https://doi.org/10.31891/csit-2025-1-12
UDC 004.94

Lesia MOCHURAD, Khrystyna DOLYNSKA, Tetiana UFIMTSEVA
Lviv Polytechnic National University

SCALABLE PARALLEL TRAINING OF DEEP NEURAL NETWORKS FOR IMAGE CLASSIFICATION USING TENSOR PROCESSING UNITS

Image classification using machine learning techniques is crucial in fields such as medicine, ecology, and agriculture, where large datasets of images need to be processed efficiently. However, traditional deep learning methods can be computationally expensive, particularly when handling massive amounts of data. This paper proposes a novel parallel training approach for deep neural networks using multiple Tensor Processing Units (TPUs) with the TensorFlow tf.distribute.Strategy API, aimed at solving the scalability issue in image classification tasks. The primary advantage of this approach is the ability to parallelize the training process without altering the model architecture, ensuring both flexibility and efficiency. By distributing the workload across multiple TPUs, the algorithm accelerates training significantly, enabling faster model convergence. Numerical experiments comparing the proposed parallel training method on 8 TPUs with a traditional sequential approach on a single Graphics Processing Unit (GPU) show that parallel training reduces training time by a factor of 4.6 while maintaining the classification accuracy achieved in sequential training. This demonstrates that the parallelized method not only speeds up the process but also retains model performance. The proposed algorithm has shown high scalability, making it suitable for processing large datasets. This scalability is particularly beneficial for tasks requiring rapid processing of large volumes of image data, such as real-time applications in environmental monitoring or wildlife research. In conclusion, parallel machine learning methods present a promising solution for improving the speed and efficiency of image classification tasks. Future research can focus on further optimizing the scalability of this approach and enhancing its performance for even larger datasets, as well as its application in time-sensitive real-world scenarios.

Keywords: image classification, data parallelism, TPU, acceleration, transfer learning, machine learning.

Леся МОЧУРАД, Христина ДОЛИНСЬКА, Тетяна УФІМЦЕВА
Національний університет «Львівська політехніка»

МАСШТАБОВАНЕ ПАРАЛЕЛЬНЕ НАВЧАННЯ ГЛИБОКИХ НЕЙРОННИХ МЕРЕЖ ДЛЯ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ ІЗ ВИКОРИСТАННЯМ ТЕНЗОРНИХ ПРОЦЕСОРІВ

Класифікація зображень з використанням методів машинного навчання є надзвичайно важливою в таких сферах, як медицина, екологія та сільське господарство, де потрібно ефективно обробляти великі масиви зображень. Проте традиційні методи глибокого навчання можуть бути обчислювально дорогими, особливо при роботі з великими обсягами даних. У цій роботі запропоновано новий підхід до паралельного навчання глибоких нейронних мереж з використанням кількох тензорних процесорних одиниць (TPU) за допомогою API tf.distribute.Strategy в TensorFlow, що дозволяє вирішити проблему масштабованості при класифікації зображень, зокрема для ідентифікації видів птахів. Основною перевагою цього підходу є можливість розпаралелювати процес навчання без зміни архітектури моделі, що забезпечує її універсальність та ефективність. Розподіляючи навантаження між кількома TPU, алгоритм значно пришвидшує навчання, забезпечуючи більш швидку збіжність моделі. Чисельні експерименти, порівнюючи запропонований метод паралельного навчання на 8 TPU з традиційним послідовним методом навчання на одному графічному процесорі (GPU), показали, що паралельне навчання скорочує час тренування в 4.6 рази, при цьому зберігаючи точність класифікації, досягнуту при послідовному навчанні. Це свідчить про те, що паралельний метод не лише пришвидшує процес, але й зберігає ефективність моделі. Запропонований алгоритм продемонстрував високу масштабованість, що робить його придатним для обробки великих обсягів даних. Така масштабованість особливо корисна для задач, що потребують швидкої обробки великих обсягів зображень, таких як реальні застосування в моніторингу довкілля або дослідженнях дикої природи. На завершення, паралельні методи машинного навчання є перспективним рішенням для покращення швидкості та ефективності класифікації зображень. Подальші дослідження можуть бути спрямовані на подальшу оптимізацію масштабованості цього підходу та підвищення його продуктивності для ще більших наборів даних, а також на його застосування в реальних часозалежних сценаріях.

Ключові слова: класифікація зображень, паралелізм даних, тензорні процесорні одиниці, прискорення, трансферне навчання, машинне навчання.

Introduction

Image classification using machine learning methods is widely used in various fields, such as medicine, satellite image analysis, agriculture, and environmental research [1], [2], [3]. These technologies allow automating recognition and analysis processes, which helps to increase accuracy and efficiency in the respective industries. For example, they are used in medicine to diagnose diseases [4], in satellite monitoring to quickly detect natural disasters [5], and in agriculture to identify bird species that may affect crop yields or biodiversity [6].

As the amount of training data and model complexity increases, the computational cost of training neural networks increases significantly. One of the effective solutions to this problem is the use of parallel computing methods that allow load distributing among several computing devices, such as multi-core processors, graphics processing units (GPUs), and tensor processing units (TPUs). This significantly reduces model training time and makes their application more practical in real-world conditions [7].

In today's big data-driven world, the speed and efficiency of training deep learning models are critical for many applications, including autonomous vehicles and real-time decision-making systems. The use of parallel algorithms can significantly speed up data processing, which is especially important for tasks related to real-time object classification [8]. For example, NVIDIA research confirms that the use of parallelized approaches helps to increase the speed of detection and decision-making in autonomous systems [9], [10], [11].

In the context of bird classification, parallel neural network training opens up new opportunities for environmental monitoring, agricultural research, and conservation [12]. Automatic species recognition helps to track changes in populations, analyze ecosystem processes, and assess the impact of the environment on biodiversity. At the same time, accelerating the model training process allows for the rapid processing of large amounts of data, which is important for scientific research and applied tasks.

