**Skin Cancer Detection Using Machine Learning and Advanced Image Analysis**

Yukta Chikate, Sahil Padyal

Northeastern University

CSYE 7105: High-Performance Parallel Machine Learning and AI

Handan Liu

**Introduction**

Skin cancer, a prevalent and potentially lethal ailment, presents a complex challenge in the healthcare industry. With rising incidence rates worldwide, it is imperative to advance diagnostic methods to swiftly and accurately detect the disease. Traditional diagnostic approaches, primarily relying on visual assessments by dermatologists followed by histopathological examination of biopsy samples, are hindered by several limitations. These include the invasiveness of procedures, potential discomfort for patients, and the considerable time required for pathological analysis. Additionally, these methods depend heavily on the dermatologist's experience, which can lead to inconsistent diagnoses and the possibility of human error.

The need for improved diagnostic techniques is underscored by the significant benefits of early skin cancer detection, which dramatically increases patient survival rates. Early and precise identification of skin lesions—distinguishing benign from malignant—can lead to timely and effective treatment, directly impacting patient outcomes.

In recent years, the field of artificial intelligence (AI), with machine learning (ML) at its forefront, has emerged as a promising catalyst for revolutionizing medical diagnostics. Deep Convolutional Neural Networks (DCNNs), a subset of ML, have shown exceptional prowess in image recognition tasks, including the analysis of dermatoscopic images for skin cancer detection. This project explores the application of DCNNs to dermatology, aiming to capitalize on their strengths to address the current diagnostic gaps.

The motivation behind this initiative stems from the global necessity for more accessible, efficient, and precise diagnostic services. Machine learning models can process and analyze dermatological images with remarkable speed, potentially outperforming human accuracy and consistency. The accessibility of such models could revolutionize the availability of diagnostic services, particularly in regions where dermatological expertise is scarce.

Furthermore, the efficiency of ML models in processing large volumes of images could significantly reduce the time required for skin cancer screening. Such advancements hold the promise of easing the strain on healthcare systems and ensuring that a higher number of at-risk patients receive timely diagnoses.

The cornerstone of this project is the HAM10000 dataset, a rich collection of dermatoscopic images encompassing a diverse range of skin lesions. This dataset provides a robust foundation for training and testing the machine learning model. Utilizing this comprehensive dataset, the project aims to develop an ML model that not only matches the diagnostic capability of dermatologists but also supports them by minimizing the dependency on biopsies and reducing the chance of diagnostic delays.

The goal of this project is multifold: to develop a reliable ML model for the classification of skin lesions, to validate its effectiveness through rigorous testing, and to demonstrate the potential of such technologies in enhancing dermatological diagnostics. The project's success could signify a transformative step in dermatology, where the integration of AI in clinical practices becomes a reality, heralding a new age of digital diagnostics that contributes to saving lives by enabling earlier and more accurate detection of skin cancer.

**Methodology**

In the development of a machine learning model for skin lesion classification, our methodology employs a rigorous data processing strategy that addresses the challenges posed by class imbalance within large dermatoscopic image datasets. The datasets in question include HAM10000, which provides a rich repository of raw image data along with a ground truth CSV file detailing lesion diagnosis, and an additional dataset, referred to as BENIGN, that comprises benign lesion images.

The first stage of our methodology involves preparing the HAM10000 dataset for ingestion by the model. This dataset consists of over 10,000 dermatoscopic images of various skin conditions and is widely recognized for its diversity and representativeness of skin lesions, making it ideal for training a robust model. To accompany these images, the HAM10000 dataset includes a ground truth CSV file that contains the correct diagnosis for each image. This file is crucial for the supervised learning approach, as it provides the labels against which the model's predictions are compared.

In handling the raw image data, particular attention is given to organizing the images in a manner conducive to model training. To this end, directories are created for each lesion category as defined by the ground truth CSV file. Python’s os and shutil libraries are leveraged for directory management, ensuring that the images are systematically categorized for easy access during the training process.

The BENIGN dataset, however, is composed solely of images classified as benign. These images are intended to augment the benign class in the HAM10000 dataset, providing a more balanced representation of benign lesions and enhancing the model's ability to accurately classify non-malignant cases.

Once the images from both datasets are compiled, the next critical step in our methodology is to address the issue of class imbalance. Class imbalance occurs when certain lesion types are underrepresented in the dataset, which can lead to a model that is biased toward the more frequent classes. To counteract this, we calculate class weights that adjust the model's focus toward less represented classes. This is accomplished by initiating a weight array with zeros and populating it with the inverse frequency of each class's representation. These weights are then normalized across all classes, effectively ensuring that the model pays proportionate attention to each class during training.

