AI

MDPI

Article

ConVision Benchmark: A Contemporary Framework to Benchmark CNN and ViT Models

Shreyas Bangalore Vijayakumar [1,*,†], Krishna Teja Chitty-Venkata [1,†,‡], Kanishk Arya [2] and Arun K. Somani [1,*]

[1] College of Enginneering, Iowa State University, Ames, IA 50011, USA; krishnat@iastate.edu
[2] Department of Computer Engineering and Technology, MIT World Peace University, Pune 411038, India; 1032212794@mitwpu.edu.in
* Correspondence: shreyasv@iastate.edu (S.B.V.); arun@iastate.edu (A.K.S.)
† These authors contributed equally to this work.
‡ Current address: Argonne National Laboratory, Lemont, IL 60439, USA.

Abstract: Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs) have shown remarkable performance in computer vision tasks, including object detection and image recognition. These models have evolved significantly in architecture, efficiency, and versatility. Concurrently, deep-learning frameworks have diversified, with versions that often complicate reproducibility and unified benchmarking. We propose ConVision Benchmark, a comprehensive framework in PyTorch, to standardize the implementation and evaluation of state-of-the-art CNN and ViT models. This framework addresses common challenges such as version mismatches and inconsistent validation metrics. As a proof of concept, we performed an extensive benchmark analysis on a COVID-19 dataset, encompassing nearly 200 CNN and ViT models in which DenseNet-161 and MaxViT-Tiny achieved exceptional accuracy with a peak performance of around 95%. Although we primarily used the COVID-19 dataset for image classification, the framework is adaptable to a variety of datasets, enhancing its applicability across different domains. Our methodology includes rigorous performance evaluations, highlighting metrics such as accuracy, precision, recall, F1 score, and computational efficiency (FLOPs, MACs, CPU, and GPU latency). The ConVision Benchmark facilitates a comprehensive understanding of model efficacy, aiding researchers in deploying high-performance models for diverse applications.

Keywords: convolutional neural networks; vision transformers; deep-learning framework; PyTorch; COVID-19; ConVision Benchmark

Citation: Bangalore Vijayakumar, S.; Chitty-Venkata, K.T.; Arya, K.; Somani, A.K. ConVision Benchmark: A Contemporary Framework to Benchmark CNN and ViT Models. *AI* **2024**, *5*, 1132–1171. https://doi.org/10.3390/ai5030056

Academic Editor: Arslan Munir

Received: 11 June 2024
Revised: 30 June 2024
Accepted: 9 July 2024
Published: 11 July 2024

1. Introduction

In computer vision (CV), the evolution of deep-learning (DL) models has revolutionized image classification. Convolutional Neural Networks (CNNs), a class of neural networks tailored for processing image data, have become pivotal in recognition tasks. CNNs have demonstrated exceptional performance in efficiently extracting spatial features from input images, making them indispensable in applications ranging from facial recognition and autonomous vehicles to the early detection and diagnosis of various diseases. The fundamental building blocks of CNNs are convolutional layers, which leverage convolutional operations to detect spatial hierarchies of features within an image. This hierarchical feature extraction process enables CNNs to discern patterns at different levels of abstraction. Each convolutional layer consists of learnable filters or kernels that slide across the input data, identifying low-level features such as edges, textures, and shapes. The output of this convolution operation is passed through a non-linear activation function, producing feature maps that capture the presence and location of these detected features. Pooling layers, interspersed between convolutional layers, play a pivotal role in down-sampling and reducing the spatial dimensions of the data, thereby enhancing computational efficiency and aiding in generalization.

However, as datasets grow and tasks become more complex, the limitations of the traditional convolutional approach become apparent. These limitations led to the exploration of alternative architectures, such as Vision Transformers (ViTs) [1]. ViTs represent a paradigm shift in computer vision, leveraging the self-attention mechanism traditionally used in Natural Language Processing (NLP). By dividing input images into fixed-size patches and treating each patch as a token, ViTs enable a global understanding of the image, capturing long-range dependencies and intricate patterns. This approach is particularly beneficial in medical imaging, where subtle visual cues and the spatial distribution of abnormalities are critical for an accurate diagnosis. The capability of ViTs to capture complex relationships between image patches makes them particularly effective for tasks requiring detailed spatial analysis. Transformers rely heavily on massive datasets for large-scale training. Furthermore, due to the lack of some inductive biases compared to CNNs, data quality significantly influences the generalization and robustness of transformers in computer vision.

The development and implementation of CNNs have historically relied on various frameworks, each with advantages and limitations. AlexNet, one of the first CNNs to achieve groundbreaking success in the ImageNet competition, was initially implemented using CUDA [2], a parallel computing platform and programming model developed by NVIDIA [3]. CUDA allowed AlexNet to leverage GPU acceleration, significantly speeding up training times. However, CUDA's low-level programming model posed challenges regarding ease of use and accessibility for researchers without a deep understanding of parallel computing. Following AlexNet, ResNet, a CNN architecture known for deep residual networks, was implemented using Caffe [4]. Caffe, an open-source deep-learning framework developed by the Berkeley Vision and Learning Center, gained popularity due to its speed and modularity [5]. However, Caffe's inflexibility and limited support for dynamic computation graphs made it less suitable for more complex and iterative model development processes. Researchers often faced difficulties modifying existing models or implementing new architectures, leading to a steep learning curve and potential reproducibility issues. The introduction of PyTorch in September 2016 marked a significant shift in the deep-learning landscape. PyTorch, developed by Facebook's AI Research lab, provided a flexible and intuitive framework for building and training neural networks [6]. Its dynamic computation graph model allowed for more straightforward debugging and iterative development. Since its alpha release, PyTorch has grown significantly, with continuous updates and new features that enhance its usability and performance. It has become the preferred framework for many researchers due to its flexibility and extensive support for various neural network architectures.

Despite these advancements, the proliferation of deep-learning frameworks has introduced challenges related to version mismatches, reproducibility, and the need for standardized validation and performance metrics. Researchers often need help with discrepancies in results when models are implemented across different frameworks or versions. This fragmentation hinders the ability to consistently benchmark and compare models, complicating the evaluation of new architectures and their improvements. To address these challenges, we propose ConVision Benchmark, a unified framework implemented in PyTorch that integrates state-of-the-art CNN and ViT models. This framework standardizes the implementation and evaluation process, bridging the gap between different frameworks and versions. Our study conducts an extensive benchmark analysis using a COVID-19 image dataset as a proof of concept. The methodology can be extended to any dataset, enhancing its applicability across various domains. The ConVision Benchmark includes tools for evaluating performance metrics such as accuracy, precision, recall, F1 score, and computational efficiency indicators, including model runtime, FLOPs, MACs, CPU latency, GPU latency, and inference memory, and training memory. This comprehensive evaluation is crucial for applications where accuracy and computational efficiency are paramount.

Through rigorous experimentation and analysis, our study provides valuable insights into the strengths and trade-offs of various CNN and ViT architectures. For instance,

the DenseNet-161 CNN model achieves exceptional accuracy with a peak performance of 95.61% on the COVID-19 dataset, making it suitable for high-precision medical diagnoses. Conversely, the MaxViT-tiny ViT model balances accuracy and efficiency. By offering a competitive accuracy of 95.02% and lower computational demands, it is ideal for real-time medical image classification. The ConVision Benchmark facilitates a deeper understanding of model efficacy, aiding researchers in deploying optimal models for diverse applications.

The main research contributions of this paper are as follows:

1. We provide a comprehensive benchmark framework in PyTorch to address library version mismatches and inconsistent validation metrics in existing frameworks;
2. We foster a deeper understanding of model architectures and their performance in medical image classification;
3. We analyze and compare the validation performance and efficiency of CNN and ViT models and provide insights into their strengths and weaknesses;
4. We incorporate an exhaustive set of performance metrics and computational complexity factors in the evaluation process;
5. To the best of our knowledge, we benchmark the most extensive array of CNN and ViT models extensively trained on the largest publicly available COVID-19 dataset in our ConVision benchmark;
6. Our research can serve as a benchmark for understanding the effectiveness of specific models, and the findings offer valuable insights to develop improved models for image classification in various domains;
7. We open-source the implementation of our models and training methodology for researchers to evaluate their custom dataset efficiently and effectively.

Organization of the Paper: The remainder of this paper is arranged as follows: Section 2 reviews the fundamentals of the CNN and ViT models, while Section 3 provides a brief overview of previous work. Section 4 outlines the benchmarking process of different models on the COVID-19 dataset. In Section 5, we provide a detailed analysis of the results and Section 6 concludes the ConVision Benchmark paper.

2. Background

CNNs have become a cornerstone in computer vision, revolutionizing tasks such as image recognition and object detection. The fundamental building blocks of CNNs are convolutional layers, which leverage convolutional operations to detect spatial hierarchies of features within an image. This hierarchical feature extraction process allows CNNs to discern patterns at different levels of abstraction, making them highly effective for image-related tasks. Each convolutional layer in a CNN consists of a set of learnable filters or kernels that slide across the input data. This sliding mechanism enables the network to identify low-level features such as edges, textures, and shapes. As the filters move across the image, they perform a convolution operation, producing feature maps highlighting the presence and location of these detected features. The output of this convolution operation is then passed through a non-linear activation function, such as the Rectified Linear Unit (ReLU), which introduces non-linearity and enables the network to learn intricate patterns and relationships. Pooling layers, interspersed between convolutional layers, play a crucial role in down-sampling and reducing the spatial dimensions of the data. This down-sampling is essential for improving computational efficiency and aiding the model's generalization. By reducing the spatial resolution, pooling layers make the network more robust to distortions and variations in input images. Typical pooling operations include max pooling and average pooling, which retain the most significant features while discarding less relevant information. At the network's end, fully connected layers integrate the hierarchical features extracted by the convolutional and pooling layers. These layers enable high-level abstraction and facilitate the final classification or regression task. The hierarchical and layered structure of CNNs allows them to automatically learn and adapt to intricate patterns in data, making them indispensable in modern deep-learning

applications. However, convolutional methods face limitations with large datasets and complex tasks, requiring extensive labeled data and significant computational resources.

In recent years, ViTs have introduced a novel approach to image processing, challenging the traditional dominance of CNNs. ViTs leverage the Transformer architecture, initially designed for NLP, to handle image data. This approach has proven effective, especially in capturing long-range dependencies and global context within images. The Transformer architecture consists of several essential components. Initially, the input image is divided into fixed-size, non-overlapping patches, treating each patch as a token. These patches are then linearly embedded to create a sequence of tokens, which form the input for the subsequent Transformer encoder. Positional information is imparted to these tokens to preserve spatial relationships, a crucial step since the Transformer architecture lacks the inherent inductive biases present in CNNs. The Transformer encoder comprises multiple layers, each containing a multi-head self-attention mechanism and a feed-forward neural network. The self-attention mechanism allows each token to attend to all other tokens, capturing long-range dependencies and providing a global image view. This global perspective contrasts with traditional architectures' local receptive fields of convolutional layers. Considering the entire image context is particularly beneficial in detecting subtle visual cues and spatial distributions of abnormalities in medical imaging. The outputs from each Transformer layer are aggregated and fed into a classification head, typically a linear layer, for the final task-specific prediction. Unlike CNNs, where hierarchical features are progressively extracted, ViTs leverage the self-attention mechanism to consider the entire image simultaneously. This global understanding is crucial for applications such as disease detection, where subtle details and their spatial relationships are of critical significance.

3. Related Work

The evolution of CNN models for image classification tasks has made significant strides since their inception. Early CNNs such as LeNet [7] laid the foundation by introducing convolutional layers, pooling operations, and fully connected layers. The groundbreaking AlexNet [2], with its deeper architecture, won the ImageNet Large Scale Visual Recognition Challenge in 2012, marking a pivotal moment. It deepened the network, utilized Rectified Linear Units (ReLUs) for activation, and used GPU acceleration, significantly boosting performance. Following this, VGGNet [8] emphasized the importance of depth in CNNs, advocating for more layers. Inception models, such as GoogLeNet [9], explored the benefits of inception modules, employing parallel filters of varying sizes. The introduction of residual connections by ResNet [4] brought about a revolutionary idea of residual learning, enabling the training of extremely deep networks by introducing skip connections. DenseNet [10] proposed an architecture in which each layer receives direct input from all preceding layers, fostering feature reuse and compact representation. The MobileNet [11] series focused on lightweight models suitable for mobile and edge devices, employing depth-wise separable convolutions. EfficientNet [12] introduced a holistic approach by optimizing network depth, width, and resolution simultaneously for enhanced efficiency.

These advancements, coupled with transfer learning and fine-tuning techniques, have propelled CNNs to remarkable performance, making them indispensable in various image classification applications. However, the research of [13], introducing the Transformer architecture, presents a notable shift in image classification. Adapting Transformers to vision tasks led to the development of ViT models. One fundamental departure from CNNs is that ViTs operate on the entire image as a sequence of patches, eliminating the need for hierarchical features. Each patch is treated as a "token", and the self-attention mechanism allows the model to capture global dependencies efficiently [1].

The ViT architecture consists of stacked transformer blocks, performing self-attention and feedforward operations. Several modifications and improvements have since been proposed to enhance ViTs' performance in image classification, such as patch embeddings, positional encodings, and hybrid architectures. ViTs have shown competitive results on large-scale image datasets, for example, ImageNet, challenging the supremacy of CNNs [14].

This evolution in computer vision emphasizes the importance of attention mechanisms and the potential to replace or complement CNNs in diverse image-related tasks.

The landscape of CNN models has also diversified, with ongoing research exploring hybrid architectures, attention mechanisms, and efficiency improvements. Models including Swin Transformer [15] and Twins-SVT [16] continue to push the boundaries of image classification capabilities, highlighting the ever-evolving nature of DL architectures. Benchmarking these models provides a systematic and objective means to evaluate and compare their performance in terms of accuracy, computational efficiency, and suitability for real-world applications, particularly in medical image classification tasks such as COVID-19 detection from chest X-ray images.

The authors of [17] introduced an explainable AI system for early COVID-19 detection from chest X-ray (CXR) images by comparing a limited number of CNN and ViT models. The system, designed for medical professionals, visualizes infected areas in CXR images, enhancing decision support. The experimental results demonstrate comparable performances between CNN and ViT models, with the EfficientNetB7 model achieving the highest accuracy. Incorporating UNet-based segmentation and rotation augmentation contributes to the system's robustness and overall performance improvement.

The study [18] comprehensively compares the performance of CNNs, ViTs, hybrid systems, and ResMLP architecture on fundus images for the detection of glaucoma. The comparative analysis reveals similar performance between CNNs and ViTs in the test set. However, external test sets, particularly Drishti-GS1 and PAPILA, indicate CNNs' superior generalization capacity. The study emphasizes the significance of dataset size and composition, suggesting ViTs' potential superiority with a more extensive training set.

A recent study [19] presented a comprehensive literature review comparing ViTs and CNNs for image classification. The comparison involves dataset characteristics, robustness, performance, evaluation, interpretability, and architecture. ViTs exhibit promising performance in various applications, including pneumonia detection, breast ultrasound classification, and skin cancer classification. However, challenges such as computational complexity and generalization issues are noted. The study suggests that ViTs are efficient on smaller datasets due to self-attention but may require more data for better generalization.

The research work of [20] addresses the challenge of auto-focusing in Digital Holography (DH) using DL, specifically comparing CNNs and ViTs. The results indicate that EfficientNetB7, a CNN architecture, outperforms ViT models, achieving better accuracy in recognizing specific locations within holograms. CNNs focus on shape details, but ViTs demonstrate robustness, particularly when considering arbitrary regions of interest.

The comparative study [21] explores the classification performance of CNNs and ViTs on small datasets, critically analyzing their suitability for image classification tasks. The study finds that Xception is suitable for high-performance tasks with limited data, ViT for large-scale data training, and ShuffleNet-V2 for scenarios prioritizing storage space over classification performance. This comparison underscores the need for further research to refine network performance through novel data augmentation techniques and explore hybrid CNN–Transformer models to enhance classification efficacy.

A unified, comprehensive framework is crucial as it standardizes the implementation and benchmarking of various models, simplifies comparisons, ensures reproducibility, and accelerates advancements by providing a common baseline for researchers. This integration streamlines the evaluation process and enhances the accessibility and scalability of model deployment across different applications. Further, benchmarking CNN and ViT models using this framework is critical in evaluating the framework and refining the selection criteria for image classification tasks, considering all these rich, diverse sets of related work and the rapid advancements of DL models. The findings help identify models that are accurate, efficient, and relevant for applications in different domains.

4. ConVision Benchmark Process

The methodology for benchmarking CNN and ViT models in this research is shown in Figure 1. It involves a systematic and comprehensive process to assess their performance on image classification tasks related to COVID-19 detection. The dataset was carefully chosen, with an unbalanced representation of COVID-19 cases, non-COVID-19 cases, and normal lung conditions. We employed image augmentation techniques such as rotations and scaling to create a robust dataset for evaluating models in a real-world scenario. The models, implemented using the PyTorch [6] framework, were initialized without pre-trained weights to avoid leveraging prior learning. Using pre-trained models can significantly reduce training time and improve performance due to the rich feature representations learned from large-scale datasets. However, training models from scratch ensures that performance is solely attributed to the dataset and training process, providing a clear baseline for evaluating different architectures. Subsequently, the models underwent a training phase using the designated training set, allowing for the adaptation of parameters to the specific characteristics of the dataset. Fine-tuning enhances model performance, and detailed information on training parameters is presented. After completing the training phase, the models were evaluated using the testing set, and a suite of performance metrics was calculated. These metrics include top-1 accuracy, precision, recall, F1 score, Matthew's Correlation Coefficient (MCC), False Positive Rate (FPR), False Negative Rate (FNR), and loss, providing a nuanced understanding of the models' classification capabilities. In addition to classification performance, the benchmarking process incorporated computational efficiency metrics such as model training time, CPU and GPU latency, number of parameters, multiply–accumulate operations (MACs), floating-point operations (FLOPs), training memory, and inference memory to assess the models' feasibility for integration into clinical workflows, where both accuracy and speed are pivotal. These metrics ensure a holistic evaluation considering both the efficacy and efficiency of CNN and ViT models. The comparison between CNN and ViT models involves a detailed analysis of their trade-offs, aiming to identify models that effectively balance accuracy and computational efficiency, providing insight into their suitability for real-time image classification tasks.

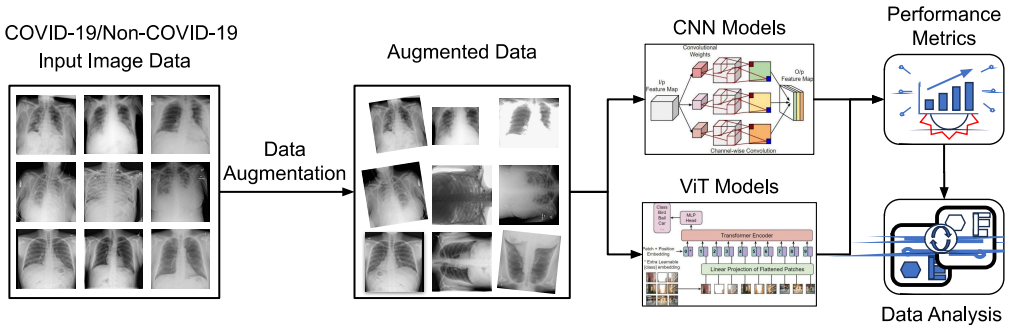


Figure 1. Block diagram of benchmark process for CNN and ViT models.

4.1. Dataset

The COVID-QU-Ex Dataset [22] is a comprehensive resource used for the benchmark process in this research. This extensive dataset includes 33,920 CXR images, providing various cases crucial for training and evaluating our models. The dataset is classified into three primary categories: 11,956 images of COVID-19 cases, 11,263 images representing non-COVID-19 infections, including viral or bacterial pneumonia, and 10,701 images of normal lung conditions. The dataset includes ground-truth lung segmentation masks, making it the most extensive dataset for lung masks ever. The dataset layout ensures a robust experimental framework for training, validation, and testing across various conditions, contributing to the reliability and generalizability of the research findings.

In our experimental setup, we used an unbalanced case by taking a subset of samples from the above dataset to test the system's robustness in scenarios where images are acquired under specific settings. The resilience and efficacy of our system were rigorously

evaluated using this imbalanced dataset, ensuring its applicability across diverse image settings and scenarios. The dataset, along with the associated codes and experiments, are made publicly available for further research and validation purposes. Table 1 illustrates each class's total number of samples. Figure 2 shows a sample CXR image for each class.

Table 1. COVID-19 dataset classification.

Split	Total	COVID-19	Non-COVID-19	Normal
Training	21,706	7658	7208	6849
Test	6788	2395	2253	2140
Validation	5417	1903	1802	1712

(**a**) (**b**) (**c**)

Figure 2. Sample CXR images: (**a**) COVID-19. (**b**) Non-COVID-19. (**c**) Normal.

4.2. Data Augmentation

Data augmentation is essential for enhancing the performance of DL models when training data are limited. This research implements a set of data augmentation techniques, even though the dataset is large enough. The rationale is that increasing the dataset size through augmentation can provide better training for ViTs, which tend to require more data to outperform CNNs.

In our research, we implemented a set of data augmentation techniques using PyTorch. These included resizing the images to a standard size (256), center cropping (224), random horizontal flipping with a probability of 0.5, random rotation up to 15 degrees, and color jittering for brightness, contrast, and saturation (each with a factor of 0.2). These transformations were designed to simulate variations encountered in real-world CXR images, such as different orientations, lighting conditions, and subtle changes in contrast. The transformation pipeline concludes with standard operations of converting the images to tensors and normalizing pixel values. These augmentation strategies exposed the model to diverse CXR image variations during training, preventing overfitting and improving its ability to generalize to new data.

4.3. CNN Processing

The preprocessing of input images in CNN models involves normalization, where the pixel values are scaled to a standardized range to facilitate consistent model training. In the case of gray-scale images, the single-channel representation is expanded to three channels to match the expected input format of CNN models. The images and corresponding labels indicating the disease category were divided into training, validation, and test sets. The training set updated the model's weights through a backpropagation algorithm coupled with a stochastic gradient descent (SGD) optimization technique. The validation set helped monitor and prevent overfitting during the training process. The test set was used to evaluate the performance of the final model on unseen data. This meticulous processing workflow ensured that the CNN model could effectively learn discriminative features for accurate COVID-19 classification while being robust to variations in the input data.