This paper aims to develop and implement a method of parallel model training for image classification using tensor processing units. The study involves evaluating the efficiency and accuracy of the developed approach in comparison with traditional training methods. It is expected that the use of parallel computing will significantly reduce the training time without losing the quality of classification.

The novelty of the study lies in the use of modern distributed learning technologies, in particular the tf.distribute.Strategy API, which provides flexibility and portability when scaling neural networks. This allows the algorithms to be adapted to different hardware platforms, including central processing unit (CPU), GPU, and TPU, making the method more versatile. This approach improves the performance of model training and allows for the effective application of deep learning for scalable image classification.

Related works

Given the importance of parallelization in data processing and model training, it should be noted that this problem is already being actively studied by the scientific community. An analysis of existing approaches to parallel learning allows us to evaluate their advantages and limitations, as well as to determine the optimal method for solving the problem posed in this paper.

One such approach is the model presented in [5]. The authors proposed a new Convolutional Neural Network (CNN) architecture that combines several parallel CNNs with a Back Propagation Neural Network (BPNN). The study was conducted on the CIFAR-10, CIFAR-100, and MNIST datasets, and the results proved to be promising of this approach. The architecture of the model involved combining several parallel CNN blocks, which were integrated into fully connected layers after convolutional operations. The analysis of the results showed that increasing the number of convolutional layers in a traditional sequential CNN improved the test accuracy, but the optimal performance was achieved by using several parallel CNN units with different depths of convolutional layers. Despite the results achieved, the proposed architecture requires further improvement. In particular, the authors note the possibility of improving performance by fine-tuning the model, in particular through the use of Batch normalization, data augmentation, and weight regularisation.

In [13], a hybrid deep learning architecture for classifying histopathological images for cancer detection was presented. The proposed approach combines Residual CNN, VGG16, and MobileNet architectures that were pre-trained on the ImageNet dataset. The methodology includes preliminary image normalisation, three times augmentation of the data in the training set, and the use of transfer learning to improve the generalisation ability of the models. A key feature of the proposed architecture is the modification of existing models by adding a parallel layer of long short-term memory (LSTM), which allows for the creation of a hybrid neural network. The training is performed using the TensorFlow-GPU framework, which provides accelerated computation and efficient use of computing resources. The study developed hybrid architectures combining LSTM models with ResNet, MobileNet, and VGG16 with 101 and 152 layers. Experimental results have shown that the proposed hybrid architectures, in particular ResNet101 and ResNet152, achieve an accuracy of up to 92%, which indicates the effectiveness of the proposed approach in histopathological image classification tasks.

The paper [14] analyses three different architectures of CNNs: LeNet, VGG, and ResNet. The authors investigate the accuracy of image classification for each of the considered models and evaluate the efficiency of their training. To speed up the training, the authors used the CUDA architecture and GeForce RTX 1080 Ti, GeForce RTX 2080 Ti, and GeForce RTX 3090 GPUs. The results of the study showed that the neural network training speed is directly proportional to the computing power of the graphics card. In particular, the training speed increased by 5% and 2% for LeNet, by 12.3% and 3.8% for ResNet18, and by 27% and 20% for VGG16, depending on the hardware used. At the same time, the influence of the GPU choice on the final model accuracy was insignificant. Instead, the training environment had a more significant impact on the classification quality than the model architecture itself. In addition to using CUDA, the authors investigated the effectiveness of parallel neural network training on 2, 4, and 8 GeForce RTX 3090 GPUs using data parallelisation. However, the results showed that an increase in the number of graphics cards does not always lead to the expected acceleration, as the training speed decreased significantly due to the increased cost of data exchange between computing devices.

Another approach to parallel CNN processing is presented in [15]. The study was aimed at developing an efficient parallel CNN model for high-precision terrain classification from satellite images. To achieve this goal, the authors modified the AlexNet architecture, which in its original form consists of twelve layers: five convolutional

layers, three maximum pooling layers, three fully connected layers, and one classification layer. In the proposed model, an additional convolutional layer was added in each of the parallel network blocks, which improved the extraction of image features. Additionally, image filtering was implemented using a convolutional layer with two channels. The proposed parallel model improved the classification accuracy by 1.44%. At the same time, the analysis time of the parallel CNN model increased to 78 minutes compared to 76 minutes for the original architecture.

The literature analysis shows that a significant number of studies focus on modifying the architecture of neural networks to improve their accuracy and accelerate training. Some approaches also consider the use of more powerful GPUs to improve training efficiency. In this paper, we aim to develop an algorithm for parallel model training without modifying its architecture, as well as to propose a parallelization option for cases where the use of multiple GPUs or a more powerful GPU is not possible.

Methodology

Parallelization

There are two main methods of distributed learning: data parallelism and model parallelism [16]. In this work, we used a data parallelization approach with synchronous training, as the goal of this work is to speed up the training of a classification model for a large number of bird images (84635 training images). During synchronous training, each device receives its piece of data and has a full copy of the model. The process starts simultaneously on all devices, which calculate different input data and gradients. After that, the devices communicate with each other and combine the gradients using a full reduction algorithm. The merged gradients are sent back to all devices, where each device updates its local copy of the scale as normal. The process is synchronous because, at each step, all devices have the same weights, even though they were trained on different data and generated different gradients.

To parallelize the model, we used tensorflow. Distribute. Strategy API to distribute training across multiple GPUs, multiple machines, or TPUs. This API allows you to parallelize existing models and training code with minimal code changes. There are several main parallelization strategies, including MirroredStrategy, TPUStrategy, MultiWorkerMirroredStrategy, ParameterServerStrategy, and CentralStorageStrategy. In this paper, we have parallelized the training of the classifier using TPUStrategy on a different number of TPUs.

