Multimedia Tools and Applications
https://doi.org/10.1007/s11042-024-19657-z

1235: SUPPORTING MEDICINE AND HEALTHCARE WITH MULTIMEDIA TOOLS

Developing diverse ensemble architectures for automatic brain tumor classification

Gergo Bogacsovics[1] · **Balazs Harangi**[1] · **Andras Hajdu**[1]

Received: 12 December 2023 / Revised: 3 April 2024 / Accepted: 7 June 2024

Abstract
Brain tumors pose a serious threat in our modern society, with a clear increase in global cases each year. Therefore, developing robust solutions that could automatically and reliably detect brain tumors in their early stages is of utmost importance. In our paper, we revisit the problem of building performant ensembles for clinical usage by maximizing the diversity of the member models during the training procedure. We present an improved, more robust, extended version of our framework and propose solutions that could be integrated into a Computer-Aided Diagnosis system to accurately classify some of the most common types of brain tumors: meningioma, glioma, and pituitary tumors. We show that the new framework based on the histogram loss can be seen as a natural extension of the former approach, as it also calculates the inner products of the latent vectors produced by each member to measure similarity, but at the same time, it also makes it possible to capture more complex patterns. We also present several variants of our framework to incorporate member models with varying dimensional feature vectors and to cope with imbalanced datasets. We evaluate our solutions on a clinically tested dataset of 3,064 T1-weighted contrast-enhanced magnetic resonance images and show that they greatly outperform other state-of-the-art approaches and the base architectures as well, achieving over 92% accuracy, 92% macro and weighted precision, 91% macro and 92% weighted F_1 score, and over 90% macro and 92% weighted sensitivity.

Keywords Deep learning · Brain tumor · Ensemble learning · Diversity · Screening systems

✉ Gergo Bogacsovics
bogacsovics.gergo@inf.unideb.hu

Balazs Harangi
harangi.balazs@inf.unideb.hu

Andras Hajdu
hajdu.andras@inf.unideb.hu

[1] Department of Data Science and Visualization, Faculty of Informatics, University of Debrecen, PO Box 400, 4002 Debrecen, Hungary

Published online: 24 June 2024

1 Introduction

As a form of disease, cancer poses a serious threat in our modern society, affecting a significant portion of the population. Recent research has shown that it affects even the younger population [1] and that the incidence and mortality rates increase drastically with aging [1]. The same phenomenon – the increased risk of being diagnosed with said tumors with age – has been observed in the case of brain or central nervous system (CNS) tumors [2]. Brain tumors can develop when the brain's cells grow irregularly and abnormally, forming a mass inside the patient's skull. This results in various symptoms, including nausea, headaches, seizures, altered mental status, and even death [3] since the skull only has limited space. The work [4] reported approximately 300,000 global cases of brain and CNS cancer in 2019, which, according to [4], meant a total of 94.35% increase since 1990. The Central Brain Tumor Registry of the United States (CBTRUS) [5] reported 84,264 cases of death between 2015 and 2019, where the cause of death was a malignant brain or CNS tumor. CBTRUS also reported [5] an estimated amount of 93,470 new cases of malignant and non-malignant (benign) brain and CNS tumors for 2022, only in the U.S. alone. According to the latest reports, brain and other nervous system-related cancers resulted in more than 18,000 estimated cases of death in the U.S. in 2023 [6], while the number of new cases was estimated to be more than 24,000 [6]. These statistics all highlight the importance of developing tools that could help with the accurate and reliable detection of these tumors. The problem, however, lies in the fact that these new cases and the sheer number of patients currently being treated, coupled with the newly diagnosed patients, generate an increasing amount of image data that the limited number of clinicians struggle to handle. An automatic Computer-Aided Diagnosis (CAD) system could help not only process and evaluate this image data but could potentially discover tumors in their earlier stages due to the increased speed of the clinical workflows. In this work, we propose a CAD system that can process magnetic resonance imaging (MRI) images and automatically classify different types of brain tumors with high accuracy and reliability. We primarily focus on three of the most commonly occurring types of brain tumors [5]: meningioma, glioma, and pituitary tumors. For our research, we used the publicly available dataset published by Jun Cheng [7] that was used in [8].

In recent years, tools that applied deep learning and particularly convolutional neural networks (CNNs) have revolutionized clinical research. These approaches provided very efficient solutions for several challenging and crucial tasks in the medical workflows, like in breast cancer [9, 10] and brain tumor [11, 12] detection and also in automatically identifying skin cancer [13]. Nevertheless, the performance of these deep learning-based techniques is rather volatile; it may vary from application to application. We can easily experience that a specific model is very efficient in one task while its performance falls back in another scenario. We can fuse the different models relying on various architectures into an ensemble as a remedy for such cases. In this way, we can raise the overall performance since member models with weak performance for a given input can be compensated with others performing better there. Naturally, we can expect improvement only if the better-performing models are in a majority within the ensemble. To address this issue, there is strong ongoing research on organizing the individual member models into an efficient ensemble [14–16]. Applying ensemble-based systems has numerous advantages in the clinical domain. A major benefit is that they can increase the overall stability and performance of the model since their statistically more robust internal mechanisms make them more resistant to outliers or irregularities in the inputs. This feature leads to a more efficient solution and better generalization characteristics.

Our paper presents a novel ensemble-based method for the reliable and accurate classification of brain tumors from T1-weighted contrast-enhanced MRI images. The improved methods we introduce in this study benefit routine healthcare by reducing the time needed to diagnose patients and raising the quality of the clinical workflow. Our approach can be taken into consideration to integrate it in a (semi-)automated CAD system to provide better healthcare services, reduce the burden on clinical experts, accelerate routine jobs, and, perhaps most importantly, enable the detection of brain tumors at an early stage. We hope to help increase the survival rate of patients with brain cancers studied in our research. For building the ensemble models, we use some of the most commonly used state-of-the-art CNN architectures as the member models of the ensembles, such as AlexNet [17, 18], MobileNetv2 [19–21], EfficientNet [22, 23], and ShuffleNet v2 [21, 24]. We show how our new method proposed in this paper builds upon our existing work [25] and that it is a natural and logical extension of our original framework with a theoretically better-founded formalization. We also detail why our new solution is more reliable and highlight how it can overcome most of the disadvantages and shortcomings of our previous solution. We also describe the theoretical uncertainty when using the cosine similarity function to compare the features extracted by the ensemble members used in our previous framework. To this end, we present a novel solution using the histogram loss [26] that further extends our previously proposed framework while being more robust and theoretically better founded. We also expand on using a weighted loss function, show its effect on our framework, and conclude that using such a weighted loss function does not hinder the optimization of diversity and shows that our framework can still converge to a local minimum. For our results, we consider all of the most popular standard metrics, such as accuracy, sensitivity, precision, and F_1 score. To showcase the improvement our approach brings to this field, we compare its performance with the current state-of-the-art detection methods considering convolutional neural network-based (CNN) architectures. We will show that our solution outperforms these commonly used models by a large margin.

2 Materials and methods

Deep learning-based solutions have been used to solve a wide variety of problems in the area of medical prognosis and diagnosis in the last decade [9–12]. The main reasons behind the popularity of such solutions are the good performance and generalization capabilities of the various novel CNN-based architectures and the fact that they can learn how to solve a given problem efficiently, which makes it possible for the models to learn the most suitable features for the given problem automatically. This latter property had an especially strong impact on the field, as robust solutions became available without human experts to manually extract features from the images. Deep learning-based solutions can do that directly, leading to an improved and accelerated workflow. Nevertheless, the underlying architecture of these deep learning-based models may strongly influence their performances. Namely, a specific architecture may be very efficient concerning a given task, while its performance may drop for another. The converse also holds, i.e., a model may perform poorly on one task but perform well on another. Accordingly, we can see a continuous research effort on fusing different models into an ensemble which generalizes better than the individual approaches [14–16]. However, even though it has already been shown that diverse ensembles result in better performance [27, 28], in practice, the quality of the ensembles is usually not considered or directly measured. In other words, when building an ensemble from some models, it is usually not checked whether said models work differently (i.e., by operating on different

sets of features) or in exactly the same way. This is problematic, as building ensembles of highly similar models leads to sub-optimal performance due to the lack of variety. On the other hand, a diverse ensemble built of dissimilar models could lead to much better results.

In this section, we present multiple viable options for training reliable ensemble models such that the diversity of the model is theoretically guaranteed and measured during the training process. First, we give a quick overview of the related works. Then, we introduce the dataset that was used for our experiments. Next, we explain the diversity problem in detail while highlighting the most pressing problems of traditional ensemble models. Then, we briefly summarize our previous findings [25] and show how we can use the features extracted by each member model to measure the overall diversity of the ensemble and how to penalize each member based on their similarities to the other members while they are being trained. After this, we highlight potential problems regarding the originally used cosine similarity. We include some concrete examples using the relatively simple MNIST [29] and Medical MNIST [30] datasets to illustrate why the application of this measure might be problematic in calculating the similarity of the ensemble members. As a remedy, we present a novel approach that extends our previous work by incorporating the histogram loss [26] instead of the cosine similarity. We also detail that the new and improved framework has superior theoretical properties that can make our approach more robust in capturing correlations and similarities between the extracted feature vectors. We then extend both frameworks so that including multiple different architectures becomes possible and lastly, we show that our proposed solution equipped with a weighted loss function can also be applied to imbalanced datasets, which are very common in healthcare due to the nature of medical data.

2.1 Related works

Developing performant and reliable CAD systems for automated medical diagnosis has been in the centre of research for a long time. In the earlier phases of this research, many solutions applied traditional image processing techniques to achieve this goal. For example, in [31], the authors examined the applicability of the Hough transform and canny edge detection to automatically extract features from medical images, while in [32] a novel clustering technique was proposed to cluster regions of CT brain scans that relied on the fuzzy set theory. However, with the appearance of deep learning, experiments that relied on such traditional image processing techniques gradually faded into the background and were replaced by machine- and deep learning-based solutions. For example, the authors of [33] proposed a novel framework to classify prostate cancer using MRI scans. The approach used a combination of two imaging modalities, namely diffusion-weighted (DW) and T2-weighted (T2W) MRI scans. The proposed framework first extracted a large number of image feature descriptors for the identified regions of interest (ROIs). Then, many standard machine learning algorithms were employed, such as support vector machine (SVM), random forest, decision tree and linear discriminant analysis (LDA). Out of these techniques, SVM achieved the best overall results with 88.75% accuracy, 81.08% sensitivity, and 95.35% specificity. This approach was later improved upon in [34], where the authors used a two-stage approach to segment the prostate and lesion and to identify the ROIs, achieving segmentation results of 99.78% and 98.52% accuracy and 93.64% and 99.25% Dice scores for the prostate and lesion, respectively.

Continuing this trend, during recent years, a large number of research focused specifically on the classification of brain tumors as well. Among these, there have been many which used the dataset published by Jun Cheng [7]. In [35], the authors proposed 5 convolutional neural network architectures, which were much simpler than other state-of-the-art methods

while still achieving satisfactory results. The highest accuracy that the authors achieved was 84.19%. This research had a number of serious limitations. First, only a few really simple architectures were used: the simplest one had only one convolutional layer and one dense layer, while even the most complex one had only three convolutional layers, followed by a single dense layer. Secondly, in an attempt to handle the imbalanced dataset, more than half of the meningioma cases and more than 200 pituitary tumor images were discarded. In [36], a modified capsule network architecture was proposed. To improve the solution's performance, the authors concatenated the bounding box coordinates of the given tumor with the output of the capsule layer before being fed through some fully connected layers. The proposed solution achieved 90.89% accuracy on the dataset. Although this approach applied deep learning, it had the limitation that it still relied on the coarse boundaries of the tumors as an additional input. In their subsequent work [37], the authors used an improved version of the architecture with two capsule network experts, achieving an accuracy of 91.3%. They also evaluated the method for lung nodule classification, where they achieved 90.7% accuracy, 89.5% specificity, and 89.5% sensitivity. In [38] a combination of wavelet and discrete cosine transform representations were applied to extract the features in the input images. After these features were extracted, they were fed through 2 hidden layers of a neural network to classify the brain tumors. The authors argued that even though the resulting framework only required a fraction of the computation that standard CNNs would need, it could still outperform the latter, achieving 84.12% accuracy.

Ensembles have also been used in the medical field with great success. In [39], a hybrid ensemble was proposed to detect lung and colon cancer. First, a number of pre-trained architectures were used to automatically extract features from the images. Then, these features were fed through a variety of machine learning models, namely random forest, SVM, logistic regression, a neural network, an extreme gradient boosting (XGB) and a light gradient boosting (LGB) algorithm. Lastly, majority voting was used to combine the outputs of the various algorithms. The authors achieved 99.05%, 100%, and 99.3% accuracy for lung cancer, colon cancer, and lung and colon cancer detection, respectively. Similarly, the authors of [40] also used an ensemble model to detect gastric cancer. After carefully selecting the most important features using chi-squared test and a mutual information method, a series of machine learning models were applied, which were the following: logistic regression, ridge regression, lasso regression, elastic net, random forest, gradient boosting decision trees, and neural networks. After training these models, the two best-performing options were chosen and integrated into an ensemble using weighted averaging. The probability of the gastric cancer occurrence was then combined with another set of features and used to predict the associated deaths. The authors concluded that out of the previously listed algorithms, the neural network and the gradient boosting trees resulted in the best ensemble model, achieving 97.9% and 76.3% for predicting gastric cancer and the number of deaths, respectively. In [41], the authors used an ensemble to predict whether if a given RNA sequence is cancerous or not. For this, they considered three types of cancers (lung adenocarcinoma, stomach adenocarcinoma, and breast invasive carcinoma) and multiple RNA sequencing datasets. First, they filtered the input data to only select genes that played a bigger role in predicting the given classes. Then, they trained multiple traditional machine learning models, such as k-nearest-neighbor (KNN), SVMs, decision trees, and gradient boosting decision trees. After this, they created a new dataset using the predictions of the individual models as inputs and the original class labels as outputs. This dataset was then used to train a neural network to predict whether the input was a normal sample or one with a tumor. The proposed methodology achieved 98.80%, 98.78%, and 98.41% accuracy for lung adenocarcinoma, stomach adenocarcinoma, and breast invasive carcinoma, respectively. The authors of [42] proposed a fuzzy distance-based ensemble

to detect cervical cancer using Pap smear images as input. They used three different deep learning models, namely Inception V3 [43], MobileNet V2, and Inception ResNet V2 [44], and applied transfer learning. Then, the outputs of these models were aggregated using an ensemble method that used the Euclidean and Manhattan distances as well as the cosine similarity to minimize the error values between the observed and ground truth values. These distances were then defuzzified using the product rule. The proposed fuzzy ensemble technique achieved 96.96% accuracy on the given dataset. Lastly, in [45] the authors used an ensemble of traditional and deep learning-based techniques for brain tumor classification. For each image, they calculated several characteristics based on the gray level co-occurrence matrix (GLCM) and combined them with the features extracted by a VGG-16 [46] neural network. Then, they used an SVM and a KNN algorithm to process the combined feature vectors. The authors evaluated the performance of the approach on two different datasets, achieving 96% and 93.3%, and 98.7% and 99% accuracy for the KNN and SVM algorithm on the datasets, respectively. Table 1 shows a brief overview of the previously mentioned solutions.

The main limitation of all of the previously mentioned ensemble-based studies is that they do not measure the diversity of the ensembles on any level. That is, there is virtually no guarantee that the members operate in different ways to justify their combination into an ensemble model. If there is substantial redundancy in the features that the members use or in their internal workings, the overall performance of the ensemble will unavoidably be sub-optimal. Therefore, it is important to build such methods that can guarantee the beforementioned diversity of the ensemble in a clear, theoretically founded way. To overcome this problem, we developed a novel framework with two versions, which we will introduce in Sections 2.5 and 2.7.

2.2 Dataset

We used the brain tumor dataset published by Jun Cheng [7] for our research to build a CAD system capable of automatically detecting and classifying brain tumors. We considered this dataset a good choice for a pragmatic examination and evaluation of the overall performance of our proposed solutions and to test whether they are robust enough for the following reasons. First, the dataset contains more than 3,000 images, which can be sufficient to train deep learning-based solutions that need huge amounts of data. It has also been tested and verified in the clinical setting and was used as the foundation of much research in the last few years [8, 35–38]. Moreover, it contains five pre-defined cross-validation splits, making it straightforward to compare our results with that of other research since the data we used for training and testing is the same used in other works. Due to this, we can directly take the metrics reported by the other papers and use them as baselines during the evaluation phase. The five pre-defined cross-validation splits make drawing statistically significant and valid conclusions easier, making it harder for the evaluation process to be affected or swayed by outliers or models that only perform well on a few of these splits.