After calculating class weights, the next focus is on preprocessing the images to fit the model's input requirements. Since the selected model architecture, ResNet50, requires a uniform input size, images are resized to a consistent resolution. The importance of this step cannot be overstated as inconsistent image sizes can lead to suboptimal model training and performance.

To ensure the integrity of the dataset and prevent data leakage, the images are checked for plagiarism using the ground truth CSV file. This step involves verifying that each image in the dataset is unique and that there are no duplicates which could artificially inflate the model's performance metrics.

Once preprocessing is complete, the datasets are split into training, validation, and test sets, following a standard approach such as an 80/20 split for training and validation. This division is pivotal for evaluating the model's performance on unseen data, ensuring that the final metrics reflect the model's true predictive power.

The methodology also considers the use of data augmentation techniques. Although not explicitly mentioned in the initial stages, data augmentation can further improve the model's generalization by exposing it to varied transformations of the training images, thereby mimicking a more extensive and diverse dataset.

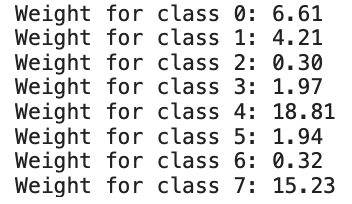
Throughout the process, the efficiency of data handling is paramount. This is where parallel computing comes into play, using Python is concurrent.futures ThreadPoolExecutor for accelerated image copying and organization. Parallel processing significantly reduces the time required to manage the data, which is essential when dealing with datasets of this magnitude.

In conclusion, the proposed methodology provides a comprehensive approach to preparing and processing dermatoscopic image datasets for the development of a machine learning model. By meticulously organizing the data, calculating class weights to address imbalance, ensuring dataset integrity, and efficiently managing large volumes of images, we establish a strong foundation for training a model that is both accurate and unbiased in its predictions. These methodological steps are crucial for the advancement of diagnostic capabilities in the field of dermatology, ultimately aiming to improve patient outcomes through early and accurate skin lesion detection.

The intricate process of developing a machine learning model for skin lesion classification using dermatoscopic images from the HAM10000 and BENIGN datasets involves meticulous steps, including class weight calculation, image preprocessing, augmentation, and multiprocess handling. These steps are critical for addressing the inherent challenges posed by class imbalances and for enhancing the performance of the model.

**Class Weights and Image File Analysis**

In the realm of medical image classification, one of the primary challenges is the imbalance in the representation of various classes within a dataset. For the HAM10000 dataset, which includes over 10,000 dermatoscopic images with associated diagnoses, and the BENIGN dataset, which supplements the benign cases, class imbalance can significantly skew the model's learning process towards more frequent classes. To counter this, we adopt a strategy of calculating class weights, which are essential in guiding the model to "pay more attention" to underrepresented classes.



The computation of class weights begins by initializing an array set to zero, matching the number of unique classes derived from the datasets. The methodology involves iterating through each class directory, calculating the number of images per class, and inversely weighing the classes based on their representation. These weights are adjusted such that less frequent classes receive higher weights, ensuring the model is equally sensitive to all classes during training.

Class weight calculation not only aids in achieving a balanced training regime but also prepares the ground for initializing the bias terms in the model. This strategic decision in the pre-modeling phase is critical for offsetting the initial learning bias that may occur due to class imbalance. After normalizing these weights by the total count and the number of classes, they are formatted into a dictionary structure that is compatible with Keras, the deep learning API used for model training.

**Image Augmentation**

Once the class weights are determined, we turn our attention to image augmentation. In a dataset as varied as HAM10000, image augmentation is a technique used to artificially expand the dataset with altered versions of existing images. This not only helps to prevent overfitting by providing the model with a more comprehensive understanding of class variations but also mimics a broader dataset, thereby improving the robustness and generalizability of the model.

Augmentation techniques can include rotations, flips, zooms, and shifts, each contributing to the model's exposure to a wide array of potential lesion presentations. By incorporating these variations, the model learns to recognize and classify lesions regardless of their orientation or size, which closely aligns with the real-world scenario where dermatoscopic images may not be uniform.

**Multiprocessing for Efficiency**

Given the extensive nature of the HAM10000 and BENIGN datasets, efficiency in data handling becomes paramount. Python's multiprocessing capabilities are leveraged to accelerate the preprocessing stage. Utilizing **ThreadPoolExecutor** from the concurrent.futures module allows for the parallel execution of file operations, such as copying images to designated directories, which markedly speeds up the process.

