Divide2Conquer (D2C): A Decentralized Approach Towards Overfitting Remediation in Deep Learning

Md. Saiful Bari Siddiqui
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
saiful.bari@bracu.ac.bd

Md Mohaiminul Islam
Department of Computer Science and Engineering
United International University
Dhaka, Bangladesh
mohaiminul@cse.uiu.ac.bd

Md. Golam Rabiul Alam
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
rabiul.alam@bracu.ac.bd

***Abstract*—Overfitting remains a persistent challenge in deep learning. It is primarily attributed to data outliers, noise, and limited training set sizes. This paper presents Divide2Conquer (D2C), a novel technique designed to address this issue. D2C proposes partitioning the training data into multiple subsets and training separate identical models on them. To avoid overfitting on any specific subset, the trained parameters from these models are aggregated and scaled periodically throughout the training phase, enabling the model to learn from the entire dataset while mitigating the impact of individual outliers or noise. Empirical evaluations on multiple benchmark datasets across various deep learning tasks demonstrate that D2C effectively improves generalization performance, particularly for larger datasets. This study verifies D2C's ability to achieve significant performance gains both as a standalone technique and when used in conjunction with other overfitting reduction methods through a series of experiments, including analysis of decision boundaries, loss curves, and other performance metrics. It also provides valuable insights into the implementation and hyperparameter tuning of D2C. Our codes are publicly available at: https://github.com/Saiful185/Divide2Conquer.**

***Index Terms*—Deep Learning, Hyperparameter, Image Classification, Overfitting, Text Classification.**

I. Introduction

Deep Learning models often do well on the data they train on, but the performance drops massively on unseen data from different distributions. This phenomenon is known as Overfitting. Many methods have been proposed to reduce overfitting over the years [8]. Early Stopping [19] is one of the most intuitive methods used for reducing overfitting. However, it stops models from utilizing the learning process properly [20]. Network reduction views outlier data points as noise and reduces the model complexity. However, this method also brings a constraint in learning complex features [21]. Data Augmentation is another way of addressing overfitting and is often the go-to method followed in many deep learning applications [22]. However, selecting the appropriate data augmentation technique for a particular dataset is often tricky. Acquiring more training data often requires monumental efforts too. Regularization is one of the most influential and popular techniques to reduce overfitting. They are associated with penalties so that the model is not entirely dependent on the training data points. Dropout is probably the most relevant overfitting-reducing method for deep learning models [4]. It is another regularizing technique that drops a portion of the connections between neural network layers. However, even dropout can't always counter overfitting and sometimes degrades the performance of a model [13]. Eventually, the model still trains on the entire training data at once, and the robust neural network structures find their ways to fit on the training set a bit too much. This is where our motivation comes from. Perhaps the neural network should not be allowed to train on the entire dataset. Maybe we should combine multiple models that train on different portions of the data.

Our study is loosely inspired by Federated Optimization [2]. The federated optimization technique suggests training in each edge device using a similar network on the data available in that local device. However, after a specific time, each device sends the parameter weights of its model(not the data) to a central server, where the weights are aggregated and averaged. Finally, the averaged weights are sent to all the local devices again and the loop continues. Federated optimization, however, is aimed at safe utilization of data situated in edge devices, not better generalization.

In our study, we implemented a novel method, Divide2Conquer (D2C). We divided our training data into multiple subsets and trained a model each(all having the same architecture) with the training subsets. After an epoch, or, a few epochs, we performed weighted averaging of all the parameter weights from all the subsets and shared the weights back with the subset models. Then the whole process is repeated for several global epochs.

The K-Fold-Cross-Validation method for evaluation proposed by Korjus K. et al. [9] has some similarities to our proposed method in the sense that it also creates subsets by partitioning the dataset and makes use of the whole training data. However, this method is different from ours since the erroneous samples still make it to the training phase (K-1/K)-th of the time in K-Fold CV. D2C method partitions data and the whole training process is partitioned too, which means none of the subsets is fed with a particular outlier while training bar one, minimizing the impact of these erroneous samples.

This work was supported by the Institute for Advanced Research (IAR), United International University under Grant UIU-IAR-02-2022-SE-06.

D2C method also has similarities with Bagging [15] in the form of using subsets of data. However, in bagging (Bootstrap Aggregating), the data is not completely separated for different models as we proposed. Instead, the technique involves creating multiple subsets of the original dataset by randomly sampling with replacement. This means that some data points may be repeated in a subset, while others may be excluded altogether. Also, in bagging, the aggregation is done by combining the output probabilities, not the model weights.

Our experiments encompass multiple datasets across different domains, and each of the cases suggests dividing the training set into multiple subsets, training them separately, and averaging the parameters after every few epochs can significantly reduce overfitting. To evaluate the performance, the primary comparison was between the performances of the base Neural Network architecture used for the subset models using the entire training data and the performance of our models using Divide2Conquer method. We summarized our contributions through this study below:

- We introduced a new method, D2C, that helps in reducing overfitting significantly, while being conceptually simple and easy to implement.
- D2C method can be applied on top of many other data augmentation and regularization techniques, and our experiments show that this results in a clear improvement in the model's generalization ability.
- We also extensively tuned the hyperparameters introduced by this method and reported the findings, providing important directions for future applications.

The paper is organized as follows. In section II, we discuss relevant literature. We establish the theoretical justification behind our hypothesis in section III. In sections IV and V, we discuss our methodology and lay out the experimental specifications. In section VI, we evaluate our approach through empirical experiments on multiple datasets and analyze the results. Section VII contains concluding remarks.

II. Related Works

Several authors have addressed the issue of overfitting while performing various classification tasks. M. Cogswell et al. proposed a new regularizer called DeCov which helps reduce overfitting in deep neural networks by Decorrelating Representations [5]. Dropout [4] was proposed by Srivastava et al. back in 2014, and since then, this technique has been extensively used in very complex neural network architectures successfully. It was indeed an outstanding contribution specifically for deep learning-based models. Batch Normalization [6] proposed by Ioffe and Sergei primarily focuses on better convergence and somewhat contributes to reducing overfitting. J. Kolluri and V.K Kotte came up with L¼ regularization to solve the problems faced by L1 and L2 regularization techniques [10]. Ensembles are also often used to improve generalization. The ensemble-based ELVD model [3] managed to outperform the traditional VGGNet and DropoutNet models in terms of reducing overfitting. Zehong Zeng et al. came up with an ensemble framework that incorporates several techniques to prevent overfitting [7]. Min-Gu Kim et al. also proposed parallel ensemble networks to reduce overfitting in ECG data and prevent the degradation of generalization performance as the training progresses [1]. We implemented a method based on federated optimization preliminarily for facial expression recognition using FedNet [12]. This model achieves excellent results in terms of generalization on both CK+ and FER-2013 datasets. Overfitting remains a prevalent issue in supervised machine learning despite methods like Early Stopping, Network Reduction, Training Set Expansion, Regularization, and Dropout being effective [8]. D2C method tries to build an approach that focuses on achieving better generalization by using this method which can be implemented on top of other overfitting-reducing techniques.

