Lower Bounds for the Thickness and the Total Number of Edge Crossings of Euclidean Minimum Weight Laman Graphs and (2,2)-Tight Graphs

Yuki KAWAKAMI¹a), Member, Shun TAKAHASHI¹b), Nonmember, Kazuhiwa SETO¹c), Takashi HORIYAMA¹d), Members, Yuki KOBAYASHI¹e), Yuya HIGASHIKAWA¹f), Members, and Naoki KATOH¹g), Member

SUMMARY We explore the maximum total number of edge crossings and the maximum geometric thickness of the Euclidean minimum-weight \((k, \ell)\)-tight graph on a planar point set \(P\). In this paper, we show that \((10/7 - \epsilon)|P|\) and \((11/6 - \epsilon)|P|\) are lower bounds for the maximum total number of edge crossings for any \(\epsilon > 0\) in cases \((k, \ell) = (2, 3)\) and \((2, 2)\), respectively. We also show that the lower bound for the maximum geometric thickness is 3 for both cases. In the proofs, we apply the method of arranging isomorphic units regularly. While the method is developed for the proof in case \((k, \ell) = (2, 3)\), it also works for different \(\ell\).

key words: Laman graph, \((k, \ell)\)-tight graph, geometric thickness, sparse graph, \(k\)-planarity

1. Introduction

A bar-joint framework is one of the main frameworks studied in combinatorial rigidity theory. It consists of rigid bars and rotatable joints. We can discuss the bar-joint framework as a graph of combinatorial rigidity theory by mapping each joint to a vertex and each bar to a straight-line edge [1]. One of the most fundamental results in combinatorial rigidity theory asserts that given a graph \(G\) realized on a generic point set in the plane (i.e., the set of the coordinates is algebraically independent over the rational field), \(G\) is rigid if and only if \(G\) contains a spanning Laman subgraph [2]. A graph \(G = (V, E)\) is a Laman graph if it satisfies \(|E| \geq 2|V| - 3\) and \(|E(H)| \leq 2|V(H)| - 3\) for any subgraph \(H\) of \(G\) with \(E(H) \neq \emptyset\). Laman graphs appear in a wide range of applications, not only statics but also mechanical design such as linkages, design of CAD systems, analysis of protein flexibility, and sensor network localization [3], [4].

The concept of the sparsity condition of a Laman graph is generalized to a \((k, \ell)\)-tight graph (see, e.g., [5]). The class of \((k, \ell)\)-tight graphs includes important graphs: a Laman graph is a \((2,3)\)-tight graph, and a spanning tree being studied in various fields is a \((1,1)\)-tight graph. Furthermore, it is known that any \((k, k)\)-tight graph can be decomposed into \(k\) edge-disjoint spanning trees [6], [7], and (6,6)-tight graphs appear in the necessary and sufficient condition of realization as an infinitesimally rigid body-hinge framework [8].

In two-dimensional generic bar-joint frameworks, the class of \((2,\ell)\)-tight graphs, including Laman graphs, plays an important role. For example, a \((2,2)\)-tight graph is minimally rigid when the joints are constrained to lie on the surface of a cylinder (since this surface allows two independent rigid-body motions) [9].

In this paper, we focus on the edge crossing of Laman graphs and \((2,2)\)-tight graphs. In order to realize a graph as a bar-joint framework on the plane in the real world, it is important to consider its edge crossing. Thus, one of our concerns is the graphs that maximize the total number of edge crossings. Another concern is the graphs that maximize the geometric thickness. The geometric thickness of graph \(G\) is the smallest number of layers necessary to partition the edge set of \(G\) into layers so that no layers have edge crossing (see, e.g., [10]).

Thus, more specifically, we at first focus on the maximum total number of edge crossings and the maximum geometric thickness of the Euclidean minimum-weight Laman graphs. The Euclidean minimum-weight Laman graph on a point set \(P\), denoted by MLG\((P)\), is the Laman graph with the minimum total edge length among all Laman graphs on \(P\). We also focus on those of the Euclidean minimum-weight \((2,2)\)-tight graphs, where the Euclidean minimum-weight \((k, \ell)\)-tight graph on \(P\), denoted by \((k, \ell)\)-MTG\((P)\), is defined similarly as a generalization of MLG\((P)\).

Bereg et al. [11] showed many properties of MLG\((P)\) for any semi-generic point set \(P\), e.g., \(6\)-planarity, no three edges cross each other, and the implication MLG\((P) \subseteq 1\text{-GG}(P)\) on the edges of the graphs, where \(1\text{-GG}(P)\) is a 1-Gabriel graph.

DOI: 10.1587/transinf.2023EDP7214

Copyright © 2024 The Institute of Electronics, Information and Communication Engineers
From the 6-planarity of MLG(P), they showed that the upper bound for the maximum number of edge crossings of MLG(P) is 6|P| − 9. They also showed that the lower bound for the maximum number of edge crossings of MLG(P) is |P| − 3. Later, Higashikawa et al. [12] improved the upper and lower bounds for the maximum total number of edge crossings of MLG(P) to 2.5|P| − 5 and (1.25 − ε)|P| for any ε > 0, respectively. Unfortunately, a gap between those bounds still exists. As for the maximum geometric thickness of MLG(P), since the geometric thickness of 1-GG(P) is at most 4 [13], MLG(P) ⊆ 1-GG(P) in [11] implies that the upper bound for the maximum geometric thickness of MLG(P) is 4. On the other hand, its lower bound is 2 since MLG(P) may have some edge crossings. Thus, we also have a gap in the maximum geometric thickness.

Furthermore, the maximum number of edge crossings and the maximum geometric thickness of (k, ℓ)-MTG(P) for general k and ℓ are in our interest. Bereg et al. [11] showed that (k, ℓ)-MTG(P) is (6k^2 + 4k + 4k − 10)-planar. In other words, each edge of (k, ℓ)-MTG(P) crosses at most (6k^2 + 4k + 4k − 10) other edges. According to this result, it is easy to see that the total number of edge crossings of (k, ℓ)-MTG(P) is at most (6k^2 + 4k + 4k − 10)(|P| − ℓ)/2. As Bereg et al. told in [11], this upper bound is not tight. In case k = 2 and ℓ = 3, we can see the bound is 22|P| − 33. There are many open questions regarding the maximum number of edge crossings and the maximum geometric thickness of the (k, ℓ)-MTG(P). As a first step for general k and ℓ, we focus on (2, 3)-MTG(P) and (2, 2)-MTG(P).

Our contribution is as follows. At first, we improve the lower bound for the maximum total number of edge crossings of MLG(P). In other words, we show a semi-generic point set P that improve the lower bound. Our idea for the proof is based on the method by Higashikawa et al. [12]: arrange the same units on a circumference, where each unit consists of carefully positioned five points. We extend this method by alternately arranging two types of units on a circumference. Each of both units consists of eight points, and their arrangement is well determined so as to derive isomorphic Euclidean minimum-weight Laman graphs and not to interfere with each other. By alternately arranging these two types of units, we derive lower bound \((\frac{14}{7} − ε)|P|\) for any ε > 0.

Our extended method has the possibility to derive a lower bound for general cases. To show the power of our method, we apply it to different ℓ. More precisely, we derive a lower bound for the maximum total number of edge crossings of (2, 2)-MTG(P) by regularly arranging different units made under the same design: Each unit differs in only one parameter regarding width, while all other parameters are the same for all units. By regularly arranging these units, we derive lower bound \((\frac{14}{7} − ε)|P|\) for any ε > 0, while no lower bounds were known. Our results on (2, 3)-MTG(P) and (2, 2)-MTG(P) suggest that the maximum total number of edge crossings depends on parameter ℓ. Recall that the upper bound by Bereg et al. [11] was with parameter k. Thus, we can open the door for the discussion with general k and ℓ.

We also address the lower bounds for the maximum geometric thickness of MLG(P) and (2, 2)-MTG(P). We use the edge-crossing graph (also called crossing dual graph) of a geometric graph G(P). Each vertex and edge of the edge-crossing graph corresponds to an edge of G(P) and the edge crossing of two edges of G(P), respectively. Interestingly, the chromatic number of the edge-crossing graph is equal to the geometric thickness of the original geometric graph G(P) [12]. We show a semi-generic point set P such that the edge-crossing graph of MLG(P) contains a cycle of odd length. Since this implies its chromatic number is at least 3, we can improve the lower bound to 3. In a similar way, we can also derive the same lower bound for the maximum geometric thickness of (2, 2)-MTG(P).

2. Preliminaries

2.1 Minimum-Weight (k, ℓ)-Tight Graphs

A graph G = (V, E) is a (k, ℓ)-sparse graph \((0 ≤ k ≤ 2k − 1)\) if it satisfies \(|E(H)| ≤ k|V(H)| − ℓ\) for any subgraph H of G with \(E(H) \neq \emptyset\), where \(E(H)\) denotes the set of edges of H. A (k, ℓ)-sparse graph is a (k, ℓ)-tight graph if it has exactly \(k|V(G)| − ℓ\) edges. In particular, (1, 1)-tight graph, i.e., the graph for the case \(k = ℓ = 1\), is called a spanning tree, and (2, 3)-tight graph is called a Laman graph.

A geometric graph G(P) = (P, \(E_p\)) on a planar point set P is obtained by embedding a graph G = (V, E) into a two-dimensional Euclidean plane by a bijection \(p : V \rightarrow P\). Each vertex \(v_i \in V\) of graph G is mapped to a point \(p(v_i) = p_i\), and each edge \((v_i, v_j) \in E\) is mapped to a line segment \(p(v_i)p(v_j) \in E_p\). In this paper, we denote \(p_i p_j\) as both the line segment \(p(v_i)p(v_j)\) and the edge \((v_i, v_j)\). The weight of edge \(p_i p_j\) is defined as the Euclidean distance between two points \(p_i\) and \(p_j\), denoted by \(\|p_i p_j\|\).

The (k, ℓ)-tight graph with the minimum total edge weight among all (k, ℓ)-tight graphs on P is called the Euclidean minimum-weight (k, ℓ)-tight graph on P, and denoted by \((k, ℓ)\)-MTG(P). In case \(k = 2\) and \(ℓ = 3\), (2, 3)-MTG(P) is also called the Euclidean minimum-weight Laman graph on P, and denoted by MLG(P). Throughout the paper, we assume that no three points in P are collinear and that all distances between two points in P are distinct, called semigeneric. From this assumption, given a semi-generic point set P, we can uniquely obtain MLG(P) and \((k, ℓ)\)-MTG(P).

2.2 \((k, ℓ)\)-Sparsity Matroid

A matroid \(M\) is an ordered pair \((E, \mathcal{L})\) consisting of a finite set E and a family \(\mathcal{L}\) of subsets of E satisfying the following conditions:

\[(C1)\) \emptyset \in \mathcal{L}.

\[(C2)\) If \(X \in \mathcal{L}\) and \(X' \subseteq X\), then \(X' \in \mathcal{L}\).

\[(C3)\) If A and B are in \(\mathcal{L}\) and \(|A| < |B|\), then there is an element \(e \in B \setminus A\) satisfying \(A \cup \{e\} \in \mathcal{L}\).

A finite set E is a ground set of \(M\). The members of \(\mathcal{L}\) are
called independent. On the contrary, a subset of $E$ that is not in $L$ is called dependent. A maximum independent set and a minimal dependent set in $M$ are called a basis and a circuit of $M$, respectively. It is known that all subsets of a circuit are independent.

Given a graph $G = (V, E)$, let $L$ be the family of subsets of $E$, where each of which induces a $(k, t)$-sparse subgraph of $G$. Then, a pair $(E, L)$ is known to be a matroid, called the $(k, t)$-sparsity matroid of $G$. It is known that a $(k, t)$-sparsity matroid has a dimension of $k + t - 1$.

In this section, we show the lower bounds for the maximum geometric thickness and the maximum total number of edge crossings of a $(k, t)$-tight graph.

**Lemma 1** (Proposition 5.3 [12]). Consider a matroid with a ground set $E$ whose elements are assigned distinct weights. Let $e^*$ be the element with the maximum weight in a circuit in $E$. Then, $e^*$ is not included in any minimum weight basis of the matroid.

**Lemma 2** (Lemma 2.2 [11]). Let $P$ be a semi-generic point set in the plane, $Q \subseteq P$, and $a, b \in Q$. Also let $E_{ab}(Q)$ be the set of edges $pq \in Q, p \neq q, ||pq|| < ||ab||$. If there exists a subset of $E_{ab}(Q)$ that induces a Laman graph on $Q$, then $ab \notin E(MLG(P))$ holds.

The above Lemma 2 by Bereg et al. [11] is a good tool for distinguishing whether an edge is in the MLG(P). The sketch of its proof is as follows. Given a semi-generic point set $P$, let $K(P) = (P, E)$ be the geometric complete graph on $P$. Lemma 2 is obtained by applying Lemma 1 to a two-dimensional rigidity matroid, where the ground set is the edge set $E$. As $P$ is a semi-generic point set, all elements (i.e., edges) of $E$ have distinct weights. The assumption of Lemma 2 says we have a Laman graph on $Q$ that is induced by some subset $E'$ of $E_{ab}(Q)$. For any edge $e \in E(K(Q)) \setminus E'$, if we add $e$ to $E'$, the resulting edge-induced graph $(Q, E' \cup \{e\})$ does not satisfy the definition of a $(2,3)$-sparse graph. This means that edge set $E' \cup \{e\}$ is not independent, i.e., it is dependent. Thus, $ab \in E(K(Q)) \setminus E'$ implies that $E' \cup \{ab\}$ is dependent. As a circuit is a minimal dependent set, there exists a circuit $E'' \cup \{ab\}$ satisfying $E'' \subseteq E'$. By definition of $E_{ab}(Q)$ in Lemma 2, $||e|| < ||ab||$ holds for any $e \in E_{ab}(Q)$. This means that, as $E'' \subseteq E' \subseteq E_{ab}(Q)$, edge $ab$ is the element with the maximum weight in the circuit $E'' \cup \{ab\}$. Therefore, Lemma 1 says $ab$ is not included in any minimum weight basis. As the minimum weight basis edge-induces the $(k, t)$-MTG(P), $ab \notin E(MLG(P))$ holds.

By a similar argument with Lemma 2, we can extend the above lemma to the general case, i.e., $(k, t)$-MTG(P) for general $k$ and $t$.

**Lemma 3.** Let $P$ be a semi-generic point set in the plane, $Q \subseteq P$, and $a, b \in Q$. Also let $E_{ab}(Q)$ be the set of edges $pq \in Q, p \neq q, ||pq|| < ||ab||$. If there exists a subset of $E_{ab}(Q)$ that induces a $(k, t)$-tight graph on $Q$, then $ab \notin E((k, t)$-MTG(P)) holds.

We can obtain Lemma 3 by replacing the two-dimensional rigidity matroid on the edge set of the geometric complete graph $K(P)$ on $P$ and Laman graphs in Lemma 2 by the $(k, t)$-sparsity matroid on the edge set of $K(P)$ and $(k, t)$-tight graph, respectively. As in Lemma 2, the ground set is $E$ and all elements of $E$ have distinct weight.

In this section, we show the lower bounds for the maximum geometric thickness and the maximum total number of edge crossings of $(2, 2)$-MTG(P).

**2.3 Geometric Thickness of Geometric Graphs**

Our focus is the crossings of the edges in a geometric graph $G(P) = (P, E)$. Two edges $e$ and $e'$ ($e \in E$) are crossing if and only if they have a common point other than their both ends. We denote the total number of edge crossings in $G(P)$ by $\sigma(G(P))$. The geometric thickness of $G(P)$ is defined as the minimum positive integer $t$ satisfying the following conditions:

- $\bigcup_{i=1}^{t} E_i = E$.
- For any integer $i$ ($1 \leq i \leq t$), geometric graph $G_i(P) = (P, E_i)$ is non-crossing (i.e., $G_i(P)$ is a plane graph).

Suppose that we are given a geometric graph $G(P)$ with geometric thickness $t$, and that we partition $E$ into $t$ edge sets $E_1, E_2, \ldots, E_{t-1}$. Then, at least one geometric graph $G_i(P) = (P, E_i)$ has edge crossing.

We introduce edge-crossing graphs of geometric graphs to understand the geometric thickness. Given a geometric graph $G(P) = (P, E)$, its edge-crossing graph $(W, F)$ is defined as follows: each vertex $e \in W$ corresponds to edge $e \in E$, and edge $(e, e')$ is in $F$ if and only if edges $e$ and $e'$ cross each other in $G(P)$. The following relationship exists between the geometric thickness of a geometric graph and the chromatic number of the edge-crossing graph.

**Lemma 4** ([12]). The geometric thickness of a geometric graph $G(P)$ is equal to the chromatic number of the edge-crossing graph of $G(P)$.

**3. Lower Bounds**

In this section, we show the lower bounds for the maximum geometric thickness and the maximum total number
of edge crossings of MLG(P) and (k, ℓ)-MTG(P) by giving semi-generic point sets P, respectively. To improve the lower bound for the maximum total number of edge crossings, we use units consisting of several points. For MLG(P), the lower bound is derived by counting the total number of edge crossings of MLG on a point set with alternately arranged two types of units. For (2, 2)-MTG(P), we consider a point set regularly arranging different units made under the same design. We also improve the lower bound for the maximum geometric thickness by showing that the bounds of both MLG(P) and (2, 2)-MTG(P) is 3. We focus on MLG(P) in Sect. 3.1, and (2, 2)-MTG(P) in Sect. 3.2.

3.1 Minimum-Weight Laman Graph

Higashikawa et al. [12] derived the lower bound by regularly arranging the same units each of which consists of five points. The key point of their method is that, by adding one unit, the number of points increases by 4, and at the same time, the number of edge crossings increases by 5. Thus, the lower bound is derived by the ratio 5/4 = 1.25. We improve the lower bound by extending the idea to arrange two types of units U¹ and Uº alternately, where each unit consists of eight points. The alternate arrangement of U¹ and Uº is illustrated in Fig. 1. Both of these two units U¹ and Uº are positioned so as to derive the isomorphic Euclidean minimum-weight Laman graphs in all units. The alternation of two different types of units instead of the same type will give two crossings between the neighboring units. Let t denote the number of units we arrange, and P(t) denote the point set when t units are arranged. Also, we denote the i-th unit by Uᵢ (0 ≤ i ≤ t − 1) and the point pᵢ in unit Uᵢ by Pᵢ(ᵢ).

First, we describe the details of units U¹ and Uº, and show the Euclidean minimum-weight Laman graph on the point set of a unit U¹. In case i is even, unit Uᵢ is obtained by translating and rotating the eight points in U¹ illustrated in Fig. 2, where six parameters d, δ, δ', τ, τ' and hₑ or hₒ satisfy the following conditions:

(A) δ' > 3δ, δ + τ'
(B) d > δ + τ, 2δ'
(C) d + δ + 2τ + τ' < hₑ, hₒ < \( \frac{\delta(\delta' - 3\delta)}{2\tau'} \), \( \frac{d(\delta' - 3\delta)}{2\tau'} \)

Then, the Euclidean minimum-weight Laman graph on a point set U¹ is the geometric graph illustrated in Fig. 2, i.e., the geometric graph edge-induced by \{p₀p₁, p₀p₂, p₁p₁, p₁p₃, p₁p₅, p₂p₃, p₂p₂, p₂p₅, p₃p₄, p₃p₆, p₄p₆, p₅p₅, p₅p₇, p₆p₇, p₆p₆\}. The Euclidean minimum-weight Laman graph on Uº is the graph obtained by replacing hₑ in Fig. 2 with hₒ.

Proof. We prove this lemma by showing that the geometric graph illustrated in Fig. 2 is a Laman graph and that all edges not illustrated in Fig. 2 are not included in MLG(U¹). For discriminating whether an edge is in MLG(U¹) or not, we use Lemma 2.

First, we show the weights of all edges in increasing order to compare them.

\[ \|p₁p₂\| = \|p₅p₆\| = 2\delta \]
\[ \|p₃p₄\| = \|p₆p₇\| = \sqrt{(\delta - \delta')² + \tau²} \]
\[ \|p₄p₅\| = \|p₅p₆\| = \sqrt{(\delta' + \delta)² + \tau²} \]
Lemma 6. The set of edges in $MLG(P(t))$ is a union of the set of edges in $MLG(U_i)$ for $0 \leq i \leq t-1$ and the set of edges $p^{(t)}_i p^{(t+1)}_i$ between two neighboring units $U_i$ and $U_{i+1}$ for $0 \leq i \leq t-2$.

Proof. We can prove this lemma by a similar argument with Lemma 5. For edges between two points in the same unit $U_i$, all edges not included in $MLG(U_i)$ are excluded from $MLG(P(t))$ by the same argument. For edges between different units, only edges $p^{(t)}_2 p^{(t+1)}_1$ are included in $E(MLG(P(t)))$ and no other edges are included. For all integer $i$ ($0 \leq i \leq t-1$), the weight of edge $||p^{(t)}_2 p^{(t+1)}_1||$ can be approximated to $2d$ by adjusting four parameters $\delta, \delta', \tau, \tau'$ very small with satisfying the conditions (A) to (C) in Lemma 5. For all even $x$, the weights of the edges $p^{(t)}_1 p^{(t+1)}_1$ and $p^{(t)}_1 p^{(t+1)}_1$ can be approximated to $\sqrt{2d}$ by making $h_0$ a small value with satisfying condition (C) in Lemma 5 (and symmetrically for edges $p^{(t+1)}_3 p^{(t+1)}_3$ and $p^{(t+1)}_3 p^{(t+1)}_3$). The weights of the other edges between different units are larger than those of these edges. Note that all edges included in $MLG(U_i)$ for each unit $U_i$ are shorter than either edge $p^{(t)}_1 p^{(t+1)}_1$ or $p^{(t)}_1 p^{(t+1)}_1$. If we consider an edge set $E$ that is shorter than the weight min$(||p^{(t)}_1 p^{(t+1)}_1||, ||p^{(t)}_1 p^{(t+1)}_1||)$, then the graph $(P(t), E)$ includes a Laman graph on the point set $P(t)$. Therefore, edges $p^{(t)}_1 p^{(t+1)}_1$ and $p^{(t)}_1 p^{(t+1)}_1$ are not included in $E(MLG(P(t)))$ by Lemma 2. Furthermore, since all edges between different units except $p^{(t)}_1 p^{(t+1)}_1$ are longer than either edge $p^{(t)}_1 p^{(t+1)}_1$ or $p^{(t)}_1 p^{(t+1)}_1$, they are not included in $E(MLG(P(t)))$.

Now let us count the number of edge crossings in $MLG(P(t))$. For each unit $U_i$ ($0 \leq i \leq t-1$), we have eight crossings in $MLG(U_i)$. In addition, for each neighboring units $U_i$ and $U_{i+1}$ ($0 \leq i \leq t-2$), we have two crossings $p^{(t)}_2 p^{(t+1)}_1$ and $p^{(t)}_2 p^{(t+1)}_1$ (and $p^{(t)}_2 p^{(t+1)}_1$). Thus, the total number of edge crossings of $MLG(P(t))$ is $8 + 2(t-1) = 10t - 2$. And since the number of points is $7t + 1$ in this case, we obtain the following equation:

$$\frac{1}{2} \left( \left( \frac{7}{4} \right) \frac{11}{4} + \frac{1}{4} \right) \leq \frac{24}{7t + 1} = \frac{10 - 2}{7t + 1}$$

Here, we determine the radius $R$ of circle $C_0$ as

$$R = \frac{\sqrt{(d - \delta)^2 + \tau^2}}{2\tau}.$$
Theorem 1. For any $\epsilon > 0$, there exists a set of semi-generic points $P$ such that the total number of edge crossings of $\text{MLG}(P)$ is greater than $(\frac{10}{7} - \epsilon)\lvert P \rvert$.

Now, we focus on the maximum geometric thickness of $\text{MLG}(P)$. The graph shown in Fig. 3 is the edge-crossing graph of $\text{MLG}(U_{(\text{even})})$. Vertex $p_i p_j$ in Fig. 3 corresponds to edge $p_i p_j$ in $\text{MLG}(U_{(\text{even})})$, and edge $(p_i, p_j, p_k, p_l)$ corresponds to an edge crossing of edges $p_i p_j$ and $p_k p_l$ in $\text{MLG}(U_{(\text{even})})$. This graph contains a cycle of length 5. Hence, this graph is not 2-colorable. In other words, its chromatic number is 3 or more. Since the geometric thickness of a geometric graph is equal to the chromatic number of the graph from Lemma 4, the geometric thickness of $\text{MLG}$ is 3 or more. Thus, we have the following theorem.

Theorem 2. There exists a set of semi-generic points $P$ such that the geometric thickness of $\text{MLG}(P)$ is greater than or equal to 3.

3.2 Minimum-Weight $(2, 2)$-Tight Graph

In Sect. 3.1, we alternately arranged two types of units.

For any $\epsilon > 0$, there exists a set of semi-generic points $P$ such that the total number of edge crossings of $\text{MLG}(P)$ is greater than $(\frac{10}{7} - \epsilon)\lvert P \rvert$.

In this subsection, we derive a lower bound for the maximum total number of edge crossings of Euclidean minimum-weight $(2, 2)$-tight graph by a new approach: While we arrange mutually different $t$ units, the position of the points in the units is designed so that the Euclidean minimum-weight $(2, 2)$-tight graphs on all units are isomorphic as shown in Fig. 4. As in Sect. 3.1, we first describe each unit $U_t$ and the rules for arranging $t$ units. Let $P(t)$ denote the point set with $t$ units. Although point set $P(t)$ is not semi-generic, by moving the points in $P(t)$ infinitesimally, we can obtain a semi-generic point set without changing the inequalities on the weights of the edges. Then we discuss the Euclidean minimum-weight $(2, 2)$-tight graph on the point set with $t$ units, and finally the total number of edge crossings of $(2, 2)$-$\text{MTG}(P(t))$.

For each $i$ in $0 \leq i \leq t - 1$, unit $U_i$ consists of six points. The relative position of the points in each unit is illustrated in Fig. 5, and is determined by three parameters $\delta, \epsilon$ and $d_i$. Two parameters $\delta, \epsilon$ are common for all units, and parameter $d_i$ is different in each unit. In each $U_i$, three points $p_{0i}^{(i)}$, $p_{1i}^{(i)}$ and $p_{2i}^{(i)}$ (respectively, $p_{3i}^{(i)}$, $p_{4i}^{(i)}$ and $p_{5i}^{(i)}$) have the same $x$-coordinate. The height of $U_i$ is always $\epsilon$. On the other hand, as $i$ increases, the width of $U_i$ also increases.

Lemma 7. Suppose that the parameters $d_i, \delta$ and $\epsilon$ satisfy the following conditions:

(i) $\delta < d_i$
(ii) $\epsilon < d_i - \delta$

Then, the Euclidean minimum-weight $(2, 2)$-tight graph on point set $U_i$ is the geometric graph illustrated in Fig. 5, i.e.,
the geometric graph induced by $E_{MTG}^{(i)} = \{p_0^{(i)}p_1^{(i)}, p_0^{(i)}p_2^{(i)}, p_1^{(i)}p_1^{(i)}, p_1^{(i)}p_2^{(i)}, p_1^{(i)}p_3^{(i)}, p_1^{(i)}p_4^{(i)}, p_1^{(i)}p_5^{(i)}, p_2^{(i)}p_2^{(i)}\}$. 