The dataset [7] contains a total of 3,064 T1-weighted contrast-enhanced MRI images, which are divided into the previously mentioned five cross-validation splits. The images were obtained from 233 patients who suffered from one of the three brain tumors meningioma, glioma, or pituitary tumor. The classes are disjoint and each image in the dataset only contains one tumor. Although there were some differences in terms of the total number of images for each category, at first glance, it did not seem to be to the extent to warrant any up- or downsampling before training the models. Due to this, we first experimented with non-

Table 1 A brief overview of the related studies and the reported accuracies

Study	Year	Dataset	Type of approach	Accuracy (%)
Xiao et al. [41]	2018	RNA sequence	ensemble	98.80%, 98.78% and 98.41%
Abiwinanda et al. [35]	2019	brain tumor	convolutional neural networks	84.19%
Afshar et al. [36]	2019	brain tumor	capsule network	90.89%
Fasihi et al. [38]	2020	brain tumor	wavelet and discrete cosine transform and dense layers	84.12%
Afshar et al. [37]	2021	lung nodules and brain tumor	capsule networks	90.7 % and 91.3%
Ayyad et al. [33]	2022	prostate cancer	machine learning	88.75%
Talukder et al. [39]	2022	lung and colon cancer	ensemble (majority voting)	99.05%, 100%, and 99.3%
Baradaran et al. [40]	2022	gastic cancer	ensemble (weighted averaging)	97.9%
Pramanik et al. [42]	2022	cervical cancer	ensemble (fuzzy)	96.96%
Balaha et al. [34]	2023	prostate cancer	attention U-Net	99.78% and 98.52%
Kibriya et al. [45]	2023	brain tumor	ensemble (custom)	96% and 99%

weighted loss functions, but later, we expanded our experiments by using a weighted loss function to test the robustness of our proposed framework (see Section 2.9). All images in the dataset originate from one of the three anatomical planes: the axial, coronal, and sagittal. The exact number of images for each type of tumor and anatomical plane in the dataset can be seen in Table 2, while Fig. 1 shows some images sampled randomly from the dataset.

2.3 Traditional ensemble methods

When constructing traditional ensembles with $n \in \mathbb{N}$ arbitrary traditional, machine-, or deep learning models, each model M_j $(j = 1, \ldots, n)$ is trained individually on its own. After each model M_j has been trained, they are organized into an ensemble. We will denote such ensembles as Ens. Given the models $M_1, \ldots, M_n$, we will formulate an ensemble Ens constructed from these models as

$$Ens := \{M_1, \ldots, M_n\}. \tag{1}$$

The most important characteristic of traditional ensemble methods is that the models $M_1, \ldots, M_n$ are trained individually, with no interaction between them during the training process. For inference, i.e., for using the ensemble for performing predictions on real or test data, the outputs of the different models are combined and aggregated in a fixed and pre-defined manner. Some of the most common techniques for performing this aggregation are majority voting [39] and weighted averaging [39, 40]. We will use Ens_M to denote a majority voting ensemble, and Ens_W to denote weighted averaging.

For majority voting, the outputs of each model M_j $(j = 1, \ldots, n)$ are taken, and the class label with the highest count (i.e., the mode of the outputs) is considered as the final prediction of the ensemble Ens. Therefore, given the $x_1, \ldots, x_m$ inputs of a particular dataset, we define the output of Ens_M for each input x_i as

$$Ens_M(x_i) := majority(M_1(x_i), \ldots, M_n(x_i)). \tag{2}$$

In (2), $majority()$ returns the class label with the most occurrences, if any, otherwise, if the outputs do not differ, a default value is returned. When two or more classes receive the same number of votes during inference, one of them is selected randomly.

In the case of weighted averaging, a coefficient $\beta_j \in [0, 1]$ is computed for each model M_j where $\sum_{j=1}^{n} \beta_j = 1$. The coefficients are computed by first evaluating each trained model according to a chosen metric, then assigning higher β_j values to models achieving better results from the perspective of the given metric, and assigning lower coefficients to weaker models. Then, the final output of the ensemble is calculated as the weighted average

Table 2 The total number of images in the original dataset [7] grouped by (a) tumor type, and (b) anatomical plane

(a)		(b)	
Tumor type	Number of images	Anatomical plane	Number of images
Meningioma	708	Axial	994
Glioma	1,426	Coronal	1,025
Pituitary tumor	930	Sagittal	1,045

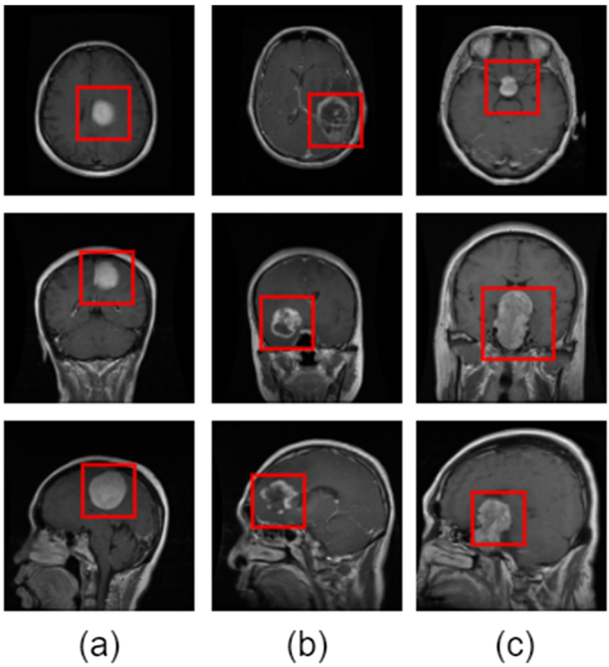

Fig. 1 Input images sampled from the three classes present in the dataset: (a) meningioma, (b) glioma, and (c) pituitary tumor. Images in the same column belong to the same class, while the red boxes indicate the locations of the tumors

of the output of each model, formalized as

$$Ens_W(x^{(i)}) := \beta_1 M_1(x_i) + \cdots + \beta_n M_n(x_i). \tag{3}$$

2.4 Diversity in ensemble models

Research has shown that diversity – if present – has a very beneficial effect on neural network-based ensembles [27, 28] and that diversity correlates with the overall performance of the ensemble [27], resulting in better performing and overall more robust models. The work [27] also mentions that apart from the overall diversity, it is also important to consider how this diversity is distributed between the ensemble models and concludes that more uniformly distributed errors between the members lead to higher levels of diversity. This suggests that even if the models of the ensemble work differently, there are still additional steps to optimize the ensemble's overall performance by ensuring that the diversity between each model is distributed uniformly. Despite this, most of the ensemble methods that are applied currently do not measure the overall diversity of the ensembles on any level [39–42]. Instead, they assume that building ensembles using models with different architectures should result in diverse ensembles. This is one of the reasons why most of the time, only simpler solutions, like majority [39] and weighted average voting [39, 40] are applied. This is highly problematic, as without any theoretical guarantees regarding diversity, there is no guarantee that the member models of the ensemble work differently. This means that, in theory, these members could use the same sets of features and operate identically, making the usual advantages of ensembles

– i.e., the increased generalization capabilities of the models – disappear. In later sections, we will show that this problem can often arise in practice and should be taken seriously when building ensemble models.

The problem regarding diversity is that it is not always clear how it should be measured. We have shown in [25] that, given an ensemble model, one trivial way to measure the model's diversity is to consider only the predictions of the member models that constitute the ensemble and the similarities between them. Then, we can penalize the models based on how similar their predictions were. However, as Fig. 2 shows, this contradicts the whole optimization process since one part of the optimization drives the predictions of all the models to be the same as the ground truth label, while the other part of the optimization process tries to force the predictions to be dissimilar to each other. As a result, this trivial approach does not lead to convergence, as these two parts cancel each other out.

In the coming sections, we first summarize our previous work [25] that provides a way of addressing this problem. We show that the cosine similarity function can be used to measure the diversity between the members of the ensemble and also give a short overview of the core ideas of our research. Then, we highlight some undesired properties of the cosine similarity function and show special cases when using it may lead to incorrect results or undesired behavior. Lastly, we introduce our new, extended framework that uses a histogram representation of the similarities of the latent feature vectors extracted by each member model.

2.5 Using the cosine similarity as a metric of diversity

This section shows how our previously proposed framework can solve the problems mentioned in Section 2.4 when building robust ensembles. We show that the increased amount of training time characterizing ensembles can be drastically decreased by training the member models simultaneously by following our proposed framework. This way, each batch is fed to all ensemble models simultaneously, reducing the number of redundant operations (i.e., loading the batches separately). Instead, all the inputs in each batch are passed to each member model, and predictions are optimized simultaneously. This is highly beneficial regarding training speed, as a huge portion of the training time is spent on I/O operations with loading data when training CNN models. This is even worse for ensembles since the member models

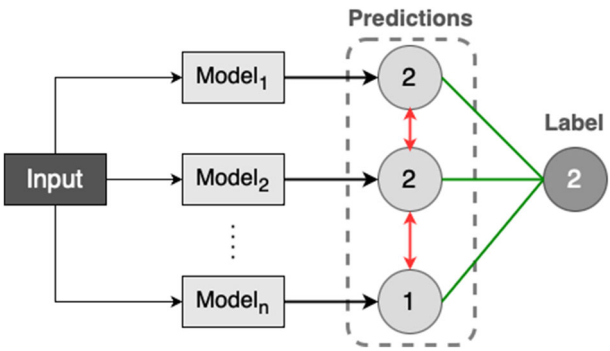

Fig. 2 Forcing each member model inside an ensemble to have dissimilar predictions (red arrows) contradicts the original optimization objective of minimizing the difference between the label and the predictions (green lines)

need to load essentially the same data, which, in the case of n models, means loading the same inputs and labels n times. By reducing the unnecessary I/O operations, we can decrease the total training time of the ensemble significantly, which means a T/n decrease, where T is the time it takes to train all member models separately. Our proposed framework also directly addresses the problem of re-using the same architecture by solving the last problem, diversity. It does so by measuring the models' diversity and driving each model to use different features. Without this step, there would be no guarantee that the member models act differently on any level. It would make re-using the same architecture nearly impossible with simpler methods like majority voting or weighted averaging since – without substantial differences between the models – there would be little to no variety or diversity in the outputs of said models.

The framework proposed in our previous work for training diverse ensembles [25] was based on the cosine similarity function. The core idea of our method is that instead of considering the direct outputs of the member models, which did not work efficiently, we can use the features extracted by the models to measure their similarity. This is a viable option to measure similarity as models operating on different sets of features should behave differently; hence, it is worth combining them in an ensemble. Our proposed framework, therefore, uses the features extracted by each model and directly optimizes diversity as a function of similarity among the member models of the ensemble during the training process. This, in practice, means a completely different internal operation compared to regular methods like majority voting or weighted average voting, where each member model of the ensemble is trained individually first, without any interaction between them, and the ensemble is only built after all the models have been trained already. In contrast, our method trains all the member models simultaneously, where inputs are fed to all the different models, batch by batch. This makes it possible for the member models to interact with each other while they are being trained, resulting in highly abstract information, like the similarity of the features extracted, to be able to flow between the member models during the training process. In [25], we showed that our solution can be applied to all CNN-based architectures. We proposed a general method that breaks down any given CNN into two main blocks, denoted by E and D. E is constructed of convolutional layers and is responsible for extracting some abstract, high-level features from the input image x_i, while D is made up of dense layers and is responsible for transforming the features extracted by E into probability scores or in other words, predictions. Using these notations, a standard forward pass for any given input image x_i can be formulated as

$$\widehat{y_i} = D(E(x_i)). \tag{4}$$

We make the following claims and use the following notations to generalize our solution. Let us enclose $n \in \mathbb{N}$ members $M_1, M_2, \ldots, M_n$ in our ensemble with corresponding sets of weights $\theta_1, \theta_2, \ldots, \theta_n \in \mathbb{R}$. Let our training set contain a total of m elements, with $x_1, x_2, \ldots, x_m$ denoting the inputs (images), and $y_1, y_2, \ldots, y_m$ denoting the ground truth labels. Furthermore, let the cost function to be optimized be denoted as $J(\theta_1, \theta_2, \ldots, \theta_n)$, or $J(\theta)$ for short. Moreover, let $E^{(j)}$ and $D^{(j)}$ denote the feature extraction (convolutional layers) and prediction (dense layers) steps for each model M_j for $j = 1, \ldots, n$ in our ensemble, respectively. Lastly, we will refer to the framework using the cosine similarity as Ens_{cos}.

Using these notations, we can declare that $E^{(j)}$ is responsible for extracting the most important features from the input image, and then $D^{(j)}$ uses these highly abstract features, represented by the extracted feature vector, and apply some non-linear transformations to it to get the predictions. Therefore, we can treat each $E^{(j)}(x_i)$ as a compact, low-dimensional representation of the original input image x_i. This representation holds all the crucial infor-

mation regarding the most important features extracted by M_j about the image x_i that may be important for the classification procedure. Therefore, if our objective is measuring the similarity between any two models M_j and M_k inside our ensemble, we can determine the similarity between $E^{(j)}(x_i)$ and $E^{(k)}(x_i)$. These two vectors will be highly similar if the models have extracted similar features and will be dissimilar otherwise.

The main benefit of this solution can be easily seen in the following scenario. Suppose that we are dealing with brain tumor classification, and one of the models only extracts information regarding the tumor's shape while not considering the texture at all, while the other model only extracts information regarding the tumor's texture but does not consider its shape. Then, we can consider them to work differently, as they operate on different sets of features, and hence, their predictions will change independently, resulting in less statistical dependence between the models (i.e., their outputs are not correlated) and an overall more robust ensemble that may resist outliers better.

The last important building block of our proposed solution is the similarity function that measures how similar the encoded latent vectors of the different member models are. For this, we can use any given $\mathcal{S}$ function that is suitable for measuring the similarity between any two arbitrary vectors in a high-dimensional latent vector space. Our previous work [25] focused on the cosine similarity measure, which is not only widely used to compare high-dimensional vectors in natural language processing [47–49], but very recent research has also shown that it can be used with great success for CNNs as well [50–52]. We will treat $E^{(j)}(x_i)$ and $E^{(k)}(x_i)$ as some arbitrary vectors in a latent vector space and use the cosine similarity function to measure the overall similarities between the encoded latent vectors produced by the member models. There are two smaller technical problems with the cosine similarity for our specific use case. The first one is that it cannot directly be built into the cost function J as a simple addition, as the output values can fall anywhere in the $[-1, 1]$ range. The second problem is that the cosine similarity of two vectors at an angle of $180°$ is -1, which is a particularly undesired property in our case, as $E^{(j)}(x_i)$ and $-E^{(j)}(x_i)$ carry the same information from the perspective of D despite the two vectors pointing to opposing directions. Therefore, for any two latent vectors u and v extracted by the member models of the ensemble, we define their similarity as

$$\mathcal{S}(u, v) = \left(\frac{u \cdot v}{\|u\| \|v\|} \right)^2, \tag{5}$$

where $\cdot$ is the inner product, and $\|.\|$ stands for the vector norm. Furthermore, the notation $dim(u)$ will be used for the dimension of a vector u. By using the modified similarity function (5), we can eliminate both of the previously mentioned problems: for all vectors u and v i) $\mathcal{S}(u, v) \in [0, 1]$, and ii) $\mathcal{S}(u, v) = 1$ if $u = -v$. This modified version of $\mathcal{S}(u, v)$ will be minimal if the two vectors are orthogonal. For our use case, this is only possible when the members use different sets of features, which inherently means they are dissimilar and make up a diverse ensemble. Accordingly, we can train the member models simultaneously by formulating the cost function for our training set as

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[\sum_{j=1}^{n} L_i(\theta_j) + \lambda \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} \mathcal{S}(E^{(j)}(x_i), E^{(k)}(x_i)) \right], \tag{6}$$

where $L_i(\theta_j)$ denotes the loss of M_j regarding (x_i, y_i), while the second term of the addition is a regularization term that penalizes members that have extracted similar features and hence have constructed similar latent vectors for any input x_i. This term calculates the similarity between M_j and the other members and adds the sum of the similarities to the loss function

multiplied by a $\lambda \in \mathbb{R}$. λ acts as a controlling parameter: the smaller λ is, the weaker the regularization effect becomes, while the larger its value is, the stronger the regularization effect becomes. Hence, λ adjusts how much emphasis we put on the diversity of the members during the optimization process. Lastly, given the ensemble $Ens_{\cos} = \{M_1, \ldots, M_n\}$ and the input image x_i, the final output of the ensemble is defined by (2). Fig. 3 shows an overview of our framework.

As we will show in Section 3, λ plays a crucial role in the overall performance of the ensemble. When we set $\lambda = 0$, the proposed method does not consider diversity at all. Using this option, there is no connection between the member models, resulting in simple majority voting behavior. Therefore, we will refer to majority voting as $\lambda = 0$. On the other hand, using $\lambda = 1$ means that diversity is just as important as the original optimization objective, which is to minimize the cross-entropy loss. As we will show later, in some cases, using this setting may also hinder the solution's overall performance, as diversity should usually be a secondary objective, with the primary objective of the optimization process being the minimization of the cross-entropy loss. The reason for this is that diversity, while playing an important role in the overall performance of the ensemble, can only lead to better results if the member models can solve the original optimization task. If the members cannot solve that, there is usually no merit in maximizing the diversity between them. Even though using this original framework will be shown to lead to good results, it still has some theoretical drawbacks not addressed in our previous work [25]. Section 2.6 explains these problems in detail and shows that there can be situations when the previously proposed framework may not work properly, while Section 2.7 contains our new and improved framework, which achieves even better results while also being theoretically more robust and overcoming the limitations and drawbacks of our previous framework.