Our study evaluated the performance of various CNN model families, as shown in Table 2. The architectural details of the models used in this paper are not provided due to space constraints. The exact topology is described in the original papers cited below.

Table 2. Family of CNN models.

AlexNet [2]	ConvNext [23]	DenseNet [10]	EfficientNet [12]
Ghost-resnet [24]	GhostNetv2 [25]	Inception [26,27]	MNASNet [28]
MobileNet [11,29,30]	NFNet [31]	RegNet [32]	ResNet [4]
ResNext [33]	Wide-ResNet [34]	ShuffleNetv2 [35]	SqueezeNet [36]
VGG [8]			

4.4. ViT Processing

Preprocessing input images in ViT models involves several distinct steps compared to CNNs. Initially, images are resized to a standard resolution and then divided into fixed-size patches, typically 16×16 pixels. Each patch is flattened into a vector and linearly embedded into a higher-dimensional space. Unlike CNNs, ViTs typically handle color images directly and do not require channel expansion for gray-scale images. ViTs treat the sequence of these embedded patches as input tokens to a transformer model. Each token is combined with positional encodings to retain spatial information, which is essential for the self-attention mechanism. Similar to CNN preprocessing, images are normalized to ensure a consistent range of pixel values. The dataset was divided into training, validation, and test sets, serving the same purposes as outlined in the CNN preprocessing section. This ViT preprocessing workflow enabled ViTs to effectively capture global context and complex patterns for accurate COVID-19 classification, demonstrating their capability to handle diverse image features and structures.

We evaluated various ViT versions in our work, as shown in Table 3. The architectural details of all the neural networks used in this paper are not provided due to space constraints. The exact topology is described in the original papers cited below.

Table 3. Family of ViT models.

BoTNet [37]	CaiT [38]	CCT [39]	CrossFormer [40]
CrossViT [41]	CvT [42]	DeepViT [43]	EdgeNeXt [44]
EfficientFormer [45]	FocalTransformer [46]	GC-ViT [47]	LeViT [48]
LVT [49]	Max-ViT [50]	MLP-Mixer [51]	MobileFormer [52]
PiT [53]	PoolFormer [54]	PVT [55]	Region-ViT [56]
SepViT [57]	Swin [15]	T2T-ViT [14]	TNT [58]
Twins [16]	VAN [59]	Vision Transformer [13]	

4.5. Training Description

The training pipeline for CNNs and ViTs involves several vital steps to train on a given dataset effectively. Setting up a training pipeline requires careful consideration of various parameters and configurations to ensure that the model converges to optimal performance. The training pipeline begins with loading the dataset and applying the necessary preprocessing steps, such as resizing images, normalization, and data augmentation. The network architectures are defined, specifying the network's layers, filters, and connectivity for CNN and attention mechanisms for ViT. Our training parameters included the learning rate of 1×10^{-1}, batch size of 256, and SGD optimization algorithm with momentum of 0.9. The learning rate schedule followed a cosine annealing pattern, with a warmup phase of 25 epochs. This dynamic adjustment helped the model converge faster initially and fine-tune more effectively later in training. Other hyperparameters, such as weight decay and gamma, were utilized. Epochs, which represent the number of times the entire dataset is processed, determine the duration of the training. We decided to run for 500 epochs empirically with the validation set used to monitor performance and prevent overfitting. All models in our study, including CNN and ViT, were trained using the "CrossEntropyLoss" function. Using "CrossEntropyLoss" ensures that the loss values across different models are directly comparable, thus providing a fair basis for evaluating model performance. The loss function was minimized during training, and the model's parameters were iteratively updated through backpropagation. Performance was evaluated using loss, top-1 accuracy, F1 score, precision,

recall, FPR, FNR, and MCC. To further understand the model's computational efficiency, we measured Training Time, MACs, FLOPs, CPU and GPU latency, and memory usage for training and inference.

The experiments were conducted on a Linux system with an AMD EPYC 7543 32-core processor, 512 GB hard disk, 1.5 TB RAM, and an NVIDIA A100 80 GB graphics accelerator card. The models were implemented using PyTorch [6] libraries in Python 3.8.

4.6. Evaluation Method

Evaluating model performance is critical in deep learning, especially for COVID-19 image classification tasks. Performance metrics are quantitative measures that provide information on neural network models' accuracy, efficiency, and reliability. Different tasks may require different metrics, and the choice of metrics depends on the problem's nature. This study employs a comprehensive set of performance metrics to benchmark CNNs and ViTs for COVID-19 image classification. These metrics gauge the models' abilities and facilitate a robust comparative analysis. The following are the performance metrics considered in this study, and their significance in benchmarking is described:

1. Top-1 accuracy: this metric represents the percentage of correctly classified instances among the total predictions. It is a fundamental metric in classification tasks, indicating how well the model performs in assigning the correct class to an input sample. In benchmarking, achieving a high top-1 accuracy is a primary goal, demonstrating the model's ability to make accurate predictions.
2. Precision: it measures the accuracy of positive predictions. It is calculated as the ratio of true positive predictions to the total predicted positives. High precision is essential when minimizing false positives is crucial, where precision reflects the model's ability to correctly identify instances of a specific class.
3. Recall: also known as sensitivity or true positive rate, it measures the ability of the model to capture all positive instances. It is calculated as the ratio of true positives to actual positives. High recall is essential when it is crucial to identify all instances of a particular class, even at the cost of more false positives.
4. F1 score: the F1 score is the harmonic mean of precision and recall. It provides a balanced measure between precision and recall, and is particularly useful when the class distribution is imbalanced. A higher F1 score indicates a better balance between precision and recall, reflecting a model's overall performance.
5. FPR: it measures the proportion of actual negatives that are incorrectly predicted as positives. It is calculated as the ratio of false positives to the total actual negatives. In applications where minimizing false positives is critical, a lower FPR is desirable.
6. FNR: it measures the proportion of actual positives that are incorrectly predicted as negatives. It is calculated as the ratio of false negatives to the total actual positives. In scenarios where avoiding false negatives is crucial, a lower FNR is desired.
7. MCC: it provides a correlation coefficient between the observed and predicted binary classifications by considering all four confusion matrix values (true positives, true negatives, false positives, and false negatives). It is beneficial in imbalanced datasets.
8. Loss: the loss function quantifies the difference between predicted and actual values. During training, the goal is to minimize this value. It serves as an optimization objective, guiding the model to make better predictions. In benchmarking, comparing the loss across models helps understand their relative performance.

The following computation efficiency metrics are considered in this study, and their significance in benchmarking is described below:

1. Model training time: it is the total time required to train a model on a dataset. It is a practical metric for assessing the efficiency of different models. When dealing with large datasets or resource-intensive models, shorter training times are desirable.
2. CPU and GPU latency: latency measures the time taken for a model to process a single input sample. CPU and GPU latency are essential metrics for real-time applications.

Lower latency is critical for applications where predictions must be made quickly, such as in autonomous vehicles or interactive systems.

3. Number of parameters: it reflects the complexity of a model. While more parameters can lead to a more expressive model, it also increases the risk of overfitting, especially in the presence of limited data. Comparing models based on the number of parameters in benchmarking helps balance complexity and generalization.
4. MACs: they represent the number of multiply–accumulate operations performed during inference. They offer insights into the computational efficiency of a model. Lower MACs are generally desirable for applications with computational constraints.
5. FLOPs: they measure the number of floating-point operations performed during inference. FLOPs, similar to MACs, provide information about the computational workload of a model. Lower FLOPs are preferred for limited computational resources.
6. Training memory and inference memory: these metrics quantify the memory requirements during training and inference. Lower memory requirements are crucial for deploying models on resource-constrained devices. In benchmarking, understanding the memory footprint helps assess a model's practical usability.

5. Results and Discussion

5.1. COVID-19 Dataset

This section presents the results of benchmarking various CNNs and ViTs on the COVID-19 image classification task Our comprehensive evaluation included performance metrics such as accuracy, precision, recall, and F1 score, alongside computational efficiency metrics like model runtime, FLOPs, MACs, and CPU and GPU latency. By analyzing these metrics, we provide insights into the strengths and weaknesses of different model architectures and their applicability in real-world scenarios. The findings highlight vital trade-offs between accuracy and computational demands, offering valuable guidance for selecting optimal models for medical image classification and other domains.

Our analysis is presented in graph plots, making understanding the findings and relationship among different metrics easier. In all of the following graphs and in subsequent sections, a circle represents CNN models, a square represents ViT models, models within the green box are preferred models, and the size of the data point represents the number of parameters of the model. The model size for each of the models used are presented in Figure 3. Figure 4 shows the accuracy versus MACs. ViT-Small-patch8 stands out as a model with high MACs (16.75 billion) but relatively low accuracy (88.4%). This behavior is atypical, as higher MACs correlate with more complex models and better performance, indicating potential inefficiencies and overfitting in ViT-Small-patch8's design. Conversely, ShuffleNetV2-x0-5 exhibits low MACs (0.04 billion), yet achieves a high accuracy of 92.85%, which is unusual since simpler models with fewer MACs often struggle to maintain high accuracy. This finding suggests that ShuffleNetV2-x0-5 is exceptionally optimized, leveraging its architecture to achieve high accuracy efficiently. These anomalies highlight the importance of architectural innovations and careful tuning in achieving optimal performance in deep-learning models.

In our analysis of Figure 5, which shows the accuracy versus FLOPs, the relationship is not always linear and similar to accuracy versus MACs. A model such as ViT-Small-patch8, which exhibits high FLOPs (16.76 billion), has relatively low accuracy (88.4%) due to inefficiencies and challenges in effectively leveraging the computational resources. Conversely, the ShuffleNetV2-x0-5 model, having low FLOPs (0.04 billion), achieves high accuracy (92.85%) due to its efficiency and effectiveness in leveraging the available resources.

Figure 6 shows the relationship between accuracy and accuracy epoch, indicating the epoch at which the model achieved its highest accuracy during training. The relationship between epoch and accuracy is not strictly deterministic, as evidenced by the extreme cases observed. For instance, the PoolFormer-S36 model with a high epoch (484) and low accuracy (85.22%) could indicate convergence issues, overfitting, or suboptimal hyperparameter tuning, leading to a longer training time without significant accuracy

improvement. Contrarily, the RegNet-y-16gf model with a low epoch (45) reaches high accuracy (93%) due to efficient architecture design, effective regularization techniques, and optimal hyperparameter settings.

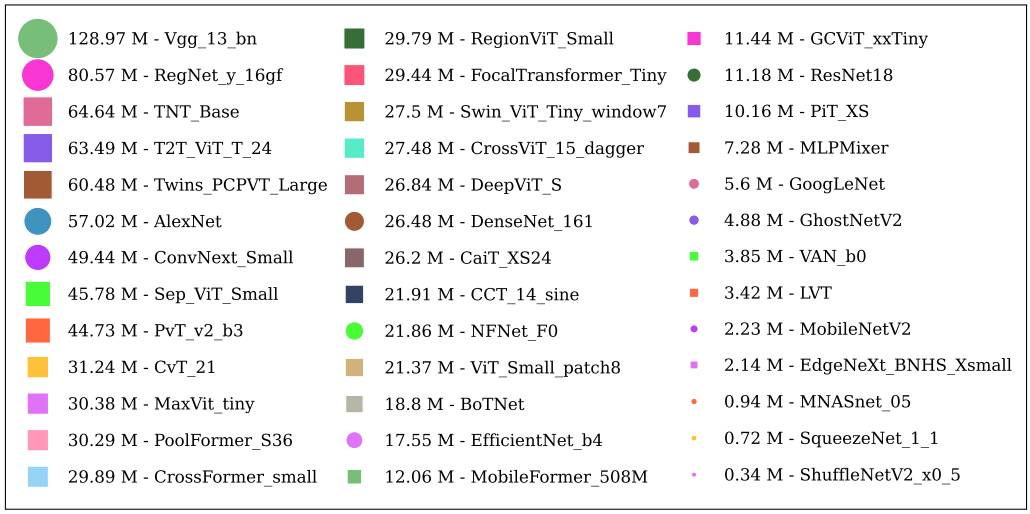


Figure 3. Model sizes of CNNs and ViTs.

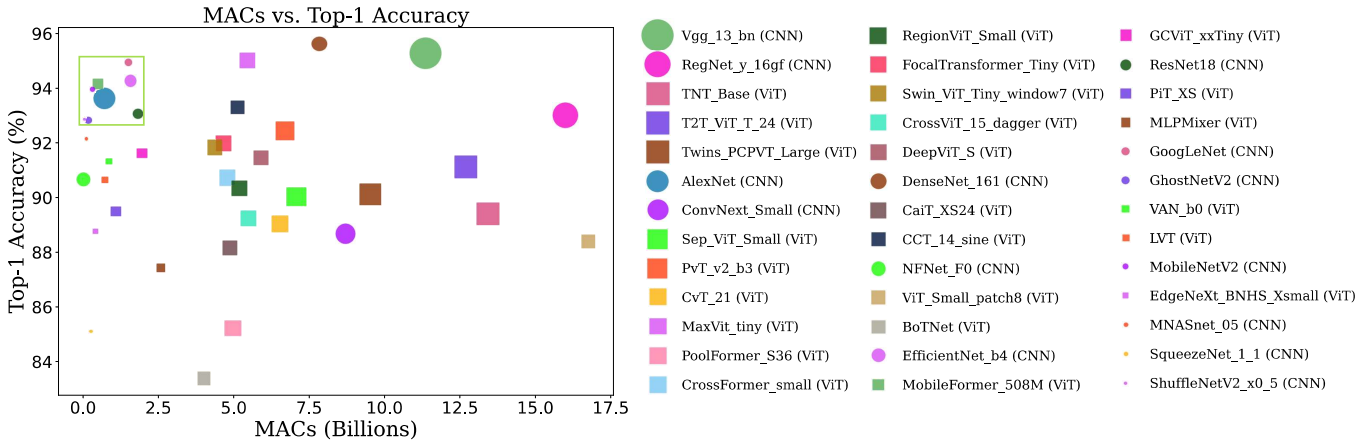


Figure 4. MACs vs. top-1 accuracy.

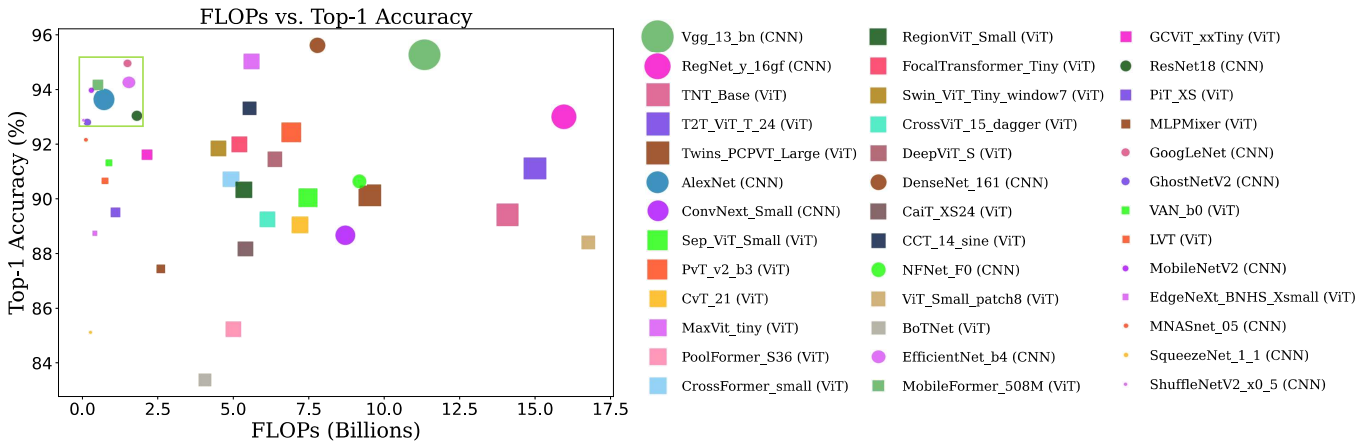


Figure 5. FLOPs vs. top-1 accuracy.

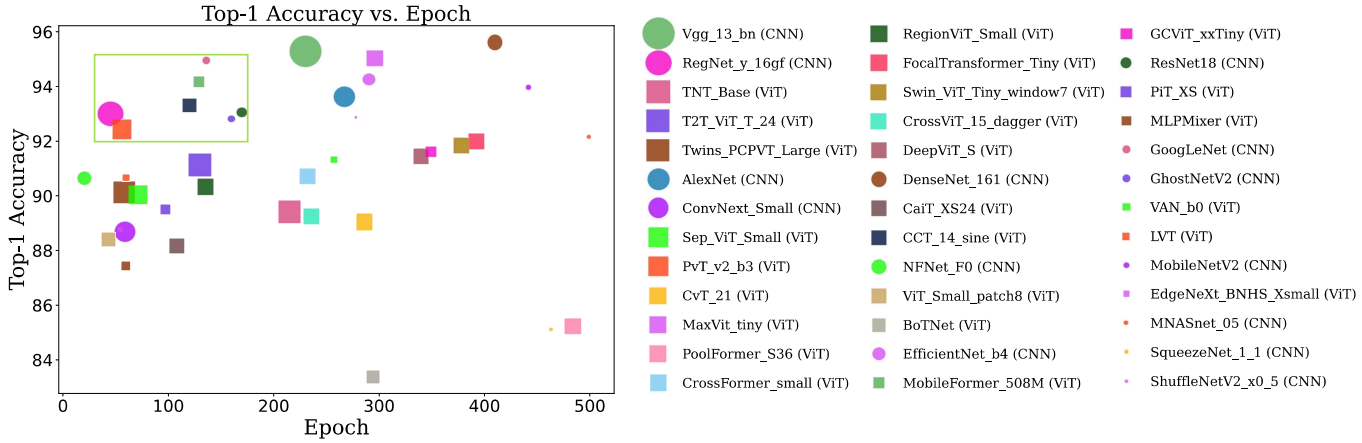


Figure 6. Top-1 accuracy vs. epoch.

Figure 7 shows the CPU latency versus accuracy. ViT-Small-patch8 exhibits high CPU latency (96.67 ms) and relatively low accuracy (88.4%). This anomaly suggests inefficiencies in model design or training, as higher latency typically indicates slower inference times, which can hinder real-time applications. MobileNetV2 showcases low CPU latency (9.13 ms) and high accuracy (93.97%). This exceptional performance is attributed to efficient architecture design and optimization, allowing faster inference times.

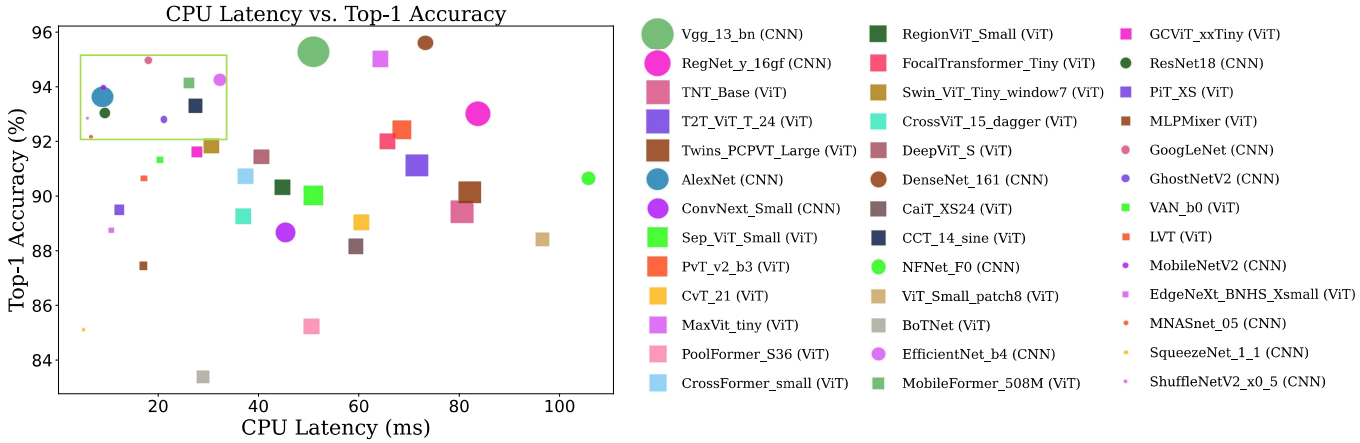


Figure 7. CPU latency vs. top-1 accuracy.

The GPU latency versus accuracy is shown in Figure 8. The CvT-21 model has high GPU latency (18.48 ms) but relatively low accuracy (89%). This behavior is not typical, as higher latency usually correlates with more parameters and better performance due to more complex computations. However, CvT-21's lower accuracy suggests inefficiencies and possible overfitting, where the model complexity does not translate to improved prediction quality. On the other hand, Vgg-13-bn demonstrates low GPU latency (1.08 ms) while achieving high accuracy (95.27%). Its performance indicates exceptional optimization in its design, effectively balancing computational efficiency with high predictive accuracy.

In the relationship between training memory and accuracy, as illustrated in Figure 9, models with more parameters generally require more training memory and often achieve higher accuracy due to their increased capacity to learn complex patterns. However, in our analysis, the NFNet-F0 model exhibits high training memory usage (1.69 GB) but relatively lower accuracy (90.64%). This discrepancy is due to overfitting and inefficiencies in the model's architecture. On the other hand, ShuffleNetV2-x0-5 stands out with low training memory usage (0.02 GB) while achieving a relatively high accuracy (92.86%).