Proof. We can confirm that the geometric graph $(U_i, E_{MTG}^{(i)})$ illustrated in Fig. 5 satisfies the definition of a (2,2)-tight graph. Then, we prove this lemma by showing that all edges in $K(U_i) \setminus E_{MTG}$ (i.e., the edges not illustrated in Fig. 5) are not included in $(2,2)$-MTG$(U_i)$, where $K(U_i)$ is the geometric complete graph on $U_i$. For discriminating whether an edge is in $(2,2)$-MTG$(U_i)$ or not, we use Lemma 3.

First, we show the weights of all edges in increasing order to compare them. In this proof, for convenience, we use $p_j$ to denote $p_j^{(i)}$.

\[
\|p_1p_2\| = \|p_3p_4\| = \delta
\]

\[
< \|p_0p_1\| = \|p_2p_3\| = \|p_4p_5\| = \sqrt{d_i^2 + \epsilon^2}
\]

\[
= \|p_1p_3\| = \|p_2p_4\| = \sqrt{(d_i + \delta)^2 + \epsilon^2}
\]

\[
< \|p_0p_2\| = \|p_1p_5\| = 2d_i + \delta
\]

\[
< \|p_0p_4\| = 3d_i^2 + 2\delta
\]

For all $i$, we have $p_{0i}p_{3i} \in K(U_i) \setminus E_{MTG}^{(i)}$. Among the edges in $K(U_i)$, we can obtain the set $E_{pop}(U_i)$ of edges shorter than $\|p_{0i}p_{3i}\|$ as $E_{pop}(U_i) = \{p_0p_1, p_0p_2, p_1p_2, p_1p_3, p_1p_4, p_2p_3, p_2p_4, p_3p_4, p_3p_5, p_4p_5\}$. We can confirm the geometric graph $(U_i, E_{pop}(U_i))$ satisfies the definition of a (2,2)-tight graph on $U_i$. By Lemma 3, we have $p_{0i}p_{3i} \notin E((2,2)$-MTG$(U_i))$.

Any other edge $p_jp_k \in K(U_i) \setminus E_{MTG}^{(i)}$ is longer than edge $p_{0i}p_{3i}$. Thus, $E_{pop}(U_i)$ is a subset of $E_{pop}(U_i)$, where $E_{pop}(U_i)$ is the set of edges shorter than $\|p_jp_k\|$. Since $(U_i, E_{pop}(U_i))$ is a (2,2)-tight graph as stated above, we have $p_jp_k \notin E((2,2)$-MTG$(U_i))$ by Lemma 3.

As illustrated in Fig. 4, we vertically arrange $t$ units so that two points $p_1^{(i)}$ and $p_1^{(i+1)}$ (respectively, $p_4^{(i)}$ and $p_4^{(i+1)}$) in each of the neighboring units $U_i, U_{i+1}$ have the same $y$-coordinate. The lengths $\|p_3^{(i)}p_1^{(i+1)}\|$ and $\|p_4^{(i)}p_2^{(i+1)}\|$ is fixed to $h$ for all $i$.

Lemma 8. Suppose that the parameters $d_i, \delta, \epsilon$ and $h$ satisfy the following conditions:

(I) $d_{i+1} > 2d_i + \delta$

(II) $\delta < d_0$

(III) $h > 2d_{i+1} + \delta$

(IV) $\epsilon < \frac{(d_i - d_0 - \delta)^2}{2h}$

The set of edges in $(2,2)$-MTG$(P(t))$ is a union of $E_{MTG}^{(i)}$ for $0 \leq i \leq t - 1$ and the set of edges $p_3^{(i)}p_1^{(i+1)}$ and $p_4^{(i)}p_2^{(i+1)}$ between two neighboring units $U_i$ and $U_{i+1}$ for $0 \leq i \leq t - 2$, where $E_{MTG}^{(i)}$ is the set of edges in $(2,2)$-MTG$(U_i)$.

Proof. We can prove this lemma by a similar argument with Lemma 7. For discriminating whether an edge is in $(2,2)$-MTG$(P(t))$ or not, we use Lemma 3. As for the edges in a unit, by the same argument, we can exclude all edges in $K(U_i) \setminus E_{MTG}^{(i)}$ for all $i$. In other words, the edges not illustrated in Fig. 5 are excluded.

Now, we discriminate the set $E_{dif}$ of edges between two different units, where $E_{dif} = \{p_x^{(i)}p_y^{(j)} | 0 \leq x, y \leq 5, 0 \leq i < j \leq t - 1\}$. Let $E_{inf}$ be a set of edges in $E_{dif}$ that are illustrated in Fig. 4, i.e., $E_{inf} = \{p_x^{(i)}p_y^{(i+1)} | 0 \leq i \leq t - 2\}$. We focus on edge $e_{ex} \in E_{dif} \setminus E_{inf}$ (i.e., the edges not illustrated in Fig. 4), and show that $e_{ex}$ is not an edge of $(2,2)$-MTG$(P(t))$ by comparing $e_{ex}$ with the edges in $E_{inf}$ and $E_{MTG}^{(i)}$. By conditions (I) to (IV) in this lemma, we can confirm that $\forall e_{ex} \in E_{dif} \setminus E_{inf} : \|e_{ex}\| < \|e_{inf}\|$ holds. By condition (III) in this lemma, $\forall e_{ex} \in E_{inf} : \|e_{ex}\| < \|e_{inf}\|$ holds for $0 \leq i \leq t - 1$. Thus, we have $\forall e_{ex} \in E_{inf} \setminus E_{inf} : \|e_{ex}\| < \|e_{inf}\|$. From above two inequalities on $e_{ex}$, for all $e_{ex} \in E_{dif}$ we have $E_{inf} \subseteq E_{ex}(P(t))$ and $E_{MTG}^{(i)} \subseteq E_{ex}(P(t))$ for $0 \leq i \leq t - 1$, where $E_{ex}(P(t))$ is the set of edges shorter than $\|e_{ex}\|$. This means that $(\bigcup_{i} E_{MTG}^{(i)}) \cup E_{inf}$ is a subset of $E_{ex}(P(t))$. We can confirm that the geometric graph induced by $(\bigcup_{i} E_{MTG}^{(i)}) \cup E_{inf}$ satisfies the definition of a (2,2)-tight graph. By Lemma 3, we have $E_{ex} \notin E((2,2)$-MTG$(P(t)))$ for all $e_{ex} \in E_{dif} \setminus E_{inf}$. □

Since there are five edge crossings in each unit $U_i (0 \leq i \leq t - 1)$ and six edge crossings for each neighboring unit $U_i$ and $U_{i+1} (0 \leq i \leq t - 2)$, the total number of edge crossings of $(2,2)$-MTG$(P(t))$ is $5t + 6(t - 1) = 11t - 6$. Since the number of points is $6t$ in this case, we obtain the following equation:

$$\sigma((2,2)$-MTG$(P(t))) = \frac{11t - 6}{6t} = \frac{11}{6} - \frac{1}{i}$$

Thus, with a sufficiently large $i$, for any $\epsilon > 0$, there exists a point set $P$ such that the total number of edge crossings $\sigma((2,2)$-MTG$(P))$ is at least $(\frac{11}{6} - \epsilon)|P|$. Although the above point set $P$ is not semi-generic, as in the previous subsection, we can obtain a semi-generic point set $P'$ such that the topology of $(2,2)$-MTG$(P')$ is the same as $(2,2)$-MTG$(P)$ by moving each point in $P$ infinitesimally.

Theorem 3. For any $\epsilon > 0$, there exists a set of semi-generic points $P$ such that the total number of edge crossings of $(2,2)$-MTG$(P)$ is greater than $(\frac{11}{6} - \epsilon)|P|$.

In the following, we discuss the thickness. We consider a point set $P_8 \subseteq P(2)$ consisting of eight points $p_2^{(0)}p_3^{(0)}p_4^{(0)}$, $p_5^{(0)}p_0^{(1)}p_1^{(1)}p_2^{(1)}$, and $p_3^{(1)}$. $(2,2)$-MTG$(P_8)$ is the geometric graph illustrated in Fig. 6, i.e., the graph induced by $p_2^{(0)}p_3^{(0)}p_4^{(0)}p_5^{(0)}p_0^{(1)}p_1^{(1)}p_2^{(1)}p_3^{(1)}p_4^{(1)}p_5^{(1)}p_0^{(1)}p_2^{(1)}$. We can
prove it by a similar argument as in the proof of Lemma 8. The graph shown in Fig. 7 is the edge-crossing graph of (2, 2)-MTG(P). This graph contains a cycle of length 5. Therefore, with the same reason as in the case of MLG(P) and Lemma 4, the geometric thickness of (2, 2)-MTG(P) is at least 3. Thus, we have the following theorem.

**Theorem 4.** There exists a set of semi-generic points P such that the geometric thickness of (2, 2)-MTG(P) is greater than or equal to 3.

4. **Concluding Remarks**

As for the maximum total number of edge crossings, with the idea of regularly arranging different units, we improved the lower bound for Euclidean minimum-weight Laman graphs and newly derived the lower bound for Euclidean minimum-weight (2, 2)-tight graphs. As for the maximum geometric thickness, we showed that the lower bounds for Euclidean minimum-weight Laman and (2, 2)-tight graphs are 3 since we have a cycle of length 5 in each of their edge-crossing graphs.

A gap, however, still exists between the upper and lower bounds for the maximum total number of edge crossings of MLG(P) for a semi-generic point set P. One of the challenges is to fill this gap. In our view, our technique has potential for further improvement by applying other units. There is also a large gap for (2, 2)-MTG(P). The reason for this large gap is due to the difficulty that the technique for the upper bound of MLG(P) cannot be directly applied to (2, 2)-MTG(P) since it may contain cliques of size 4. Bereg et al. [11] showed 6-planarity of MLG(P). However, it does not hold in (2, 2)-MTG(P). In fact, there exists a point set P such that an edge crosses other seven edges in (2, 2)-MTG(P). In the point set P shown in Fig. 8, the parameters δ, δ, ε, and h are defined to satisfy the conditions in Lemma 8. We can confirm that edge p1p4 crosses seven other edges. Thus, new techniques are necessary to address the upper bound of the maximum total number of edge crossings of (2, 2)-tight graph (and also for general (k, ℓ)-tight graph). For the lower bound, we believe that our technique of arranging different units made under the same design is promising for general k and ℓ.

A gap also exists for the maximum geometric thickness of MLG(P). Higashikawa et al. [12] gave a suggestion for improving the upper bound. A planar triangle-free graph is 3-colorable, and the edge-crossing graph of MLG(P) is triangle-free for any semi-generic point set P. Thus, if we prove the planarity of the edge-crossing graphs, the upper bound becomes 3 (i.e., we have the matching upper and lower bounds). As for the maximum geometric thickness for (2, 2)-MTG(P), the upper bound is open. Moreover, it is not known whether the edge-crossing graph of (2, 2)-MTG(P) is triangle-free or not.

**Acknowledgments**

This research was partially supported by JSPS KAKENHI Grant Numbers 20H05964, 22H03549, 22H00513, 23K17158, 23H03349, and 23H03350.

**References**


Yuki Kawakami received the B.Eng. degree from Hokkaido University, Sapporo, Japan, in 2022. He is currently a master student at graduate school of information science and technology, Hokkaido University. His current research interests include computational geometry.

Shun Takahashi received the B.Eng. degree from Hokkaido University, Sapporo, Japan, in 2022. He is currently a master student at graduate school of information science and technology, Hokkaido University. His current research interests include combinatorics on strings.

Kazuhisa Seto received the B.Eng., M.Info., and Ph.D. degrees in Informatics from Kyoto University in 2008, 2010, and 2013, respectively. He is now an associate professor at Hokkaido University. His research interests include satisfiability problems and circuit complexity.

Takashi Horiyama received the B.E. and M.E. degrees in information science and Ph.D. in informatics from Kyoto University, Kyoto, Japan in 1995, 1997 and 2004, respectively. He was a research associate at Nara Institute of Science and Technology from 1999, a research associate at Kyoto University from 2002, and an associate professor at Saitama University from 2007. Since 2019, he is a professor at Hokkaido University. His current interests include computational geometry and algorithm design.

Yuki Kobayashi is a researcher of computational design born in Aichi, Japan. He has studied architectural design with Prof. S. Takamatsu and theoretical computer science with Prof. N. Katoh at Kyoto University. He was awarded Young CAADRIA Award 2014. He has worked as an assistant professor at Tokyo Institute of Technology since 2015. Currently he is a lecturer at Osaka Metropolitan University.

Yuya Higashikawa is an associate professor in Graduate School of Information Science at University of Hyogo. He was an assistant professor in Faculty of Science and Engineering at Chuo University from 2015 to 2018, and also a JSPS Research Fellowship for Young Scientists (PD) at Kyoto University from 2014 to 2015. He received the B. Eng, M. Eng. and Dr. Eng. degrees from Kyoto University in 2008, 2010 and 2014, respectively. His interest is on design and analysis of algorithms, combinatorial optimization, discrete mathematics, computational geometry, and operations research. He is a member of IPSJ and OR Soc. Japan.

Naoki Katoh received the B.Eng., M.Eng. and Dr.Eng. degrees in Applied Mathematics and Physics from Kyoto University in 1973, 1975 and 1981, respectively. He was a Assistant Professor during 1981–1982, an Associate Professor during 1982–1990, and a Professor during 1990–1997, respectively, at Department of Management Science of Kobe University of Commerce. And also he was a Professor during 1997–2015, 2015–2019, and 2019–, at Department of Architecture and Architectural Systems of Kyoto University, Graduate School of Science and Technology of Kwansei Gakuin University, and School of Social Information Science of University of Hyogo, respectively. Currently he is a Professor in Graduate School of Information Science at University of Hyogo. He was rewarded Hao Wang Award in 2000. His research interests include the design and analysis of combinatorial and geometric algorithms, data mining and architectural information systems.
A Sequential Approach to Detect Drifts and Retrain Neural Networks on Resource-Limited Edge Devices

Kazuki SUNAGA(a), Takeya YAMADA(b), Nonmembers, and Hiroki MATSUTANI(c), Member

SUMMARY A practical issue of edge AI systems is that data distributions of trained dataset and deployed environment may differ due to noise and environmental changes over time. Such a phenomenon is known as a concept drift, and this gap degrades the performance of edge AI systems and may introduce system failures. To address this gap, retraining of neural network models triggered by concept drift detection is a practical approach. However, since available compute resources are strictly limited in edge devices, in this paper we propose a fully sequential concept drift detection method in cooperation with an on-device sequential learning technique of neural networks. In this case, both the neural network retraining and the proposed concept drift detection are done only by sequential computation to reduce computation cost and memory utilization. We use three datasets for experiments and compare the proposed approach with existing batch-based detection methods. It is also compared with a DNN-based approach without concept drift detection. The evaluation results of the proposed approach show that the proposed method is capable of detecting each of four concept drift types. The results also show that, while the accuracy is decreased by up to 0.9% compared to the existing batch-based detection methods, it decreases the memory size by 88.9%–96.4% and the execution time by 45.0%–87.6%. As a result, the combination of the neural network retraining and the proposed concept drift detection method is demonstrated on Raspberry Pi Pico that has 264 kB memory.

key words: edge AI, concept drift, on-device learning, OS-ELM

1. Introduction

With the rapid spread of AI (Artificial Intelligence) and IoT (Internet-of-Things) technologies, the number of IoT devices connected to the Internet continues to grow significantly. In cloud-based AI systems, IoT devices typically collect data at deployed edge environments and send the data to datacenters via the Internet. In this case, IoT devices focus on the data collection, and cloud servers are in charge of big data analysis and sophisticated machine learning tasks using plenty of compute resources. In addition, edge intelligence [1] in which some machine learning tasks such as performance and efficiency of edge devices have been improved significantly. Although conventional edge AI systems focus on prediction tasks, recently an on-device learning approach of neural networks is proposed for resource-limited IoT devices [2].

However, there are some limitations on such on-device learning approaches. First, there is a limitation on computation power and cost. Since they are often battery-powered, low-power consumption is required. In addition, deployed environments around the edge devices may change over time. That is, data distribution observed by the edge devices may shift as time goes by. For example, data distributions of trained dataset and deployed environment may differ due to noise and environmental changes. This gap degrades the performance of edge AI systems and may introduce system failures. To address this gap, a concept drift detection is a well-known approach [3].

Since edge devices are resource-limited, in this paper we propose a lightweight concept drift detection method for resource-limited edge devices. The contributions of this paper are as follows.

- We propose a fully sequential concept drift detection method to be combined with the on-device sequential learning approach of neural networks.
- Since both the neural network retraining and the proposed concept drift detection are done only by sequential computation, we demonstrate that the combined approach is implemented on Raspberry Pi Pico that has 264 kB memory.

The proposed approach is compared to existing concept drift detection methods in terms of accuracy using practical datasets. It is compared with a DNN-based approach without concept drift detection. It is also evaluated in terms of execution time and memory utilization on edge devices.

The rest of this paper is organized as follows. Section 2 overviews concept drift detection methods. Section 3 explains the proposed detection method. Section 4 describes the experimental setup including the datasets, counterparts, and platforms. Section 5 shows the evaluation results in terms of the accuracy, execution time, and memory utilization on edge devices. Section 6 concludes this paper.

2. Background and Related Work

2.1 Concept Drift Types

A concept drift [5] is known as a phenomenon where statistical properties of target data change over time. It is sometimes
caused by changes on hidden variables which cannot be observed directly. There are various types of concept drifts, and representative ones [5] are illustrated in Fig. 1. In the figure, the vertical and horizontal axes represent data distribution and elapsed time, respectively. The sudden drift is a concept drift in which a data distribution changes suddenly. In the sudden drift, an old data distribution before the concept drift does not appear after the concept drift. The gradual drift is a concept drift in which an old data distribution is gradually replaced with a new data distribution. Both the old and new distributions appear during the concept drift. In the incremental drift, the data distribution is incrementally shifted from an old distribution to a new distribution during the concept drift. In the reoccurring drift, after the data distribution has been changed to a new one, the old data distribution reoccurs.

2.2 Concept Drift Countermeasures

There are various approaches to address the concept drifts, and they can be classified into active approaches and passive approaches [6]. In this paper, a machine learning model that solves classification or regression tasks is called a “discriminative model”, and a model that detects concept drifts is called a “detection model”. An example of their relationship is illustrated in Fig. 2.

2.2.1 Passive Approach

In the passive approach, a discriminative model is retrained whenever a new data arrives. Since the discriminative model can be always trained with the latest data, its accuracy tends to be high. It does not use any detection model. However, it requires computation resource and memory to retrain a discriminative model. This may limit its application to resource-limited edge devices. To enable retraining of neural networks on resource-limited edge devices, OS-ELM [7] is used as an online sequential learning algorithm of neural networks that have a single hidden layer in ONLAD [8]. Since it sequentially updates weight parameters of neural networks when new training samples come, the memory utilization is quite small compared to batch training algorithms. ONLAD is classified as a passive approach. Specifically, OS-ELM is combined with a forgetting mechanism to forget old collected data and follow a concept changes quickly. The training batch size is fixed to one so that pseudo inverse operation of matrixes can be eliminated.

2.2.2 Active Approach

In the active approach, a machine learning model is retrained only when a concept drift is detected. It thus requires a detection model in addition to a discriminative model. Since it may be difficult to address all the concept drift types introduced in Sect. 2.1 at the same time for any applications, existing active approaches often focus on some specific concept drift types [9]. There are several detection models and they can be broadly classified into two methods below and their ensemble [5].

The first detection method is an error-rate based drift detection method. This method monitors prediction errors of a discriminative model using labeled teacher data, and it detects a concept drift when the error-rate exceeds a threshold value. DDM [10] and ADWIN [11] are the error-rate based drift detection methods. DDM (Drift Detection Method) uses two threshold levels: warning level and drift level. When an error-rate reaches the warning level, it starts a retraining of a discriminative model. When the error-rate reaches the drift level, the retrained discriminative model replaces the old model. The number of samples required to judge concept drifts, which is called window size, is fixed at DDM. In ADWIN (Adaptive Windowing), the window size is adaptively adjusted based on test statistics. Since these approaches need a labeled teacher dataset to detect a concept drift, they are not suited to resource-limited edge devices with a limited memory capacity.

The second detection method is a distribution-based drift detection method. Quant Tree [12] and SPLL [13] are the distribution-based drift detection methods. Quant Tree detects concept drifts by using a histogram. Although the size of histogram increases as the number of features (the number of dimensions) increases in typical histogram-based detection methods, it can be fixed in Quant Tree. Also, the test statistics to detect concept drifts does not depend on training and test datasets. SPLL detects concept drifts by using semi-parametric log-likelihood. Input data samples are clustered by using k-means method, and then the resultant clusters
3. Proposed Detection Method

Figure 2 illustrates an overview of the proposed concept drift detection method. Specifically, the proposed lightweight detection method is combined with the on-device sequential learning approach of neural networks\(^2\) as a discriminative model. In this section, the discriminative model assumed in this paper is briefly illustrated first. Then the proposed detection method is explained.

3.1 Discriminative Model

Assume data can be classified into one of multiple labels. In the discriminative model, the same number of OS-ELM based neural networks (called “instances”) as the number of labels in the training dataset are used. For each label in the training dataset, a discriminative model instance is trained with the data belonging to the label. Each discriminative model instance forms an autoencoder\(^14\) for unsupervised anomaly detection. That is, the numbers of input and output layer nodes of the discriminative model instances are the same, and each instance is trained so that its output can reconstruct a given input data with a smaller number of hidden nodes.

In the test phase, a reconstruction error (anomaly score) is calculated by comparing the input and output data in each model instance, and the smallest anomaly score among all the instances is used as the final prediction result (see lines 6 and 7 in Algorithm 1). For the sequential training, a single model instance that outputs the smallest anomaly score (i.e., the “closest” instance) trains the input data sequentially. We employ this architecture for anomaly detection on multiple normal patterns as in \(^2\).

3.2 Concept Drift Detection

At first, the proposed concept drift detection method calculates a centroid of trained data for each label. It records the same number of trained centroids as the number of labels. Then it sequentially updates the centroid with recent test data for each label whenever it predicts. It maintains test centroids in addition to the trained centroids. A drift rate is calculated based on a sum of the distance between the trained centroid and corresponding test centroid for each label (see line 14 in Algorithm 1). Then a concept drift is detected when the drift rate exceeds a pre-determined threshold value.

The proposed method is illustrated below. In the initial training phase, a discriminative model is trained with initial samples. Assume there are initial samples which are labeled as one of three different colors as shown in Fig. 3 (a). In the case of unsupervised learning, it is assumed that these initial samples can be labeled with a clustering algorithm such as k-means. A centroid of the initial samples is calculated for each label, as shown in Fig. 3 (b). In this figure, the centroids are represented as deeper-colored points. They are referred as “trained centroids”. The proposed method thus records the same number of trained centroids as the number of labels in the initial training phase.

In the prediction phase, the discriminative model predicts a label for each test sample. The centroids are sequentially updated based on each test sample and its predicted label. They are referred as “test centroids”. When the centroids are calculated, it is possible to assign a higher weight to a newer sample (a lower weight to an older sample) so that they can represent “recent” test centroids. Assume a new test sample comes and it is labeled as “blue” by the discriminative model. The recent test centroid of “blue” label is then sequentially updated. The data distribution is relatively stable before a concept drift occurs. We can thus expect that distances between the trained centroids and the recent test centroids are small as shown in Fig. 3 (c).

Next, let us illustrate a case when a concept drift happens. Assume the data distribution is changed and new test samples appear as shown in yellow circles in Fig. 3 (d). If these new test samples are labeled as “blue” by the discriminative model, the recent test centroid of “blue” label is moved to near the yellow circles in Fig. 3 (d). The distance between the test centroid and the trained centroid increases due to the new data distribution. A drift rate is calculated based on a sum of these distances, and a concept drift is detected when the drift rate exceeds a pre-determined threshold value. Please note that in the proposed method, the distances can be sequentially updated when it predicts. It thus requires
much less memory than batch-based concept drift detection methods introduced in Sect. 2.2.2.

Algorithm 1 shows the proposed detection method mentioned above. The inputs to the proposed algorithm are a discriminative model, a test dataset, the number of classes \( C \), the number of dimensions \( D \), recent coordinates \( \text{cor} \), trained coordinates \( \text{train\_cor} \), numbers of samples in each label, a window size \( W \), error threshold \( \theta_{\text{error}} \), and drift threshold \( \theta_{\text{drift}} \).

Algorithm 2 is the discriminative model reconstruction. The inputs to the proposed algorithm are a discriminative model \( \text{model} \), the number of classes \( C \), the number of dimensions \( D \), the number of samples in each label, a window size \( W \), number to search initial coordinates \( N_{\text{search}} \), number to update coordinates \( N_{\text{update}} \), and number to finish reconstruction \( N \).

Algorithm 3 is the label coordinates initialization. The inputs to the proposed algorithm are the number of classes \( C \), the number of dimensions \( D \), and the window size \( W \).
Based on the selected label, its centroid is then sequentially updated. This part is very similar to a sequential k-means algorithm. The detail implementation of Update_Coord() function is shown in Algorithm 4. Please note that since there is a possibility that initial coordinates selected by Init_Coord() are outliers, the centroids are further refined by Update_Coord() function using more samples.

In the third part, the discriminative model is retrained. In lines 8 and 9, a label that minimizes the distance between the incoming data and centroid of the label is selected, and then the discriminative model is sequentially updated by the incoming data and the selected label.

The fourth part is similar to the third part, but as shown in lines 11 and 12, a label is predicted by the discriminative model being retrained, and then the discriminative model is sequentially updated by the incoming data and the predicted label.

3.4 Threshold

The threshold value $\theta_{drift}$ is used to detect a concept drift in line 17 of Algorithm 1. For each trained sample, a distance between the sample and the centroid of its predicted label is calculated and stored in $dist$ array. In this paper, $\theta_{drift}$ is calculated based on the mean and standard deviation of $dist$ array, as shown below:

$$
\mu = \frac{1}{N}\sum_{i \in N} dist[i]
$$

$$
\theta_{drift} = \mu + z\sqrt{\frac{1}{N}\sum_{i \in N}(dist[i] - \mu)^2},
$$

(1)

where $N$ is the number of trained samples and $dist[i]$ is a distance between the $i$-th sample and the centroid of a cluster the $i$-th sample belongs to. $z$ is a tuning parameter and we simply assume $z = 1$ in this paper.

4. Evaluation Setup

This section describes datasets, counterparts of the proposed method, and platforms for the evaluations.

4.1 Datasets

Three datasets used in the evaluations are described below.