2.6 The limitations of the cosine similarity function

Although in [25] we showed that the proposed framework performs quite well for our use case, there are still some theoretical limitations that, depending on the area where the framework is

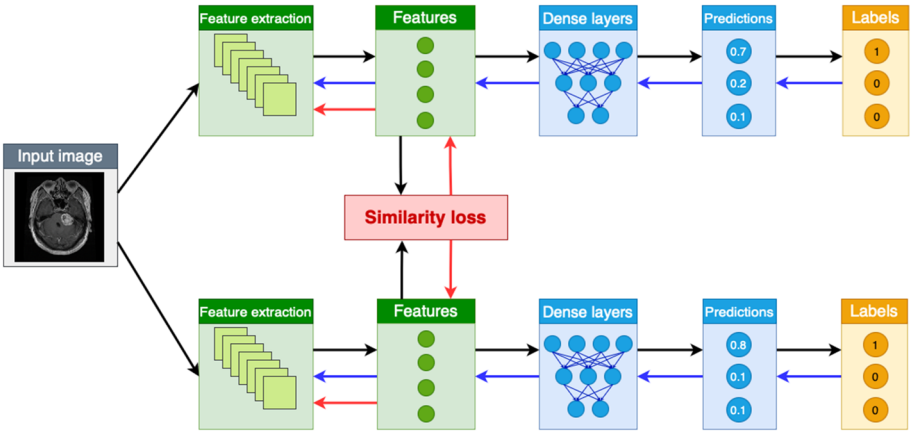

Fig. 3 An overview of our framework that calculates the similarity of the members to measure the diversity of the ensemble. The framework considers both the original optimization objective (blue arrows), as well as the similarity loss (red arrows) to update the weights of the members of the ensemble model

being applied, may hinder the training process or make the proposed solution work in a sub-optimal fashion. The following subsections show that such problems are not only theoretical but also arise in practice and should be addressed directly. We use the MNIST [29] and Medical MNIST [53] datasets as simple yet expressive examples and show that different models, even when using different architectures, may extract highly similar sets of features and hence work in a really similar way. We chose these specific datasets because they contain enough images to train neural networks and have been used extensively as benchmarks in the literature [54–58]. Moreover, the Medical MNIST dataset has also been used extensively as a benchmark in recent literature [57, 58], and contains medical images, more specifically computed tomography (CT) scans. Furthermore, it has a diverse set of classes (abdomen CT, breast MRI, chest CT, chest X-ray, hand CT, and head CT), with one of the classes (head CT) containing images slightly similar to the brain tumor dataset we focus on here.

2.6.1 Failing to recognize out-of-sequence similarities

For our first experiments, we used the MNIST [29] and Medical MNIST [30] datasets, as well as a very basic neural network architecture as simple examples to prove that the problem described in Section 2.6.2 does happen in practice. We used a very basic neural network architecture with only two convolutional and three linear layers (shown in Fig. 4) to avoid overfitting to the relatively simple datasets. In the case of the MNIST dataset, we used the original train and test splits proposed in [29] for training and evaluating the models. For the

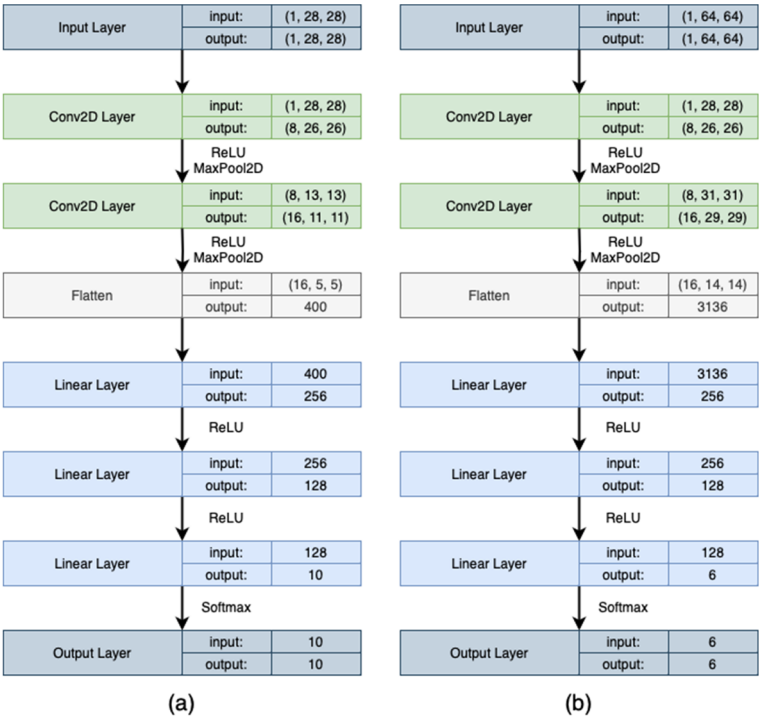

Fig. 4 An overview of the simple architecture used in our experiments for the (a) MNIST and (b) Medical MNIST dataset

Medical MNIST dataset, there are no such splits defined. Therefore, we split the original dataset into training, validation, and test splits in a 6:2:2 ratio for each class, respectively, resulting in 60% of the original data being used for training, 20% for validating, and 20% for testing purposes.

For the MNIST dataset, each model was trained for 10 epochs, while in the case of the Medical MNIST dataset, the models were trained for 5 epochs due to the lower number of classes. We used the standard learning rate of 0.001 for both datasets, a stochastic gradient descent (SGD) optimizer with a momentum of 0.9, and the cross-entropy loss. Each model converged to a (local) minimum by the end of the training, achieving at least 98% accuracy on the training and test sets for the MNIST and 99% in the case of the Medical MNIST dataset, respectively. For a more in-depth summary of the results of each model, see Table 3.

Our goal with this experiment was to see how likely it is for different models sharing the same architecture to rely on highly similar sets of features after being trained and observe how the cosine similarity handles these cases. For this, we trained five different versions of the previously mentioned basic architecture on the MNIST dataset. Each version of the model M_j ($j = 1, \ldots, 5$) was trained independently from any of the other models, using weights θ_j that were randomly initialized. After each model M_j was trained, we compared the outputs of their last convolutional layers $E^{(j)}$ on a set of test images x_i, which outputs correspond to $E^{(j)}(x_i)$ for our framework. Then, we first compared these extracted latent vectors $E^{(j)}(x_i)$ directly by examining their values and comparing their heatmap representations between the different models M_j. Our goal was to observe how random the activations (i.e., the higher values) and their positions inside the latent vectors u_j are in practice.

As seen in Fig. 5, the results may seem relatively random and not correlated at first glance. That is, the extracted latent vectors' heatmap representation seems diverse enough. However, when looking at the histogram representation of these same vectors, some slight similarities can be seen, which may indicate similarities between the extracted feature vectors. For example, the histograms of the fourth and fifth models and those of the first, second, and third ones seem similar. If we measure the means and standard deviations of these models (shown in Table 4), then we can see that indeed this is the case; models M_1, M_2, and M_3, as well as models M_4, and M_5 seem to produce feature vectors with highly similar means and standard

Table 3 A summary of the results of the different models trained on the MNIST dataset

Dataset	Model	Loss (train)	ACC (train)	Loss (test)	ACC (test)
MNIST	M_1	0.038	0.988	0.005	0.986
MNIST	M_2	0.037	0.988	0.003	0.989
MNIST	M_3	0.038	0.989	0.037	0.986
MNIST	M_4	0.039	0.988	0.129	0.987
MNIST	M_5	0.039	0.988	0.003	0.987
Medical MNIST	M_1	0.006	0.998	0.000	0.998
Medical MNIST	M_2	0.014	0.996	0.003	1.000
Medical MNIST	M_3	0.012	0.996	0.001	0.998
Medical MNIST	M_4	0.012	0.996	0.001	1.000
Medical MNIST	M_5	0.010	0.997	0.001	1.000

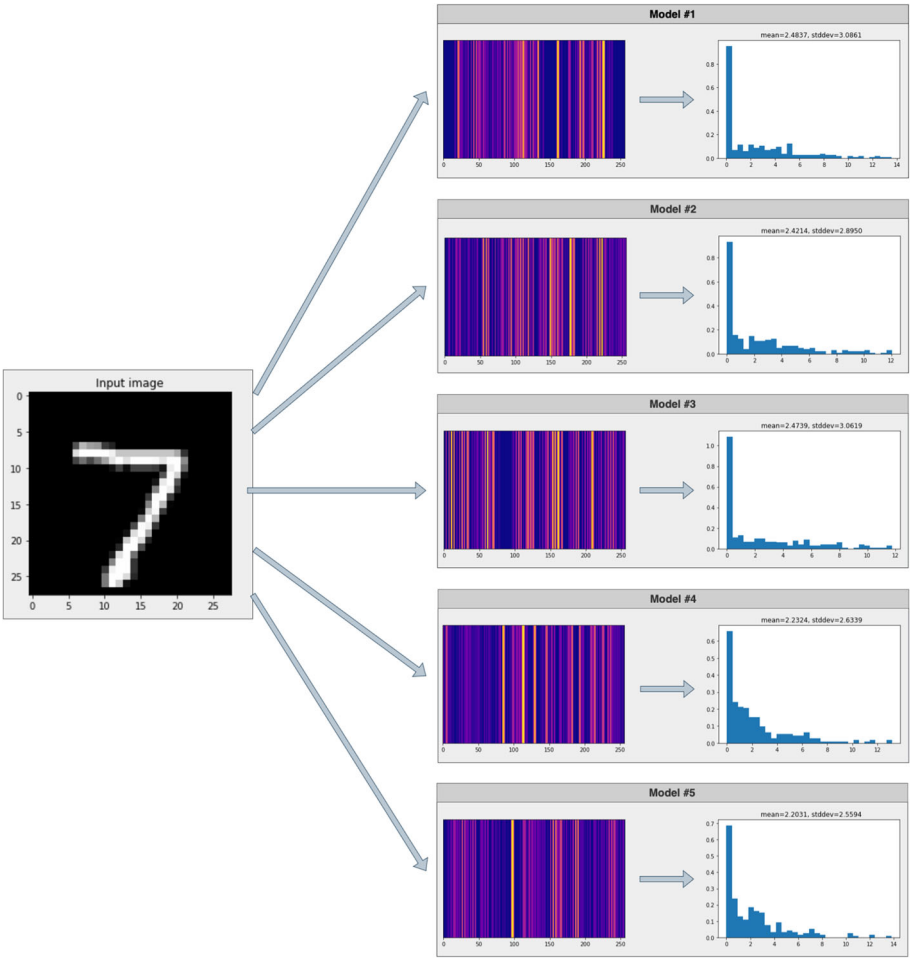

Fig. 5 A heatmap representation of the latent vectors (right) generated by each model for a given input image (left)

deviations. The results of this experiment, therefore, seem to support our previous claim that there may be cases when simply aggregating some already trained networks may result in a sub-optimal ensemble, as there may be some similarities between the member models, lowering the overall robustness and decreasing the abilities of the ensemble to generalize well.

2.6.2 Lack of tolerance against permutations

Even though the cosine similarity has been used with great results both for the area of natural language processing [47–49] and computer vision as well [50–52], it inherently has some theoretical drawbacks that may result in sub-optimal performance in some cases using our previous framework [25]. One of the biggest drawbacks of the cosine similarity is its inability to take spatial relationships into account. This is highly problematic for our use case, where

Table 4 The means and standard deviations of the latent vectors extracted by each model M_j for the test images shown in Figs. 5 and 7, respectively

Dataset	Model	Mean	Std. dev.
MNIST	M_1	2.4837	3.0861
MNIST	M_2	2.4214	2.8950
MNIST	M_3	2.4739	3.0619
MNIST	M_4	2.2324	2.6339
MNIST	M_5	2.2031	2.5594
Medical MNIST	M_1	0.7835	1.4593
Medical MNIST	M_2	0.7656	1.0289
Medical MNIST	M_3	0.9252	1.0917
Medical MNIST	M_4	0.7007	0.7907
Medical MNIST	M_5	0.7842	1.0442

we would like to notice any similarities between two latent vectors extracted by different models, where the order of the features is not guaranteed to be the same for every model. For example, let us take two hypothetical extracted feature vectors u and v. Let us define these two vectors as $u := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ and $v := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$. In this setup, the extracted feature vectors coincide; therefore, it is no wonder that the similarity metric defined in Section 2.5 will return

$$\mathcal{S}(u, v) = \left(\frac{u \cdot v}{\|u\| \|v\|} \right)^2 = 1, \tag{7}$$

meaning that the framework recognizes that the latent vectors are the same. However, if we randomly shuffle the elements of any of these vectors, the result will be vastly different. For example, for $u := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ and $v' := [9, 4, 8, 10, 1, 6, 7, 3, 5, 2]$, the result will be

$$\mathcal{S}(u, v') = \left(\frac{u \cdot v'}{\|u\| \|v'\|} \right)^2 = 0.4561, \tag{8}$$

which would signal a significantly lower relationship between u and v'. However, that is not the case, as the feature vectors u and v' encode the exact same features, just in different order. [59] has also drawn attention to the possibility of such permutations for CNNs.

For a more in-depth comparison, we ran many experiments examining how much this may affect the calculation of the similarities between the two vectors. During these experiments, we considered vectors with varying dimensions, ranging from 10 to 10,000, which, according to the current literature [17, 19, 22, 24], seem to cover the dimension of the possibly extracted feature vectors of all of the most commonly used CNNs. Then, we ran multiple tests for each setting, calculating the squared cosine similarity according to (5) between the original randomly generated latent vector u and its permutated version v'. To observe the variance of the similarities, we repeated these experiments several times, with different repetitions ranging from 10 to 1,000,000. Our motivation behind using different repetitions was that by increasing the sample size (in this case, the number of repetitions), we should get gradually more statistically accurate results. Therefore, by conducting these experiments, we should be able to observe the overall trend indicated by the orange line in Fig. 6. A summarized overview of the results of our measurements can also be seen in Fig. 6.

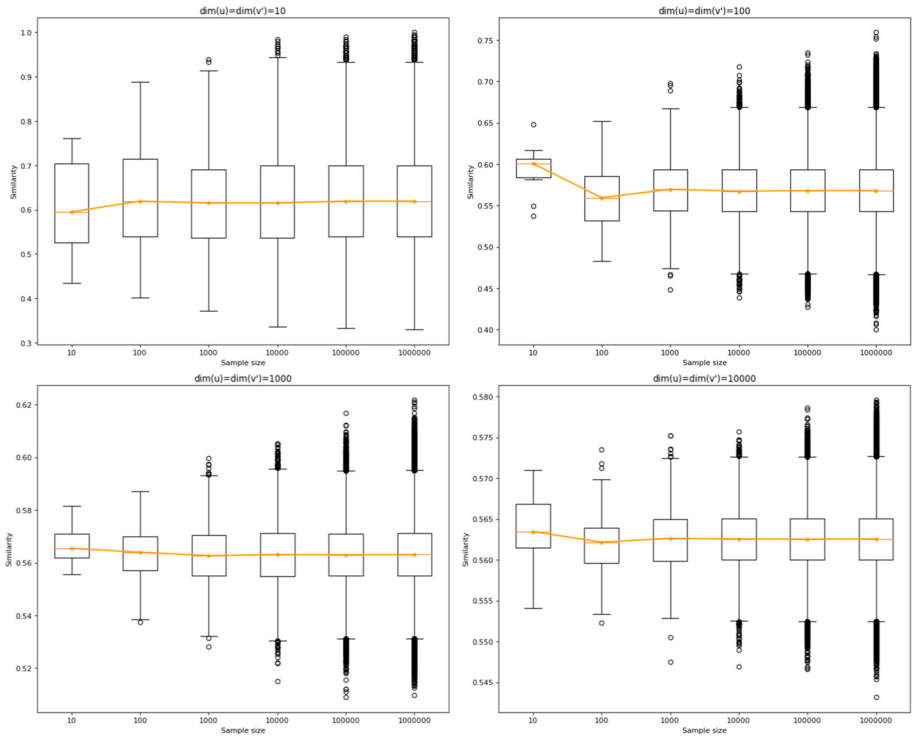

Fig. 6 The squared cosine similarities measured between a latent vector u and its randomly shuffled variant v', calculated for various sample sizes (horizontal axis) and varying vector dimensions

It can be observed that the average cosine similarity for any randomly permutated vector v' and the original vector u tends to be close to 0.6 for low-dimensional feature vectors ($dim(u) = dim(v') < 100$). However, as we increase the dimensions of these feature vectors, the effectiveness of the cosine similarity for recognizing permutated vectors decreases. Current state-of-the-art CNN architectures [17, 19, 22, 24] tend to use feature vectors with dimensions falling between 1024 and 4096. In this range (1,000 to 10,000, the bottom row of Fig. 6), we can see that usually only roughly 56% of the similarity is being recognized, although both vectors contain the same sets of features. This is highly problematic due to the random nature of the training process of neural networks and the randomized weight initialization procedure. In theory, this randomness can lead to two models that learn to extract the same sets of features in a slightly different order, i.e., the activations are the same, but their order is different. In this case, the framework proposed in [25] will fail to recognize these similarities and will, therefore, not perform optimally, as there could be a leakage in the similarities recognized by the solution. As we have shown, this can lead to simply failing to recognize roughly 50% of the similarities, ultimately leading to an ensemble less diverse than possible. In Section 2.6.1, we showed that this is, instead of being a purely theoretical problem, in fact, a real problem that can occur in practice. Now, we show that the problem of failing to recognize the similarity between a vector and its permutated version shown in the first part of this section may also occur in practice. We use again the simple MNIST and Medical MNIST datasets from Section 2.6.1 to show that when training multiple models of the same architecture we may end up with similar, slightly permutated feature vectors.

