AdaBoost-CNN: An Adaptive Boosting algorithm for Convolutional Neural Networks to classify Multi-Class Imbalanced datasets using Transfer Learning

Aboozar Taherkhani[a,*], Georgina Cosma[b], T. M McGinnity[c,d]

[a]*School of Computer Science and Informatics, De Montfort University, Leicester, UK*
[b]*Department of Computer Science, School of Science, Loughborough University, Loughborough, UK*
[c]*School of Science and Technology, Nottingham Trent University, Nottingham, UK*
[d]*Intelligent Systems Research Centre, Ulster University, Northern Ireland, Derry, UK*

Abstract

Ensemble models achieve high accuracy by combining a number of base estimators and can increase the reliability of machine learning compared to a single estimator. Additionally, an ensemble model enables a machine learning method to deal with imbalanced data, which is considered to be one of the most challenging problems in machine learning. In this paper, the capability of Adaptive Boosting (AdaBoost) is integrated with a Convolutional Neural Network (CNN) to design a new machine learning method, AdaBoost-CNN, which can deal with large imbalanced datasets with high accuracy. AdaBoost is an ensemble method where a sequence of classifiers is trained. In AdaBoost, each training sample is assigned a weight, and a higher weight is set for a training sample that has not been trained by the previous classifier. The proposed AdaBoost-CNN is designed to reduce the computational cost of the classical AdaBoost when dealing with large sets of training data, through reducing the required number of learning epochs for its ingredient estimator. AdaBoost-CNN applies transfer learning to sequentially transfer the trained knowledge of a CNN estimator to the next CNN estimator, while updating the weights of the samples in the training set to improve accuracy and to reduce training time. Experimental results revealed that the proposed AdaBoost-CNN achieved 16.98% higher accuracy compared to the classical AdaBoost method on a synthetic imbalanced dataset. Additionally, AdaBoost-CNN reached an accuracy of 94.08% on 10,000 testing samples of the synthetic imbalanced dataset, which is higher than the accuracy of the baseline CNN method, i.e. 92.05%. AdaBoost-CNN is computationally efficient, as evidenced by the fact that the training simulation time of the proposed method is 47.33 seconds, which is lower than the training simulation time required for a similar AdaBoost method without transfer learning, i.e. 225.83 seconds on the imbalanced dataset. Moreover, when compared to the baseline CNN, AdaBoost-CNN achieved higher accuracy when applied to five other benchmark datasets including CIFAR-10 and Fashion-MNIST. AdaBoost-CNN was also applied to the EMNIST datasets, to determine its impact on large imbalanced classes, and the results demonstrate the superiority of the proposed method compared to CNN.

Key words: Deep Learning, Ensemble Models, AdaBoost, Imbalanced Data, Transfer Learning.

1. Introduction

Ensemble approaches have shown their exceptional capabilities in improving the accuracy of classical machine learning approaches such as Support Vector Machines (SVM) [1] and decision trees [2][3], and are often used to overcome the difficulty of training imbalanced data [1]. An ensemble method combines a number of weak classifiers to generate a machine learning method that is better than its

* Corresponding Author (aboozar.taherkhani@dmu.ac.uk)

ingredient simple classifiers. Class-imbalance is considered as one of the most challenging problems in machine learning, and it occurs in a learning task where there are considerably less data instances in one class (the minority class) compared to the other class (the majority class). A balanced dataset consists of approximately the same number of training samples in each class. Boosting ensemble approaches are effective methods to overcome the challenges encountered by machine learning algorithms when learning from data with imbalanced classes [1] [4]. AdaBoost is an ensemble approach which can identify misclassified instances that occur because of the disjunct problem. The disjunct problem is apparent in datasets which contain instances in a class that are clustered in a number of separate small groups, and each group contains a small number of instances that cannot be disregarded and should be trained [1]. Although class imbalance has been comprehensively studied in classical machine learning methods, it has received less attention in the context of deep learning [5].

Different ensemble methods have been used in various applications. Viola and Jones [6] have designed a face detector that works with a large pool of simple classifiers and creates a strong classifier for human face detection. Galar et al. [3] have proposed tree-based ensemble approaches for the task of classifying imbalanced data. They have used bagging- and boosting- based ensemble methods to classify imbalanced data. Nejatian et al. [7] have proposed sub-sampling and ensemble clustering techniques for learning tasks in which the number of samples in a minority class is less than the number of samples in a majority class, and they applied these techniques for breast cancer detection. AdaBoost is a well-known algorithm and the seminal work of Freund and Schapire [8] has attracted significant attention from researchers. Lee et al. [1] have used SVM as a weak classifier for AdaBoost to propose a method to deal with imbalanced data. Lee et al.'s [1] approach uses an SVM margin to define different types of instances, such as borderline instances, and the AdaBoost assigns different instances with different scores, to achieve higher accuracy on imbalanced data. The AdaBoost method proposed in [1] is used for binary classification tasks.

Standard learning algorithms are usually designed to work with balanced datasets and need additional procedures to handle imbalanced datasets [9][10][11]. Namkoong et al. [12] developed an optimisation method that takes more samples from instances with a high level of importance during stochastic gradient descent. Another common approach involves resampling an equal number of training data for each class to make a balanced dataset. Real-world applications usually generate imbalanced datasets [13] and for this reason approaches for dealing with imbalanced data are needed. Designing a learning method for imbalanced datasets is an important challenge [7], especially in tasks where the cost of incorrect classification of an instance from the minority class, for example misclassifying a cancer patient as a healthy person, is critically high [14].

The two main methods to deal with imbalanced data are those that work directly on data and those based on algorithms. Buda et al. [5] have shown that data imbalance affects the performance of CNN, and they have used different methods based on data, such as oversampling, to process imbalanced data. The impact of class imbalance increases when the scale of the task increases, particularly when large data is utilized [5]. CNN approaches are known to be suitable for classifying large data, however it has no algorithmic strategy for dealing with imbalanced data and there is a need for methods that can classify imbalanced data using the CNN.

Since AdaBoost is an algorithmic method that has been used to overcome the challenges of classifying imbalanced data in classical machine learning methods [1], this paper proposes a method which extends AdaBoost's capabilities to classifying large data using the CNN. In order to embed the capability of CNN in AdaBoost for the task of dealing with imbalanced data, this paper proposes a new algorithm, AdaBoost-CNN, which couples CNN's superior capabilities in analysing and finding patterns in large data with AdaBoost's capabilities of dealing with large imbalanced data. To achieve this it was necessary to construct a new AdaBoost that can be applied for CNN, as a simple application of the standard AdaBoost method does not improve the performance. In the deep structure of CNN which

consists of a high number of layers, there is a large number of learning parameters, and consequently a high number of training samples are usually required to tune the parameters. Reducing the number of effective training samples in the sequential procedure of a conventional AdaBoost can reduce the performance of the AdaBoost. The proposed method, i.e. AdaBoost-CNN, uses the transfer learning property of deep learning methods to overcome this difficulty and to reduce the computational cost of the proposed AdaBoost, making it superior to the conventional AdaBoost and CNN methods.

The paper is structured as follows. Section 2 introduces related works and various ensembles of CNNs. Then, the principle of the proposed AdaBoost-CNN, is discussed in Section 3. Thereafter, experimental results are discussed in Section 4. The experimental section describes the five datasets and experiments carried out to evaluate the performance of the proposed method. Finally, a conclusion is presented in Section 5.

2. Related works

The high performance of ensemble methods has encouraged researchers to design different ensembles of CNNs for different applications in different fields. For example, the top results achieved for the ImageNet Large Scale Visual Recognition Competition (ILSVRC2015) [15] are based on ensemble methods. Ciresan et al. [16] have proposed an ensemble method for CNNs where a number of deep neural columns, i.e. CNNs, are trained on inputs, which are pre-processed in different ways, and thereafter the predictions of the neural columns are averaged. The method is one of the first methods to achieve an accuracy close to human accuracy when applied to the MNIST dataset [16]. Combining the network outputs with a simple average might not be the best approach because it assigns the same level of importance to each network regardless of their accuracies. Frazao and Alexandre [17] have proposed a weighted ensemble method that applies different weights for each CNN. A CNN with a better performance is given a higher weight. Consequently, a better CNN has more influence on the final result. Wen et al. [18] have improved the weighted ensemble method proposed by Frazao and Alexandre [17]. They have proposed a probability-based fusion method for CNNs to recognize facial expression. They constructed different estimators by randomly varying the parameters and architecture of a CNN. The output probabilities of each CNN for different classes are combined to make the probability-based fusion method. The accuracy of each CNN on the validation data was used to generate a weight for the CNN to improve the method. Kim et al. [19] have proposed a method to train an ensemble of CNNs. They have used a variety of network structures, random initial weights, and input normalization to generate different CNN models. Then they have used a hierarchical committee to fuse the output of the trained CNN. Kawana et al. [20] have proposed an ensemble of CNNs for human pose estimation. Each CNN in the ensemble model is optimized for a limited variety of poses. Their method combines the responses of various CNNs for final estimation by considering the interdependencies between the different responses. They divided training data into a number of subsets with similar poses. Then they trained a CNN with each of the subset training data. Finally, they integrated the output of the trained CNNs by feeding their output to a further CNN.

Wang et al. [21] have used a cascaded ensemble of CNNs for breast cancer grading through mitotic count at specific time. They count the number of cells which are in the process of dividing. Tajbakhsh et al. [22] have proposed an automatic polyp detection using an ensemble of CNNs. Different polyp features such as colour, texture, and shape are considered and each CNN become specialized on one type of feature. Ijjina et al. [23] have proposed an ensemble method for human action recognition in video. The maximum value of outputs across all the classifiers is considered as the final output of the ensemble method. Lyksborg et al. [24] have proposed an ensemble of CNNs for accurate volumetric tumour segmentation in magnetic resonance images.

3. The principle of the proposed AdaBoost-CNN algorithm

Multiple weak classifiers are combined with AdaBoost techniques to make a single strong classifier. In this technique a group of weak classifiers are trained sequentially. Each classifier is trained based on the errors of the previous classifier, and a weight is assigned to each training sample to show the degree to which the sample is not trained properly with a weak classifier. The weight of a sample is reduced exponentially if it is trained correctly by the previous weak classifier. Each new weak classifier is trained with more weights for those samples that are not trained appropriately based on the results of the previous weak classifier.

This paper adapts the multi-class AdaBoost method proposed by Zhu et al. [2] to design an AdaBoost method for CNN. The proposed new method is called AdaBoost-CNN. In the baseline multi-class AdaBoost, a variant of the SAMME (Stagewise Additive Modeling using a multi-class Exponential loss function) learning algorithm, called SAMME.R (R for Real) [2] is used. SAMME.R uses the real value of the probability that an input sample belongs to different classes [2].

Suppose the training dataset is $(\boldsymbol{x}_1, c_1), \ldots, (\boldsymbol{x}_n, c_n)$, where $\boldsymbol{x}_i$ is a p-dimensional input vector, i.e. $\boldsymbol{x}_i \in R^p$, and c_i is the output corresponding to $\boldsymbol{x}_i$, and $c_i \in \{1, 2, \ldots, K\}$, where K is the total number of classes. The training goal is to fit a classifier, $C(x)$, using training data. The trained classifier can then be used to find the class label of unseen testing data. A weight is considered for each sample in the training data, as a result there is a data weight vector called $D = \{d_i\}$ where $i = 1, 2, \ldots n$, and n is the number of training samples.