This strategy is based on the Single Programme, Multiple Data (SPMD) parallelism model. SPMD involves the simultaneous execution of the same program on multiple datasets. The same model is replicated across all cores (TPUs in this work), and each core processes a separate dataset independently. The gradients computed by each core are then aggregated to update the model parameters. Google Cloud TPUs are specially designed artificial intelligence accelerators optimized for training large artificial intelligence models. Cloud TPUs are designed to scale cost-effectively for a wide range of AI workloads, including training, fine-tuning, and results. Cloud TPUs provide the versatility to accelerate workloads on leading AI frameworks, including PyTorch, JAX, and TensorFlow. Unlike cloud GPUs, a TPU is a specialized application-specific integrated circuit (ASIC) designed by Google for neural networks. TPUs have specialized features, such as a matrix multiplication unit (MXU) and a proprietary interconnect topology, making them ideal for accelerating AI training.

To parallelize the model training between several TPUs connected via Google Colab, we used tf.distribute.TPUStrategy. With the help of TPUClusterResolver, the program establishes a connection to the TPUs cluster by finding available devices. After that, tf.config.experimental_connect_to_cluster is executed to connect to the cluster. Next, the TPU system is initialized using tf.tpu.experimental.initialise_tpu_system. With the help of tf.distribute.TPUStrategy, a distribution strategy is created that provides the possibility of distributed training to all available TPUs in the cluster. As a result of these actions, we get a list of logical devices (TPUs) available for use in the cluster, the maximum number of which is 8.

During the experiments, a different number of TPUs were used, and the tf.tpu.experimental.DeviceAssignment class was used to set their number. When using allocation strategies, variables created within the strategy were duplicated across all replicas and synchronized using all-reduce algorithms. The all-reduce algorithm is used to synchronize global variables between model replicas running in parallel on different devices. The basic idea is that each replica reports its local changes to all other replicas and performs a reduction operation (e.g., sum or average) on all received values. This ensures global consistency between replicas and increases the speed and efficiency of parallel model training. For example, when training a deep neural network on a cluster of TPUs, the full reduction algorithm synchronizes the model weights on different TPUs to avoid training mismatches and increase the training speed.

Data loading, model loading, and compilation take place in the visibility of the block with strategy.scope(), to execute the code in the context of the TPUStrategy distributed learning strategy. This allows building and training the model in parallel on multiple TPU devices. Next, the methods from the tf.keras.Model APIs were used to train and evaluate the model, namely fit() with the training and validation dataset as arguments and predict() with the test dataset as arguments.

Proposed parallel training algorithm

1. Importing the necessary libraries. At the initial stage, all the necessary libraries are imported, including TensorFlow for model building and training, NumPy for working with datasets, Matplotlib for visualization, as well as additional libraries for image processing and analysis.
2. Data loading and preliminary processing. The training, validation, and test datasets are loaded into the appropriate directories using tf.keras.utils.image_dataset_from_directory. Additionally, the birds.csv file containing aggregated information about the dataset, including class names and distribution, is loaded.
3. Data verification and correction. Before starting training, the quality of the data is checked. Selected images are visualized to assess their quality, statistical information about the number of images in each class is reviewed, and possible errors in the birds.csv file, including incorrect class names, are corrected.
4. Data augmentation. To increase the model's robustness to changes in the data and increase the diversity of the training set, augmentation methods such as random reflection, brightness, contrast, and image scaling are used.
5. Preparing data for training. The data is converted to TFRecord format, which allows you to work efficiently with large image sets. Prepared TFRecord files are written to the Google Cloud Storage bucket. To represent the category labels, sklearn.preprocessing.LabelEncoder is used, which converts each of the 525 classes into a numerical value, which facilitates model training.
6. Creating a model architecture. The function for building a convolutional neural network keras.Sequential is declared. The model architecture includes convolutional (Conv2D), normalization (BatchNormalization), activation (ReLU, Softmax), pooling (MaxPooling2D), and fully connected (Dense) layers.
7. Distribution of computations between devices. For the efficient use of computing resources, the model training process takes place within with strategy.scope(), which ensures the automatic distribution of tasks between the selected devices (TPU or GPU). The same block synchronizes the weights between devices, as well as downloads data from the Google Cloud Storage bucket using num_parallel_reads=AUTOTUNE. This allows TensorFlow to optimize data reading performance depending on the available computing resources.
8. Model training. The training process is implemented in parallel mode using model.fit(), which allows efficient use of available computing power. The training history is stored in the history variable for further analysis of the results.
9. Analysis of training results. To evaluate the stability of training, graphs of training curves are plotted based on the loss and accuracy values obtained from history. Visualization helps to identify possible problems, such as overtraining or insufficient training of the model.
10. Evaluation of model performance. The model is tested on a separate dataset, where the accuracy, recall, precision, and F1-score metrics are calculated. In addition, the confusion matrix is visualized, which allows to evaluation of the classification performance for each class separately.

Figure 1 presents a flowchart of the proposed algorithm, illustrating its key processing steps and decision points.

The proposed algorithm ensures efficient training of a deep neural network without changes in its architecture, using parallelization technologies. The implementation of parallel training on multiple computing devices (TPU/GPU) can significantly reduce the processing time of large image sets, which is critical for computer vision tasks.

Data analysis and preprocessing

For the image classification task of bird species, the BIRDS 525 SPECIES dataset [17] was selected. This dataset comprises 525 bird species and contains approximately 90,000 images, each in JPEG format with a resolution of 224 × 224 pixels in color. The dataset was split into training, validation, and test sets, with the training set consisting of 84,635 images (94%), while the validation and test sets each contained 2,625 images (3%). While the training set exhibits some class imbalance, both the validation and test sets maintain uniform class distributions, with five images per species.

To enhance model generalization, data augmentation techniques were applied:

- Horizontal flipping (horizontal_flip=True): Random 50% probability flip to improve robustness to orientation.
- Width and height shifting (width_shift_range=0.05, height_shift_range=0.05): Random shifts up to 5% of image dimensions for spatial invariance.
- Rotation (rotation_range=18): Random rotations within ±18 degrees to simulate different perspectives.
- Zooming (zoom_range=0.05): Up to 5% scaling variation to improve recognition across sizes.
- Brightness adjustment (brightness_range=[0.8, 1.2]): Random brightness changes within 80%–120% for illumination adaptation.
- Augmentation was implemented using ImageDataGenerator from Keras.