We used the already trained models $M_1, \ldots, M_5$ and extracted their corresponding feature vectors $u_1, \ldots, u_5$ for a test image (see Fig. 7).

As the next step, we measured the cosine similarity between the extracted u_j vectors. Even though, in this case, the similarity could be easily seen with the naked eye in Fig. 7, the extracted cosine similarity values shown in Fig. 8 are really low and fail to capture this. This is highly problematic, as even though we can observe a clear and indistinguishable correlation between the histograms of the u_j vectors, these similarities will be lost when we optimize our cost function J as introduced in Section 2.5.

To solve this issue, we can use a similarity metric that is not sensitive to the order of the elements when comparing the two feature vectors. Earlier, we showed that one intuitive way of measuring similarity is by comparing the distributions of the two latent vectors. One theoretically founded way of measuring this is to use a histogram representation of the similarities and then compare these higher-level representations to penalize vectors that are similar but come from different sources. There are a variety of possible solutions for measuring this similarity, ranging from calculating the intersection between the histograms, calculating their correlation, or using a chi-square test. The main idea, however, is that a larger overlap between two histograms indicates a higher degree of similarity, while a small overlap means a lower degree of similarity. By measuring using histograms instead of the similarities between the latent feature vectors directly, we lose all information regarding the order of the elements that could negatively affect the similarity calculation, as the improved measure only considers the aggregated, higher-level representation. In Fig. 8, we can see that despite having no information about the order of the elements, we can get better and more realistic similarity values for the correlation metric when we only consider the histogram representation of the latent feature vectors. We can also see that using the histogram similarities, we could still retain the most important and striking similarities (e.g., between models M_3 and M_4, or M_2 and M_5) while recognizing that models M_1 and M_4 are not similar (which were deemed as similar by the cosine similarity metric). It is therefore easy to see that by incorporating this approach into our ensemble framework we could further increase its accuracy, as features that would be lost using the cosine similarity function could be retained with the histogram similarity. This would effectively give the ensemble the opportunity to recognize more complex patterns when measuring the similarity between the member models. In the next section, we introduce our improved framework, which builds upon the idea of using the histograms of the feature vectors that can capture these relations more reliably and with much higher accuracy than our previous framework.

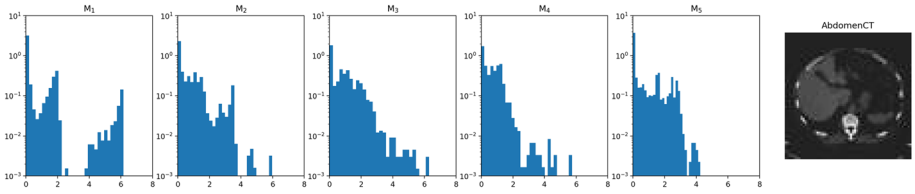

Fig. 7 A sample input image from the Medical MNIST dataset (right) and an overview of the distribution of the extracted features by different models (left). The horizontal axis represents the values inside the feature vectors, and the vertical one shows their corresponding frequencies inside the vector on a logarithmic scale

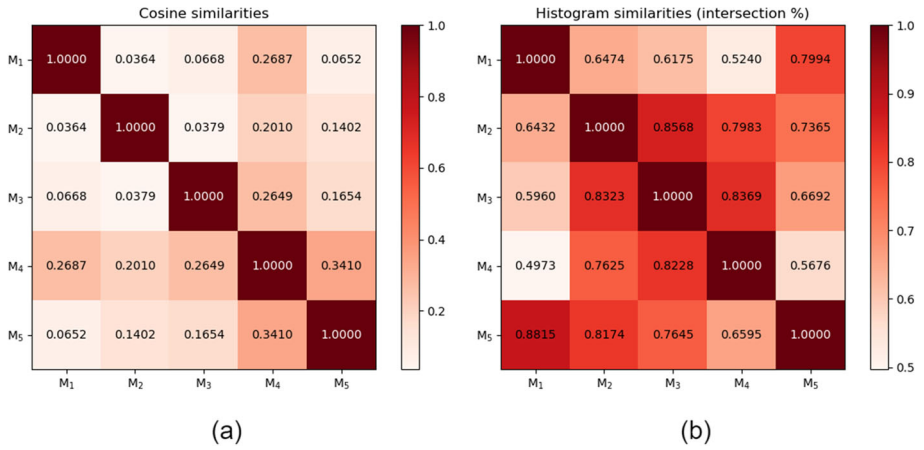

Fig. 8 The correlations measured between the (a) cosine, and (b) histogram similarities calculated between the feature vectors extracted by the five models trained on the Medical MNIST dataset for the sample image shown in Fig. 7. The values shown in (b) have been normalized by using the maximum value in each row of the heatmap

2.7 Histogram-based similarity

In Section 2.6, we have shown that the cosine similarity measure fails to capture some intricate similarities in the following cases: a) when the same feature vector is permutated, and b) when even though the order of the feature vectors is not the same, their distributions are similar. To solve these shortcomings, we propose a new, improved version of our framework that uses the histogram loss [26] instead of the original cosine similarity when calculating the $\mathcal{S}(u, v)$ similarity between any two latent vectors u and v. We will use a modified version of the histogram loss [26] which is differentiable and can be used with neural networks as the base of our solution to measure the similarities between the histogram representations of the extracted latent vectors of the member models in the ensemble. This way, the similarity metric does not rely on the order of the elements of the feature vectors, resulting in an overall more robust and theoretically founded solution.

For our new framework, instead of measuring the similarities between any two pairs of latent vectors directly, we reformulate our previous problem as measuring the probability $p_{reverse}$ [26] that any two random latent vectors u_j and u_k extracted by two different models M_j and M_k (negative pair) are more similar to each other than latent vectors extracted by the same models (positive pair). We will use this probability as a form of regularization when training the models, penalizing those that produce latent vectors more similar to other models than those produced by the same model. This should, in theory, result in a more efficient and "tighter" distribution of the latent vectors generated by each model inside the ensemble since we penalize any overlap between said distributions. Due to the fewer overlaps, we should also be able to reduce the variance of the distributions, as the models are forced to generate latent vectors with more compact and concentrated distributions (see Fig. 9). This is because, for each similarity measuring step, a larger batch of latent vectors u_j originating from all of the ensemble models and their similarities are considered by calculating the inner products.

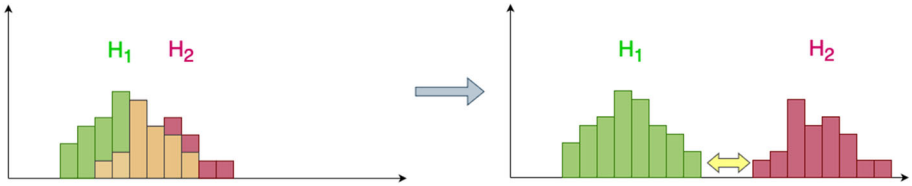

Fig. 9 By using the histogram representation, we can directly penalize any overlap (orange) between two histograms H_1 (green) and H_2 (red)

During the optimization process, each model M_j is forced to generate latent vectors with a higher probability of being sampled from the distribution of the similarities belonging to the same model M_j than any other model M_k. The benefit of this approach is that the overall distribution of the similarities is taken into account during training, which holds more information than simply relying on the individual similarities of some arbitrarily sampled latent vector pairs.

To calculate this probability, which the authors of [26] refer to as "the probability of reverse" $p_{reverse}$, we need to approximate the two probability distributions p^+ and p^-, which refer to the estimate of the probability distributions of the similarities of latent vectors produced by the same model M_j and by a distinct model M_k, respectively. For estimating p^+ and p^-, following the conventions of [26], we construct sample sets denoted by $\mathcal{S}^+$ and $\mathcal{S}^-$ that contain the similarity of the sample latent vector pairs, calculated using the inner product, produced by the same models ($\mathcal{S}^+$) and by different models ($\mathcal{S}^-$). Namely, we formulate $\mathcal{S}^+$ and $\mathcal{S}^-$ as

$$\begin{aligned} \mathcal{S}^+ &= \left\{ s_{k,k} = u \cdot v \ : \ u = E^{*(k)}(x_i), v = E^{*(k)}(x_j), i \neq j \right\}, \\ \mathcal{S}^- &= \left\{ s_{k,l} = u \cdot v \ : \ u = E^{*(k)}(x_i), v = E^{*(l)}(x_j), k \neq l \right\}. \end{aligned} \tag{9}$$

At this point, it can be observed that this method is a natural and logical extension of our previous framework [25], as it also builds on the core idea of using the inner product to measure the similarities of the sample latent vector pairs. However, instead of using these similarities directly, it generates a histogram representation using multiple observed similarities to approximate their distributions.

For our proposed framework, we feed the same batch of size b of inputs $\{x_1, x_2, \ldots, x_b\}$ into the different models M_j of the ensemble to extract the latent vectors $E^{(j)}(x_i)$. For the proposed framework to work, the member models' outputs must also be normalized due to the limitations discussed in [26]. For this, we follow the recommendations of [26] and apply L2 normalization to each $E^j(x_i)$ latent vector, calculating $E^{*(j)}(x_i) = ||E^{(j)}(x_i)||$. Then, we construct the sample sets $\mathcal{S}^+$ and $\mathcal{S}^-$ by defining each sample $s_{j,k}$ using the normalized latent vectors $E^{*(j)}(x_i)$ extracted by each member M_j. We assign the sample $s_{j,k}$ to sample set $\mathcal{S}^+$ if $k = j$, meaning that both of the feature vectors originated from the same model, otherwise we assign the sample to $\mathcal{S}^-$.

For calculating the similarity between any two models inside the ensemble, we approximate the p^+ and p^- distributions with histograms H^+ and H^-, each with R bins, respectively. Following the workflow described in [26], we define both H^+ and H^- with uniformly spaced bins, with their nodes $-1 = t_1, \ldots, t_R = 1$ filling the $[-1, 1]$ interval – as the $E^{*(j)}(x_i)$ vectors are L2 normalized – and with the step size $\Delta = \frac{2}{R-1}$, according to the following,

slightly modified formulae

$$H^+ = \left[h_r^+ = \frac{1}{|\mathcal{S}^+|} \sum_{j=1}^{n} \delta_{j,j,r} \ : \ r = 1, \ldots, R \right],$$
$$H^- = \left[h_r^- = \frac{1}{|\mathcal{S}^-|} \sum_{j=1}^{n} \sum_{\substack{k=1 \\ k \neq j}}^{n} \delta_{j,k,r} \ : \ r = 1, \ldots, R \right]. \tag{10}$$

In (10), linear interpolation is used to decide which node of the histogram the given similarity belongs to by using the weights $\delta_{j,k,r}$, which are defined as

$$\delta_{j,k,r} = \begin{cases} (s_{j,k} - t_{r-1})/\Delta, & \text{if} s_{j,k} \in [t_{r-1}; t_r], \\ (t_{r+1} - s_{j,k})/\Delta, & \text{if} s_{j,k} \in [t_r; t_{r+1}], \\ 0, & \text{otherwise}, \end{cases} \tag{11}$$

for $j, k = 1, \ldots, n, r = 1, \ldots, R$.

As seen in (10), when calculating H^+, we only consider the similarities between the same model, while when calculating H^-, we only consider the similarities between distinct models. The histograms H^+ and H^- approximate the real distributions p^+ and p^- of the similarities between the latent vectors extracted by each model. Using them, we can directly look for any overlap (see Fig. 9) between the similarities of the latent vectors extracted by each model M_j inside the ensemble. For measuring this overlap, we use the formula introduced in [26], which is defined as

$$\mathcal{S}(H^+, H^-) = \sum_{r=1}^{R} \left(h_r^- \sum_{q=1}^{r} h_q^+ \right). \tag{12}$$

Since both the forward pass and the histogram loss depend on the normalized $E^{*(j)}$ vectors as well as the histograms H^+ and H^-, the cost function also needs to be slightly modified. For our improved framework, we define the cost function $J^*(\theta)$ as

$$J^*(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[\sum_{j=1}^{n} L_i(\theta_j) + \lambda \mathcal{S}(H^+, H^-) \right]. \tag{13}$$

Lastly, as in the case of the Ens_{cos} ensemble, for any given input image x_i in the dataset, the output of the improved framework $Ens_{hist} = \{M_1, \ldots, M_n\}$ is defined by (2).

2.8 Comparing the proposed frameworks with traditional ensemble methods

This section shows how our proposed frameworks relate to traditional ensemble methods. Moreover, we also present how they distinguish themselves from these traditional solutions. When using $\lambda = 0$, given $E^{(1)}, \ldots, E^{(n)}$ for the models $M_1, \ldots, M_n$, respectively and the

loss function L_i, the cost function $J(\theta)$ for $Ens_{\cos}$ becomes

$$\begin{aligned} J(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\left[\sum_{j=1}^{n} L_i(\theta_j) + \lambda \sum_{j=1}^{n-1}\sum_{k=j+1}^{n} \mathcal{S}(E^{(j)}(x_i), E^{(k)}(x_i))\right] \\ &= \frac{1}{m}\sum_{i=1}^{m}\left[\sum_{j=1}^{n} L_i(\theta_j) + 0 \sum_{j=1}^{n-1}\sum_{k=j+1}^{n} \mathcal{S}(E^{(j)}(x_i), E^{(k)}(x_i))\right] \\ &= \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n} L_i(\theta_j). \end{aligned} \tag{14}$$

Similarly, for the ensemble Ens_{hist}, the cost function $J^*(\theta)$ can also be simplified to

$$\begin{aligned} J^*(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\left[\sum_{j=1}^{n} L_i(\theta_j) + \lambda \mathcal{S}(H^+, H^-)\right] \\ J^*(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\left[\sum_{j=1}^{n} L_i(\theta_j) + 0 \mathcal{S}(H^+, H^-)\right] \\ J^*(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n} L_i(\theta_j). \end{aligned} \tag{15}$$

We can observe that both (14) and (15) take on the form of a standard cost function when using $\lambda = 0$. This in practice means that there is no interaction between the member models during the optimization process, which results in models trained in complete isolation. Furthermore, there is no metric defined that would measure the diversity of the models. After combining these models into an ensemble $Ens_{\cos} = \{M_1, \ldots, M_n\}$ or $Ens_{hist} = \{M_1, \ldots, M_n\}$, the output of the ensembles trained with $\lambda = 0$ will be exactly the same as that of an ensemble using majority voting. Therefore, we have shown that the frameworks presented in Sections 2.5 and 2.7 default to simple majority voting behavior when using $\lambda = 0$, resulting in an ensemble that does not measure diversity at all. Consequently, we will refer to majority voting as $\lambda = 0$ in the subsequent parts.

2.9 Using a weighted cost function

In this section, we expand both our original and improved frameworks to handle imbalanced datasets where there is a significant disparity between the total occurrences of the labels. In such cases, some labels appear disproportionally more often in the dataset than others. This is especially common in the medical field, where there are usually orders of magnitude more healthy medical records than records containing any lesion, further increasing the difficulty of training solutions that can recognize chronic lesions or diseases. A common technique for training efficient solutions for such imbalanced datasets is using a weighted loss function that weights the loss calculated for the given class in inverse proportion to its frequency in the dataset, ultimately making the training process smoother and providing models that can recognize rare classes.

When using a weighted cost function, the weights β_i are calculated for all inputs x_i in the dataset to increase the influence of the label y_i on the overall cost during the training procedure. To define a weighted configuration for our framework, it is imperative to note that

our cost functions (6) and (13) are constructed of two parts: the first part optimizes the original loss L_i and the second part optimizes the similarity S. This is extremely important because, using a weighted cost function, the objective is to weight each L_i depending on the rarity of the class y_i. However, the second part of the cost function measures the similarity between the ensemble members, which is independent of the labels y_i. Instead, this second part only depends on the latent vectors $E^{(j)}(x_i)$ derived by each model M_j for the corresponding input x_i. Therefore, it is only necessary to change the first parts of formulae (6) and (13). Then, a weighted cost function $J(\theta)$ can be defined as

$$J_{weighted}(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[\sum_{j=1}^{n}\beta_i L_i(\theta_j) + \lambda\sum_{j=1}^{n-1}\sum_{k=j+1}^{n}\mathcal{S}(E^{(j)}(x_i), E^{(k)}(x_i))\right], \text{ and}$$

$$J^{*}_{weighted}(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[\sum_{j=1}^{n}\beta_i L_i(\theta_j) + \lambda\mathcal{S}(H^{+}, H^{-})\right] \tag{16}$$

for the original framework using the cosine similarity and our newly proposed framework, which uses the histogram loss.