The use of multiprocessing is extended to the loading and augmentation of images, where the data-intensive tasks are distributed across multiple cores, reducing the time taken to make the dataset model-ready. This approach ensures that the large volumes of data are handled in an optimized manner, leading to quicker iterations and faster development cycles.

The methodology adopted in this project addresses the class imbalance problem by calculating and applying class weights to ensure fair representation during the model's training. This is complemented by a comprehensive image augmentation strategy that introduces a variety of modified images to the model, further supporting its ability to generalize from the training data to real-world dermatoscopic images.

The multiprocessing approach ensures efficient handling of the data, significantly reducing preprocessing times and allowing the model to be trained on a fully prepared dataset without undue delay. The combination of these methods—class weight balancing, image augmentation, and multiprocessing—establishes a robust pipeline for training a machine learning model capable of accurate and reliable skin lesion classification.

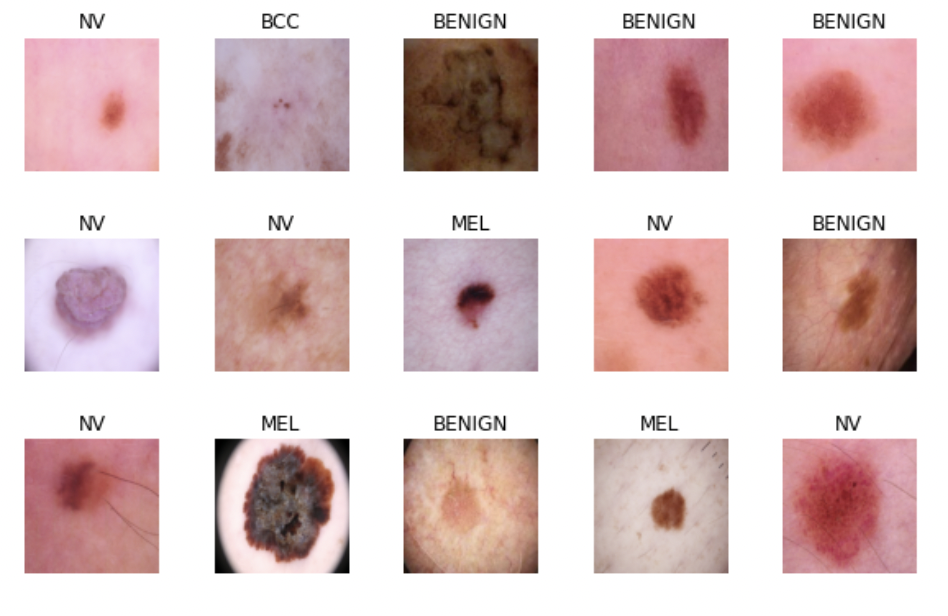
In total, these methods form a comprehensive and balanced methodology designed to tackle the nuances of medical imaging data, facilitating the development of a model that is both precise and practical for clinical use. With the implementation of these techniques, the potential of machine learning to revolutionize the field of dermatological diagnostics is realized, leading to improved diagnostic accuracies and better patient outcomes.

In the pursuit of creating a precise and robust deep learning model for the classification of skin lesions, our approach is grounded in methodical data preparation, strategic training-validation-test splitting, advanced image augmentation, and efficient multiprocessing techniques. The ultimate goal is to harness the power of deep learning to classify images accurately, reducing the time to insights, which is critical in medical diagnostics.

**Image Augmentation: Expanding Dataset Diversity**

After splitting, the dataset proceeds through an image augmentation phase. This is a technique used to artificially expand the dataset by creating modified versions of existing images through transformations such as rotations, flips, and shifts. Augmentation is crucial for several reasons; it prevents overfitting by providing a more varied set of images for the model to learn from, it simulates a broader range of image presentations, and it strengthens the model's robustness against variations in new data.

Implementing augmentation requires careful consideration. For instance, certain transformations that are plausible in one context may not be applicable in another. In the case of dermatoscopic images, care is taken to ensure that augmentations do not distort the critical features that are indicative of specific skin lesions.



Multiprocessing: Streamlining Image Processing

Given the computationally intensive nature of handling large image datasets, particularly when applying augmentations, multiprocessing is introduced to streamline the preprocessing workflow. By distributing the workload across multiple CPU cores, image processing tasks are performed in parallel, significantly reducing the time required to prepare the dataset. Multiprocessing is particularly beneficial for tasks that are independent of each other, such as processing different image files, making it a perfect fit for the augmentation phase.

**Image Augmentation and Multiprocessing**

Image augmentation is an essential technique in machine learning, particularly useful in scenarios like medical image analysis where the dataset is limited and the need for generalization is high. Augmentation artificially expands the dataset by generating transformed versions of existing images through operations such as rotations, translations, and flips. This not only simulates a larger dataset but also introduces a variety of conditions under which the model must perform accurately, thus enhancing its robustness and ability to generalize from limited data.