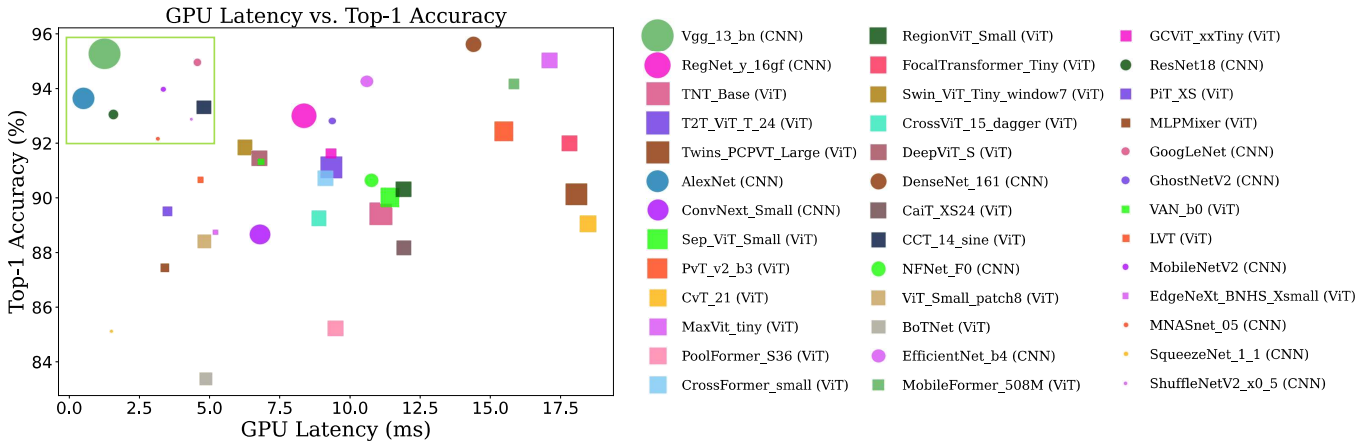


Figure 8. GPU latency vs. top-1 accuracy.

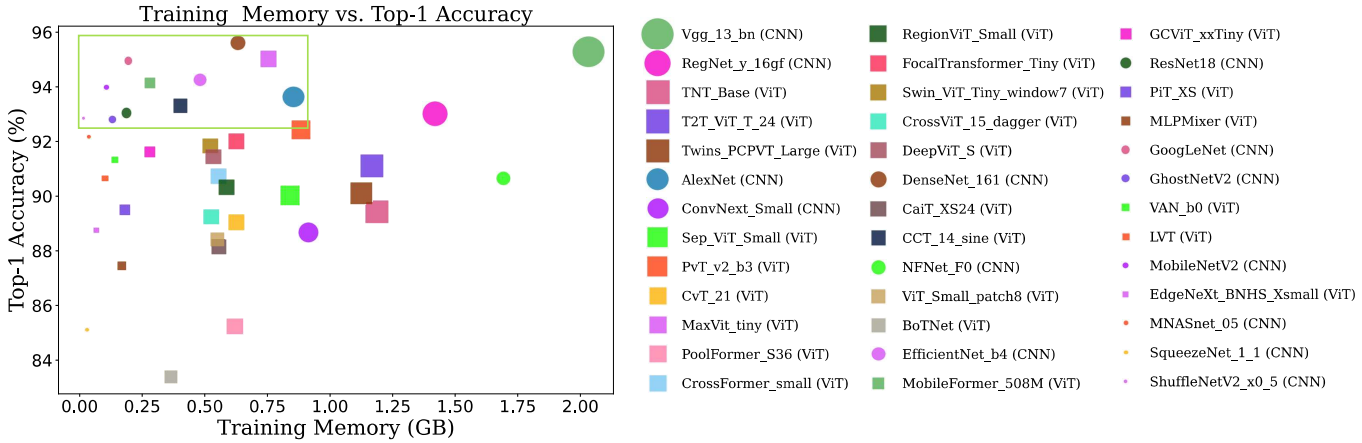


Figure 9. Training memory vs. top-1 accuracy.

Analyzing the relationship between inference memory, accuracy, and the number of parameters reveals interesting insights from the data. Figure 10 shows the inference memory versus accuracy graph. Typically, models with more parameters and inference memory tend to have higher accuracy due to their capacity to learn complex features. The trend is similar to the training memory versus accuracy graph but on a smaller scale, except for TNT-Base. TNT-Base has a high inference memory (247.23 MB) but a relatively lower accuracy (89.4%), suggesting inefficiencies and overfitting despite its capacity.

Figure 11 shows the training time versus accuracy. ViT-Small-patch8 exemplifies the case of high training time with low accuracy; it has a training time per epoch of 184.67 s, yet achieves a modest accuracy of 88.4%. On the other hand, GoogLeNet presents an interesting case of low training time with relatively high accuracy; it has a training time per epoch of only 17.22 s and achieves an accuracy of 94.94%. Typically, models with such quick training times do not achieve high accuracy due to the simplified architecture and fewer parameters. These exceptions highlight that model efficiency and architecture optimization play critical roles, sometimes allowing models with fewer parameters and faster training times to perform exceptionally well or causing models with high computational demands to underperform due to inefficiencies.

In examining the relationship between MCC, epoch, and the number of parameters from Figure 12, we observed some interesting extremes from the given data. Typically, a higher MCC correlates with more epochs and a more significant number of parameters due to the increased complexity and training duration improving model performance. However, the AlexNet model demonstrates an exceptional case of high MCC (indicative of high accuracy and balanced class prediction) achieved in relatively few epochs (19) and with a moderate number of parameters (57 Million). Conversely, the SqueezeNet-1-1

model, with fewer parameters (0.72 Million) and lower MCC, requires many epochs (443) to converge. These anomalies suggest that AlexNet is exceptionally efficient, optimizing its architecture to quickly achieve high performance, whereas SqueezeNet-1-1's prolonged training indicates difficulties in learning the dataset effectively.

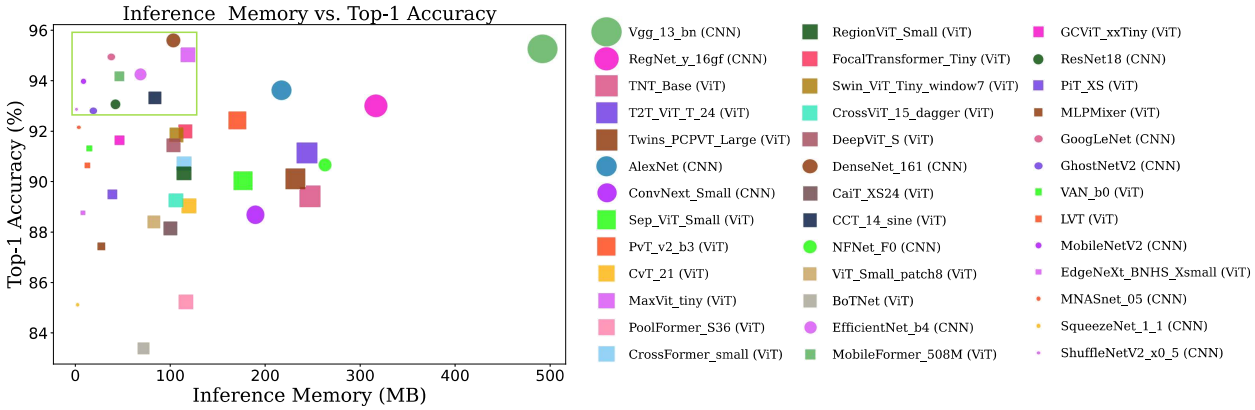


Figure 10. Inference memory vs. top-1 accuracy.

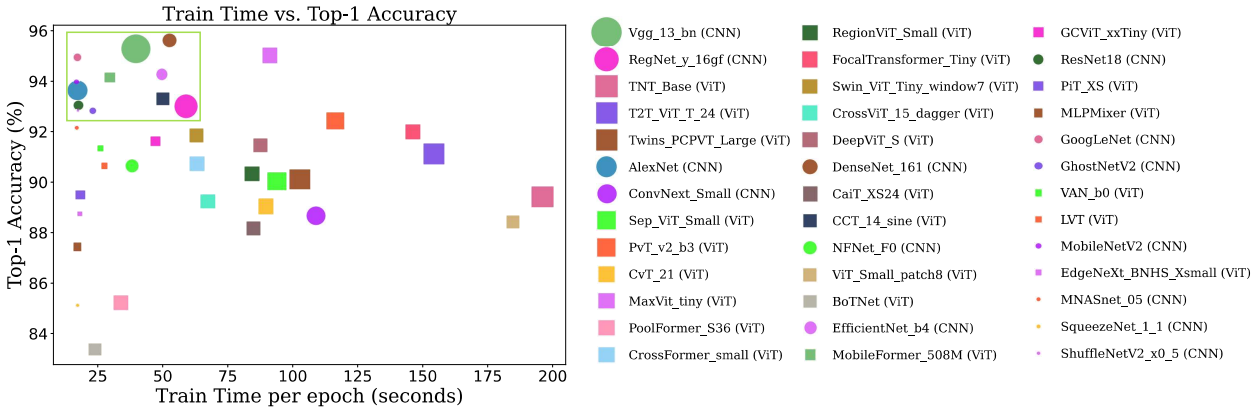


Figure 11. Training time vs. top-1 accuracy.

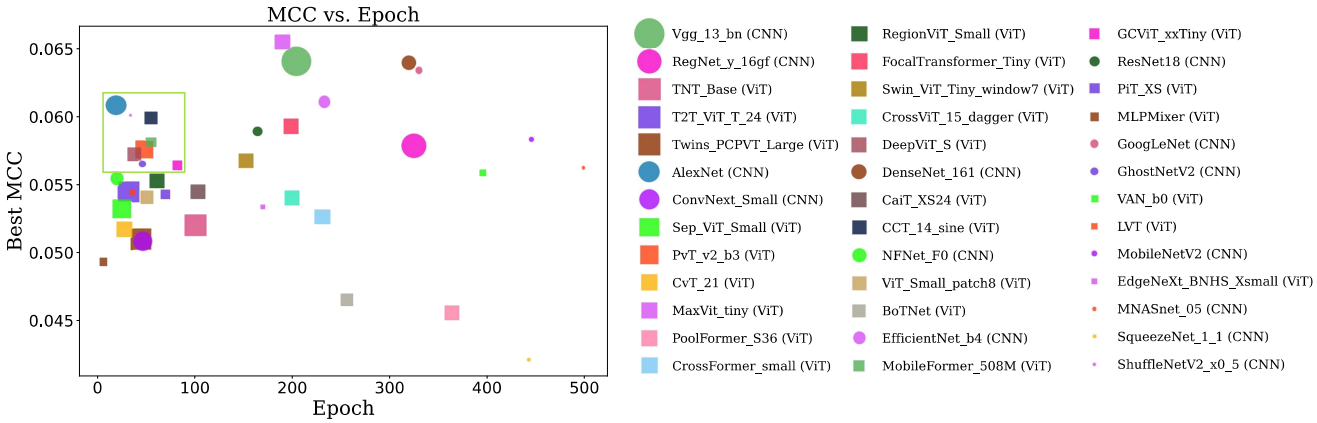


Figure 12. MCC vs. epoch.

The correlation between precision, epoch, and the number of parameters in deep-learning models follows a trend where models with more parameters often achieve higher precision but require more epochs to converge. This behavior suggests that larger models can capture more complex patterns in the data but also need more training to optimize all their parameters effectively. Figure 13 shows precision versus epoch. However, through analysis, CCT-14-sine, having 21.91 million parameters, achieves a relatively higher precision after 18 epochs. Another extreme case is the MNASnet-05 model, which has 0.94 million

parameters and reaches high precision after 487 epochs. These deviations are due to their architectural differences.

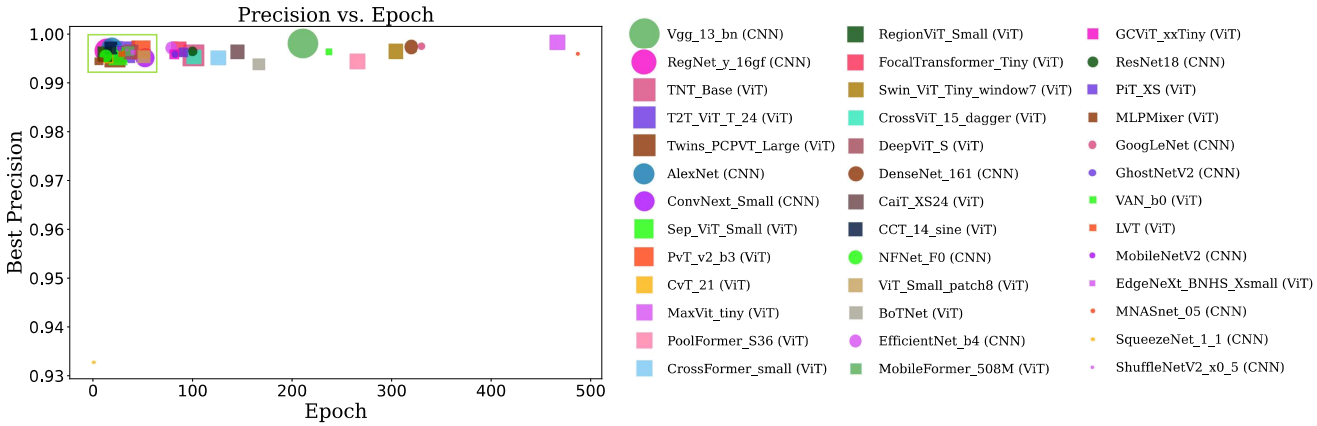


Figure 13. Precision vs. epoch.

Figure 14 shows the recall versus epoch. The RegNet-y-16gf model achieves a high recall of 0.93 with a relatively low epoch count of 45. This result indicates that RegNet-y-16gf converges quickly during training, reaching high performance rapidly, which is notable given its relatively high number of parameters (80.57 million). On the other hand, MNASnet-05, with a relatively lower recall of 92% and a high epoch count of 499, takes significantly longer to converge, despite having a low number of parameters (0.94 million). Typically, models with fewer parameters should train faster due to reduced computational complexity, and models with higher parameters often require more epochs to avoid overfitting. These observed extremes suggest that RegNet-y-16gf is highly efficient and well optimized, whereas MNASnet-05's prolonged training is due to suboptimal training dynamics, both deviating from the expected behavior of deep-learning models.

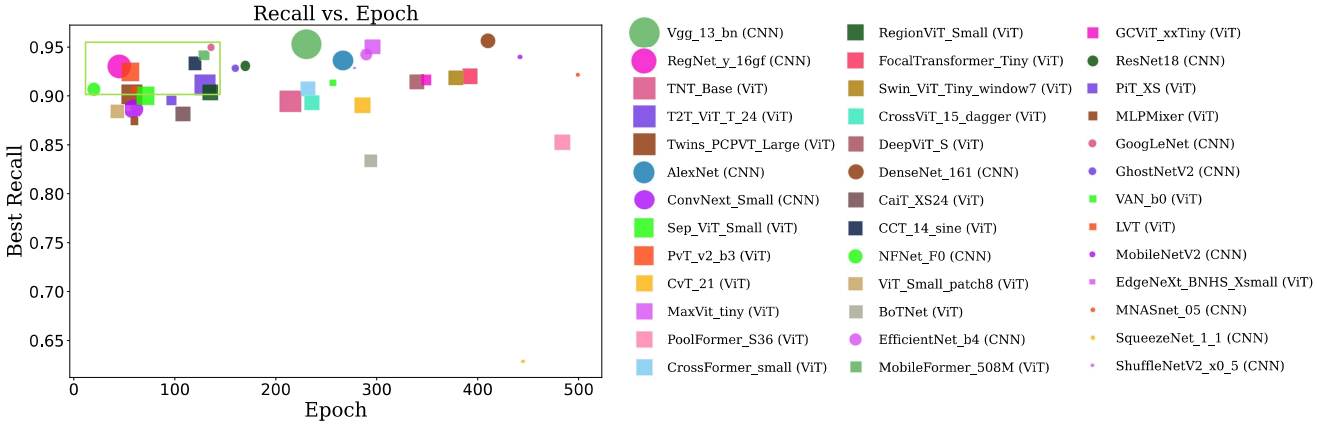


Figure 14. Recall vs. epoch.

Figure 15 shows the CPU latency versus MACs. In analyzing the relationship between CPU latency, MACs, and the number of parameters, two extreme cases illustrate abnormal behaviors in deep-learning models. The NFNet-F0 model exhibits high latency (105.72 ms) despite having low MACs (22.2 million) and a moderate number of parameters (21.85 million). Despite its computational simplicity, this high latency suggests potential inefficiencies in how the model processes data or the operations are structured. On the other hand, Vgg-13-bn showcases relatively low latency (50.98 ms) despite having relatively high MACs (11.35 billion) and a substantial number of parameters (0.12 billion). Typically, models with higher MACs have higher latencies due to the increased computational load.

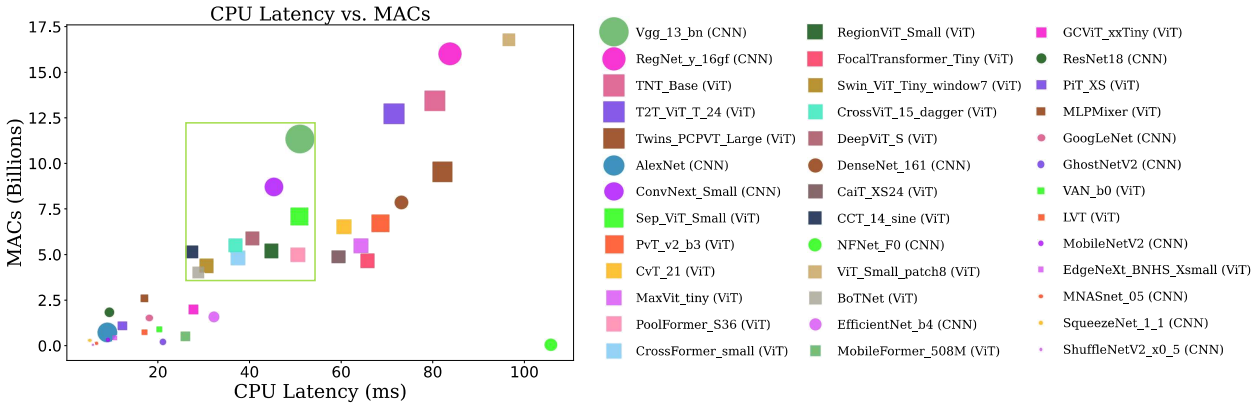


Figure 15. CPU latency vs. MACs.

Figure 16 shows the GPU latency versus MACs in which most of the models with higher MACs and more parameters exhibit higher GPU latency due to increased computational complexity. MobileFormer-508M, with a high GPU latency of 15.85 ms but low MACs (0.5 billion) and a moderate number of parameters (12.06 million), suggests inefficiencies possibly due to the suboptimal use of GPU resources or architectural bottlenecks. On the other hand, DenseNet-161, with a low GPU latency (14.4 ms), yet high MACs (7.84 billion) and a significant number of parameters (26.48 million), indicates an efficient architecture and excellent GPU optimization, resulting in faster computations despite its complexity.

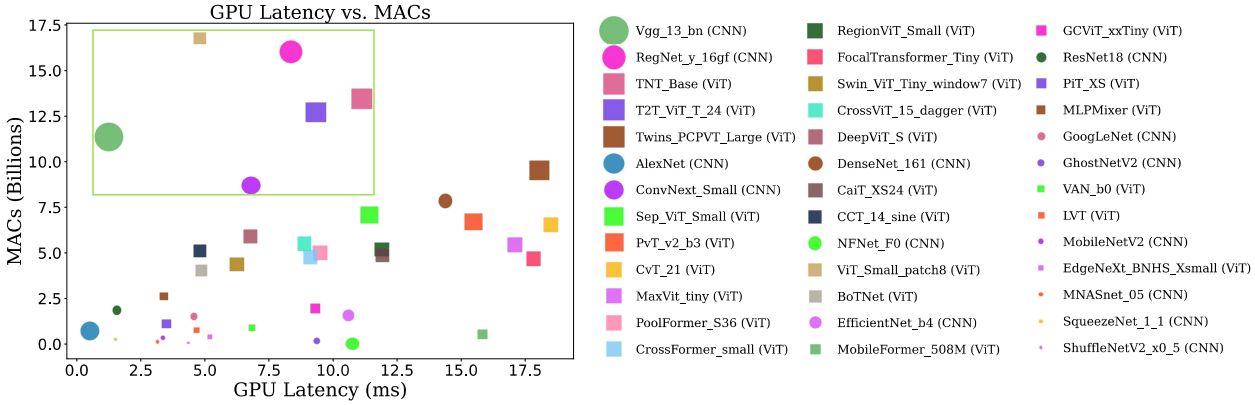


Figure 16. GPU latency vs. MACs.

Figure 17 shows the training time versus MACs. The FocalTransformer-Tiny model exhibits a high training time per epoch (146.15 s) despite having low MACs (4.66 billion) and a relatively small number of parameters (29.44 million). This unusual behavior is due to inefficient parallelization and bottlenecks in the data pipeline, making the training process slower than expected for its computational complexity. On the other hand, Vgg-13-bn shows a low training time per epoch (39.61 s) but has high MACs (11.35 billion) and a large number of parameters (128.97 million). This finding is interesting because models with higher MACs and parameter counts generally require more computational resources and time to train. The efficient training time for Vgg-13-bn suggests highly optimized implementations that minimize the expected computational burden.

In analyzing the relationship between training memory and MACs from Figure 18, we observed two extreme cases that defy typical expectations. The NFNet-F0 model has a high training memory usage of 1.69 GB but relatively low MACs at 22.2 billion. Generally, a high training memory correlates with a high number of MACs and parameters, reflecting the model's complexity. NFNet-F0's behavior indicates that it is highly parameter-intensive without proportional computational efficiency. On the other hand, ViT-Small-patch8 demonstrates low training memory usage (0.55 GB) but high MACs at 16.75 billion.

This result suggests that ViT-Small-patch8 is designed to be memory-efficient despite its computational complexity, leveraging advanced architectural strategies to optimize memory usage during training. The trend follows inference memory versus MACs except that the memory sizes are scaled down, as shown in Figure 19.