4.1.1 Synthetic Dataset

First, we evaluate the proposed method using sine and cosine waves to demonstrate its capability to detect each type of concept drifts shown in Fig. 1. Each data has 100 features, and a random noise of $[-0.1, 0.1]$ is added to each feature. We use $\sin \theta$, $\cos \theta$, and $-\sin \theta$ as a training dataset. The following four patterns are used as test datasets.

1. The first dataset focuses on a sudden drift. We use $\sin \theta$, $\cos \theta$, and $-\sin \theta$ as test data before a concept drift, and those with a phase shift of 135 degrees for each as test data after the concept drift. The concept drift occurs at the 1998th data point.

2. The second dataset focuses on a gradual drift. The dataset used is almost the same as the first dataset, but in the second dataset, both the patterns are mixed between the 1998th data point and the 2808th data point so that a gradual drift can be reproduced.

3. The third dataset focuses on an incremental drift. The dataset used is almost the same as the first dataset, but in the third dataset, the phase gradually shifts in a continuous manner from the 1998th data point to the 2808th data point so that an incremental drift can be reproduced.

4. The fourth dataset focuses on a reoccurring drift. The dataset used is almost the same as the first dataset, but in the fourth dataset, data after the concept drift appear only between the 1998th data point and the 2808th data point. Those before the concept drift reoccur after the 2808th data point so that reoccurring drift can be reproduced.

4.1.2 NSL-KDD Dataset

NSL-KDD [16] is a famous dataset which can be used to evaluate network intrusion detection methods. As data distribution of the dataset shifts from the training data to the test data, it can also be used for evaluating concept drift detection methods [3]. This change in data distribution is attributed to changes in attack patterns. The original dataset contains a lot of samples with 23 labels. In this paper, we use selected samples labeled with “normal” and “satan” for training, and “normal” and “ipsweep” for testing. We further select 1523 and 13709 samples for the initial training and test, respectively. A concept drift occurs at the 6110th data point.

4.1.3 Cooling Fan Dataset

The cooling fan dataset [17] contains vibration patterns of various cooling fans measured by an industrial accelerometer PCB M607A11. These vibration patterns were measured at a silent environment and a noisy environment near a ventilation fan. The vibration pattern is represented as a frequency spectrum ranging from 1 Hz to 511 Hz. Thus, the number of features is 511 in the cooling fan dataset. The data of four vibration patterns (0 rpm, 1500 rpm, 2000 rpm, and 2500 rpm) are used. Specifically, those observed in a silent environment are used as a training dataset, and those
Table 1 Detector and discriminative model parameter settings in synthetic dataset

<table>
<thead>
<tr>
<th>Train methods</th>
<th>Proposed</th>
<th>Quant Tree</th>
<th>SPLL</th>
<th>Baseline</th>
<th>ONLAD</th>
<th>DNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layers &amp; nodes</td>
<td>OS-ELM (sequential)</td>
<td>OS-ELM (w/ forgetting)</td>
<td>OS-ELM (w/ forgetting)</td>
<td>Backprop &amp; SGD (mini-batch)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hyperparameters of discriminative model</td>
<td>instances : 2, ( N = 400 ), ( N_{\text{update}} = N/5 )</td>
<td>instances : 2, ( N = 400 ), ( N_{\text{update}} = N/5 )</td>
<td>instances : 2, ( N = 400 ), ( N_{\text{update}} = N/5 )</td>
<td>batch : 10, epoch : 10, learning rate : 0.005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drift detection</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hyperparameters of detector</td>
<td>W : 100, batch : 400, histograms : 32</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2 Detector and discriminative model parameter settings in NSL-KDD dataset

<table>
<thead>
<tr>
<th>Train methods</th>
<th>Proposed</th>
<th>Quant Tree</th>
<th>SPLL</th>
<th>Baseline</th>
<th>ONLAD</th>
<th>DNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layers &amp; nodes</td>
<td>OS-ELM (sequential)</td>
<td>OS-ELM (w/ forgetting)</td>
<td>OS-ELM (w/ forgetting)</td>
<td>Backprop &amp; SGD (mini-batch)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hyperparameters of discriminative model</td>
<td>instances : 2, ( N = 400 ), ( N_{\text{update}} = N/5 )</td>
<td>instances : 2, ( N = 400 ), ( N_{\text{update}} = N/5 )</td>
<td>instances : 2, ( N = 400 ), ( N_{\text{update}} = N/5 )</td>
<td>batch : 10, epoch : 10, learning rate : 0.005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drift detection</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hyperparameters of detector</td>
<td>W : 100, batch : 400, histograms : 32</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3 Detector and discriminative model parameter settings in fan dataset

<table>
<thead>
<tr>
<th>Train methods</th>
<th>Proposed</th>
<th>Quant Tree</th>
<th>SPLL</th>
<th>Baseline</th>
<th>ONLAD</th>
<th>DNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layers &amp; nodes</td>
<td>OS-ELM (sequential)</td>
<td>OS-ELM (w/ forgetting)</td>
<td>OS-ELM (w/ forgetting)</td>
<td>Backprop &amp; SGD (mini-batch)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hyperparameters of discriminative model</td>
<td>instances : 2, ( N = 400 ), ( N_{\text{update}} = N/5 )</td>
<td>instances : 2, ( N = 400 ), ( N_{\text{update}} = N/5 )</td>
<td>instances : 2, ( N = 400 ), ( N_{\text{update}} = N/5 )</td>
<td>batch : 10, epoch : 10, learning rate : 0.005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drift detection</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hyperparameters of detector</td>
<td>W : 100, batch : 400, histograms : 32</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.2 Evaluated Methods

In this paper, the following six combinations are evaluated and compared as concept drift countermeasures.

1. Detector: the proposed method, Discriminative model: OS-ELM
2. Detector: none (no concept drift detection), Discriminative model: OS-ELM
3. Detector: Quant Tree, Discriminative model: OS-ELM
4. Detector: SPLL, Discriminative model: OS-ELM
5. Detector: none, Discriminative model: ONLAD (OS-ELM with a forgetting mechanism)
6. Detector: none, Discriminative model: DNN (3 hidden layers)

The first method is our proposal, and the second method is a baseline without concept drift detection. The first, third, and fourth methods are classified as the active detection approach, while the fifth method is the passive approach. The sixth method is a DNN-based approach without concept drift detection.

With ONLAD that uses OS-ELM and a lightweight forgetting mechanism. By adding the DNN-based approach, we also compare OS-ELM with a DNN.

The synthetic dataset is used to verify the operation of the proposed method. Table 1 shows the hyperparameters used for the synthetic dataset. The numbers of input and output layer nodes of the model are 100 and that of the hidden layer nodes is 22. Table 2 shows the hyperparameters used for the NSL-KDD dataset. In the OS-ELM based discriminative model, the numbers of input and output layer nodes are 37 and that of the hidden layer nodes is 22. In Quant Tree, the batch size is 400 and the number of histograms is 32. In SPLL, the batch size is 400. In ONLAD, the numbers of input, hidden, and output layer nodes are the same as those of the OS-ELM model. The forgetting rate is 0.99. The DNN model has three hidden layers and is trained by a back-propagation algorithm with a SGD optimizer. Also, Table 3 shows the hyperparameters used for the cooling fan dataset. The six methods are set up accordingly.

4.3 Evaluation Platforms

The above-mentioned six methods are running on Raspberry Pi 4 Model B [18]. In addition, due the memory size constraints, only the proposed method is demonstrated on Raspberry Pi Pico [19]. These methods are evaluated in terms of memory utilization in Raspberry Pi 4 Model B. The execution time breakdown of the proposed method is further analyzed in Raspberry Pi Pico. Table 4 shows the specifications of Raspberry Pi 4 Model B and Raspberry Pi Pico.
5. Evaluation Results

This section shows evaluation results of the six methods listed in Sect. 4.2 in terms of the accuracy, delay to detect concept drifts, memory utilization, and execution time.

5.1 Synthetic Dataset

First, the proposed concept drift detection method is evaluated for the four concept drift types. Figure 4 (a), (b), (c), and (d) are the evaluation results for sudden drift, gradual drift, incremental drift, and reoccurring drift, respectively. The X-axis indicates the number of data, and Y-axis is the error between input and output in each graph. The first vertical bar at the 1998th data point in Fig. 4 represents an occurrence of concept drift, and the second vertical bar at the 2808th data point represents the end of concept drift in the cases of (b), (c), and (d). Please note that the second bar in (d) indicates the data distribution before the concept drift occurred has re-occurred. It can be seen from each figure that loss is sharply increased when a concept drift occurs. In the sudden drift, it is observed that loss is decreased by learning after the drift detection. In the gradual drift and incremental drift, there may appear “intermediate concept” during the transformation, so if the intermediate concept is learned, the loss may be high even after the end of drift point. However, in the proposed method, a retraining flag is set based on the loss. Therefore, it can be observed that the proposed method is able to learn the data after the drift even if an intermediate concept is learned. In the reoccurring drift, the emphasis is on how to find the best matched historical concept in a short time. Since, the proposed method uses centroids of the data, the best matched one can be selected simply by storing these centroids and comparing them with centroids of the most recent data.

5.2 NSL-KDD Dataset

The six methods are evaluated with NSL-KDD dataset. Figure 5 shows evaluation results in terms of the accuracy of the discriminative model. Table 5 summarizes the accuracy and the delay to detect a concept drift. The delay means the number of samples needed to detect a concept drift after the concept drift actually happens. A vertical bar at the 6110th data point in Fig. 5 shows a concept drift. The results also show that the proposed method can detect the concept drift as well as the batch-based Quant Tree and SPLL methods in the NSL-KDD dataset. After the concept drift is detected, the accuracy of the proposed method becomes high compared to the baseline method that does not detect concept drifts. As a result, the proposed method outperforms the baseline method without concept drift detection, ONLAD with a forgetting mechanism, and DNN by 25.4%, 41.9%, and 21.7%, respectively, while the accuracy of the ONLAD model gradually decreases even before the concept drift happens.

The results also show that the proposed method can detect the concept drift as well as the batch-based Quant Tree and SPLL methods in the NSL-KDD dataset. After the concept drift is detected, the accuracy of the proposed method becomes high compared to the baseline method that does not detect concept drifts. As a result, the proposed method outperforms the baseline method without concept drift detection, ONLAD with a forgetting mechanism, and DNN by 25.4%, 41.9%, and 21.7%, respectively, while the
accuracy is decreased by up to 0.9% and 0.8% compared to the batch-based SPLL and Quant Tree methods, respectively. Please note that the proposed method needed less samples to detect the concept drift compared to the batch-based Quant Tree and SPLL methods.

5.3 Cooling Fan Dataset

Next, we show the results of the evaluation on the cooling fan dataset in Fig. 6. Table 6 summarizes the accuracy and the delay to detect a concept drift. A vertical bar at the 400th data point in Fig. 6 shows a concept drift as well as Fig. 5. In the cooling fan dataset, the accuracy becomes high in the methods that detect the concept drifts. As a result, the proposed method outperforms the baseline method without concept drift detection, ONLAD with a forgetting mechanism, and DNN by 33.6%, 33.5%, and 32.1%, respectively, and the accuracy is increased by up to 0.5% and 0.3% compared to the batch-based SPLL and Quant Tree methods, respectively. It also needed less samples to detect the concept drift compared to the batch-based Quant Tree and SPLL methods.

5.4 Memory Utilization

Table 7 shows the evaluation results in terms of the memory utilization on Raspberry Pi 4 Model B. The cooling fan dataset is used for the memory size evaluation; in this case, the batch size of the Quant Tree and SPLL methods is 235 while it is one in the proposed method.

The results show that the proposed method uses much less memory size compared to the batch-based Quant Tree and SPLL methods. Specifically, the proposed method decreases the memory utilization by up to 96.4% and 88.9% compared to SPLL and Quant Tree, respectively. This is because in the batch-based concept drift detection methods, data samples are stored in the device memory to detect concept drifts, while the proposed method processes data samples one by one and detects concept drifts sequentially; thus, the proposed method does not store past samples in the device memory.

Please note that since RAM size of Raspberry Pi Pico is only 264 kB as shown in Table 4, the batch-based Quant Tree and SPLL methods cannot operate on Raspberry Pi Pico. It is known that Raspberry Pi Pico is available from only $4 [19]. The memory size reduction by the proposed approach is beneficial in terms of the hardware cost, because our proposed system can be implemented on this low-cost device. In addition, the memory size reduction by the proposed approach can extend the applicable range of microcontrollers. For example, ultra-low power products of STMicroelectronics STM32 microcontrollers include several series, such as STM32L0, STM32L4, STM32L4+, STM32L5, and STM32U5 [21]. Especially, in the cases of STM32L4, STM32L4+, and STM32L5 series, their SRAM sizes typically range from 40 kB to 640 kB. Thus, our approach can be implemented on a wider range of these microcontrollers compared to the counterparts. In Sect. 5.5, only the proposed method shows the execution time on Raspberry Pi Pico in addition to that on Raspberry Pi 4 Model B.

5.5 Execution Time

Tables 8 and 9 show evaluation results in terms of the execution times on Raspberry Pi 4 Model B and Raspberry Pi Pico, respectively. The same cooling fan dataset is used for this evaluation.

Table 8 shows the execution time to process the cooling fan dataset that contains 1600 samples in total. As shown, the execution time of the proposed method is much less than that of Quant Tree and SPLL. Specifically, the proposed method decreases the execution time by up to 87.6% and 45.0% compared to SPLL and Quant Tree, respectively. Since SPLL executes k-means clustering, the execution time of SPLL is increased compared to the others. In the case of the proposed approach, there is a possibility to meet a timing constraint with a lower operating frequency or less compute resources. Since a higher operating frequency and more compute resources tend to increase the power consumption and hardware cost, the proposed approach is beneficial in terms of the power consumption and miniaturization of tar-
get devices. Although the proposed method increases the execution time by 58.2% compared to the baseline method without concept drift detection, it significantly improves the accuracy compared to the baseline as shown in Figs. 5 and 6. Thus, the proposed method is an attractive option when concept drifts are expected in target applications.

Table 9 further analyzes the execution time breakdown for a single sample by the proposed method on Raspberry Pi Pico. In the table, “Label prediction” and “Distance computation” are corresponding to lines 6 and 14 in Algorithm 1, respectively. “Model retraining without label prediction” is done in lines 8 and 9, while “Model retraining with label prediction” is done in lines 11 and 12 in Algorithm 2. “Label coordinates initialization” is Init_Coord() function in Algorithm 3, and “Label coordinates update” is Update_Coord() function in Algorithm 4. The results show that the additional computation time for the concept drift detection is less than the label prediction time of the discriminative model. Please note that the latency is within a few hundred milliseconds even in such a low-end edge device.

6. Conclusions

In edge AI systems, data distributions of trained dataset and deployed environment may differ due to noise and environmental changes over time. This gap degrades the performance of edge AI systems and may introduce system failures. To address this gap, retraining of neural network models triggered by concept drift detection is a practical approach. As practical concept drift detection, error-rate based detection methods and distribution-based detection methods have been used. However, since such a batch-based processing and use of labeled teacher dataset are not suited for resource-limited edge devices, in this paper we proposed a fully sequential concept drift detection method in cooperation with the on-device sequential learning technique of neural networks. Both the neural network retraining and the proposed concept drift detection are done only by sequential computation to reduce computation cost and memory utilization. The proposed approach was compared with existing concept detection methods and a DNN model without concept drift detection.

Evaluation results of the proposed approach showed that while the accuracy is decreased by up to 0.9% compared to existing batch-based detection methods, it decreases the memory size by 88.9%–96.4% and the execution time by 45.0%–87.6%. Thanks to these significant decreases on the memory size and computation cost, the combination of the neural network retraining and the proposed concept drift detection method was demonstrated on Raspberry Pi Pico that has 264 kB memory. A possible extension of this work is a combination of multiple detection models with different window sizes to address more complicated concept drift behaviors since it is expected that the sizes of detection models are less dominant compared to the discriminative model. We are also planning to apply the proposed system to more practical applications, such as anomaly detection on rotating machines as in [22].

Acknowledgments

This work was partially supported by JST AIP Acceleration Research JPMJCR23U3, Japan.

References


Table 8 Execution time (sec) for 1600 samples on Raspberry Pi 4 Model B

<table>
<thead>
<tr>
<th></th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quant Tree</td>
<td>7.50</td>
</tr>
<tr>
<td>SPLL</td>
<td>33.35</td>
</tr>
<tr>
<td>Baseline (no concept drift detection)</td>
<td>2.61</td>
</tr>
<tr>
<td>Proposed method</td>
<td>4.13</td>
</tr>
</tbody>
</table>

Table 9 Execution time breakdown (msec) for 1 sample by proposed method on Raspberry Pi Pico

<table>
<thead>
<tr>
<th></th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label prediction</td>
<td>148.87</td>
</tr>
<tr>
<td>Distance computation</td>
<td>10.58</td>
</tr>
<tr>
<td>Model retraining without label prediction</td>
<td>25.42</td>
</tr>
<tr>
<td>Model retraining with label prediction</td>
<td>166.65</td>
</tr>
<tr>
<td>Label coordinates initialization</td>
<td>25.59</td>
</tr>
<tr>
<td>Label coordinates update</td>
<td>6.05</td>
</tr>
</tbody>
</table>


Kazuki Sunaga received the B.E. degree from Keio University in 2023. He is currently a master course student in Keio University.

Takeya Yamada received the B.E. and M.E. degrees from Keio University in 2021 and 2023, respectively.

Hiroki Matsutani received the BA, M.E., and Ph.D. degrees from Keio University in 2004, 2006, and 2008, respectively. He is currently a professor in the Department of Information and Computer Science, Keio University. His research interests include the areas of computer architecture and interconnection networks.
Dataset of Functionally Equivalent Java Methods and Its Application to Evaluating Clone Detection Tools

Yoshiki HIGO\(^{a)}\), Senior Member

SUMMARY Modern high-level programming languages have a wide variety of grammar and can implement the required functionality in different ways. The authors believe that a large amount of code that implements the same functionality in different ways exists even in open source software where the source code is publicly available, and that by collecting such code, a useful data set can be constructed for various studies in software engineering. In this study, we construct a dataset of pairs of Java methods that have the same functionality but different structures from approximately 314 million lines of source code. To construct this dataset, the authors used an automated test generation technique, EvoSuite. Test cases generated by automated test generation techniques have the property that the test cases always succeed. In constructing the dataset, using this property, test cases generated from two methods were executed against each other to automatically determine whether the behavior of the two methods is the same to some extent. Pairs of methods for which all test cases succeeded in cross-running test cases are manually investigated to be functionally equivalent. This paper also reports the results of an accuracy evaluation of code clone detection tools using the constructed dataset. The purpose of this evaluation is assessing how accurately code clone detection tools could find the functionally equivalent methods, not assessing the accuracy of detecting ordinary clones. The constructed dataset is available at github (https://github.com/YoshikiHigo/FEMPDataset).

key words: functionally-equivalent methods, source code analysis, dataset, code clone

1. Introduction

Current programming languages have a rich syntax, and there are many ways for developers to implement the functionality they need. For example, in the case of Java, the for statement, the while statement, recursive functions, Stream, etc. can be used to perform repetitive processing. In the refactoring patterns proposed by Fowler\(^{[1]}\), the implementations before and after refactoring have the same external behavior, which means refactoring can be regarded as an implementation change of a functionality. Thus, there are countless ways to implement a certain functionality, and developers implement the necessary functionality according to their own preferences and/or the policies of their software development project.

The authors believe that there is a large amount of source code with the same functionality but different structures in open source software (hereafter referred to as SFDS code\(^{a)}\)), which can be a useful dataset for various studies in software engineering. For example, SFDS code can be used to evaluate code clone detection tools. Since it is desirable for source code that implement the same functionality to be detected as code clones, the performance of code clone detection tools can be evaluated by examining the degree to which SFDS code are detected as code clones. Moreover, by using SFDS code, we can investigate which implementations are superior in terms of running performance, such as memory usage and execution speed, and which implementations are superior in terms of software quality, such as ISO/IEC 25010\(^{[2]}\).

However, it is not easy to identify and collect SFDS code. It is not realistic to manually find SFDS code from open source software, and if existing code clone detection tools are used, only SFDS code that can be detected by existing code clone detection tools will be collected. Therefore, SFDS code collected in such a way cannot be used to evaluate code clone detection tools.

In this study, SFDS code to be collected are limited to pairs of methods that return the same output (return value) when the same inputs (arguments) are given (hereafter, FE methods\(^{b)}\)). The key idea of this study is to automatically obtain candidate pairs of FE methods using an automated test generation technique, EvoSuite by limiting the detection of SFDS code to the detection of FE methods. The obtained candidates of FE method pairs are manually verified to be truly functionally equivalent.

In this study, we selected IJADataset\(^{[3]}\) as the target for detecting FE method pairs in Java. IJADataset consists of approximately 2.74 million source files (a total of approximately 314 million lines of source code) and includes approximately 23 million methods. From this dataset, 13,710 candidate FE method pairs were automatically detected, of which 2,194 were visually investigated. As a result, a dataset of 1,342 FE method pairs was constructed. Our dataset also includes 852 method pairs that were determined not to be functionally equivalent by visual inspection. The constructed dataset is available at GitHub\(^{***}\).

We also evaluated code clone detection tools using the constructed dataset. As a result, we found that there are many FE method pairs that cannot be detected by the token-based clone detection technique, and that the detection techniques based on abstract syntax trees and deep learning have a strong tendency to incorrectly detect method pairs that are functionally inequivalent.

\(^{a)}\)Same-Functionality but Different Structures
\(^{b)}\)Functionally Equivalent
\(^{***}\)https://github.com/YoshikiHigo/FEMPDataset

Copyright © 2024 The Institute of Electronics, Information and Communication Engineers
2. Definition of FE Methods

In this study, two methods are considered to be functionally equivalent if they return the same output (return value) when the same inputs (arguments) are given. Many code clone detection techniques measure the similarity of the source code of the target methods to determine them as code clones [4]–[6], but this study does not use such code similarity to determine whether given two methods are functionally equivalent or not. There are also techniques that determine code clones by using the state of processing (main memory state) of the target methods by running them [7], but this study does not use such a calculation process to determine whether given two methods are functionally equivalent or not.

In research of code clones, detected code clones are often classified according to their similarity as follows.

**Type-1** The code snippets are entirely identical except for changes that may exist in the white spaces and comments.

**Type-2** The structure of the code snippets is the same while the identifiers’ names, types, white spaces, and comments differ.

**Type-3** In addition to changes in identifiers, variable names, data types, and comments, some parts of the code can be deleted or updated, or some new parts can be added.

**Type-4** Two code snippets have different texts but the same functionality.

FE methods detected in this study are code clones of a part of Type-4 according to the above classification.

3. Key Idea for Automatically Identifying Candidate FE Method Pairs

In this study, we automatically collect candidate pairs of FE methods by using the static features (method signatures) and dynamic behavior (test results) of Java methods. Whether or not the obtained candidate FE method pairs are truly functionally equivalent is investigated manually. Therefore, it is important to collect as many candidate FE method pairs as possible automatically.

The static features of the Java methods used in this study are the return value type and the parameter types. As the first step in obtaining candidate FE method pairs, methods with the same return value type and parameter types are classified into the same group. We do not consider throwing exceptions as part of static features since throwing an exception or not can be checked as dynamic behavior.

The dynamic behavior of the Java methods used in this study is the result (success/failure) of the execution of a given test case. Test cases are generated from all methods in the same group using an automated test generation technique, EvoSuite. Test cases generated by automated test generation techniques have the property that the test cases always succeed. For every pair of methods, the generated test cases are executed against each other to automatically determine if the method pair has the same behavior. The key idea of this research is that if method A succeeds in all tests generated from method B and method B succeeds in all tests generated from method A, then the behavior of methods A and B are equivalent to some extent and their functions may be equivalent.

Based on this key idea, a dataset of FE method pairs is constructed by automatically obtaining pairs of methods from a large amount of open source software that have been successfully executed for cross-testing and visually investigating them. Although methods with identical or similar source code may be functionally equivalent, such methods can be detected by existing code clone detection tools [8]. The objective of this study is to build a dataset of FE method pairs that are not similar in source code.

4. Procedure of Dataset Construction

In this study, the following procedure is used to construct a dataset of FE method pairs.

**STEP-1** classifying the methods included in the target projects.

**STEP-2** generating test cases from each method.

**STEP-3** running test cases against the other method of the pair to obtain candidate FE method pairs.

**STEP-4** browsing the source code of each candidate FE method pair to determine whether the pair is truly functionally equivalent.

Figure 1 shows an overview of the above construction procedure. **STEP-1** through **STEP-3** are performed automatically using a tool that we developed, and only **STEP-4** is performed manually. The details of each step are described below.

4.1 **STEP-1**

**STEP-1** analyzes the source code of the target projects to extract methods, and classifies the extracted methods based on their return value type and parameter types. When extracting methods, the following information is retrieved for each method and registered in the database.

- method name,
- return value type and parameter types,
- (original) source code,
- normalized source code,
- the number of statements and conditional predicates,
- file path,
- start line and end line.

In the normalized source code, all variable names have been renamed to special names. A normalization example is shown in Fig. 2(b). This normalization allows a group of methods that have the same structure but differ only in variables to be treated as a single method. This normalization eliminates the need to handle each method that has the same structure but differs only in variables individually after
Fig. 1  Steps to obtain pairs of functionally equivalent Java methods

STEP-2, allowing more efficient processing.

Not all methods that exist in the target project are extracted. In this study, methods with the following characteristics are not considered for extraction.

- Methods including reference types defined in neither java.lang nor java.util in their return value, parameters, or method bodies.
- Methods whose return type is void.
- Methods including only a single program statement.

The reason for using the first condition is that in the case of using reference types that are not in the java.lang package, it is necessary to write import statements at the beginning of the source file, or to write the reference types by their fully qualified names. In addition, if the reference type is not included in the standard Java library, it is necessary to prepare its class file (jar file), which requires further compilation preparation. The reason why the reference types included in the java.util package are considered as methods for extraction is because frequently used types such as java.util.List and java.util.Set are included in this package. The authors believe that handling those reference types in the java.util package will dramatically increase the number of methods that are extracted.

The reason for using the second condition is that it is difficult to automatically determine which value within a method is the final results of the method’s computation for a method whose return value is void. If the return type is not void, the method’s return value can be judged to be the final results of the method’s computation.

The third condition is used because the source code of Java contains many setters and getters that have only a single program statement, and they are inappropriate as targets for the detection of SFDS code.

The classification of the extracted methods is based on the return value type and parameter types of the methods. Methods that have exactly the same return type and parameter types are classified into the same group. After all methods have been classified, if there are multiple methods in each group with exactly the same normalized source code, only one of them is retained in the group. The reason for this is that it is no doubt that methods with the identical code have the same behavior, and the detection of such method pairs with the same implementation is not appropriate for the purpose of this study. Note that groups consisting of only a single method are not subject to processing after STEP-2.

4.2 STEP-2

In STEP-2, each method is cut into a file and its test cases are generated. The following processing is performed when the methods are cut into files.