Although no change would directly affect the calculation of the similarities, the overall behavior and the outputs of the new weighted cost functions shown in (16) changed due to the introduced weights β_i. Therefore, in this paper, we also examine the effect of using a weighted cost function for our frameworks using the dataset published by Jun Cheng [7]. In Section 2.2 and Table 2, we have shown a slight imbalance between the different classes. Namely, there were almost as many images belonging to the glioma class as in the other two classes (meningioma and pituitary tumor), combined. Therefore, this dataset presents a good opportunity to study the effects of using a weighted loss function following our proposed frameworks.

2.10 Using different architectures

This section extends our original and improved frameworks to function with different architectures. So far, we have only discussed re-using the exact same architecture, which was a limitation. It stemmed from formulae (6) and (13), which both relied on calculating the similarity $\mathcal{S}(E^{(j)}(x_i), E^{(k)}(x_i))$ of the two feature vectors produced by the models M_j and M_k. During the similarity calculation (see (5) and (9) for the framework using the cosine similarity and the histogram loss, respectively), both methods utilized the inner product to calculate the similarity. However, the inner product can only be used for two vectors u and v of the same dimension $dim(u) = dim(v) = l$, which limits the usage of our framework to neural networks and members M_j and M_k that have the exact same feature vector dimension $dim(E^{(j)}(x_i)) = dim(E^{(k)}(x_i)) = l$.

To solve this problem, it is required to bring the feature vectors to a common dimension before measuring the similarity between them. This is not a trivial problem to solve, as reducing the dimension to the value of the smallest feature vector may result in losing important features in longer feature vectors. It is also unclear which components to drop from longer feature vectors, as some of them encode special features and are activated only when these particular features are present in the input image. In this case, dropping such a component, which usually contains 0 values, may decrease the performance. Another approach can be to expand all feature vectors to be of the same dimension as the one having the maximum. During our experiments, we found that this does not lead to either worse results or insta-

bilities. Therefore, in this paper, we propose a simple way to address this issue by applying zero padding to the feature vectors $E^{(k)}(x_i)$ to bring them to the same dimension as the ones produced by the model M_j that has the largest dimension. An overview of this method can be seen in Fig. 10.

3 Results and discussion

In this section, we provide a comprehensive, in-depth overview of our training, validation, and testing methodologies, explain how the data was split into training, validation, and test sets, and detail the various settings, weighting procedures (if any), and hyperparameters that were used during our experiments. We also introduce the metrics that were used during the evaluation process and compare the different methodologies introduced in the previous

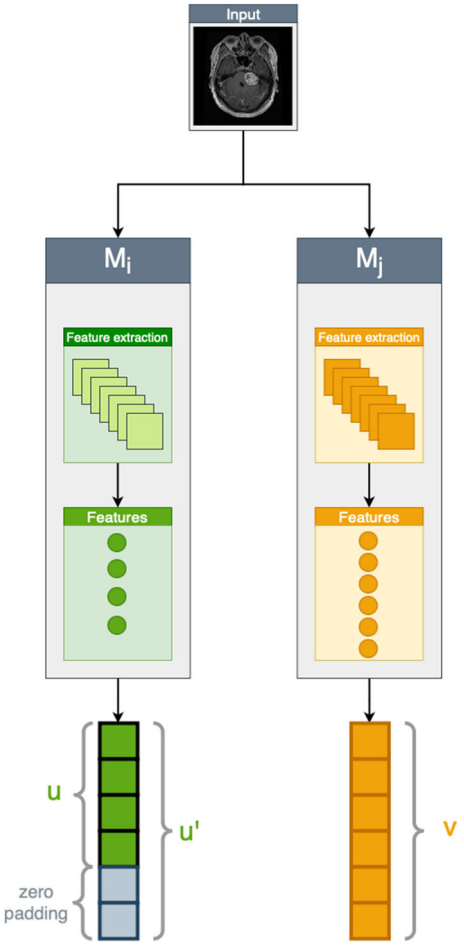

Fig. 10 We can bring latent vectors u and v of different dimensions to a common dimension by applying zero padding on the vector with fewer elements. The resulting u' vector will have the same dimension as v, with the last elements being zero

section, both with our original framework and other state-of-the-art approaches. At the end of this section, we also draw some experimental conclusions.

3.1 Hyperparameter tuning

Before training the models, we tuned the hyperparameters of each architecture separately. During this process, except for the total number of epochs, we only used the training set to configure the hyperparameters to avoid any possible bias towards the validation sets. We searched for the optimal value of the batch size, learning rate, and number of epochs and looked for the best optimizer. When searching for the optimal batch size, we considered multiple factors, such as any substantial oscillations in the loss – from batch to batch and from epoch to epoch – that would make the learning process unstable. For the learning rate $\alpha \in (0, 1)$, we used a schedule for a total of 100 epochs on the training set, evaluated the effectiveness and performance of a wide range of learning rates, and picked the one that i) had the lowest loss value and ii) had a sufficiently large environment $[\alpha - \epsilon, \alpha + \epsilon]$ with $\epsilon \in (0, 1)$, where the loss values neither excessively oscillated nor increased substantially.

After the hyperparameter tuning phase, we found that batch size 32 gave the best results for all models, leading to minimal oscillations and smooth decreases in the loss values. For the optimizer, we found Adam [60] to be the most efficient, while for the learning rate, the value of 0.0001 achieved the best overall performance for each architecture: it led to continuous and smooth decreases in the loss value while also being relatively fast, compared to lower learning rates. Furthermore, it did not seem to get stuck in bad-performing local optima during our experiments. For our newly proposed framework that used the histogram loss, we also considered multiple settings when determining the optimal number of bins. After evaluating multiple different options and considering the change of training loss, as well as validation loss during training, we determined that $R = 129$ was the most suitable option for the number of bins in our use case, resulting in $\Delta = 0.015625 \approx 0.02$. This was also in line with the findings of the original paper [26], which also showed $\Delta = 0.02$ as a suitable choice, balancing the level of detail (i.e., number of bins) and performance.

3.2 Cross-validation and training

To measure the overall performance of our frameworks, we trained and then evaluated each model using cross-validation. This is of utmost importance when evaluating deep learning-based solutions, which have inherently random internal workings capable of affecting the test results. By having a higher number of cross-validation folds, it is possible to reduce such noise and get a clearer view of the performance of the given model. As previously mentioned, we used the original folds and splits provided by the authors of the dataset [7] to train and test our models. The official splits divide the dataset of 3,064 MRI images into training and test parts and contain five folds. One drawback of the official splits is that they only contain the training and test parts. To measure and detect the degree of over- and underfitting in our models during training and to monitor the behavior of the models, i.e., to stop the training procedure if the validation loss keeps increasing while the training loss decreases in case of overfitting, we defined a custom validation split using the training data from the five original folds. During this procedure, we split the original training part of the five folds into training and validation parts in a 8:2 ratio, respectively. After this, each model was trained using the training parts of the five newly generated folds, validated using the validation part of the current fold, and then tested using the official test part of the given fold defined in [7]. The

number of epochs was determined using the validation set; each model was trained until the observed validation loss started increasing rapidly. In the training process, the weights with the lowest validation loss were saved, which were used in the testing phase to evaluate the performance of each model.

During the training process, we also applied transformations and data augmentation to the inputs to increase the generalization capabilities of the networks. Each input image was resized to fit the expected input shape of the neural networks (224 × 224 pixels) and was also normalized. For data augmentation, we have considered several potential techniques. After a shorter prototyping phase, we have selected the ones that were relevant and applicable to our problem. These were the following: random cropping, horizontal and vertical flips. When we applied random cropping, we first resized the images to 110% of the target size (224 × 224 pixels) before cropping to the target size. We found that the value of 110% was a good balance between generating sufficiently varied augmented images while still not cropping too much out of the image, which could potentially lead to losing valuable information.

3.3 Evaluation

As discussed in Section 3.2, all of the algorithms were evaluated on the five official test splits of the dataset [7]. During this evaluation procedure, we considered several of the most important metrics to give a comprehensive and in-depth overview of the performance of the several different methodologies and architectures. Namely, we calculated the overall accuracy (ACC), as well as the per class accuracy (ACC_{class_name}), sensitivity (SE), precision ($PREC$), and F_1 score for each architecture. To measure these values, we calculated the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for each class. For our use case, an increase in TP and TN meant that the ground truth labels matched the outputs of the model, both being positive in the case of TP and both being negative in the case of TN. We considered cases FP when the network output signaled the presence of the given class while the ground truth label was zero and FN in the opposite case, meaning that the model failed to recognize the presence of a given class. The exact formulae of the used metrics are as follows

$$\begin{aligned} ACC &= \frac{TP + TN}{TP + TN + FP + FN}, \\ SE &= \frac{TP}{TP + FN}, \\ PREC &= \frac{TP}{TP + FP}, \\ F_1 &= \frac{2\,TP}{2\,TP + FP + FN}. \end{aligned} \tag{17}$$

Additionally, our dataset contained multiple classes, three in total. Since the metrics introduced in (17) are calculated at class level, it was also important to decide how to calculate a unified metric that merged the results for the different classes into one metric. For this, we calculated the macro- and weighted average of each metric.

3.4 Project setup

The experiments were conducted in the cloud using the Azure Machine Learning service. The training took place on a virtual machine (VM) with 6 cores, a total of 112GB RAM, and

a Tesla V100 GPU. All the related code was written in the Python programming language and the PyTorch framework.

3.5 Experimental results

In this section, we give an exhaustive and in-depth overview of the performance of the various algorithms and report our experimental results. As noted in Section 3.3, we calculated various metrics using the pre-defined folds and splits to evaluate the overall performance of each model. All the models were trained on each of the five training folds (which we split into training and validation parts, as discussed in Section 3.2) and were then evaluated using the original test sets defined by the authors of [7]. We have also compared our results with several other state-of-the-art approaches and standard techniques, such as majority voting, which we used as baselines to evaluate and compare with our results. With the $\lambda = 0$ setup, both our original and improved frameworks apply the standard majority voting, as there is no connection between any of the models during training, which leads to each model being trained separately. Therefore, our tables refer to the standard majority voting approach as $\lambda = 0$. We also considered several other works that used the same dataset, both traditional [38] and deep learning-based ones [35–37, 45], and used them as simple baselines and compared their reported results with the performance of our frameworks. For [45], we considered two different methods: one that uses the GLCM and VGG-16 features as described in [45], and an additional one, which follows the same procedure as described in the original paper, but only uses the GLCM matrix as inputs. We will refer to the first as GLCM+VGG16 and to the second simply as GLCM in all subsequent tables. The type of architecture used (SVM or KNN) will be specified under the Type column of each table. Lastly, for each baseline, cells containing a "-" symbol in one of the columns belonging to a metric contain values that were not reported in the corresponding original study, while for columns corresponding to a given setting, the same "-" symbol indicates a setting that does not apply to the given architecture.

As noted in Section 3.3, we calculated the macro- and weighted average for each metric introduced earlier and provided dedicated tables for each setting. Macro average calculates the class-level metrics and then averages them without taking the imbalance into account, while for the weighted average, the weights reflect how often a given class occurs in the test set. The results shown are calculated at 95% confidence levels and sorted by the type of model and training methodology in the following order: state-of-the-art baseline approaches and weighted averaging ($WeightedAvg.$), followed by our previous and new frameworks, evaluated using different values of λ. We also included the results of our newly proposed hybrid ensemble architecture presented in Section 2.10, where we selected the best-performing base networks MobileNetv2, EfficientNet, and ShuffleNetv2. We constructed ensembles using different values of λ and both our original cosine similarity-based framework, as well as our newly proposed histogram loss-based framework. Using the aforementioned three models, for the sake of brevity, we refer to these approaches as Ens in our tables. First, we measured how our newly presented methods compare with our original framework, such as using different architectures (abbreviated as Ens in the tables) and the newly proposed histogram-based framework. In Table 5, we compare our newly proposed approaches' accuracy – global and class-level – with our previous framework based on the cosine similarity measure and several state-of-the-art baselines. ACC indicates the overall global accuracy, while the columns ACC_G, ACC_M, and ACC_{PT} correspond to the class-level accuracies

Table 5 A summary of the measured global and class-level accuracies

Algorithm	Type	λ	ACC	ACC_M	ACC_G	ACC_{PT}
Afshar et al. [36]	–	–	0.909	–	–	–
Abiwinanda et al. [35]	–	–	0.842	–	–	–
Fasihi et al. [38]	–	–	0.841	–	–	–
MIXCAPS-BoxCaps [37]	–	–	0.913	–	–	–
GLCM [45]	SVM	–	0.764 ± 0.032	0.831 ± 0.036	0.801 ± 0.018	0.897 ± 0.019
GLCM [45]	KNN	–	0.654 ± 0.013	0.750 ± 0.018	0.701 ± 0.014	0.857 ± 0.019
GLCM+VGG16 [45]	SVM	–	0.918 ± 0.013	0.924 ± 0.011	$\mathbf{0.947 \pm 0.005}$	$\mathbf{0.964 \pm 0.013}$
GLCM+VGG16 [45]	KNN	–	0.919 ± 0.012	0.932 ± 0.010	0.943 ± 0.014	0.962 ± 0.008
Weighted Avg.	–	–	0.912 ± 0.017	0.929 ± 0.012	0.94 ± 0.017	0.956 ± 0.017
AlexNet	base	–	0.868 ± 0.042	0.894 ± 0.037	0.91 ± 0.013	0.932 ± 0.040
MobileNetv2	base	–	0.893 ± 0.032	0.919 ± 0.016	0.922 ± 0.026	0.945 ± 0.027
ShuffleNetv2	base	–	0.865 ± 0.029	0.887 ± 0.025	0.905 ± 0.027	0.939 ± 0.020
EfficientNet	base	–	0.898 ± 0.018	0.917 ± 0.014	0.933 ± 0.016	0.945 ± 0.013
AlexNet	cosine	0.0	0.875 ± 0.036	0.902 ± 0.027	0.909 ± 0.017	0.939 ± 0.032
MobileNetv2	cosine	0.0	0.888 ± 0.026	0.911 ± 0.018	0.920 ± 0.018	0.945 ± 0.027
ShuffleNetv2	cosine	0.0	0.877 ± 0.038	0.897 ± 0.028	0.913 ± 0.027	0.945 ± 0.025
EfficientNet	cosine	0.0	0.896 ± 0.022	0.916 ± 0.018	0.929 ± 0.012	0.948 ± 0.023
Ens	cosine	0.0	0.903 ± 0.019	0.916 ± 0.012	0.939 ± 0.015	0.952 ± 0.019
AlexNet	histogram	0.0	0.875 ± 0.036	0.902 ± 0.027	0.909 ± 0.017	0.939 ± 0.032
MobileNetv2	histogram	0.0	0.888 ± 0.026	0.911 ± 0.018	0.920 ± 0.018	0.945 ± 0.027
ShuffleNetv2	histogram	0.0	0.877 ± 0.038	0.897 ± 0.028	0.913 ± 0.027	0.945 ± 0.025
EfficientNet	histogram	0.0	0.896 ± 0.022	0.916 ± 0.018	0.929 ± 0.012	0.948 ± 0.023
Ens	histogram	0.0	0.903 ± 0.019	0.916 ± 0.012	0.939 ± 0.015	0.952 ± 0.019
AlexNet	cosine	0.5	0.882 ± 0.031	0.903 ± 0.026	0.922 ± 0.013	0.938 ± 0.031
MobileNetv2	cosine	0.5	0.915 ± 0.018	0.931 ± 0.014	0.945 ± 0.012	0.954 ± 0.019
ShuffleNetv2	cosine	0.5	0.892 ± 0.030	0.907 ± 0.025	0.928 ± 0.018	0.949 ± 0.024

Table 5 continued

Algorithm	Type	λ	ACC	ACC_M	ACC_G	ACC_{PT}
EfficientNet	cosine	0.5	0.904 ± 0.023	0.920 ± 0.014	0.940 ± 0.013	0.949 ± 0.026
Ens	cosine	0.5	0.908 ± 0.019	0.922 ± 0.015	0.942 ± 0.01	0.952 ± 0.019
AlexNet	histogram	0.5	0.887 ± 0.034	0.909 ± 0.030	0.919 ± 0.013	0.946 ± 0.031
MobileNetv2	histogram	0.5	0.908 ± 0.013	0.929 ± 0.009	0.935 ± 0.009	0.953 ± 0.017
ShuffleNetv2	histogram	0.5	0.895 ± 0.021	0.912 ± 0.014	0.929 ± 0.014	0.949 ± 0.019
EfficientNet	histogram	0.5	0.909 ± 0.015	0.923 ± 0.013	0.942 ± 0.009	0.954 ± 0.020
Ens	histogram	0.5	0.901 ± 0.017	0.917 ± 0.012	0.937 ± 0.007	0.948 ± 0.022
AlexNet	cosine	1.0	0.881 ± 0.045	0.909 ± 0.029	0.917 ± 0.022	0.937 ± 0.041
MobileNetv2	cosine	1.0	0.904 ± 0.027	0.922 ± 0.020	0.936 ± 0.017	0.951 ± 0.023
ShuffleNetv2	cosine	1.0	0.893 ± 0.028	0.901 ± 0.019	0.918 ± 0.026	0.944 ± 0.025
EfficientNet	cosine	1.0	0.905 ± 0.021	0.919 ± 0.014	0.937 ± 0.010	0.953 ± 0.030
Ens	cosine	1.0	0.901 ± 0.032	0.917 ± 0.029	0.939 ± 0.015	0.945 ± 0.027
AlexNet	histogram	1.0	0.877 ± 0.042	0.902 ± 0.027	0.913 ± 0.022	0.939 ± 0.040
MobileNetv2	histogram	1.0	$\mathbf{0.921 \pm 0.005}$	$\mathbf{0.935 \pm 0.005}$	0.945 ± 0.011	0.963 ± 0.017
ShuffleNetv2	histogram	1.0	0.902 ± 0.019	0.923 ± 0.014	0.928 ± 0.017	0.952 ± 0.022
EfficientNet	histogram	1.0	0.904 ± 0.024	0.921 ± 0.022	0.939 ± 0.004	0.947 ± 0.024
Ens	histogram	1.0	0.906 ± 0.018	0.916 ± 0.018	0.943 ± 0.005	0.952 ± 0.020

Bold entries signify the highest values

belonging to the glioma, meningioma, and pituitary tumor classes, respectively. It can be seen that our proposed frameworks surpass each baseline algorithm. In the case of [45], the approach that relied on both the GLCM and VGG-16 features and then trained an SVM achieved slightly better accuracies for the glioma and pituitary tumor classes. However, these improvements were only 0.2% and 0.1%, respectively, which can be considered to be within margin of error, while on the other hand, our framework achieved more than 1% more accuracy for the meningioma class, which is a substantial and significant difference. Our MobileNetv2-based framework also achieved the best overall accuracy, outperforming all other algorithms considered in our study.