The process of augmentation can be computationally intensive, especially when dealing with high-resolution images and large datasets. To address this, multiprocessing is employed to parallelize the augmentation process, allowing for multiple image transformations to be executed simultaneously across different CPU cores. This efficient use of computing resources speeds up the preparation of the augmented dataset, ensuring that the model has a wealth of varied images to learn from without incurring prohibitive time costs.

The screenshot shows the importation of necessary libraries for image manipulation and parallel processing, namely matplotlib for plotting, torch and torchvision for transformations and dataset management, multiprocessing for concurrent execution, and numpy for numerical operations. These libraries are foundational tools that enable the manipulation and preparation of image data for machine learning tasks.

A collage of different skin diseases

Description automatically generated

**Data Splitting**

Initially, the project undertakes the pivotal task of data splitting. Proper data splitting is vital to prevent model overfitting and to ensure the model is generalized well enough to perform accurately on unseen data. We adopted a standard 80/20 split, allocating 80% of the data for training to allow the model to learn the intricacies of the various classes of skin lesions and 20% for validation and testing. This primary split ensures that the model is exposed to a vast array of data points during the training phase.

To refine this further, the 20% set aside is then split again, this time adopting an 80/10/10 partition. This translates to 80% of the original dataset being used for training, 10% for validation, and the remaining 10% for testing. This meticulous segmentation allows for a balanced approach where the validation set is used to tune hyperparameters, while the test set serves as an unbiased evaluation of the final model's performance.

The datasets in question, HAM10000 and BENIGN, each present their unique challenges. The HAM10000 dataset comprises over 10,000 images annotated with a ground truth CSV file for lesion identification. The BENIGN dataset exclusively contains images of benign lesions, supplementing the benign class within HAM10000. The synergy of these two datasets provides a comprehensive coverage of skin lesion types, essential for developing a well-rounded diagnostic tool.

without any idle time. This parallel processing of data loading and model training is key to accelerating the learning process.

**Customizing ResNet for Skin Lesion Classification**

ResNet, known for its deep architecture and residual learning framework that facilitates the training of networks that are substantially deeper than those previously used, is highly suitable for complex tasks such as image classification. For our project, the ResNet model is adapted to address the particular challenge of skin lesion classification. A custom ResNet class is defined, which extends the pre-trained ResNet model provided by PyTorch's model library.

In the customized ResNet model, the original fully connected layer at the end of the network, which is typically tailored for the ImageNet dataset, is replaced with a new fully connected layer whose output features correspond to the number of classes in the skin lesion dataset. If a bias initializer is provided, it is applied to the new layer to facilitate faster convergence during training.

The forward method of the custom model is designed to leverage the powerful feature extraction capabilities of the ResNet architecture. It does this by-passing input data through the retained layers, applying a global average pooling layer to reduce the spatial dimensions to a single value per channel, and then flattening the results to form a vector that can be fed into the custom fully connected layer for classification.

**ResNet Model: Training and Classification**

The last step in the process is training the ResNet model. ResNet, or Residual Network, is a type of convolutional neural network that is known for its deep architecture and the use of skip connections, or shortcuts, to jump over some layers. Pre-trained on ImageNet, the ResNet model is adapted for the task of skin lesion classification by replacing the final classification layer with one that corresponds to the number of classes in the dataset.

Training involves fine-tuning the pre-trained network weights using the augmented and original images from the HAM10000 and BENIGN datasets. This process is carried out in multiple stages, where initially, only the classifier layer is trained while the rest of the network weights are frozen. Gradually, more layers of the network are unfrozen and trained to adapt to the specific features of the skin lesion images.

Throughout the training process, the data is fed into the model in batches using a DataLoader, which has been configured for multi-threading to efficiently manage the input pipeline. The DataLoader uses the concept of prefetching, where while the GPU is busy with computations for the current batch of data, the CPU simultaneously prepares the next batch, thereby ensuring that the GPU always has data to work on

**Training Pipeline**

Training a deep learning model is a cyclic process comprising forward passes, where inputs are passed through the model to obtain predictions, and backward passes, where gradients are computed for each parameter in the model to minimize the loss. The loss function used here is cross-entropy loss, which is well-suited for multi-class classification tasks like ours.

An optimizer is employed to adjust the parameters of the model based on the gradients calculated during the backward pass. Adam optimizer, known for its adaptive learning rate capabilities, is chosen for this purpose. Before each pass, gradient values are reset to zero to prevent accumulation from previous iterations.