III. Theoretical Framework

A. The Hypothesis

Overfitting happens when a model fits too well on the data that it trains on. It effectively captures even the random characteristics from the training data, randomness that would be insignificant in real-world applications. The presence of outliers and noisy data is a fundamental reason behind overfitting. In this study, we investigate a probable method to minimize the effect of these outliers and noisy data.

One way of minimizing the contribution of outliers/noise can be dividing the training data into multiple shards and training each shard separately. That way, each data point will only occur once in one of the several data shards. The representative samples would be close to each other in the feature space and would be present in each of the data shards more or less uniformly. However, the individual outliers/noise would only be able to impact one of the training processes. After training, some kind of averaging can be done to combine the results of all the models. Averaging would mean that the effect of the outliers/ noise would be reduced by a factor of N, where N is the number of data shards. Let us discuss the key factors behind our hypothesis of this method being able to address overfitting:

1) Weighted averaging of parameters helps in combining the knowledge learned by different subset models. However, extreme parameter updates driven by noise or outliers in individual models are moderated when averaging across multiple models. If a model encounters a noisy sample that deviates significantly from the overall data distribution, it might adjust its parameters excessively to fit that sample, leading to overfitting. However, if the training data is divided into subsets and trained separately, even if one model's parameters are influenced by noisy or outlier samples, their impact is diluted when combined with the parameter weights of other models during averaging. As a result, the averaged parameters should reflect a more balanced representation of the underlying data distribution, mitigating the influence of individual noisy or outlier samples.
2) Averaging the parameter weights post-training is particularly helpful in combating overfitting since it has a reg-

ularizing effect. Averaging parameter weights regularizes the model at the parameter level. It moderates extreme updates driven by noise or outliers in individual models, leading to more stable and generalizable parameter values. This helps prevent overfitting by discouraging models from fitting to the noise or idiosyncrasies present in their training subsets. By averaging parameter weights, the central model should converge to a shared solution that reflects the collective knowledge learned from different perspectives. This consensus learning approach encourages models to learn generalizable patterns and reduces reliance on individual model predictions that may be prone to overfitting.

3) Averaging parameter weights should smooth out the decision boundary learned by individual models. Extreme or noisy parameter updates that result in sharp or jagged decision boundaries in individual models are likely to be moderated and smoothed when combined through averaging, consequently helping in generalizing well to unseen data and reducing the risk of overfitting.

In summary, Divide2Conquer method helps mitigate the effects of noise and outliers. They can still affect the central model. However, the effect should be much less. Also, dividing the training set into too many subsets can result in very small data sizes in subset models, which may hamper the performance of these models to some extent even after averaging.

B. Mathematical Intuition

To formulate an underlying intuition behind our hypothesis, we must first define overfitting formally. We follow the groundwork laid out by [15]. Let us consider a Dataset D with training set T and test set R so that,

$$T \cup R = \emptyset$$

Let, T consist of datapoints $\{(x_i, y_i)\}_{i=1}^n$ and test set R consist the datapoints $\{(x_i, y_i)\}_{i=1}^m$. Also, let f be the function that maps each datapoint in the training set to its corresponding output label(true observations), i.e., $f(x_i) = f_i$, where f_i is the true corresponding output label for x_i. This model is unknown and we wish to train a model that replicates f as closely as possible. The problem is both the ground truth model and the true labels are unknown to us. Realistically, there is always some noise in the training dataset. We assume that the training output may be corrupted with some additive noise ϵ_i. So, we can say, $y_i = f_i + \epsilon_i$, where ϵ_i is some Gaussian noise, $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Since, the mean of the distribution is zero we can say that $\mathbb{E}(\epsilon_i) = 0$, hence also $Var(\epsilon_i) = 0$ and as the variance of a random variable x can be denoted as $Var(X) = \mathbb{E}(x^2) - \mathbb{E}(x)$ we have,

$$\mathbb{E}(\epsilon_i^2) = Var(\epsilon_i) + \mathbb{E}(\epsilon_i) \Rightarrow \mathbb{E}(\epsilon_i^2) = 0 + \sigma^2 = \sigma^2$$

Our assumption is that the input of the training dataset $\{x_i\}_{i=1}^n$ and their noise-added outputs $\{y_i\}_{i=1}^n$ are available to us, and our goal is to estimate the true model f by a model $\hat{f}$ in order to predict the labels $\{y_i\}_{i=1}^n$ for the input data $\{x_i\}_{i=1}^n$. Let the estimated observations be $\{\hat{y}_i\}_{i=1}^n$. It is preferable that $\{\hat{y}_i\}_{i=1}^n$ are close as possible to $\{y_i\}_{i=1}^n$. Mathematically, we can consider this as a minimization problem looking to minimize some error function such as MSE (Mean squared error). The MSE of estimated outputs from the trained model with respect to the dataset outputs are: $\textbf{MSE} = \frac{1}{n}\sum_{i=1}^n (\hat{y}_i - y_i)^2 = \mathbb{E}((\hat{y}_i - y_i)^2)$. Let us consider a single arbitrary datapoint (x_0, y_0) and the corresponding prediction of the trained model $\hat{f}$ for the input x_0 is $\hat{y}_0$. Ghojogh (2023) [15] showed that if the sample instance $(x_0, y_0) \notin T$ i.e., it belongs to the test set R, MSE of the predicted label can be expressed as -

$$\textbf{MSE} = \mathbb{E}((\hat{y}_0 - y_0)^2) = \mathbb{E}((\hat{y}_0 - f_0)^2) + \sigma^2 \tag{1}$$

Here we can observe from the first term of the R.H.S. of the equation that the MSE of the estimation of output labels with respect to the dataset labels can actually be represented by the MSE of the estimation with respect to the true uncorrupted labels $\{f_i\}_{i=1}^n$ plus some effect of the noise that exists in the dataset. Taking *Monte-Carlo approximation* [16] of the expectation terms in Eq.(1) we have:

$$\begin{aligned} \frac{1}{m}\sum_{i=1}^m (\hat{y}_i - y_i)^2 &= \frac{1}{m}\sum_{i=1}^m (\hat{y}_i - f_i)^2 + \sigma^2 \\ &= \sum_{i=1}^m (\hat{y}_i - f_i)^2 + m\sigma^2 \end{aligned} \tag{2}$$

Here the term $\sum_{i=1}^m (\hat{y}_i - y_i)^2$ is the total error between the model predictions and the dataset labels which we can call the *empirical observed error*, e. On the other hand, $\sum_{i=1}^m (\hat{y}_i - f_i)^2$ represents the total error between prediction labels and the true unknown labels, namely E. Hence, $e = E + m\sigma^2$. Thus, the observed error truly reflects the true error because the term $m\sigma^2$ remains constant. Which is the theoretical justification for evaluating model performance using unseen test data. But, if $(x_0, y_0) \in T$, deriving the M.S.E yields:

$$\textbf{MSE} = \mathbb{E}((\hat{y}_0 - y_0)^2) = \mathbb{E}((\hat{y}_0 - f_0)^2) + \sigma^2 - 2\sigma^2\,\mathbb{E}(\frac{\partial \hat{y}_0}{\partial y_0}) \tag{3}$$