- Insert ‘import java.util.*;’ at the beginning of the file. This is to enable compilation even if the target
The files from which each method is extracted are compiled individually just before EvoSuite is executed. Since each file includes only a single method, the compilation fails if the method uses a class field or calls other methods that were originally defined in the same class. EvoSuite is not executed for methods that fail to compile.

In STEP-1, methods with no parameters are not explicitly excluded, but methods where five or more test cases are not generated are excluded in STEP-2. Since no more than five test cases are generated from a method with no parameters, all methods with no parameters are excluded in STEP-2.

4.3 STEP-3

In STEP-3, tests are executed mutually for each method that belongs to the same group and has successfully generated five or more test cases. In Fig. 1(c), test cases are executed against each other for three methods: Method-A, Method-B, and Method-C. Method-A succeeds in all test cases generated by Method-B, and Method-B also succeeds in all test cases generated by Method-A. Therefore, Method-A and Method-B are candidates for functionally equivalent method pairs. Method-C did not succeed in one of the test cases generated from Method-A, and it did not succeed in all the test cases generated from Method-B. Therefore, the pair of Method-A and Method-C, and the pair of Method-B and Method-C are not candidates for functionally equivalent method pairs.

4.4 STEP-4

STEP-4 visually checks the source code of the candidate functionally equivalent method pairs obtained with STEP-3 to see if they are truly pairs of methods with the same behavior.

5. FEMP Dataset

In this section, we describe the constructed dataset (hereafter referred to as FEMP dataset). The FEMP dataset was constructed for the source code included in IJADataset [3]. IJADataset consists of approximately 2.74 million source files, totaling 314 million lines of source code, and contains approximately 23 million methods. In STEP-1, 257,012 methods were extracted and they were classified into 14,030 groups. In STEP-2, five or more test cases were successfully generated from 27,759 methods. In STEP-3, we obtained 13,710 candidate functionally equivalent method pairs. The visual verification in STEP-4 was performed by three master’s course students belonging to the Graduate School of Information Science and Technology, Osaka University. The three students had programming experience using Java. First, the three students individually viewed the source code of each candidate pair and judged whether they

---

The code example for Fig. 2(a) is as follows:

```java
package com.intelij.openapi.util;

public class StringUtil extends StringUtil {
    @Contract(pure = true)
    public static @NonNull String repeat(@NonNull String s, int count) {
        if (count == 0) return "";
        StringBuilder sb = new StringBuilder(s.length() * count);
        for (int i = 0; i < count; i++) {
            sb.append(s);
        }
        return sb.toString();
    }
}
```

The code example for Fig. 2(b) is as follows:

```java
import java.util.*;

public class Target {
    public static void main(String[] args) {
        String variable = "Hello, World!";
        StringBuilder sb = new StringBuilder(variable.length() * 2);
        for (int i = 0; i < variable.length(); i++) {
            sb.append(variable.charAt(i));
        }
        System.out.println(sb.toString());
    }
}
```

The code example for Fig. 2(c) is as follows:

```java
import java.util.*;

public class Target {
    public static void main(String[] args) {
        String variable = "Hello, World!";
        StringBuilder sb = new StringBuilder(variable.length() * 2);
        for (int i = 0; i < variable.length(); i++) {
            sb.append(variable.charAt(i));
        }
        System.out.println(sb.toString());
    }
}
```
were equivalent or not. Next, the three students discussed each candidate pair that they evaluated differently, to determine whether they were functionally equivalent or not. However, since it is not realistic to visually check all 13,710 method pairs obtained in the STEP-3, the following procedure was used to extract a part of them.

1. Initialize the target method pair $P$ and the set of methods $M$ in $P$ as empty, respectively.
2. Arrange the 13,710 method pairs in ascending order by ID, and perform the following processes in order.
   • If both methods comprising the method pair are not included in $M$, the method pair is added to $P$ and both methods are added to $M$.
   • If at least one method comprising the method pair is contained in $M$, nothing is done.

After the above procedure, 2,194 method pairs were included in $P$. Those 2,194 method pairs all consisted of different methods. For those 2,194 method pairs, each student took 44 hours and 48 minutes, 33 hours and 19 minutes, and 43 hours and 25 minutes, respectively, to determine whether they were functionally equivalent or not. Subsequently, the discussion of each method pair, for which the evaluation was divided, took 9 hours and 28 minutes in total.

STEP-4 resulted in 1,342 of the 2,194 method pairs being determined to be functionally equivalent, with the remaining 852 not being functionally equivalent. The number of method pairs that were discussed by three students with different evaluations was 296. This dataset is available on Github.

Figure 3 is an example of a method pair that was determined to be functionally equivalent in STEP-4. Both methods implement the function of returning the smallest value among the three double values given as parameters.

The method `min` implements this function only using the `if`-statement, while the method `minimum` implements it using the `if`-statement and the ternary operator. On the other hand, Fig. 4 is an example of a method pair whose functions were determined to be not equivalent in STEP-4. Both methods determine the equivalence of two two-dimensional arrays given as parameters. The functionality differs in that the method `ArrayEquals` returns `false` if an empty array is given, while the method `equals` returns `true` if an empty array is given. EvoSuite generated nine test cases for the method `ArrayEquals` and eight test cases for the method `equals`, but no test cases were generated to find the functional differences between the two methods.

Next, we describe the characteristics of the method signatures in the FEMP dataset. The 1,342 functionally equivalent method pairs consist of 468 signatures, and the 852 functionally inequivalent method pairs consist of 338 signatures. In the dataset, 713 (53.1%) functionally equivalent and 317 (37.2%) functionally inequivalent method pairs include only primitive types such as `int` and `char` in their return value type and parameter types. Table 1 shows the top 10 most frequent signatures of functionally equivalent and functionally inequivalent method pairs in the FEMP dataset.

This table also shows that the functionally equivalent method pairs tend to be composed of primitive types only. This fact means that the proportion of method pairs that are determined to be functionally inequivalent in the dataset construction STEP-4 is higher when non-primitive types are included in the signatures. This indicates that it is more difficult to generate sufficient test cases when non-primitive types are included in the signature than when only primitive types are.

\textsuperscript{1}The 13,710 method pairs obtained in the STEP-3 have unique integer values as their IDs.

\textsuperscript{11}https://github.com/YoshikiHigo/FEMPDataset

\textsuperscript{111}FEMP dataset also includes test cases generated by EvoSuite.
6. Accuracy Evaluation of Clone Detection Tools

Herein, we describe the results of an accuracy evaluation of clone detection tools as an example of utilization of the FEMP dataset. The purpose of this evaluation is assessing how accurately code clone detection tools could find the functionally equivalent methods, not assessing the accuracy of detecting ordinary clones. In this evaluation, the 1,342 method pairs determined to be functionally equivalent in STEP-4 should be detected as clone pairs, and the 852 method pairs determined not to be functionally equivalent should not be detected as clone pairs. The former set is called EMP (Equivalent Method Pairs) and the latter set is called IMP (Inequivalent Method Pairs).

6.1 Clone Detection Tools to be Evaluated

Three clone detection tools were targeted in this evaluation: NIL [12], InferCode [13], and ASTNN [14].

NIL quickly identifies possible clone candidate method pairs using the inverted index and N-gram of the lexical sequence obtained from the target source code. It applies the longest common subsequence algorithm to those candidates to determine whether they are clone pairs or not. NIL is a tool for detecting large-variability clones, which are difficult to detect using conventional clone detection techniques. In the comparison with other clone detection tools LVMapper [15] and CCAligner [16] performed on two open source systems, the number of large-variability clones detected by NIL was 354 and 398 (86% and 88% precisions), whereas LVMapper detected 355 and 389 (64% and 60% precisions) and CCAligner detected 184 and 284 (43% and 49% precisions).

InferCode is a pre-training model using a convolutional neural network [17] based on abstract syntax trees and tree structures. It can be used for unsupervised learning tasks such as source code clustering and supervised learning tasks such as source code classification. Bui et al. have implemented clone detection in InferCode as an unsupervised learning task. In this comparison, we use InferCode as a clone detection technique based on unsupervised learning.

ASTNN is a model based on abstract syntax trees and regression neural networks. It learns lexical information contained in the target source code and syntactic information per program statement. For clone detection, ASTNN outputs a value between 0 and 1. Two input methods are determined to be a clone pair if this value is greater than or equal to a threshold value. Zhang et al. experimented on BigCloneBench and OJClone with a threshold value of 0.5. For OJClone, precision and recall were 98.9% and 92.7%, respectively. For BigCloneBench, the detection accuracy was evaluated for each clone category. The results showed that for Weakly Type-3/Type-4, which are clones with a syntactic similarity of less than 50%, precision and recall were 99.8% and 88.5%, respectively.

In the papers of NIL, InferCode, and ASTNN ([12]–[14]), those tools are evaluated using BigCloneBench [18] and OJClone [19]. BigCloneBench categorizes all functions into one of 43 different categories and then assumes that all methods that are assigned to a functionality are also clones of each other. The methods are not functionally equivalent and this is not claimed. Also, OJClone is a dataset built from competitive programming code, which is different in nature from the code included in OSS.

6.2 How to Detect Clones

Herein, we explain how we configured the three tools to detect code clones.

We installed NIL according to the instructions in GitHub of NIL. Clone detection was performed by outputting each method in EMP and IMP as a separate file and giving these two files to NIL as clone detection targets. In other words, we ran NIL 2,194 times (1,342 times for EMP and 852 times for IMP). Because some methods are small

https://github.com/kusumotolab/NIL

Table 1: Top 10 signatures of functionally equivalent and inequivalent method pairs

<table>
<thead>
<tr>
<th># pairs</th>
<th>Signature</th>
<th>Only primitive</th>
<th># pairs</th>
<th>Signature</th>
<th>Only primitive</th>
</tr>
</thead>
</table>
in size, the minimum number of lines and minimum number
of tokens for a method to be detected as a clone were both
set to 1\footnote{However, since NIL is internally processed using N-gram with
N=5, a minimum of five words are required to be detected as a clone.}. 

We installed InferCode according to the instructions
provided in GitHub\footnote{https://github.com/bdqghi/infercode} of InferCode. We used InferCode
to obtain vector data for each method in EMP and IMP,
and detected clones by changing the threshold of the cosine
similarity of the method pairs from 0 to 1 in 0.001 steps.

We obtained a pre-trained model of ASTNN according
to the description in ASTNN’s GitHub\footnote{https://github.com/zhangj111/astnn} Next, the 1,342
method pairs in EMP were divided into 10 blocks. The
blocks were classified so that the ratio of fine tuning, val-
ification, and test data was 8:1:1, and clone detection using
ASTNN was performed so that all blocks were once test data.
Since the output of ASTNN is a number between 0 and 1, the
threshold for being a clone was varied from 0 to 1 in 0.001
increments, and the results were evaluated\footnote{The hyperparameters used for fine tuning were: batch size:32,
epoch:5, learning size:2e-3, vector size of word2vec:128, hidden
dimension:100, encode dimension:128.}.

6.3 Evaluation Measures for Clone Detection Results

Three measures, recall\(^E\)(\(t\)), recall\(^I\)(\(t\)), and accuracy\((\(t\)) were used to evaluate the clone
detection results of tool \(t\) in this evaluation. First, the definitions of EMP\((t)\) and IMP\((t)\)
used in these evaluation measures are as follows.

**EMP\((t)\)** a set of method pairs in EMP that have been cor-
rectly determined to be functionally equivalent (detected as
clones) by detection tool \(t\).

**IMP\((t)\)** a set of method pairs in IMP whose functions are
correctly determined to be not functionally equivalent
detected as clones).

Using the above definitions, we define accuracy\((\(t\)
), recall\(^E\)(\(t\)), and recall\(^I\)(\(t\)) as follows. Note that \(|A|\)
denotes the number of elements in the set \(A\).

\[
\text{recall}\(^E\)(\(t\)) = \frac{|\text{EMP}(t)|}{|\text{EMP}|} \\
\text{recall}\(^I\)(\(t\)) = \frac{|\text{IMP}(t)|}{|\text{IMP}|} \\
\text{accuracy}(\(t\)) = \frac{|\text{EMP}(t)| + |\text{IMP}(t)|}{|\text{EMP}| + |\text{IMP}|}
\]

6.4 Detection Results

The detection results of NIL are shown in Table 2. The
recall\(^E\)(NIL) was 34.5% (=463/1,342), recall\(^I\)(NIL) was
72.54% (=618/852), and accuracy(NIL) was 49.27% (=1,081/2,194). These results indicate that clone detection
by NIL has many omissions in finding functionally equiva-
lent methods, and also includes some false positives.

The results of the InferCode evaluation are shown in Fig. 5 (a). When the threshold is 0.557
or less, recall\(^E\)(InferCode) is more than 99%, but recall\(^I\)(InferCode) is less than 0.35% at the same time.
In other words, almost all functionally equivalent method
pairs are detected as clones, but almost all functionally in-
equivalent method pairs are also detected as clones. Increas-
ing the threshold value improves recall\(^I\)(InferCode), but
worsens recall\(^E\)(InferCode) at the same time. The value
of accuracy(InferCode) was the highest when the thresh-
old value was 0.949, which was 64.04%. At this time, recall\(^E\)(InferCode) was 83.83% and recall\(^I\)(InferCode)
was 32.86%.

The evaluation results of ASTNN are shown in Fig. 5 (b). Overall, the shape of the graph of ASTNN is similar to that of
InferCode. For thresholds below 0.531, recall\(^E\)(ASTNN)
is greater than 99%, while recall\(^I\)(ASTNN) is only 0.59% at
the same time. At the threshold values of 0.677=0.681,
accurary(ASTNN) has the highest value of 61.30%, at
which recall\(^E\)(ASTNN) and recall\(^I\)(ASTNN) have values
of 94.19% and 9.62%, respectively.

We found that the clone detection tool NIL, which is
based on lexical sequences, cannot detect many functionally
equivalent method pairs as clones and it does not have a
high ability to detect functionally equivalent methods. We
also found that InferCode and ASTNN, clone detection tools

\[
\begin{array}{l|ll}
\text{Table 2} & \text{Detection results of NIL} \\
\hline
\text{True value} & \text{EMP} & \text{IMP} \\
\hline
463 & 234 & 879 \\
234 & 618 & 879 \\
\hline
\end{array}
\]
based on abstract syntax trees and deep learning, can detect functionally equivalent method pairs as clones, but they also detect functionally inequivalent methods as clones. From the above results, we conclude that new methods need to be developed to properly find functionally equivalent method pairs.

7. Related Work

This research is inspired by the literature [20]. In the literature, the idea of obtaining a set of functionally equivalent methods by executing the generated test cases against each other is presented\(^1\), and a dataset of functionally equivalent method sets was constructed from the Borge’s dataset [21]. The differences between this paper and the literature are as follows.

- The Borge’s dataset [21] includes approximately 36 million lines of code, whereas the IJADataset used in this paper has approximately 314 million lines of code. There is also a difference in the size of the constructed datasets: the dataset constructed in the literature [20] includes 276 functionally equivalent method sets, while the dataset in this paper contains 1,342 functionally equivalent method pairs.
- While all methods that were able to generate tests were subject to test execution in the literature [20], in this paper, test execution was not performed when the number of test cases generated was less than five. The reason for this is that in the process of constructing the dataset in the literature [20], when the number of tests generated was small, in most cases the methods were judged to be not functionally equivalent by the human eye even if mutual execution of test cases was successful.
- In the dataset construction process, visual checks were performed by one author in the literature [20], whereas in the dataset construction process in this paper, the final decision on whether the functions are equivalent was made after independent evaluation and discussion by three persons.
- This paper evaluates three code clone detection tools as an example of the use of the constructed dataset, but no such use case is given in the literature [20].

Svajlenko et al. constructed the BigCloneBench dataset [22]. There are only 43 different functionalities in the current BigCloneBench version. At the end of the dataset construction procedure, manual validation was performed. However, the target of the human validation work is the Java methods discovered by keywords and code patterns. Therefore, functionally equivalent methods that are not found by keywords and code patterns are not included in the BigCloneBench dataset. In addition, the target methods were not executed during the dataset construction procedure, and no functional equivalence checks were performed from a dynamic perspective.

Liu et al. constructed a dataset of functionally equivalent programs using data from past programming competitions. They collected functionally equivalent programs for about 5,000 questions [23]. In this dataset, programs developed by multiple users to a question are treated as functionally equivalent programs. Zhao et al. publish a dataset of programs with the same functionality in Google Code Jam [19], and Mou et al. also publish a dataset of programs submitted to programming education support systems [17]. On the other hand, unlike their datasets, our dataset is not a program for competitive programming, but source code with functionally equivalent features included in the OSS.

Rabin et al. have developed a tool, ProgramTransformer, which changes the structure of a given program [24]. The tool possesses a number of rules for changing the structure of a program, and changes the structure based on those rules. For example, it automatically rewrites repetitions described by for-statements into while-statements, and changes variable names.

8. Threats To Validity

Herein, we describe threads to validity in this research.

8.1 Code Normalization

The normalization in STEP-1 prevents the proposed procedure in Sect. 4 from outputting Type-1 and Type-2 clones as functionally equivalent method pairs. This normalization also considerably reduces the number of methods targeted after STEP-2, which considerably improves the overall processing speed of the proposed procedure.

However, there is a negative aspect to this normalization. Figure 6 shows two artificial methods that end up in exactly the same source code due to the normalization. These methods are not simple Type-2 clones because they perform semantically different operations. Therefore, these methods can be included in the dataset constructed in this study. However, since the source codes of these two methods are exactly the same after normalization, one of the two methods will be removed in STEP-1 if both of them exist in the same source code for them.

---

\(^{1}\)In the literature [20], Higo et al. obtained a set of functionally equivalent methods, whereas in this paper we determine functional equivalence by pairs of methods rather than a set.
the target source code. The proposed procedure in Sect. 4 is a way to construct a large dataset as efficiently as possible, and does not detect all functionally equivalent methods that exist in the target source code. Therefore, it is not a major problem to miss such corner-case methods.

8.2 Human Judgement

In STEP-4, human judgements whether the target method pairs are truly functionally equivalent or not. Since the judgements were made by human, it was not possible to completely eliminate subjectivity, and the possibility of judgment errors cannot be denied. In order to minimize these possibilities as much as possible, this research decided to obtain consensus among three persons instead of two.

9. Conclusion

In this study, we identified functionally equivalent method pairs by automatically generating test cases for Java methods extracted from open source software and executing the generated test cases against each other. Functionally equivalent method pairs were extracted from the 314 million lines of Java source code included in the IJADataset dataset. As a result, we obtained 13,710 candidate functionally equivalent method pairs. Of these, 2,194 method pairs were visually verified, and 1,342 functionally equivalent method pairs were obtained. This study also used this dataset to evaluate the accuracy of three code clone detection tools. This dataset can also be used to examine code quality, such as code maintainability and understandability.

The current challenge is that it takes a very long time to find candidate functionally equivalent method pairs, since all combinations of methods with equal return value and parameter types that could generate tests are tested against each other. In this experiment, it took approximately 40 days for STEP-3 alone. In the future, we plan to introduce heuristics to reduce the number of method combinations that need to be executed mutually in order to find candidate functionally equivalent method pairs more quickly.

The large percentage (852/2,194=38.8%) of method pairs that were determined to be functionally inequivalent by visual inspection also needs to be improved. If there are few method pairs that are determined to be functionally inequivalent by visual inspection (if the mutual execution of test cases sifts out most of the methods that are not functionally equivalent), then the need for visual inspection would be eliminated and a large data set could be created almost automatically. We are aiming for higher quality automatic test generation by improving the test generation algorithm in EvoSuite.

Acknowledgments

This research was supported by JSPS KAKENHI Japan (JP20H04166, JP21K18302, JP21K11829, JP21H04877, JP22H03567, JP22K11985).

References


MuSRGM: A Genetic Algorithm-Based Dynamic Combinatorial Deep Learning Model for Software Reliability Engineering

Ning 
Duksan 
Suntae

SUMMARY In the software testing phase, software reliability growth models (SRGMs) are commonly used to evaluate the reliability of software systems. Traditional SRGMs are restricted by their assumption of a continuous growth pattern for the failure detection rate (FDR) throughout the testing phase. However, the assumption is compromised by Change-Point phenomena, where FDR fluctuations stem from variations in testing personnel or procedural modifications, leading to reduced prediction accuracy and compromised software reliability assessments. Therefore, the objective of this study is to improve software reliability prediction using a novel approach that combines genetic algorithm (GA) and deep learning-based SRGMs to account for the Change-point phenomenon. The proposed approach uses a GA to dynamically combine activation functions from various deep learning-based SRGMs into a new mutated SRGM called MuSRGM. The MuSRGM captures the advantages of both concave and S-shaped SRGMs and is better suited to capture the change-point phenomenon during testing and more accurately reflect actual testing situations. Additionally, failure data is treated as a time series and analyzed using a combination of Long Short-Term Memory (LSTM) and Attention mechanisms. To assess the performance of MuSRGM, we conducted experiments on three distinct failure datasets. The results indicate that MuSRGM outperformed the baseline method, exhibiting low prediction error (MSE) on all three datasets. Furthermore, MuSRGM demonstrated remarkable generalization ability on these datasets, remaining unaffected by uneven data distribution. Therefore, MuSRGM represents a highly promising advanced solution that can provide increased accuracy and applicability for software reliability assessment during the testing phase.

key words: genetic algorithm, software reliability growth models, LSTM, attention, deep-learning

1. Introduction

Reducing the development time and cost while maintaining software quality is a challenge faced by all software companies. To address this issue, SRGMs have gained popularity as a measure of software quality [1]. These models use statistics to establish a correlation between failure data and a known function [2], such as an exponential function [3]. If the correlation is confirmed, the function can be used to predict the number of future failures that will occur in the software system.

Many SRGMs have been developed and applied during software testing to assess software reliability [4]. The two primary categories of SRGMs are the concave model and the S-shaped model [5], [6], as shown in Fig. 1. The concave model is based on the assumption that failures are more readily identified and resolved during the initial testing phases. Consequently, FDR experienced a rapid decline at the beginning, gradually tapering off over time and forming a concave trend [7]. In contrast, the S-shaped model suggests that FDR undergoes a gradual rise in the initial stages of testing [8], [9]. This initial phase often reflects the learning process of the testing team and the complexity of uncovering less obvious faults. As testing progresses and the team’s understanding deepens, the FDR is anticipated to continue its upward trend [10]. When using SRGMs, one of these categories must be selected and associated with the failure data [11]. Traditional SRGMs assume that the FDR follows a continuous growth pattern, such as constant, exponential, or power-law, throughout the testing phase [12]. However, this assumption is invalid if there are unexpected changes in testers or testing strategies, known as change-points. The change in FDR due to change-points affects the accuracy of software reliability assessments using traditional SRGMs.

Researchers have explored different strategies to enhance the predictive performance of SRGMs. One approach involves considering change-points to improve fault prediction accuracy and achieve satisfactory forecasting precision. Jing Zhao et al. [13] and Vikas Dhaka with Nidhi Nijhawan [14] have incorporated change-points and environmental factors, significantly enhancing accuracy. Nevertheless, these methods still rely on non-homogeneous Poisson process (NHPP) functions, leading to a noticeable gap in predictive accuracy compared to artificial intelligence models. Another avenue of research focuses on leveraging deep learning techniques to enhance software reliability prediction [15]. Wu and Huan [16] proposed a method to transform traditional SRGMs into deep-learning models, yielding promising results. However, their approach only wraps the traditional SRGM with deep learning techniques and does...

Copyright © 2024 The Institute of Electronics, Information and Communication Engineers
not effectively address the change-point problem.

To address this issue, we introduced the MuSRGM, a novel model that integrates Genetic Algorithms (GA) and deep learning-based SRGMs. Our approach involves decomposing deep learning-based SRGMs into varied hidden layers, each defined by distinct activation functions. GA then dynamically combines these layers via a fitness function to generate the optimized MuSRGM. MuSRGM’s innovation extends to its approach to failure data, treating it as a time series for comprehensive analysis. Embedded within MuSRGM, Long Short-Term Memory (LSTM) networks and an attention mechanism collaboratively work to unravel the nuances of failure data. LSTM is employed to discern the overall trends in the distribution of failure data, while the attention mechanism is adept at identifying anomalies within this time series that may signal change-points. Such an integration bolsters MuSRGM’s ability to navigate the complexities inherent in software testing, particularly in contexts where change-points are present.

MuSRGM demonstrated superior prediction accuracy on three diverse datasets, as evidenced by a significant enhancement in mean squared error (MSE) performance, showing improvements of 71.51%, 41.39%, and 96.02% respectively, surpassing the baseline deep learning-based SRGMs. These findings highlight the effectiveness of MuSRGM in improving prediction performance. The consistent enhancement in MSE across diverse datasets underscores MuSRGM’s robust generalization capability, affirming its efficacy in software reliability prediction. Unlike other research methods that match data using traditional SRGMs, MuSRGM starts from the data and customizes the model based on the unique characteristics of the failure dataset using GA. As a result, MuSRGM surpasses the performance of existing methods documented in the literature.

The main contributions of this paper are summarized as follows:

- The proposed method utilizes a genetic algorithm to dynamically combine the activation functions of deep learning-based SRGMs. This allows for the generation of MuSRGM, tailored to the specific features of the failure data.
- To further improve the accuracy of prediction, the MuSRGM incorporates both Long Short-Term Memory (LSTM) and Attention mechanisms. These techniques enable the SRGMs to effectively capture the irregular variation patterns of the failure data, leading to more accurate identification of change-point phenomena in the data.
- We separately examined the performance of combining the Attention and LSTM mechanisms with neural networks for analyzing failure datasets. The experimental results demonstrate that combining the Attention and LSTM mechanisms can further improve the predictive accuracy of neural networks for failure datasets.

The organization of the remaining sections in this paper is as follows. Section 2 offers a background and literature review for this study. In Sect. 3, we present the implementation details of the proposed method. Section 4 outlines the experimental setup and analyzes the results obtained from the experiments. The conclusion of this paper is presented in Sect. 5.

2. Background and Related Work

In this section, we aim to provide a comprehensive overview of the background knowledge necessary to understand the proposed approach for MuSRGM. The background to MuSRGM included the concepts of SRGMs, the change-point phenomenon, and Deep Learning-based SRGMs.

2.1 SRGMs

Software reliability growth models assume that the occurrence of software failures is a stochastic process, so traditional SRGMs are characterized as NHPP functions with specific parameters. These function parameters provide valuable information, such as the expected total number of failures in a project, the rate at which failures are detected, and so on. These functions can be used in future tests for future failure rates or the number of defects remaining in the code [17], [18]. One of the most well-known SRGMs is the Goel-Okumoto (GO) Model, which was introduced by Goel and Okumoto in 1979 and follows the equation:

$$m(t) = a(1 - e^{-bt}), a > 0, b > 0$$

Where $a$ is the total expected number of failures within the software project, $b$ is the rate of failure discovery, and $m(t)$ is the expected cumulative number of failures discovered in the corresponding time.