Given the use of Tensor Processing Units (TPUs) for training, efficient data loading was essential. TPUs process data at high speeds, requiring optimized input pipelines to prevent bottlenecks. Since cloud storage (Google

Cloud Storage, GCS) introduces connection overhead, storing individual images as thousands of files is inefficient. Instead, images were combined into fewer files using TFRecord format, which enables parallel reading.

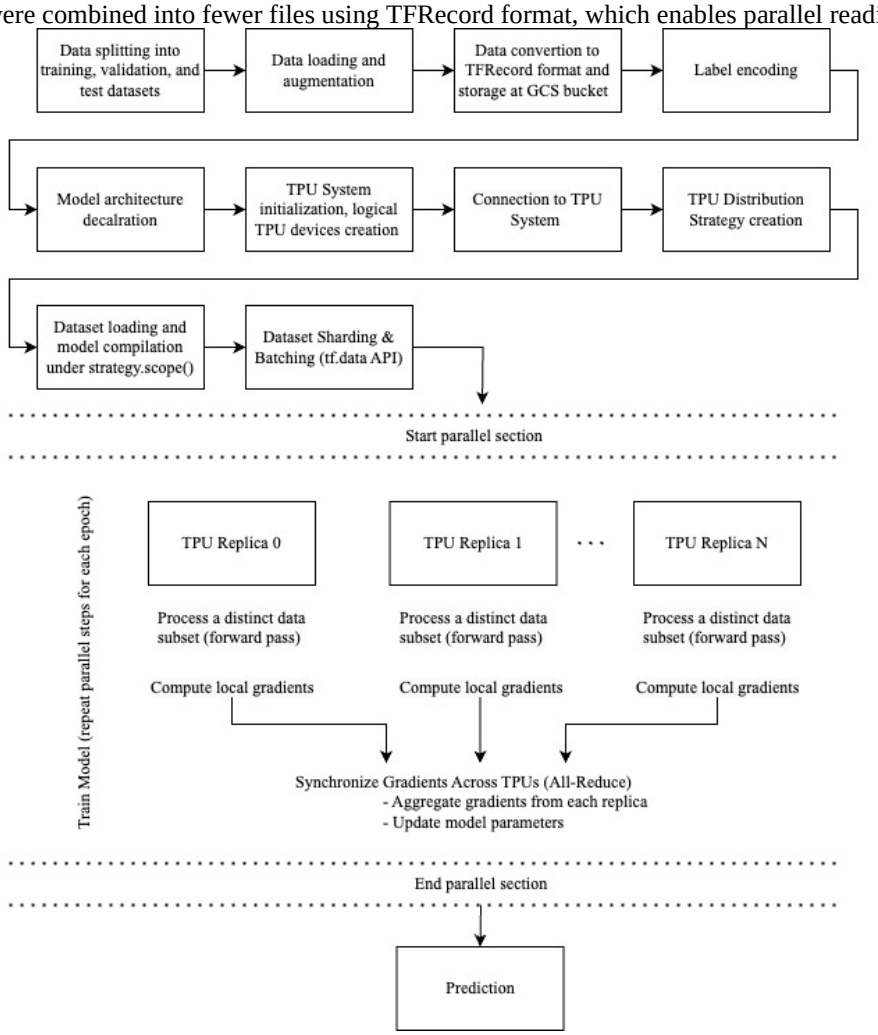

Fig. 1. Flowchart of the proposed algorithm

To ensure efficient training on TPUs, the dataset was stored in TFRecord format and uploaded to Google Cloud Storage (GCS) following these steps:

1. Creating a GCS bucket named birds_trspo_cli.
2. Uploading preprocessed images (training, validation, test sets) to the bucket.
3. Transforming images into TFRecord format, applying:
   - Size normalization for uniformity.
   - JPEG recompression to reduce storage and transmission overhead.
   - Parallel dataset transformations using .map() with num_parallel_calls=AUTOTUNE, allowing TensorFlow to dynamically optimize parallel operations based on available resources.
4. Structuring TFRecord files, ensuring each file contains a subset of the dataset, optimizing read performance.
5. Storing structured TFRecord files in the corresponding directories on GCS.

Once prepared, the dataset was loaded into tf.data.TFRecordDataset within the strategy.scope() execution context, enabling efficient parallel data processing for model training and evaluation.

Model architecture

The study in [18] conducted extensive experiments with various CNN architectures. Convolution is the process of applying a filter to input data, which results in activation. Repeated application of the same filter generates a feature map, highlighting the location of detected features in the input data, such as images. CNNs' key advantage is their ability to automatically learn numerous filters in parallel, optimizing for image classification tasks. This results in highly specific features that can be detected anywhere within an input image.

The most effective approach was transfer learning [19], a machine learning technique where a model trained for one task is reused as a starting point for another. This method is widely used in deep learning,

particularly for computer vision and natural language processing, as training deep neural networks requires substantial computational resources and time. The base model utilized in this study was EfficientNetB5, a CNN architecture known for its efficiency and high performance. As part of the EfficientNet family, it is larger and more powerful than its counterparts. Pretrained on the ImageNet dataset, which contains over 14 million labeled images across 21,841 classes, EfficientNetB5 captures a diverse range of generalizable features. With over 30 million parameters, it achieves state-of-the-art accuracy across various tasks. Its architecture includes convolutional layers, batch normalization, and activation functions, with a total depth of 566 layers. The scaling method in EfficientNetB5 balances model depth, width, and resolution, ensuring adaptability to different computational constraints while maintaining high accuracy.