The data weights are initialized by $d_i = 1/n$. Then, M networks, i.e. CNNs, are trained sequentially. In the first iteration of the sequential learning approach, the first CNN weights are initialized randomly and trained for one or more epochs based on the difficulty of the learning task. The first CNN, $C^{m=1}(\boldsymbol{x})$, where m is the number of estimators, is trained on all the training samples with the same weight of $1/n$. There are no differences in importance, i.e. weights, of different training samples for the first CNN. After training, the output of the CNN is calculated for training samples. In AdaBoost-CNN the output of the CNN is a K-dimensional output vector for an input sample. The vector contains the predicted values for the K classes. Each element in the output vector is a real-valued confidence-rated prediction related to a class. The output vector for an input sample $\boldsymbol{x}_i$ is $\boldsymbol{P}(\boldsymbol{x}_i) = [p_k(\boldsymbol{x}_i)], k = 1, \ldots K$, and shows the probabilities that the applied input belongs to the K classes. During testing of an input, the input is assigned to the class with the highest probability. The output of the first CNN, $\boldsymbol{P}^{m=1}(\boldsymbol{x}_i) = [p_k^{m=1}(\boldsymbol{x}_i)]$ is used for updating the data weights, $D = \{d_i\}$ by *(1)*.

$$d_i^{m+1} = d_i^m \exp\left(-\alpha \frac{K-1}{K} \boldsymbol{Y}_i^T \log\left(\boldsymbol{P}^m(\boldsymbol{x}_i)\right)\right), \; i = 1, \ldots, n \qquad (1)$$

where d_i^m is the weight of the i^{th} training sample used by the m^{th} CNN, α is a learning rate, $\boldsymbol{Y}_i$ is the label vector coresponding to the i^{th} training sample, $\boldsymbol{P}^m(\boldsymbol{x}_i)$ is the output vector of the m^{th} CNN in response to the i^{th} training sample. Equation (1) is obtained from the SAMME.R algorithm [2], and in this paper, (1) is used for updating the sample weights for a CNN. If the logarithm of the output vector of the m^{th} CNN, $\boldsymbol{P}^m(\boldsymbol{x}_i)$, and the output label $\boldsymbol{Y}_i^T$ are correlated, and their inner product has a high value, the exponential function in (1) has a lower value (because of the negative sign). The low value of the exponential function in (1) consequently reduces the weight of the training sample for the next CNN, because the current output is close to the label vector and shows that the training sample has been trained by the current CNN. After updating the weights related to all training samples for the current CNN, they are normalized by dividing them by the overall sum of the weights. The trained CNN is saved and the learning of the next CNN is started. In Zhu's AdaBoost algorithm [2] a completely new classifier is initialised randomly to be trained as the next classifier. However, in this paper a new method is proposed to make it suitable for CNN. The classical AdaBoost is not suitable for CNN, because CNN

causes strong correlations between the desired labels, $\boldsymbol{Y}_i^T$, and the actual outputs of the CNN for a large number of training samples. The high correlations consequently reduce the value of the exponential element in (1) for the corresponding samples. As a result, the weights only have sensible values for a small number of training samples that are not trained by the previous CNN. The number of untrained samples is small compared to the large number of CNN learning parameters and, based on the classical AdaBoost method, the subsequent CNN is focused on a small set of untrained samples. Full training of the succeeding CNN from scratch on the small number of training samples forces the CNN to become over fitted on the small set of the data. Additionally, training a CNN from scratch has a high computation cost.

For the subsequent CNN, instead of starting the training of the CNN from a random initial condition, it is proposed that the learning parameters of the trained CNN in the current iteration are transferred to the subsequent CNN such that it learns using the transferred parameters. Transfer learning is one interesting characteristic of CNN and helps the following CNN preserve the previous knowledge acquired in the learning process of the previous CNN. Because the transferred CNN gains good knowledge about the overall data, it does not need to be trained for a high number of learning epochs. Transferring the current learning parameters to the next CNN also reduces the computational cost. After the transfer stage, the previous procedure is repeated for the new CNN, i.e. the CNN is trained for one epoch, the trained CNN output vector is extracted for each training sample and the output is used to update the data weights, $D = \{d_i\}$, and then the weights are normalised. This procedure is repeated for all the CNNs in AdaBoost.

Fig. 1 shows the schematic diagram of the proposed AdaBoost-CNN. The data weights are initialised by $D_1 = \{d_i = 1/n\}$, and the first CNN is trained using the initial data weight. Then the first CNN, $C^1(\boldsymbol{x})$, is used to update the data weights for the second CNN, $D_2 = \{d_i\}$. Additionally, the trained $C^1(\boldsymbol{x})$ is transferred to the second CNN. This procedure is continued to train the M^{th} CNN, $C^M(\boldsymbol{x})$. A detailed pseudocode of the proposed AdaBoost for CNN is provided in Table 1. In each iteration of the sequential learning approach, first the classifier corresponding to that iteration is trained using training data and corresponding data weights, $D = \{d_i\}$. Then based on the result of the trained classifier the data weights are updated for the next iteration. These two actions are performed sequentially for M weak classifiers.

3.1. Training a CNN with a weighted sample

A CNN is usually constructed by stacking a number of convolutional layers, pooling layer, and a fully connected layer [25]. A CNN has a hierarchical structure such that the bottom layers collect the low-level features, whereas the high-level layers extract more complex features which contain more abstract information. The bottom layers of a CNN contain a number of convolutional layers which can collect local information from the input, and map the local information to the next layer in different feature maps. CNN uses a number of shared weights called a kernel, $\boldsymbol{W}$, to map an input to a feature map. Suppose that there are a number of feature maps in the l^{th} layer. Equation (2) can be used to calculate the activity of the i^{th} feature map in the l^{th} layer, $\boldsymbol{y}_i^l$.

$$\boldsymbol{y}_i^l = \sum_j f(\boldsymbol{w}_{i,j}^l * \boldsymbol{y}_j^{l-1} + \boldsymbol{b}_i^l) \qquad (2)$$

where $\boldsymbol{w}_{i,j}^l$ is the convolutional kernel which is used to map the j^{th} feature map in the $(l-1)^{th}$ layer to the i^{th} feature map in the next layer (the l^{th} layer), $\boldsymbol{b}_i^l$ is the bias related to the i^{th} feature map in the l^{th} layer. A nonlinear activation function, such as the rectified linear units (ReLU) function or sigmoid function, f (.), is used in the convolutional layer. The '*' is the convolutional operator sign. A max pooling layer is used after each convolutional layer, and passes the maximum value in a local window. The pooling layer reduces the computational cost by reducing the number of features [26].

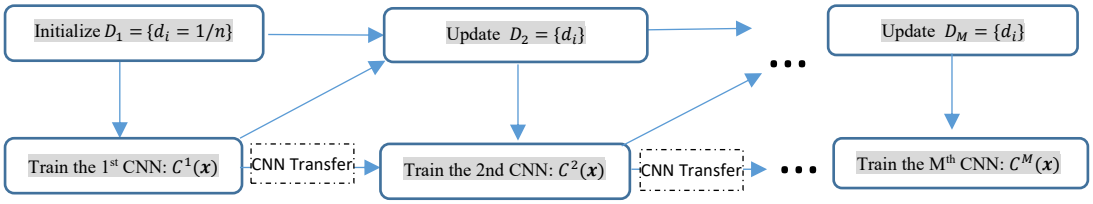

Fig. 1. Schematic diagram of the proposed AdaBoost-CNN which works based on CNN transfer learning.

Table 1 Pseudo code of the proposed AdaBoost-CNN.

Initialize the i^{th} data sample weight with $d_i = 1/n$ where $i = 1, 2, \ldots, n$, and n is the total number of training samples, and initialize M, i.e. the total number of CNNs.

For $m = 1$ to M:

1) **If** $m == 1$:
   Train the first CNN, i.e. $C^{m=1}(\boldsymbol{x})$, on the training data using the initial sample weights, $D_{m=1} = \{d_i = 1/n\}$.
   else:
   Transfer the learning parameters of the previous CNN, $C^{m-1}(\boldsymbol{x})$, to the m^{th} CNN, i.e. $C^m(\boldsymbol{x})$.
   Train the m^{th} CNN, i.e. $C^m(\boldsymbol{x})$, on the training data for one epoch using the sample weight vector $D_m = \{d_i\}$.
2) Obtain the output of the m^{th} CNN, i.e. class probability estimates, for all the K classes: $p_k^{(m)}(\boldsymbol{x})$ where $k = 1, 2, \ldots, K$.
3) Update the data sample weight D_m based on $p_k^{(m)}(\boldsymbol{x})$ using (1).
4) Re-normalize the updated data sample weights, D_m.
5) Save the m^{th} CNN, i.e. $C^m(\boldsymbol{x})$.

Fully connected hidden layers are connected next to the previous convolutional layers. The extracted features from convolutional layers are flattened and feed to the fully connected layer.

$$\boldsymbol{F}^l = f(\boldsymbol{W}^l(\boldsymbol{F}^{l-1})^T + b^l) \tag{3}$$

where $\boldsymbol{F}^l$ is the output of the l^{th} hidden layer, $\boldsymbol{W}^l$ is the weight matrix that connect the l^{th} hidden layer to the previous layer, and b^l is the bias related to the l^{th} hidden layer. $f(.)$ is a non-linear function. Note that the output of the last convolutional layer is flattened to a vector before applying to the next fully connected layer.

A logistic regression model is put on top of the previous layers to construct a categorical output. A SoftMax function is used to convert the output of the regression model to a probability distribution of the classes as shown in (4).

$$\boldsymbol{Z} = softmax(\boldsymbol{W}^o(\boldsymbol{F}^L)^T + b^o) \tag{4}$$

where $\mathbf{Z}$ is the output vector of the network which has an element corresponding to each class; $\boldsymbol{W}^o$ is the weight matrix that connects the output of the last fully connected layer to the output layer; $\boldsymbol{F}^L$ is the output of the last fully connected hidden layer where $\boldsymbol{L}$ is the number of the output neurons which is equal to the total number of classes; and b^o is the bias related to the output layer.

The CNN is trained by the back propagation learning algorithm. Cross entropy is used to calculate error in the learning algorithm. In this paper, each sample has a weight, d_i, and the sample weights are introduced in the error function as shown in (5).

$$E_i = -\sum_{c=1}^{L} t_i^c \log\left(z_i^c\right) d_i \tag{5}$$

where E_i is the error related to the i^{th} sample, t_i^c is the c^{th} element of the label vector corresponding to the i^{th} sample, z_i^c is the c^{th} element of the output vector for the i^{th} sample, and d_i is the sample weight corresponding to the i^{th} input sample.

3.2. Testing with AdaBoost-CNN

After training the M base classifiers, the M CNNs, the resulted AdaBoost-CNN is ready for testing. Equation (6) is used to predict the output class of an input.

$$C(\boldsymbol{x}) = \underset{k}{\operatorname{argmax}} \sum_{m=1}^{M} h_k^m(\boldsymbol{x}) \tag{6}$$

where $h_k^m(\boldsymbol{x})$ is calculated by (7).

$$h_k^m(\boldsymbol{x}) = (K-1)\left(\log\left(p_k^m(\boldsymbol{x})\right) - \frac{1}{K}\sum_{\tilde{k}=1}^{K} \log\left(p_{\tilde{k}}^m(\boldsymbol{x})\right)\right) \tag{7}$$

where $p_k^m(\boldsymbol{x})$ is the k^{th} element of the output vector of the m^{th} CNN when $\boldsymbol{x}$ is applied as its input. Equation (7) was obtained in [2] by using the Lagrange optimization on constrained problem to find an improved estimator in a multi-class AdaBoost.

The Python implementation of the proposed method, AdaBoost-CNN, is publicly available in a GitHub repository (https://github.com/a-taherkhani/AdaBoost_CNN).

4. Experimental Results

This section describes the datasets which are used to perform the experiments and reports the experimental results.

4.1. Datasets

Five datasets were utilised for the experiments. These datasets were a synthetic dataset [2], CIFAR-10 [27][28], Fashion-MNIST [29], EMNIST (an Extended version of MNIST) [30], and a Human Activity Recognition (HAR) dataset [31] [32]. The datasets are described in the subsections that follow.