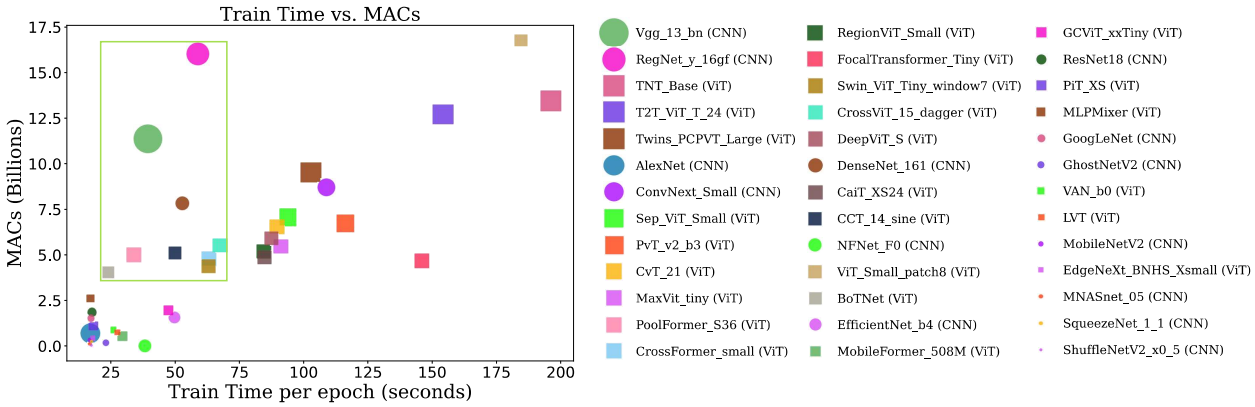


Figure 17. Training time vs. MACs.

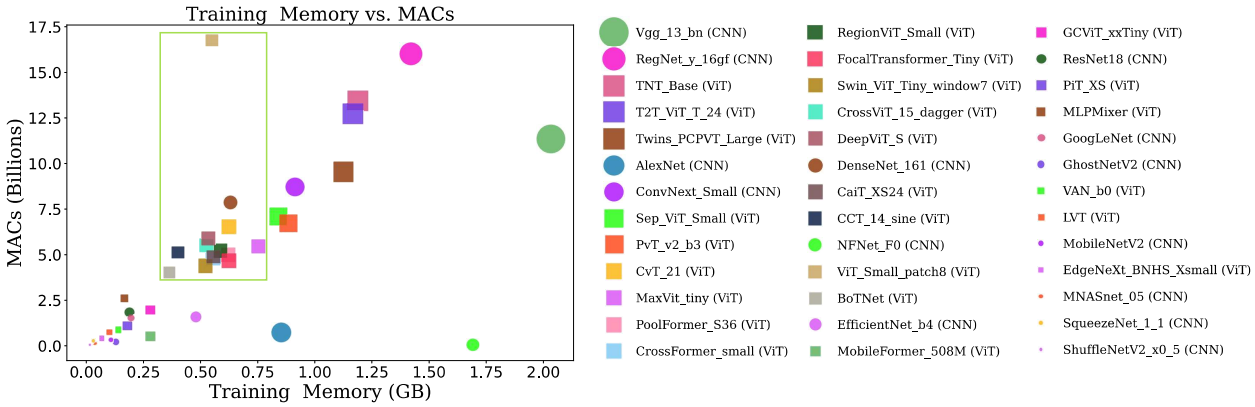


Figure 18. Training memory vs. MACs.

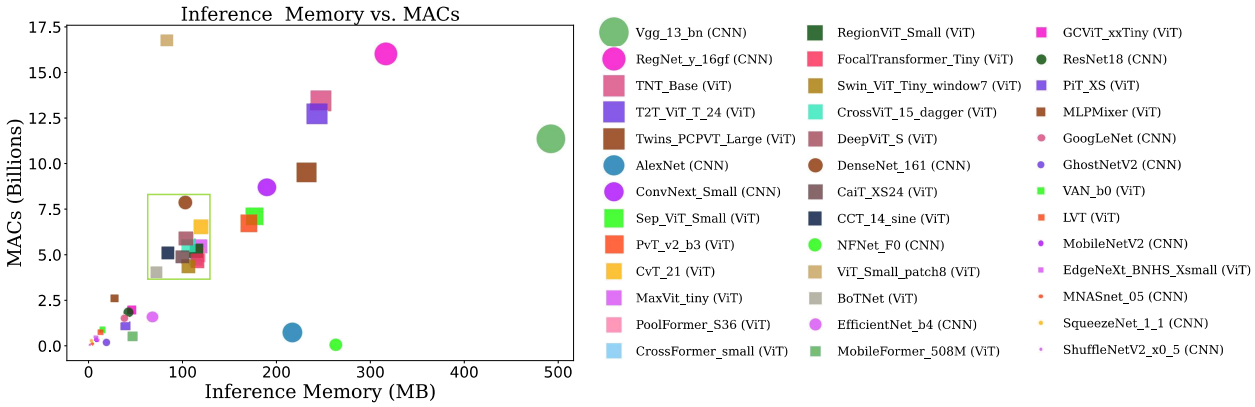


Figure 19. Inference memory vs. MACs.

Figure 20 shows the training time versus CPU latency. The Vgg-13-bn model exhibits a lower training time per epoch (39.61 s) with a relatively low CPU latency (50.98 ms) despite having a significant number of parameters (128.97 million). Typically, a complex model has a high training time and tends to have a higher latency due to the computational demands during inference. The NFNet-F0 model demonstrates a low training time per epoch (38.26 s), yet a high CPU latency (105.72 ms), despite having a minimal number of parameters (21.85 million). Usually, a model with fewer parameters and a short training time should

also have a lower latency, as simpler models usually require fewer computational resources. These deviations suggest that Vgg-13-bn is highly optimized for inference despite its complexity, while NFNet-F0 suffers from inefficiencies in its implementation that increase latency despite its simplicity.

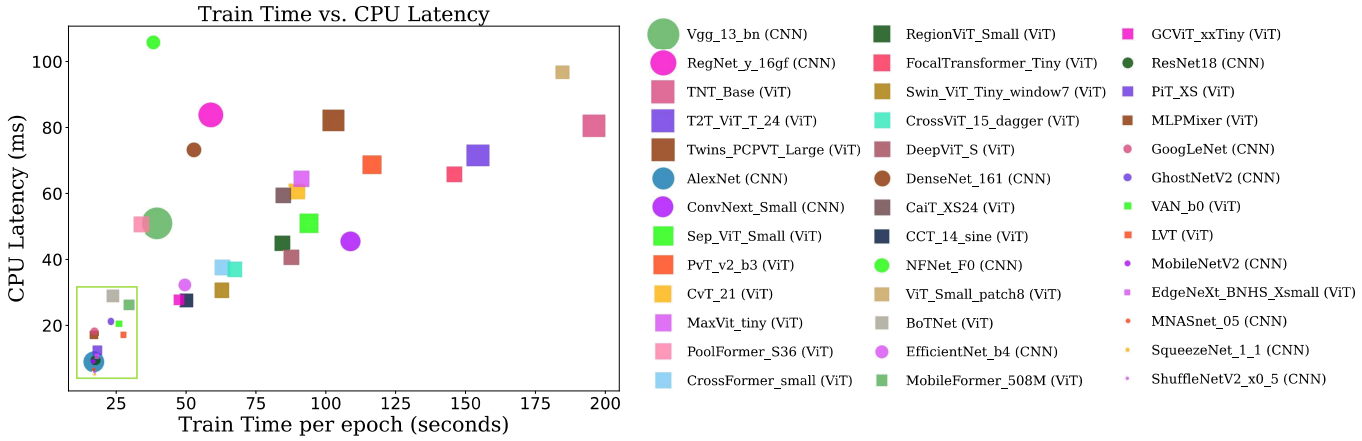


Figure 20. Training time vs. CPU latency.

Figure 21 shows the training time versus GPU latency. The ViT-Small-patch8 model presents an unusual scenario, with a high training time (184.67 s per epoch) but a relatively low GPU latency (4.80 ms). Usually, models with extended training times also exhibit higher latencies due to their complexity. However, ViT-Small-patch8's architecture incorporates efficient mechanisms for inference, leading to its lower latency despite prolonged training. On the other hand, MobileFormer-508M demonstrates a low training time (29.69 s per epoch) but a higher GPU latency (15.84 ms). This behavior is also atypical, as models with shorter training times generally have streamlined architectures that should result in lower latencies. MobileFormer-508M's higher latency stems from specific architectural components that, while reducing training time, do not optimize inference speed as effectively.

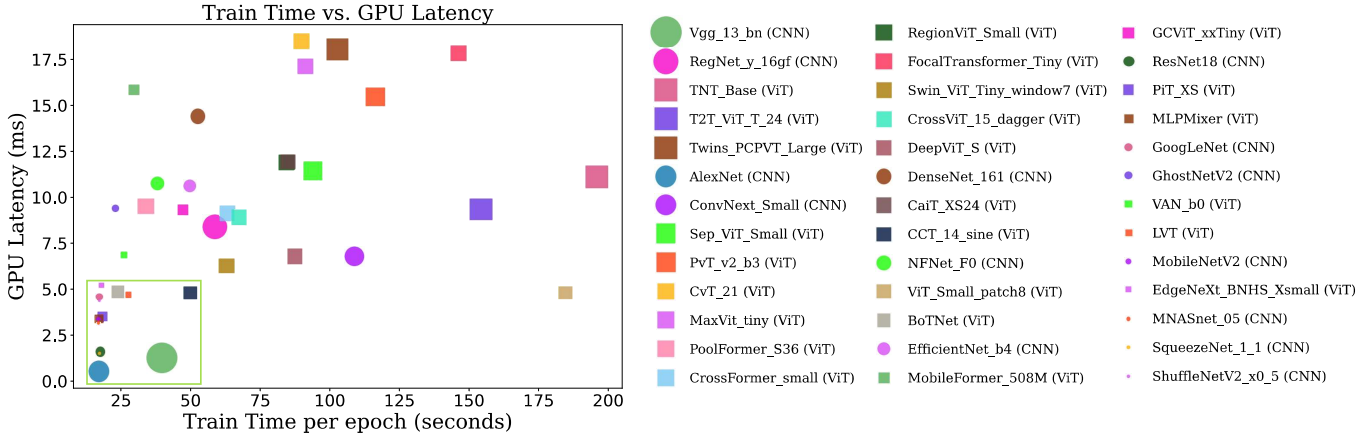


Figure 21. Training time vs. GPU latency.

The analysis of the training time versus training memory from Figure 22 shows that the ViT-Small-patch8 model has a high training time per epoch (184.67 s) but a relatively low training memory usage (0.55 GB), which is atypical since high training times usually correlate with higher memory usage due to the complexity of operations involved. Vgg-13-bn demonstrates a low training time per epoch (39.61) yet a high training memory usage (2.03 GB). This behavior is also abnormal because models with high memory usage typically involve extensive computations and, therefore, longer training times. A similar pattern

is observed for training time versus inference memory. The memory sizes are also scaled down, as shown in Figure 23.

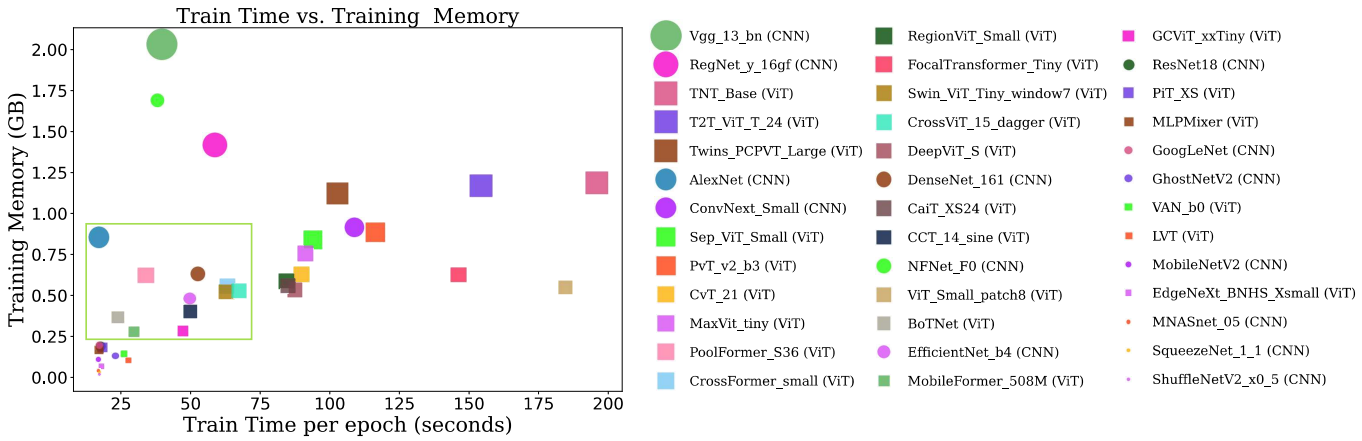


Figure 22. Training time vs. training memory.

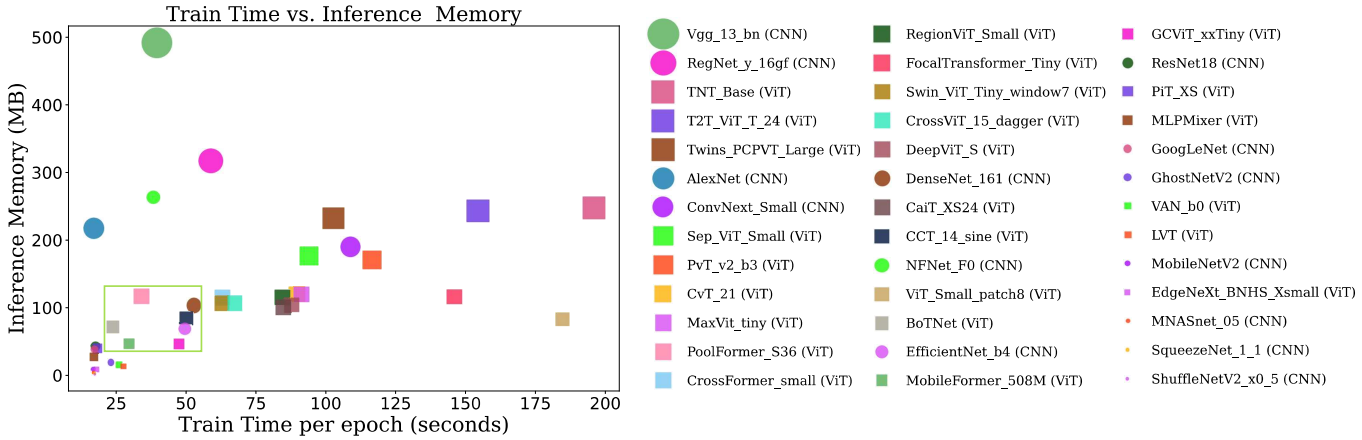


Figure 23. Training time vs. inference memory.

Figure 24 illustrates the training memory versus GPU latency, where models with more parameters tend to have a higher training memory usage and GPU latency. MobileFormer-508M has a high GPU latency (15.85 ms) but a relatively low training memory usage (0.28 GB). This is due to MobileFormer-508M's architecture having complex operations, causing delays despite its efficient memory footprint. Vgg-13-bn demonstrates a low GPU latency (1.26 ms) but a high training memory usage (2.03 GB), suggesting that Vgg-13-bn is optimized for rapid execution at the cost of increased memory consumption.

In general, a model's inference and training memory are influenced by its number of parameters. Figure 25 shows the training memory versus inference memory in which models with more parameters require more memory during training and inference. However, NFNet-F0 and AlexNet slightly deviate from this almost linear relationship by having lower and higher inference memories, respectively, compared to their training memory.

Table 4 provides the performance results of some high-accuracy CNNs and ViTs for COVID-19 detection among all the other trained models. The models were evaluated based on top-1 accuracy, recall, and loss, with corresponding epochs indicating the best performance for each metric. DenseNet-161 achieves the highest top-1 accuracy at 95.61%, with a recall of 0.96, demonstrating its robustness for high-precision medical diagnoses, albeit with increased computational complexity. Models like Vgg-13-bn and MaxVit-tiny also perform well, balancing accuracy and computational demands effectively.

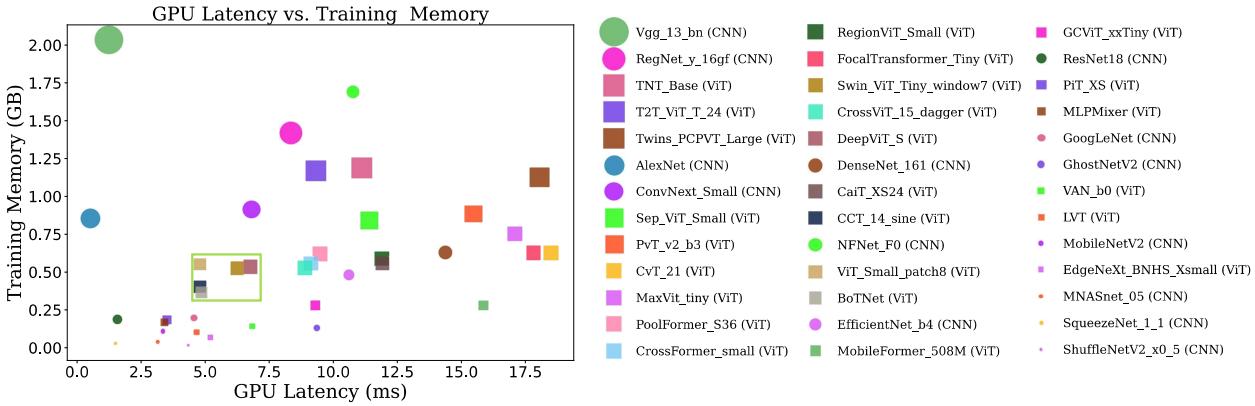


Figure 24. GPU latency vs. training memory.

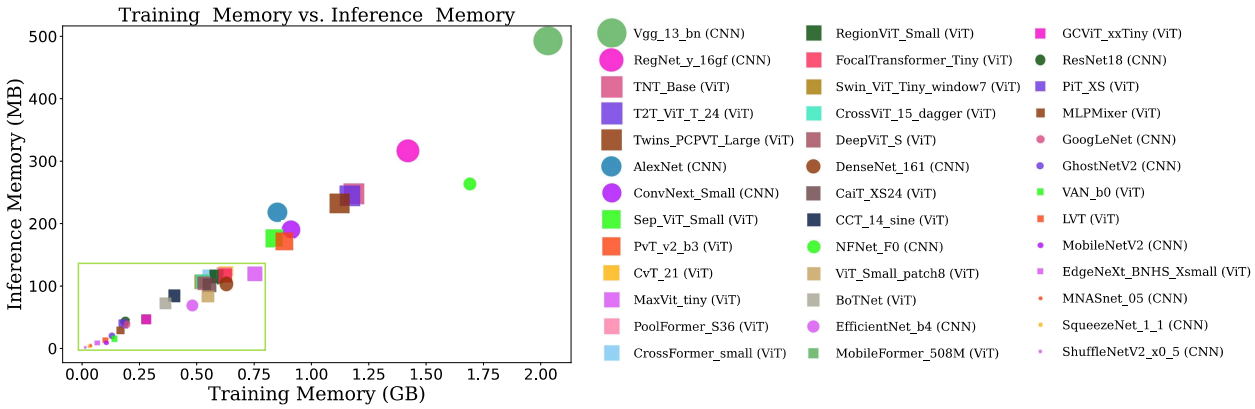


Figure 25. Training memory vs. inference memory.

Table 5 presents a detailed analysis of the computational efficiency of high-accuracy CNNs and ViTs used for COVID-19 detection among all the other trained models. Key metrics such as the number of parameters, multiply–accumulate operations (MACs), floating-point operations (FLOPs), training time per epoch, CPU and GPU latency, and memory usage during training and inference are compared across models. The findings highlight the trade-offs between accuracy and computational efficiency, guiding the selection of appropriate models based on specific application requirements. For instance, DenseNet-161 provides robust performance for high-precision diagnostics, while models like MaxVit-tiny and MobileFormer-96M are more suited for real-time deployment due to their lower computational demands. This analysis underscores the importance of considering accuracy and efficiency when choosing models for deployment in medical image classification and other computationally intensive tasks.

The results show that high-accuracy models (accuracy $> 90\%$) have minimal variation in performance metrics: F1 scores range from 0.94 to 0.98, recall from 0.90 to 0.96, precision stays nearly constant at 0.99, and MCC varies slightly from 0.04 to 0.06. This consistency is due to the models' robustness, optimization, and the large training dataset.

CNN models like AlexNet, GoogLeNet, and EfficientNet rely on convolutional layers for feature extraction. Factors such as depth, width, and skip connections (ResNet) influence their performance. EfficientNet uses compound scaling to balance network dimensions. DenseNet-161 achieves the highest accuracy at 95.61% and an F1 score of 0.98, indicating strong precision–recall balance, making it effective for medical image classification. However, its high computational complexity requires careful deployment consideration. In contrast, MobileNetV2 is efficient with the lowest parameters (2,227,715), low MACs and FLOPs, minimal training time, and low CPU and GPU latency, making it ideal for real-time applications in resource-limited environments.

Table 4. Performance results of CNN and ViT models for COVID-19 detection with accuracy $\geq$ 93%.

Model	Top-1 Accuracy (%)	Top-1 Accuracy Epoch	Recall (%)	Recall Epoch	Best Loss	Best Loss Epoch
DenseNet-161	95.61	410	0.96	410	0.21	226
Vgg-13-bn	95.27	230	0.95	230	0.16	39
DenseNet-121	95.26	401	0.95	401	0.23	204
DenseNet-169	95.24	327	0.95	327	0.2	12
Vgg-19-bn	95.08	187	0.95	187	0.18	24
MaxVit-tiny	95.02	296	0.95	296	0.17	26
Vgg-11-bn	95.01	293	0.95	293	0.17	27
Vgg-16-bn	94.99	131	0.95	131	0.2	28
GoogLeNet	94.95	136	0.95	136	0.25	16
DenseNet-201	94.9	383	0.95	383	0.24	11
Vgg-16	94.27	130	0.94	130	0.21	20
EfficientNet-b4	94.25	290	0.94	290	0.19	30
MobileFormer-508M	94.15	129	0.94	129	0.25	24
MobileFormer-294M	94.11	125	0.94	125	0.26	9
Vgg-11	94.08	154	0.94	154	0.22	12
EfficientNet-b0	94.03	253	0.94	253	0.2	26
Vgg-13	94	90	0.94	90	0.22	11
MobileNetV2	93.97	442	0.94	442	0.26	17
Vgg-19	93.89	176	0.94	176	0.2	17
EfficientNet-b1	93.74	371	0.94	371	0.22	31
EfficientNet-b3	93.64	154	0.94	154	0.21	31
MobileFormer-151M	93.62	134	0.94	134	0.23	16
AlexNet	93.62	267	0.94	267	0.22	19
MobileFormer-214M	93.58	444	0.94	444	0.25	22
EfficientNet-v2-s	93.41	361	0.93	361	0.22	28
CCT-14-sine	93.3	120	0.93	120	0.24	33
MobileNetV1	93.27	154	0.93	154	0.3	460
MobileFormer-96M	93.21	89	0.93	89	0.3	26
CCT-7-sine	93.12	170	0.93	170	0.27	30
ResNet18	93.05	170	0.93	170	0.29	443
EfficientNet-b2	93	257	0.93	257	0.27	26
RegNet-y-16gf	93	45	0.93	45	0.31	11

Another high-performance CNN model, RegNet-y-16gf, achieves a commendable accuracy, but its significantly higher number of parameters, MACs, and FLOPs result in longer training times and increased latency. These results suggest that, despite its accuracy, this model may be more suitable for scenarios where computational resources are not a limiting factor. Efficient models exhibit relatively lower computational demands while maintaining competitive performance.