Each batch of images from the dataset is passed through the custom ResNet model during the forward pass, and the loss is computed by comparing the model's predictions against the true labels. During the backward pass, the optimizer updates the model's weights to minimize this loss.

**GPU Acceleration**

With the advent of powerful GPU computing, training deep neural networks has become increasingly feasible. The custom ResNet model is moved to a GPU device, if available, which significantly speeds up the matrix operations involved in training due to the parallel nature of GPUs. This is a key factor in handling the computational load of deep learning models, especially when dealing with high-resolution images and large datasets.

**Evaluation and Fine-Tuning**

After training, the model is set to evaluation mode, which disables certain layers like dropout and batch normalization that behave differently during training versus testing. This ensures that the evaluation of the model's performance on the validation set is accurate and reflects its ability to generalize to new data.

Fine-tuning is the process of making small adjustments to the pre-trained model to better adapt it to the new task. This involves training the model further on the skin lesion dataset, allowing the model to refine the features it has learned from the ImageNet dataset and adjust them to better suit the specific patterns and characteristics found in skin lesions.

The ResNet model's adaptability makes it an excellent choice for a variety of image classification tasks, including the classification of skin lesions. Through careful customization, training, and fine-tuning, the model is prepared to assist in the diagnostic process, providing valuable insights and potentially improving the accuracy and efficiency of skin cancer detection.

The development of an accurate machine learning model involves not just the architectural design and the training process, but also the implementation of sophisticated mechanisms to monitor and enhance the training phase. These mechanisms, known as callbacks, are crucial in optimizing the training process, ensuring that the model not only learns effectively but also does so within the constraints of computational efficiency.

**The Role of Callbacks in Training**

In the context of training a deep learning model like ResNet for the classification of skin lesions, callbacks serve as checkpoints that monitor the training process at each epoch. They provide the ability to perform actions at various stages of training, such as adjusting the learning rate or halting the training early to prevent overfitting.

Initially, a model's performance might show a steady decline in loss, indicating learning and improvement. However, as training progresses, improvements can plateau or even degrade, which is when callbacks become particularly useful. One such callback is EarlyStopping, which monitors a specific performance metric—validation loss, in this case—and stops training when that metric ceases to improve. Another is ReduceLROnPlateau, which reduces the learning rate when the model's performance hits a plateau, allowing for finer adjustments in the model's weights and potentially leading to better performance with more epochs.

EarlyStopping and Reduced Learning Rate

In this project, the EarlyStopping callback is set to monitor the validation loss, with a 'patience' parameter indicating how many epochs to wait after the last time the validation loss improved before stopping the training. This not only prevents overfitting but also saves computational resources by avoiding unnecessary training epochs.

The initial training might show the loss starting at 1.3188, but after applying these callbacks and allowing the model to train and learn from the data, the validation loss can decrease significantly. For instance, the validation loss might reduce to 1.2353, demonstrating the model's improved performance thanks to the effective use of callbacks.

Reducing the Learning Rate on Plateaus

The ReduceLROnPlateau callback works hand-in-hand with EarlyStopping by reducing the learning rate when the validation loss stops improving. This allows for smaller steps in weight adjustments, which can lead to more nuanced learning and potentially better model performance. By carefully tuning the learning rate, the model can escape local minima or plateaus in the loss landscape, which could be preventing further improvement.

**EarlyStopping and Reduced Learning Rate**

In this project, the EarlyStopping callback is set to monitor the validation loss, with a 'patience' parameter indicating how many epochs to wait after the last time the validation loss improved before stopping the training. This not only prevents overfitting but also saves computational resources by avoiding unnecessary training epochs.

The initial training might show the loss starting at 1.3188, but after applying these callbacks and allowing the model to train and learn from the data, the validation loss can decrease significantly. For instance, the validation loss might reduce to 1.2353, demonstrating the model's improved performance thanks to the effective use of callbacks.

Reducing the Learning Rate on Plateaus

The ReduceLROnPlateau callback works hand-in-hand with EarlyStopping by reducing the learning rate when the validation loss stops improving. This allows for smaller steps in weight adjustments, which can lead to more nuanced learning and potentially better model performance. By carefully tuning the learning rate, the model can escape local minima or plateaus in the loss landscape, which could be preventing further improvement.

**Training**

**Using Model Parallel to Train**

**Setup**

**Device Selection**: Dynamically selects GPU if available, otherwise falls back to CPU.

**Model Cloning**: Creates a clone of the model to keep the original model's weights independent.

**Loss** **Function**: Utilizes CrossEntropyLoss, optionally weighted by class.

**Optimizer**: Employs Adam optimizer with a predefined learning rate.

**Learning** **Rate** **Scheduler**: Implements ReduceLROnPlateau for adaptive learning rate adjustment.

**Early** **Stopping**: Incorporates early stopping mechanism to halt training when validation loss ceases to decrease, preventing overfitting.