4.1.1. Synthetic Data

A multi-dimensional standard normal distribution is used to construct an imbalanced synthetic classification dataset. There are three classes in the synthetic dataset, and it contains 12,300 samples, where each sample $\boldsymbol{x}$ is a ten-dimensional vector, i.e. $\boldsymbol{x} \in R^{10}$. The ten variables in $\boldsymbol{x}$ are drawn from a ten-dimensional standard normal distribution. The three classes are arranged as described in [2]. Equation (8) describes the three classes.

$$c = \begin{cases} 1, & 0 \le \sum x_j^2 < \chi_{1/3}^2 \\ 2, & \chi_{1/3}^2 \le \sum x_j^2 < \chi_{2/3}^2 \\ 3, & \chi_{2/3}^2 \le \sum x_j^2 \end{cases} \tag{8}$$

where $\chi_{k/3}^2$ for $k = 1\ and\ 2$ is $\left(\frac{k}{3}\right)100\%$ quantile of the χ^2 distribution generated from the ten-dimensional standard normal distribution, and $\sum x_j^2$ is the Euclidean distance of $\boldsymbol{x}$ from the origin of the Euclidean space. Data related to different classes are separated by nested concentric multi-dimensional spheres. As shown in (8), the samples from class 1 are distributed around the origin of a

sphere, and the samples from class 2 are distributed between the surface of two spheres, and they are far from the origin, i.e. $(x_1, x_2, \dots, x_{10})$ where $x_i = 0 \; for \; i = 1, \dots 10$, compared to the samples in the first class, and so on. A total of 2,300 instances are extracted as training data, such that 800, 500 and 1,000 samples belong to class 1, 2, and 3, respectively, to create a class-imbalanced training dataset. A test set is constructed by extracting 10,000 independent samples. Each sample in the test set belongs to one of three classes with equal probability. There are approximately the same number of testing data points in the test set for the three classes. The number of the samples belonging to each class of the dataset are shown in Table 2.

Table 2 Number of samples for different classes of the synesthetic dataset

Class	# Samples	Percent
	(a) Training	
1	800	34.78%
2	500	21.74%
3	1,000	43.48%
Total	2,300	100%
	(b) Testing	
1	3,326	33.26%
2	3,336	33.36%
3	3,338	33.38%
Total	10,000	100%

4.1.2.CIFAR-10 Dataset

CIFAR-10 includes 60,000 colour images in which 50,000 images are used for training. The size of each colour image is 32×32 pixels. In the training dataset, each class contains 5,000 training images, and the remaining 10,000 images are used for testing. Each image belongs to one of ten classes. The labels of the ten classes are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Fig. 2 shows a set of sample images extracted from the dataset. None of the classes have overlapping samples. For instance, all the samples in the automobile class are different from the samples in the truck class. The same number of training samples exists in each class, and each class has 5,000 training samples. The testing data is also balanced data and there are 1,000 testing samples for each class.

4.1.3.Fashion-MNIST dataset

Fashion-MNIST contains 70,000 grey scale images with the size of 28× 28 pixels. Each pixel has an integer value from 0 to 255. The original data has 60,000 and 10,000 training and testing samples, and each sample belongs to one of ten classes. Fig. 3 shows a set of sample images extracted from the dataset. The labels of the ten classes are: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot. Fashion-MNIST is a balanced dataset and thus each class has the same number of samples. Each class in the training set has 6,000 samples, and each class in the testing set has 1,000 samples.

4.1.4.EMNIST dataset

EMNIST [30] is an extended version of the MNIST dataset [33], and it is a variant of the NIST dataset which contains images of handwritten digits, lowercase and uppercase letters. Each grayscale image in EMNIST has the size of 28×28 pixels, and there are 784 features for each sample. The EMNIST classification task is a more challenging task than the MNIST classification task, as it involves letters and digits. EMNIST contains different data splits, and some of the splits are balanced datasets and the others are imbalanced datasets.

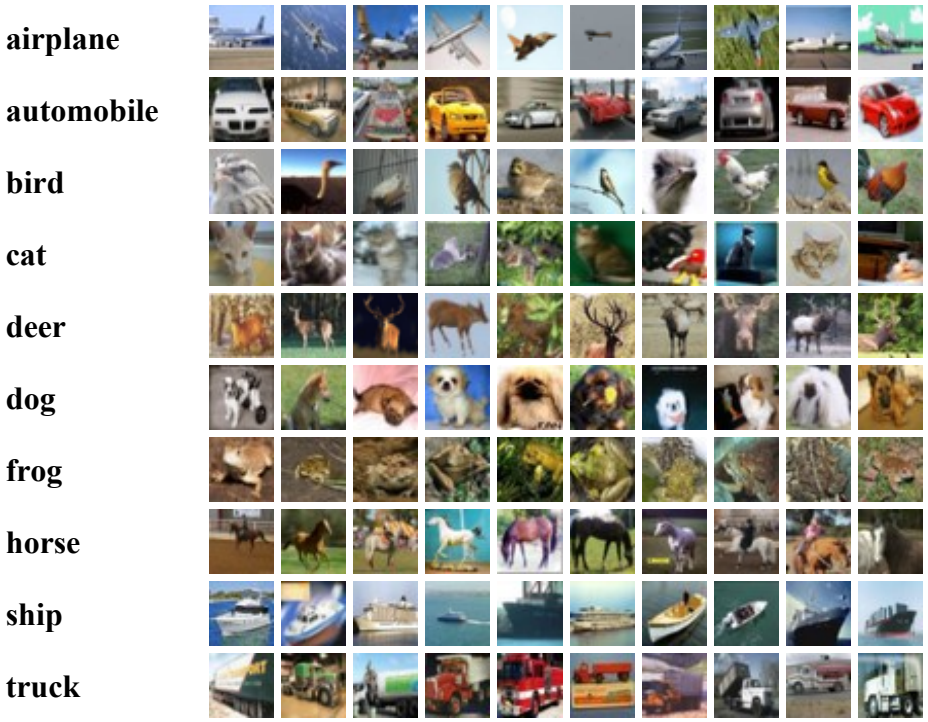

Fig. 2. Ten sample images for each of the 10 classes of the CIFAR-10 dataset [34]

Fig. 3. Samples from Fashion-MNIST.

There are six splits (versions) of the EMNIST data depending on their content of digits or letters, and the number of classes. The two splits that are used for the most difficult classification tasks are the EMNIST by-class and EMNIST by-merge datasets. These two datasets contain more digit samples than letter samples as the nature of the original dataset is based on digits. The uneven distribution of samples in each class makes these imbalanced datasets. EMNIST by-class and EMNIST by-merge datasets each contain 814,255 samples, and the samples are assigned to different classes of digits and letters. Samples in the EMNIST by-class dataset are assigned to 62 classes, and the labels of the classes are 0-9 digits, 26 uppercase and 26 lowercase letters. Fig. 4 shows the number of samples in the different classes of ENMIST by-class. The 62 classes in the dataset have different numbers of samples.

EMNIST by-merge has 47 classes. The number of classes in this data set is lower than the number of classes in EMNIST by-class. The two datasets have the same number of digit classes, however, they have a different number of letter classes. The uppercase and lowercase classes in EMNIST by-class are similar and can cause error; these classes are merged and construct the EMNIST by-merge dataset. EMNIST by-merge is an imbalanced dataset and there are different number of samples in different classes as shown in Fig. 5. The 814,255 samples in the two datasets are divided into training and testing sets, and there are 697,932 training samples in each training dataset as shown in Table 3.

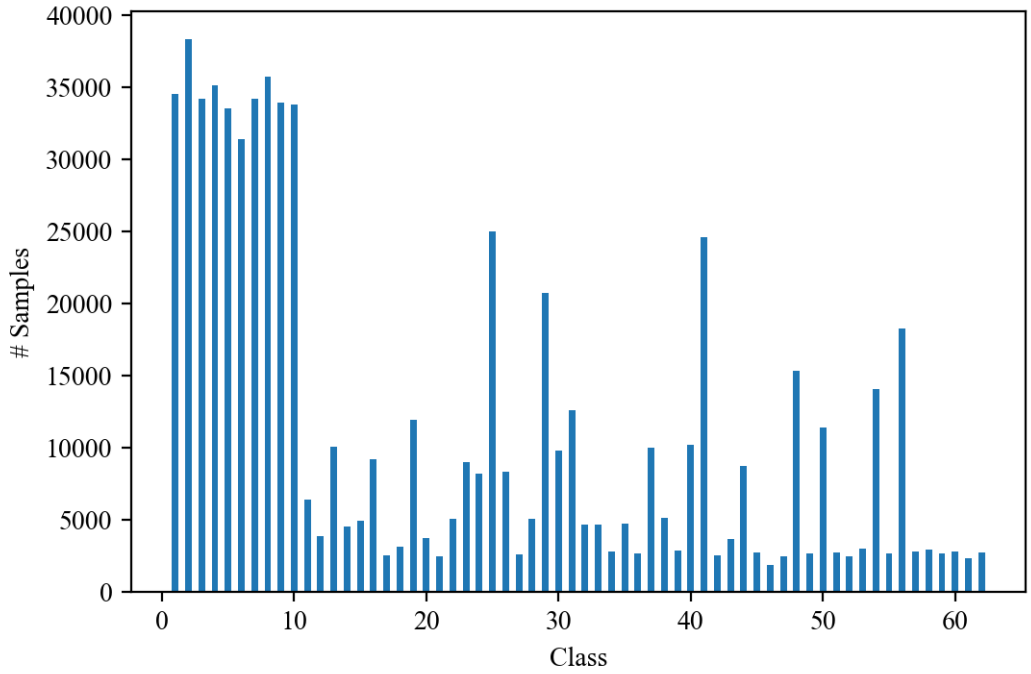

Fig. 4. Histogram of the number of training samples in the 62 classes of the EMNIST by-class dataset. EMNIST by-class is an imbalanced dataset.

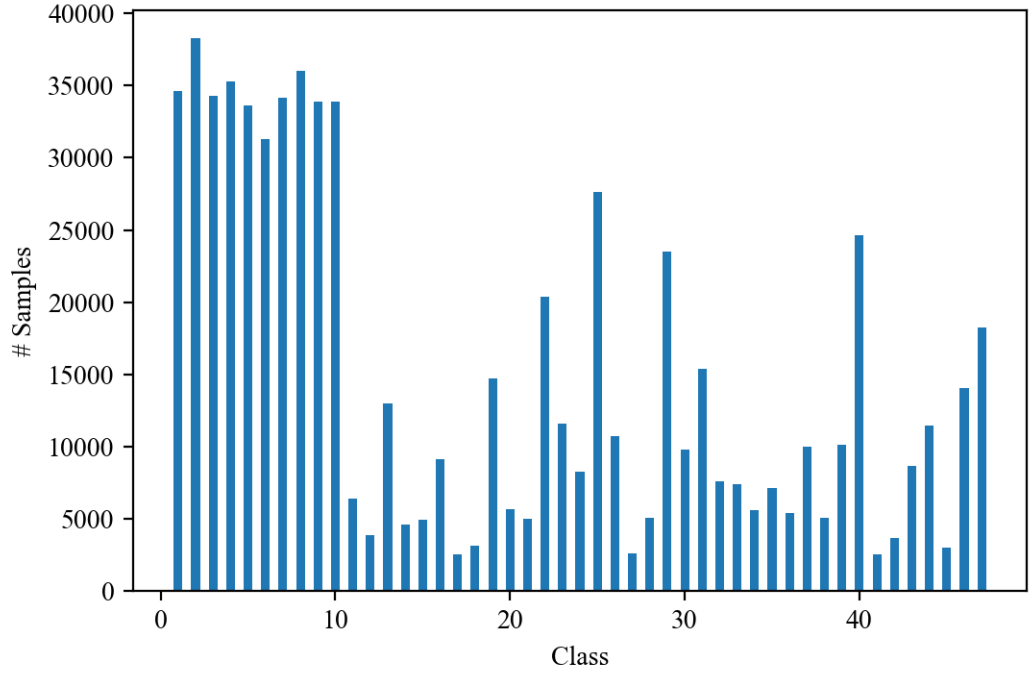

Fig. 5. EMNIST by-merge has different numbers of training samples in its 47 classes, and it is an imbalanced dataset.