2.2 Change-Point

Traditional SRGMs assume that FDR stays a continuous
growth pattern throughout the testing phase, correlating with the count of remaining system failures. However, this assumption may not hold true in actual testing scenarios. The FDR can vary due to factors like changes in testing teams, methodologies, or equipment, leading to what are known as ‘change-points’. These change-points, representing abrupt deviations in failure detection trends, demonstrate the non-linear nature of software testing and challenge the efficacy of traditional SRGMs. This highlights the need for more adaptive and versatile models to accurately assess software reliability.

2.3 Deep Learning-Based SRGMs

In the pursuit of refining parameter estimation for SRGMs, a significant stride was taken by Wu and Huan [16]. They introduced an innovative paradigm through Deep Learning-based SRGMs and extensively elucidated the process of transforming conventional SRGMs into deep learning models. We take the Goel-Okumoto (GO) Model as an example to illustrate the method. The GO model is given by Eq. (1).

\[ \lambda(t) = \lambda_0 e^{\beta t} \]

The equation above can be interpreted as a composite function with a variable \( t \), which can be decomposed into three distinct sub-functions. This is shown in the following Table 1.

A deep learning network consists of an input layer, a hidden layer, and an output layer [19]. As shown in Fig. 2. Therefore, the deep learning network can be considered as a composite function mapped from the input \( x \) to the output \( y \). The expression is shown as follows: \( y = activation3(activation2(activation1(x))) \) [20]. Where \( x \) is the input value of the neural network and \( y \) is the output value of the neural network [21]. \( activation1() \), \( activation2() \), and \( activation3() \) are activation functions of the hidden layers of the neural network. Assuming that the bias of each hidden is \( b \), the neural networks can be listed in the following Table 2.

It can be seen that the GO model has a similar structure to the Neural network, so the weights of the Neural network can be used to simulate the parameters \( a \) and \( b \) in the GO model [22]. The specific method is shown in Fig. 3. With the weights \( w_{ij}^2 \) corresponding to \( a \) and \( w_{ij}^1 \) corresponding to \( b \). In this way, the parameter values of the SRGMs can be determined by the Neural network.

2.4 Related Work

Various methods have been tried to predict software reliability, traditionally focusing on statistical and model-based approaches. Nevertheless, with the dynamic nature of software development, identifying and adapting to change-points has become crucial for accurate reliability predictions. Inoue [23] introduced a modeling framework that employs change-points to evaluate software reliability, noting significant shifts in hazard rates associated with these points. Zhao [13] developed an SRGM that integrates change-points with environmental factors from different testing phases, enhancing the model’s reflection of actual testing environments. Samal et al. [24] further expanded this concept, considering the effects of incomplete debugging and change-points on SRGMs, and demonstrated impressive accuracy in fitting failure data.

In parallel with these developments, the advancement of AI technologies, especially deep learning, has led to a new wave of software reliability prediction methods. Lo [25] improved the accuracy of reliability predictions by directly feeding failure data into a Feed-Forward Neuro Network (FFNN). Cai et al. [26] divided failure data into time segments \( X_1, X_2, X_3, \ldots, X_n + 1 \) and used \( X_1, X_2, X_3, \ldots, X_t \) as input to a neural network to predict \( X_{n+1} \). As RNNs matured, Fu et al. [27] and Gusmanov [28] both attempted to analyze failure data and predict software reliability using LSTM, achieving good results. Su [29], Wang [17], and Lakshmanan [19] have contributed to the combination of traditional SRGMs with neural networks. Wu and Huan [16] analyzed the characteristics of SRGMs and decomposed them into activation functions in neural networks, completing the transformation from traditional SRGMs to deep learning-based SRGMs.
After a thorough analysis of relevant studies, this paper proposes a method that integrates GA with deep learning-based SRGMs. Our approach fully exploits the properties of different SRGMs (concave and S-shaped types) and does not rigidly rely on fixed models to fit data, but instead generates models dynamically based on data characteristics. Consequently, it can fit the change-point phenomenon well.

3. Proposed Approach

In this section, we provide a detailed description of the MuSRGM. We begin with an overview of the research methodology, as illustrated in Fig. 4, followed by an explanation of each step, including data collection, data processing, Individual evaluation, and GA operation. The change-point phenomenon happens when there is a sudden change in the FDR, which violates the assumption of a continuous FDR throughout the testing phase. Therefore, a single S-shaped or Concave model cannot accurately fit the data distribution of the failure dataset. By combining the characteristics of S-shaped and Concave models, a better approximation of the failure distribution in real testing situations can be achieved. MuSRGM combines GA, deep learning-based SRGMs, LSTM, and Attention techniques to fully leverage the advantages of different technologies and enhance the accuracy of software reliability prediction. Firstly, the failure data is a time-series sequence that can be learned using LSTM to capture the inherent connections between the data. The role of the Attention mechanism is to identify anomalies in the time series, i.e., the change-point phenomenon. By combining GA with deep learning-based SRGMs, GA dynamically selects the best activation function combination based on the characteristics of the failure data, resulting in the optimal reliability assessment model.

Algorithm 1 shows the pseudo-code for MuSRGM. The input data consists of three parts: the failure dataset, the activation set of deep learning-based SRGMs, and the hyper-parameters set required for training deep learning models. The activation function set and the set of hyperparameters are combined as the GA initialization population. The fitness function is used to calculate the fitness value of each individual in the population, and the parameters of the neural network with the lowest fitness value are output as the result when the stopping condition of the GA operation is met.

3.1 Data Collection

Before using the GA algorithm, we need to gather the data. This data consists of two types: (1) the failure dataset used for training the deep learning model, and (2) the set of parameters for the GA algorithm. The failure dataset consists of two features: the time of failure detection and the cumulative number of failures, which are illustrated in Fig. 5. The time of failure detection can be represented by calendar time, system running time, or the number of tests.

Recording the calendar time of failures during testing can be impractical and expensive. Additionally, testing is often conducted asynchronously, and the probability of detecting similar types of failures can vary from test to test. Therefore, system running time is not a good metric.

![Fig. 4 Overview of proposed method](image-url)
stead, a fixed period of system operation is the most suitable metric for counting the number of failures. The cumulative number of failures refers to the count of unique failures identified in the code during a specific system run period.

The parameter set of our GA-based method comprises two types of data: activation functions from deep learning-based SRGMs, and hyperparameters for model training. Following Wu and Huan [16], we transform traditional SRGMs into deep learning models with three hidden layers, each incorporating activation functions as part of the parameter set. Given the classification of traditional SRGMs into Concave and S-shaped models according to the FDR, our three-layer deep learning-based SRGMs are designed to embody these patterns. We adopted various models, such as the Logarithmic Growth Curve (LGC), Goel-Okumoto (GO), Delayed S-shaped Curve (DSS), Musa–Okumoto Logarithmic Poisson (Musa), and Yamada Exponential (YEX), to represent the spectrum of SRGMs, each with significant applications and research in software reliability. Specifically, the LGC and GO models, characteristic of their concave nature, are optimal for the early defect discovery stage when the discovery rate declines as testing advances. Conversely, DSS, Musa, and YEX, as S-shaped models, effectively capture the incremental complexity at different testing phases. Table 3 details these models’ activation functions.

MuSRGM combines deep learning-based SRGMs with LSTM networks and attention mechanisms. Key hyperparameters, such as learning rate, training epochs, batch size, and time steps, are optimized using GA for maximum efficacy. The objective of MuSRGM is to blend the features of both Concave and S-shaped models, effectively fitting the change-point phenomenon by systematically combining model elements and hyperparameters.

3.2 Processing Data

Our goal is to predict the number of failures that will occur in future system operating cycles based on the cumulative number of failures provided by the failure dataset. To achieve this, we use the LSTM model, which treats the failure data as a time series. Each input is a vector of failure data, and the output is a single value representing the number of failures at the next time. Figure 6 shows the data shape of the input and output data. To train and test the deep learning model,
we divide the failure dataset into a training set (90%) and a testing set (10%), as illustrated in Fig. 7.

When initiating the GA algorithm, population initialization is necessary [30]. Figure 8 illustrates the exact execution process. Each chromosome (individual solution) consists of two parts: the set of activation functions of deep learning-based SRGMs and the set of hyperparameters for the model [31]. We specify the corresponding search space for each parameter. Random values are assigned to each parameter in the search space during the initial run of the GA algorithm.

### 3.3 Individual Evaluating

After each iteration of the genetic algorithm, the next generation of the population is generated. The offspring need to be evaluated using a fitness function [32], [33]. The fitness function assesses the proximity of an individual solution to the ideal solution [34], and we use Mean Squared Error (MSE) as the criterion for the fitness function. A lower MSE indicates a higher fitness of the solution. MuSRGM is also integrated into the fitness function, as shown in Fig. 9. It includes a dynamic combination SRGM consisting of an LSTM layer, an Attention layer, and three hidden layers composed of activation functions extracted from deep learning-based SRGMs.

The combination of the LSTM and Attention mechanism is designed to capture the change-point phenomenon that occurs during the testing process. In the GA algorithm, each iteration of the fitness function builds an SRGM model.
using the current solution (a single chromosome). The training dataset is passed to the SRGM model for training, and after the model is trained, the test dataset is used to calculate the MSE value (fitness value). This process is repeated until the GA termination condition is met. The model with the smallest fitness value (MSE value) is the best model generated by the GA algorithm.

3.4 GA Related Operating

When the termination condition of the GA algorithm is not met, it undergoes the selection, Crossover, or Mutation operation to generate the next generation and calls the fitness function again for evaluation. This process is repeated until the termination condition is met. The execution process of selection, Crossover, or Mutation is illustrated in Fig. 10. Once the termination condition is satisfied, the model with the lowest fitness value is considered the best model, and the associated parameters of this model are the final output.

4. Evaluation

To validate the proposed model, the following research questions were developed.

RQ1: How does the proposed method compare to other state-of-the-art methods in terms of fitting the change-point phenomenon?

RQ2: How does the proposed method perform on different failure data sets in terms of Generalizability?

RQ3: Does the addition of the attention mechanism effectively improve the prediction performance of the SRGMs?

RQ1 aims to verify the effectiveness of our proposed method in fitting the change-point phenomenon compared to other state-of-the-art methods. RQ2 investigates the generalizability of the proposed method for different failure datasets. RQ3 examines whether the addition of the attention mechanism can effectively enhance the predictive performance of the model.

4.1 Experimental Setup

4.1.1 Datasets

We validate the proposed model using three datasets: Eclipse’s Java Development Tools (JDT) project, the networking component of the Linux Kernel, and Eclipse’s platform project. Detailed descriptions of the datasets are presented in Table 4.

Figure 11 displays the cumulative failure curve graphs for three datasets. It can be observed that FDR does not follow a constant curve. At a specific point during testing (indicated by the dashed circle), the FDR suddenly changes. Figure 11 (a) shows a noticeable change in FDR during the middle of the test. Additionally, Fig. 11 (b) and (c) indicate that two distinct changes in FDR occur during the testing process. This phenomenon is known as a change-point, which frequently occurs during the practical testing phase.

During the software testing phase, change-points may arise from intentional or unintentional factors. However, the large project datasets we rely on do not provide specific classification information on change-points. Nevertheless, the flexible design of the MuSRGM model enables it to analyze and capture all types of change-points. This model leverages the combination of comprehensive analysis of historical failure data with the strengths of deep learning and genetic algorithms to identify significant changes during the testing process, regardless of whether these changes are planned or random.

4.1.2 Evaluation Metrics

We compare the performance of the proposed model with other models using several evaluation criteria.

1. Mean-Square Error (MSE): A measure of the difference between the predicted and actual values, calculated as the average squared difference between them. It is a non-negative value, and a lower MSE indicates a better fit between the predicted and actual values. The MSE is calculated as follows:

\[ \text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 \]

2. Mean Absolute Percentage Error (MAPE): A metric that uses statistics to evaluate prediction accuracy. It is calculated as follows:
### Symmetric Mean Absolute Percentage Error (SMAPE)

A metric that uses percentage error to evaluate the expected accuracy, which is calculated as follows:

\[
\text{SMAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{\hat{Y}_i - Y_i}{Y_i^\#} \right|
\]

### Mean Percentage Error (MPE)

A measure that uses statistics to evaluate the mean of the percentage error between the predicted and actual values:

\[
\text{MPE} = \frac{100\%}{n} \sum_{i=1}^{n} \frac{Y_i - \hat{Y}_i}{Y_i}
\]

In the formula above, \(\hat{Y}_i\) represents the number of failures predicted using the model at the time \(i\), \(Y_i\) represents the true number of failures at the time \(i\), and \(n\) represents the total amount of data.

### Mean Time Between Failures (MTBF)

A measure that calculates the average time elapsed between two consecutive failures in a system, which is calculated as follows:

\[
\text{MTBF} = \frac{T_{\text{total}}}{N}
\]

\(T_{\text{total}}\) represents the total running time during the observation period, and \(N\) is the total number of failures observed in the system.

#### Baseline Methods

In comparison with related work, we establish baseline methods using the research methods mentioned and use these baselines to evaluate the superiority of our model performance. We establish a total of four baseline methods.

1. **Neural network**

   Traditional SRGMs rely on formulas with parameters for interpretation, but due to the limitations of these parameters, none of them can achieve accurate predictions on all failure data. In contrast, neural networks use weights to adjust the gap between input data and labeled data without the same limitations on parameters, leading to higher accuracy.

2. **LSTM**

   LSTM is a type of RNN network that can predict future outcomes based on previous knowledge, and it can capture the intrinsic relationship between data, which results in higher prediction accuracy.

3. **Deep learning-based SRGMs**

   Deep learning-based SRGMs combine the characteristics of traditional SRGMs and neural networks. They simulate the parameters of SRGMs using the weights of neural networks, and adjust the influence of each SRGM on the final output through error backward propagation, making them more adaptable to different failure data sets.

4. **GA-based SRGMs**

   Before the application of AI techniques in SRGMs, GA was a popular approach for parameter optimization. In this study, GA is used in combination with five SRGMs, and the resulting prediction results are used as one of the baseline methods for comparison.

5. **SRGM with change-point**

   SRGM with change-point comprehensively analyzes the dynamics of software reliability by incorporating FDR before and after the change-points. This integration provides a holistic perspective on how software reliability evolves, taking into account variations in FDR during different phases of testing.

#### Experimental Result

**RQ1: How does the proposed method compare to other state-of-the-art methods in terms of fitting the change-point phenomenon?**

The purpose of this investigation is to evaluate the effectiveness of our proposed method. We compared the predictive performance of GA-based models, neural networks, LSTM networks, and MuSRGM on three real-world datasets. The prediction results are presented in Tables 5-7. Upon reviewing the results, it is evident that the traditional SRGMs based on GA exhibit poor performance for the three datasets, with an MSE that is one order of magnitude higher than all deep learning-based methods. However, there are also significant differences in the MSE of each deep learning-based method. The LSTM-based method demonstrates the poorest performance, as evidenced by the highest MSE score among all models for dataset 1 (3372.14, as shown in Table 5). The MSE differences among the various methods are not significant for dataset 2, and the prediction differences are relatively small. For dataset 3, the deep learning-based SRGMs exhibit the worst performance with the highest MSE score of 7078.87 (as shown in Table 7). After observing the three datasets, MuSRGM demonstrates superior performance compared to all baseline methods, validating the effectiveness of combining SRGMs dynamically based on the characteristics of failure data in improving prediction accuracy. In summary, these findings confirm that our proposed method outperforms all other methods in the literature. The MuSRGM model yields identical values for SMAPE and MPE across all three datasets, as shown in Tables 5, 6, and 7 (0.000589, 0.001192, and 0.000388, respectively). This is due to the small range of data values and the relatively even distribution of prediction errors compared to true values. This further confirms that the predictive performance of MuSRGM is better than other methods.

Furthermore, prediction accuracy is gauged by the proximity of predicted MTBF to actual test data MTBF. An analysis of the data presented in Table 8 reveals a significant edge of Deep learning-based SRGMs over those GA and those incorporating change-points. This discovery not only highlights the exceptional predictive capabilities of neural networks but also emphasizes their effectiveness and adapt-
ability in handling complex data patterns. Importantly, MuSRGM’s MTBF predictions closely match the actual values of 0.8234, 0.0706, and 0.2500 for DS1, DS2, and DS3, respectively, with predicted values of 1.2271, 0.0787, and 0.4104. This demonstrates MuSRGM’s accuracy in fitting the change-point phenomenon and highlights its predictive precision.

**RQ3: Does the addition of the attention mechanism effectively improve the prediction performance of the models?**

The purpose of this research question is to evaluate the effectiveness of adding the Attention mechanism to the software reliability prediction model. We apply the Neural network and LSTM without the Attention mechanism, as well as the Neural network and LSTM with the Attention mechanism, to three failure datasets and present the prediction results in Tables 9-11. The results for the Neural network and LSTM without the Attention mechanism show that the Neural network performs better than the LSTM. However, the addition of the Attention mechanism has a varying impact on the prediction performance of the models. Specifically, the combination of the Attention mechanism and Neural network yields worse results than using the Neural network alone. On the other hand, combining the Attention mechanism with the LSTM greatly improves its prediction performance. Based on the results in Table 9, the prediction performance of the LSTM with attention model is slightly lower than that of the neural network for DS1 (see 227.59). However, for DS2 and DS3, the prediction across all three datasets. In contrast, deep learning-based methods demonstrated significant differences in predicting models for different datasets. For instance, while the LSTM model demonstrated a low MSE score on dataset 2 (109.75, as shown in Table 6), it performed poorly on datasets 1 and 3. Conversely, the deep learning-based SRGMs exhibited strong performance on datasets 1 and 2 but fared poorly on dataset 3, as evidenced by its high MSE score of 7078.87 (as shown in Table 7). Therefore, no single model could perform well on all datasets. However, MuSRGM employs natural selection to search the parameter space and uses a fitness function to determine the best solution. Consequently, our proposed method performed well on all three datasets: DS1, DS2, and DS3. We conclude that our proposed method is more applicable than existing methods in the literature.
performance of the LSTM with attention model surpasses that of the neural network. These findings suggest that the combination of LSTM and attention can effectively capture the change-point phenomenon in time series data, leading to an improvement in the model’s prediction performance.

4.3 Threats to Validity

This section addresses the limitations of our proposed method. As it is widely acknowledged, obtaining a failure dataset is the most challenging aspect of software reliability research. We endeavored to test the model’s performance using several datasets, but as each dataset was sourced from different software development projects, the test results were either extremely positive or negative. To mitigate this, we evaluated the proposed model using the same dataset as the one proposed by Wu and Huan [16]. We hope that relevant institutions or companies can provide more accurate failure datasets, enabling us to make more precise predictions about software reliability.

Additionally, to validate the effectiveness of our model, we established 9 baselines for performance comparison. While implementing each baseline based on relevant research papers, there may be some discrepancies compared to the original methods proposed in those papers.

5. Conclusion

Controlling software development costs by shortening testing cycles while ensuring software quality is a major challenge facing today’s software industry. SRGMs have gained attention for predicting the number of failures that may occur in future system operations, helping estimate the optimal time for system release. However, traditional SRGMs, which assume a continuous FDR throughout testing phases, are challenged by the occurrence of change-points; these events contradict these assumptions and compromise the accuracy of predictions as well as the reliability assessment of the software.

In this paper, we propose a novel approach for software reliability prediction by combining Deep learning-based SRGMs with LSTM and Attention mechanisms to capture the change-point phenomenon in failure data sets. The Activation function is extracted from the Deep learning-based SRGMs and used to form a new set of mutated SRGMs. These mutated SRGMs are then evaluated using GA to find the optimal SRGM model with the lowest MSE value. The new SRGMs take advantage of both Concave and S-shaped types, resulting in higher prediction accuracy and greater applicability.

We validated the proposed method with three different failure datasets and demonstrated that our model’s prediction accuracy is better than that of the state-of-the-art research method. Furthermore, our proposed method achieved excellent prediction results for all three datasets, indicating its superior applicability compared to the state-of-the-art research method. This will help accurately evaluate software reliability, thereby shortening the testing cycle and saving software development costs.

Currently, the training data used in the proposed model only has a single feature, which is the cumulative number of failures. In order to further improve the predictive performance of the model, we will look for more failure data features to enrich the training dataset. For example, the type of failure, the severity level of the failure, and so on. Combining enriched data features and the proposed model will further improve the accuracy of the predictions.

Acknowledgments

This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (RS-2023-00259099) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation). Additionally, this research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2022R1I1A30692233).

References


Ning Fu earned his Bachelor’s degree in Software Engineering from Dalian Jiaotong University, China, in 2004. He furthered his education by completing a Master’s degree in Software Engineering at Nanjing University of Science and Technology, China, in 2016. Currently, he is dedicated to pursuing a Ph.D. at Chonbuk National University in South Korea. Between May 2009 and December 2020, Ning Fu held the position of Senior Lecturer at Henan Agricultural Vocational College, China.

Duksan Ryu earned a bachelor’s degree in computer science from Hanyang University in 1999 and a Master’s dual degree in software engineering from KAIST and Carnegie Mellon University in 2012. He received his Ph.D. degree from the School of Computing, KAIST in 2016. His research areas include software analytics based on AI, software defect prediction, mining software repositories, and software reliability engineering. He is currently an associate professor in the software engineering department at Jeonbuk National University.

Suntae Kim is a Full Professor in the Department of Software Engineering at Chonbuk National University. He earned his B.S. degree in computer science and engineering from Chung-Ang University in 2003, and both his M.S. and Ph.D. degrees in computer science and engineering from Sogang University in 2007 and 2010, respectively. His research focuses on software architecture, artificial intelligence, blockchain, and financial technology.
A Ranking Information Based Network for Facial Beauty Prediction

Haochen LYU*, Jianjun LI†,‡, Yin YE†††, Nonmembers, and Chin-Chen CHANG††††, Member

SUMMARY The purpose of Facial Beauty Prediction (FBP) is to automatically assess facial attractiveness based on human aesthetics. Most neural network-based prediction methods do not consider the ranking information in the task. For scoring tasks like facial beauty prediction, there is abundant ranking information both between images and within images. Reasonable utilization of this information during training can greatly improve the performance of the model. In this paper, we propose a novel end-to-end Convolutional Neural Network (CNN) model based on ranking information of images, incorporating a Rank Module and an Adaptive Weight Module. We also design pairwise ranking loss functions to fully leverage the ranking information of images. Considering training efficiency and model inference capability, we choose ResNet-50 as the backbone network. We conduct experiments on the SCUT-FBP5500 dataset and the results show that our model achieves a new state-of-the-art performance. Furthermore, ablation experiments show that our approach greatly contributes to improving the model performance. Finally, the Rank Module with the corresponding ranking loss is plug-and-play and can be extended to any CNN model and any task with ranking information. Code is available at https://github.com/nehcoah/Rank-Info-Net.

1. Introduction

Facial attractiveness plays a significant role in our daily lives and has attracted much attention from scholars who have conducted extensive research on the subject[1], [2]. In recent years, people have become increasingly concerned about their facial beauty. It is worth noting that having a more beautiful appearance can give individuals an advantage in various aspects of life, such as public speaking, presentations, job interviews, and career opportunities. With the advancement of computer technology, there has been a shift towards utilizing computers to assess people’s appearance, giving rise to Facial Beauty Prediction (FBP). FBP aims to automatically evaluate facial attractiveness based on human aesthetics standards. It can also be applied in conjunction with practical tasks such as face beautification[3], [4], automatic face make-up[5], and plastic surgery[6].

In recent years, there have been significant breakthroughs in FBP tasks. These methods can be broadly categorized into two types: handcrafted feature-based and deep learning-based approaches. In the earlier studies, researchers evaluated facial attractiveness based on heuristics rules, such as facial landmarks, facial texture representations, symmetry and golden ratio proportions[7]. However, these methods had limitations and lacked fine-grained extraction and application of facial features. With the flourishing of deep learning, it has demonstrated unique capabilities in various domains, including facial beauty prediction. Various types of Convolutional Neural Networks (CNN), such as VGG[8], ResNet[9], MobileNet[10]–[12], EfficientNet[13], etc., have been applied to FBP tasks. The powerful feature extraction ability of these networks allows for a more comprehensive measurement of facial beauty. Specifically, facial beauty prediction methods can roughly be divided into regression methods and classification methods. Regardless of whether it is classification or regression, most researchers are dedicated to optimizing the feature extraction methods of CNN networks or exploring better ways to utilize the extracted facial features to achieve higher accuracy. Additionally, most facial beauty prediction datasets, such as the SCUT-FBP5500 dataset[14], are obtained by collecting ratings from a large number of volunteers, which serve as the corresponding ground truth. Some researchers utilize the label distribution from all volunteers’ ratings for each image and employ Label Distribution Learning (LDL) to optimize the performance of the model[15], [16].

However, for tasks such as facial beauty prediction and most scoring tasks, both classification and regression methods fail to effectively or even utilize the potential ranking information of images. We have also noticed that some researchers have applied pairwise ranking methods to facial beauty prediction tasks[17], [18], but these methods require two backbone networks in the training process in order to introduce ranking loss, which increases the model parameters and extends the training time. Therefore, we propose a new end-to-end model based on ranking information, which only requires one backbone network in both training and testing stages. The overall framework is shown in Fig. 1. We add a Rank Module and an Adaptive Weight Module to the traditional convolutional neural networks and design a method to extract ranking information and the corresponding pairwise ranking loss functions to fully utilize the underlying ranking information in the images. The outputs of the classifier
and the Rank Module are integrated through the Adaptive Weight Module to obtain predicted scores. Considering the inference capability and training efficiency required for extracting ranking information, we finally choose ResNet-50 as the backbone network for our experiments in this paper. The Rank Module is responsible for learning the ranking information of the images, while the classifier learns the label distribution, and the regression loss is used to constrain the overall predicted scores. We test our method on the SCUT-FBP5500 dataset [14] and obtain exciting results. The experimental results show that our model achieves a new state-of-the-art performance. Ablation experiments also demonstrate the crucial effect of the proposed Rank Module and the pairwise ranking loss functions on improving accuracy of the model. Furthermore, our proposed Rank Module is plug-and-play and can be extended to any convolutional neural networks, while the methods of extracting ranking information and the pairwise ranking loss functions can be applied to most tasks with ranking information, including various scoring tasks and age estimation tasks, etc.

The main contributions of this paper can be summarized into three points:

- We propose a novel end-to-end network based on ranking information. This model extends the traditional Convolutional Neural Networks (CNN) by incorporating a Rank Module and an Adaptive Weight Module, along with corresponding pairwise ranking loss functions. The model demonstrates transferability, as the Rank Module and Adaptive Weight Module can be seamlessly integrated into any CNN-based network model. Moreover, the ranking information-based approach can be applied to any task that involves ranking information.
- Compared to other methods based on ranking information, our proposed model only requires one backbone network. Under the same backbone network conditions, our approach addresses the issue of excessive model parameters during the training phase, saving training time while also achieving better performance.
- We conducts various experiments on the SCUT-FBP5500 dataset and the final results achieves a new state-of-the-art performance. Additionally, the results from ablation experiments demonstrate that our method significantly improves the performance of the network.

This paper is structured in a manner that includes an overview of the related work in Sect. 2, followed by a detailed description of our proposed method in Sect. 3. The experimental results are presented in Sect. 4 and Sect. 5 provides a conclusion of our study. Finally, some development plans, funds and institutions are acknowledged at the end.