ViT models split input images into fixed-size patches, embed them linearly, and process them with Transformer layers. Factors such as the number of layers, attention heads, and tokenization strategy affect ViT performance. Max-ViT achieves a top-1 accuracy of 95.02% but with higher parameters, MACs, and FLOPs, indicating a trade-off between accuracy and computational cost. MobileFormer, with a competitive accuracy (94.15%) and low parameters, achieves one of the shortest training times per epoch (17.61 s), making it ideal for real-time applications.

Comparing the overall performance of CNN and ViT models, the findings indicate that ViT models tend to have longer training times due to the self-attention mechanism, which involves processing the entire sequence at once. CNNs, especially smaller ones, converge faster. However, the accuracy of well-configured ViT models can be competitive with CNNs. ViT models can learn global context, making them suitable for tasks where understanding the entire input is crucial. At the same time, CNNs are known for their hierarchical feature extraction, which can be beneficial for capturing local patterns.

Table 5. Computation efficiency of CNN and ViT models for COVID-19 detection with accuracy $\geq$ 93%.

Model	Multiply–Accumulate Operations (million)	MACs (billion)	FLOPs (billion)	Training Time per Epoch (s)	CPU Latency (ms)	GPU Latency (ms)	Training Memory (GB)	Inference Memory (MB)
DenseNet-161	26.48	7.84	7.78	52.73	73.24	14.4	0.63	103.1
Vgg-13-bn	128.97	11.35	11.33	39.61	50.98	1.26	2.03	492.01
DenseNet-121	6.96	2.9	2.86	29.16	34.84	9.91	0.23	27.03
DenseNet-169	12.49	3.43	3.4	35.04	46.82	14.04	0.34	48.48
Vgg-19-bn	139.59	19.69	19.66	51.7	73.04	1.71	2.21	532.56
MaxVit-tiny	30.38	5.46	5.61	91.16	64.36	17.1	0.75	118.93
Vgg-11-bn	128.78	7.63	7.62	25.71	40.52	1.08	1.99	491.3
Vgg-16-bn	134.28	15.52	15.49	45.63	62.06	1.52	2.12	513.16
GoogLeNet	5.6	1.51	1.5	17.23	18.13	4.57	0.2	38.12
DenseNet-201	18.1	4.39	4.34	43.93	61.95	17.11	0.46	70.19
Vgg-16	134.27	15.47	15.47	35.48	60.87	1.3	2.07	513.09
EfficientNet-b4	17.55	1.58	1.54	49.72	32.27	10.61	0.48	68.25
MobileFormer-508M	12.06	0.5	0.51	29.69	26.1	15.85	0.28	46.47
MobileFormer-294M	9.51	0.29	0.29	23.48	21.14	15.82	0.21	36.69
Vgg-11	128.78	7.61	7.61	20.46	40.37	0.95	1.97	491.25
EfficientNet-b0	4.01	0.41	0.4	20.93	12.67	5.37	0.14	15.7
Vgg-13	128.96	11.3	11.3	30.26	49.86	1.08	1.99	491.96
MobileNetV2	2.23	0.33	0.31	16.88	9.13	3.36	0.11	8.76
Vgg-19	139.58	19.63	19.63	41.03	73.43	1.52	2.16	532.47
EfficientNet-b1	6.52	0.61	0.59	29.56	19.09	7.48	0.21	25.43
EfficientNet-b3	10.69	0.97	0.95	36.41	24.44	8.72	0.31	41.52
MobileFormer-151M	6.32	0.15	0.15	18.74	19.1	15.93	0.14	24.49
AlexNet	57.02	0.71	0.71	17.03	8.95	0.51	0.85	217.5
MobileFormer-214M	7.83	0.21	0.21	20.64	20.91	16.17	0.17	30.29
EfficientNet-v2-s	20.18	2.9	2.88	32.62	36.04	11.8	0.46	79.03
CCT-14-sine	21.91	5.12	5.53	50.1	27.48	4.8	0.4	84.13
MobileNetV1	3.21	0.59	0.58	16.7	7.17	1.71	0.09	12.35
MobileFormer-96M	3.31	0.1	0.1	17.61	14.24	12.3	0.08	12.88
CCT-7-sine	4.5	1.47	1.61	17.46	9.7	2.52	0.09	17.38
ResNet18	11.18	1.82	1.82	17.69	9.42	1.57	0.19	42.7
EfficientNet-b2	7.71	0.7	0.68	30.98	19.95	7.5	0.24	30.01
RegNet-y-16gf	80.57	16.01	15.96	58.87	83.78	8.36	1.42	316.53

ViT models generally exhibit higher MACs and FLOPs than CNNs because ViT processes the entire image as a sequence of patches, increasing computational requirements. CNNs, on the other hand, demonstrate more parameter-efficient designs, achieving competitive performance with fewer parameters compared to some ViT models. Additionally, CNNs tend to have lower latency during CPU and GPU inference than ViT models, partly due to the sequential processing nature of ViTs, which can lead to longer dependency chains. ViT models often demand higher training and inference memory as they process image patches independently, resulting in more significant intermediate representations.

These results support decision-making about which models might suit specific use cases based on computational requirements, memory consumption, and performance. The hierarchical and layered structure of CNNs and the self-attention mechanism of ViTs have enabled these models to achieve state-of-the-art performance in COVID-19 CXR image classification tasks. However, as the dataset grows and tasks become more complex, the limitations of the traditional convolutional approach and the challenges of integrating transformers into computer vision will need to be addressed through continued research and development.

5.2. Additional Dataset Analysis

To demonstrate the broad applicability of our framework across various domains in computer vision, we conducted performance evaluations on two additional datasets: the flower recognition [60] and Multi-class Weather datasets [61]. These evaluations provided insights into how both CNN and ViT models performed in different contexts, offering a comprehensive analysis that further substantiates our framework's adaptability. The results and analysis of these evaluations are detailed below, highlighting the strengths and weaknesses of each model type in handling diverse visual data and tasks.

5.2.1. Flower Recognition Dataset

The flower recognition dataset consists of 4242 labeled images of various flowers, sourced from Flickr, Google Images, and Yandex Images [60]. The dataset is divided into five classes: chamomile, tulip, rose, sunflower, and dandelion, each containing approximately 800 photos. The images are in different proportions and roughly have a resolution of 320 × 240 pixels, without being resized to a uniform dimension. This dataset is useful for training models to recognize and identify different types of flowers from photographs. Figure 26 shows sample images of the flower recognition dataset.

(**a**) (**b**) (**c**) (**d**) (**e**)

Figure 26. Sample flower images: (**a**) daisy; (**b**) dandelion; (**c**) rose; (**d**) sunflower; (**e**) tulip.

In examining the relationship between accuracy and epoch from Figure 27, we notice that GoogLeNet leads with the highest accuracy (78.01%), followed closely by MaxVit-tiny (75.93%) and then DenseNet-161 (73.73%). MaxVit-tiny achieves its peak performance faster (248 epochs) compared to GoogLeNet and DenseNet-161, both of which require 274 epochs. MNASnet-05 achieves an accuracy of 36.11% on the flower dataset, with its peak performance reached after 273 epochs of training. MNASnet-05 is designed with a focus on mobile efficiency, emphasizing reduced computational complexity and power consumption. While this makes it suitable for deployment on resource-constrained devices, it also means that the model lacks the depth and capacity to capture intricate patterns and features in the data as effectively as larger, more complex models.

When comparing training time versus accuracy from Figure 28, we find that GoogLeNet emerges as the most efficient, achieving the highest accuracy with a training time per epoch of just over 3.2 s. MaxVit-tiny and DenseNet-161 also deliver strong performances but they require significantly longer training times per epoch of approximately 11.75 and 7.21 s. Vgg-13-bn and MobileFormer-508M provide competitive accuracies of around 73.38% and 72.57%, with moderate training times per epoch of 5.56 and 4.47 s, making them efficient choices for accuracy relative to training time. In contrast, T2T-ViT-T-24 lags behind with an accuracy of 51.39%, despite an extensive training period of 19.38 s, indicating a less favorable trade-off between accuracy and training time compared to the other models.

In terms of FLOPs versus accuracy, the analysis of Figure 29 shows GoogLeNet as the most efficient, achieving the highest accuracy with only 1.5 billion FLOPs. MaxVit-tiny, with a slightly lower accuracy, requires significantly more computational power at 5.6 billion FLOPs. DenseNet-161 also has a high accuracy but with an even higher computational cost of 7.7 billion FLOPs. MobileFormer-508M strikes a balance with a good accuracy and a relatively low computational requirement of 5 billion FLOPs, making it an efficient choice. In contrast, T2T-ViT-T-24 has one of the lowest accuracies and the highest computational demand, indicating a less favorable trade-off between accuracy and

computational efficiency. Analyzing MACs versus accuracy from Figure 30, we observe that the behavior follows a similar trend as FLOPs versus accuracy.

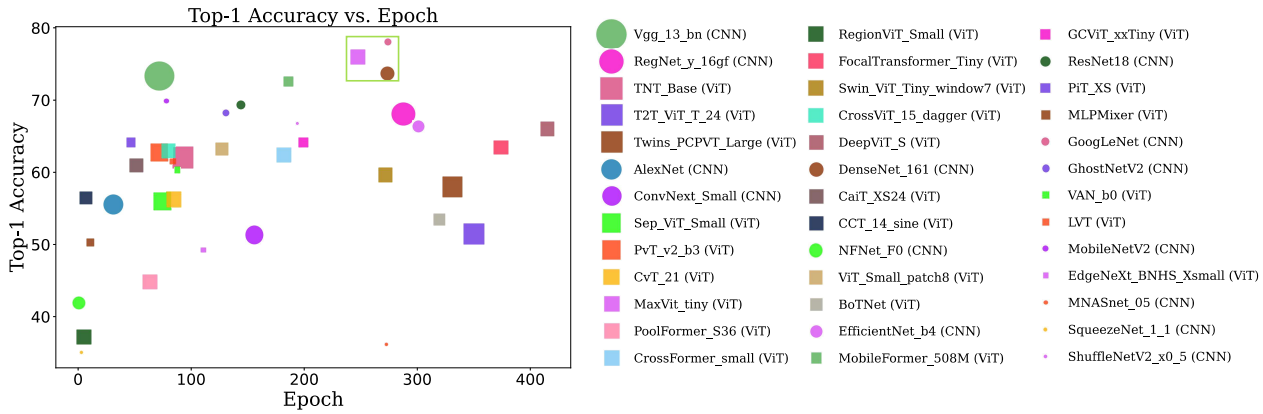


Figure 27. Top-1 accuracy vs. epoch.

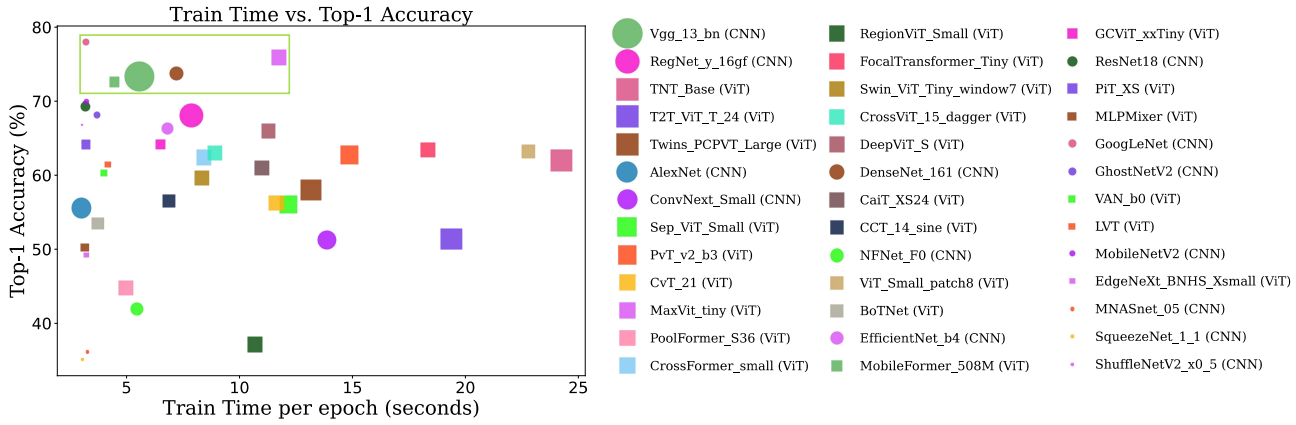


Figure 28. Training time vs. top-1 accuracy.

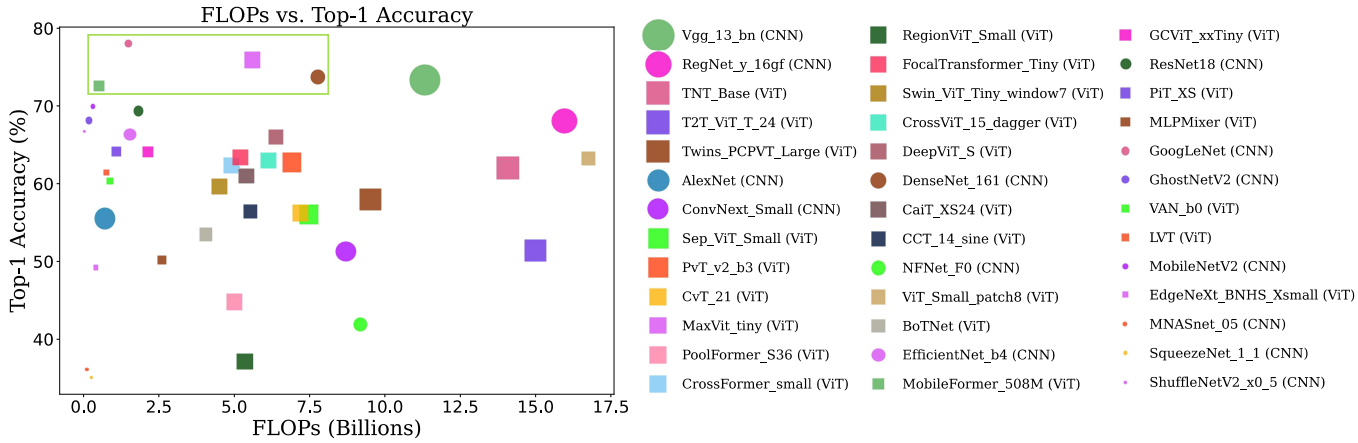


Figure 29. FLOPs vs. top-1 accuracy.

Finally, examining F1 score versus epoch from Figure 31 reveals that GoogLeNet and MaxVit-tiny show strong performance with high F1 scores, indicating their effectiveness in balanced classification tasks. GoogLeNet achieves the best F1 score but requires more epochs, while MaxVit-tiny provides a competitive F1 score with fewer epochs. In contrast, MNASnet-05 underperforms significantly, indicating potential issues with its architecture or suitability for the task.

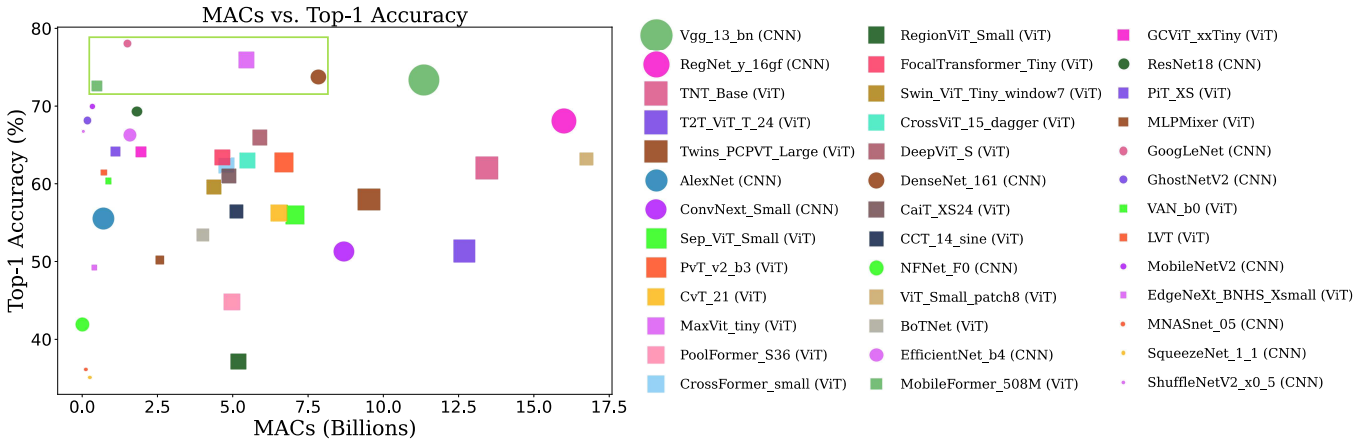


Figure 30. MACs vs. top-1 accuracy.

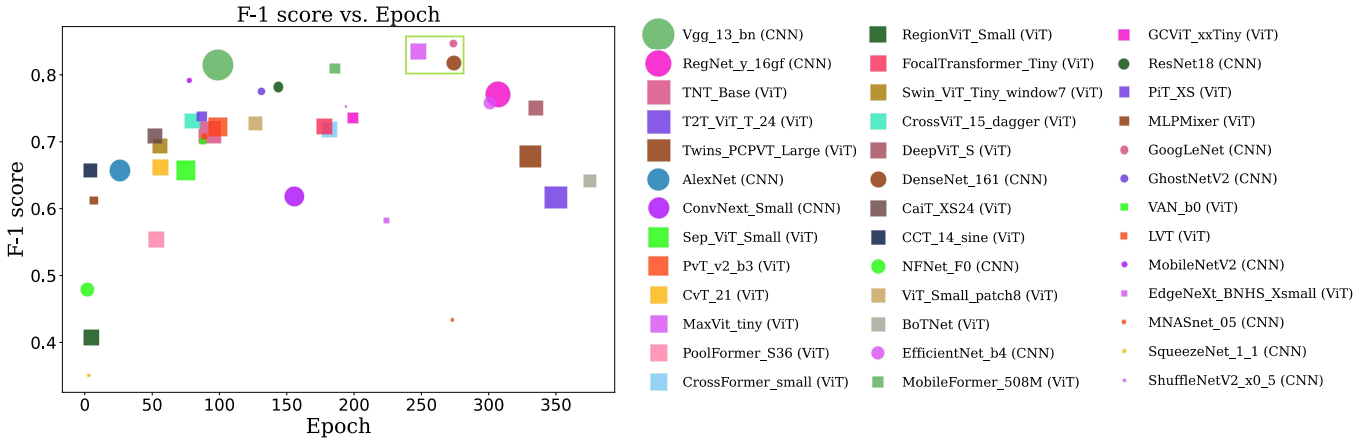


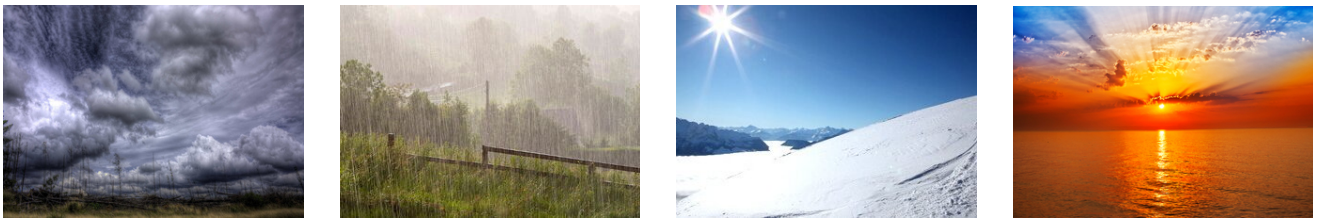
Figure 31. F1 score vs. epoch.

For the flower recognition task with the chosen dataset, GoogLeNet emerges as a consistently strong performer across multiple performance metrics, particularly in achieving a high accuracy with the efficient use of computational resources. MaxVit-tiny and DenseNet-161 also show strong performance in specific metrics, highlighting the importance of architecture design in balancing training time, accuracy, and computational efficiency.

5.2.2. Weather Classification Dataset

The Multi-class Weather Dataset [61] is designed for image classification tasks focused on recognizing various weather conditions. It includes 1125 images distributed across four categories: sunrise (357 images), shine (253 images), rain (215 images), and cloudy (300 images). This dataset supports outdoor weather analysis by providing a platform for extracting features to identify different weather scenarios. Figure 32 shows sample images of the weather classification dataset.

Figure 33 shows accuracy versus epoch. MaxVit-tiny achieves the highest accuracy at 98.21%, reaching this peak at epoch 338. The large number of epochs suggests that MaxVit-tiny benefits from extended training to fine-tune its parameters, leading to high accuracy. MobileNetV2 follows with an accuracy of 96.8%, obtained at epoch 52. MobileNetV2 converges much faster, indicating its efficiency in learning and adjusting its parameters early in the training process making it efficient for scenarios where training time is limited. MNASnet-05, however, shows considerably lower accuracy, suggesting that it may not be suitable for the task at hand.



(**a**) (**b**) (**c**) (**d**)

Figure 32. Sample weather images: (**a**) cloudy; (**b**) rain; (**c**) shine; (**d**) sunrise.