Using the same approximation as Eq.(3) on the training set T we get:

$$\begin{aligned} &\frac{1}{n}\sum_{i=1}^n (\hat{y}_i - y_i)^2 = \frac{1}{n}\sum_{i=1}^n (\hat{y}_i - f_i)^2 + \sigma^2 + 2\sigma^2 \frac{1}{n}\sum_{i=1}^n \frac{\partial \hat{y}_i}{\partial y_i} \\ \implies &\sum_{i=1}^n (\hat{y}_i - y_i)^2 = \sum_{i=1}^n (\hat{y}_i - f_i)^2 + n\sigma^2 + 2\sigma^2 \sum_{i=1}^n \frac{\partial \hat{y}_i}{\partial y_i} \end{aligned} \tag{4}$$

From Eq.(4) we now understand while the model is training the observed empirical error calculated on the training dataset i.e, training error is not a clear representation of the true error because now $e = E + n\sigma^2 - 2\sigma^2 \sum_{i=1}^n \frac{\partial \hat{y}_0}{\partial y_0}$. Hence, even if we

are able to obtain a small value of e, the term $2\sigma^2 \sum_{i=1}^{n} \frac{\partial \hat{y}_0}{\partial y_0}$ can grow large as the training progresses and hide the true value of a substantial and large true error E (Figure 1). We can conclude from this analysis that the final term in Eq.(3) is a measure of the overfitting of the model. Which is also known as the complexity of a model. Upon a closer look at this final term, we can derive some more interesting insight. $\frac{\partial \hat{y}_0}{\partial y_0}$ is essentially the rate of change of the predicted label $\hat{y}_i$ for the input x_i with respect to the change in dataset label y_i. It makes sense that prediction labels would vary if the corresponding output in the dataset varies as the sample is considered from within the training set. It also reflects how dependent the model is on the training set, as the term grows when the predicted output varies too much with respect to changes in the training samples. This fits the classical definition of overfitting i.e., fitting too tightly to the training data and thus performing way worse when evaluated with actual test data. Evaluation on test data accurately represents the true error from Eq.(2).

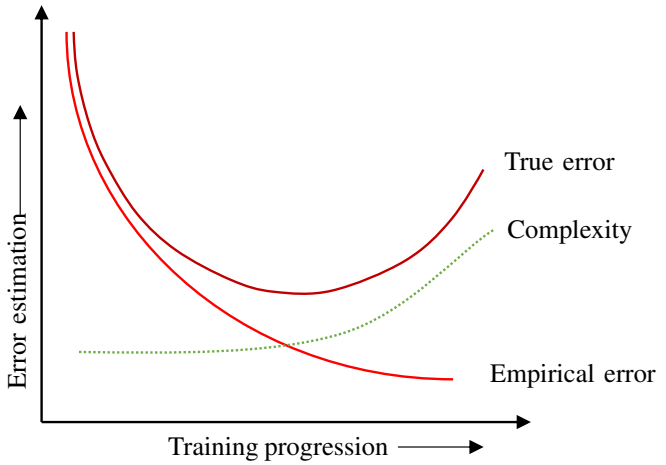

Fig. 1: When the corresponding complexity term grows and dominates the expression in Eq.(4) for empirical error, the model starts to overfit as empirical error becomes no longer a true representation of true error.

Statistical analysis shows that a model which overfits has high variance and low bias. This is because the model becomes overly complex accommodating for all the outliers in the training data and fails to generalize on unseen test data.

To see the impact of **D2C** we consider the training set T divided into k equal partitions, namely $|T_1| = |T_2| = ... = |T_k| = |T|/k$. Then, we train the model $\hat{f}_j$ using the j-th data chunk, where $j \in \{1, .., k\}$. As a result, we have k separately trained models. Finally, we aggregate these subset models into one central model $\hat{f}$. This process is repeated for p global epochs. For the sake of simplicity, we consider each $\hat{f}_j$ is a regression model with weight matrix $\boldsymbol{W_j} = \{w_{j1}, ..., w_{jn}\}$ and bias b_j. For an input vector $\boldsymbol{x} = \{x_1, ..., x_n\}$ the prediction of the subset model on the t-th epoch can be described as: $\hat{f}_j(\boldsymbol{x}) = \boldsymbol{W_j}^t \boldsymbol{x}^T + b_j$. In the case of a single global epoch, we now wish to aggregate the weight matrices creating the central weight matrix $\bar{\boldsymbol{W}} = \{\bar{w}_1, ..., \bar{w}_n\}$. Here the weight matrix for the $(t+1)$-th epoch is:

$$\bar{\boldsymbol{W}}^{t+1} = \frac{1}{k} \sum_{j=1}^{k} \boldsymbol{W_j}^t \text{ and } b^{t+1} = \frac{1}{k} \sum_{j=1}^{k} b_j^t \tag{5}$$

Hence, the output of the central model at $(t+1)$-th epoch:

$$\begin{aligned}
&\hat{f}(\boldsymbol{x})^{t+1} = \bar{\boldsymbol{W}}^{t+1} \boldsymbol{x}^T + b_j \\
&\implies \hat{f}(\boldsymbol{x})^{t+1} = \frac{1}{k} \sum_{j=1}^{k} \boldsymbol{W_j}^t \boldsymbol{x}^T + \frac{1}{k} \sum_{j=1}^{k} b_j^t \\
&\implies \hat{f}(\boldsymbol{x})^{t+1} = \frac{1}{k} \sum_{j=1}^{k} \left(\boldsymbol{W_j}^t \boldsymbol{x}^T + b_j^t \right) \\
&\implies \hat{f}(\boldsymbol{x})^{t+1} = \frac{1}{k} \sum_{j=1}^{k} \hat{f}(\boldsymbol{x})_j^t
\end{aligned} \tag{6}$$

Let $e_{j,t}$ denote the empirical error of the j-th model while estimating some arbitrary instance on the t-th global epoch. We assume this error to be a random variable with normal distribution having mean zero, i.e., $e_{jt} \sim \mathcal{N}(0, s)$. Hence, $\mathbb{E}(e_j t) = 0$ and $Var(e_j t) = s$. Let us denote the estimation of the product of two trained models using two different data chunks by c, i.e, $\mathbb{E}(e_{jt} \times e_{lt}) = c$. So, we have:

$$\mathbb{E}(e_{jt}^2) = Var(e_{jt}) + \mathbb{E}(e_{jt}) \Rightarrow \mathbb{E}(e_{jt}^2) = 0 + s = s$$

Again for $j, l \in \{1, ..., k\}$ where $j \neq l$ we have:

$$\begin{aligned}
&Cov(e_{jt}, e_{lt}) = \mathbb{E}(e_{jt} \times e_{lt}) - \mathbb{E}(e_{jt})\, \mathbb{E}(e_{lt}) \\
&\implies Cov(e_{jt}, e_{lt}) = c - 0 \times 0 = c \\
&\implies Cov(\hat{f}(\boldsymbol{x})_j^t, \hat{f}(\boldsymbol{x})_l^t) = c - 0 \times 0 = c
\end{aligned} \tag{7}$$