Table 3. Characteristics of EMNIST by-class and EMNIST by-merge datasets

Dataset	**# Classes**	**# Training**	**# Testing**	**# Total**
By-class	62	697,932	116,323	814,255
By-merge	47	697,932	116,323	814,255

4.1.5.Human Activity Recognition dataset

The Human Activity Recognition (HAR) dataset was collected using the Sensor HAR App [31] which is installed on a smartphone. The HAR dataset contains 24,075 samples, and each sample has 60 features which are extracted using smartphone accelerometer sensors. For instance, the mean values of x, y and z components of the acceleration during the activity are the features in this dataset. Each sample in the dataset belongs to one of the following five classes: Sitting, Standing, Walking, Running, and Dancing [32]. Fig. 6 shows the distribution of the samples in the different classes of the human activity datasets. Class 2 has the largest number of samples and it contains 25.84% of all the samples in the dataset. Class 5 has the smallest number of samples, i.e. 11.02% (see Table 4).

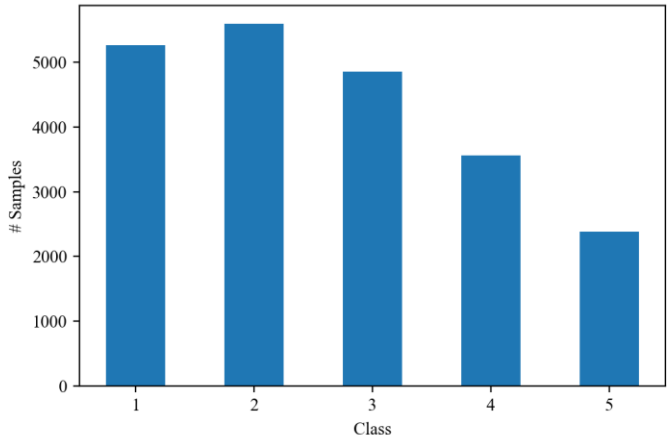

Fig. 6 Number of samples across the various classes.

Table 4. Samples in different classes of the HAR dataset.

Class	# Samples	Percent
1	5850	24.30%
2	6220	25.84%
3	5396	22.41%
4	3956	16.43%
5	2653	11.02%
Total	24,075	100%

4.2. Experimental methodology

The performance of the proposed AdaBoost-CNN is compared against the benchmark CNN using a synthetic dataset [2], CIFAR-10 [27][28], Fashion-MNIST [29], EMNIST (an Extended version of MNIST) by-class [30], EMNIST by-merge [30], and the HAR dataset [31] [32]. In order to obtain comparable results, the same CNN structure is used in CNN and AdaBoost-CNN.

The configuration of the CNN baseline classifier used for the synthetic dataset is shown in Table 5. It has a one-dimensional (1D) convolution layer with a ReLU activation function, followed by a 1D Max-Pooling layer with a pooling size of 2x1. The convolutional layer filter size is 3x1. The layer has 32 feature maps. Then, a Dropout layer is used to randomly exclude 20% of neurons. After that, two fully connected hidden layers with 128 and 64 neurons are used. The ReLU activation function is applied in the hidden layers. Finally, an output layer with three neurons and a SoftMax function is used. The 'Adagrad' optimizer with the initial values used in Keras was also used. This optimizer updates parameter values depending on how frequently a parameter is updated. A parameter with frequent updates will have a smaller learning rate [35]. Categorical cross entropy is used as the loss function.

Table 5. Configuration of CNN used for the synthetic dataset.

Layers	**Configuration**
1D Convolution	32 filters, 3x1 kernel and ReLU
Max-Pooling	2x1 kernel
Dropout	20%
Fully connected	128 Neurons, ReLU
Dropout	20%
Fully connected	64 Neurons, ReLU
Fully connected	3 Neurons, SoftMax

The structure of the CNNs which are used for CIFAR-10 and Fashion-MNIST is shown in Table 6. The number of classes, i.e. 10, and the number of training data for these datasets are higher than those of the synthetic datasets. Consequently, a deeper CNN configuration is used for the datasets. There are 10 neurons in the output layer of the CNN. A similar structure for CNN shown in Table 6 is used for EMNIST by-class and EMNIST by-merge datasets. The only difference is the number of output neurons which is equal to the number of classes in each dataset. The CNN used for EMNIST by-class and EMNIST by-merge have 62 and 47 out neurons, respectively. The HAR dataset comprises 1D datasets, therefore, 1D convolution layers and 1D Max-Pooling layers are used. The remaining characteristics of the CNNs used for the HAR dataset are the same as the ones shown in Table 6.

For the CFAR-10, Fashion-MNIST, EMNIST, and HAR datasets, the RMSprop optimizer, which is an optimizer with an adaptive learning rate, has been used to train each CNN. The initial learning rate of the optimizer is set to 0.0001, and the learning rate decay over each update is set to $11e^{-6}$. Categorical cross entropy is used as the loss function.

Table 6. Configuration of CNN used for CIFAR-10 and Fashion-MNIST.

Layers	**Configuration**
2D Convolution	32 filters, 3x3 kernel and ReLU
2D Convolution	32 filters, 3x3 kernel and ReLU
Max-Pooling	2x2 kernel
Dropout	25%
2D Convolution	64 filters, 3x3 kernel and ReLU
2D Convolution	64 filters, 3x3 kernel and ReLU
Max-Pooling	2x2 kernel
Dropout	25%
Fully connected	512 Neurons, ReLU
DropOut	50%
Fully connected	10 Neurons, SoftMax

4.3. Experimental results

In this section the performance of AdaBoost-CNN is evaluated on the five datasets described in Section 4.1. Then, the effect of different levels of imbalance is investigated.

4.3.1. Experimental results on a synthetic dataset

The performance of the baseline AdaBoost method introduced in [2], CNN and the proposed AdaBoost-CNN are evaluated on a synthetic dataset (see Section 4.1.1.). The original AdaBoost method [2] is used as a baseline classifier.

Results when using a conventional AdaBoost with Decision Tree: AdaBoost with decision tree is a conventional AdaBoost method which can be used to deal with imbalanced data. In this conventional AdaBoost, 600 decision tree classifiers were used as weak classifiers. The maximum depth of each decision tree is set to 2. The accuracy of the conventional AdaBoost method [2] on the training dataset is 91.78%, and its accuracy on the testing dataset is 77.08%. The accuracy of the conventional AdaBoost with decision tree is compared to a single CNN and the proposed method in Table 8. The proposed method has a 16.98% higher accuracy than the conventional AdaBoost with decision tree.

Results when using the CNN baseline classifier: In the proposed method, sample weights are used to control the learning on different training samples. In Table 7 the number of layers in the base network, shown in Table 5, is changed, and networks with 4 to 10 layers are constructed. Then, the testing accuracy values of the different networks are reported to find an optimum number of layers. The results in Table 7 illustrate that the testing accuracy values are higher when there are 7 layers in the network. The base network with optimum number of layers, i.e. 7 layers, has a testing accuracy of 92.05%, which is lower than the testing accuracy of the proposed AdaBoost-CNN, i.e. 94.08%.

Uncontrolled learning of a large CNN with a high number of training parameters on a limited number of training data could cause overfitting on the training data and reduce its testing accuracy. Therefore, finding an appropriate strategy to train a deep learning method to increase the testing accuracy is required. In the proposed method, sample weights are used to control the learning on different training samples.

Table 7. Accuracy of the base network with different number of layers

#Layers	Testing accuracy (%)	Layers
4	81.15	Fully-Dropout-Fully-Fully
5	70.81	Dropout-Fully-Dropout-Fully-Fully
6	90.91	Conv- Dropout-Fully-Dropout-Fully-Fully
7	92.05	Conv-Pooling-Dropout-Fully-Dropout-Fully-Fully (the original base model in Table 5)
8	91.05	Conv-Pooling-Conv-Dropout -Fully-Dropout-Fully-Fully
9	90.27	Conv-Pooling - Conv- Pooling-Dropout-Fully-Dropout-Fully-Fully
10	89.99	Conv-Pooling-Conv-Pooling-Conv-Dropout-Fully-Dropout-Fully-Fully

The training and testing accuracy values of a single CNN applied to the synthetic data are reported in Table 8 (b). CNN is trained for different numbers of learning epochs as shown in the third column of Table 8 (b). The testing accuracy of the baseline CNN method increases from 91.30% to 92.05% when the number of learning epochs is increased from 5 to 10, but the testing accuracy is reduced when the number of learning epochs is increased to 15. The imbalanced nature of the training data causes the

trained CNN to become biased to the class with the highest number of training data and consequently reduces the accuracy of the CNN when the number of training epochs is increased (see Table 8 (b)).

Table 8. Accuracy values of (a) Conventional AdaBoost with Decision Tree (AdaBoost-Decision-Tree), (b) a single CNN, and (c) the proposed AdaBoost-CNN on synthetic data.

Training accuracy	Testing accuracy on 10,000 testing data	# Epochs/ # Estimators
(a) AdaBoost-Decision-Tree		
91.78%,	77.08%	600 Estimators
(b) A single CNN		
95.00%	91.30%	5 Epochs
95.74%	92.05%	10 Epochs
95.21%	91.25%	15 Epochs
(c) The Proposed AdaBoost-CNN		
94.65%	93.46%	8 Estimators
95.61%	94.08%	10 Estimators
95.87%	94.06%	15 Estimators
95.78%	93.91%	20 Estimators

Results when using the proposed AdaBoost-CNN classifier: The training and testing accuracies of AdaBoost-CNN on the synthetic data are shown in Table 8 (c). The CNN with 7 layers which was used in the previous experiment is used as the base estimator in AdaBoost-CNN. Performance is evaluated for different numbers of estimators in the AdaBoost-CNN as shown in the third column of Table 8 (c). Each estimator is trained with its sample weights for one learning epoch. The testing accuracy of the methods on 10,000 testing samples increases from 93.46% to 94.08% when the number of estimators is increased from 8 to 10. Accuracy is reduced to 93.91% when a higher number of estimators (i.e. 20) is used. When the number of estimators is increased the number of untrained samples for the new estimators is reduced, and the new estimators are trained with a very low number of training samples with high weights (the weights related to the other training samples are small enough to be neglected; because they are already trained with the previous estimators). Consequently, the new estimators are not trained appropriately on the entire dataset and the overall performance of the method cannot be improved when the number of estimators is increased than 10 (see Table 8 (c)). AdaBoost-CNN has reached its highest testing accuracy, i.e. 94.08%, when 10 estimators are used, and this accuracy is higher than the best accuracy obtained by a single CNN, i.e. 92.05% (see Table 8). The proposed AdaBoost-CNN correctly recognised 203 testing samples more than the single CNN on the testing data.

The accuracy values of the proposed AdaBoost-CNN, the single CNN and the conventional AdaBoost with decision three are compared in Table 10. The highest accuracy obtained for each method is reported in Table 10. AdaBoost-CNN has 10 estimators and each estimator is trained with its sample weights for one learning epoch. In total there are 10 learning epochs for AdaBoost-CNN, and for the single CNN. The CNN achieved its highest accuracy at 10 learning epochs (see Table 8 (c)). The results show that AdaBoost-CNN has the highest accuracy compared to the two other methods. AdaBoost-CNN reached an accuracy of 94.08%, which is 2.03% higher than the performance of the single CNN. The experimental results also show that the accuracy of AdaBoost-CNN is much higher than that of the conventional AdaBoost which uses decision tree as estimator. AdaBoost-CNN has 17% higher testing accuracy compared to the conventional AdaBoost with decision tree. The results show that the conventional AdaBoost with decision tree cannot achieve the accuracy level of a single CNN, and its

accuracy is far below the accuracy of a single CNN. Therefore, the conventional AdaBoost with decision tree was not considered in the remaining experiments.

The number of learning epochs are set in such way that the number of all training epochs for all the estimators is the same as the number of learning epochs used for a single CNN so as to make a fair comparison. Table 8 shows that the single CNN can achieve its maximum accuracy on the synthetic data when it is trained for 10 learning epochs. The same number of training epochs is used for AdaBoost-CNN. Therefore 10 estimators with a single learning epoch can be used to have the same number of learning epochs for AdaBoost-CNN and the single CNN. Table 8 (c) shows that when 10 estimators (each of which is trained for one learning epoch) are used, the proposed method reaches the highest accuracy.