The optimized model proposed in [18] was implemented in this study, with the following architecture:

1. Input: (224, 224, 3) RGB image
2. Data Augmentation
3. Pretrained Model: EfficientNetB5 for feature extraction
4. Dense (1024, activation="relu")
5. BatchNormalization()
6. Dropout (0.4)
7. Dense (512, activation="relu")
8. BatchNormalization()
9. Dropout (0.3)
10. Dense (256, activation="relu")
11. BatchNormalization()
12. Dropout (0.2)
13. Dense (525, activation="softmax")

The final dense layer contains as many neurons as the number of classes in the dataset and employs the softmax activation function. The Adam optimizer was used, as in [18], along with sparse categorical cross-entropy as the loss function. This loss function is particularly effective for multi-class classification tasks where each sample belongs to a single category.

Computational complexity analysis

The computational complexity of the model depends on its architecture, the number of layers, and the input-output data size. The complexity of each stage is evaluated as follows:

- Input: Image size of 224x224x3 pixels, represented as D = 224x224x3.
- Data Augmentation: Minimal computational cost compared to other stages and applied only during training.
- Pretrained Model: Since pretrained weights remain unchanged, this step is not considered in complexity analysis.
- Dense Layers: The complexity of a fully connected layer is
- $$O(N * M * D), \quad (1)$$

  where N – the number of dataset records, M – the number of neurons in the layer, and D – the input feature size.
- BatchNormalization and Dropout: Their complexity is negligible compared to dense layers.

Thus, the overall complexity is primarily determined by the dense layers

$$O(L * N * M * D), \quad (2)$$

where L – number of dense layers, N – number of dataset records, M – number of neurons per layer, D – input feature size per dense layer.

Parallelization and speedup analysis

Parallelization across multiple devices reduces computational complexity [20]. Ideally, execution time decreases by a factor of P, where P is the number of processors used. The speedup S_P is given by:

$$S_P = \frac{T_1(N)}{T_P(N)} = \frac{O(L * N * M * D)}{O(L * \frac{N}{P} * M * D)}, \quad (3)$$

where $T_1(N)$ – sequential execution time, $T_P(N)$ – parallel execution time with P processors.

The efficiency formula can be derived by considering the following: since the purpose of this metric is to determine how optimally the algorithm distributes work between devices and how well data is exchanged between processors, it is worth considering how many processors a machine contains when deriving the formula. So, the efficiency formula is E_P:

$$E_P = \frac{S_P(N)}{P} = \frac{O(L * N * M * D)}{P * O(L * \frac{N}{P} * M * D)}. \quad (4)$$

Experiments

Time costs of sequential and parallel implementations

The model was trained on 84,635 training images and 2625 validation images using eight TPUs. Training was performed for 10, 15, and 20 epochs using an early stopping mechanism that controlled validation losses with patience set to 12. Training for 15 epochs proved to be the most optimal, since with this number of epochs, the value of the val sparse categorical accuracy metric was close to 0.94, corresponding to the results of the first phase of training described in [18]. The average training time for one epoch was 130 seconds. The model quality assessment by metrics is presented in Table 1.

Table 1

Performance metrics of the classification model after 15 epochs of training on 8 TPU

Accuracy	Precision	Recall	F1 score
0.9622	0.9685	0.9622	0.9613

The training time of the model depends on the number of TPU devices used. Table 2 presents the duration of a single epoch (in seconds) for different numbers of TPU devices. Figure 2 illustrates the comparison of training time per epoch (in seconds) for model training on 1, 2, 4, and 8 TPU devices.

Table 2

Training time per epoch on TPU depending on the number of TPU devices

	Number of TPU			
	1	2	4	8
Time, s	237	176	152	130

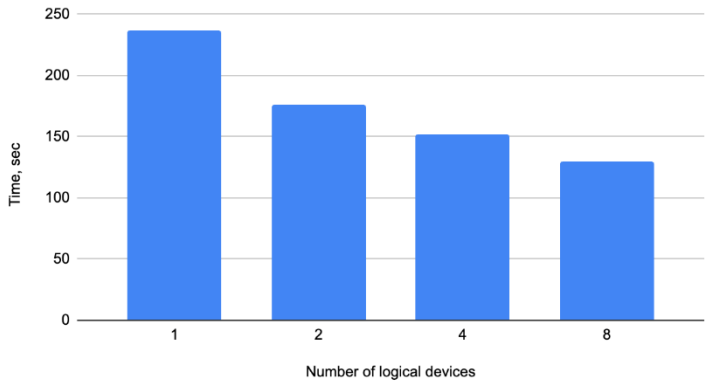

Fig. 2. Comparison of training time per epoch (in seconds) on 1, 2, 4, and 8 TPU devices

As shown in the time experiments in Figure 1, the fastest model training occurs with all available TPU devices. Therefore, we now analyze the time costs and accuracy during parallelization on 8 TPU devices (see Table 3). Figure 3 illustrates the learning curve showing the loss history for each epoch. Figure 4 presents the learning curve depicting the accuracy achieved for each epoch.