From Eq.(6) and Eq.(7) we can deduce:

$$\begin{aligned}
Var(\hat{f}(\boldsymbol{x})^{t+1}) &= \frac{1}{k^2} Var\left(\sum_{j=1}^{k} \hat{f}(\boldsymbol{x})_j^t \right) \\
&= \frac{1}{k^2} \sum_{j=1}^{k} Var\left(\hat{f}(\boldsymbol{x})_j^t \right) + \frac{1}{k^2} \sum_{i=1, j=1}^{k} \\
&\sum_{i \neq l}^{k} Cov\left(\hat{f}(\boldsymbol{x})_j^t, \hat{f}(\boldsymbol{x})_l^t \right) \\
&= \frac{1}{k^2} sk + \frac{1}{k^2} ck(k-1) = \frac{s + c(k-1)}{k}
\end{aligned} \tag{8}$$

Eq.(8) provides us with an interesting insight. If the two subset models are highly correlated i.e, their predictions and empirical error are very close to each other we can assume $s \approx c$. Hence, the variance of the central model at $(t+1)$-th epoch can be reduced to: $Var(\hat{f}(\boldsymbol{x})^{t+1}) \approx (s + s(k-1))/k \approx s$. On the other hand, if the two models are very dissimilar the same expression gives us: $Var(\hat{f}(\boldsymbol{x})^{t+1}) \approx (s + 0(k-1))/k \approx s/k$. As $Var(e_j t) = Var(\hat{f}(\boldsymbol{x})^t) = s$, each of the subset models at epoch t has a variance of prediction s, which means if the subset models at epoch t are completely different, the variation of estimate decreases by a factor of k on epoch $(t+1)$.

Consequently, if the subset models are identically trained the variance remains almost unchanged. This result is similar to the one proposed by Breiman,1996 [17] and Bühlmann et al.,2000 [18] through the idea of *'Bagging'*, which is a meta algorithm aggregating multiple model outputs. The analysis clearly shows that our approach does not at the least worsen the problem of overfitting in this scenario but rather improves the central model on each epoch if certain conditions are met. However, in reality, the subset models are neither completely different nor exactly the same due to the relative distribution of the data subsets and their shared model architecture. So, the degree to which D2C ameliorates overfitting depends on the model architecture and the correlation among the subset models. Our results show repeating the process in every central epoch tends to reduce overfitting in most cases. A limitation of this analysis is that deep neural networks are mostly black boxes and their architectures vary tremendously, which makes establishing a generalized mathematical representation of them extremely difficult. Hence, our assumption of a simple regression model is not an exact reflection of the actual implementation. Despite that, it describes an ideal and simple scenario where the proposed method should improve the model. Moreover, it also gave us the insight that this approach can be used in conjunction with other overfitting reduction techniques such as *Dropouts*(Srivastava et al., 2014) [4]. If dropouts are stacked on top of our method in every iteration of local training, the neurons are randomly removed with some probability p(usually $p = 0.5$). This introduction of randomness ensures that subset models are different from each other increasing the chances of improvement when the models are aggregated.

IV. Proposed Methodology

A. Creating Training Subsets

To train multiple Neural Networks parallelly using different data, we need to divide the whole training set into multiple shards. At first, random shuffling is applied to the training set. After that, it is divided into multiple subsets. This is done while keeping the class distribution constant. Now each of these Training Subsets is fed into a subset model. All the subset models must have precisely the same neural network architecture, hence the same number of trainable parameters. Otherwise, averaging of the parameters wouldn't be possible.

B. Training & Averaging of Trained Parameters

Instead of one training loop, the implementation of our method requires two loops. The subset training loop is nested within the global iteration for averaging. The process is started by creating a central model with the same architecture as the subset models. Then the weights of central and subset models are initialized. Inside the inner loop, the subset training is repeated iteratively for each training subset. Then after a particular number of local epochs, the parameters from each of the subset models are scaled and then added to the local weight list to send to the outer loop where they are averaged. All the scaled local training weights are added up in the outer loop, and weighted aggregation and averaging are performed for each of the parameters. Finally, the central model is modified, and the parameters are updated to these averaged aggregates. At this stage, the initialization phase weights of each subset model are set to the central model's current parameters. An entire global training epoch thus concludes. The next step is to retrain each subset model on the respective training subset. This whole process is then repeated for several global epochs. The averaging process here is simply the execution of weighted parameter averaging. The weights are measured based on the fraction of data instances belonging to each training subset. The weighted averaging formula we're utilizing is given below.

$$f(\omega) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(\omega) \text{ where, } F_k(\omega) = \frac{1}{n} \sum_{i \in Pk}^{i} (f_i(\omega))$$

The whole process is summarized in Figure 2.

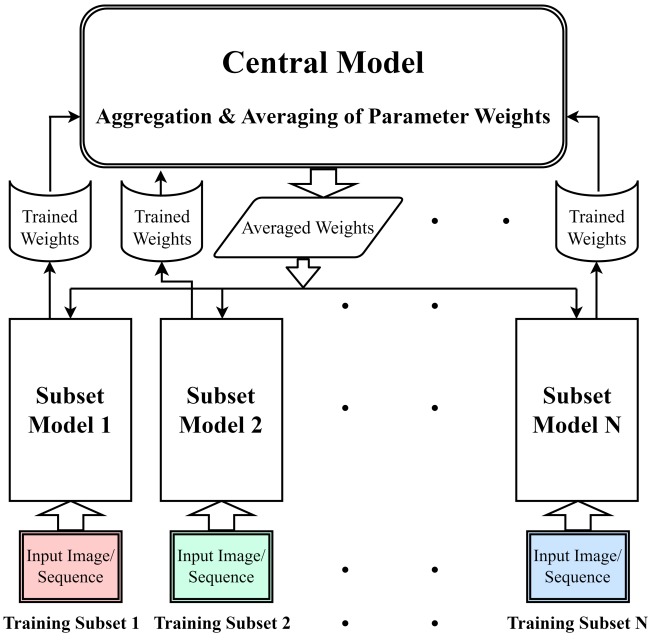

Fig. 2: Divide2Conquer Method at a Glance – Training on each subset, averaging the trained parameters, setting the Central model parameters to these values, and sharing the averaged parameters back to the Subset models to keep the loop running.