**Parallelism**

**Initialization**: Moves the cloned model to the specified device and wraps it with nn.DataParallel for distributed computing across multiple GPUs.

**Data** **Loading**: Ensures input data and labels are moved to the correct device during training and validation phases.

**Training Loop**

**Epoch** **Iteration**: Cycles through a predefined number of epochs.

**Batch Processing**:

Performs forward pass, computes loss, and conducts backpropagation.

Accumulates training loss for performance metrics.

**Validation Phase**:

Evaluates the model on a separate validation set without gradient updates.

Computes validation loss for performance monitoring and early stopping.

**Performance Monitoring**

**Learning Rate Adjustment**: Adjusts the learning rate based on the validation loss trend.

**Early Stopping Trigger**: Monitors validation loss for early termination of training to prevent overfitting.

**Training Time**: Measures and reports the total training duration.

Post-Training

**Best Model Restoration**: Loads the model weights from the checkpoint with the lowest validation loss.

The approach effectively leverages modern GPU architectures to accelerate training while maintaining flexibility and ease of use. By employing model cloning, adaptive learning rate scheduling, and early stopping, the methodology aims to achieve high accuracy and generalization in detecting skin cancer from dermatoscopic images.

**Using DataParallel Methodlogy**

**Training Setup**

**Model Preparation**

* **Model Cloning:** Creates a deep copy of the model to ensure that the original model's weights remain independent. This cloned model is used for training to avoid modifying the original model directly.
* **Device Selection:** Dynamically chooses the computation device based on availability, preferring GPUs over CPU to maximize training efficiency.

**Training Components**

* **Optimizer:** Utilizes the Adam optimizer for adjusting model weights, initiated with a predefined learning rate.
* **Loss Function:** Employs CrossEntropyLoss, potentially weighted to handle class imbalances in the dataset.
* **Learning Rate Scheduler:** Implements ReduceLROnPlateau for dynamic adjustment of the learning rate based on validation loss, aiding in convergence and preventing overfitting.
* **Early Stopping:** Integrates an early stopping mechanism to terminate training early if the validation loss stops improving, preventing overfitting and saving computational resources.

**Training Process**

**Initialization**

* Moves the model to the specified device (GPU/CPU) and wraps it with torch.nn.DataParallel. This wrapper distributes batches of data across the multiple GPUs, paralleling the training process.

**Training Loop**

1. **Epoch Iteration:** Runs through the training dataset in batches for a predefined number of epochs.
2. **Batch Processing:** For each batch:
   * Transfers input data and labels to the appropriate device.
   * Performs a forward pass to generate predictions**.**
   * Calculates loss using the defined loss function.
   * Conducts backpropagation to update model weights.
   * Updates the optimizer and accumulates the loss for performance tracking.

**Validation Phase**

* **Loss Calculation:** Computes the loss on a separate validation dataset to monitor the model's performance on unseen data.
* **Performance Monitoring:** Adjusts the learning rate based on validation loss and checks for early stopping conditions.

**Efficiency and Performance**

* **Training Time Measurement:** Records the total time taken for the training process, highlighting the efficiency gains from using data parallelism**.**
* **Resource Utilization:** Reports the number of GPUs utilized during training, showcasing the scalability of the training process.

**Post-Training**

* **Best Model Restoration:** Loads the model weights from the checkpoint associated with the lowest observed validation loss, ensuring that the best-performing model is used for inference or further evaluation.