Table 3

Training time (in seconds) for 15 epochs on 8 TPU devices

Time, s	Number of Epochs	Average Time per Epoch, s	Accuracy
2084	15	130	0.9467

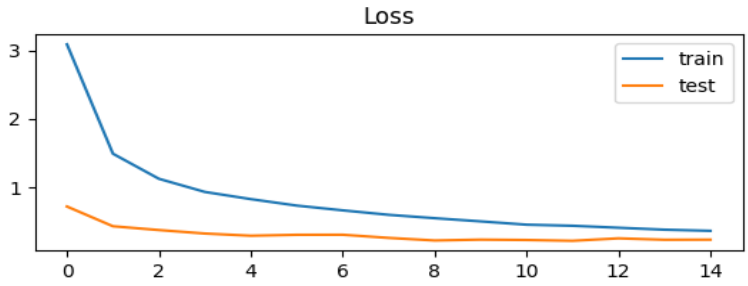

Fig. 3. Learning curve – Loss history per epoch

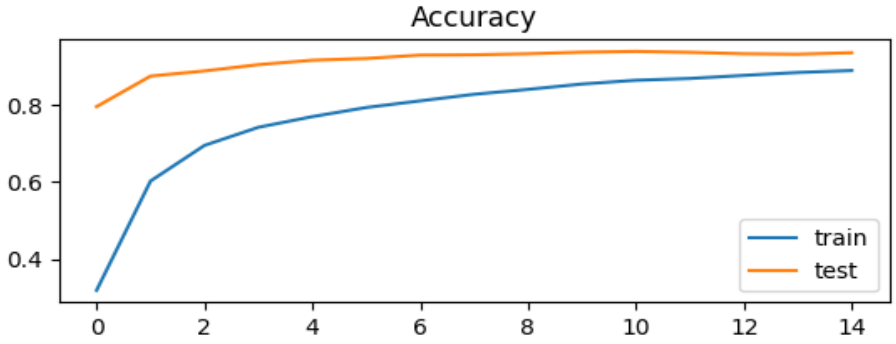

Fig. 4. Learning curve – Accuracy achieved per epoch

Analyzing Figure 3 and Table 3, it can be observed that to achieve the reference validation accuracy of 0.9463 with parallelization on 8 TPU devices, the training took 15 epochs instead of 50 when executed sequentially on a GPU. The time per epoch decreased by a factor of 4.6, while the validation accuracy remained unchanged. Therefore, it can be concluded that the model training was accelerated by a factor of 4.6.

Discussion of research results

Acceleration and Efficiency in Parallelization

The apriori estimation of acceleration is as follows:

$$S_P = \frac{T_1(N)}{T_P(N)} = \frac{O(L * N * M * D)}{O(L * \frac{N}{P} * M * D)}, \quad (5)$$

where $T_1(N)$ – is the time complexity of sequential execution of the algorithm; $T_P(N)$ – is the time complexity of parallel execution on P processors.

Thus, it is expected that the acceleration of the model will approach the number of processors used for parallelization. The actual acceleration achieved with parallelization on TPU devices is summarized in Table 4 and visualized in Figure 5.

Table 4

Actual acceleration metrics of the parallel algorithm depending on the number of TPU devices used

	Number of TPU			
	1	2	4	8
S_P	1	1.3466	1.5592	1.8231

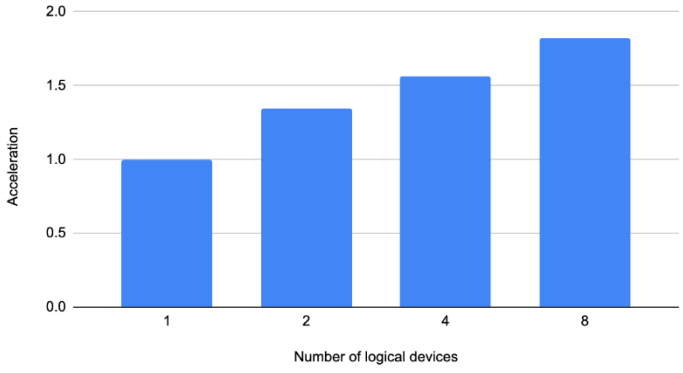

Fig. 5. Acceleration of the model per epoch on 1, 2, 4, and 8 TPU devices

It can be observed that with the increase in the number of devices, acceleration grew, but did not approach the theoretical value. The apriori estimation of efficiency is as follows:

$$E_P = \frac{S_P(N)}{P} = \frac{O(L * N * M * D)}{P * O(L * \frac{N}{P} * M * D)}, \quad (6)$$

where P – the number of processors; $S_P(N)$ – the acceleration of parallel execution of the algorithm for P processors.

Thus, it is expected that with optimal use of all available processors under ideal conditions, the efficiency will be equal to 1. It should also be noted that it is impossible to achieve maximum efficiency when only part of the

available processors is used. In other words, the maximum efficiency achievable with P processors out of N devices will be P/N. In the case of parallelizing training on TPU, the total number of devices available on the machine initiated through tf.distribute.cluster_resolver.TPUClusterResolver was 8. Therefore, the number of processors P used for training efficiency calculations was 8. The actual efficiency achieved with parallelization on TPU devices is presented in Table 5 and illustrated in Figure 6.

Table 5

Actual efficiency metrics of the parallel algorithm depending on the number of TPU devices used

	Number of TPU			
	1	2	4	8
E_P	0.125	0.1683	0.1949	0.2279

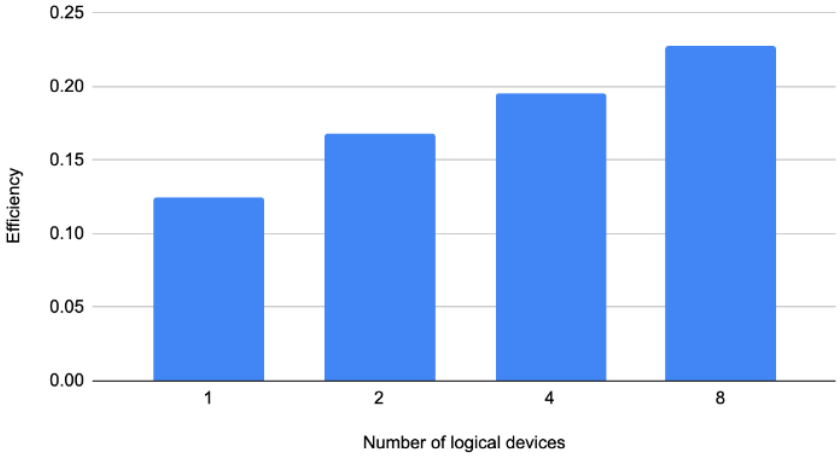

Fig. 6. Efficiency of the model per epoch on 1, 2, 4, and 8 TPU devices

Looking at the maximum efficiency achieved on 8 processors, it is again clear that the ideal value was not reached. However, it can be stated that the highest efficiency of 0.23 was achieved with 8 TPU devices.