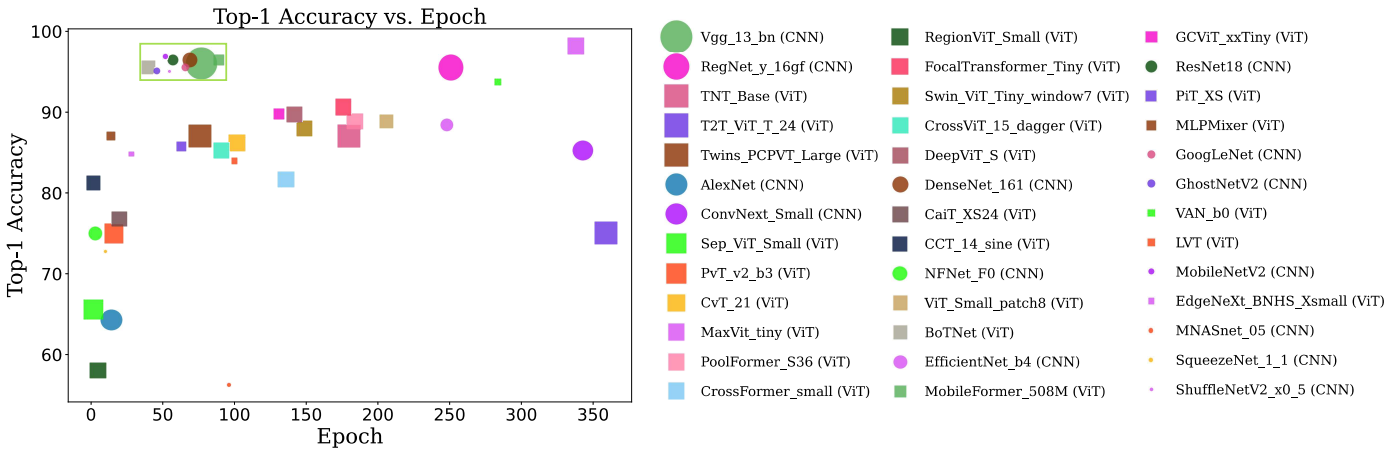


Figure 33. Top-1 accuracy vs. epoch.

The relationship between training time and accuracy is shown in Figure 34. ResNet18 achieves an accuracy of 96.43% with a training time per epoch of 1.97 s. This demonstrates ResNet18's ability to quickly and effectively learn from the data, offering high accuracy in a relatively short training period. MobileNetV2 slightly outperforms ResNet18 with a training time of 2.13 s. While it takes marginally longer to train, MobileNetV2's architecture, optimized for mobile and edge devices, provides a highly accurate model without a significant increase in training time. On the other hand, T2T-ViT-T-24 achieves a considerably lower accuracy of 75% and requires a much longer training time of 5.8 s.

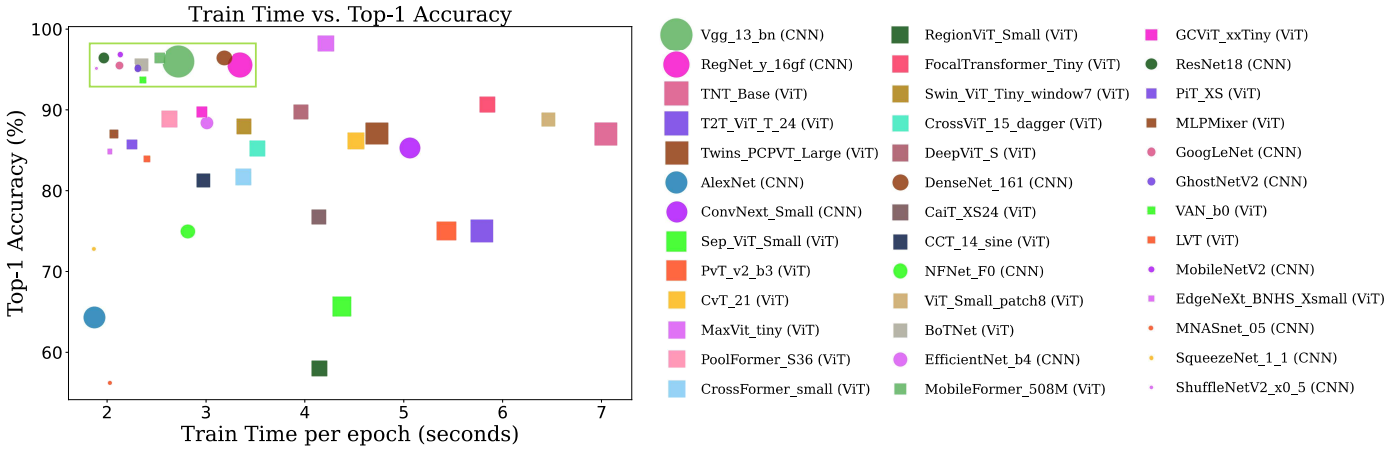


Figure 34. Training time vs. top-1 accuracy.

Figure 35 shows the FLOPs versus accuracy graph. MobileNetV2 stands out for its exceptional balance of high accuracy and low computational cost, making it highly efficient. MobileFormer-508M also provides a high accuracy but with a higher computational cost. T2T-ViT-T-24 exhibits a much lower accuracy and significantly higher computational requirements, indicating that it may not be the most efficient choice for tasks where compu-

tational efficiency and high accuracy are both critical. The trend of MACs versus accuracy, as shown in Figure 36, follows a similar trend as FLOPs versus accuracy.

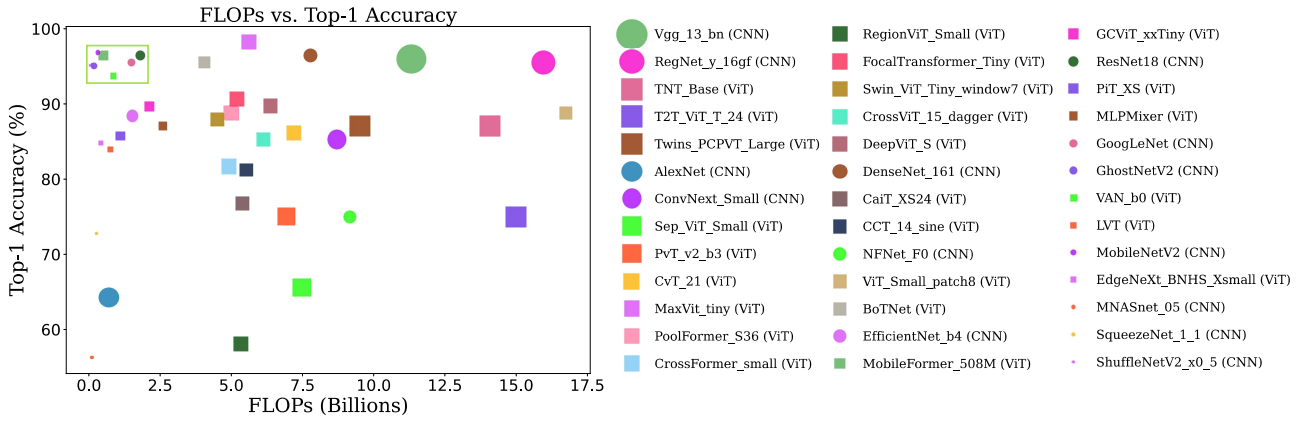


Figure 35. FLOPs vs. top-1 accuracy.

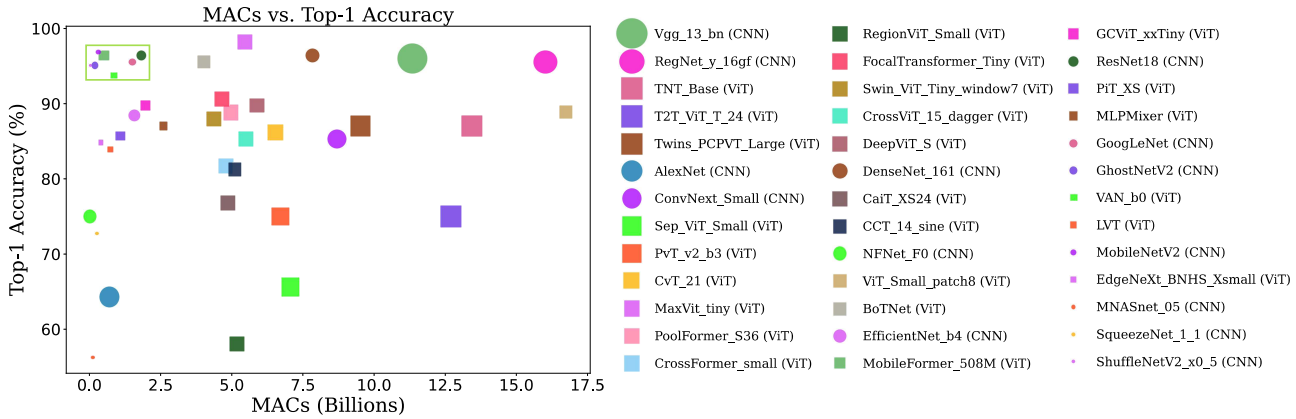


Figure 36. MACs vs. top-1 accuracy.

The analysis of F1 score versus epoch is presented in Figure 37. MaxVit-tiny achieves an impressive F1 score of 0.98 at epoch 389, indicating a robust performance in precision and recall metrics. This model likely benefited from extensive training, allowing it to consistently improve its performance across multiple evaluation metrics. MobileNetV2, with a slightly lower but still strong F1 score of 0.97 at epoch 52, demonstrates a rapid convergence and an effective utilization of training epochs.

In summary, for weather classification tasks with the chosen dataset, MaxVit-tiny excels in achieving a higher accuracy and F1 score metrics, making it suitable for applications demanding high precision and recall. Conversely, MobileNetV2 offers efficient training and competitive performance metrics early in the training process, making it suitable for applications requiring rapid deployment and operation under resource constraints.

Overall, the analysis for all the datasets identifies the unique strengths of each architecture and underscores the importance of aligning the choice of model with a specific task. The study emphasizes the need for a wise selection process, ensuring that the chosen architecture optimally balances accuracy and computational efficiency based on the demands of the particular application in various domains. As the field continues to evolve, further research could explore efficient hybrid approaches that leverage the strengths of both architectures, potentially leading to even more nuanced and efficient solutions for image classification.

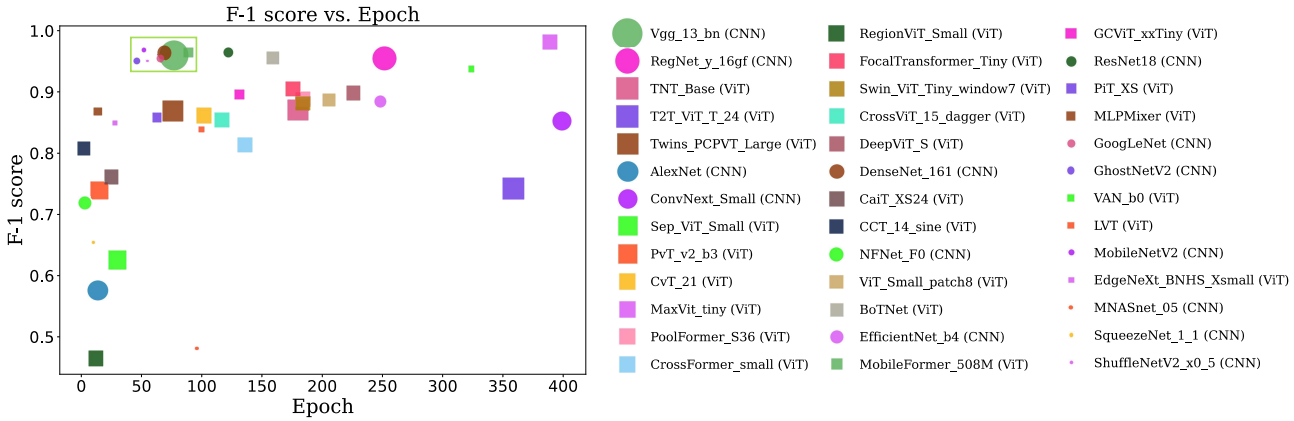


Figure 37. F1 score vs. epoch.

6. Conclusions

In this paper, we established ConVision Benchmark, a robust framework to address common challenges in deep-learning research, such as library version mismatches, difficulties reproducing results, and a lack of unified validation and hardware performance metrics. The ConVision Benchmark is implemented in PyTorch, integrating state-of-the-art CNN and ViT models to standardize the implementation and evaluation process across various datasets. It includes tools for assessing performance metrics and computational efficiency indicators. We presented a detailed comparative analysis to show the effectiveness of the framework and demonstrated the differences between CNNs and ViTs using three different datasets for image classification.

The analysis indicates that CNNs demonstrate a high accuracy and computational efficiency, making them suitable for real-time applications and resource-constrained environments. For example, DenseNet achieves a higher accuracy with a robust F1 score, but its computational complexity requires careful consideration for deployment. In contrast, GoogLeNet and MobileNet excel with minimal parameters and latency, striking an excellent balance between performance and efficiency. ViT models, including MaxViT and MobileFormer, can capture global context and learn complex patterns, making them advantageous for tasks requiring comprehensive understanding. However, these models often demand more computational resources and exhibit higher latency. MaxViT, despite its high accuracy, involves a trade-off with its increased computational cost. MobileFormer, though computationally efficient, achieves competitive accuracy with one of the lowest training times per epoch, highlighting its potential for real-time processing. The choice between CNN and ViT models hinges on specific task requirements, computational resources, and the trade-off between accuracy and efficiency. Hybrid models that combine the strengths of both architectures present a promising future direction, potentially revolutionizing image classification by enhancing accuracy and efficiency.

These findings underscore the versatility and applicability of our framework across different datasets and highlight the importance of selecting appropriate model architectures based on specific task requirements. This research serves as a foundational exploration, paving the way for further investigations into optimized image classification architectures as DL advances. The findings provide valuable insights for guiding the selection of appropriate models for diverse image classification applications in the medical field and other domains.

Author Contributions: Conceptualization, K.T.C.-V.; methodology, K.T.C.-V.; software, K.T.C.-V., S.B.V. and K.A.; validation, K.T.C.-V. and S.B.V.; formal analysis, K.T.C.-V.; investigation, K.T.C.-V., S.B.V. and K.A.; resources, A.K.S.; data curation, K.T.C.-V., S.B.V. and K.A.; writing—original draft preparation, S.B.V.; writing—review and editing, K.T.C.-V. and A.K.S.; visualization, K.T.C.-V. and S.B.V.; supervision, A.K.S.; project administration, A.K.S.; funding acquisition, A.K.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received support from the Philip and Virginia Sproul Professorship at Iowa State University.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original data presented in the study are openly available at https://github.com/krishnateja95/COVID19_Benchmarking accessed on 9 June 2024.

Acknowledgments: The authors would like to thank Yiming Bian for constructive criticism of the research work and Himani Kohli for the thorough review of this paper.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

Appendix A.1. Performance Results of CNN and ViT Models for COVID-19 Detection

In Appendix A.1, we present a detailed performance comparison of various CNN and ViT models on the COVID-19 detection task. The results are summarized in Table A1, highlighting key performance metrics such as the best top-1 accuracy, the epoch at which this accuracy was achieved, the best recall, the epoch at which the best recall was achieved, the best loss, and the epoch corresponding to the best loss. These metrics provide insights into the effectiveness and convergence characteristics of different models, allowing for a thorough evaluation of their performance in detecting COVID-19 from medical images.

Table A1. Performance results of CNN and ViT models for COVID-19 detection.

Model	Top-1 Accuracy (%)	Top-1 Accuracy Epoch	Recall (%)	Recall Epoch	Best Loss	Best Loss Epoch
DenseNet-161	95.61	410	0.96	410	0.21	226
Vgg-13-bn	95.27	230	0.95	230	0.16	39
DenseNet-121	95.26	401	0.95	401	0.23	204
DenseNet-169	95.24	327	0.95	327	0.2	12
Vgg-19-bn	95.08	187	0.95	187	0.18	24
MaxVit-tiny	95.02	296	0.95	296	0.17	26
Vgg-11-bn	95.01	293	0.95	293	0.17	27
Vgg-16-bn	94.99	131	0.95	131	0.2	28
GoogLeNet	94.95	136	0.95	136	0.25	16
DenseNet-201	94.9	383	0.95	383	0.24	11
Vgg-16	94.27	130	0.94	130	0.21	20
EfficientNet-b4	94.25	290	0.94	290	0.19	30
MobileFormer-508M	94.15	129	0.94	129	0.25	24
MobileFormer-294M	94.11	125	0.94	125	0.26	9
Vgg-11	94.08	154	0.94	154	0.22	12
EfficientNet-b0	94.03	253	0.94	253	0.2	26
Vgg-13	94	90	0.94	90	0.22	11
MobileNetV2	93.97	442	0.94	442	0.26	17
Vgg-19	93.89	176	0.94	176	0.2	17
EfficientNet-b1	93.74	371	0.94	371	0.22	31
EfficientNet-b3	93.64	154	0.94	154	0.21	31
MobileFormer-151M	93.62	134	0.94	134	0.23	16
AlexNet	93.62	267	0.94	267	0.22	19
MobileFormer-214M	93.58	444	0.94	444	0.25	22
EfficientNet-v2-s	93.41	361	0.93	361	0.22	28
CCT-14-sine	93.3	120	0.93	120	0.24	33

Table A1. *Cont.*

Model	Top-1 Accuracy (%)	Top-1 Accuracy Epoch	Recall (%)	Recall Epoch	Best Loss	Best Loss Epoch
MobileNetV1	93.27	154	0.93	154	0.3	460
MobileFormer-96M	93.21	89	0.93	89	0.3	26
CCT-7-sine	93.12	170	0.93	170	0.27	30
ResNet18	93.05	170	0.93	170	0.29	443
EfficientNet-b2	93	257	0.93	257	0.27	26
RegNet-y-16gf	93	45	0.93	45	0.31	11
CCT-14	92.9	206	0.93	206	0.24	26
EfficientNet-b5	92.87	382	0.93	382	0.26	38
ShuffleNetV2-x0-5	92.86	278	0.93	278	0.23	25
GhostNetV2	92.81	160	0.93	160	0.29	9
ResNet34	92.81	271	0.93	271	0.3	8
ShuffleNetV2-x2-0	92.78	395	0.93	395	0.31	12
RegNet-y-800mf	92.77	58	0.93	58	0.31	16
RegNet-y-1-6gf	92.75	61	0.93	61	0.25	14
ShuffleNetV2-x1-5	92.75	236	0.93	236	0.31	5
RegNet-y-8gf	92.72	41	0.93	41	0.29	15
RegNet-y-3-2gf	92.72	68	0.93	68	0.31	11
ShuffleNetV2-x1-0	92.6	488	0.93	447	0.28	15
MobileFormer-26M	92.58	95	0.93	95	0.26	18
ResNext50	92.53	58	0.93	58	0.28	10
MobileNet-V3-large	92.47	422	0.92	422	0.36	61
PvT-v2-b3	92.43	56	0.92	56	0.24	19
RegNet-x-8gf	92.43	218	0.92	218	0.32	5
RegNet-x-1-6gf	92.38	254	0.92	254	0.33	467
RegNet-y-32gf	92.34	64	0.92	64	0.3	13
MobileFormer-52M	92.34	135	0.92	135	0.26	16
RegNet-y-400mf	92.32	110	0.92	110	0.3	23
RegNet-x-16gf	92.24	434	0.92	434	0.3	8
ResNet50	92.15	233	0.92	233	0.31	19
MNASnet-05	92.15	499	0.92	499	0.39	499
RegNet-x-400mf	92.13	437	0.92	437	0.33	22
RegNet-x-32gf	92.09	488	0.92	488	0.32	452
RegNet-x-3-2gf	92.02	107	0.92	92	0.32	15
FocalTransformer-Tiny	91.99	392	0.92	392	0.28	32
MNASnet-13	91.99	499	0.92	499	0.45	499
PvT-v2-b2	91.97	121	0.92	121	0.25	16
CCT-7	91.96	78	0.92	78	0.27	28
Swin-ViT-Tiny-window7	91.84	378	0.92	378	0.28	24
MNASnet-075	91.82	499	0.92	499	0.46	499
RegNet-x-800mf	91.81	401	0.92	401	0.33	14
FocalTransformer-Small	91.76	269	0.92	269	0.29	35
EfficientNet-v2-m	91.74	450	0.92	450	0.31	38
wide-ResNet50	91.74	286	0.92	286	0.27	13
mobilenet-v3-small	91.74	469	0.92	469	0.39	34
PvT-v2-b4	91.63	80	0.92	80	0.26	46
GCViT-xxTiny	91.62	349	0.92	349	0.26	29
Swin-ViT-Base	91.6	340	0.92	340	0.27	31
MNASnet-10	91.56	499	0.92	499	0.42	499
Swin-ViT-Small-window7	91.56	369	0.92	369	0.28	33
PvT-v2-b5	91.48	46	0.91	46	0.25	19
ResNet101	91.44	481	0.91	481	0.32	14
DeepViT-S	91.44	340	0.91	340	0.27	32