**Conclusion**

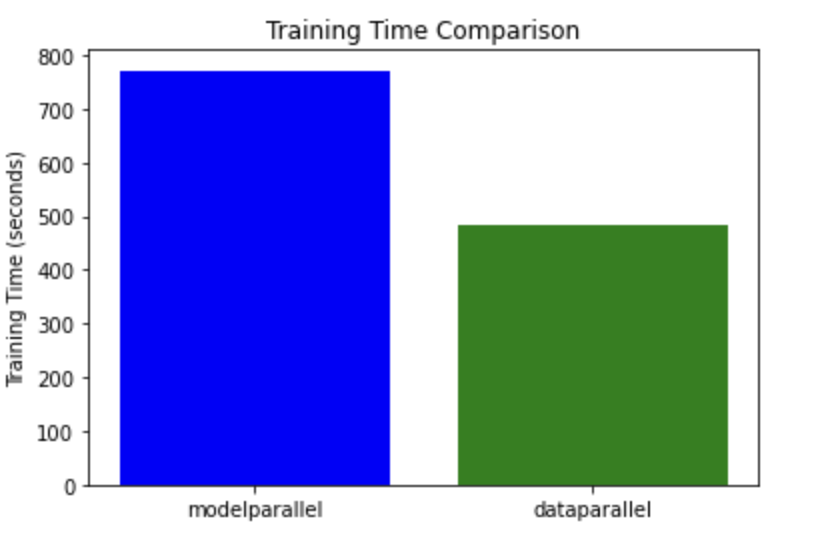
Training with data parallelism is a highly effective methodology for leveraging multiple GPUs, significantly reducing training time while maintaining or improving model performance. By distributing data across available computational resources, this approach optimizes training efficiency and enables the handling of larger datasets and more complex models. Through the integration of adaptive learning rate adjustments and early stopping, the methodology also ensures robust training dynamics, aiding in the convergence to optimal model parameters.

**Final Results**

**Training Time Comparison**

**The training process was conducted using two parallelism strategies:**

1. **Model Parallelism:** This approach involves splitting the model across multiple GPUs, with each GPU handling a portion of the model's layers. The training time observed using model parallelism was 771.23 seconds.
2. **Data Parallelism:** In contrast, data parallelism involves distributing the data across multiple GPUs, with each GPU processing a different subset of the data using a complete replica of the model. The training time recorded for data parallelism was 484.45 seconds.