Experiments were carried out to evaluate the performance (training and evaluation accuracy) of the single CNN across different learning epochs. 15% of training samples were randomly selected to construct the validation set. Note that the testing data were not used during training. Fig. 7 shows that after 10 learning epochs, there was no significant improvement in the accuracy on the validation set. The single CNN is used as a building block of AdaBoost-CNN. Therefore, any degradation or improvement on the accuracy of the single CNN as a result of the number of learning epochs would lead to a similar change in the accuracy of the proposed AdaBoost-CNN. The same number of learning epochs are used for both methods for a fair comparison.

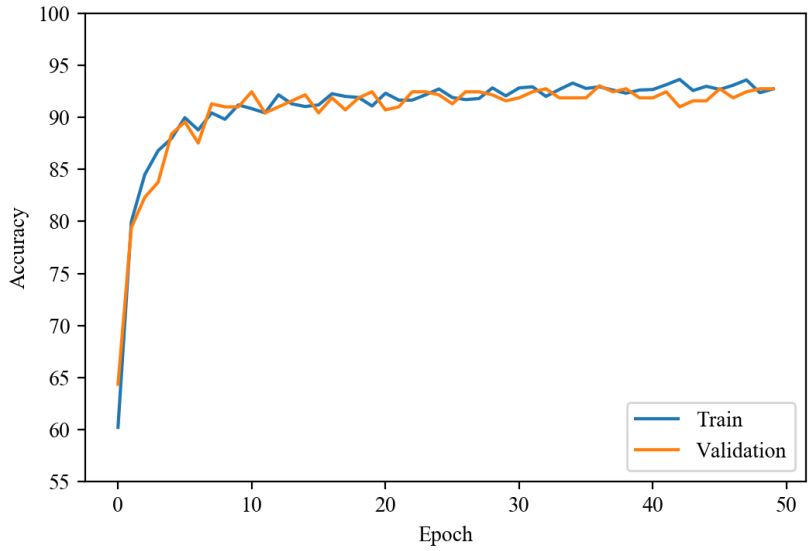

Fig. 7. Training and validation accuracies of the single CNN across different learning epochs

Results when using the Deep Residual Network (ResNet): ResNet is a well-known deep structure for CNNs, and it has achieved state-of-the-art results in different applications [36]. The performance of AdaBoost-CNN is compared to the ResNet used in [36]. ResNet has three residual blocks, and it uses global average pooling at the layer of the network before its output layer. ResNet contains 504,387 trainable parameters. In Table 9, ResNet is compared to AdaBoost-CNN that has 291,870 trainable parameters. ResNet is trained with different numbers of learning epochs and its training and testing accuracies are reported in Table 9. ResNet achieved its highest testing accuracy, i.e. 82.81%, with 12 learning epochs which is lower than the testing accuracy of the proposed AdaBoost-CNN, i.e. 94.08%. AdaBoost-CNN has 10 estimators each of which is trained for one learning epoch. The lower number of trainable parameters for the proposed AdaBoost-CNN leads to a lower computation time compared to ResNet (see Table 9).

Table 9 Comparison of the proposed AdaBoost-CNN with ResNet

# Epoch/Estimator	Testing accuracy (%)	Training accuracy (%)	Computation time (Sec.)	# Parameter
ResNet				
10 Epochs	78.52	89.38	73.09	504,387
12 Epochs	82.81	94.87	85.85	504,387
14 Epochs	80.82	93.65	99.79	504,387
Proposed AdaBoost-CNN				
10 Estimators	94.08%.	95.61%	47.33	291,870

Results when using the CNN with the weighted loss function: New experiments were carried out with the weighted loss function. In particular, the weighted loss function, which is a general solution to deal with imbalanced data, was used with CNN to construct a CNN with weighted loss function (CNN-Weighted-Loss (row 2 of Table 10)) to compare with the proposed AdaBoost-CNN (row 1 of Table 10). The experimental results show that the testing accuracy of AdaBoost-CNN is 94.08%, higher than the accuracy of CNN-Weighted-Loss, which achieved a testing accuracy of 93.09%. Using the weighted loss function with CNN improved accuracy by 1.04% compared to when using CNN with the regular loss function (row 4 Table 10), which gave an accuracy of 92.05%.

Results when using a conventional oversampling method: In order to compare against other approaches that deal with class imbalance, the proposed method is compared to the conventional oversampling method called the Synthetic Minority Oversampling Technique (SMOTE) [37]. The results are shown in Table 10 in the method row called 'CNN-Over-Sampling'. The results show that the testing accuracy of the proposed method reached 94.08% which is higher than the accuracy of CNN-Over-Sampling. The testing accuracy of CNN-Over-Sampling is 92.45%.

Results when using a voting method with CNNs (Voting-CNNs): In order to compare the proposed method with a method that has the same number of training parameters, a number of CNNs are trained and then voting is used to assign a label to an applied input. The number of CNNs used in the voting method is the same as the number of estimators in AdaBoost-CNN. Therefore, the same number of training parameters is used for both methods. The testing accuracy of the voting method is 92.47%, which is lower than the testing accuracy of AdaBoost-CNN, i.e. 94.08% (see Table 10). AdaBoost-CNN correctly recognized 161 testing samples more than the voting method.

Table 10. Comparison of the best results achieved by various methods on synthetic data.

Row	Method	Training accuracy	Testing accuracy on 10,000 testing data [b]
1	Proposed AdaBoost-CNN	95.61%	94.08%
2	CNN-Weighted-Loss	95.83%	93.09%
3	CNN-Over-Sampling	95.38%	92.45%
4	CNN	95.74%	92.05%
5	AdaBoost-Decision-Tree	91.78%	77.08%
6	Voting-CNNs	96.52%	92.47%,
7	ResNet	94.87%	82.81%

Investigating the accuracy of each CNN estimator in the proposed AdaBoost-CNN: Fig. 8 shows the training and testing accuracies of different CNN estimators in AdaBoost-CNN. The 'x' axis shows the number of different CNN estimators in the proposed AdaBoost-CNN. There are 10 CNN estimators in the AdaBoost-CNN. Fig. 8 shows that the fourth estimator has achieved the highest training accuracy of 94.04% compared to the other estimators in AdaBoost-CNN. The accuracy of the fourth CNN estimator on the testing accuracy is 91.85%. However, a single CNN which is trained for 10 learning epochs cannot reach a testing accuracy higher than 90.97%, even though the training accuracy of the single CNN is 94.91%. Therefore reducing the effect of already trained samples in subsequent epochs can be useful to prevent over fitting a CNN. Note that the overall performance of AdaBoost-CNN is higher than the testing accuracy of its ingredient CNN estimators, and the single CNN. AdaBoost-CNN can achieve testing accuracy of 94.08%. The improvement in AdaBoost-CNN can be related to its ability to prevent overfitting. Finding a method to prevent overfitting in deep learning, that can get very small loss on complex training data, is a known challenge [38]. In each learning epoch, sample weights reflect how much the current CNN is trained by each training sample. If a training sample is trained by a CNN estimator, its weight is reduced and consequently its effect on training is reduced for the next CNN. Therefore, the next CNN is not overtrained on the samples that are already trained, and this might prevent overfitting of the next CNN on the samples that already trained in the previous learning procedure.

In AdaBoost-CNN, different CNNs are trained on all the training samples with different sample weights. If a sample is trained properly by a number of previous estimators, the weight related to that sample is reduced exponentially. Consequently, this weight has a very small value compared to other weights and therefore its effect on the training of the next estimator can be neglected. If there are disjoint clusters of samples in a class then the small cluster of training samples, which were not trained with the previous CNNs, acquire high value weights and they will be trained by the subsequent CNN. Therefore, the subsequent CNN becomes expert on the training samples with the high weights. The combination of different CNNs which are expert on different groups of training samples results in a strong classifier.

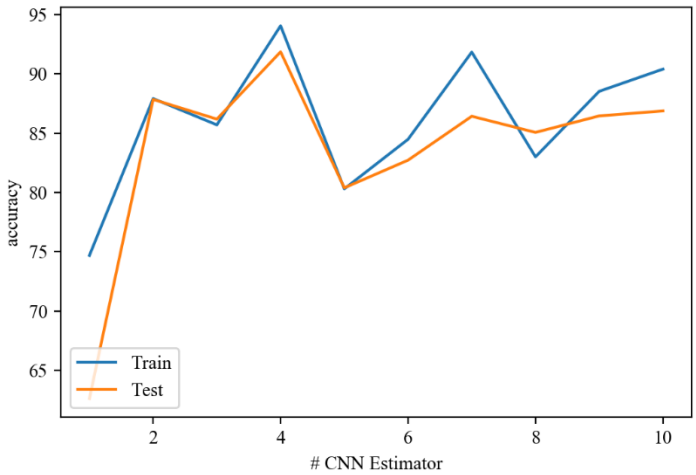

Fig. 8 Accuracy of different CNN estimators in the proposed AdaBoost-CNN algorithm

Investigating the importance of the transfer learning: Transfer learning is an important characteristic of AdaBoost-CNN. To evaluate the effect of transfer learning on computation cost and accuracy, AdaBoost-CNN is compared to an AdaBoost method with CNN estimators where each CNN estimator is trained from scratch for a number of training epochs. The first estimator in this AdaBoost method is trained for 10 epochs, then the sample weights are evaluated and the second CNN estimator is trained for 10 epochs from scratch using the sample weights obtained from the previous CNN estimator and so

on for the next CNNs. The overall testing accuracy of this AdaBoost with CNN estimators is 90.65%, which is lower than the testing accuracy of the proposed AdaBoost-CNN which achieved a testing accuracy of the 94.08%. Moreover, the proposed method reduces the computation cost by reducing the number of training epoch in subsequent CNN estimators. In the proposed AdaBoost-CNN, instead of training each subsequent estimator from scratch for a high number of learning epochs, it uses transfer learning and each pretrained CNN estimator is trained for a small number of training epochs to train the next estimator. The simulation time for the AdaBoost that trains all CNN from scratch for 10 epochs is 225.83 seconds, whereas the computation time for AdaBoost-CNN is 47.33 seconds. The simulation was run using an Intel(R) Core(TM) i7-6700HQ @ 2.60GHz 2.59GHZ processor with 64.0 GB installed memory (RAM), and 64-bit operating system. An NVIDIA Quadro M1000M GPU was used to train both methods. The accuracy and computation time values for AdaBoost without transfer learning are shown in Table 11 when each estimator is set up to train for different numbers of epochs (see the first column).

Table 11. Accuracy and computation time for AdaBoost without transfer learning, and for the proposed AdaBoost-CNN.

# Epoch	Testing accuracy (%)	Training accuracy (%)	Computation time (Sec.)
AdaBoost without transfer learning			
10	90.65	92.00	225.83
15	91.83	95.30	324.07
20	91.18	96.09	415.35
25	89.93	94.17	475.45
Proposed AdaBoost-CNN			
10	94.08%.	95.61%	47.33

4.3.2. Experimental results on CIFAR-10

Three sets of experiments were conducted on the CIFAR-10 dataset as described in the following sub sections.

Effect of different numbers of CNN estimators in the AdaBoost-CNN: The first experimental results on CIFAR-10 are shown in Table 12. Different numbers of CNN estimators in the AdaBoost-CNN and different numbers of learning epochs for each estimator are tested. AdaBoost-CNN has training and testing accuracies of 97.49% and 79.00% respectively when two estimators with 25 training epochs are used. The two estimators in AdaBoost-CNN are trained on the weighted training samples for 25 epochs.

Table 12. Accuracies of the proposed AdaBoost-CNN in the first experiment on CIFAR-10.