After getting an overview of the overall accuracy of each model, we evaluated them using the sensitivity SE, precision $PREC$, and F_1 score. For our first set of experiments, we used the macro averaging method when aggregating the per-class results of each metric. In other words, during these experiments, we treated each class equally during the evaluation procedure and calculated the results for all metrics. As seen in Table 6, the newly proposed histogram-based method achieved results similar to our previous framework for the $\lambda = 0.5$ setup; however, it greatly outperformed it with a larger value $\lambda = 1.0$. The best-performing model was the same that achieved the highest accuracies in Table 5: the ensemble that was constructed using three MobileNetv2 models. This model achieved over 90% for each metric: 90.5% sensitivity, 92.1% precision, and 91.1% F_1 score. Similarly to Table 5, one version of [45] achieved 0.1% higher sensitivity when compared to our methods. However, this approach achieved significantly worse results in every other metric: 1.4% lower precision and 0.5% lower F_1 score.

Next, to get a better overview of the performance of the different models, we calculated the weighted sensitivity SE, precision $PREC$, and F_1 score. Each weight was calculated using the support value of the given class, meaning that classes that appear more often would get assigned higher weights. As seen in Table 7, the previously top-performing MobileNetv2 ensemble architecture (see Tables 5 and 6) that used our newly proposed histogram-based framework achieved the best overall results, achieving over 92% for each of the three metrics.

After evaluating the various methods without using any class weights during training, we examined if using a weighted cost function, introduced in Section 2.9, led to any significant changes or problems during the training procedure. During these experiments, each sample was weighted based on the frequency of its label. We calculated the weights β_G, β_M, β_{PT} corresponding to the given class before training the given architecture. We assigned lower weights to classes with larger cardinalities and higher weights to classes with smaller cardinalities in the training set.

As seen in Table 8, we can state that using a weighted cost function did not lead to any instabilities or problems regarding convergence for the examined methods. Although using this approach did not lead to better results as compared with the MobileNetv2 architecture in Table 5, we can observe a drastic improvement for the EfficientNet architecture, which achieved 1.5% higher accuracy as compared to the reported results in Table 5. These results highlight the importance of the framework being able to handle a weighted cost function, as, depending on the architecture, some models, like EfficientNet, may respond exceptionally well to introducing class weights in the loss function, leading to better results for the given architecture.

Next, we also evaluated the effect of using a weighted cost function on each architecture using the macro average of the sensitivity, precision, and F_1 scores. The results can be seen

Table 6 A summary of the measured sensitivity, precision, and F_1 scores using macro averaging

Algorithm	Type	λ	SE	$PREC$	F_1
GLCM [45]	SVM	–	0.749 ± 0.034	0.747 ± 0.036	0.744 ± 0.036
GLCM [45]	KNN	–	0.596 ± 0.016	0.610 ± 0.016	0.585 ± 0.009
GLCM+VGG16 [45]	SVM	–	**0.906 ± 0.016**	0.907 ± 0.012	0.906 ± 0.015
GLCM+VGG16 [45]	KNN	–	0.902 ± 0.012	0.916 ± 0.012	0.907 ± 0.012
Weighted Avg.	–	–	0.898 ± 0.015	0.91 ± 0.017	0.902 ± 0.017
AlexNet	base	–	0.857 ± 0.037	0.871 ± 0.043	0.856 ± 0.045
MobileNetv2	base	–	0.876 ± 0.032	0.894 ± 0.027	0.881 ± 0.033
ShuffleNetv2	base	–	0.844 ± 0.031	0.863 ± 0.028	0.850 ± 0.030
EfficientNet	base	–	0.885 ± 0.018	0.893 ± 0.018	0.886 ± 0.018
AlexNet	cosine	0.0	0.854 ± 0.040	0.874 ± 0.037	0.861 ± 0.040
MobileNetv2	cosine	0.0	0.869 ± 0.026	0.890 ± 0.028	0.875 ± 0.029
ShuffleNetv2	cosine	0.0	0.859 ± 0.043	0.875 ± 0.033	0.864 ± 0.041
EfficientNet	cosine	0.0	0.885 ± 0.030	0.890 ± 0.022	0.885 ± 0.028
Ens	cosine	0.0	0.888 ± 0.015	0.898 ± 0.019	0.891 ± 0.019
AlexNet	histogram	0.0	0.854 ± 0.040	0.874 ± 0.037	0.861 ± 0.040
MobileNetv2	histogram	0.0	0.869 ± 0.026	0.890 ± 0.028	0.875 ± 0.029
ShuffleNetv2	histogram	0.0	0.859 ± 0.043	0.875 ± 0.033	0.864 ± 0.041
EfficientNet	histogram	0.0	0.885 ± 0.030	0.890 ± 0.022	0.885 ± 0.028
Ens	histogram	0.0	0.888 ± 0.015	0.898 ± 0.019	0.891 ± 0.019
AlexNet	cosine	0.5	0.866 ± 0.033	0.879 ± 0.033	0.869 ± 0.035
MobileNetv2	cosine	0.5	0.902 ± 0.019	0.910 ± 0.017	0.905 ± 0.019
ShuffleNetv2	cosine	0.5	0.872 ± 0.038	0.888 ± 0.029	0.877 ± 0.035
EfficientNet	cosine	0.5	0.891 ± 0.020	0.901 ± 0.023	0.893 ± 0.025
Ens	cosine	0.5	0.896 ± 0.018	0.902 ± 0.018	0.897 ± 0.019
AlexNet	histogram	0.5	0.871 ± 0.036	0.886 ± 0.036	0.875 ± 0.037
MobileNetv2	histogram	0.5	0.893 ± 0.010	0.908 ± 0.014	0.898 ± 0.013
ShuffleNetv2	histogram	0.5	0.877 ± 0.021	0.892 ± 0.023	0.882 ± 0.023
EfficientNet	histogram	0.5	0.894 ± 0.013	0.907 ± 0.019	0.898 ± 0.017
Ens	histogram	0.5	0.886 ± 0.014	0.898 ± 0.018	0.888 ± 0.018
AlexNet	cosine	1.0	0.862 ± 0.048	0.884 ± 0.043	0.867 ± 0.049
MobileNetv2	cosine	1.0	0.884 ± 0.028	0.904 ± 0.030	0.892 ± 0.030
ShuffleNetv2	cosine	1.0	0.880 ± 0.033	0.888 ± 0.025	0.880 ± 0.031
EfficientNet	cosine	1.0	0.888 ± 0.015	0.904 ± 0.025	0.892 ± 0.022
Ens	cosine	1.0	0.887 ± 0.035	0.897 ± 0.034	0.889 ± 0.036
AlexNet	histogram	1.0	0.860 ± 0.038	0.878 ± 0.039	0.863 ± 0.045
MobileNetv2	histogram	1.0	0.905 ± 0.011	**0.921 ± 0.009**	**0.911 ± 0.008**
ShuffleNetv2	histogram	1.0	0.886 ± 0.018	0.901 ± 0.018	0.891 ± 0.019
EfficientNet	histogram	1.0	0.892 ± 0.022	0.900 ± 0.027	0.893 ± 0.027
Ens	histogram	1.0	0.891 ± 0.019	0.900 ± 0.022	0.894 ± 0.020

Bold entries signify the highest values

Table 7 A summary of the measured sensitivity, precision, and F_1 scores using weighted averaging

Algorithm	Type	λ	SE	$PREC$	F_1
GLCM [45]	SVM	–	0.764 ± 0.032	0.764 ± 0.037	0.760 ± 0.036
GLCM [45]	KNN	–	0.654 ± 0.013	0.635 ± 0.014	0.628 ± 0.013
GLCM+VGG16 [45]	SVM	–	0.918 ± 0.013	0.919 ± 0.012	0.918 ± 0.013
GLCM+VGG16 [45]	KNN	–	0.919 ± 0.012	0.920 ± 0.011	0.918 ± 0.012
Weighted Avg.	–	–	0.912 ± 0.017	0.916 ± 0.014	0.912 ± 0.017
AlexNet	base	–	0.868 ± 0.042	0.882 ± 0.030	0.868 ± 0.041
MobileNetv2	base	–	0.893 ± 0.032	0.899 ± 0.025	0.892 ± 0.032
ShuffleNetv2	base	–	0.865 ± 0.029	0.872 ± 0.027	0.865 ± 0.029
EfficientNet	base	–	0.898 ± 0.018	0.903 ± 0.017	0.898 ± 0.018
AlexNet	cosine	0.0	0.875 ± 0.036	0.880 ± 0.033	0.874 ± 0.037
MobileNetv2	cosine	0.0	0.888 ± 0.026	0.895 ± 0.020	0.887 ± 0.027
ShuffleNetv2	cosine	0.0	0.877 ± 0.038	0.884 ± 0.033	0.877 ± 0.038
EfficientNet	cosine	0.0	0.896 ± 0.022	0.902 ± 0.018	0.896 ± 0.023
Ens	cosine	0.0	0.903 ± 0.019	0.908 ± 0.014	0.903 ± 0.018
AlexNet	histogram	0.0	0.875 ± 0.036	0.880 ± 0.033	0.874 ± 0.037
MobileNetv2	histogram	0.0	0.888 ± 0.026	0.895 ± 0.020	0.887 ± 0.027
ShuffleNetv2	histogram	0.0	0.877 ± 0.038	0.884 ± 0.033	0.877 ± 0.038
EfficientNet	histogram	0.0	0.896 ± 0.022	0.902 ± 0.018	0.896 ± 0.023
Ens	histogram	0.0	0.903 ± 0.019	0.908 ± 0.014	0.903 ± 0.018
AlexNet	cosine	0.5	0.882 ± 0.031	0.890 ± 0.026	0.882 ± 0.031
MobileNetv2	cosine	0.5	0.915 ± 0.018	0.918 ± 0.017	0.915 ± 0.019
ShuffleNetv2	cosine	0.5	0.892 ± 0.030	0.896 ± 0.028	0.891 ± 0.032
EfficientNet	cosine	0.5	0.904 ± 0.023	0.911 ± 0.016	0.904 ± 0.023
Ens	cosine	0.5	0.908 ± 0.019	0.912 ± 0.017	0.908 ± 0.019
AlexNet	histogram	0.5	0.887 ± 0.034	0.893 ± 0.032	0.887 ± 0.035
MobileNetv2	histogram	0.5	0.908 ± 0.013	0.912 ± 0.012	0.908 ± 0.013
ShuffleNetv2	histogram	0.5	0.895 ± 0.021	0.900 ± 0.018	0.895 ± 0.021
EfficientNet	histogram	0.5	0.909 ± 0.015	0.914 ± 0.012	0.909 ± 0.015
Ens	histogram	0.5	0.901 ± 0.017	0.907 ± 0.012	0.901 ± 0.017
AlexNet	cosine	1.0	0.881 ± 0.045	0.889 ± 0.039	0.880 ± 0.046
MobileNetv2	cosine	1.0	0.904 ± 0.027	0.908 ± 0.024	0.904 ± 0.027
ShuffleNetv2	cosine	1.0	0.881 ± 0.031	0.890 ± 0.026	0.882 ± 0.031
EfficientNet	cosine	1.0	0.905 ± 0.021	0.910 ± 0.017	0.904 ± 0.021
Ens	cosine	1.0	0.901 ± 0.032	0.907 ± 0.028	0.901 ± 0.032
AlexNet	histogram	1.0	0.877 ± 0.042	0.887 ± 0.032	0.876 ± 0.043
MobileNetv2	histogram	1.0	**0.921 ± 0.005**	**0.923 ± 0.004**	**0.921 ± 0.005**
ShuffleNetv2	histogram	1.0	0.902 ± 0.019	0.905 ± 0.016	0.901 ± 0.019
EfficientNet	histogram	1.0	0.904 ± 0.024	0.910 ± 0.018	0.904 ± 0.023
Ens	histogram	1.0	0.906 ± 0.018	0.907 ± 0.020	0.905 ± 0.019

Bold entries signify the highest values

Table 8 A summary of the measured global and class-level accuracies when the frameworks were trained using a weighted cost function

Algorithm	Type	λ	ACC	ACC_M	ACC_G	ACC_{PT}
Afshar et al. [36]	–	–	0.909	–	–	–
Abiwinanda et al. [35]	–	–	0.842	–	–	–
Fasihi et al. [38]	–	–	0.841	–	–	–
MIXCAPS-BoxCaps [37]	–	–	0.913	–	–	–
GLCM [45]	SVM	–	0.764 ± 0.032	0.831 ± 0.036	0.801 ± 0.018	0.897 ± 0.019
GLCM [45]	KNN	–	0.654 ± 0.013	0.750 ± 0.018	0.701 ± 0.014	0.857 ± 0.019
GLCM+VGG16 [45]	SVM	–	0.918 ± 0.013	0.924 ± 0.011	0.947 ± 0.005	$\mathbf{0.964 \pm 0.013}$
GLCM+VGG16 [45]	KNN	–	0.919 ± 0.012	$\mathbf{0.932 \pm 0.010}$	0.943 ± 0.014	0.962 ± 0.008
Weighted Avg.	–	–	0.915 ± 0.019	0.925 ± 0.015	0.949 ± 0.006	0.956 ± 0.023
AlexNet	base	–	0.879 ± 0.038	0.898 ± 0.032	0.912 ± 0.018	0.947 ± 0.033
MobileNetv2	base	–	0.889 ± 0.026	0.903 ± 0.021	0.927 ± 0.016	0.948 ± 0.021
ShuffleNetv2	base	–	0.881 ± 0.014	0.890 ± 0.018	0.928 ± 0.015	0.943 ± 0.022
EfficientNet	base	–	0.898 ± 0.023	0.911 ± 0.018	0.940 ± 0.010	0.944 ± 0.023
AlexNet	cosine	0.0	0.884 ± 0.043	0.907 ± 0.024	0.916 ± 0.030	0.944 ± 0.034
MobileNetv2	cosine	0.0	0.902 ± 0.023	0.922 ± 0.012	0.929 ± 0.020	0.954 ± 0.024
ShuffleNetv2	cosine	0.0	0.890 ± 0.025	0.906 ± 0.020	0.930 ± 0.019	0.945 ± 0.023
EfficientNet	cosine	0.0	0.897 ± 0.019	0.912 ± 0.014	0.935 ± 0.008	0.947 ± 0.029
Ens	cosine	0.0	0.912 ± 0.021	0.921 ± 0.018	0.947 ± 0.004	0.956 ± 0.024
AlexNet	histogram	0.0	0.884 ± 0.043	0.907 ± 0.024	0.916 ± 0.030	0.944 ± 0.034
MobileNetv2	histogram	0.0	0.902 ± 0.023	0.922 ± 0.012	0.929 ± 0.020	0.954 ± 0.024
ShuffleNetv2	histogram	0.0	0.890 ± 0.025	0.906 ± 0.020	0.930 ± 0.019	0.945 ± 0.023
EfficientNet	histogram	0.0	0.897 ± 0.019	0.912 ± 0.014	0.935 ± 0.008	0.947 ± 0.029
Ens	cosine	0.0	0.912 ± 0.021	0.921 ± 0.018	0.947 ± 0.004	0.956 ± 0.024
AlexNet	cosine	0.5	0.892 ± 0.032	0.914 ± 0.022	0.920 ± 0.021	0.950 ± 0.027