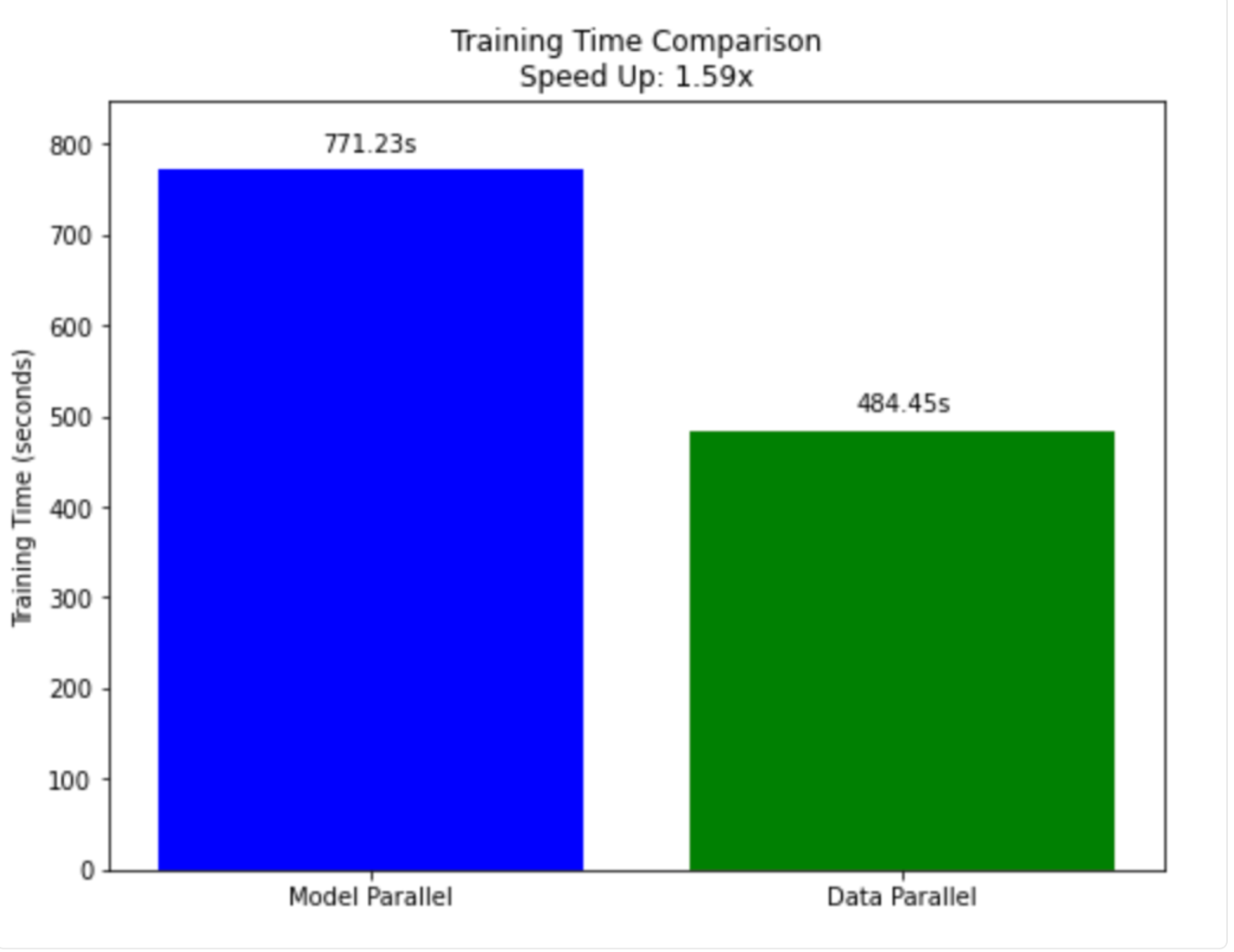


**Speedup Calculation**

The speedup achieved by switching from model parallelism to data parallelism was calculated using the formula:

**Speedup=**Training Time with Model ParallelismTraining Time with Data ParallelismSpeedup=Training Time with Data ParallelismTraining Time with Model Parallelism​

Based on the observed training times, the calculated speedup is approximately 1.59x. This indicates that training the model using data parallelism is nearly 60% faster than using model parallelism under the conditions tested.



**Conclusion**

The comparative analysis between model parallelism and data parallelism techniques has yielded insightful results on the efficiency of parallel computing strategies in machine learning model training.

* **Efficiency of Data Parallelism:** The results demonstrate that data parallelism significantly reduces training time compared to model parallelism. By distributing data across multiple GPUs and leveraging the full model on each GPU, data parallelism provides a substantial speedup in training times.
* **Applicability of Model Parallelism:** While model parallelism showed longer training times in this comparison, it remains a valuable strategy for training models that are too large to fit into the memory of a single GPU. The choice between model and data parallelism should be based on the specific requirements and constraints of the training task, including model size, memory limitations, and available computational resources.
* **Strategic Considerations:** The decision to use data parallelism or model parallelism should consider both the technical aspects and the specific goals of the training process. Data parallelism is particularly effective for accelerating training times when model size is not a limiting factor, while model parallelism is essential for handling large models that exceed GPU memory limits.

**Dataset Descriptions**

<https://www.kaggle.com/datasets/surajghuwalewala/ham1000-segmentation-and-classification/code>

The original [HAM10000 dataset](https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/DBW86T) cannot be easily downloaded in jupyter notebooks from the source.

Also, the other HAM10000 datasets on Kaggle do not have the segmentation masks provided by Philipp Tschandl [here](https://doi.org/10.1038/s41591-020-0942-0) and [here](https://www.kaggle.com/tschandl/ham10000-lesion-segmentations)

So I created this dataset where one can access all the source data in one place without any transformations.

Cases include a representative collection of all important diagnostic categories in the realm of pigmented lesions:

* Actinic keratoses and intraepithelial carcinoma / Bowen's disease (AKIEC),
* basal cell carcinoma (BCC),
* benign keratosis-like lesions (solar lentigines / seborrheic keratoses and lichen-planus like keratoses, BKL),
* dermatofibroma (DF),
* melanoma (MEL),
* melanocytic nevi (NV)
* vascular lesions (angiomas, angiokeratomas, pyogenic granulomas and hemorrhage, VASC).

**2nd dataset**

<https://www.kaggle.com/datasets/bhaveshmittal/melanoma-cancer-dataset>

**Description**

Welcome! This dataset, comprising 13,900 meticulously curated images, is a valuable resource for advancing the field of dermatology and computer-aided diagnostics. Dive into the intricate world of melanoma, where every pixel holds the potential to redefine early detection.

**Context**

Melanoma, a deadly form of skin cancer, demands prompt and accurate diagnosis. Leveraging state-of-the-art technology, this dataset empowers researchers and practitioners to develop robust machine-learning models capable of distinguishing between benign and malignant lesions. The images, uniformly sized at 224 x 224 pixels, offer a comprehensive view of melanoma's diverse manifestations.

**Sources and Inspiration**

This dataset draws inspiration from the critical need for advanced diagnostic tools in dermatology. The images are compiled from diverse sources and showcase the intricate features that challenge traditional diagnostic methods. By sharing this dataset on Kaggle, we invite the global data science community to collaborate, innovate, and contribute towards developing reliable models for melanoma classification.

**References**

* 1. <https://www.kaggle.com/code/quackaddict7/legions-of-lesions-detecting-skin-cancer-with-cv/notebook> for copying images and image augumentation
  2. <https://pytorch.org/tutorials/intermediate/model_parallel_tutorial.html> modell parallel training
  3. <https://stackoverflow.com/questions/51858067/parsing-csv-into-pytorch-tensors> parsing csv to tensors