Table A1. *Cont.*

Model	Top-1 Accuracy (%)	Top-1 Accuracy Epoch	Recall (%)	Recall Epoch	Best Loss	Best Loss Epoch
Swin-ViT-Large-window7	91.37	402	0.91	402	0.29	33
Swin-ViT-Small	91.34	437	0.91	437	0.28	31
PvT-v2-b2-Linear	91.32	54	0.91	54	0.29	26
VAN-b0	91.32	257	0.91	257	0.28	238
PvT-v2-b1	91.32	35	0.91	35	0.25	22
GCViT-xTiny	91.28	169	0.91	169	0.26	33
ResNext101	91.25	358	0.91	358	0.33	6
Swin-ViT-Base-window7	91.15	407	0.91	407	0.29	34
T2T-ViT-T-24	91.12	130	0.91	130	0.3	32
Swin-ViT-Tiny	91.07	389	0.91	389	0.26	29
T2T-ViT-19	90.95	169	0.91	169	0.3	32
GCViT-Tiny	90.93	111	0.91	111	0.27	24
wide-ResNet101	90.9	456	0.91	456	0.4	5
GCViT-Tiny2	90.87	81	0.91	81	0.25	31
ResNet152	90.81	108	0.91	108	0.37	17
T2T-ViT-14-wide	90.79	121	0.91	114	0.32	15
CrossFormer-small	90.72	232	0.91	232	0.29	24
T2T-ViT-14	90.69	133	0.91	133	0.29	31
LVT	90.65	60	0.91	60	0.28	28
NFNet-F0	90.65	20	0.91	20	0.27	20
CrossFormer-base	90.63	332	0.91	332	0.28	40
PvT-v2-b0	90.62	71	0.91	71	0.28	24
CrossFormer-large	90.6	192	0.91	192	0.27	30
DeepViT-L	90.54	214	0.91	214	0.3	87
CrossFormer-tiny	90.54	419	0.91	419	0.3	31
PvT-Large	90.48	150	0.9	150	0.31	32
T2T-ViT-10	90.34	245	0.9	245	0.3	32
RegionViT-Small	90.32	135	0.9	135	0.29	15
T2T-ViT-14-resnext	90.26	236	0.9	236	0.29	26
VAN-b1	90.17	64	0.9	64	0.31	11
Twins-PCPVT-Large	90.11	58	0.9	58	0.28	21
Twins-SVT-Base	90.04	52	0.9	52	0.31	18
Sep-ViT-Small	90.03	71	0.9	71	0.31	25
VAN-b2	89.97	43	0.9	43	0.33	14
T2T-ViT-24	89.95	221	0.9	210	0.32	28
Sep-ViT-Base	89.85	104	0.9	104	0.28	22
Twins-PCPVT-Base	89.81	57	0.9	57	0.31	18
T2T-ViT-7	89.81	191	0.9	191	0.29	39
PvT-Tiny	89.79	153	0.9	153	0.32	36
RegionViT-Base	89.7	140	0.9	140	0.3	25
Twins-SVT-Large	89.7	63	0.9	63	0.3	13
Sep-ViT-Tiny	89.7	56	0.9	56	0.29	23
PvT-Small	89.67	87	0.9	87	0.32	32
RegionViT-Medium	89.64	179	0.9	179	0.3	17
T2T-ViT-T-19	89.58	104	0.9	104	0.32	24
T2T-ViT-12	89.54	229	0.9	229	0.29	33
PiT-XS	89.5	97	0.89	97	0.33	21
RegionViT-Tiny	89.47	157	0.89	157	0.32	19
TNT-Base	89.41	215	0.89	215	0.32	38
Twins-SVT-Small	89.39	57	0.89	57	0.31	21
T2T-ViT-T-14	89.39	134	0.89	134	0.31	31
Twins-PCPVT-Small	89.3	49	0.89	49	0.29	18

Table A1. *Cont.*

Model	Top-1 Accuracy (%)	Top-1 Accuracy Epoch	Recall (%)	Recall Epoch	Best Loss	Best Loss Epoch
CrossViT-15-dagger	89.25	236	0.89	236	0.32	31
PiT-Small	89.11	103	0.89	103	0.34	18
CrossViT-9-dagger	89.04	135	0.89	135	0.33	34
CvT-21	89.04	286	0.89	286	0.37	24
CrossViT-Small	88.85	254	0.89	254	0.32	41
CrossViT-15	88.77	207	0.89	207	0.33	36
PiT-TI	88.74	120	0.89	120	0.35	16
CrossViT-Base	88.74	179	0.89	179	0.34	33
EdgeNeXt-BNHS-Xsmall	88.74	54	0.89	54	0.3	38
TNT-Small	88.72	171	0.89	171	0.31	45
ConvNext-Small	88.67	59	0.89	59	0.32	59
CrossViT-18	88.57	265	0.89	265	0.33	41
ViT-Small-patch8	88.41	43	0.88	43	0.32	43
Sep-ViT-Lite	88.35	27	0.88	27	0.32	27
ViT-Small-patch16	88.29	204	0.88	204	0.34	37
EdgeNeXt-BNHS-Small	88.27	253	0.88	253	0.35	42
CvT-13	88.21	68	0.88	68	0.35	27
CaiT-XS24	88.16	108	0.88	108	0.33	80
CrossViT-Tiny	87.99	218	0.88	218	0.32	46
CaiT-XS36	87.95	84	0.88	84	0.33	72
CrossViT-9	87.82	200	0.88	200	0.33	41
ViT-Tiny-patch16	87.67	182	0.88	182	0.36	39
CaiT-XXS24	87.51	102	0.88	102	0.35	88
ConvNext-Tiny	87.48	68	0.87	68	0.35	68
CaiT-S24	87.46	74	0.87	74	0.34	59
MLPMixer	87.43	60	0.87	60	0.37	6
ViT-Base-patch16	87.29	186	0.87	186	0.36	34
ConvNext-Base	87.24	47	0.87	47	0.35	47
ResMLP	87.11	38	0.87	38	0.36	31
EdgeNeXt-Small	87.09	69	0.87	69	0.35	54
CaiT-XXS36	87.05	74	0.87	74	0.35	74
EdgeNeXt-Base	86.93	54	0.87	54	0.36	46
ViT-Large-patch32	86.73	167	0.87	167	0.4	33
EdgeNeXt-Xsmall	86.65	57	0.87	57	0.37	42
ViT-Base-patch32	86.34	155	0.86	154	0.41	36
ViT-Small-patch32	86.06	129	0.86	129	0.43	25
EdgeNeXt-BNHS-Xxsmall	85.92	58	0.86	58	0.4	38
GCViT-Small	85.75	65	0.86	65	0.39	32
PoolFormer-S36	85.22	484	0.85	484	0.39	442
SqueezeNet-1-1	85.11	463	0.63	445	0.59	437
PoolFormer-S12	83.91	37	0.84	37	0.42	37
NFNet-F1	83.44	10	0.83	10	0.44	10
BoTNet	83.38	294	0.83	294	0.45	151
PiT-Base	81.51	20	0.82	20	0.47	20
PoolFormer-M36	81.38	20	0.81	20	0.48	14
PoolFormer-S24	80.38	38	0.8	38	0.49	33
PvT-Medium	75.84	4	0.76	4	0.6	4
SqueezeNet-1-0	45.54	2	0.46	9	1.02	1
Inception-Resnet-v2	34.69	18	0.35	18	1.06	16

Appendix A.2. Computation Efficiency of CNN and ViT Models for COVID-19 Detection

This section provides a comparative analysis of the computational efficiency of various CNN and ViT models used for COVID-19 detection. Table A2 summarizes critical metrics such as the number of parameters, multiply–accumulate operations (MACs), floating-point operations (FLOPs), training time per epoch, CPU latency, GPU latency, training memory, and inference memory. These metrics are essential for understanding the computational demands and efficiency of different models, which are crucial factors when deploying these models in real-world applications where resources may be limited.

Table A2. Computation efficiency of CNN and ViT models for COVID-19 detection.

Model	Number of Parameters (Million)	MACs (Billion)	FLOPs (Billion)	Training Time per Epoch (s)	CPU Latency (ms)	GPU Latency (ms)	Training Memory (GB)	Inference Memory (MB)
ShuffleNetV2-x0-5	0.34	0.04	0.04	17.4	5.87	4.36	0.02	1.46
SqueezeNet-1-1	0.72	0.26	0.26	17.36	5.17	1.5	0.03	2.78
SqueezeNet-1-0	0.74	0.73	0.73	17.16	7.43	1.55	0.04	2.83
MNASnet-05	0.94	0.12	0.11	16.97	6.62	3.16	0.04	3.78
EdgeNeXt-BNHS-Xxsmall	1.16	0.2	0.2	17.81	7.27	4.1	0.04	4.52
ShuffleNetV2-x1-0	1.26	0.15	0.15	18.21	8.59	4.46	0.04	4.95
mobilenet-v3-small	1.52	0.06	0.06	17.22	5.03	3.43	0.04	5.99
MNASnet-075	1.89	0.23	0.22	17.2	8.17	3.3	0.07	7.43
EdgeNeXt-Xsmall	2.14	0.4	0.41	18.29	12.06	5.5	0.07	8.31
EdgeNeXt-BNHS-Xsmall	2.14	0.41	0.41	18.09	10.66	5.21	0.07	8.32
MobileFormer-26M	2.21	0.03	0.03	17.44	12.1	12.18	0.05	8.72
MobileFormer-52M	2.21	0.03	0.03	17.3	12.51	12.23	0.05	8.72
MobileNetV2	2.23	0.33	0.31	16.88	9.13	3.36	0.11	8.76
ShuffleNetV2-x1-5	2.48	0.31	0.3	17.19	9.62	4.25	0.07	9.66
MNASnet-10	3.11	0.33	0.32	17.5	9.71	3.46	0.09	12.08
MobileNetV1	3.21	0.59	0.58	16.7	7.17	1.71	0.09	12.35
MobileFormer-96M	3.31	0.1	0.1	17.61	14.24	12.3	0.08	12.88
Sep-ViT-Lite	3.4	0.49	0.57	18.94	12.79	6.75	0.08	13.08
PvT-v2-b0	3.41	0.53	0.57	25.2	13.22	4.72	0.1	13.09
LVT	3.42	0.73	0.76	27.65	17.16	4.68	0.1	13.12
VAN-b0	3.85	0.87	0.87	26.11	20.39	6.84	0.14	14.91
RegNet-y-400mf	3.9	0.42	0.41	17.43	11.59	6.14	0.09	15.21
T2T-ViT-7	4	0.98	1.16	21.85	11.11	3.95	0.1	15.51
EfficientNet-b0	4.01	0.41	0.4	20.93	12.67	5.37	0.14	15.7
mobilenet-v3-large	4.2	0.23	0.22	17.19	8.98	4.14	0.1	16.31
CCT-7-sine	4.5	1.47	1.61	17.46	9.7	2.52	0.09	17.38
CCT-7	4.5	1.47	1.61	17.43	9.71	2.55	0.09	17.38
PiT-TI	4.54	0.5	0.5	16.95	8.98	3.41	0.09	17.66
GhostNetV2	4.88	0.18	0.18	23.17	21.16	9.38	0.13	19.14
MNASnet-13	5.01	0.55	0.54	19.98	12.04	3.31	0.13	19.36
RegNet-x-400mf	5.1	0.43	0.42	17.47	11.02	5.39	0.1	19.85
EdgeNeXt-Small	5.27	0.96	0.97	23.73	16.06	5.58	0.14	20.29
EdgeNeXt-BNHS-Small	5.28	0.96	0.97	20.22	13.87	5.22	0.13	20.3
ShuffleNetV2-x2-0	5.35	0.6	0.59	17.32	12.92	4.34	0.12	20.62
ViT-Tiny-patch16	5.49	1.08	1.08	17.36	10.78	3.18	0.11	21.22
T2T-ViT-10	5.58	1.29	1.53	26.47	14.23	5.01	0.13	21.55
GoogLeNet	5.6	1.51	1.5	17.23	18.13	4.57	0.2	38.12
RegNet-y-800mf	5.65	0.86	0.85	18.23	13.3	5.49	0.13	21.87
MobileFormer-151M	6.32	0.15	0.15	18.74	19.1	15.93	0.14	24.49
EfficientNet-b1	6.52	0.61	0.59	29.56	19.09	7.48	0.21	25.43
RegNet-x-800mf	6.59	0.82	0.81	17.21	12.13	3.92	0.14	25.45
T2T-ViT-12	6.63	1.5	1.77	29.52	16.16	5.68	0.15	25.58
CrossViT-Tiny	6.65	1.3	1.57	26.52	17.51	7.96	0.15	25.84
DenseNet-121	6.96	2.9	2.86	29.16	34.84	9.91	0.23	27.03
EfficientNet-b2	7.71	0.7	0.68	30.98	19.95	7.5	0.24	30.01
MobileFormer-214M	7.83	0.21	0.21	20.64	20.91	16.17	0.17	30.29

Table A2. *Cont.*

Model	Number of Parameters (Million)	MACs (Billion)	FLOPs (Billion)	Training Time per Epoch (s)	CPU Latency (ms)	GPU Latency (ms)	Training Memory (GB)	Inference Memory (MB)
CrossViT-9	8.07	1.54	1.85	28.48	16.75	7.16	0.17	31.28
RegNet-x-1-6gf	8.28	1.63	1.62	19.06	17.01	4.7	0.19	31.98
CrossViT-9-dagger	8.29	1.68	1.99	29.22	17.61	7.43	0.18	32.14
MobileFormer-294M	9.51	0.29	0.29	23.48	21.14	15.82	0.21	36.69
PiT-XS	10.16	1.1	1.1	18.32	12.31	3.5	0.18	39.2
RegNet-y-1-6gf	10.32	1.65	1.63	21.78	24.06	10.79	0.22	39.87
EfficientNet-b3	10.69	0.97	0.95	36.41	24.44	8.72	0.31	41.52
ResNet18	11.18	1.82	1.82	17.69	9.42	1.57	0.19	42.7
PoolFormer-S12	11.38	1.81	1.82	17.48	17.63	3.36	0.23	44.42
GCViT-xxTiny	11.44	1.96	2.14	47.39	27.75	9.31	0.28	46.19
CaiT-XXS24	11.72	2.18	2.53	54.46	37.67	11.8	0.28	45.11
MobileFormer-508M	12.06	0.5	0.51	29.69	26.1	15.85	0.28	46.47
PvT-Tiny	12.33	1.86	1.94	27.91	18.65	4.34	0.26	48.7
DenseNet-169	12.49	3.43	3.4	35.04	46.82	14.04	0.34	48.48
RegionViT-Tiny	13.31	2.31	2.43	52.56	31.93	11.85	0.3	50.98
VAN-b1	13.34	2.51	2.5	35.84	29.37	5.48	0.32	51.57
PvT-v2-b1	13.5	2.04	2.12	45.28	25.29	4.76	0.29	51.56
RegNet-x-3-2gf	14.29	3.22	3.2	24.6	27.22	6.37	0.3	55.01
ResMLP	14.94	3.01	3.01	32	14.76	2.93	0.27	58.06
CaiT-XXS36	17.06	3.24	3.77	80.26	57.31	17.66	0.42	65.6
EfficientNet-b4	17.55	1.58	1.54	49.72	32.27	10.61	0.48	68.25
EdgeNeXt-Base	17.91	2.92	2.95	45.6	26.18	5.54	0.38	69.99
RegNet-y-3-2gf	17.93	3.22	3.2	28.75	30.39	8.31	0.37	69.61
DenseNet-201	18.1	4.39	4.34	43.93	61.95	17.11	0.46	70.19
BoTNet	18.8	4.02	4.06	23.98	28.94	4.86	0.37	72.1
GCViT-xTiny	19.42	2.71	2.94	62.8	37.31	12.06	0.43	76.8
CvT-13	19.61	4.08	4.58	58.61	39.41	11.54	0.4	75.07
EfficientNet-v2-s	20.18	2.9	2.88	32.62	36.04	11.8	0.46	79.03
PoolFormer-S24	20.84	3.39	3.41	23.6	33.95	6.59	0.42	80.66
T2T-ViT-T-14	21.08	4.35	6.11	74.47	38.4	5.92	0.44	80.8
T2T-ViT-14	21.08	4.35	4.8	55.18	28.44	6.35	0.4	80.81
T2T-ViT-14-resnext	21.08	4.35	4.8	87.15	30.72	6.4	0.46	80.81
ResNet34	21.29	3.68	3.67	17.28	17.02	2.62	0.35	82.18
ViT-Small-patch8	21.37	16.76	16.76	184.67	96.67	4.8	0.55	83.08
ViT-Small-patch16	21.59	4.25	4.25	41.85	23.29	3.11	0.38	83.09
NFNet-F0	21.86	0.02	9.18	38.27	105.73	10.76	1.69	263.07
CCT-14-sine	21.91	5.12	5.53	50.1	27.48	4.8	0.4	84.13
CCT-14	21.91	5.12	5.53	50.08	27.86	4.89	0.4	84.13
PvT-v2-b2-Linear	22.04	3.76	3.91	87.54	44.64	10.11	0.49	85.04
ViT-Small-patch32	22.48	1.12	1.12	17.68	10.95	3.16	0.35	85.94
PiT-Small	22.78	2.42	2.42	31.43	21.77	3.65	0.39	87.48
ResNext50	22.99	4.29	4.26	28.87	32.07	4.12	0.45	88.61
TNT-Small	23.3	4.85	5.24	118.45	43.93	10.97	0.49	91.95
ResNet50	23.51	4.13	4.11	22.31	27.27	3.95	0.43	89.95
Twins-SVT-Small	23.55	2.82	2.82	42.83	31.37	8.28	0.44	90.98
PvT-Small	23.58	3.69	3.83	49.5	33.56	8.27	0.48	91.69
Twins-PCPVT-Small	23.59	3.68	3.68	47.79	36.44	7.33	0.47	90.47
T2T-ViT-14-wide	24.23	4.97	5.24	52.51	23.82	3.17	0.42	93.39
PvT-v2-b2	24.85	3.9	4.05	80.47	43.29	9.24	0.53	94.93
VAN-b2	26.06	5.01	5	64.12	57.48	10.47	0.6	100.61
CrossViT-Small	26.13	5.08	5.63	65	33.22	7.91	0.5	100.72
CaiT-XS24	26.2	4.87	5.4	84.85	59.42	11.93	0.56	100.48
DenseNet-161	26.48	7.84	7.78	52.73	73.24	14.4	0.63	103.1
CrossViT-15-dagger	27.48	5.49	6.13	67.44	37.02	8.9	0.53	106.18
Swin-ViT-Tiny-window7	27.5	4.37	4.51	62.9	30.61	6.26	0.52	106.45
Swin-ViT-Base-window7	27.5	4.37	4.51	62.94	31.31	6.45	0.52	106.45

Table A2. *Cont.*

Model	Number of Parameters (Million)	MACs (Billion)	FLOPs (Billion)	Training Time per Epoch (s)	CPU Latency (ms)	GPU Latency (ms)	Training Memory (GB)	Inference Memory (MB)
Swin-ViT-Tiny	27.5	4.38	4.64	65.37	32.72	5.9	0.54	108.14
GCViT-Tiny	27.58	4.32	4.79	90.34	59.13	19.18	0.62	112.12
ConvNext-Tiny	27.81	4.46	4.47	60.66	25.91	3.65	0.52	107.11
EfficientNet-b5	28.35	2.46	2.41	67.67	45	12.76	0.72	110.67
FocalTransformer-Tiny	29.44	4.66	5.22	146.14	65.75	17.82	0.62	116
RegionViT-Small	29.79	5.19	5.35	84.44	44.81	11.9	0.59	114.75
CrossFormer-small	29.89	4.79	4.92	63.15	37.44	9.12	0.56	114.42
PoolFormer-S36	30.29	4.97	5	33.99	50.58	9.5	0.62	116.89
MaxVit-tiny	30.38	5.46	5.61	91.16	64.36	17.1	0.75	118.93
Sep-ViT-Tiny	30.4	4.28	4.53	60.46	32.76	6.82	0.56	116.46
CvT-21	31.24	6.54	7.21	89.71	60.54	18.49	0.63	119.56
GCViT-Tiny2	33.84	5.56	6.21	110.91	67.88	23.66	0.76	139.19
RegNet-y-8gf	37.17	8.04	8	41.54	49.33	6.97	0.71	143.66
RegNet-x-8gf	37.66	8.05	8.02	33.09	46.26	6.53	0.67	144.42
CaiT-XS36	38.19	7.25	8.05	125.61	87.81	16.97	0.81	146.36
T2T-ViT-T-19	38.64	7.8	9.81	111.49	54.7	7.3	0.75	147.87
T2T-ViT-19	38.64	7.8	8.5	91.81	43.65	8.18	0.71	147.88
RegionViT-Medium	40.41	7.22	7.43	106.99	55.75	16.46	0.76	155.33
CrossViT-18	42.42	8.21	9.05	99.93	48.94	9.79	0.78	163.05
ResNet101	42.51	7.86	7.83	33.93	49.66	7.91	0.75	162.89
PvT-Medium	43.31	6.46	6.69	76.58	53.59	14.11	0.82	167.01
Twins-PCPVT-Base	43.32	6.46	6.46	72.79	57.61	12.48	0.8	165.82
NFNet-F1	43.7	0.04	16.92	66.52	189.01	20.18	3.17	500.84
PvT-v2-b3	44.73	6.7	6.92	116.41	68.55	15.47	0.88	170.85
Sep-ViT-Small	45.78	7.07	7.48	93.93	50.93	11.43	0.84	176.7
CaiT-S24	46.44	8.63	9.35	118.42	80.53	12.09	0.91	177.99
Swin-ViT-Small-window7	48.79	8.54	8.77	107.99	56.78	12.5	0.9	188.27
Swin-ViT-Large-window7	48.79	8.55	8.77	125.08	58.6	12.65	0.92	188.46
Swin-ViT-Small	48.79	8.58	9.43	121.63	64.03	10.84	0.98	196.6
ConvNext-Small	49.44	8.7	8.7	108.96	45.36	6.8	0.91	189.73
GCViT-Small	50.11	7.87	8.57	135.43	73.2	18.98	1.05	199.24
FocalTransformer-Small	50.74	8.87	9.75	244.35	115.78	35.54	1.07	201.32
CrossFormer-base	51.2	8.96	9.19	107.93	63.41	17.06	0.94	196.1
RegNet-x-16gf	52.24	16.04	15.99	53.72	74.41	6.54	0.98	202.09
EfficientNet-v2-m	52.86	5.45	5.41	53.7	65.57	16.43	1.05	207.29
Inception-Resnet-v2	54.31	6.5	6.48	39	82.18	19.93	0.91	210.51
Twins-SVT-Base	55.3	8.36	8.36	92.75	58.72	11.19	0.99	214.47
PoolFormer-M36	55.32	8.76	8.8	48	75.19	9.47	1.07	211.63
AlexNet	57.02	0.71	0.71	17.03	8.95	0.51	0.85	217.5
ResNet152	58.15	11.6	11.56	46.75	72.23	12.03	1.04	222.54
DeepViT-L	58.38	12.16	13.19	184.18	82.44	13.83	1.12	223.34
PvT-Large	60.47	9.53	9.85	109	79.53	20.33	1.14	232.44
Twins-PCPVT-Large	60.48	9.52	9.53	102.84	82.13	18.06	1.12	231.77
PvT-v2-b4	62.04	9.82	10.14	165.58	97.98	23.3	1.22	237.76
T2T-ViT-T-24	63.49	12.7	15	154.36	71.54	9.34	1.17	243.59
T2T-ViT-24	63.49	12.7	13.69	134.95	63.92	10.35	1.13	243.6
TNT-Base	64.64	13.44	14.09	196.07	80.53	11.11	1.19	247.23
wide-ResNet50	66.84	11.45	11.42	33.16	60.34	4.02	1.11	257.28
Swin-ViT-Base	70.09	12.75	13.69	164.92	89.29	15.83	1.36	278.57
RegionViT-Base	71.67	12.79	13.07	161.49	85.48	16.52	1.3	276.61
PiT-Base	72.5	10.55	10.56	105.05	59.57	3.98	1.22	279.65
RegNet-y-16gf	80.57	16.01	15.96	58.87	83.78	8.36	1.42	316.53
Sep-ViT-Base	81.33	12.54	13.08	144.88	73.92	11.35	1.42	312.46
PvT-v2-b5	81.44	11.38	11.76	170.68	109.8	29.59	1.5	311.92
ViT-Base-patch16	85.65	16.86	16.87	136.99	65.71	4.78	1.39	327.43
ResNext101	86.75	16.54	16.47	69.47	95.97	7.93	1.55	336.15

Table A2. *Cont.*

Model	Number of Parameters (Million)	MACs (Billion)	FLOPs (Billion)	Training Time per Epoch (s)	CPU Latency (ms)	GPU Latency (ms)	Training Memory (GB)	Inference Memory (MB)
ViT-Base-patch32	87.42	4.37	4.37	35.79	27.77	3.3	1.33	333.75
ConvNext-Base	87.55	15.37	15.38	171.72	73.34	6.8	1.53	334.24
CrossFormer-large	90.95	15.85	16.15	168.25	90.3	16.84	1.61	351.68
Twins-SVT-Large	98.25	14.83	14.84	146.98	85.83	11.1	1.69	379.97
CrossViT-Base	103.57	20.13	21.22	188.87	90.13	7.96	1.79	407.83
RegNet-x-32gf	105.3	31.88	31.81	100.28	132.68	6.82	1.85	402.73
wide-ResNet101	124.84	22.84	22.79	55.16	107.95	8.07	2.05	484.85
Vgg-11	128.78	7.61	7.61	20.46	40.37	0.95	1.97	491.25
Vgg-11-bn	128.78	7.63	7.62	25.71	40.52	1.08	1.99	491.3
Vgg-13	128.96	11.3	11.3	30.26	49.86	1.08	1.99	491.96
Vgg-13-bn	128.97	11.35	11.33	39.61	50.98	1.26	2.03	492.01
Vgg-16	134.27	15.47	15.47	35.48	60.87	1.3	2.07	513.09
Vgg-16-bn	134.28	15.52	15.49	45.63	62.06	1.52	2.12	513.16
Vgg-19	139.58	19.63	19.63	41.03	73.43	1.52	2.16	532.47
Vgg-19-bn	139.59	19.69	19.66	51.7	73.04	1.71	2.21	532.56
RegNet-y-32gf	141.34	32.4	32.34	88.81	146.35	8.37	2.41	551.99
ViT-Large-patch32	305.46	15.26	15.27	114.15	90.16	6.2	4.63	1165.68

Appendix A.3. Performance and Computational Results of CNN and ViT Models for Flower Recognition

In this section, we provide a comprehensive performance and computation metrics comparison of different CNN and ViT models for the flower recognition task. The summarized results, found in Table A3, include crucial metrics such as the accuracy, F1 score, MACs, FLOPs, and training time per epoch.