Training accuracy (%)	Testing accuracy (%)	#estimators (M)	#epochs
97.49	**79.00**	2	25
96.16	77.81	3	25
77.40	66.21	10	3

Effect of changing the number of learning epochs of the first CNN estimator: In the second set of AdaBoost-CNN experiments on CIFAR-10, the first estimator is trained for a high number of epochs compared to the other estimators in the AdaBoost-CNN. The number of learning epochs for the first

estimator is increased from 20 to 48 epochs as shown in the fifth column of Table 13. A high number of learning epochs for the first estimator increases the accuracy of the first CNN, causing the next CNN, which works on the results of the first CNN, to have a higher accuracy. Table 13 shows that this method can increase the testing accuracy of AdaBoost to 80.13%, which is higher than the accuracy of 77.89% (see Table 12) achieved by the base CNN in 50 learning epochs. As shown in the last row of Table 13, the first estimator is trained for 48 learning epochs, and the second and the third estimator is trained for one learning epoch. Therefore, altogether there are 50 learning epochs for AdaBoost-CNN. When a single CNN is trained for 50 learning epochs it cannot achieve an accuracy more than 77.89% (see Table 12); that is 2.24% lower than the accuracy of the AdaBoost-CNN.

Table 13. Accuracy of the proposed AdaBoost-CNN when different numbers of learning epochs are used for the first estimator.

No.	Training accuracy	Testing accuracy on 10,000 testing data	#estimator (M)	#epoch for the first estimator	#epoch for the other estimators
1	93.10%	79.11%	5	20	1
2	94.17%	79.08%	6	20	1
3	95.67%	79.38%	2	23	1
4	95.42%	79.48%	3	23	1
5	94.34%	79.05%	4	23	1
6	91.79%	76.84%	5	23	1
7	98.47%	79.69%	4	47	1
8	98.93%	**80.13%**	3	48	1

Effect of partially training the next CNN estimators: In the final experiment on the CIFAR10 dataset, all the layers of the first estimator are trained for 48 learning epochs. For the subsequent estimators only the fully connected layers are trained, and the weights related to the previous layers, i.e. the weights of the convolutional layers, are kept fixed. The corresponding trained weights of the first CNN are transferred to the fixed weights of the next CNN, i.e. the weights are set to the corresponding weight values of the first trained CNN, and after that the fixed weights are not trained. This prevents overtraining of the next estimator to a small set of training samples which are not trained appropriately by the previous estimator. Additionally, the fixed weights keep the knowledge related to previously trained samples and this reduces the error of the estimator on the previously trained samples. The experimental results in Table 14 show that by using the proposed AdaBoost-CNN, testing accuracy reached 81.40% on the CIFAR-10 data. This accuracy is higher than the previously obtained testing accuracy value of 80.13% reported in the last row of Table 13. In the lower part of Table 14 the accuracies of the single CNN are shown. The single CNN has lower accuracies of 78.99% and 77.89% for training and testing data, respectively for 50 epochs. All parameters of the single CNN, such as the number of layers, and optimization parameters, are set to the same values as used for each CNN in AdaBoost-CNN (see Table 6).

Table 14. Comparison of the proposed AdaBoost-CNN with a single CNN on CIFAR-10 dataset

Method	Training accuracy (%)	Testing accuracy (%)	#estimators (M)	#epochs
Proposed AdaBoost-CNN	99.74%.	81.40%	3	50 (for all the three estimators)
CNN	**78.99**	**77.89**	1	50

4.3.3. Experimental results on Fashion-MNIST

In this subsection the effect of partially training subsequent CNN estimators after the first completely trained estimator, is investigated. Then the performance of the proposed AdaBoost-CNN is compared to a single CNN.

Effect of partially training the next CNN estimators: In the first experiment shown in the first row of Table 15, all the layers are trained for all the CNN estimators. AdaBoost-CNN has 4 estimators. The first estimator is trained for 22 learning epochs and the other 3 estimators are trained for 1 learning epoch. Altogether there are 25 learning epochs for AdaBoost-CNN. AdaBoost-CNN reached a testing accuracy of 92.99% (see Table 15).

Table 15. Accuracy of the proposed AdaBoost-CNN on Fashion-MNIST.

Partially training	Training accuracy	Testing accuracy on 10,000 testing data
No	96.50%	92.99%
Yes	97.34%	93.36%

In the next experiment, where the results are shown in the second row of Table 15, the first estimator of AdaBoost-CNN is trained thoroughly by the Fashion-MNIST data and the training of the subsequent estimators is restricted to the fully connected layers, and their convolutional layers' weights are fixed to the value of the first trained estimator. The results in the second row of Table 15 show an improvement of the testing accuracy compared to the previous where all the layers are trained for all estimators.

Comparison of AdaBoost-CNN to the single CNN: The accuracies of AdaBoost-CNN and CNN on Fashion-MNIST are compared in Table 16. The testing accuracy of AdaBoost-CNN is 93.36% which is higher than the testing accuracy of CNN, 92.00%.

Table 16. Accuracies of the proposed AdaBoost-CNN and the single CNN on Fashion-MNIST.

Method	Training accuracy	Testing accuracy on 10,000 testing data	#estimator (M)	# Epochs for the 1st estimator	#epoch for the other estimators
Proposed AdaBoost-CNN	97.34%	93.36%	4	22	1
CNN	92.40%	92.00%	1	25	NA

4.3.4. Experimental results on EMNIST

Experimental results on EMNIST by-class: The performance of AdaBoost-CNN compared to that of CNN on EMNIST by-class is reported in Table 17. CNN is trained for 15 learning epochs, and AdaBoost-CNN has four estimators. The first estimator of AdaBoost-CNN is trained for 12 epochs and the number of learning epochs of the other three estimators are set to one. The results show that AdaBoost-CNN has reached higher testing accuracy than CNN. Note that the total number of learning epochs is set to 15 for both methods. AdaBoost-CNN's misclassification rate on the training data is 1.70% lower than the misclassification of CNN, and it has 11,864 less misclassified samples than CNN.

The balanced accuracy score has been used to compute accuracy for imbalanced datasets [39]. In the balanced accuracy score, a weighted average is used to calculate the overall accuracy. The weights related to different samples are specified based on the number of samples in the class of the samples. The score varies from 0 to 1, and when a classifier correctly classifies all samples, its balanced accuracy score will be 1. The balanced accuracy score is multiplied by 100 to get balanced accuracy percentage. If a dataset is balanced, the balanced accuracy percentage will be equal to the usual accuracy percentage. The balanced accuracy percentages are computed when applying AdaBoost-CNN and the baseline CNN on the EMNIST by-class and EMNIST by-merge datasets which are large imbalanced datasets, and each of which contains 814,255 samples. The results are shown in Table 17 and Table 18. The testing 'balanced accuracy' for AdaBoost-CNN on EMNIST by-class is 75.66%, whereas the single CNN cannot reach a testing 'balanced accuracy' more than 73.27%.

Table 17. Accuracy and balanced accuracy of the proposed AdaBoost-CNN and CNN on the EMNIST by-class dataset.

Method	**Training accuracy (%)**	**Testing accuracy (%)**	**Training balanced accuracy (%)**	**Testing balanced accuracy (%)**	**# Estimator/ Epochs**
Proposed AdaBoost-CNN	88.51	87.74	77.57	75.66	4 estimators (15 epochs for all estimators)
CNN	86.81	86.22	73.83	73.27	15 epochs

Fig. 9 shows the accuracy values of AdaBoost-CNN and CNN on testing data for different classes. The different classes in the EMNIST by-class dataset have different number of samples as shown in Fig. 4. The *mean accuracy* across all classes for AdaBoost-CNN is 75.69%, which is higher than that of the CNN on EMNIST by-class, i.e. 73.45%. Note that the *mean accuracy* evaluation metric assigns the same weight to all the classes independent of the number of samples in each class. However, *total accuracy* is calculated based on the total number of correct identifications regardless of the class that the samples belong. On average, AdaBoost-CNN has achieved higher accuracy than CNN across the different classes. For instance, the accuracy of AdaBoost-CNN on class 39 is 8.66 times higher than the accuracy of CNN in the same class. There are accuracy improvements in the other classes such as the classes 19 and 48 as shown in Fig. 9. On average the *mean accuracy* of AdaBoost-CNN on different classes is 2.24% higher than that of CNN.

The number of training samples in the 62 classes of the EMNIST by-class dataset are shown in Fig. 4. Each class contains a portion of EMNIST by-class dataset. The number of training samples in a class is normalised in the range [0,100], as shown in Fig. 9. The normalised value for the number of training samples in a class is obtained by dividing the number of training samples in the class by the number of samples in the class that has the highest number of training samples compared to the other classes. Then the results are multiplied by 100 to achieve a number in the range [0, 100] to plot versus accuracy in Fig. 9.

Experimental results on EMNIST by-merge: Table 18 compares the testing accuracies of the proposed method and CNN on EMNIST by-merge dataset. The total testing accuracy of AdaBoost-CNN is 91.02%, which is higher than the accuracy of CNN, i.e. 89.86%. Additionally, AdaBoost-CNN increased the total testing accuracy by 1.16%. The 'testing balanced accuracy' values for AdaBoost-CNN and the single CNN on EMNIST by-merge dataset are 89.37% and 87.57% respectively. Fig. 10 compares the testing accuracy of AdaBoost-CNN and CNN on the EMNIST by-merge dataset for the 47 classes of the dataset. The mean testing accuracy on the 47 classes for AdaBoost-CNN is 89.26% which is higher than that of CNN, 88.40%.

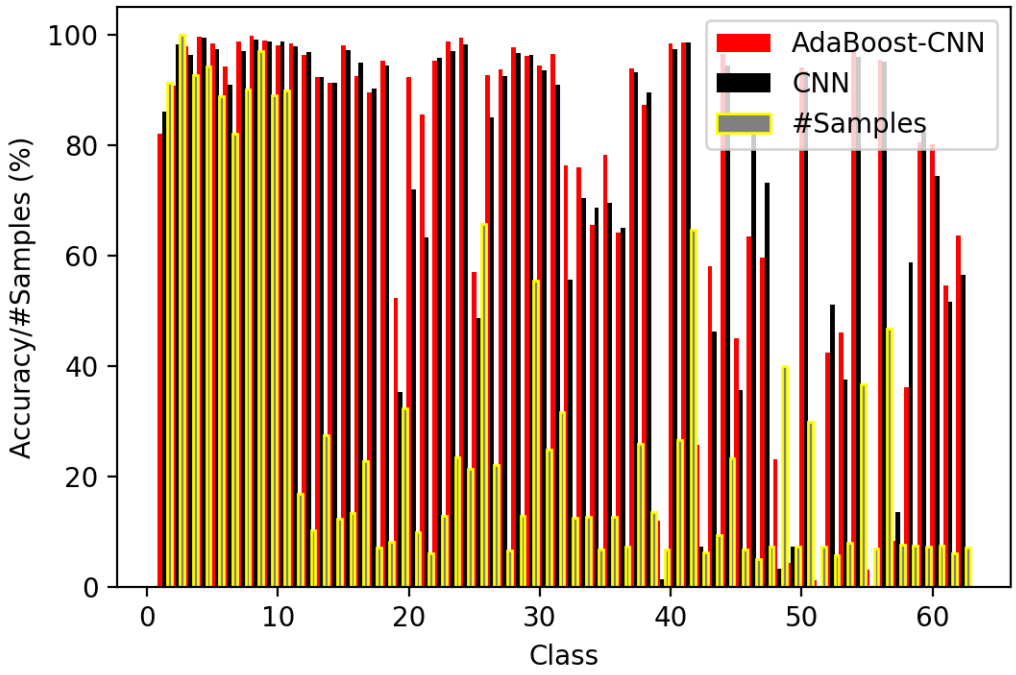

Fig. 9. Testing accuracies of the Proposed AdaBoost-CNN and CNN on EMNIST by-class for different classes. The normalised value of the number of samples for each class (#samples) is also shown.

Table 18. Accuracies and balanced accuracies of the proposed AdaBoost-CNN and CNN on EMNIST by-merge dataset.