C. Hyperparameter Tuning

Tuning the overall model consists of two stages. Firstly, the subset model and its hyperparameters should be tuned using a subset/entire training set and the validation set to find a suitable model to train each training subset. Next, that base model is utilized to tune the Central model. The new **global hyperparameters** that need tuning, in this case, are the **Number of Subsets of the Training Set** and the **Number of Epochs before each round of Global Averaging**. Testing for different values for these hyperparameters, the implementation and the evaluation of our method are done using the best

combination. The appropriate number of training subsets can be determined first by varying the number of training subsets and observing the performance metrics like test/validation accuracy, F1 score, log loss, and also the validation loss curve. The appropriate number of training epochs in each subset before the averaging is done each time is likely to be a small number $(1 \sim 2)$ since the subset model can quickly overfit the training subset with a reduced size. So, it makes more sense to set a small number of epochs initially and tune the number of training subsets. Once the number of subsets is determined, the same process can be repeated by varying the number of training epochs and determining the appropriate value of it.

V. Experimental Setup

A. Datasets

For image classification, we used the FER-2013 [11] dataset. FER-2013 is a large dataset containing more than 35000 training images. It consists of grayscale images depicting facial expressions of seven basic emotions: anger, disgust, fear, happiness, sadness, surprise, and neutral. The dataset's diversity and quality vary due to its internet-based collection method, which may introduce noise and variability in the images. That is why this dataset is particularly important in our study, which aims to minimize overfitting caused by the mentioned issues and improve the generalization performance

For text classification, we used another benchmark dataset, AG NEWS [14]. It is a widely used benchmark dataset for text classification tasks. It consists of news articles collected from the AG's corpus news collection, covering four major categories: World, Sports, Business, and Science/Technology. The dataset contains approximately 127,600 news articles. This dataset is perfectly balanced in terms of class distribution. Its large size, balanced distribution of categories, and relevance to real-world news content make it the perfect candidate for evaluating the generalization ability of models employing our proposed method. This dataset is crucial to testing our method in Big Data scenarios.

B. Experimental Details

- **Image Classification:** At first, all the images are reshaped and normalized as part of Data Preprocessing. We applied one-hot encoding to the labels to use cross-entropy loss later. We split the training data into training and validation sets using Scikit-Learn and applied a 90-10 split. We then divided the training set into multiple subsets. Before constructing the model, processing each training subset into tensors, and batching them was our last step. Each of our subset models was comprised of 4 Convolutional (32 to 256 kernels progressively) and MaxPooling layer (2X2) blocks followed by one fully connected hidden layer (128 neurons). Each block contained two identical convolutional and one pooling layer. We used the Adam optimizer, and the learning rate was 0.001.
- **Text Classification:** In this case, we created a corpus using all the words that occur in our entire training dataset. Then we made a word index and word embedding for all the words in the corpus. After that, we created word sequences using that word embedding for each data instance. We applied padding to make each training data point equal in length. In the case of these 1D text sequences, the model we used consisted of 3 bidirectional LSTM layers(128, 64, and 32 units) and two dense layers(128 and 64 neurons) followed by the output layer. It took the embeddings as input, and we used the 100-dimensional version of GloVe from Stanford for these embeddings. We used the Adam optimizer, and the learning rate was 0.001.

For our experiments, in both cases, we used a 90-10 Training – Validation split and used the readily available test sets provided with the datasets. We used dropouts and batch normalization between the layers in all these models. This is also important because as we mentioned earlier, our proposed method can be used on top of other methods that address overfitting.

VI. Results and Discussions

As discussed in the previous section, we will use loss and accuracy curves, Test Accuracy/F1 Score, and log loss to analyze the results and evaluate our method. We also varied the two new global hyperparameters we talked about in the previous section: the **Number of Subsets of the Training Set** and the **Number of Epochs before each round of Global Averaging**. To refer to them concisely, we will use the variables N and E respectively. This means, in the tables, figures, and discussions of this section, N will refer to the Number of Training Subsets and E will refer to the Number of Epochs before each round of Global Averaging.

A. Visualizing Decision Boundary

At first, we will try to visualize the change in decision boundaries due to applying D2C method. As we discussed earlier, in the case of overfitting, the decision boundary often becomes overly complex and jagged. Applying a method that addresses overfitting should result in a smoother decision boundary. To test our method, we used a simple binary classification dataset. This data set was synthetic and created using the Scikit-Learn library. Then the 240-sample large dataset was divided into a training and a test set using a 75-25 split. The neural network architecture we used for these experiments comprised of three hidden layers having 100 neurons each. We didn't use any other method that treats overfitting in this case. This helped us understand the difference made by our proposed method alone. To begin with, we followed the traditional approach and fed the entire training dataset to the model. After training the model for a certain number of epochs, we got the decision boundary shown in Figure 3:

As we can see from Figure 3, the decision boundary in this plot is highly complex and non-linear. The model creates intricate regions to distinguish between the two classes. The boundary closely follows individual points, resulting in a jagged, convoluted shape. The complexity of the decision

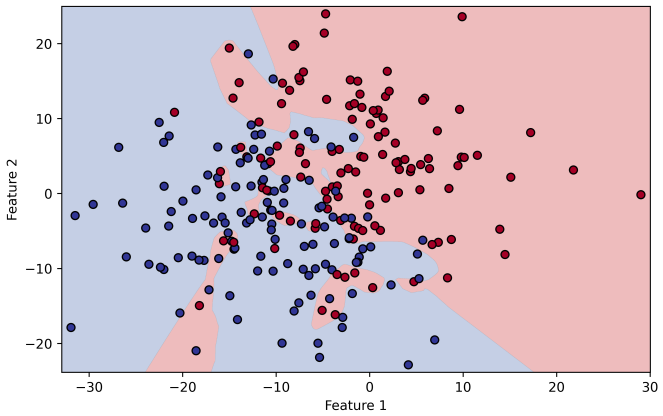

Fig. 3: Decision Boundary using the traditional approach.

boundary suggests that the model is overfitting to the training data. It is trying too hard to separate every point, including noise, rather than focusing on general patterns in the data. Evidence of overfitting is seen in how the model creates small pockets of blue in the red region and vice versa. This approach is likely to produce very good results on training data, but it is likely to struggle with new, unseen data due to its sensitivity to specific details in the training set. The highly irregular boundary reflects poor generalization ability. The accuracies we got using the training set and the test set also depict the same picture. The training accuracy using this traditional approach was 99.44%, but the test accuracy was just 68.33%, showing significant overfitting.

Applying Divide2Conquer method meant we had to divide our training set into multiple subsets. Then each of the subsets was trained parallelly and after a few epochs, the trained weights from each subset were averaged to get the final weights. In this case, we divided the training set into 3 subsets and got the decision boundary shown in Figure 4.

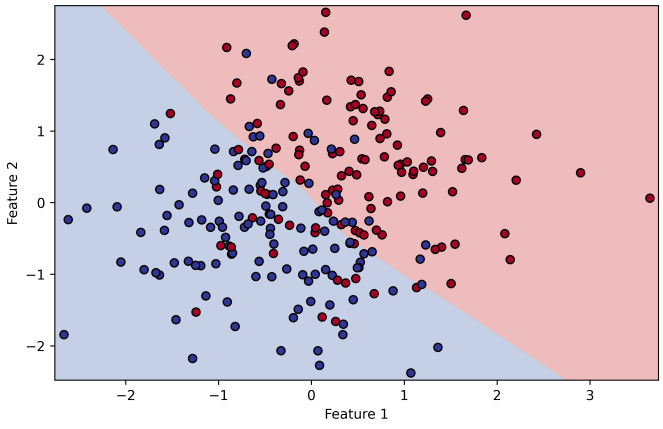

Fig. 4: Decision Boundary using D2C method for 3 subsets.