Table A3. Performance results of CNN and ViT models for flower recognition.

Model	Number of Parameters (Million)	MACs (Billion)	FLOPs (Billion)	Training Time per Epoch (s)	Top-1 Accuracy (%)	Top-1 Accuracy Epoch	F1 Score	F1 Score Epoch
SqueezeNet-1-1	0.73	0.26	0.26	3.06	35.07	3	0.35	3
ShuffleNetV2-x0-5	0.35	0.04	0.04	3.03	66.78	194	0.75	194
MobileNetV2	2.23	0.33	0.31	3.24	69.91	78	0.79	78
MNASnet-05	0.94	0.12	0.11	3.28	36.11	273	0.43	273
EdgeNeXt-BNHS-Xsmall	2.14	0.41	0.41	3.23	49.19	111	0.58	224
DenseNet-161	26.48	7.84	7.78	7.21	73.73	274	0.82	274
ResNet18	11.18	1.82	1.82	3.17	69.33	144	0.78	144
GoogLeNet	5.61	1.51	1.5	3.2	78.01	274	0.85	274
MLPMixer	7.28	2.59	2.59	3.16	50.23	11	0.61	7
PiT-XS	10.17	1.1	1.1	3.2	64.12	47	0.74	87
GhostNetV2	4.88	0.18	0.18	3.69	68.17	131	0.78	131
VAN-b0	3.85	0.87	0.87	4	60.3	88	0.7	88
BoTNet	18.81	4.02	4.06	3.74	53.47	320	0.64	375
LVT	3.42	0.73	0.76	4.17	61.46	84	0.71	89
MobileFormer-508M	12.06	0.5	0.51	4.47	72.57	186	0.81	186
GCViT-xxTiny	11.44	1.96	2.14	6.5	64.12	199	0.74	199
EfficientNet-b4	17.56	1.58	1.54	6.82	66.32	301	0.76	301
PoolFormer-S36	30.29	4.97	5	4.97	44.79	64	0.55	53
NFNet-F0	21.86	0.02	9.18	5.46	41.9	1	0.48	2
CCT-14-sine	21.91	5.12	5.53	6.88	56.48	7	0.66	4
Swin-ViT-Tiny-window7	27.5	4.37	4.51	8.35	59.61	272	0.69	56

Table A3. *Cont.*

Model	Number of Parameters (Million)	MACs (Billion)	FLOPs (Billion)	Training Time per Epoch (s)	Top-1 Accuracy (%)	Top-1 Accuracy Epoch	F1 Score	F1 Score Epoch
CrossFormer-small	29.89	4.79	4.92	8.44	62.38	182	0.72	182
CrossViT-15-dagger	27.48	5.49	6.13	8.9	62.96	80	0.73	80
RegionViT-Small	29.79	5.19	5.35	10.7	37.15	5	0.41	5
DeepViT-S	26.84	5.9	6.38	11.29	65.97	415	0.75	335
CaiT-XS24	26.2	4.87	5.4	10.99	61	52	0.71	52
MaxVit-tiny	30.38	5.46	5.61	11.75	75.93	248	0.83	248
AlexNet	57.02	0.71	0.71	3	55.56	31	0.66	26
ViT-Small-patch8	21.37	16.76	16.76	22.8	63.19	127	0.73	127
FocalTransformer-Tiny	29.44	4.66	5.22	18.33	63.43	374	0.72	178
CvT-21	31.24	6.54	7.21	11.65	56.25	85	0.66	56
Sep-ViT-Small	45.78	7.07	7.48	12.17	56.02	75	0.66	75
RegNet-y-16gf	80.58	16.01	15.96	7.88	68.06	288	0.77	307
PvT-v2-b3	44.73	6.7	6.92	14.87	62.73	72	0.72	99
ConvNext-Small	49.44	8.7	8.7	13.87	51.27	156	0.62	156
Twins-PCPVT-Large	60.48	9.52	9.53	13.15	57.99	331	0.68	331
TNT-Base	64.64	13.44	14.09	24.24	62.04	93	0.71	93
T2T-ViT-T-24	63.49	12.7	15	19.38	51.39	350	0.62	350

Appendix A.4. Performance and Computational Results of CNN and ViT Models for Weather Classification

In this section, we provide a comprehensive performance and computation metrics comparison of different CNN and ViT models for the weather classification task. The summarized results, found in Table A4, include crucial metrics such as the accuracy, F1 score, MACs, FLOPs, and training time per epoch.

Table A4. Performance results of CNN and ViT models for weather classification.

Model	Number of Parameters (Million)	MACs (Billion)	FLOPs (Billion)	Training Time per Epoch (s)	Top-1 Accuracy (%)	Top-1 Accuracy Epoch	F1 Score	F1 Score Epoch
SqueezeNet-1-1	0.72	0.26	0.26	1.87	72.77	10	0.65	10
ShuffleNetV2-x0-5	0.35	0.04	0.04	1.89	95.09	55	0.95	55
MNASnet-05	0.94	0.12	0.11	2.03	56.25	96	0.48	96
EdgeNeXt-BNHS-Xsmall	2.14	0.41	0.41	2.03	84.82	28	0.85	28
MLPMixer	7.28	2.59	2.59	2.07	87.05	14	0.87	14
MobileNetV2	2.23	0.33	0.31	2.13	96.88	52	0.97	52
GoogLeNet	5.6	1.51	1.5	2.12	95.54	66	0.95	66
ResNet18	11.18	1.82	1.82	1.97	96.43	57	0.96	122
LVT	3.42	0.73	0.76	2.41	83.93	100	0.84	100
GhostNetV2	4.88	0.18	0.18	2.31	95.09	46	0.95	46
PiT-XS	10.17	1.1	1.1	2.25	85.71	63	0.86	63
VAN-b0	3.85	0.87	0.87	2.36	93.75	284	0.94	324
MobileFormer-508M	12.06	0.5	0.51	2.53	96.43	89	0.96	89
BoTNet	18.81	4.02	4.06	2.35	95.54	40	0.96	159
GCViT-xxTiny	11.44	1.96	2.14	2.96	89.73	131	0.9	131
EfficientNet-b4	17.56	1.58	1.54	3.01	88.39	248	0.88	248
CCT-14-sine	21.91	5.12	5.53	2.98	81.25	2	0.81	2
NFNet-F0	21.86	0.02	9.18	2.82	75	3	0.72	3
DenseNet-161	26.48	7.84	7.78	3.19	96.43	69	0.96	69
Swin-ViT-Tiny-window7	27.5	4.37	4.51	3.38	87.95	149	0.88	184
CrossViT-15-dagger	27.48	5.49	6.13	3.52	85.27	91	0.85	117
DeepViT-S	26.84	5.9	6.38	3.96	89.73	142	0.9	226

Table A4. *Cont.*

Model	Number of Parameters (Million)	MACs (Billion)	FLOPs (Billion)	Training Time per Epoch (s)	Top-1 Accuracy (%)	Top-1 Accuracy Epoch	F1 Score	F1 Score Epoch
CaiT-XS24	26.2	4.87	5.4	4.14	76.79	20	0.76	25
ViT-Small-patch8	21.37	16.76	16.76	6.46	88.84	206	0.89	206
PoolFormer-S36	30.29	4.97	5	2.63	88.84	184	0.89	184
CrossFormer-small	29.89	4.79	4.92	3.38	81.7	136	0.81	136
RegionViT-Small	29.79	5.19	5.35	4.15	58.04	5	0.46	12
AlexNet	57.02	0.71	0.71	1.87	64.29	14	0.58	14
MaxVit-tiny	30.38	5.46	5.61	4.21	98.21	338	0.98	389
CvT-21	31.24	6.54	7.21	4.52	86.16	102	0.86	102
FocalTransformer-Tiny	29.44	4.66	5.22	5.84	90.63	176	0.9	176
Sep-ViT-Small	45.78	7.07	7.48	4.38	65.63	2	0.63	30
PvT-v2-b3	44.73	6.7	6.92	5.43	75	16	0.74	15
ConvNext-Small	49.44	8.7	8.7	5.06	85.27	343	0.85	399
Twins-PCPVT-Large	60.48	9.52	9.53	4.73	87.05	76	0.87	76
RegNet-y-16gf	80.58	16.01	15.96	3.34	95.54	251	0.96	252
Vgg-13-bn	128.97	11.35	11.33	2.73	95.98	77	0.96	77
T2T-ViT-T-24	63.49	12.7	15	5.8	75	359	0.74	359
TNT-Base	64.64	13.44	14.09	7.05	87.05	180	0.87	180

References

1. Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An image is worth 16 × 16 words: Transformers for image recognition at scale. *arXiv* **2020**, arXiv:2010.11929.
2. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 84–90. [CrossRef]
3. NVIDIA. CUDA. 2006. Available online: https://developer.nvidia.com/cuda-zone (accessed on 20 February 2024).
4. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
5. Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; Darrell, T. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv* **2014**, arXiv:1408.5093.
6. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 1–12.
7. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
8. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
9. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 1–9.
10. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 2261–2269.
11. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
12. Tan, M.; Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In Proceedings of the International Conference on Machine Learning, PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 6105–6114.
13. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*; Curran Associates: Red Hook, NY, USA, 2017; Volume 30.
14. Yuan, L.; Chen, Y.; Wang, T.; Yu, W.; Shi, Y.; Jiang, Z.H.; Tay, F.E.; Feng, J.; Yan, S. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Montreal, BC, Canada, 11–17 October 2021; pp. 538–547.
15. Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Montreal, BC, Canada, 11–17 October 2021; pp. 9992–10002.
16. Chu, X.; Tian, Z.; Wang, Y.; Zhang, B.; Ren, H.; Wei, X.; Xia, H.; Shen, C. Twins: Revisiting the design of spatial attention in vision transformers. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 9355–9366.
17. Nafisah, S.I.; Muhammad, G.; Hossain, M.S.; AlQahtani, S.A. A Comparative Evaluation between Convolutional Neural Networks and Vision Transformers for COVID-19 Detection. *Mathematics* **2023**, *11*, 1489. [CrossRef]

18. Alayón, S.; Hernández, J.; Fumero, F.; Sigut, J.; Díaz-Alemán, T. Comparison of the Performance of Convolutional Neural Networks and Vision Transformer-Based Systems for Automated Glaucoma Detection with Eye Fundus Images. *Appl. Sci.* **2023**, *13*, 12722. [CrossRef]
19. Maurício, J.; Domingues, I.; Bernardino, J. Comparing Vision Transformers and Convolutional Neural Networks for Image Classification: A Literature Review. *Appl. Sci.* **2023**, *13*, 5521. [CrossRef]
20. Cuenat, S.; Couturier, R. Convolutional Neural Network (CNN) vs. Vision Transformer (ViT) for digital holography. In Proceedings of the 2022 2nd International Conference on Computer, Control and Robotics (ICCCR), Shanghai, China, 18–20 March 2022; pp. 235–240.
21. Koay, H.V.; Chuah, J.H.; Chow, C.O. Convolutional neural network or vision transformer? Benchmarking various machine learning models for distracted driver detection. In Proceedings of the TENCON 2021–2021 IEEE Region 10 Conference (TENCON), Auckland, New Zealand, 7–10 December 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 417–422.
22. Tahir, A.M.; Chowdhury, M.E.H.; Qiblawey, Y.; Khandakar, A.; Rahman, T.; Kiranyaz, S.; Khurshid, U.; Ibtehaz, N.; Mahmud, S.; Ezeddin, M. COVID-QU-Ex Dataset. 2022. Available online: https://www.kaggle.com/datasets/anasmohammedtahir/covidqu (accessed on 15 April 2023).
23. Liu, Z.; Mao, H.; Wu, C.Y.; Feichtenhofer, C.; Darrell, T.; Xie, S. A convnet for the 2020s. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 18–24 June 2022; pp. 11966–11976.
24. Han, K.; Wang, Y.; Tian, Q.; Guo, J.; Xu, C.; Xu, C. Ghostnet: More features from cheap operations. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 1577–1586.
25. Tang, Y.; Han, K.; Guo, J.; Xu, C.; Xu, C.; Wang, Y. GhostNetV2: Enhance Cheap Operation with Long-Range Attention. *arXiv* **2022**, arXiv:2211.12905.
26. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 2818–2826.
27. Szegedy, C.; Ioffe, S.; Vanhoucke, V.; Alemi, A. Inception-v4, inception-resnet and the impact of residual connections on learning. In Proceedings of the AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; Volume 31.
28. Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Sandler, M.; Howard, A.; Le, Q.V. Mnasnet: Platform-aware neural architecture search for mobile. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 2815–2823.
29. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520.
30. Howard, A.; Sandler, M.; Chu, G.; Chen, L.C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; et al. Searching for mobilenetv3. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Republic of Korea, 27 October–2 November 2019; pp. 1314–1324.
31. Brock, A.; De, S.; Smith, S.L.; Simonyan, K. High-performance large-scale image recognition without normalization. In Proceedings of the International Conference on Machine Learning, PMLR, Virtual, 18–24 July 2021; pp. 1059–1071.
32. Radosavovic, I.; Kosaraju, R.P.; Girshick, R.; He, K.; Dollár, P. Designing network design spaces. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 10425–10433.
33. Xie, S.; Girshick, R.; Dollár, P.; Tu, Z.; He, K. Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 5987–5995.
34. Zagoruyko, S.; Komodakis, N. Wide residual networks. *arXiv* **2016**, arXiv:1605.07146.
35. Ma, N.; Zhang, X.; Zheng, H.T.; Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 122–138.
36. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. *arXiv* **2016**, arXiv:1602.07360.
37. Srinivas, A.; Lin, T.Y.; Parmar, N.; Shlens, J.; Abbeel, P.; Vaswani, A. Bottleneck transformers for visual recognition. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 16514–16524.
38. Touvron, H.; Cord, M.; Sablayrolles, A.; Synnaeve, G.; Jégou, H. Going deeper with image transformers. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Montreal, QC, Canada, 10–17 October 2021; pp. 32–42.
39. Hassani, A.; Walton, S.; Shah, N.; Abuduweili, A.; Li, J.; Shi, H. Escaping the big data paradigm with compact transformers. *arXiv* **2021**, arXiv:2104.05704.
40. Wang, W.; Yao, L.; Chen, L.; Lin, B.; Cai, D.; He, X.; Liu, W. CrossFormer: A Versatile Vision Transformer Hinging on Cross-scale Attention. *arXiv* **2021**, arXiv:2108.00154.
41. Chen, C.F.; Fan, Q.; Panda, R. CrossViT: Cross-Attention Multi-Scale Vision Transformer for Image Classification. *arXiv* **2021**, arXiv:2103.14899.
42. Wu, H.; Xiao, B.; Codella, N.; Liu, M.; Dai, X.; Yuan, L.; Zhang, L. CvT: Introducing Convolutions to Vision Transformers. *arXiv* **2021**, arXiv:2103.15808.
43. Zhou, D.; Kang, B.; Jin, X.; Yang, L.; Lian, X.; Jiang, Z.; Hou, Q.; Feng, J. DeepViT: Towards Deeper Vision Transformer. *arXiv* **2021**, arXiv:2103.11886.

44. Maaz, M.; Shaker, A.; Cholakkal, H.; Khan, S.; Zamir, S.W.; Anwer, R.M.; Khan, F.S. EdgeNeXt: Efficiently Amalgamated CNN-Transformer Architecture for Mobile Vision Applications. *arXiv* **2022**, arXiv:2206.10589.
45. Li, Y.; Yuan, G.; Wen, Y.; Hu, J.; Evangelidis, G.; Tulyakov, S.; Wang, Y.; Ren, J. EfficientFormer: Vision Transformers at MobileNet Speed. *arXiv* **2022**, arXiv:2206.01191.
46. Yang, J.; Li, C.; Zhang, P.; Dai, X.; Xiao, B.; Yuan, L.; Gao, J. Focal Self-attention for Local-Global Interactions in Vision Transformers. *arXiv* **2021**, arXiv:2107.00641.
47. Hatamizadeh, A.; Yin, H.; Heinrich, G.; Kautz, J.; Molchanov, P. Global Context Vision Transformers. *arXiv* **2023**, arXiv:2206.09959.
48. Graham, B.; El-Nouby, A.; Touvron, H.; Stock, P.; Joulin, A.; Jégou, H.; Douze, M. LeViT: A Vision Transformer in ConvNet's Clothing for Faster Inference. *arXiv* **2021**, arXiv:2104.01136.
49. Yang, C.; Wang, Y.; Zhang, J.; Zhang, H.; Wei, Z.; Lin, Z.; Yuille, A. Lite Vision Transformer with Enhanced Self-Attention. *arXiv* **2021**, arXiv:2112.10809.
50. Tu, Z.; Talebi, H.; Zhang, H.; Yang, F.; Milanfar, P.; Bovik, A.; Li, Y. MaxViT: Multi-Axis Vision Transformer. *arXiv* **2022**, arXiv:2204.01697.
51. Tolstikhin, I.; Houlsby, N.; Kolesnikov, A.; Beyer, L.; Zhai, X.; Unterthiner, T.; Yung, J.; Steiner, A.; Keysers, D.; Uszkoreit, J.; et al. MLP-Mixer: An all-MLP Architecture for Vision. *arXiv* **2021**, arXiv:2105.01601.
52. Chen, Y.; Dai, X.; Chen, D.; Liu, M.; Dong, X.; Yuan, L.; Liu, Z. Mobile-Former: Bridging MobileNet and Transformer. *arXiv* **2022**, arXiv:2108.05895.
53. Heo, B.; Yun, S.; Han, D.; Chun, S.; Choe, J.; Oh, S.J. Rethinking Spatial Dimensions of Vision Transformers. *arXiv* **2021**, arXiv:2103.16302.
54. Yu, W.; Luo, M.; Zhou, P.; Si, C.; Zhou, Y.; Wang, X.; Feng, J.; Yan, S. MetaFormer Is Actually What You Need for Vision. *arXiv* **2022**, arXiv:2111.11418.
55. Wang, W.; Xie, E.; Li, X.; Fan, D.P.; Song, K.; Liang, D.; Lu, T.; Luo, P.; Shao, L. Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions. *arXiv* **2021**, arXiv:2102.12122.
56. Chen, C.F.; Panda, R.; Fan, Q. RegionViT: Regional-to-Local Attention for Vision Transformers. *arXiv* **2022**, arXiv:2106.02689.
57. Li, W.; Wang, X.; Xia, X.; Wu, J.; Li, J.; Xiao, X.; Zheng, M.; Wen, S. SepViT: Separable Vision Transformer. *arXiv* **2023**, arXiv:2203.15380.
58. Han, K.; Xiao, A.; Wu, E.; Guo, J.; Xu, C.; Wang, Y. Transformer in Transformer. *arXiv* **2021**, arXiv:2103.00112.
59. Guo, M.H.; Lu, C.Z.; Liu, Z.N.; Cheng, M.M.; Hu, S.M. Visual Attention Network. *arXiv* **2022**, arXiv:2202.09741.
60. Mamaev, A. Flowers Recognition, Version 2. 2021. Available online: https://kaggle.com/datasets/alxmamaev/flowers-recognition/data (accessed on 22 June 2024).
61. Ajayi, G. Multi-Class Weather Dataset for Image Classification, Mendeley Data. 2018. V1. Available online: https://data.mendeley.com/datasets/4drtyfjtfy/1 (accessed on 22 June 2024).