Method	**Training accuracy (%)**	**Testing accuracy (%)**	**Training balanced accuracy (%)**	**Testing balanced accuracy (%)**	**# Estimator/ Epochs**
Proposed AdaBoost-CNN	91.78	91.02	90.56	89.37	4 (15 epochs for all estimators)
CNN	90.25	89.86	87.93	87.57	15

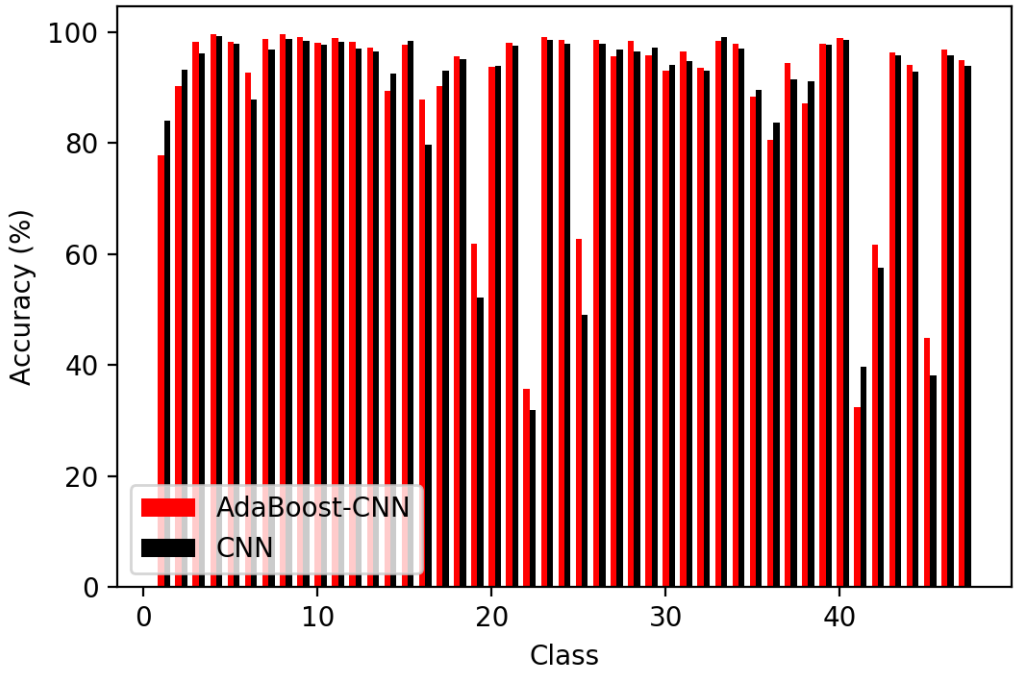

Fig. 10. Testing accuracy of the proposed AdaBoost is compared to the accuracy of CNN on EMNIST by-merge dataset on the dataset's 47 classes.

4.3.5. Experimental results on the Human Activity Recognition (HAR) data

In order to select the most appropriate learning epoch for a single CNN, experiments were carried out with a number of learning epochs and the performance (in terms of accuracy) of the CNN was evaluated across the various epochs. Table 19 shows the accuracy of the single CNN on the HAR dataset for different numbers of learning epochs. The results show that 50 learning epochs are an optimal value for the CNN to reach a high testing accuracy on this dataset. Table 20 shows the accuracy values of AdaBoost-CNN with different number of CNN estimators. The total number of learning epochs for all estimators in each row in Table 20 is 50. The results show that the AdaBoost-CNN achieved the highest accuracy with 5 estimators, where the first CNN estimator was trained for 46 epochs and the other five subsequent CNN estimators were trained for one epoch.

Table 21 compares the accuracy of AdaBoost-CNN with CNN on the HAR dataset. The results show that AdaBoost-CNN increases the testing accuracy from 96.84% to 97.71%. Fig. 11 shows the accuracies of the methods for the five classes in the HAR dataset. For the first three classes the two methods have accuracy close to 100%. AdaBoost-CNN shows the most improvement to be for class 4. There are 395 samples in class 4, and the accuracy of the proposed method on this class is 4.56% higher than CNN (see Fig. 11).

Table 19 Accuracy of a single CNN on the HAR dataset for different numbers of learning epochs

Training accuracy (%)	Testing accuracy (%)	# Epochs
97.18	96.55	80
97.45	96.84	50
97.07	96.13	25

Table 20 Accuracy of the proposed AdaBoost-CNN for different numbers of estimators. The total number of learning epochs was 50.

Training accuracy (%)	Testing accuracy (%)	# Estimators
98.10	97.38	4
98.22	97.71	5
98.12	97.26	6

Table 21 Accuracy of the proposed AdaBoost-CNN and CNN on the HAR dataset.

Method	Training accuracy (%)	Testing accuracy (%)	# Estimators/ Epochs
Proposed AdaBoost-CNN	98.22	97.71	5 estimator
CNN	97.45	96.84	50 Epochs

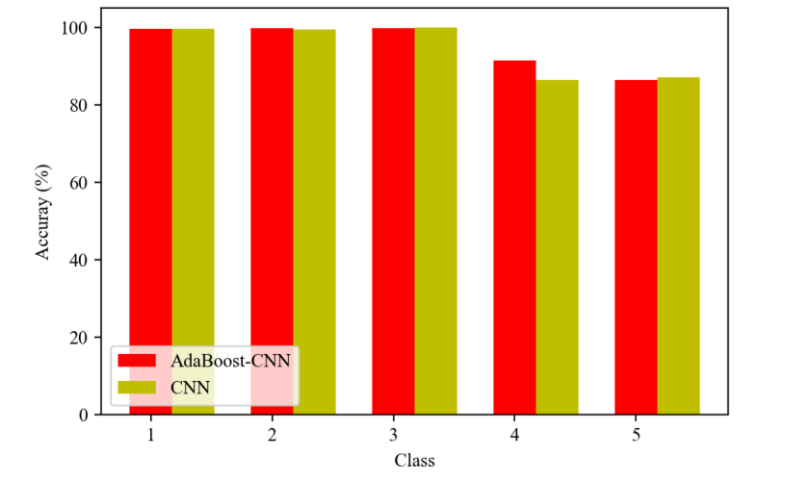

Fig. 11. Accuracies of the proposed AdaBoost-CNN on different classes are compared to those of CNN on the HAR dataset.

4.3.6. Effect of different levels of imbalance in the training data using the Synthetic Data

In this section the performance of AdaBoost-CNN is investigated when the degree of imbalance in the training data changes using the Synthetic dataset described in section 4.1.1. The synthetic data contains three classes. Initially, class 1 contains 800 cases, class 2 contains 1,000 cases, and class 3 also contains 1,000 cases. To evaluate the impact of imbalance on the performance of CNN and AdaBoost-CNN, the number of cases in class 2 are modified, whilst the number of cases in classes 1 and 3 remain stable. For example, when the number of training samples of different classes change from [800, 1,000, 1,000] to [800, 900, 1,000], the imbalance of the data increases. Table 22 shows the training and testing accuracies for 10 different experiments, where a different imbalanced dataset with a different number of training samples is used in each experiment. The last column of Table 22 shows the number of training data contained in each class.

Considering the results of both CNN and AdaBoost-CNN, as shown in Table 22, in experiments 1 to 2, the degree of imbalance increases and the training and testing accuracies reduce. In experiment 3, where the dataset is more balanced the accuracies increase. In experiment 3, class 2 and class 3 have the same number of samples, making it a more balanced dataset. In experiments 5 and onwards, when the difference between the number of samples between class 2 and 3 becomes larger, the performance of the CNN diminishes, compared to AdaBoost-CNN. In particular, after experiment 4, the difference

between the testing accuracy of the CNN and AdaBoost-CNN increases, with AdaBoost-CNN clearly outperforming CNN in all the experiments. Difference in performance reached its highest in experiment 10, when the degree of imbalance was increased, i.e. by changing the number of cases in class 2 from 1,000 to 100, and this resulted in 20.66% higher accuracy when using AdaBoost-CNN compared to when using the CNN.

The results in Table 22 reveal that data imbalance affects the accuracy of the CNN, and that improvements in CNN are required so it can effectively classify imbalance data. The last experiment shows the results of the datasets with the largest degree of imbalance. Comparing the results of experiments 1 and 10 testing accuracy of CNN is reduced from 93.85% to 72.37%, whereas for AdaBoost-CNN the reduction was minor from 93.94% to 93.03%. Comparing the average performance across the experiments of the two methods, CNN achieved 89.27% whereas AdaBoost-CNN achieved 93.77%.

Fig. 12 shows the testing accuracy improvement when using AdaBoost-CNN compared to the single CNN when increasing the difference in the number of samples contained in the imbalanced dataset. Fig. 12 shows that when the level of imbalance is low in the training data the accuracy of the single CNN is close to that of the proposed method. However, the proposed method gives higher accuracy compared to the CNN when the degree of imbalance in the data increases. Fig. 13 compares the testing accuracies of AdaBoost-CNN and CNN for different levels of imbalance. Fig. 13 reveals that the proposed method has higher accuracy for higher imbalance in the training data.

Table 22. Accuracy values of single CNNs and the AdaBoost-CNN on the different datasets. The number of training data in the second class is reduced from 1,000 to 100 to generate different levels of imbalance.

No.	**CNN**		**Proposed AdaBoost-CNN**		**Diff. in Training accuracy (%)**	**Diff. in Testing accuracy (%)**	**# training samples in the three classes**
	Training accuracy (%)	**Testing accuracy (%)**	**Training accuracy (%)**	**Testing accuracy (%)**			
1.	96.36	93.85	96.00	93.94	-0.36	+0.09	[800, 1,000, 1,000]
2.	94.19	92.65	94.26	92.71	+0.07	+0.06	[800, 900, 1,000]
3.	95.85	93.84	95.69	94.03	-0.16	+0.19	[800, 800, 1,000]
4.	95.24	92.76	94.20	93.66	-1.04	+0.9	[800, 700, 1,000]
5.	95.63	92.09	95.50	93.81	-0.13	+1.72	[800, 600, 1,000]
6.	95.74	92.05	95.61	94.08	-0.13	+2.03	[800, 500, 1,000]
7.	96.36	91.49	95.55	94.10	-0.81	+2.61	[800, 400, 1,000]
8.	95.71	88.65	96.33	94.18	+0.62	+5.53	[800, 300, 1,000]
9.	95.95	82.90	97.00	94.20	+1.05	+11.3	[800, 200, 1,000]
10.	96.16	72.37	97.73	93.03	+1.57	+20.66	[800, 100, 1,000]
Average	95.71	89.27	95.79	93.77	+0.07	+4.51	

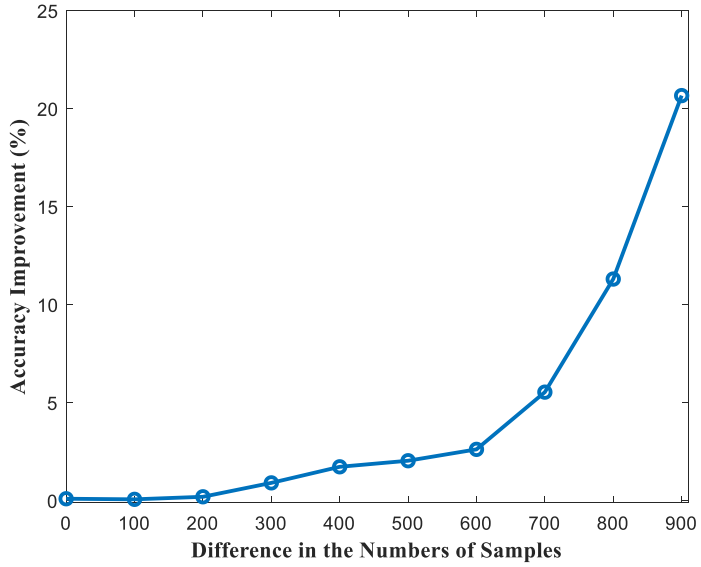

Fig. 12 Testing accuracy improvement of the proposed method is increased when the level of imbalance, i.e. the difference between the number of samples in class 2 and 3, in the data is increased.