Scalability of the algorithm with varying training and validation data

To assess the horizontal scalability of the parallel algorithm, measurements of the execution time for one epoch were taken while simultaneously increasing both the dataset size and the number of processors. Assuming the full dataset size is N, the training and validation dataset sizes were varied as parts of N, specifically $1/8N$, $1/4N$, $1/2N$, and N. These results are summarized in Table 6.

Table 6

Time to complete one epoch with varying TPU counts and dataset portions for model training (seconds)

Dataset portion	Number of TPU			
	1	2	4	8
0.125 N	30	24	21	19
0.25 N	59	48	41	37
0.5 N	118	97	81	72
N	237	176	152	130

As shown in Table 6, increasing the number of TPU devices decreases the training time for the same amount of data, and the difference in time becomes more significant as the dataset size grows. Table 7 shows the acceleration achieved with different numbers of TPUs and varying portions of the dataset used for training the model. Table 8 illustrates the efficiency achieved with different numbers of TPUs and varying portions of the dataset used for training.

Table 7

Acceleration with varying TPU counts and dataset portions for model training

Dataset portion	Number of TPU			
	1	2	4	8
0.125 N	1	1.25	1.4286	1.5789
0.25 N	1	1.2292	1.4390	1.5946
0.5 N	1	1.2165	1.4568	1.6389
N	1	1.3466	1.5592	1.8231

The greatest acceleration, 1.82, was achieved with 8 TPU devices while training on the full dataset, as indicated in Table 7.

Table 8

Efficiency with varying TPU counts and dataset portions for model training

Dataset portion	Number of TPU			
	1	2	4	8
0.125 *N*	0.125	0.1563	0.1786	0.1974
0.25 *N*	0.125	0.1536	0.1799	0.1993
0.5 *N*	0.125	0.1521	0.1821	0.2049
N	0.125	0.1683	0.1949	0.2279

As with acceleration, the highest efficiency from parallelization was observed when training on the full dataset using 8 TPU devices, as shown in Tables 7 and 8. However, certain limitations must be acknowledged. First, while parallelization reduces training time, efficiency does not scale linearly with the number of TPU devices due to communication overhead and resource allocation constraints. Second, the approach relies on access to specialized hardware, which may limit its applicability in environments with restricted computational resources. Finally, variations in dataset characteristics could impact the generalization of results, necessitating further validation of diverse image classification tasks.

In conclusion, the parallelized algorithm on TPU demonstrates a certain level of scalability, as increasing the number of devices reduces data processing time. However, the time difference between training on a given dataset size using fewer devices and the same dataset size doubled with more devices is not substantial. For example, if we compare the training time for one epoch on 12*N* data with 4 TPU (81 seconds) versus training on the full dataset (*N*) with 4 TPU (152 seconds) and on the full dataset with 8 TPU (130 seconds), the time savings ratio is 152/130 = 1.17, which is not a significant acceleration.

When comparing the time for sequential training of one epoch on 1 GPU (598 seconds) as done in the study in [18], and the time on 8 TPU (130 seconds), we achieved a 4.6 times faster training. The highest accuracy reported in [18] on the first phase of training was 0.9463. In our experiments, we reached a similar accuracy of 0.9467. However, with the use of 8 TPU devices, this accuracy was achieved starting from the 15th epoch, as opposed to the 50th epoch when using 1 GPU in [18].

Conclusions

The increasing volume and complexity of data generated and analyzed in modern scientific and technological research necessitate the use of parallel machine learning methods. Training machine learning models on large datasets requires significant computational resources, making the use of a single processor inefficient. Parallel approaches enable substantial acceleration of the training process and improve the efficiency of analyzing large-scale datasets.

A review of scientific literature has shown that many researchers focus on improving the architecture of classification models to enhance their accuracy and training speed. One of the widely adopted approaches involves utilizing CUDA technology and optimizing model structures to achieve effective parallelization.

In this study, a parallel algorithm was implemented for training a bird classification model using data parallelism on multiple TPUs. The conducted experiments confirmed the scalability of the algorithm, assessed by analyzing the training time when both the dataset size and the number of computing devices increased proportionally. While the results demonstrate a certain degree of scalability, there is room for further improvement.

A comparative analysis with previous studies demonstrated that the proposed approach accelerated model training by a factor of 4.6 compared to the sequential algorithm on a single GPU. At the same time, key classification metrics remained at a high level, even when using 8 TPUs for parallel training.

Thus, the findings of this study confirm the potential of parallel machine learning for image classification tasks. Future research can focus on developing even more efficient parallelization approaches that offer better scalability while accounting for the complexity and size of input data. Additionally, future work should address the limitations identified in this study. The non-linear efficiency scaling with increasing TPU devices suggests the need for optimized workload distribution strategies. Moreover, exploring techniques to reduce the dependency on high-performance hardware could enhance the method's accessibility. Finally, further investigation is required to validate the approach across different datasets and real-world scenarios to ensure robustness and adaptability. As data volume and complexity continue to grow, the importance of parallel machine learning methods is expected to increase in the coming years.

References

[1] D. P. Mishra, S. Mishra, S. Jena, and S. R. Salkuti, “Image classification using machine learning,” *Indones. J. Electr. Eng. Comput. Sci.*, vol. 31, no. 3, p. 1551, Sep. 2023, doi: 10.11591/ijeecs.v31.i3.pp1551-1558.

[2] C. Rubio-Manzano, “Image classification using deep and classical machine learning models on small datasets: a complete comparative,” *CLEI Electron. J.*, vol. 27, no. 1, Apr. 2024, doi: 10.19153/cleiej.27.1.1.

[3] K. Chandni, A. P. Singh, A. K. Rai, and A. S. Chauhan, "Image Recognition Using Machine Learning Techniques," in *2023 International Conference on Sustainable Emerging Innovations in Engineering and Technology (ICSEIET)*, Ghaziabad, India: IEEE, Sep. 2023, pp. 347–350. doi: 10.1109/ICSEIET58677.2023.10303370.