Table 8 continued

Algorithm	Type	λ	ACC	ACC_M	ACC_G	ACC_{PT}
MobileNetv2	cosine	0.5	0.912 ± 0.017	0.928 ± 0.012	0.943 ± 0.011	0.954 ± 0.017
ShuffleNetv2	cosine	0.5	0.904 ± 0.033	0.919 ± 0.026	0.937 ± 0.016	0.951 ± 0.029
EfficientNet	cosine	0.5	0.918 ± 0.023	$\mathbf{0.932 \pm 0.020}$	0.950 ± 0.007	0.955 ± 0.022
Ens	cosine	0.5	$\mathbf{0.921 \pm 0.016}$	$\mathbf{0.932 \pm 0.010}$	0.950 ± 0.009	0.959 ± 0.017
AlexNet	histogram	0.5	0.890 ± 0.028	0.907 ± 0.020	0.922 ± 0.016	0.951 ± 0.029
MobileNetv2	histogram	0.5	0.909 ± 0.022	0.926 ± 0.013	0.941 ± 0.010	0.952 ± 0.026
ShuffleNetv2	histogram	0.5	0.913 ± 0.023	0.927 ± 0.014	0.941 ± 0.013	0.957 ± 0.024
EfficientNet	histogram	0.5	0.912 ± 0.020	0.924 ± 0.018	0.943 ± 0.014	0.956 ± 0.021
Ens	histogram	0.5	0.906 ± 0.026	0.922 ± 0.017	0.943 ± 0.016	0.948 ± 0.025
AlexNet	cosine	1.0	0.886 ± 0.032	0.908 ± 0.022	0.921 ± 0.016	0.943 ± 0.032
MobileNetv2	cosine	1.0	0.902 ± 0.017	0.919 ± 0.014	0.935 ± 0.014	0.950 ± 0.017
ShuffleNetv2	cosine	1.0	0.896 ± 0.031	0.913 ± 0.023	0.934 ± 0.023	0.945 ± 0.026
EfficientNet	cosine	1.0	0.919 ± 0.016	0.929 ± 0.014	$\mathbf{0.952 \pm 0.008}$	0.956 ± 0.018
Ens	cosine	1.0	0.914 ± 0.018	0.927 ± 0.019	0.945 ± 0.013	0.956 ± 0.020
AlexNet	histogram	1.0	0.896 ± 0.027	0.913 ± 0.027	0.928 ± 0.004	0.952 ± 0.027
MobileNetv2	histogram	1.0	0.909 ± 0.024	0.924 ± 0.019	0.941 ± 0.017	0.953 ± 0.020
ShuffleNetv2	histogram	1.0	0.907 ± 0.026	0.919 ± 0.020	0.940 ± 0.020	0.956 ± 0.022
EfficientNet	histogram	1.0	0.914 ± 0.023	0.928 ± 0.021	0.946 ± 0.009	0.955 ± 0.022
Ens	histogram	1.0	0.902 ± 0.020	0.915 ± 0.015	0.938 ± 0.014	0.951 ± 0.022

Bold entries signify the highest values

in Table 9. Similarly to the previous results, the EfficientNet-based ensemble architecture achieved good results, while the hybrid method using three different architectures (denoted by Ens) had the best results.

Finally, we used the weighted averaging method to measure the same metrics SE, $PREC$, and F_1 score. As shown in Table 10, the best-performing architectures were the ones using the EfficientNet network, achieving close to 92% sensitivity, precision, and F_1 score, and the hybrid architecture denoted by Ens, which achieved the best results with 92.1% sensitivity, 92.5% precision, and 92.1% F_1 score, respectively.

4 Further validation and limitations

Having evaluated our frameworks using the previously detailed metrics, it can be stated that the model can solve the problem with sufficient accuracy and reliability. However, for deep learning-based solutions it is just as important to understand what the given model bases its decisions on. For CNN-based methods, a widely used and accepted way to achieve this is by applying GRAD-CAM [61] on one of the layers of the convolutional layers, usually on the last one. Therefore, we also applied GRAD-CAM [61] to get a visual feedback and rough understanding of what the ensemble is focusing on, as well as to gauge the applicability and usefulness of the proposed solutions.

However, it is no trivial task to apply GRAD-CAM to our frameworks. This is because they do not connect the members with a joint dense layer. Instead, only the similarity between the extracted features is measured. That is, there is no shared layer between the member models, from which the gradients could be computed and to which GRAD-CAM could be applied. Therefore, we applied GRAD-CAM to the last convolutional layer of the members inside the ensemble to visualize which parts of the input image each of them focuses on. Then, we averaged the resulting GRAD-CAM heatmaps generated by each member to get a general idea of how the ensemble works in general. If the proposed frameworks work correctly, that would mean that the member models rely on different sets of features, potentially highlighting different parts of the images, while the averaged results should cover most of the area of the tumors. This should in turn be visible from the obtained GRAD-CAM results as well. The GRAD-CAM visualizations of the best-performing model from Table 5 can be seen in Fig. 11.

In Fig. 11, we can see that all of the member models can locate the tumors in all of the images. This shows that by using our proposed frameworks, each of the members will still be able to solve the original task and hence can potentially be used on their own as well. However, by looking at the areas highlighted by the models, we can also see that each model puts more emphasis on different areas of the same image. Namely, it seems that M_1 learned to observe features that are spread out more evenly inside and around the tumor boundaries. M_2 on the other hand seems to capture more vertical features from the images that relate more to the length of the tumors along the y axis of the images. Lastly, M_3 seems to put extra emphasis on the borders of the tumors. The fact that the GRAD-CAM of these three models differ so much seem to support our initial hypothesis. That is, the models trained according to our frameworks seem to learn highly diverse, non-redundant features. This can be highly useful in a variety of cases. For example, in the third row of Fig. 11, we can see how the average GRAD-CAM image seems to better capture the location of the tumor which spreads out in all directions.

Table 9 A summary of the measured sensitivity, precision, and F_1 scores using macro averaging when the frameworks were trained using a weighted cost function

Algorithm	Type	λ	SE	$PREC$	F_1
GLCM [45]	SVM	–	0.749 ± 0.034	0.747 ± 0.036	0.744 ± 0.036
GLCM [45]	KNN	–	0.596 ± 0.016	0.610 ± 0.016	0.585 ± 0.009
GLCM+VGG16 [45]	SVM	–	0.906 ± 0.016	0.907 ± 0.012	0.906 ± 0.015
GLCM+VGG16 [45]	KNN	–	0.902 ± 0.012	**0.916 ± 0.012**	0.907 ± 0.012
Weighted Avg.	–	–	0.909 ± 0.019	0.907 ± 0.020	0.905 ± 0.022
AlexNet	base	–	0.868 ± 0.040	0.874 ± 0.036	0.867 ± 0.040
MobileNetv2	base	–	0.879 ± 0.020	0.884 ± 0.025	0.877 ± 0.026
ShuffleNetv2	base	–	0.876 ± 0.006	0.874 ± 0.014	0.869 ± 0.013
EfficientNet	base	–	0.888 ± 0.028	0.888 ± 0.022	0.885 ± 0.026
AlexNet	cosine	0.0	0.872 ± 0.043	0.881 ± 0.037	0.872 ± 0.045
MobileNetv2	cosine	0.0	0.890 ± 0.023	0.900 ± 0.020	0.893 ± 0.023
ShuffleNetv2	cosine	0.0	0.883 ± 0.025	0.884 ± 0.022	0.880 ± 0.025
EfficientNet	cosine	0.0	0.882 ± 0.018	0.893 ± 0.021	0.884 ± 0.021
Ens	cosine	0.0	0.906 ± 0.019	0.904 ± 0.022	0.901 ± 0.024
AlexNet	histogram	0.0	0.872 ± 0.043	0.881 ± 0.037	0.872 ± 0.045
MobileNetv2	histogram	0.0	0.890 ± 0.023	0.900 ± 0.020	0.893 ± 0.023
ShuffleNetv2	histogram	0.0	0.883 ± 0.025	0.884 ± 0.022	0.880 ± 0.025
EfficientNet	histogram	0.0	0.882 ± 0.018	0.893 ± 0.021	0.884 ± 0.021
Ens	histogram	0.0	0.906 ± 0.019	0.904 ± 0.022	0.901 ± 0.024
AlexNet	cosine	0.5	0.877 ± 0.034	0.891 ± 0.029	0.881 ± 0.034
MobileNetv2	cosine	0.5	0.901 ± 0.018	0.909 ± 0.017	0.902 ± 0.019
ShuffleNetv2	cosine	0.5	0.893 ± 0.033	0.901 ± 0.032	0.893 ± 0.037
EfficientNet	cosine	0.5	0.905 ± 0.026	0.914 ± 0.025	0.908 ± 0.026
Ens	cosine	0.5	**0.910 ± 0.016**	**0.916 ± 0.017**	**0.911 ± 0.017**
AlexNet	histogram	0.5	0.876 ± 0.027	0.885 ± 0.028	0.878 ± 0.030
MobileNetv2	histogram	0.5	0.900 ± 0.020	0.905 ± 0.020	0.899 ± 0.024
ShuffleNetv2	histogram	0.5	0.902 ± 0.025	0.907 ± 0.021	0.902 ± 0.026
EfficientNet	histogram	0.5	0.901 ± 0.018	0.909 ± 0.022	0.902 ± 0.021
Ens	histogram	0.5	0.893 ± 0.024	0.902 ± 0.025	0.894 ± 0.028
AlexNet	cosine	1.0	0.889 ± 0.014	0.897 ± 0.017	0.891 ± 0.016
MobileNetv2	cosine	1.0	0.889 ± 0.014	0.897 ± 0.017	0.891 ± 0.016
ShuffleNetv2	cosine	1.0	0.883 ± 0.036	0.891 ± 0.027	0.884 ± 0.034
EfficientNet	cosine	1.0	0.906 ± 0.014	0.915 ± 0.022	0.908 ± 0.017
Ens	cosine	1.0	0.903 ± 0.017	0.909 ± 0.020	0.904 ± 0.019
AlexNet	histogram	1.0	0.883 ± 0.028	0.892 ± 0.031	0.885 ± 0.031
MobileNetv2	histogram	1.0	0.896 ± 0.021	0.906 ± 0.026	0.899 ± 0.024
ShuffleNetv2	histogram	1.0	0.897 ± 0.027	0.901 ± 0.024	0.897 ± 0.027
EfficientNet	histogram	1.0	0.903 ± 0.025	0.910 ± 0.025	0.904 ± 0.026
Ens	histogram	1.0	0.890 ± 0.018	0.897 ± 0.020	0.890 ± 0.021

Bold entries signify the highest values

Table 10 A summary of the measured sensitivity, precision, and F_1 scores using weighted averaging when the frameworks were trained using a weighted cost function

Algorithm	Type	λ	SE	$PREC$	F_1
GLCM [45]	SVM	–	0.764 ± 0.032	0.764 ± 0.037	0.760 ± 0.036
GLCM [45]	KNN	–	0.654 ± 0.013	0.635 ± 0.014	0.628 ± 0.013
GLCM+VGG16 [45]	SVM	–	0.918 ± 0.013	0.919 ± 0.012	0.918 ± 0.013
GLCM+VGG16 [45]	KNN	–	0.919 ± 0.012	0.920 ± 0.011	0.918 ± 0.012
Weighted Avg.	–	–	0.915 ± 0.019	0.922 ± 0.014	0.916 ± 0.018
AlexNet	base	–	0.879 ± 0.038	0.886 ± 0.034	0.879 ± 0.038
MobileNetv2	base	–	0.889 ± 0.026	0.898 ± 0.017	0.890 ± 0.024
ShuffleNetv2	base	–	0.881 ± 0.014	0.895 ± 0.010	0.883 ± 0.013
EfficientNet	base	–	0.898 ± 0.023	0.905 ± 0.024	0.898 ± 0.024
AlexNet	cosine	0.0	0.884 ± 0.043	0.892 ± 0.033	0.884 ± 0.042
MobileNetv2	cosine	0.0	0.902 ± 0.023	0.907 ± 0.019	0.902 ± 0.023
ShuffleNetv2	cosine	0.0	0.890 ± 0.025	0.899 ± 0.022	0.891 ± 0.025
EfficientNet	cosine	0.0	0.897 ± 0.019	0.904 ± 0.016	0.897 ± 0.018
Ens	cosine	0.0	0.912 ± 0.021	0.919 ± 0.015	0.913 ± 0.020
AlexNet	histogram	0.0	0.884 ± 0.043	0.892 ± 0.033	0.884 ± 0.042
MobileNetv2	histogram	0.0	0.902 ± 0.023	0.907 ± 0.019	0.902 ± 0.023
ShuffleNetv2	histogram	0.0	0.890 ± 0.025	0.899 ± 0.022	0.891 ± 0.025
EfficientNet	histogram	0.0	0.897 ± 0.019	0.904 ± 0.016	0.897 ± 0.018
Ens	cosine	0.0	0.912 ± 0.021	0.919 ± 0.015	0.913 ± 0.020
AlexNet	cosine	0.5	0.892 ± 0.032	0.898 ± 0.027	0.892 ± 0.033
MobileNetv2	cosine	0.5	0.912 ± 0.017	0.917 ± 0.014	0.912 ± 0.017
ShuffleNetv2	cosine	0.5	0.904 ± 0.033	0.911 ± 0.024	0.904 ± 0.033
EfficientNet	cosine	0.5	0.918 ± 0.023	0.922 ± 0.023	0.918 ± 0.023
Ens	cosine	0.5	$\mathbf{0.921 \pm 0.016}$	$\mathbf{0.925 \pm 0.013}$	$\mathbf{0.921 \pm 0.016}$
AlexNet	histogram	0.5	0.890 ± 0.028	0.895 ± 0.023	0.890 ± 0.028
MobileNetv2	histogram	0.5	0.909 ± 0.022	0.915 ± 0.015	0.909 ± 0.022
ShuffleNetv2	histogram	0.5	0.913 ± 0.023	0.917 ± 0.020	0.913 ± 0.024
EfficientNet	histogram	0.5	0.912 ± 0.020	0.918 ± 0.017	0.912 ± 0.020
Ens	histogram	0.5	0.906 ± 0.026	0.912 ± 0.021	0.906 ± 0.026
AlexNet	cosine	1.0	0.886 ± 0.032	0.892 ± 0.028	0.885 ± 0.034
MobileNetv2	cosine	1.0	0.902 ± 0.017	0.906 ± 0.016	0.902 ± 0.017
ShuffleNetv2	cosine	1.0	0.896 ± 0.031	0.903 ± 0.027	0.896 ± 0.031
EfficientNet	cosine	1.0	0.919 ± 0.016	0.923 ± 0.016	0.918 ± 0.016
Ens	cosine	1.0	0.914 ± 0.018	0.917 ± 0.018	0.914 ± 0.018
AlexNet	histogram	1.0	0.896 ± 0.027	0.901 ± 0.026	0.897 ± 0.027
MobileNetv2	histogram	1.0	0.909 ± 0.024	0.914 ± 0.021	0.909 ± 0.024
ShuffleNetv2	histogram	1.0	0.907 ± 0.026	0.912 ± 0.023	0.908 ± 0.026
EfficientNet	histogram	1.0	0.914 ± 0.023	0.919 ± 0.021	0.915 ± 0.023
Ens	histogram	1.0	0.902 ± 0.020	0.909 ± 0.014	0.903 ± 0.019

Bold entries signify the highest values

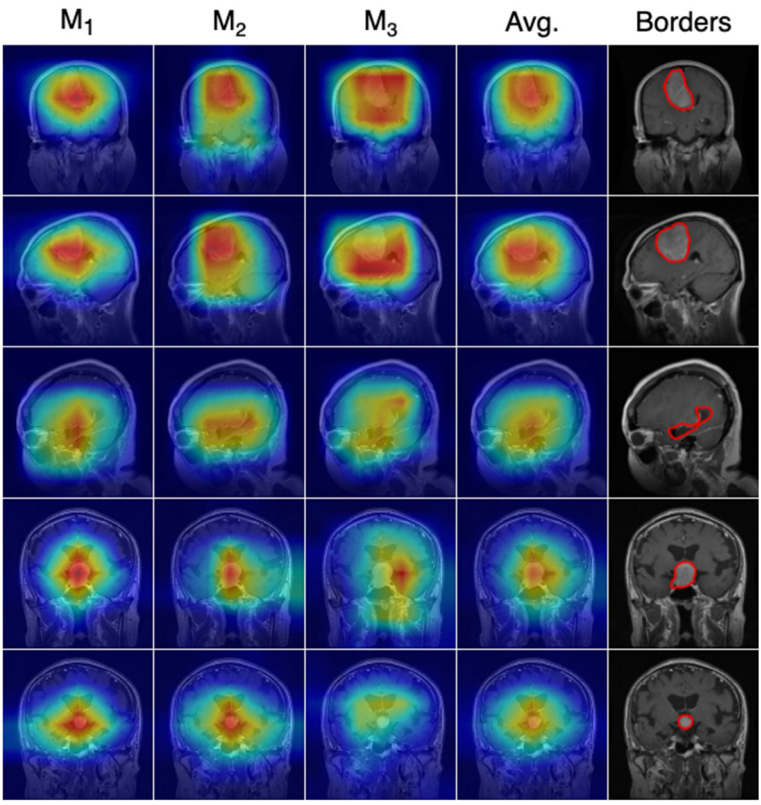

Fig. 11 GRAD-CAM visualizations of the member models of the proposed framework and the average of the GRAD-CAM heatmaps for a few sample inputs. The ground truth borders of the tumors are highlighted in red

Lastly, it is also important to mention the potential limitations of the frameworks. Even though the proposed frameworks are sufficiently generic and could potentially be used in other domains as well, as of now, they have only been validated for the task of image classification. Furthermore, although the frameworks increase the diversity of the ensemble which can help in increasing the accuracy, they still rely on the member models. Hence, the architecture of these members and the quality of the training data all play a significant role in the final performance of the frameworks. See for example Tables 5, 6, 7 8, 9, and 10, where it can be clearly seen that older architectures, such as AlexNet perform worse than more modern architectures such as MobileNetv2 and EfficientNet for the base models. This difference stays even after applying our proposed frameworks. That is, ensembles that use the MobileNetv2 and EfficientNet architectures tend to perform better than those that used AlexNet. Therefore, it is important to carefully choose the architecture of the member models and select the ones that should be used for solving the given problem. Lastly, if most of the models inside the ensemble fail to classify a given image, the ensemble naturally will not be able to do so either. In Fig. 12, we can see a few cases where the ensemble failed to classify the images correctly. In these cases, there was either no consensus in the predictions or the wrong class was predicted. For these images, the main reason why the members failed to predict the correct class labels were likely that the tumors were too small (first, second and fifth image in Fig. 12), the tumor blended in with the background (third image in Fig. 12), or the image was to blurry (fourth image in Fig. 12).