2. Related Work

2.1 Pairwise Method of Ranking

Learning to Rank is widely applied for document ranking, such as recommendation algorithms, and can be roughly divided into three methods: Pointwise, Pairwise, and Listwise. Many researchers have summarized these methods [20], [21]. In the Pairwise approach, numerous applications have emerged, including Ranking SVM [22], RankBoost [23], RankNet [24], LambdaRank [25], and so on.
With the development of deep learning, some researchers have applied these ranking methods, especially the pairwise methods, to various fields. The introduction of Siamese networks [26] has paved the way for the application of pairwise methods in computer vision. Its dual-branch network structure provides conditions for obtaining pair information. Therefore, most researchers choose similar network structures when migrating pairwise methods to other tasks to facilitate pair information extraction.Gattupalli et al. [27] carefully selected images from the AVA dataset and constructed a new dataset of image pairs with relative labels. They also proposed a neural network-based method to train the ranking information. Lin et al. [18] also proposed a general convolutional neural network architecture based on the Siamese network’s framework and applied the pairwise method to the facial beauty prediction task.

2.2 Facial Beauty Prediction

Before the popularity of deep learning, researchers primarily used traditional methods for facial beauty prediction, including studying facial symmetry, texture features of the face and the golden ratio proportion. However, these methods had significant limitations as they lacked fine-grained feature extraction of facial characteristics and lacked a relatively systematic feature extraction approach, leading to poor performance. With the development of deep learning, a series of evaluation methods based on Convolutional Neural Network (CNN) have emerged.

Gray et al. [28] initially proposed a feature extraction method similar to CNN, eliminating the need for manual annotation of facial features for prediction. After the introduction of the VGG network [8], Xu et al. [29] applied it to facial beauty prediction tasks. Inspired by psychology, Xu et al. [30] introduced a hierarchical model called PI-CNN, using a cascaded fine-tuning approach to optimize predictors. Liang et al. [31] combined deep convolutional networks based on scattering transform with facial texture and shape features, proposing the RegionScarNet model. To address the issue of fixed-parameter convolutional kernels failing to fully utilize facial attributes, Lin et al. [32] introduced AaCNN, which can adaptively adjust the kernel size of the network. Xu et al. [33] proposed CRNet, which can simultaneously perform classification and regression tasks. Xu et al. [34] introduced a hierarchical multi-task network capable of simultaneously identifying the gender, race, and facial attractiveness of face images. Similarly, Xu [35] developed the multi-task model, which can automatically recognize facial attractiveness scores and gender. Lin et al. [17] presented R2-ResNeXt, which shared the weights of two ResNeXt networks [36] and used ranking loss to optimize network performance during training. Subsequently, they proposed a general CNN architecture called R²CNN [18], which considers facial beauty prediction as a ranking-guided regression problem, using two CNNs to simultaneously perform ranking and regression tasks. Fan et al. [15] reshaped facial attractiveness as a label distribution learning problem and proposed an end-to-end framework, incorporating low-level geometric features for feature-level fusion. Later, Liu et al. [16] introduced a lightweight end-to-end FBP method, which achieved promising results by training with an improved label distribution learning approach based on [15]. Wei et al. [37] proposed a method that utilizes facial landmarks to compute facial features with a low computational cost. Saeed et al. [38] proposed FIAC-Net, which is a light deep convolutional neural network for facial images attractiveness assessment. Later, they [39] integrated three regression-loss functions to capitalize on the unique traits of each loss function in the facial beauty prediction. Bougourzi et al. [40] proposed an architecture with two backbones (2B-IncRes) and introduced a parabolic dynamic law to control the behavior of the robust loss parameters during training. Yang et al. [41] aimed to train a model for assessing facial beauty using transfer learning while also using the fine-grained image model to separate similar images by first learning features.

3. Method

In this section, we will provide a detailed description of our proposed method. To make full use of the ranking information of images, this paper extends the pairwise method to the Facial Beauty Prediction (FBP) task. In tasks where there is a sequential ranking relationship between samples, it is evident that the reasonable utilization of these ranking information during the training process will improve the performance of the model.

3.1 Network Architecture

To better integrate the pairwise method into FBP tasks, we optimize the traditional neural network architecture. As shown in Fig. 1, we take ResNet50 as backbone network and add a Rank Module and an Adaptive Weight Module. Specifically, we utilize the Adaptive Weight Module to adjust the contributions of the classifier and the Rank Module, and combine them with weighted summation to obtain the final result. The classifier consists of a single fully connected layer, while the Adaptive Weight Module and the Rank Module are composed of fully connected layers, a GELU activation layer, and a Dropout layer. We apply the pairwise method to the Rank Module. Additionally, we draw inspiration from [16] and apply the label distribution learning to the classifier. Finally, we constrain the integrated output of the Adaptive Weight Module using regression methods.

3.2 Label Distribution Learning

In most datasets for facial beauty prediction, multiple individuals rate the same images and the ground truth score is determined by taking the average of their ratings. In the SCUT-FBP500 dataset, all images were rated by 60 volunteers on a scale of 1 to 5. This dataset is also used in the experimental part of this paper and detailed information about the dataset will be provided in Sect. 4.1. To make
these ratings informative for facial beauty prediction, we follow [16] and apply label distribution learning. At the image level, we can calculate the mean $\mu$ and variance $\sigma$ of all volunteer ratings for each image. Using these statistics as reference, we model the label distribution corresponding to each image using a Gaussian distribution. Considering network inference capability and training efficiency, we choose a sampling interval of $\Delta l = 0.05$. This means we divide the range $[1, 5]$ into 80 equal intervals and perform 80 sampling iterations within the range. Suppose the current sampling interval is $I_j = [s_j, s_j + \Delta l]$; then, the corresponding probability $q_j$ for the current interval can be calculated using the probability distribution function $F(x|\mu, \sigma)$ of the Gaussian distribution.

$$q_j = F(s_j + \Delta l|\mu, \sigma) - F(s_j|\mu, \sigma) \tag{1}$$

We combine all the sampled values to obtain the label distribution, denoted as $q$, and normalize $q$ using $L1$ normalization. Additionally, we apply a softmax operation to the output results of the classifier and denote the resulting distribution as $\tilde{q}$. The label distribution loss $L_{dis}$ can be calculated using the Euclidean distance.

$$L_{dis} = \frac{1}{n} \sum_{i=1}^{n} \|\tilde{q}^{(i)} - q^{(i)}\|_2 \tag{2}$$

where $n$ represents the number of samples in a batch.

3.3 Ranking Loss within Images

We denote the output of the Rank Module as $r \in \{r_1, r_2, \cdots, r_c\}$, where $c$ represents the total number of categories. For each image, ideally, the predicted probability distribution of model should resemble a Gaussian distribution, with the highest probability value located at the ground truth label and decreasing towards both sides, forming an ordered sequence. Empirically, when the probability distribution obtained by the network exhibits this ideal pattern, more accurate results can be achieved. Therefore, we utilize the ranking information to optimize the predicted probability distribution into the ideal state.

We sample the feature distribution and use the ranking method to optimize the network prediction. Specifically, we start sampling from the ground truth position in $r$, moving towards both sides. The sampled data is denoted as $(r_i, m_i)$, where $i$ refers to the $i$-th class, and $m_i$ is the given sequential tag during the sampling process. For example, we denote the sampled data set as $\mathcal{R} \in \{(r_0, 0), (r_{-1}, 1), (r_{-1}, 1), (r_{-2}, 2), \cdots\}$, where $r$ corresponds to the ground truth label of the current image. In the sequential tags, smaller values indicate positions closer to the beginning in the sequence. During the sampling process, we specify that the sequential tags of the sampled data on both sides of the ground truth label are the same if they have the same interval, which means they hold the same positions in the overall sequence.

Additionally, due to the different feature distributions presented by different images, we set a threshold in sampling to flexibly adjust the sampling range. The sampling threshold is denoted as $t$, and the pseudocode for the sampling algorithm is shown in Algorithm 1, which aims to generate reasonable samples for subsequent loss calculation. As different images have different feature distributions after feature extraction, thanks to the existence of the sampling threshold, the sampling range of the above process will be located in a reasonable range of each image, as the shaded area shown in Fig. 2 (a), ensuring the utilization of as complete feature information as possible for different images. Note that we will perform sampling on all the images.

After obtaining the all the samples, we will select two samples with different sequential tags to form a sample pair for further processing each time. We donate these two samples as $a = (r_a, m_a)$ and $b = (r_b, m_b)$, and define the score $S_{a,b}$ of the sample pair as

$$S_{a,b} = \frac{\exp(r_a - r_b)}{1 + \exp(r_a - r_b)} \tag{3}$$

The label for the pair of samples, denoted as $y_{a,b}$, is defined as follows: if $m_a < m_b$, then $y_{a,b} = 1$; otherwise,
Finally, the loss function of the sample pair is referred to as
\[
L_{\text{inner}}(a, b, y_{a,b}) = -y_{a,b} \log(S_{a,b}) - (1 - y_{a,b}) \log(1 - S_{a,b})
\]  \quad (4)

We will consider all samples with different sequential tags when calculating the loss. Figure 2 (b) illustrates the working mechanism of \( L_{\text{inner}} \). Ideally, \( L_{\text{inner}} \) can arrange the sampled samples in the specified order.

3.4 Ranking Loss between Images

For different images in a batch, their ground truth labels are not completely identical. There is also a ranking order relationship between different images based on ground truth label. In order to calculate the ranking loss between images more conveniently, we simplify the feature information of images in the same batch to the expectation \( \mathbb{E} \in \{ \mathbb{E}_1, \mathbb{E}_2, \ldots, \mathbb{E}_n \} \), where \( n \) is the total number of images in the current batch. The expectation \( \mathbb{E}_k \) for each image in the current batch is defined as
\[
\mathbb{E}_k = \sum_{i=1}^{c} \hat{r}_i \ast i, \quad k = 1, 2, \ldots, n
\]  \quad (5)

where \( \hat{r} \) represents the result of applying softmax to the output of the Rank Module \( r \), and \( c \) represents the number of classes. Similar to the approach mentioned in Sect. 3.3, we can denote the expectations and labels of all the images in a batch as a set like \( \{ (\mathbb{E}_1, \ell_1), (\mathbb{E}_2, \ell_2), \ldots, (\mathbb{E}_n, \ell_n) \} \), where \( (\mathbb{E}_j, \ell_j) \) is the \( j \)-th image in the batch, with the expectation \( \mathbb{E}_j \) and the ground truth label \( \ell_j \). Here, we define that images with bigger ground truth scores indicates positions closer to the beginning in the sequence. Similarly, we select two samples with different labels for processing each time. We donate these two samples as \( u = (\mathbb{E}_{u}, \ell_u) \) and \( v = (\mathbb{E}_{v}, \ell_v) \), and the score between the two samples, named \( S_{u,v} \), is calculated as
\[
S_{u,v} = \frac{\exp(\mathbb{E}_u - \mathbb{E}_v)}{1 + \exp(\mathbb{E}_u - \mathbb{E}_v)}
\]  \quad (6)

The label for the pair of samples, denoted as \( y_{u,v} \), is defined as follows: if \( \ell_u > \ell_v \), then \( y_{u,v} = 1 \); otherwise, \( y_{u,v} = 0 \). Finally, the loss function between these two samples is referred to as
\[
L_{\text{outer}}(u, v, y_{u,v}) = -y_{u,v} \log(S_{u,v}) - (1 - y_{u,v}) \log(1 - S_{u,v})
\]  \quad (7)

Here we will select samples with different labels for processing. When there is a deviation in the relative positioning of expectations between images, ideally, \( L_{\text{outer}} \) can move the overall probability distribution to the correct position. As shown in Fig. 3, we divide the relative positional deviations of probability distribution into two categories. First, there is a relative positional error, as shown in Fig. 3 (a), where the blue curve and orange curve correspond to the probability distributions for images with labels 3.0 and 4.0, respectively. The upper graph of Fig. 3 (a) shows that their relative positions are incorrect; theoretically, the blue curve should be on the left side of the orange curve. In this case, \( L_{\text{outer}} \) will pull the blue curve to the left and the orange curve to the right, placing them in the correct positions. The second category is the relative positions are correct but the intervals are too close, as shown in Fig. 3 (b), where the red curve and green curve represent the probability distributions for images with labels 3.0 and 4.0, respectively. From the upper graph of Fig. 3 (b), we can see that although their relative positions are correct, they are too close to each other. In this case, \( L_{\text{outer}} \) will pull both curves apart, to some extent, making them move away from each other.

3.5 Adaptive Weight Module

Furthermore, we introduce an Adaptive Weight Module to integrate the outputs of the classifier and the Rank Module. We denote the output of the classifier as \( f \in \{ f_1, f_2, \ldots, f_c \} \), the output of the Adaptive Weight Module as \( w \in \{ w_1, w_2, \ldots, w_c \} \), and the output of the Rank Module as \( r \in \{ r_1, r_2, \ldots, r_c \} \). The Adaptive Weight Module combines the outputs of the classifier and the Rank Module to generate a new output, denoted as \( p \in \{ p_1, p_2, \ldots, p_c \} \), which serves as the final output of the network. Specifically, each \( p_i \) can be represented as follows:
\[
p_i = w_i \ast f_i + (1 - w_i) \ast r_i, \quad i = 1, 2, \ldots, c
\]  \quad (8)

where \( c \) represents the number of classes. We define the predicted facial beauty score \( x \) as
\[
x = \sum_{i=1}^{c} \hat{p}_i \ast i
\]  \quad (9)

where \( \hat{p} \) is the result of applying softmax to \( p \). For the final prediction of the network, we choose the Smooth L1 loss to minimize the discrepancy between the ground truth and the predicted scores by the network,
\[
L_{\text{reg}}(x, y) = \begin{cases} 0.5(x - y)^2, & \text{if } |x - y| < 1 \\ |x - y| - 0.5, & \text{otherwise} \end{cases}
\]  \quad (10)
Table 1 Comparison with state-of-the-art on SCUT-FBP5500 dataset with five-folds cross validation. Best results are marked in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>MAE ↓</th>
<th>RMSE ↓</th>
<th>PC ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANet [32]</td>
<td>ResNet18</td>
<td>0.223b</td>
<td>0.2934</td>
<td>0.5055</td>
</tr>
<tr>
<td>Co-attention learning [42]</td>
<td>MobileNetV2x2</td>
<td>0.2020</td>
<td>0.2660</td>
<td>0.9260</td>
</tr>
<tr>
<td>MT-ResNet [35]</td>
<td>ResNet-50</td>
<td>0.2459</td>
<td>0.3208</td>
<td>0.8905</td>
</tr>
<tr>
<td>R\ CNN [18]</td>
<td>ResNetXt-50</td>
<td>0.2120</td>
<td>0.2800</td>
<td>0.9142</td>
</tr>
<tr>
<td>Dual Label Distribution [16]</td>
<td>MobileNetV2</td>
<td>0.1964</td>
<td>0.2585</td>
<td>0.9276</td>
</tr>
<tr>
<td>FLAC-Net + Loss Ensembles [39]</td>
<td>FLAC-Net</td>
<td>0.2028</td>
<td>0.2614</td>
<td>0.9305</td>
</tr>
<tr>
<td>Dynamic ER-CNN [40]</td>
<td>ResNetXt-50 + Inception-v3</td>
<td>0.1998</td>
<td>0.2653</td>
<td>0.9262</td>
</tr>
<tr>
<td>Ours</td>
<td>ResNet-50</td>
<td>0.1913</td>
<td>0.2551</td>
<td>0.9288</td>
</tr>
</tbody>
</table>

where \( y \) is the ground truth label and \(| \cdot |\) represents absolute value. The final loss function can be expressed as follows:

\[
\mathcal{L} = \lambda_1 \mathcal{L}_{inner} + \lambda_2 \mathcal{L}_{outer} + \lambda_3 \mathcal{L}_{reg} + \lambda_4 \mathcal{L}_{dis}\]

(11)

where \( \lambda_1, \lambda_2, \lambda_3, \) and \( \lambda_4 \) are hyperparameters that balance the four losses.

4. Experiments

In this section, we design various experiments to validate our proposed method. We provide a detailed analysis of the experimental setup and compare the results with state-of-the-art works. We also conducted ablation experiments to demonstrate the effectiveness of the Rank Module and the pairwise ranking loss functions.

4.1 Dataset and Evaluation Metrics

The SCUT-FBP5500 dataset consists of 5500 facial images. A total of 60 volunteers were asked to rate each photo on a scale ranging from 1 to 5. The ground truth label for each image is obtained by averaging the ratings from these 60 volunteers. In addition to the ground truth label for each image, the dataset also provides detailed rating scores from the 60 volunteers for each image.

In terms of evaluation criteria, we use Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Pearson Correlation coefficient (PC) to measure the performance of the model. A more outstanding model will exhibit lower MAE and RMSE values while having a higher PC value. The specific formulas for calculation are shown in Eq. (12).

\[
MAE = \frac{1}{N} \sum_{i=1}^{N} |x(i) - y(i)|
\]

\[
RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x(i) - y(i))^2}
\]

\[
PC = \frac{\sum_{i=1}^{N} (y(i) - \bar{y})(x(i) - \bar{x})}{\sqrt{\sum_{i=1}^{N} (y(i) - \bar{y})^2} \sqrt{\sum_{i=1}^{N} (x(i) - \bar{x})^2}}
\]

(12)

where \( N \) represents the number of images in the test set, \( x \) is the predicted score by the network, \( y \) is the ground truth label, \( \bar{x} = \frac{1}{N} \sum_{i=1}^{N} x(i) \), and \( \bar{y} = \frac{1}{N} \sum_{i=1}^{N} y(i) \).

Table 2 Details of five-folds cross validation on SCUT-FBP5500 dataset.

<table>
<thead>
<tr>
<th>Split</th>
<th>MAE ↓</th>
<th>RMSE ↓</th>
<th>PC ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1919</td>
<td>0.2569</td>
<td>0.9266</td>
</tr>
<tr>
<td>2</td>
<td>0.1948</td>
<td>0.2626</td>
<td>0.9238</td>
</tr>
<tr>
<td>3</td>
<td>0.1925</td>
<td>0.2593</td>
<td>0.9278</td>
</tr>
<tr>
<td>4</td>
<td>0.1885</td>
<td>0.2475</td>
<td>0.9338</td>
</tr>
<tr>
<td>5</td>
<td>0.1887</td>
<td>0.2491</td>
<td>0.9321</td>
</tr>
<tr>
<td>Avg</td>
<td>0.1913</td>
<td>0.2551</td>
<td>0.9288</td>
</tr>
</tbody>
</table>

4.2 Implementation Details

In this paper, we do not perform extensive operations on the input images. For each input image, we first resize it to 256×256. During the training phase, the images are randomly cropped to 224×224 and subjected to random horizontal flipping with a probability of 0.5. During the testing phase, the images are center-cropped to 224×224. We use ResNet-50 as the backbone network, initialized with ImageNet-pretrained weights, and modify the output layer’s channel to 80. We use the SGD optimizer with a momentum of 0.9 and weight decay of 0.0005. The initial learning rate is set to 0.001 and decreased by a factor of 0.3 every 15 epochs. Each model is trained for 90 epochs with a batch size of 64. Regarding the threshold \( t \) mentioned in Sect. 3.3, it is set to 0.95 for the first 15 epochs and then increased to 0.98 for the remaining epochs. Additionally, we set the hyperparameters \( \lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 1 \) in Eq. (11). All experiments are conducted on an NVIDIA Titan GPU. We perform five-fold cross-validation and the average results are reported.

4.3 Comparison with the State of the Art

We compared our approach with recent state-of-the-art works, as shown in Table 1. Additionally, detailed information for each split of the five-fold cross-validation is provided in Table 2. The data in Table 1 demonstrates that our method performs significantly better than other approaches.

On the SCUT-FBP5500 dataset, our method achieves state-of-the-art performance on MAE and RMSE. Specifically, compared to previous methods that utilize pairwise ranking method, such as R\ CNN [18], our model shows significant improvements. Furthermore, compared to methods using the same backbone, our results demonstrate a substantial advancements. In comparison to the recently proposed method Dual Label Distribution [16] and Dynamic ER-CNN [40], our model still outperforms it with superior...
Table 3 Ablation study on different combinations of methods. Best results are marked in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>MAB ↓</th>
<th>RMSE ↓</th>
<th>PC ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>B + LD</td>
<td>0.1940</td>
<td>0.2602</td>
<td>0.9258</td>
</tr>
<tr>
<td>B + LD + RI</td>
<td>0.1932</td>
<td>0.2591</td>
<td>0.9262</td>
</tr>
<tr>
<td>B + LD + RO</td>
<td>0.1924</td>
<td>0.2576</td>
<td>0.9275</td>
</tr>
<tr>
<td>B + LD + RI + RO</td>
<td>0.1913</td>
<td>0.2551</td>
<td>0.9288</td>
</tr>
</tbody>
</table>

Table 4 Ablation study on different setting of hyperparameters. Best results are marked in bold.

<table>
<thead>
<tr>
<th>Hyperparameters</th>
<th>MAB ↓</th>
<th>RMSE ↓</th>
<th>PC ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_1 = 1, A_2 = 0</td>
<td>0.1932</td>
<td>0.2591</td>
<td>0.9262</td>
</tr>
<tr>
<td>A_1 = 0, A_2 = 1</td>
<td>0.1924</td>
<td>0.2576</td>
<td>0.9275</td>
</tr>
<tr>
<td>A_1 = 2, A_2 = 1</td>
<td>0.1921</td>
<td>0.2580</td>
<td>0.9272</td>
</tr>
<tr>
<td>A_1 = 1, A_2 = 2</td>
<td>0.1919</td>
<td>0.2569</td>
<td>0.9278</td>
</tr>
<tr>
<td>A_1 = 1, A_2 = 1</td>
<td>0.1913</td>
<td>0.2551</td>
<td>0.9288</td>
</tr>
</tbody>
</table>

performance. Our proposed method slightly lags behind FIAC-Net with loss ensembles [39] in terms of PC metrics. It is worth noting that FIAC-Net is a network specifically designed for assessing facial attractiveness. However, overall, the results achieved by our method are inspiring and reach the state-of-the-art level.

4.4 Ablation Study

4.4.1 Different Combinations of Methods

In order to clarify the improvement of each method on the network performance, we conduct experiments by combining different methods with each other, and the results are shown in Table 3. We use B to represent training the network using L_reg, LD to represent the label distribution, which is L_dis. RI to represent L_inner, and RO to represent L_outer. The hyperparameters A_i in each ablation experiment are set to 1.

From Table 3, we can observe that even without using ranking information during the training phase, we have already obtained a remarkable model. This is because, in order to comprehensively utilize ranking information, we change the number of fully connected channels of the network to 80, enabling the network to output more refined prediction results.

With the introduction of the Ranking Module and the pairwise ranking loss, the accuracy of the network has been further improved. Both L_inner and L_outer have a certain enhancing effect on the network, and the best result is obtained by combining the two methods. Therefore, we can conclude that the Ranking Module with pairwise ranking loss plays a significant role in facial beauty prediction tasks. It can effectively extract ranking information from image features and optimize them accordingly.

4.4.2 Different Hyperparameters

In order to clarify the contributions of L_inner and L_outer to the improvement of model performance respectively, we conduct multiple experiments by setting different values for the hyperparameters A_1 and A_2 in Eq. (11). Meanwhile, the hyperparameters λ_3 and λ_4 for L_reg and L_dis are set to 1 in this section of experiments. The results are shown in Table 4.

From the experimental results, it can be observed that the different value of hyperparameters also has a significant impact. A larger A_1 will make the network more focused on adjusting the predicted probability distribution of each image towards an ideal state, which is discussed in Sect. 3.3, while a larger A_2 will prioritize the ranking relationships between images. For the SCUT-FBP5500 dataset, the most outstanding results are achieved when A_1 = A_2 = 1. However, for different tasks, determining the values of A_1 and A_2 based on the characteristics of the specific task is crucial in achieving optimal performance.

4.5 Visualization

To better illustrate the effect of L_inner during training stage, we select some samples and visualized their probability distributions as shown in Fig. 4.

In the example shown above in Fig. 4, although the probability distribution predicted by the model with L_inner is not very smooth itself, the jagged regions are significantly reduced compared to the distribution without L_inner. In the example shown below in Fig. 4, the probability distribution predicted by the model without L_inner does not even approximate the ideal shape mentioned in Sect. 3.3. However, when L_inner is added, the distribution exhibits a rudimentary form of the ideal state, and the jagged regions of the distribution are also greatly reduced.

Based on the above, we can come to the conclusion that L_inner is able to utilize ranking information effectively during the training phase, optimizing the probability distribution of the images to a relatively ideal state and improving the accuracy of network predictions.

5. Conclusions

In this paper, we propose a novel end-to-end network architecture based on ranking information. We introduce a Rank Module and an Adaptive Weight Module to the Convolutional Neural Network (CNN) model, along with pairwise
ranking loss functions. Unlike most methods that utilize ranking information, our approach only requires a single backbone network during training stage instead of sharing parameters between two backbone networks. This significantly reduces the training time of the network and even achieves better performance. Our experimental results on the SCUT-FBP5500 dataset reach a new state-of-the-art performance, and ablation experiments demonstrate that our method greatly assists in improving the performance of the model. Moreover, the Rank Module and Adaptive Weight Module we designed can be easily transferred to almost all CNN models. Additionally, the corresponding ranking information-based methods can be applied to any dataset that involves ranking information, such as most rating tasks and age estimation tasks.

Acknowledgments

This work was supported in part by the Key Research and Development Plan of Zhejiang: No.2021C03131; National Science Fund of China no.61871170; CEC Cooperation Funds: Intelligent Recognition Collaboration Project and TinyML Recognition Algorithms and Applications; Opening Fund of key Laboratory of Data Link Technology: CLDL-20202207.

References

2019.


Haochen Lyu is a graduate student at Hangzhou Dianzi University. He received the B.Sc. degree in computer science from Hangzhou Dianzi University in 2022. His research interests include image processing and video understanding algorithm.

Jianjun Li received the B.Sc. degree in information engineering from XiDian University, Xi’An, China, and the M.Sc. and Ph.D degrees in electrical and computer from The University of Western Ontario and University of Windsor, Canada separately. He is currently working at Hangzhou Normal University as a chair professor and a joint professor at Hangzhou Dianzi University. His research interests include computer vision algorithms, micro-electronics, audio, video and image processing algorithms and implementation.

Yin Ye received the B.Sc. and M.Sc. degree in information engineering from XiDian University, Xi’an, China. She is currently working at Huada Electronic Design Corp., Ltd. as the Chief Engineer. Her work areas include information security technology, tinyML system design and IC design.