[4] L. Mochurad, A. Ilkiv, and Y. Mochurad, "A deep learning approach for medical image classification using XAI and convolutional neural networks," in *Responsible and Explainable Artificial Intelligence in Healthcare*, Elsevier, 2025, pp. 183–220. doi: 10.1016/B978-0-443-24788-0.00008-X.

[5] O. Tsaryniuk, "Application of Machine Learning Methods for Satellite Image Classification: a Literature Review and Overview of Key Frameworks," *Nauk. Res. Pap. Comput. Sci.*, vol. 6, pp. 36–40, Mar. 2024, doi: 10.18523/2617-3808.2023.6.36-40.

[6] J. Zellweger-Fischer, J. Hoffmann, P. Korner-Nievergelt, L. Pfiffner, S. Stoeckli, and S. Birrer, "Identifying factors that influence bird richness and abundance on farms," *Bird Study*, vol. 65, no. 2, pp. 161–173, Apr. 2018, doi: 10.1080/00063657.2018.1446903.

[7] Y. Wang, J. Fu, and B. Wei, "A novel parallel learning algorithm for pattern classification," *SN Appl. Sci.*, vol. 1, no. 12, p. 1647, Dec. 2019, doi: 10.1007/s42452-019-1687-6.

[8] S. A. Salman, S. A. Dheyab, Q. M. Salih, and W. A. Hammood, "Parallel Machine Learning Algorithm," *Mesopotamian J. Big Data*, pp. 12–15, Jan. 2023, doi: 10.58496/MJBD/2023/002.

[9] D. C. Youvan, "Parallel Precision: The Role of GPUs in the Acceleration of Artificial Intelligence," 2023, doi: 10.13140/RG.2.2.21937.76641.

[10] A. A. Khan *et al.*, "Secure Remote Sensing Data With Blockchain Distributed Ledger Technology: A Solution for Smart Cities," *IEEE Access*, vol. 12, pp. 69383–69396, 2024, doi: 10.1109/ACCESS.2024.3401591.

[11] A. Y. K. Wong *et al.*, "Greening of Automated Hydroponic System Based on IoT and Enhanced Solar Energy with Electricity Saving Box," *J. Adv. Res. Appl. Sci. Eng. Technol.*, pp. 142–150, Oct. 2024, doi: 10.37934/araset.62.1.142150.

[12] H.-T. Vo, N. N. Thien, and K. C. Mui, "Bird Detection and Species Classification: Using YOLOv5 and Deep Transfer Learning Models," *Int. J. Adv. Comput. Sci. Appl.*, vol. 14, no. 7, 2023, doi: 10.14569/IJACSA.2023.01407102.

[13] K. J. Dsouza and Z. A. Ansari, "HISTOPATHOLOGY IMAGE CLASSIFICATION USING HYBRID PARALLEL STRUCTURED DEEP-CNN MODELS," *Appl. Comput. Sci.*, vol. 18, no. 1, pp. 20–36, Mar. 2022, doi: 10.35784/acs-2022-2.

[14] J. Lu, "Neural Network Analysis Based on Data Parallelism," *J. Phys. Conf. Ser.*, vol. 2428, no. 1, p. 012041, Feb. 2023, doi: 10.1088/1742-6596/2428/1/012041.

[15] I. Atik, "Parallel Convolutional Neural Networks and Transfer Learning for Classifying Landforms in Satellite Images," *Inf. Technol. Control*, vol. 52, no. 1, pp. 228–244, Mar. 2023, doi: 10.5755/j01.itc.52.1.31779.

[16] S. Karagiannakos, "Distributed Deep Learning training: Model and Data Parallelism in Tensorflow." Feb. 14, 2025. [Online]. Available: https://theaisummer.com/distributed-training/

[17] S. Thite, "BIRDS 525 SPECIES- IMAGE CLASSIFICATION." Feb. 14, 2025. [Online]. Available: https://www.kaggle.com/code/sunilthite/birds-525-species-image-classification

[18] L. Mochurad and S. Svystovych, "A New Efficient Classifier for Bird Classification Based on Transfer Learning," *J. Eng.*, vol. 2024, no. 1, p. 8254130, Jan. 2024, doi: 10.1155/2024/8254130.

[19] A. H. Ali, M. G. Yaseen, M. Aljanabi, S. A. Abed, and C. Gpt, "Transfer Learning: A New Promising Techniques," *Mesopotamian J. Big Data*, pp. 29–30, Feb. 2023, doi: 10.58496/MJBD/2023/004.

[20] L. Mochurad, M. Davidekova, and S.-A. Mitoulis, "Parallel rapidly exploring random tree method for unmanned aerial vehicles autopilot development using graphics processing unit processing," *IAES Int. J. Artif. Intell. IJ-AI*, vol. 14, no. 1, p. 712, Feb. 2025, doi: 10.11591/ijai.v14.i1.pp712-723.

Lesia Mochurad **Леся Мочурад**	PhD, Associate Professor of Department of Artificial Intelligence, Lviv Polytechnic National University, Lviv, Ukraine, e-mail: lesia.i.mochurad@lpnu.ua. https://orcid.org/0000-0002-4957-1512 Scopus Author ID: 57210286606, ResearcherID: AAZ-9150-2020	кандидат технічних наук, доцент, доцент кафедри систем штучного інтелекту національного університету "Львівська політехніка", Львів, Україна.
Khrystyna Dolynska **Христина Долинська**	student of Department of Artificial Intelligence, Lviv Polytechnic National University, Lviv, Ukraine, e-mail: khrystyna.dolynska.kn.2021@lpnu.ua https://orcid.org/0009-0005-0473-9640	студентка кафедри систем штучного інтелекту національного університету "Львівська політехніка", Львів, Україна.
Tetiana Ufimtseva **Тетяна Уфімцева**	student of Department of Artificial Intelligence, Lviv Polytechnic National University, Lviv, Ukraine, e-mail: tetiana.ufimtseva.kn.2021@lpnu.ua https://orcid.org/0009-0007-8087-6385	студентка кафедри систем штучного інтелекту національного університету "Львівська політехніка", Львів, Україна.