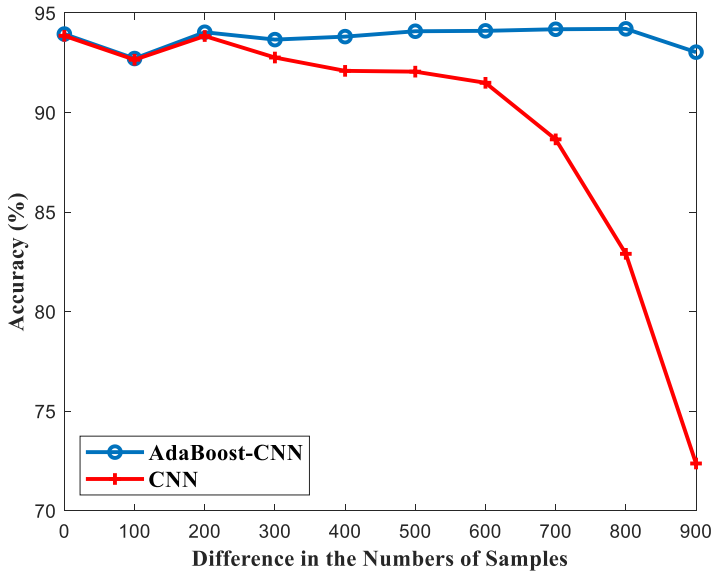

Fig. 13. Testing accuracies of the proposed AdaBoost-CNN and CNN for different levels of imbalance, i.e. the difference between the number of samples in class 2 and 3.

5. Conclusion

In this paper a multi-class AdaBoost for CNN, called AdaBoost-CNN, is proposed. In the proposed method a number of CNNs are used as base estimators. The CNNs are trained sequentially. The errors of an earlier CNN are used to update the sample weights for its next CNN. After updating the sample weights, the trained CNN learning parameters are transferred to the next CNN. Transfer learning in the proposed AdaBoost method increases the accuracy. The updated sample weights are used during training of the next CNN. The training sample weights are incorporated to the cross-entropy error

function in the CNN back propagation learning algorithm. The ability of the proposed AdaBoost-CNN in the processing imbalanced data is tested.

The proposed method is a modified version of the traditional AdaBoost to make AdaBoost compatible with deep learning. The proposed AdaBoost-CNN is designed to be trained on large data by taking the same number of learning epochs which is used by a single CNN estimator, and can still achieve higher accuracy than the single CNN.

The high number of training samples in large data, results in high computation cost on each learning epoch and a specific AdaBoost method is required to be applied when compared to smaller datasets. Additionally, deep learning methods comprise a high number of learning parameters which should be adjusted in each training epoch. Consequently reducing the number of training epochs of all estimators is more crucial when working with deep learning methods to process large data.

In this paper the knowledge acquired during training of a CNN estimator is transferred to the next estimator instead of initializing an estimator with random learning parameters. Consequently, the training parameters of CNN estimators are initially set in an appropriate state and they are trained in one epoch on the weighted training samples. However, the method proposed by Yang et al. [40] does not consider the high number of training parameters of deep learning methods and the high computation cost of big data processing, and instead of reducing the number of learning epochs it actually increases the number of training epochs by repeating the training procedure on the trained estimators. This of course increases the required computational effort as compared to the approach reported herein.

A multi-dimensional standard normal distribution is used to generate synthetic dataset for different classes in which samples in different classes are separated by nested concentric multi-dimensional spheres. The performance of the proposed AdaBoost-CNN is compared to the classical AdaBoost with a decision tree classifier on the synthetic dataset. The proposed method outperformed the classical AdaBoost with Decision Tree by 16.98%. Moreover, AdaBoost-CNN gave a 2.03% improvement in testing accuracy compared to the base CNN. The accuracy of CNN was 92.05%, whereas the proposed method achieved an accuracy of 94.08%. Additionally, two well-known datasets, CIFAR-10 and Fashion-MNIST, were used to test the accuracy of the proposed method. AdaBoost-CNN outperformed the single CNN by 3.51% on CIFAR-10, and misclassified 136 fewer samples compared to the CNN on the Fashion-MNIST test set.

AdaBoost-CNN was also applied to two EMNIST imbalanced datasets (i.e. EMNIST by-class and EMNIST by-merge) which are large datasets compared to the well-known dataset MNIST. The experimental results show that the proposed AdaBoost-CNN improves the overall accuracy on EMNIST datasets, and across the various imbalanced classes. Importantly, the accuracy of the proposed method on one of the classes is more than 8 times higher than the accuracy of CNN on the same class. Additionally, the proposed method improved accuracy on the EMNIST by-merge dataset. The effect of different levels of imbalance in the training data was also investigated. Experiment results on the synthetic data revealed that the improvement of accuracy of AdaBoost-CNN can be more than 20% for highly imbalanced data on 10,000 testing samples. The performance of AdaBoost-CNN on multi-modal data analysis will be investigated in future research where different modalities of data will be trained by different CNNs.

Acknowledgment

The work was funded by The Leverhulme Trust Research Project Grant RPG-2016-252 entitled “Novel Approaches for Constructing Optimised Multimodal Data Spaces”.

References

[1] W. Lee, C. H. Jun, and J. S. Lee, "Instance categorization by support vector machines to adjust weights in AdaBoost for imbalanced data classification," *Inf. Sci. (Ny).*, vol. 381, pp. 92–103, 2017.

[2] J. Zhu, H. Zou, S. Rosset, and T. Hastie, "Multi-class AdaBoost," *Stat. Interface*, vol. 2, no. 3, pp. 349–360, 2009.

[3] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera, "Ordering-based pruning for improving the performance of ensembles of classifiers in the framework of imbalanced datasets," *Inf. Sci. (Ny).*, vol. 354, pp. 178–196, 2016.

[4] N. H. C. Lima, A. D. D. Neto, and J. D. De Melo, "Creating an ensemble of diverse support vector machines using Adaboost," *Proc. Int. Jt. Conf. Neural Networks*, pp. 1802–1806, 2009.

[5] M. Buda, A. Maki, and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks," pp. 1–23, 2017.

[6] P. Viola and M. J. Jones, "Robust Real-time Object Detection," *Int. J. Comput. Vis.*, no. February, pp. 1–30, 2001.

[7] S. Nejatian, H. Parvin, and E. Faraji, "Using sub-sampling and ensemble clustering techniques to improve performance of imbalanced classification," *Neurocomputing*, vol. 0, pp. 1–12, 2017.

[8] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, pp. 23–37, 1997.

[9] Z. Sun, Q. Song, X. Zhu, H. Sun, B. Xu, and Y. Zhou, "A novel ensemble method for classifying imbalanced data," *Pattern Recognit.*, vol. 48, no. 5, pp. 1623–1637, 2015.

[10] S. Vucetic and Z. Obradovic, "Classification on Data with Biased Class Distribution," in *Machine Learning: ECML 2001*, 2001, pp. 527–538.

[11] C. Chen, A. Liaw, and L. Breiman, "Using random forest to learn imbalanced data," *Univ. California, Berkeley*, no. 1999, pp. 1–12, 2004.

[12] J. Duchi and H. Namkoong, "Variance-based regularization with convex objectives," in *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2016, no. 3, pp. 1–10.

[13] D. Masko and P. Hensman, "The Impact of Imbalanced Training Data for Convolutional Neural Networks," *Bachelor thesis, KTH, Sch. Comput. Sci. Commun.*, 2015.

[14] J. Wang, X. Yang, H. Cai, W. Tan, C. Jin, and L. Li, "Discrimination of Breast Cancer with Microcalcifications on Mammography by Deep Learning," *Sci. Rep.*, vol. 6, no. June, pp. 1–9, 2016.

[15] "ImageNet Large Scale Visual Recognition Challenge 2015. ImageNet. ILSVRC2015," 2015. [Online]. Available: http://image-net.org/challenges/LSVRC/2015/.

[16] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column Deep Neural Networks for Image Classification," *CVPR*, pp. 3642–3649, 2012.

[17] X. Frazão and L. Alexandre, "Weighted Convolutional Neural Network Ensemble," *Prog. Pattern Recognition, Image Anal. Comput. Vision, Appl.*, vol. 8827, no. Springer, Cham, pp. 674–681, 2014.

[18] G. Wen, Z. Hou, H. Li, D. Li, L. Jiang, and E. Xun, "Ensemble of Deep Neural Networks with Probability-Based Fusion for Facial Expression Recognition," *Cognit. Comput.*, vol. 9, no. 5, pp. 597–610, 2017.

[19] B.-K. Kim, J. Roh, S.-Y. Dong, and S.-Y. Lee, "Hierarchical committee of deep convolutional neural networks for robust facial expression recognition," *J. Multimodal User Interfaces*, vol. 10, no. 2, pp. 173–189, 2016.

[20] Y. Kawana, N. Ukita, J. Bin Huang, and M. H. Yang, "Ensemble convolutional neural networks for pose estimation," *Comput. Vis. Image Underst.*, no. December 2017, pp. 0–1, 2018.

[21] H. Wang *et al.*, “Cascaded ensemble of convolutional neural networks and handcrafted features for mitosis detection,” *Proc. SPIE 9041, Med. Imaging 2014 Digit. Pathol.*, vol. 9041, p. 90410B, 2014.

[22] N. Tajbakhsh, S. R. Gurudu, and J. Liang, “Automatic Polyp Detection in Colonoscopy Videos Using an Ensemble of Convolutional Neural Networks,” in *IEEE 12th International Symposium on Biomedical Imaging (ISBI)*, 2015, pp. 79–83.

[23] E. P. Ijjina and C. Krishna Mohan, “Hybrid deep neural network model for human action recognition,” *Appl. Soft Comput. J.*, vol. 46, pp. 936–952, 2016.

[24] M. Lyksborg, O. Puonti, M. Agn, and R. Larsen, “An Ensemble of 2D Convolutional Neural Networks for Tumor Segmentation BT - Image Analysis,” 2015, pp. 201–211.

[25] A. Krizhevsky, I. Sutskever, and H. Geoffrey E., “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems 25 (NIPS2012)*, 2012, pp. 1–9.

[26] K. A, S. I, and H. GE, “ImageNet Classification with Deep Convolutional Neural Networks,” in *In: Advances in neural information processing systems*, 202AD.

[27] A. Krizhevsky, “Learning multiple layers of features from tiny images,” 2009.

[28] A. Krizhevsky, V. Nair, and G. Hinton, “The CIFAR-10 dataset,” *online http//www. cs. toronto. edu/kriz/cifar. html*, 2014.

[29] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms,” pp. 1–6, 2017.

[30] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, “EMNIST: Extending MNIST to handwritten letters,” *Proc. Int. Jt. Conf. Neural Networks*, vol. 2017-May, pp. 2921–2926, 2017.

[31] Amine El Helou, “Sensor HAR recognition App - File Exchange - MATLAB Central.” [Online]. Available: https://uk.mathworks.com/matlabcentral/fileexchange/54138-sensor-har-recognition-app. [Accessed: 03-Jul-2018].

[32] “Human Activity Recognition Simulink Model for Smartphone Deployment - MATLAB & Simulink - MathWorks United Kingdom.” [Online]. Available: https://uk.mathworks.com/help/supportpkg/android/examples/human-activity-recognition-simulink-model-for-smartphone-deployment.html. [Accessed: 03-Jul-2018].

[33] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[34] A. Krizhevsky, “Learning Multiple Layers of Features from Tiny Images,” *... Sci. Dep. Univ. Toronto, Tech. ...*, pp. 1–60, 2009.

[35] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.

[36] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, “Deep learning for time series classification: a review,” *Data Min. Knowl. Discov.*, 2019.

[37] N. V Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE : Synthetic Minority Over-sampling Technique,” vol. 16, pp. 321–357, 2002.

[38] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, “Understanding deep learning requires rethinking generalization,” 2016.

[39] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, “The balanced accuracy and its posterior distribution,” in *International Conference on Pattern Recognition The*, 2010, pp. 3125–3128.

[40] S. Yang, L. Chen, T. Yan, Y. Zhao, and Y. Fan, “An ensemble Classification Algorithm fro Convolutional Neural Network based on AdaBoost,” pp. 401–406, 2017.