Chin-Chen Chang received the Ph.D. degree in computer engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1987. On numerous occasions, he was invited to serve as a visiting professor, the chair professor, an honorary professor, an honorary director, an honorary Chairperson, a distinguished alumni, a distinguished researcher, and a research fellow by universities and research institutes. He has been the Chair Professor with the Department of Information Engineering and Computer Science, Feng Chia University, Taichung, Taiwan, since February 2005. His current research interests include database design, computer cryptography, image compression, and data structures.
Federated Learning of Neural ODE Models with Different Iteration Counts

Yuto HOSHIINO[1], Hiroki KAWAKAMI[2], Nonmembers, and Hiroki MATSUTANI[3], Member

SUMMARY Federated learning is a distributed machine learning approach in which clients train models locally with their own data and upload them to a server so that their trained results are shared between them without uploading raw data to the server. There are some challenges in federated learning, such as communication size reduction and client heterogeneity. The former can mitigate the communication overheads, and the latter can allow the clients to choose proper models depending on their available computation resources. To address these challenges, in this paper, we utilize Neural ODE based models for federated learning. The proposed flexible federated learning approach can reduce the communication size while aggregating models with different iteration counts or depths. Our contribution is that we experimentally demonstrate that the proposed federated learning can aggregate models with different iteration counts or depths. It is compared with a different federated learning approach in terms of the accuracy. Furthermore, we show that our approach can reduce communication size by up to 89.4% compared with a baseline ResNet model using CIFAR-10 dataset.

key words: federated learning, neural networks, neural ODE

1. Introduction

In traditional cloud-based machine learning systems, sending personal data to cloud servers has become problematic from a privacy perspective. Federated learning [1] is a distributed machine learning approach that can keep privacy-sensitive raw data decentralized. In the federated learning, clients receive a model from the server. Then they train the model with their own data and upload trained parameters to the server. The server aggregates the trained parameters received from the clients and sends back the aggregated parameters to the clients. These steps are repeated until the training process is converged. This eliminates the need to upload privacy-sensitive raw data to the server.

However, there are some challenges in the federated learning, such as communication size reduction and client heterogeneity. Communication size affects communication delay and power consumption of clients. It is affected by the machine learning model size. Regarding the client heterogeneity, not all clients always have the same hardware, compute resources, or training data. A client may use a deeper model for high accuracy, while another client may use a shallower model to reduce the computation cost. In this paper, we exploit Neural ODE [2] as a federated learning model to address these challenges.

For image recognition tasks, one of methods to improve accuracy is increasing the number of convolutional layers to build a deeper neural network. ResNet [3] is one of well-known CNN models that stack many residual blocks that contain convolutional layers and shortcut connections. Neural ODE utilizes a similarity to ODE (Ordinary Differential Equation) to implement deep neural networks consisting of residual blocks. Since it can be approximated to a ResNet model by repeatedly using the same weight parameters, it can reduce the weight parameters. In addition, it can be approximated to ResNet models with different depths by changing the iteration counts without increasing the number of parameters. dODENet [4] is a lightweight model that combines the ideas of Neural ODE and depthwise separable convolution [5] to further reduce the parameter size and computation cost. These Neural ODE models are smaller than ResNet.

In this paper, we introduce a flexible federated learning that allows clients to use models with different iteration counts and reduces the communication size by using Neural ODE based models. Our contributions are listed below*

• We propose to use the Neural ODE models for federated learning so that a server can aggregate models with different iteration counts. This can enhance the client heterogeneity since clients can use models with different iteration counts. In addition, using the Neural ODE models can significantly reduce the communication size.
• We experimentally demonstrate that the proposed flexible federated learning can aggregate these models with different iteration counts. It is compared with a federated learning approach that uses knowledge distillation. We discuss the pros and cons of the proposed approach.

The rest of this paper is organized as follows. Section 2 introduces baseline technologies behind our proposal. Section 3 proposes the flexible federated learning approach and shows the feasibility of the proposed approach. Section 4 evaluates the proposed approach in terms of the accuracy and communication size. Section 5 discusses pros and cons of the proposed approach against counterparts. Additional evalu*

*An early stage of this work appeared in our workshop paper [6]. In this paper, we experimentally demonstrate that the proposed approach can aggregate models with different iteration counts. It is compared to a federated learning approach that uses knowledge distillation.
lations with different hyper-parameters are also conducted. Section 6 concludes this paper.

2. Related Work

2.1 Federated Learning

Federated Averaging (FedAvg) is a basic federated learning algorithm proposed in [1]. Algorithm 1 shows the server- and client-side flows, where \( K \) is the total number of clients, \( k \) is their index, and \( D_k \) is data at client \( k \). Also, \( B \) is the size of a local mini-batch, \( E \) is the number of epochs to be trained by each client, and \( \eta \) is a given learning rate. In this algorithm, the first step is to initialize global weight parameters of the model. Then, \( m \) clients are randomly selected from \( K \) clients, and the server sends the global parameters to the selected clients. The size of \( m \) is determined by the client participating rate \( r \). The weight parameters are updated at each epoch (\( E \) epochs in total) by each client based on the formula in line 13. After \( E \) updates, the clients send their trained local parameters to the server. The server aggregates the received local parameters by taking the average based on the formula in line 8, where \( n \) is the total number of data and \( n_k \) is the total number of data at client \( k \). The aggregated parameters are then sent back to the clients as global parameters. The above steps are repeated \( r \) rounds.

Many federated learning technologies have been studied since FedAvg was proposed in 2016. These technologies are surveyed in [7]. Data heterogeneity is one of important research challenges in these studies since it is a major cause of accuracy degradation. For instance, since a local model is optimized toward the local optima by the client, it may be distant from other clients. Thus, their averaged global model may be far from a part of clients. To deal with this problem, FedProx [8] uses an additional proximal term to limit the number of local updates, and SCAFFOLD [9] uses a variance reduction to correct local updates. These algorithms aim to improve the local training step of FedAvg. In contrast, Personalized Federated Averaging [10] and Adaptive Personalized Federated Learning [11] aim to make personalized models that can achieve good accuracy in local clients.

Another challenge is the clients’ model heterogeneity. Since not all clients always have the same compute resources, selecting a proper model for each client can help the client heterogeneity. FedFeNN [12] and FedDF [13] address the model heterogeneity. In FedHeNN, each client trains its own model but pulls the representations learned by different clients closer by adding a proximal term to the client’s loss function [12]. FedDF is a federated learning algorithm that utilizes a knowledge distillation at the model aggregation step of the federated learning server. In the knowledge distillation step at the server, instead of averaging local parameters received from clients, a batch of sample data is predicted by the local parameters and their output logits are averaged. The averaged logits are then used for updating the client models at the server. Although FedDF allows a federated learning of different model architectures, model training is needed at the aggregation step of the server in addition to local training at the clients. This means that server-side samples are needed for the server-side knowledge distillation. In [14], a model agnostic federated learning approach that aims to improve participant model performance through learning from other participants via public dataset is proposed. It also relies on public dataset.

2.2 ResNet and Neural ODE

ResNet [3] is a well-known neural network architecture that can increase the number of stacked layers or building blocks by introducing shortcut connections. Using a shortcut connection, an input feature map to a building block is temporally saved, and then it is added to the original output of the building block to generate the final output of the block. In this paper, one building block in ResNet is called ResBlock.

ODE is composed of an unknown function and its ordinary derivatives. To obtain an approximate numerical solution, an ODE solver such as the first-order Euler method and higher-order Runge-Kutta method can be used. Based on a similarity between the network structure with shortcut connections and the ODE solver, one building block can be interpreted as one step in the ODE solver as suggested in [2]. Assuming that the Euler method is used as an ODE solver, it can be interpreted that the first-order approximation is applied to solve an output of the building block. In this paper, one building block is called ODEBlock, and the whole network architecture consisting of ODEBlocks is called ODENet.

2.3 Depthwise Separable Convolution

CNN is composed of multiple layers, such as convolutional layers, pooling layers, and fully-connected layers. Although CNN achieves a good accuracy in image recognition tasks, each convolutional layer has many parameters. Let \( N \), \( M \), and \( N_K \) be the number of input channels, the number of output channels, and the kernel size of one side, respectively. The number of parameters in one convolutional layer is \( NMN_K^2 \).
Depthwise separable convolution [5] divides this convolutional layer into two convolutional steps: depthwise convolutional step and pointwise convolutional step. In the depthwise convolutional step, a convolutional operation involving only spatial direction (the size is $N^2_k$) is applied for each input feature map. Different weight parameters are used for each of $N$ input channels; thus its weight parameter size is $N N^2_k$. Then, an output feature map of the depthwise convolutional step is fed to the pointwise convolutional step as an input. A $1 \times 1$ convolutional operation is applied for each input feature map and for each output channel; thus its weight parameter size is $N M$. The weight parameter size of the depthwise separable convolution is $N N^2_k + N M$ in total, which is approximately $N^2_k$ times reduction, assuming that $N, M \gg N_k$.

As a low-cost CNN model, dsODENet [4] applies the depthwise separable convolution to convolutional layers of ODEBlocks in order to further reduce the parameter size. It was originally proposed to be implemented on resource-limited FPGA (Field-Programmable Gate Array) devices [4]. In this paper, we use dsODENet as a federated learning model architecture in addition to ODENet. Their detailed structures are illustrated in the next section.

3. Proposed Federated Learning

In the traditional federated learning such as FedAvg, the server and all clients have to use the same model. For example, models with different layer depths cannot be averaged when ResNet is used as a model architecture. However, in a real environment, client devices are not the same and are likely to have different compute resources, such as memory capacity and computation power. Since the traditional federated learning cannot aggregate models with different depths, a common model used by all the clients should be carefully selected. In addition, it is necessary to reduce the communication size involved in exchanging weight parameters between the server and clients. In this paper, we use ODENet and dsODENet to enable a flexible federated learning between models with different layer iteration counts and significantly reduce the communication size. These target models are illustrated in Sect. 3.1. We discuss the feasibility of the proposed federated learning in Sects. 3.2 and 3.3.

3.1 Target Models

In this section, we first illustrate the structures and sizes of ResNet and ODENet. Then, we illustrate dsODENet.

Figures 1 and 2 show basic structures of ResNet and the corresponding ODENet. They consist of seven blocks including conv1 and fc. In the ResNet model, conv1 performs convolutional operations as a pre-processing layer, and fc is a post-processing fully-connected layer. After the conv1, $C$ physically-stacked ResBlocks are executed in block1. block2_1 is a downsampling ResBlock to reduce the feature map size, and $C$ physically-stacked ResBlocks on the output of block2_1 are executed in block2_2. The same operation is performed for block3_1 and block3_2 too.

ODENet replaces ResBlocks in Fig. 1 with ODEBlocks as shown in Fig. 2. In ResNet, $C$ ResBlocks are physically-stacked in block1, block2_2, and block3_2, while ODENet replaces these $C$ ResBlocks with a single ODEBlock. Instead, ODEBlock is executed $C$ times in block1, block2_2, and block3_2. A downsampling ODEBlock is executed only once in block2_1 and block3_1, respectively.

Here, we analyze the numbers of parameters of ResNet and ODENet. Let $O(L)$ be the number of parameters in one ResBlock and ODEBlock. In ResNet, $C$ ResBlocks are physically-stacked in one block, so the number of parameters is $O(CL)$. In contrast, ODENet repeats ODEBlock $C$ times in one block, so the number of parameters is $O(L)$. The parameter size reduction by ODENet becomes large as $C$ is increased. As shown, the communication size can be reduced by using ODENet as a federated learning model instead of ResNet.

In addition, the number of parameters can be further reduced by using dsODENet [4] as mentioned in Sect. 2.3. Figure 3 (a) shows a structure of an ODEBlock in ODENet. Figure 3 (b) shows that of dsODEBlock in dsODENet, in which each convolutional layer of the ODEBlock is replaced
with two convolutional steps: the depthwise convolutional step and the pointwise convolutional step. Conv1 and Conv2 are convolutional layers, and ReLU (Rectified Linear Unit) is an activation function. This modification can reduce the model and communication sizes compared with ODENet.

### 3.2 Weight Compatibility with Different Depths

In the case of ResNet, models with different depths have different numbers of stacked ResBlocks (i.e., different $C$) in their block1, block2_2, and block3_2. If we use FedAvg for federated learning, basically these different ResNet models cannot be averaged at the server due to the model incompatibility. In the case of ODENet, one ODEBlock is repeated $C$ times in block1, block2_2, and block3_2 of ODENet. In other words, ODENet models with different depths differ only in the numbers of iterations of ODEBlocks, not in the number of ODEBlocks. Therefore, the structure of the ODENet models with different $C$ is the same, so their parameters can be averaged at the server. Using ODENet as a federated learning model enables a flexible federated learning between models with different iteration counts and fully utilizes the performance of each client device. dsODENet [4] also enables such a flexible federated learning between models with different iteration counts for the same reason as ODENet.

Here, we examine if the above observation can work. This section focuses on the weight parameter compatibility of models which have different $C$ to demonstrate the feasibility of the proposed federated learning. Specifically, inference accuracies of ODENet and dsODENet models which were trained for the same or different $C$ are evaluated. Please note that, in the following, we use the total number of executed convolutional layers (denoted as $N$) to represent the depths of the ResNet, ODENet, and dsODENet models. We assume $N = 6C + 6$ in this experiment and use $N = 34$, 50, and 101. For example, the inference accuracy of ODENet-34 is evaluated using weight parameters trained as ODENet-50. We selected two models from $N = 34$, 50, and 101: one for training and another for inference. The experiment is performed 100 times for each combination. CIFAR-10 dataset [15] is used for the training and inference. Here, we consider all the combinations of two models from ODENet-34, ODENet-50, and ODENet-101 (e.g., ODENet-34 and ODENet-50). One of the two models (e.g., ODENet-34) is trained with the whole CIFAR-10 dataset, and then the trained weight parameters are used and tested as another model (e.g., ODENet-50). The same experiment is also performed for dsODENet.

Tables 1 and 2 show the inference accuracy of every combination. Figure 4 shows box-plots of accuracies of ODENet-34, 50, and 101 using weight parameters trained as ODENet-50. Figure 5 shows those of dsODENet. The results from Fig. 4 and Table 1 show that the accuracies are almost the same regardless of the tested models if the trained model is the same. The results from Fig. 5 and Table 2 also show the same tendency in dsODENet.

The same experiment is also performed for dsODENet. Table 1 and 2 show the inference accuracy of every combination. Figure 4 shows box-plots of accuracies of ODENet-34, 50, and 101 using weight parameters trained as ODENet-50. Figure 5 shows those of dsODENet. The results from Fig. 4 and Table 1 show that the accuracies are almost the same regardless of the tested models if the trained model is the same. The results from Fig. 5 and Table 2 also show the same tendency in dsODENet. Figure 6 shows box-plots of accuracies of ODENet-50 using weight parameters trained as ODENet-34, 50, and 101, respectively. Figure 7 shows those of dsODENet-50. The results from Fig. 6 and Table 1 show that the accuracies depend on $N$ of the trained model. The results from Fig. 7 and Table 2 show the same tendency in dsODENet.

1In Fig. 1, ResBlocks are executed $3C + 2$ times, each contains two convolutional layers. The pre- and post-processing (conv1 and fc) layers are also included in $N$; thus $N = 2(3C + 2) + 2$. 

---

**Table 1** Accuracy of ODENet models trained and tested with same or different depths

<table>
<thead>
<tr>
<th>Trained as</th>
<th>Tested as</th>
<th>Accuracy [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODENet-34</td>
<td>ODENet-34</td>
<td>75.46 ± 0.41</td>
</tr>
<tr>
<td>ODENet-50</td>
<td>ODENet-50</td>
<td>75.36 ± 0.43</td>
</tr>
<tr>
<td>ODENet-101</td>
<td>ODENet-101</td>
<td>73.26 ± 0.46</td>
</tr>
</tbody>
</table>

**Table 2** Accuracy of dsODENet models trained and tested with same or different depths

<table>
<thead>
<tr>
<th>Trained as</th>
<th>Tested as</th>
<th>Accuracy [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dsODENet-34</td>
<td>dsODENet-34</td>
<td>75.67 ± 0.53</td>
</tr>
<tr>
<td>dsODENet-50</td>
<td>dsODENet-50</td>
<td>75.64 ± 0.63</td>
</tr>
<tr>
<td>dsODENet-101</td>
<td>dsODENet-101</td>
<td>73.45 ± 0.68</td>
</tr>
<tr>
<td>dsODENet-34</td>
<td>dsODENet-34</td>
<td>72.24 ± 0.61</td>
</tr>
<tr>
<td>dsODENet-50</td>
<td>dsODENet-50</td>
<td>72.10 ± 0.62</td>
</tr>
<tr>
<td>dsODENet-101</td>
<td>dsODENet-101</td>
<td>72.00 ± 0.69</td>
</tr>
<tr>
<td>dsODENet-34</td>
<td>dsODENet-34</td>
<td>72.28 ± 0.64</td>
</tr>
<tr>
<td>dsODENet-50</td>
<td>dsODENet-50</td>
<td>72.11 ± 0.62</td>
</tr>
<tr>
<td>dsODENet-101</td>
<td>dsODENet-101</td>
<td>72.13 ± 0.64</td>
</tr>
</tbody>
</table>
As mentioned above, the inference accuracies of the ODENet and dsODENet models are reasonable even if the trained $N$ and tested $N$ are different. This means that ODENet and dsODENet models have a weight parameter compatibility with different depths. Furthermore, Table 1 shows that the accuracies of models that were trained as ODENet-101 are higher than those that were trained as ODENet-34 and ODENet-50. Table 2 also shows a similar tendency in dsODENet. These results indicate that using a deeper model for training can help to enhance the accuracy in ODENet and dsODENet models.

### 3.3 Federated Learning with Different Depths

In Sect. 3.2, we showed the weight parameter compatibility in ODENet and dsODENet models with different depths. In this section, we examine the feasibility of federated learning between ODENet models with different depths. Specifically, we perform federated learning of two ODENet models from among $N = 34$, 50, and 101 to see if it works correctly. In this experiment, FedAvg is used as a federated learning algorithm. The number of clients $K$ is only 2, the client participating rate $r$ is 1, the number of local epochs $E$ is 5, and the number of communication rounds is 20. In this experiment, the whole CIFAR-10 dataset is randomly partitioned into two clients evenly (i.e., not biased). ODENet-50 is used as a global model regardless of the combinations of two models. The same experiment is performed for dsODENet too.

Figures 8 and 9 show training curves (epoch number vs. loss value) of the ODENet and dsODENet models, respectively. A loss value in Fig. 8 shows a test loss computed by a global model (i.e., ODENet-50) which has been aggregated from two clients. In Fig. 8, the black line shows the loss values when federated learning is performed between ODENet-50 and ODENet-50. The red line shows those between ODENet-34 and ODENet-50. The green line shows those between ODENet-50 and ODENet-101. The blue line shows those between ODENet-34 and ODENet-101. Figure 9 shows experimental results for dsODENets. Meanings of line colors are the same as those in Fig. 8 but models used are dsODENets. The horizontal axis of these figures represents the number of epochs, and the vertical axis represents the loss value. As shown, the loss values decrease as the number of epochs is increased, and then they are converged around 50 epochs in all the combinations. This indicates that
these model combinations of different depths can be trained successfully. This and previous sections demonstrated that ODENet and dsODENet are capable of federated learning between models with different depths.

4. Evaluations

In this section, we evaluate the proposed flexible federated learning approach that uses FedAvg as a federated learning algorithm and ODENet, dsODENet, and ResNet as client models. They are compared with FedDF which is a model agnostic federated learning approach using a knowledge distillation. Finally, the proposed approach is evaluated in terms of the model size and communication cost.

Python 3.8.5, PyTorch 1.8.1 [16], and torchvision 0.9.1 are used for the model implementation. A machine with Ubuntu 18.04.5 LTS (64-bit), Intel Core i7-10700K CPU @ 3.8GHz, 32GB DDR4 SDRAM, and NVIDIA GeForce RTX 3090 GPU is used for the evaluation in this paper.

4.1 Accuracy

Although Sect. 3.3 showed the feasibility of the proposed federated learning, only two clients were used in the experiments. However, it is expected that more clients participate in a practical federated learning scenario. In this section, we increase the number of clients and conduct additional experiments.

In addition, it is expected that there is a bias in the data distribution for each client in the case of real environments. This means that the data distribution for each client is non-iid. In this experiment, Dirichlet distribution is thus used to make non-iid data environments. Dirichlet distribution is a kind of continuous multivariate probability distributions, and its data distribution is controlled by a vector $\alpha$. We use $\alpha = 1, 10, \text{and} 100$. CIFAR-10 is used as a dataset. Figures 10–12 show examples of data distributions with $\alpha = 1, 10, \text{and} 100$ as Dirichlet distribution parameters, respectively.
In this experiment, the number of clients $K$ is 30, the number of local epochs $E$ is 40, and the number of communication rounds is 100. Among the 30 clients, the numbers of clients that use ODENet-34, ODENet-50, and ODENet-101 are 10, 10, and 10, respectively. ODENet-50 is used as a global model. In Algorithm 1, $r$ is the client participating rate in the aggregation at each communication round. In this experiment, $r$ is set to 0.2, which means that six models are randomly selected from the 30 clients. FedAvg is used in the proposed federated learning approach, and the results are compared with those of FedDF.

Tables 3–5 show the accuracies of ODENet-50 and dsODENet-50 with FedAvg and FedDF when $\alpha = 1$, 10, and 100, respectively. Top5 accuracies are mostly high regardless of $\alpha$ in both the models. Top1 accuracies are decreased when $\alpha$ is small (e.g., $\alpha = 1$). In this case, the data distribution is highly biased, and the biased data distribution negatively affects the accuracy. Although addressing the data heterogeneity is a crucial challenge in federated learning, many studies have been conducted to overcome this issue as mentioned in [7] and thus addressing this is beyond the scope of this paper.

When we compare the proposed approach using FedAvg with FedDF, the accuracies of the proposed approach are lower than those of FedDF. This result is reasonable since FedDF introduces additional training overheads for the knowledge distillation (e.g., prediction and training of client models) at the server in addition to local training at the clients while the proposed approach does not impose such overheads as well as the conventional FedAvg based approach. To bring the evaluation condition of FedDF closer to that of the proposed approach, here we limit the number of samples to be used in the knowledge distillation at the server.

Table 6 shows the evaluation results of FedDF when varying the number of samples used for knowledge distillation ($\alpha = 10$).

Table 7 compares ResNet, ODENet, and dsODENet in terms of the model size, the forward/backward pass size, and the total memory size for training. “Model size” means the necessary memory size to retain the model. “Memory size during training” includes the model size, the forward/backward pass size, and the input/output data sizes. These values are measured by using torchinfo which is a tool that reports the parameter size information.

The communication is required between the server and clients in each communication round. In Table 7, “# of parameters transferred” is the sum of the weight parameters of convolutional layers and fully-connected layers in these models. In the following discussion, it is simply denoted as communication size. Table 7 shows that both the ODENet and dsODENet models reduce the communication sizes compared with the corresponding ResNet model. Compared with ResNet, the communication size of ODENet-50 is 10.6% of the original ResNet model. In the case of ResNet and dsODENet, the communication size of dsODENet-50 is 5.3% of the ResNet model. These results show that the number of server-side samples available for the knowledge distillation is limited. $\alpha = \infty$ means that there is no limitation in the number of server-side samples; that is, the whole CIFAR-10 training dataset can be used for the knowledge distillation in FedDF. However, please note that using the whole CIFAR-10 training dataset for the knowledge distillation significantly increases the training time of FedDF. We thus reasonably stop the knowledge distillation if test accuracy is not improved during the latest 10 training batches assuming that the batch size is 100. $\alpha$ is set to 10 as the data distribution parameter. The other conditions are same as those in Table 4. As shown in Table 6, the accuracy of FedDF increases as the number of server-side samples used in the knowledge distillation is increased. When the number of the available server-side samples is small, the accuracy of FedDF becomes close to (or lower than) the proposed approach as shown in Tables 4 and 6. Please note that “Time” in Table 6 represents the average time interval between two successive communication rounds. Such server-side overheads of FedDF are discussed in Sects. 5.1 and 5.2.

### 4.2 Memory and Communication Sizes

Table 7 compares ResNet, ODENet, and dsODENet in terms of the model size, the forward/backward pass size, and the total memory size for training. “Model size” means the necessary memory size to retain the model. “Memory size during training” includes the model size, the forward/backward pass size, and the input/output data sizes. These values are measured by using torchinfo which is a tool that reports the parameter size information.

The communication is required between the server and clients in each communication round. In Table 7, “# of parameters transferred” is the sum of the weight parameters of convolutional layers and fully-connected layers in these models. In the following discussion, it is simply denoted as communication size. Table 7 shows that both the ODENet and dsODENet models reduce the communication sizes compared with the corresponding ResNet model. Compared with ResNet, the communication size of ODENet-50 is 10.6% of the original ResNet model. In the case of ResNet and dsODENet, the communication size of dsODENet-50 is 5.3% of the ResNet model. These results show that the

<table>
<thead>
<tr>
<th>Model</th>
<th>Algorithm</th>
<th>Top5 [%]</th>
<th>Top1 [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODENet</td>
<td>FedAvg</td>
<td>97.02</td>
<td>69.03</td>
</tr>
<tr>
<td></td>
<td>FedDF</td>
<td>98.23</td>
<td>73.73</td>
</tr>
<tr>
<td>dsODENet</td>
<td>FedAvg</td>
<td>97.24</td>
<td>67.66</td>
</tr>
<tr>
<td></td>
<td>FedDF</td>
<td>98.40</td>
<td>75.67</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Algorithm</th>
<th>Top5 [%]</th>
<th>Top1 [%]</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODENet</td>
<td>5</td>
<td>97.22</td>
<td>69.53</td>
<td>427.89</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>97.09</td>
<td>70.49</td>
<td>439.03</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>97.95</td>
<td>74.37</td>
<td>460.23</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>98.57</td>
<td>76.48</td>
<td>693.21</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>98.71</td>
<td>78.32</td>
<td>760.04</td>
</tr>
<tr>
<td></td>
<td>$\infty$</td>
<td>98.87</td>
<td>78.55</td>
<td>1406.73</td>
</tr>
<tr>
<td>dsODENet</td>
<td>5</td>
<td>97.18</td>
<td>69.52</td>
<td>427.98</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>97.16</td>
<td>69.82</td>
<td>434.77</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>97.93</td>
<td>73.88</td>
<td>466.62</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>98.41</td>
<td>76.69</td>
<td>699.93</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>98.73</td>
<td>78.32</td>
<td>767.08</td>
</tr>
<tr>
<td></td>
<td>$\infty$</td>
<td>98.81</td>
<td>78.63</td>
<td>1454.90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th># of server samples</th>
<th>Top5 [%]</th>
<th>Top1 [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODENet</td>
<td>5</td>
<td>97.22</td>
<td>69.53</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>97.09</td>
<td>70.49</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>97.95</td>
<td>74.37</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>98.57</td>
<td>76.48</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>98.71</td>
<td>78.32</td>
</tr>
<tr>
<td></td>
<td>$\infty$</td>
<td>98.87</td>
<td>78.55</td>
</tr>
<tr>
<td>dsODENet</td>
<td>5</td>
<td>97.18</td>
<td>69.52</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>97.16</td>
<td>69.82</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>97.93</td>
<td>73.88</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>98.41</td>
<td>76.69</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>98.73</td>
<td>78.32</td>
</tr>
<tr>
<td></td>
<td>$\infty$</td>
<td>98.81</td>
<td>78.63</td>
</tr>
</tbody>
</table>

Table 7 compares ResNet, ODENet, and dsODENet in terms of the model size, the forward/backward pass size, and the total memory size for training. “Model size” means the necessary memory size to retain the model. “Memory size during training” includes the model size, the forward/backward pass size, and the input/output data sizes. These values are measured by using torchinfo which is a tool that reports the parameter size information.