As we can see in Figure 4, the decision boundary is much simpler, dividing the feature space almost diagonally. This suggests a less flexible, more generalized model, due to averaging the weights from multiple subsets. These boundaries are smooth and do not follow the exact positions of individual data points as closely. The models are more generalized, focusing on the overall trend in the data rather than trying to accommodate every individual point, exhibiting much less overfitting compared to what we saw in the traditional approach using the same model architecture. By averaging the weights from different subsets, the model smooths out the noise in the data and prevents overfitting to specific training samples. This clear improvement in generalization ability stems from the fact that each of the noisy samples in the dataset occurs only once in one of the subsets. So, irregularity around the individual noisy sample does not affect all the subsets, but rather only one of them. Hence, after averaging the weights, the impact of the individual outlier is minimized. We can see further proof of better generalization by comparing the training and test accuracies for each model. These accuracies are summarized in Table I provided below. However, perhaps the most important characteristic of D2C method is that it can be used on top of other counter-overfitting measures. For example, we can use dropouts in the local architecture and also divide the training set into multiple subsets. In the following subsections, we will investigate whether using this method on top of other techniques offers any added advantage or not.

TABLE I: Performance of the traditional approach vs different values of N Using D2C method.

Model Settings	*Training Accuracy*	*Test Accuracy*
No Subsets	**0.9944**	0.6833
N=2 (2 Subsets)	0.7611	0.75
N=3 (3 Subsets)	0.7944	**0.8333**
N=4 (4 Subsets)	0.7889	**0.8333**

B. Analyzing Loss and Accuracy Curves

The loss curve is a definitive indicator of overfitting. If the validation loss curve follows the training loss curve closely, it usually means good generalization. Even if the loss curve shows an upward trend, if Divide2Conquer method manages to slow down the rate at which the loss increases, it necessarily indicates a reduction in overfitting. Let us examine the loss curves generated in the two domains we have experimented on.

Image Classification: The FER2013 dataset represents a particularly overfitting-prone task in Facial Expression Recognition, hence crucial in our study. Let us first take a look at the loss curves generated using the FER2013 dataset. First, the number of epochs before each round of global averaging, E was kept constant at 1 and the number of subsets, N was varied between 1 and 7. Later we also analyzed the impact of changing the number of epochs, E, by varying it keeping N constant, and observing the loss and accuracy curves. To compare the curves effectively, in each case, we trained our model in such a way that the total number of global epochs equals the number of global epochs for E=1 divided by E. That way, the total number of local epochs during training

was kept constant, as each of the global epochs represented E number of local epochs in the subsets.

The loss and accuracy curves from Figure 5 and Figure 6 show proof of what we inferred before. The validation loss starts increasing quite abruptly after a few epochs in the case of the implementation based on the traditional approach. The overfitting in this case is very much apparent. The rate of increase in validation loss as the training progresses goes down very rapidly as we apply D2C method and increase the number of subsets. The validation accuracy, however, doesn't come down as the loss increases (hence early stopping is not an option). Rather counterintuitively, the validation accuracy maintains an increasing trend almost throughout the 100 epochs. We also notice a slight improvement in the peak stable accuracy with our proposed method over the traditional approach. So, dividing the training set into multiple subsets improves both validation accuracy and loss in this case.

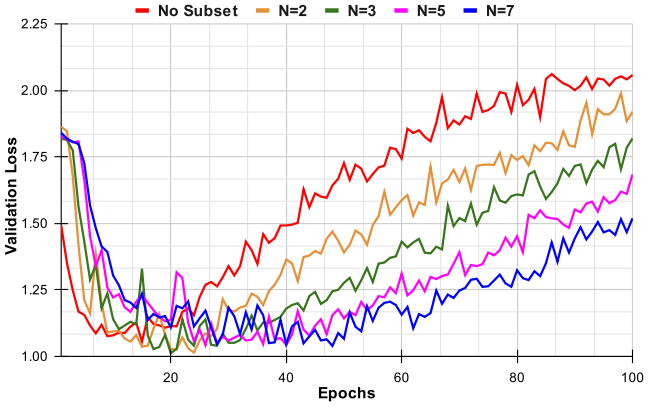

Fig. 5: Validation loss curves for the traditional & D2C method for different numbers of subsets, N, on FER2013.

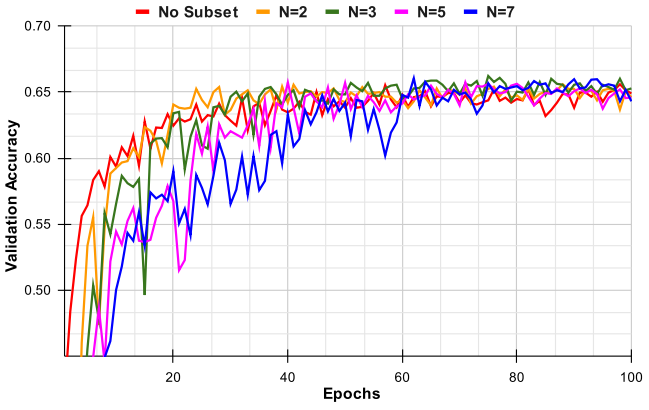

Fig. 6: Validation accuracy curves for the traditional & D2C method for different numbers of subsets, N, on FER2013.

We can also see from Figure 7 that when we vary the number of epochs before each round of global averaging, E, there is no clear indication of any kind of change in the trend or the rate of change in the validation losses as the training

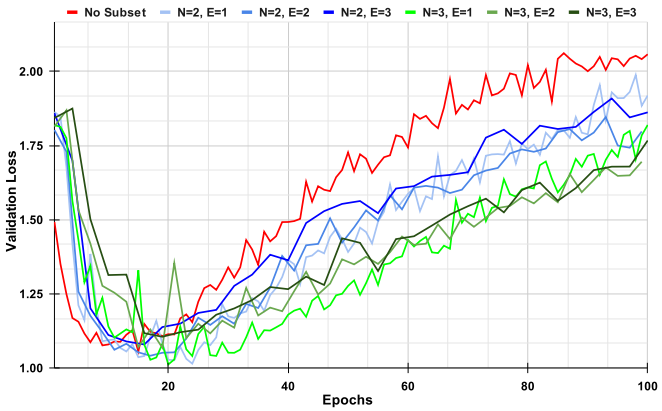

Fig. 7: Validation loss curves for different numbers of epochs before each round of global averaging, E, while keeping the number of subsets, N = 2 & 3.

progresses. Here, for both N=2 (shades of blue) and N=3 (shades of green), the validation curves are quite similar for different values of E throughout the training phase. Hence, we can conclude that varying the number of epochs before each round of global averaging, E does not have any obvious implications on log loss as we saw in the case of varying the number of subsets, N. However, an increased number of local epochs before each round of global averaging means more scope for learning from each of the subsets before averaging, sometimes even overfitting on a particular subset.

