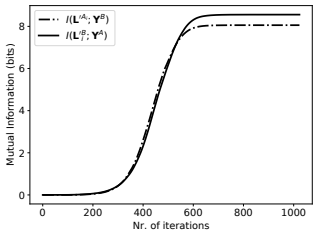


**Figure 8: Iterations of Line 19-22 and 24-27 in Algorithm 2. The shown example is on the `f7` layer of the VGG-16 networks trained and merged on CelebA.**

Table 5 summarises the inference accuracy and FLOPs of the *pre-trained* single-task networks.

# C VISUALISATION OF ALGORITHM 2

Fig. 8 illustrates two iterations of Line 19-22 and 24-27 in Algorithm 2 by showing $I(\mathbf{L}_i'^A; \mathbf{Y}^B)$ and $I(\mathbf{L}_i'^B; \mathbf{Y}^A)$ against the number of iterations. Here we use the `f7` layer of VGG-16 trained and merged for CelebA dataset as an example. The tuning parameter $\alpha$ is set to infinitely large in order to show all the possible cases of the iterations. From Fig. 8, we can observe three phases:

(1) In the first phase, $I(\mathbf{L}_i'^A; \mathbf{Y}^B)$ and $I(\mathbf{L}_i'^B; \mathbf{Y}^A)$ remains small, indicating that the selected $\mathbf{L}_i'^A$ and $\mathbf{L}_i'^B$ provides little information about the other task.
(2) In the second phase, $I(\mathbf{L}_i'^A; \mathbf{Y}^B)$ and $I(\mathbf{L}_i'^B; \mathbf{Y}^A)$ start to increase as it is impossible to add more neurons to $\mathbf{L}_i'^A$ and $\mathbf{L}_i'^B$ while keeping $I(\mathbf{L}_i'^A; \mathbf{Y}^B)$ and $I(\mathbf{L}_i'^B; \mathbf{Y}^A)$ close to zero.
(3) In the third phase, $I(\mathbf{L}_i'^A; \mathbf{Y}^B)$ and $I(\mathbf{L}_i'^B; \mathbf{Y}^A)$ start to saturate as the newly joined neurons contain mostly information already included in existing $\mathbf{L}_i'^A$ and $\mathbf{L}_i'^B$.

In practice, the parameter $\alpha$ tuned as remains small, and the iterations in Algorithm 2 as well as Algorithm 3 usually stop at the end of the first phase or the beginning of the second phase.