The communication is required between the server and clients in each communication round. In Table 7, “# of parameters transferred” is the sum of the weight parameters of convolutional layers and fully-connected layers in these models. In the following discussion, it is simply denoted as communication size. Table 7 shows that both the ODENet and dsODENet models reduce the communication sizes compared with the corresponding ResNet model. Compared with ResNet, the communication size of ODENet-50 is 10.6% of the original ResNet model. In the case of ResNet and dsODENet, the communication size of dsODENet-50 is 5.3% of the ResNet model. These results show that the
use of ODENet and dsODENet can significantly reduce the communication size between the server and clients. The communication size increases as $N$ is increased in ResNet. On the other hand, communication sizes are constant regardless of $N$ in the cases of ODENet and dsODENet. This is because the number of physically-stacked blocks is the same in ODENet and dsODENet even if $N$ is different, and only the number of iterations of each block is different.

Please note that, since the model size is small in ODENet and dsODENet, the required memory capacity can also be reduced by these models compared with the original ResNet model. As shown in Table 7, the memory sizes to train ODENet-50 and dsODENet-50 models are 52% and 70% of that of ResNet-50, respectively. This result demonstrates that our proposed approach is beneficial in terms of the memory size for the training.

## 5. Discussions

Here, we discuss pros and cons of the proposed approach against counterparts. Additional evaluations with different hyper-parameters are also conducted.

### 5.1 Overall Comparisons

Here, we compare the proposed approach with ResNet + FedAvg and ResNet + FedDF in terms of the accuracy, communication size, and computation time. As for the comparison to ResNet + FedAvg, since FedAvg cannot aggregate ResNet models with different depths, we compare the proposed approach with the ResNet + FedAvg cases where all the clients have the same ResNet model. We assume that 30 clients join the federated learning where 10, 10, and 10 clients have ResNet-34, ResNet-50, and ResNet-101 models, respectively. The evaluation conditions are the same as those of Tables 3, 4, and 5. ResNet-50 is used as a global model. The number of communication rounds is 100, and the number of local epochs $E$ is 40. $\alpha$ is set to 10. The client participating rate $r$ is 0.2.

Table 8 shows the comparison results. In this table, “Time” represents the average time interval between two successive communication rounds. “Communication size” represents the average parameter size that a single client sends to the server in a single communication round. “Different depth” indicates whether models with different depths can be aggregated. Although baseline accuracies of ODENet and dsODENet are inherently lower than ResNet regardless of the federated learning algorithms (FedAvg and FedDF), the time interval of our proposed approach is mostly shorter than the other solutions. Also, our approach is advantageous in terms of the communication size. Another benefit of our approach against FedDF is that it does not require server-side samples, while it can aggregate models with different depths, which is a benefit against ResNet + FedAvg.

### 5.2 Comparisons to FedDF

As shown in Tables 3, 4, and 5, FedDF presents a higher accuracy compared to FedAvg. However, as shown in Table 6, accuracy of FedDF becomes comparable to our proposed approach if 5 or 10 server-side samples are only available. In addition, in FedDF, clients need to upload their models to the server[13], and thus its communication size is comparable to that of FedAvg. Please note that FedDF can improve the accuracy by increasing the number of server-side samples for the server-side knowledge distillation. As shown in Table 8, the time interval between two successive communication rounds of FedDF increases as the number of server-side samples increases, and this time interval becomes longer than that of our proposed approach. One of important benefits of the classic federated learning is that the server does not have to retain privacy-sensitive training samples. The simplicity of the server-side processing would also be a benefit. On the other hand, FedDF requires the server-side training samples. We need to consider cases where concrete or accurate server-side training samples which can be uploaded to cloud servers may not be available. Our approach does not require server-side samples, while it can aggregate models with different depths, which is a benefit against ResNet + FedAvg. As shown in Table 7, the model and training memory sizes of ODENet and dsODENet are smaller than those of ResNet. Actually, a smaller memory size can help the client heterogeneity since clients with a tighter resource limitation can join the federated learning. As such, in our approach, clients with rich compute resources can use deeper models, while those with limited resources can use shallower models.

### 5.3 Unbalanced Client Heterogeneity

In Sect. 4.1, 30 clients use one of three models with different depths (e.g., ODENet-34, ODENet-50, and ODENet-101). Ratios of these three models are balanced; that is, 10 clients,
10 clients, and 10 clients use ODENet-34, ODENet-50, and ODENet-101, respectively. Here, we conduct experiments of unbalanced cases. Specifically, we examine two cases: 1) 24 clients use ODENet-34, three clients use ODENet-50, and three clients use ODENet-101; 2) three clients use ODENet-34, three clients use ODENet-50, and 24 clients use ODENet-101. The former case is referred to as “weak-biased” and the latter case is referred to as “strong-biased”. The other evaluation conditions are the same as those in Sect. 4.1. Tables 9 and 10 show the evaluation results of these two cases. As shown in Tables 9 and 10, accuracy of the strong-biased case is slightly higher than that of the weak-biased case.

### 5.4 Homogenous Model Case

Here, we conduct experiments where all the clients have the same model, which is an ideal situation. Table 11 shows the results. $\alpha$ is set to 10. The other evaluation conditions are the same as those in Sect. 4.1. As shown in Table 11, although ODENet and dsODENet can significantly reduce the number of parameters (please see “Model size”) compared to ResNet, ODENet and dsODENet show a lower accuracy compared to ResNet. This result is reasonable since the numbers of parameters of ODENet and dsODENet are significantly small compared to ResNet. Also, the time intervals of ODENet and dsODENet are shorter than those of ResNet. When all the clients have the same model, the time intervals of ODENet and dsODENet are shorter than those of heterogenous client cases shown in Table 8. However, our proposed approach can aggregate models with different depths, while ResNet cannot allow such a heterogeneity, which is a benefit of our proposed approach.

### 5.5 Heterogeneous Epoch Number

As an alternative to our approach, customizing the number of local epochs $E$ for each client depending on the compute resource of each client would help the client heterogeneity. For example, we can increase the number of epochs for stronger clients, while we can decrease the number of epochs for weaker clients. Here, we conduct this experiment. Table 12 shows the results. We use CIFAR-10 dataset. FedAvg is used for the federated learning algorithm. The number of communication rounds is 100. The number of clients $K$ is 30, $r$ is set to 0.2; thus, six clients join the federated learning. The important difference to our proposed approach is that all the clients have the same model, while the number of epochs is different for each client. Specifically, the number of local epochs $E$ is randomly selected from 20, 40, and 80 for each client. The selected epoch number is statically fixed; thus, it is not changed during the federated learning.

As shown in Table 12, the accuracy of this heterogeneous epoch number approach is higher than our proposed approach. However, regarding the client heterogeneity, the heterogeneous epoch number approach can address the CPU
performance heterogeneity of clients since the number of epochs can be tuned depending on the CPU performance of each client. On the other hand, our proposed approach can customize the model depths depending on the CPU performance and memory size of clients; thus, our approach can address the client heterogeneity of both the CPU performance and memory capacity of clients. We believe that this heterogeneous epoch number approach can be combined with our proposed approach, and exploring this possibility is our future work.

5.6 Different Client Number and Participating Rate

We conduct an additional experiment in which the number of clients $K$ is increased to 50, in which 10 clients use ODENet-34, 30 clients use ODENet-50, and 10 clients use ODENet-101. Table 13 shows the results. In addition, we conduct another experiment in which the client participating rate $r$ is 0.1. Table 14 shows the results. In both the cases, the results are consistent with the results presented in Sect. 4.1.

6. Conclusions

In this paper, we proposed a flexible federated learning approach that can aggregate models with different iteration counts by utilizing ODENet and dsODENet as federated learning models. We demonstrated that these models with different iteration counts can be aggregated correctly (i.e., having the weight compatibility) in the cases of ODENet and dsODENet. Then, the proposed approach simply using FedAvg was compared with FedDF in terms of the accuracy. The experiment results showed that the higher accuracy of FedDF come from additional knowledge distillation overheads at the server. On the other hand, the proposed approach can simply aggregate models with different iteration counts without the server-side training nor uploading training samples to the server. In addition, ODENet and dsODENet were evaluated in terms of the model and communication sizes. Compared with ResNet-50, ODENet-50 and dsODENet-50 successfully reduced the communication sizes by 89.4% and by 94.7%, respectively. These results showed that our approach can significantly reduce the communication overhead while enabling the aggregation of models with different iteration counts.

As a future work, we will evaluate the feasibility of federated learning with ANODE [17] which is a Neural ODE based approach that utilizes a checkpointing method. We are also planning to improve accuracy when $\alpha$ is small. We will combine our approach with state-of-the-art federated learning algorithms.

References


Yuto Hoshino received the BE degree from Keio University in 2022. He is currently a master course student in Keio University.

Hiroki Kawakami received the BE and ME degrees from Keio University in 2021 and 2023, respectively.

Hiroki Matsutani received the BA, ME, and PhD degrees from Keio University in 2004, 2006, and 2008, respectively. He is currently a professor in the Department of Information and Computer Science, Keio University. His research interests include the areas of computer architecture and machine learning.
Real-Time Video Matting Based on RVM and Mobile ViT*

Chengyu WU†‡, Jiangshan QIN†, Xiangyang LI††, Ao ZHAN†‡, Nonmembers,  
and Zhengqiang WANG†††, Member

SUMMARY
Real-time matting is a challenging research in deep learning. Conventional CNN (Convolutional Neural Networks) approaches are easy to misjudge the foreground and background semantic and have blurry matting edges, which result from CNN’s limited concentration on global context due to receptive field. We propose a real-time matting approach called RMViT (Real-time matting with Vision Transformer) with Transformer structure, attention and content-aware guidance to solve issues above. The semantic accuracy improves a lot due to the establishment of global context and long-range pixel information. The experiments show our approach exceeds a 30% reduction in error metrics compared with existing real-time matting approaches.

key words: mobile ViT, video matting, deep learning, attention mechanism

1. Introduction

Matting is a popular technology in the field of computer vision. It can effectively separate the foreground objects that people are interested in from pictures or videos. High-resolution real-time video matting have great commercial value in industries such as live streaming, but it is also challenging in deep learning research.

At present, there are three representative studies on CNN-based real-time video matting, i.e., Background Video Matting V2 (BGMv2) [1], Robust Video Matting (RVM) [2], and Matting Objective Decomposition Network (MODNet) [3]. BGMv2 has constructed a high-precision model that requires users to input a static background as a constraint to achieve real-time video matting. RVM and MODNet only need the original video frames to achieve video matting of human figures. However, BGMv2 requires user guidance and a stable environment; The ability of RVM and MODNet to capture global relationships in images is insufficient. These approaches use conventional CNN structures, which have low image accuracy in dynamic and complex backgrounds, making it easy to misjudge some background objects as foreground. Moreover, the matting result is prone to generating hollow areas, making it difficult to achieve the theoretical expectations in practical use.

Video Matting with Transformer (VMFormer) [4] adopts ViT (Vision Transformer) for matting task. VMFormer outperforms MODNet, BGMv2 and RVM in terms of accuracy. However, both its encoder and decoder use Transformer, which results in the model parameters being about twice as large as RVM model. Experiment shows that VMFormer only has 3 FPS (Frames Per Second) of inference speed processing on Nvidia Geforce RTX4060 with 1080p resolution. At present, approaches like VMFormer are difficult to be applied to fields requiring real-time performance, such as live broadcast.

Existing real-time approaches are not sensitive to long-range pixels. They frequently misjudge pixels’ semantic and are not robust enough with complex background. To solve current issues of deep video matting, we design RMViT model based on the idea of RVM. Separable self-attention mechanism is introduced into the matting task to capture global information of the image. An encoder with a hybrid structure of Mobile ViT V3 [5] and inverted residual block [6] is established. The hybrid structure retains the characteristics of CNN inductive bias, and give full play to the respective advantages of CNN and Transformer. We also design an improved recurrent decoder module based on attention and content-aware guidance. The decoder is joined with CBAM [7] and CARAFE [8] operators, which has a significant improvement in upsampling process.

2. Issues and Challenges

Matting is a technology that separates foreground objects from a picture. The mathematical model is shown in Eq. (1):

\[
I = aF + (1 - a)B
\]  

(1)

Where \(I\) is the given picture, \(F\) is the foreground image, \(B\) is the background image, and \(a\) is the opacity of the foreground image. It’s under-constrain because there are 3 unknown factors with only 1 equation. Most approaches add constrains manually to solve this issue, e.g., BGMv2 requires a static background image \(B\) for input, and RVM

---

*This work was supported by the First Batch of “Pioneer” R&D Programs of Zhejiang Province in 2023 under grant 2023C01041, the Open Fund of Key Laboratory of Big Data Intelligent Computing, Zhejiang Sci-Tech University, Hangzhou, 310018, P.R.China.

†††The authors are School of Information Science and Engineering, Zhejiang Sci-Tech University, Hangzhou, 310018, P.R.China.

‡‡‡The authors are with School of Communication and Information Engineering, Chongqing University of Posts and Telecommunications, Chongqing, 400065, P.R.China.

\[a\] E-mail: jerry916@zstu.edu.cn

DOI: 10.1587/transinf.2023EDL8071
uses prior knowledge of portrait semantic. Nevertheless, added constrains result in accuracy loss or reusing cost. Existing user-guidance-free real-time matting approaches are easy to generate blur matting edges or semantic misjudgements, mainly because they are not sensitive enough to the context of images.

3. Proposed Real-Time Matting Approach

To solve current matting issues, we propose RMViT model that includes a feature extraction encoder with a hybrid structure of Mobile ViT and MobileNet V3 \[6\], bottleneck block, and a recurrent decoder with attention and content-aware mechanisms. The model accepts video frames or images as input, and outputs alpha matte as the result.

According to the model structure in Fig. 1, the original image is downsampled with factor \(k\) after input. In addition, to restore high-resolution details even after downsampling the original image, we adopt Fast Guided Filter (FGF) \[9\], which refines the low-resolution alpha image output to reconstruct the original resolution alpha matte.

3.1 Feature Extraction Encoder Based on Hybrid Structure

We propose a hybrid encoder using Inversed Residual (IR) block of MobileNet V3 and Mobile ViT V3 Block. The encoder accepts the initial downsampled image as input and outputs the processed feature map to the bottleneck block. We do not use pure Transformer structure due to following issues:

a) Mobile ViT V3 is slower than inverted residual block, which will affect the real-time performance of the model;

b) Pure ViT model lacks inductive bias characteristics and is sensitive to capacity and augmentation of the dataset.

The encoder’s parameters in this structure are as shown in Table 1. Where “IR”, “MViT" refer to inverted residual block and Mobile ViT V3 block respectively, \(L\) refers to the amount of Transformers in corresponding Mobile ViT block. “in”, “out”, “ker” and “exp” refer to input channel, output channel, convolution kernel size and expand channel size of IR blocks respectively. “se” refers to whether corresponding IR block uses short cut. “s” and “d” refer to convolution stride and dilation respectively. “act” refers to activation for IR blocks. IR blocks use hard-swish and ReLU6 as activation functions, which are represented by “HS” and “RE” respectively.

The hybrid encoder adopts separable self-attention mechanism by introducing Mobile ViT V3. Compared to most classic ViT model, the proposed structure is more lightweight. It can significantly improve its sensitivity to global context without bringing in too much computation cost.

3.2 Decoder with Attention and Content-Aware Guidance

On the basis of RVM recurrent decoder, we propose a recurrent decoder based on attention and content-aware mechanisms. The ablation experiment shows that the accuracy of the model is significantly improved after using this decoder. The decoder block is shown in Fig. 2.

There are three decoder blocks in the model as shown in Fig. 1. The main input of each decoder block comes from the previous decoder block or bottleneck block; It also accepts skip connections (SC) feature map processed by CBAM as

![Fig. 1 Model structure](image-url)
3.3 Refine Module Based on Fast Guided Filter

Considering the demand of real-time performance on high-resolution video frames, we adopt Fast Guided Filter (FGF) as a refine module. As shown in Fig. 1, the original input frame is first downsampled by factor $k$ and then processed by the encoder-decoder base network. The base network outputs a low-resolution alpha matte and send it to FGF together with original input frame. FGF module then generates refine alpha matte on original resolution. The value of $k$ can be adjusted to accommodate different input resolutions. Note that the encoder-decoder base network can process frames standalone in case of low-resolution or non-real-time operation.

3.4 Training

To achieve better performance, we used specific training methods and multiple datasets. We apply AMP (Automatically Mixed Precision) and Adam optimizer to speed up training and accelerate convergence.

The datasets we used in training process are as follows:
1. Video foreground dataset: Video Matting 240K [1];
2. Video background dataset: Deep Video Matting (DVM) [11];
3. Portrait segmentation datasets: COCO [12], Supervisely Person Dataset [13], YoutubeVIS 2021 [14];
4. High-resolution foreground image datasets: PPM-100 [3], P3M-10K [15], AIM-500 [16], Adobe Matting Dataset [17], Distinctions 646 [18],
5. Image background dataset: Indoor CVPR 09 [19].

We divide the matting training process into three parts with total 35 epochs aiming at different circumstances, as follows:

- **Part 1**: In part 1 we train the model without FGF on Video Matting 240K and DVM for 20 epochs, and only use low-resolution ($512 \times 512$) video sequences with a total length of 20 frames.

- **Part 2**: In part 2 we train the model still without FGF on hybrid resolution video sequences for 3 epochs. The hybrid video sequences are from the same datasets as part 1, containing low-resolution frames ($512 \times 512$) with a length of 10 and high-resolution frames ($2048 \times 2048$) with a length of 3.

- **Part 3**: In part 3 we train the model for high-resolution image matting task for 12 epochs. The model is trained on high-resolution foreground image datasets (P3M-10K, Distinctions 646, etc.) and image background dataset (Indoor CVPR 09). We add FGF, setting the initial downsample factor $k$ to 0.25 during the former 10 epochs and 1.0 during the latter 2 epochs.

Portrait segmentation training is interspersed throughout the entire training process. We insert one segmentation training step after every 2 matting training steps to ensure the model’s sensitivity to human figures.

To ensure segmentation and matting performance, we apply different losses and weight them as a total loss $L_t$. Considering details at the edge of foreground images, we apply pyramid Laplacian loss $L_{lap}$ [20] as well as L1 loss $L_1$. Moreover we apply a temporal coherence loss $L_c$ to reduce flickers of generated frames. The losses are as follows:

$$ L_1 = ||\hat{a} - a||_1 $$

$$ L_{lap} = \sum_{j=1}^{5} 2^{j-1} ||L_j(\hat{a}) - L_j(a)||_1 $$

$$ L_c = ||\frac{d\hat{a}}{dt} - \frac{d\alpha}{dt}||_2 $$

$$ L_d = L_1 + \frac{1}{5} L_{lap} + 5L_c $$

Where, $\hat{a}$ represents predicted alpha matte, $a$ represents ground truth, and $L_j(a)$ represents the computed result of the $s$-th layer of Laplacian pyramid based on $a$.

4. Experiments

The training and evaluating process uses Nvidia Geforce RTX 4060 and RTX 3060 for multi-card training, with AMD Ryzen 9 5950X CPU and mixed precision throughout the entire process. During the evaluating process, RTX 2070 laptop and Intel Core i7-9750H are also used for speed test.

4.1 Comparison Experiments

We compare the accuracy of the existing model and the proposed model on four indicators, i.e., MAD, MSE, Connectivity Error (Conn), and Gradient Error (Grad). The evaluation dataset used in this experiment is Video Matte 240K HD, with a total of 50 video clips. For the convenience of data display, we magnify MAD and MSE results by 1000 times, while the Conn and Grad values are reduced to one thousandth of the original values. The experimental results of numerical evaluation are shown in Table 2. It can be seen that RMViT has better results in image retrieval.
accuracy compared to CNN-based models. BGMv2 uses a static background as the constraint, it is unstable in dynamic backgrounds and has poor anti-interference ability. Similarly, MODNet outperforms RMViT in the Grad metric, but all other error metrics are higher than RMViT. RVM has good stability for background changes, but its lack of global information perception and the lack of attention mechanism in the decoder make it comprehensively lag behind RMViT in evaluation experiments.

Figure 3 shows the visualization results of video matting, where two video frames under dynamic backgrounds are selected from Video Matte 240K HD [1]. From Fig. 3, it can be observed that RMViT can relatively clearly recognize the edges of the foreground in dynamic backgrounds. This means that it can more accurately extract foreground targets and has good semantic recognition ability when dealing with dynamic scenes. MODNet performs well in image edge detection, but there are semantic recognition errors that mistakenly recognizes background pixels as foreground or loses foreground pixels. RVM results in blurry edges and inability to clearly identify foreground and background pixels, and lacks sufficient stability and accuracy when dealing with dynamic backgrounds.

According to the results above, RMViT shows its accuracy in complex dynamic background matting. RMViT has fewer semantic errors and sharper edges in matting results. It also can be seen that RMViT has reduced Grad by 35% and Conn by 48% respectively, compared to RVM, meaning that our approach is more robust under such circumstance.

### 4.2 Size and Speed

We compared existing approaches to ours on size and speed evaluation. Table 3 shows that our model is lighter and has fewer parameters compared to BGM and MODNet. Compared with RVM, our model has only increased the number of parameters by 1.3% and size by 2.8%, but it has exceeded a 30% reduction in error metrics, indicating the effectiveness of our approach.

To verify the real-time performance in actual use, we test different approaches in a practical environment which includes video capturing, video codecs, frame preprocessing, data parallel and rendering. The models are tested on a laptop equipped with Intel Core i7-9750H CPU and RTX 2070 laptop GPU. The TDP (Thermal Design Power) of CPU and GPU are set to 80W and 115W respectively, so as to demonstrate whether the approaches can maintain their real-time performance on mid-end or low-end devices. All models are tested on a 1080p video sequence.

As shown in Table 4, FPS and GPU core usage are measured. In the experiment we set downsample factor \( k = 0.18 \). The result shows that RMViT has analogous speed compared to existing real-time approaches. BGMv2 and MODNet exceed 75% of GPU usage while RMViT and RVM are within 40%. RMViT achieves 30 FPS on 1080p resolution as well as relatively low GPU usage cost in practical environment, showing that our method is considered real-time for conven-

---

### Table 2: Comparison of matting evaluation scores

<table>
<thead>
<tr>
<th>Approaches</th>
<th>MAD</th>
<th>MSE</th>
<th>Grad</th>
<th>Conn</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGMv2</td>
<td>23.45</td>
<td>18.74</td>
<td>10.66</td>
<td>41.96</td>
</tr>
<tr>
<td>MODNet</td>
<td>9.73</td>
<td>4.60</td>
<td>9.43</td>
<td>11.96</td>
</tr>
<tr>
<td>RVM</td>
<td>10.80</td>
<td>3.72</td>
<td>18.26</td>
<td>15.63</td>
</tr>
<tr>
<td>RMViT(Ours)</td>
<td>7.12</td>
<td>2.32</td>
<td>11.79</td>
<td>8.05</td>
</tr>
</tbody>
</table>

### Table 3: Comparison of model size

<table>
<thead>
<tr>
<th>Model</th>
<th>Size(MB)</th>
<th>Params(millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGMv2</td>
<td>19.3</td>
<td>5.04</td>
</tr>
<tr>
<td>MODNet</td>
<td>25.0</td>
<td>8.78</td>
</tr>
<tr>
<td>RVM</td>
<td>14.5</td>
<td>3.77</td>
</tr>
<tr>
<td>RMViT(Ours)</td>
<td>14.9</td>
<td>3.82</td>
</tr>
</tbody>
</table>

### Table 4: Comparison of speed

<table>
<thead>
<tr>
<th>Model</th>
<th>Speed(FPS)</th>
<th>GPU Usage(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGMv2</td>
<td>33</td>
<td>86</td>
</tr>
<tr>
<td>MODNet</td>
<td>28</td>
<td>75</td>
</tr>
<tr>
<td>RVM</td>
<td>35</td>
<td>36</td>
</tr>
<tr>
<td>RMViT(Ours)</td>
<td>30</td>
<td>37</td>
</tr>
</tbody>
</table>
Table 5 Ablation experiment

<table>
<thead>
<tr>
<th>Model</th>
<th>MAD</th>
<th>MSE</th>
<th>Grad</th>
<th>Conn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ablation model 1</td>
<td>8.01</td>
<td>2.37</td>
<td>14.24</td>
<td>9.62</td>
</tr>
<tr>
<td>Ablation model 2</td>
<td>10.87</td>
<td>3.58</td>
<td>18.32</td>
<td>15.44</td>
</tr>
<tr>
<td>FGF-free model</td>
<td>6.71</td>
<td>2.05</td>
<td>7.22</td>
<td>6.98</td>
</tr>
<tr>
<td>Original model</td>
<td>6.14</td>
<td>1.44</td>
<td>10.76</td>
<td>5.97</td>
</tr>
</tbody>
</table>

4.3 Ablation Experiment

To investigate the impact of Mobile ViT, CBAM, CARFAFE and FGF modules on the overall accuracy of the model, an ablation experiment is conducted, as follows:

a) Ablation model 1: Retain Mobile ViT V3 module in the encoder, and eliminate the optimization of attention and content-aware mechanisms in the decoder;

b) Ablation model 2: Simultaneously eliminate Mobile ViT V3 modules in the encoder and the attention and content-aware mechanisms in the decoder;

c) FGF-free model: Only eliminate FGF, and directly process high-resolution videos instead of downsampling by factor \( k \) initially;

d) Original model: Do not eliminate any structure and includes all modules.

In this experiment, former 10 video clips from Video Matte 240K HD test set are selected. Default downsampling factor \( k \) is set to 0.25.

The experiment results are shown in Table 5. It can be seen that self-attention mechanism brought about by the ViT structure in the encoder, as well as the attention and content-aware mechanisms in the decoder, have a significant improvement effect on model accuracy. Result of FGF-free model shows that there is a reduction on Grad, while other three error metrics slightly increase. FGF and initial downsampling process do not have significant negative impact on accuracy. It is considered feasible to speed up inference with FGF refine module.

5. Conclusion

We introduce a matting approach named RMViT due to the low accuracy and semantic misjudgement. We proposed a hybrid matting model under circumstance of real-time matting task. The proposed approach adds separable self-attention mechanism in hybrid encoder, and designs decoder modules joined with attention and content-aware guidance, which make the model establish enough global context information so that the approach makes fewer semantic mistakes and sharper edges. The experiments show that our approach is better than MODNet, BGMv2 and RVM while ensuring real-time performance.

References