Text Classification: We used AG News, a benchmark dataset to test our method primarily in the NLP domain. Its large size and balanced distribution mean this is a perfect candidate for applying our proposed method of dividing the training set into multiple subsets in the Big Data scenario. Let us observe the validation loss and accuracy curves generated using the AG News dataset. The loss and accuracy curves

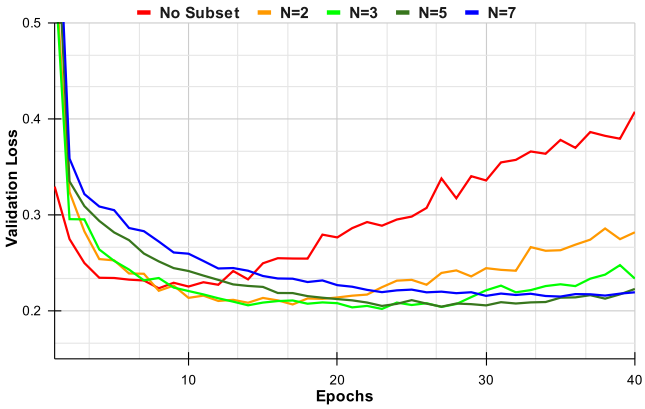

Fig. 8: Validation loss curves for the traditional method & D2C method for different numbers of subsets, N, on AG News.

generated for AG News depict the clear improvements brought by the proposed method over the traditional approach. The loss curve from Figure 8 shows the trend we observed before,

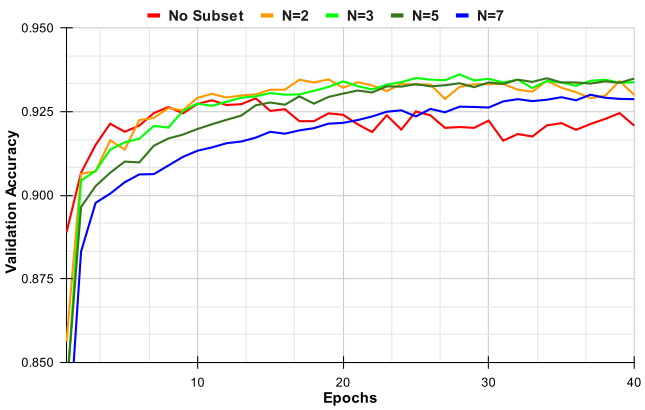

Fig. 9: Validation accuracy curves for the traditional & D2C method for different numbers of subsets, N, on AG News.

which is that dividing the training data into more & more subsets delays and minimizes the increase in validation loss as the training progresses. The validation loss keeps decreasing with each epoch for longer when using D2C method on the AG News dataset but starts to increase after very few epochs again in the case of the traditional approach. The convergence though, slows down a bit in this case too. More importantly, with no subsets, the validation loss shoots up sharply after around 8 epochs, which indicates severe overfitting and impacts the accuracy too as the training progresses as we can see from Figure 9. However, when we divide the training set into multiple subsets, the sharp increase in the validation loss is mitigated drastically. Even with only two subsets, the improvement is quite significant. When the number of subsets is increased further to more than two, the loss curve becomes quite stable. From the accuracy curve from Figure 9, we can observe that the accuracy crosses the 93% mark and remains stable when the number of subsets is 2,3 or 5. However, the validation accuracy for the traditional approach increases till the 14th epoch and then it starts to decrease again. This deterioration in accuracy can be linked to the fact that the loss kept increasing drastically and at some point, it started hurting the accuracy too, exhibiting overfitting.

The experiments on FER2013 and AG News prove that dividing the training set into multiple subsets can indeed address overfitting and enhance the performance of deep learning models. So, in both cases, Divide2Conquer method improves both accuracy and loss compared to the traditional approach, which can be accredited to the overfitting resistant nature of models using our method. This also shows that dividing the training set into multiple subsets works in multiple domains and a diverse set of tasks.

C. Evaluating Accuracy, F1 Score, ROC_AUC & Log Loss

Let us finally take a look at the final Accuracy, F1 Score, and Log Loss values that we got from the predefined test sets of the datasets we used. the several experiments we performed on the two datasets. For these values, we used the average from our multiple repetitive runs (to test for repeatability) for each case. Each of the experiments was repeated at least 3 to 5 times.

Image Classification: Table II summarizes the findings from the predefined test set of the FER2013 based on our proposed method.

TABLE II: Performance Metrics with Traditional Approach vs. Different Settings Using D2C Method on FER2013.

Model Settings	*Accuracy*	*F1 Score*	*Log Loss*	*ROC_AUC*
No Subsets	0.6498	0.6420	2.0112	0.89934
N=2, E=1	0.6511	0.6394	1.9101	0.89875
N=2, E=2	**0.6567**	0.6444	1.8450	**0.9032**
N=2, E=3	0.6490	0.6389	1.8718	0.8975
N=3, E=1	0.6491	0.6356	1.6540	0.90104
N=3, E=2	0.6538	**0.6465**	1.6990	0.89870
N=3, E=3	0.6539	0.6428	1.7315	0.9002
N=5, E=1	0.6500	0.6368	1.5157	0.90153
N=7, E=1	0.6521	0.6432	**1.4821**	0.90148

We can observe from Table II that the best accuracy and ROC_AUC were achieved using Divide2Conquer method. The number of subsets, N, was only **2** in this case for the best model. The number of epochs before each round of global averaging, E, was **2**. D2C approach showed a significant improvement with an accuracy of **65.67%** over the traditional approach with no subsets, which yielded an accuracy of **64.98%**. Also, as we can see from Table II, there is a trend of decreasing log loss as we increase the number of subsets. However, no such trend is visible consistently when we increase the number of epochs, E. The decrease in Log Loss is important since a lower log loss indicates that the model's predicted probabilities align better with the true class labels. This means the model is more confident and accurate in its predictions, even if the accuracies are similar. The best model in terms of F1 score, however, is the model with **3** subsets and **2** epochs before each round of global averaging. It means that this model performs better when we consider the classes individually. This is important since the class distribution of FER2013 is not balanced. Our proposed method showed a significant improvement in terms of F1 score too, with an F1 score of **0.6465**, while the traditional approach with no subsets yielded an F1 score of **0.642**. This improvement in the accuracy and F1 score clearly shows the fact that D2C method addresses Overfitting better than the traditional approach on this dataset and hence performs better.

Text Classification: Table III summarizes the findings from the predefined test set of the AG News based on our proposed method.