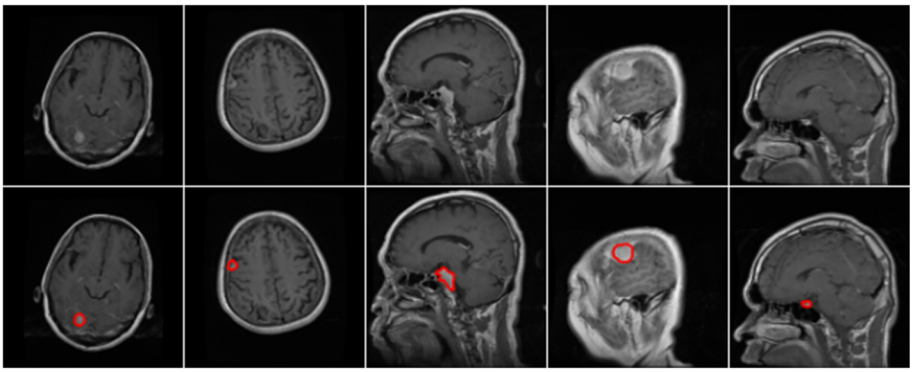

Fig. 12 Sample input images that the framework failed to correctly classify (top row) and the tumor boundaries highlighted in red (bottom row)

5 Conclusions

In our work, we revisited the problem of building state-of-the-art, diverse ensemble models for the accurate and reliable classification of brain tumors. For our research, we considered some of the most commonly used, state-of-the-art CNN-based architectures, like AlexNet, MobileNetv2, ShuffleNetv2, and EfficientNet. Using these base architectures, we built ensembles that directly maximize the diversity between the member models. We also compared our experimental results to other works that used the same dataset and traditional ensemble approaches, like majority voting and averaging the outputs of the models. During our experiments, we used a publicly available and clinically tested dataset containing MRI images of some of the most commonly occurring brain tumors, namely meningioma, glioma, and pituitary tumors. We also showed that this dataset provided an excellent opportunity to evaluate the performance of each model as it contained five folds of training and test parts defined by the authors of the dataset.

We showed that we can use the last layers of CNNs and treat them as low-dimensional vectors in a latent vector space that captures the most important aspects and features of the given input image and use these vectors to increase diversity inside the ensemble. For this aim, we first summarized our previous framework using cosine similarity to measure the similarity between the extracted latent feature vectors of the ensemble members. We detailed that this framework had several advantages and improvements compared to other widely used ensemble techniques, such as majority voting or calculating the weighted average of the outputs of the members of the ensemble. Namely, the proposed framework makes it possible to train the ensemble members simultaneously and calculate the outputs and gradients for each member using the same batch, thus decreasing the training time drastically. We also showed that using this approach, we can actually use the exact same architecture multiple times inside the ensemble since, by measuring the similarity between the feature vectors, we can theoretically guarantee that the members will operate on highly dissimilar sets of features, something that is not measured or guaranteed by other approaches, like majority voting. Lastly, we also showed that using diversity as a guiding feature for optimization actually leads to better performance when evaluating the architectures.

In this paper, we also highlighted some theoretical problems with our previous framework and showed that these problems can arise in practice as well using some simple datasets, like MNIST and Medical MNIST. Then, we proposed our new and improved framework that uses the histogram loss instead of the cosine similarity to overcome these problems. We detailed the theoretical background of the histogram loss and showed the advantages of using it. Namely, unlike in the case of cosine similarity, this metric can recognize patterns and similarities in two vectors even if the order of their similar components is not the same. We also presented several improved versions of the frameworks that make it possible to use them even when the member models produce feature vectors with varying sizes and for imbalanced datasets as well, using a weighted cost function. To prove the effectiveness of our newly proposed framework, we compared its results with our previous framework and concluded that the new framework performed remarkably better than the previous one. We concluded that each variant of the proposed ensembles could solve the task with high accuracy, sensitivity, precision, as well as F_1 score, making them a potential choice for integrating into a CAD system for reliable detection of brain tumors at their early stages. As shown in Section 3, our newly proposed histogram-based framework greatly outperformed both our previous results and the baselines. The best-performing model was the ensemble containing three MobileNetv2 networks and trained using a λ value of 1.0, achieving 92.1% global accuracy and 93.5%, 94.5%, and 96.3% accuracy for the meningioma, glioma, and pituitary tumor classes, respectively. The only metric that did not improve when compared with our previous framework was the accuracy belonging to the glioma class, which may indicate that the ensemble reached its maximum performance for this particular class and dataset. Moreover, it also surpassed several other state-of-the-art approaches from the current literature and the base architectures, achieving more than 92% macro precision, more than 91% macro F_1 score, and over 90% sensitivity, as well as over 92% weighted precision, more than 92% weighted F_1 score, and over 92% weighted sensitivity. When using a weighted cost function, the EfficientNet-based and the hybrid ensemble architectures (constructed of the MobileNetv2, ShuffleNetv2, and EfficientNet architectures) using the cosine similarity measure were found to perform the best. The proposed framework also surpassed methods that relied on multi-step processing of carefully extracted hand-crafted features, showing that it is possible to reduce the complexity and the time needed to deliver performant and reliable deep learning-based solutions, further demonstrating its usefulness in practice. Even though in this work we focused on the task of image classification, the proposed framework can be used for other tasks as well due to their flexible nature. Therefore, in the future, we plan to experiment with applying the frameworks for segmentation and object detection as well. We made the source code of the developed frameworks available at [62].

Acknowledgements This research was supported by the ÚNKP-23-3-II-DE-119 New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development, and Innovation Fund, and by the project TKP2021-NKTA-34, implemented with the support provided by the National Research, Development and Innovation Fund of Hungary under the TKP2021-NKTA funding scheme.

Funding Open access funding provided by University of Debrecen.

Data Availability The dataset used in this research was published by the authors of [8] and is publicly available at https://doi.org/10.6084/m9.figshare.1512427.v5.

Declarations

Conflict of Interests The authors declare that they have no conflicts of interest.

References

1. Miller KD, Fidler-Benaoudia M, Keegan TH et al (2020) Cancer statistics for adolescents and young adults, 2020. CA: a cancer journal for clinicians. 70(6), 443–459
2. Miller KD, Ostrom QT, Kruchko C et al (2021) Brain and other central nervous system tumor statistics, 2021. CA: a cancer journal for clinicians. 71(5), 381–406
3. Alentorn A, Hoang-Xuan K, Mikkelsen T (2016) Presenting signs and symptoms in brain tumors. Handb Clin Neurol 134:19–26
4. Fan Y, Zhang X, Gao C et al (2022) Burden and trends of brain and central nervous system cancer from 1990 to 2019 at the global, regional, and country levels. Arch Public Health 80(1):1–14
5. Ostrom QT, Price M, Neff C et al(2022) Cbtrus statistical report: primary brain and other central nervous system tumors diagnosed in the united states in 2015–2019. Neuro-oncology. 24(Supplement_5), 1–95
6. Siegel RL, Miller KD, Wagle NS et al (2023) Cancer statistics, 2023. CA: a cancer journal for clinicians. 73(1), 17–48
7. Cheng J Brain tumor dataset. https://doi.org/10.6084/m9.figshare.1512427.v5. Accessed 12 Dec 2023
8. Cheng J, Yang W, Huang M et al (2016) Retrieval of brain tumors by adaptive spatial pooling and fisher vector representation. PLoS ONE 11(6):0157112
9. Gao F, Wu T, Li J et al (2018) Sd-cnn: a shallow-deep cnn for improved breast cancer diagnosis. Comput Med Imaging Graph 70:53–62
10. Titoriya A, Sachdeva S (2019) Breast cancer histopathology image classification using alexnet. In: 2019 4th International Conference on Information Systems and Computer Networks (ISCON), IEEE, pp 708–712
11. Khairandish MO, Sharma M, Jain V et al (2022) A hybrid cnn-svm threshold segmentation approach for tumor detection and classification of mri brain images. Irbm 43(4):290–299
12. Kibriya H, Masood M, Nawaz M et al (2022) Multiclass classification of brain tumors using a novel cnn architecture. Multimed Tools Appl 81(21):29847–29863
13. Zhang N, Cai Y-X, Wang Y-Y et al (2020) Skin cancer diagnosis based on optimized convolutional neural network. Artif Intell Med 102
14. Hekal AA, Moustafa HE-D, Elnakib A (2022) Ensemble deep learning system for early breast cancer detection. Evolutionary Intelligence, 1–10
15. Gupta N, Bhatele P, Khanna P (2019) Glioma detection on brain mris using texture and morphological features with ensemble learning. Biomed Sig Process Control 47:115–125
16. Nguyen QH, Do TT, Wang Y et al (2019) Breast cancer prediction using feature selection and ensemble voting. In: 2019 International Conference on System Science and Engineering (ICSSE) IEEE, pp 250–254
17. Krizhevsky A, Sutskever I, Hinton GE (2017) Imagenet classification with deep convolutional neural networks. Commun ACM 60(6):84–90
18. Lu S, Wang S-H, Zhang Y-D (2021) Detection of abnormal brain in mri via improved alexnet and elm optimized by chaotic bat algorithm. Neural Comput Appl 33:10799–10811
19. Sandler M, Howard A, Zhu M et al (2018) Mobilenetv2: inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4510–4520
20. Arfan TH, Hayaty M, Hadinegoro A (2021) Classification of brain tumours types based on mri images using mobilenet. In: 2021 2nd International Conference on Innovative and Creative Information Technology (ICITech) IEEE, pp 69–73
21. Roslidar R, Saddami K, Arnia F et al (2019) A study of fine-tuning cnn models based on thermal imaging for breast cancer classification. In: 2019 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom) IEEE, pp 77–81

22. Tan M, Le Q (2019) Efficientnet: rethinking model scaling for convolutional neural networks. In: International Conference on Machine Learning, PMLR pp 6105–6114
23. Shah HA, Saeed F, Yun S et al (2022) A robust approach for brain tumor detection in magnetic resonance images using finetuned efficientnet. IEEE Access 10:65426–65438
24. Ma N, Zhang X, Zheng H-T et al (2018) Shufflenet v2: practical guidelines for efficient cnn architecture design. In: Proceedings of the European Conference on Computer Vision (ECCV), pp 116–131
25. Bogacsovics G, Harangi B, Hajdu A (2023) Increasing the diversity of ensemble members for accurate brain tumor classification. In: 2023 IEEE 36th International Symposium on Computer-Based Medical Systems (CBMS) IEEE, pp 529–534
26. Ustinova E, Lempitsky V (2016) Learning deep embeddings with histogram loss. Advances in neural information processing systems. 29
27. Liu L, Wei W, Chow K-H et al (2019) Deep neural network ensembles against deception: ensemble diversity, accuracy and robustness. In: 2019 IEEE 16th International conference on Mobile Ad Hoc and Sensor Systems (MASS) IEEE, pp 274–282
28. Zhang S, Liu M, Yan J (2020) The diversified ensemble neural network. Adv Neural Inf Process Syst 33:16001–16011
29. LeCun Y, Bottou L, Bengio Y et al (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
30. apolanco3225: Medical MNIST Classification. GitHub (2017)
31. Ruppertshofen H, Lorenz C, Rose G et al (2013) Discriminative generalized hough transform for object localization in medical images. Int J Comput Assist Radiol Surg 8:593–606
32. Chaira T (2011) A novel intuitionistic fuzzy c means clustering algorithm and its application to medical images. Appl Soft Comput 11(2):1711–1717
33. Ayyad SM, Badawy MA, Shehata M et al (2022) A new framework for precise identification of prostatic adenocarcinoma. Sensors 22(5):1848
34. Balaha HM, Ayyad SM, Alksas A et al (2023) Early diagnosis of prostate cancer using parametric estimation of ivim from dw-mri. In: 2023 IEEE International Conference on Image Processing (ICIP) IEEE, pp 2910–2914
35. Abiwinanda N, Hanif M, Hesaputra ST et al (2019) Brain tumor classification using convolutional neural network. In: World Congress on Medical Physics and Biomedical Engineering 2018: June 3-8, 2018, Prague, Czech Republic, vol 1. Springer, pp 183–189
36. Afshar P, Plataniotis KN, Mohammadi A (2019) Capsule networks for brain tumor classification based on mri images and coarse tumor boundaries. In: ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) IEEE, pp 1368–1372
37. Afshar P, Naderkhani F, Oikonomou A et al (2021) A capsule network-based mixture of experts for lung nodule malignancy prediction. Pattern Recog 116
38. Fasihi MS, Mikhael WB (2020) Mri brain tumor classification employing transform domain projections. In: 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS) IEEE, pp 1020–1023
39. Talukder MA, Islam MM, Uddin MA et al (2022) Machine learning-based lung and colon cancer detection using deep feature extraction and ensemble learning. Expert Syst Appl 205
40. Baradaran Rezaei H, Amjadian A, Sebt MV et al (2022) An ensemble method of the machine learning to prognosticate the gastric cancer. Annals of Operations Research, 1–42
41. Xiao Y, Wu J, Lin Z et al (2018) A deep learning-based multi-model ensemble method for cancer prediction. Comput Methods Programs Biomed 153:1–9
42. Pramanik R, Biswas M, Sen S et al (2022) A fuzzy distance-based ensemble of deep models for cervical cancer detection. Comput Methods Programs Biomed 219
43. Szegedy C, Vanhoucke V, Ioffe S et al (2016) Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2818–2826
44. Szegedy C, Ioffe S, Vanhoucke V et al (2017) Inception-v4, inception-resnet and the impact of residual connections on learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol 31
45. Kibriya H, Amin R, Kim J et al (2023) A novel approach for brain tumor classification using an ensemble of deep and hand-crafted features. Sensors 23(10):4693
46. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556
47. Pattnaik S, Nayak AK (2019) Summarization of odia text document using cosine similarity and clustering. In: 2019 International Conference on Applied Machine Learning (ICAML) IEEE, pp 143–146
48. Singh R, Singh S (2021) Text similarity measures in news articles by vector space model using nlp. Journal of The Institution of Engineers (India): Series B. 102, 329–338

49. Pal S, Chang M, Iriarte MF (2021) Summary generation using natural language processing techniques and cosine similarity. In: International conference on intelligent systems design and applications, Springer, pp 508–517
50. Caron M, Touvron H, Misra I et al (2021) Emerging properties in self-supervised vision transformers. In: Proceedings of the IEEE/CVF international conference on computer vision, pp 9650–9660
51. Chen W, Liu Y, Wang W et al (2022) Deep learning for instance retrieval: a survey. IEEE Transactions on Pattern Analysis and Machine Intelligence
52. Somepalli G, Singla V, Goldblum M et al (2023) Diffusion art or digital forgery? investigating data replication in diffusion models. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 6048–6058
53. Krizhevsky A, Hinton G, et al (2009) Learning multiple layers of features from tiny images
54. Kaziha O, Bonny T (2019) A comparison of quantized convolutional and lstm recurrent neural network models using mnist. In: 2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA) IEEE, pp 1–5
55. Alvear-Sandoval RF, Sancho-Gómez JL, Figueiras-Vidal AR (2019) On improving cnns performance: the case of mnist. Inf Fusion 52:106–109
56. Cheng K, Tahir R, Eric LK et al (2020) An analysis of generative adversarial networks and variants for image synthesis on mnist dataset. Multimed Tools Appl 79:13725–13752
57. Slany E, Ott Y, Scheele S et al (2022) Caipi in practice: towards explainable interactive medical image classification. In: IFIP International conference on artificial intelligence applications and innovations, Springer, pp 389–400
58. Bhatia L, Samet S (2023) A decentralized data evaluation framework in federated learning. Research and Applications, Blockchain, p 100152
59. Chollet F (2021) Deep Learn Python, 2nd edn. Manning, New York
60. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv:1412.6980
61. Selvaraju RR, Cogswell M, Das A et al (2017) Grad-cam: visual explanations from deep networks via gradient-based localization. In: Proceedings of the IEEE international conference on computer vision, pp 618–626
62. Bogacsovics G, Harangi B, Hajdu A Brain tumor classification software. https://github.com/gergobogacsovics/BrainTumorClassification. Accessed 30 Mar 2024