We can clearly observe from Table III that the best accuracy and F1 Score were achieved using our proposed Divide2Conquer method. The number of subsets, N, was **5** in this case for the best model. The number of epochs before each round of global averaging, E, was **2**. Our proposed method showed a significant improvement with an accuracy of **92.95%** over the traditional approach with no subsets, which yielded an accuracy of **92.43%**. Using the Divide2Conquer method

TABLE III: Performance Metrics with Traditional Approach vs. Different Settings Using D2C Method on AG News

Model Settings	*Accuracy*	*F1 Score*	*Log Loss*	*ROC_AUC*
No Subsets	0.9243	0.9243	0.2611	0.98764
N=2, E=1	0.9291	0.9291	0.2556	0.98873
N=3, E=1	0.9288	0.9288	0.2520	0.98872
N=3, E=2	0.9275	0.9275	0.2567	0.98852
N=5, E=1	0.9293	0.9292	0.2412	0.98881
N=5, E=2	**0.9295**	**0.9296**	0.2451	0.98875
N=7, E=1	0.9271	0.9271	**0.2353**	**0.98916**

decreased the error by more than **0.5%** in this case. Also, the trend of decreasing log loss as we increase the number of subsets is visible here too. The best method in terms of ROC_AUC also happens to be the model with the highest number of subsets (**7**). It also seems that the log loss increases a bit when we increase the number of epochs, E.

The effect of using D2C method is more apparent in this dataset compared to the previous one. The accuracy improved significantly compared to the traditional approach in every variation of our proposed method (for all values of N and E). AG News is the largest dataset we've used for our experiments. For each class, we had 30000 samples in this dataset. So, despite dividing the dataset into more and more subsets, each subset had enough data to train on and at the same time take advantage of the regularizing effect that dividing the training set introduces. As a result, the performance of the models improved significantly, addressing the issue of overfitting that usually is prevalent in text classification tasks. The Divide2Conquer method can exploit the large datasets very well in this way, which traditional models are unable to do. This finding outlines the promising signs D2C method shows when it comes to Big Data scenarios.

VII. Conclusion

Our study demonstrates that dividing training data into multiple subsets and averaging the weights iteratively can treat overfitting and produce significant performance gains through stability and better generalization. In this paper, we applied our proposed Divide2Conquer method and tested it on multiple datasets from different domains. This method outperformed the traditional approach in all the datasets in terms of key metrics and showed significantly better resistance against overfitting. The D2C method also showed a consistent ability to mitigate the increasing trend of validation loss, and to produce smoother decision boundaries. These results prove our hypothesis of this method being able to help generalize better and resist overfitting to be true. Moreover, the Divide2Conquer method brings even more improvement when the dataset is larger, outlining its promise in Big Data scenarios, especially since D2C method can be applied on top of other generalization techniques. Thus, the proposed D2C method opens the door for future research and experiments using this in several domains.

References

[1] Kim, M., Choi, C. & Pan, S. Ensemble networks for user recognition in various situations based on electrocardiogram. *IEEE Access*. **8** pp. 36527-36535 (2020)

[2] Konečnỳ, J., McMahan, B. & Ramage, D. Federated optimization: Distributed optimization beyond the datacenter. *ArXiv Preprint ArXiv:1511.03575*. (2015)

[3] Shanmugavel, A., Ellappan, V., Mahendran, A., Subramanian, M., Lakshmanan, R. & Mazzara, M. A novel ensemble based reduced overfitting model with convolutional neural network for traffic sign recognition system. *Electronics*. **12**, 926 (2023)

[4] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The Journal Of Machine Learning Research*. **15**, 1929-1958 (2014)

[5] Cogswell, M., Ahmed, F., Girshick, R., Zitnick, L. & Batra, D. Reducing overfitting in deep networks by decorrelating representations. *ArXiv Preprint ArXiv:1511.06068*. (2015)

[6] Ioffe, S. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv Preprint ArXiv:1502.03167*. (2015)

[7] Zeng, Z., Liu, Y., Lu, X., Zhang, Y. & Lu, X. An Ensemble Framework Based on Fine Multi-Window Feature Engineering and Overfitting Prevention for Transportation Mode Recognition. *Adjunct Proceedings Of The 2023 ACM International Joint Conference On Pervasive And Ubiquitous Computing & The 2023 ACM International Symposium On Wearable Computing*. pp. 563-568 (2023)

[8] Ying, X. An overview of overfitting and its solutions. *Journal Of Physics: Conference Series*. **1168** pp. 022022 (2019)

[9] Korjus, K., Hebart, M. & Vicente, R. An efficient data partitioning to improve classification performance while keeping parameters interpretable. *PloS One*. **11**, e0161788 (2016)

[10] Kolluri, J., Kotte, V., Phridviraj, M. & Razia, S. Reducing overfitting problem in machine learning using novel L1/4 regularization method. *2020 4th International Conference On Trends In Electronics And Informatics (ICOEI)(48184)*. pp. 934-938 (2020)

[11] Zahara, L., Musa, P., Wibowo, E., Karim, I. & Musa, S. The facial emotion recognition (FER-2013) dataset for prediction system of micro-expressions face using the convolutional neural network (CNN) algorithm based Raspberry Pi. *2020 Fifth International Conference On Informatics And Computing (ICIC)*. pp. 1-9 (2020)

[12] Siddiqui, M., Shusmita, S., Sabreen, S. & Alam, M. FedNet: Federated Implementation of Neural Networks for Facial Expression Recognition. *2022 International Conference On Decision Aid Sciences And Applications (DASA)*. pp. 82-87 (2022)

[13] Li, X., Chen, S., Hu, X. & Yang, J. Understanding the disharmony between dropout and batch normalization by variance shift. *Proceedings Of The IEEE/CVF Conference On Computer Vision And Pattern Recognition*. pp. 2682-2690 (2019)

[14] Zhang, X., Zhao, J. & LeCun, Y. Character-level convolutional networks for text classification. *Advances In Neural Information Processing Systems*. **28** (2015)

[15] Ghojogh, B. & Crowley, M. The theory behind overfitting, cross validation, regularization, bagging, and boosting: tutorial. *ArXiv Preprint ArXiv:1905.12787*. (2019)

[16] Ghojogh, B., Nekoei, H., Ghojogh, A., Karray, F. & Crowley, M. Sampling algorithms, from survey sampling to Monte Carlo methods: Tutorial and literature review. *ArXiv Preprint ArXiv:2011.00901*. (2020)

[17] Breiman, L. Bagging predictors. *Machine Learning*. **24** pp. 123-140 (1996)

[18] Bühlmann, P. & Yu, B. Analyzing bagging. *The Annals Of Statistics*. **30**, 927-961 (2002)

[19] Morgan, N. & Bourlard, H. Generalization and parameter estimation in feedforward nets: Some experiments. *Advances In Neural Information Processing Systems*. **2** (1989)

[20] Prechelt, L. Early stopping-but when?. *Neural Networks: Tricks Of The Trade*. pp. 55-69 (2002)

[21] Fürnkranz, J. Pruning algorithms for rule learning. *Machine Learning*. **27** pp. 139-172 (1997)

[22] Sun, Y., Wang, X. & Tang, X. Deep learning face representation from predicting 10,000 classes. *Proceedings Of The IEEE Conference On Computer Vision And Pattern Recognition*. pp. 1891-1898 (2014)