


Research Article

Fine-Tuning Depth Analysis: Identifying the Sweet Spot for Maximum Accuracy in CNNs

Adebayo Rotimi Philip¹ 

¹Dept. of Artificial Intelligence/African Centre Excellence on Technology Enhanced Learning (ACETEL), National Open University, Lagos, Nigeria

*Corresponding Author: 

Received: 21/Apr/2025; Accepted: 22/May/2025; Published: 30/Jun/2025. | DOI: <https://doi.org/10.26438/ijsrcse.v13i3.703>

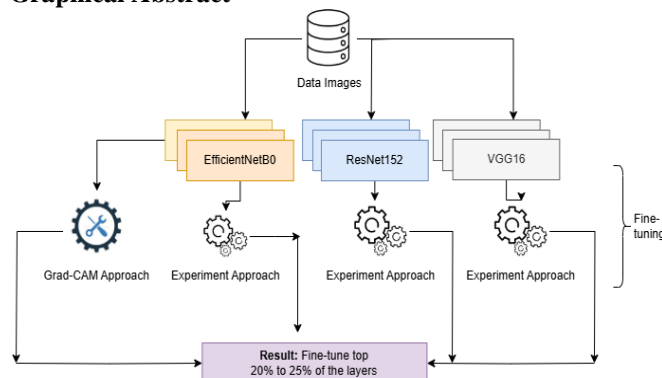


Copyright © 2025 by author(s). This is an Open Access article distributed under the terms of the [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/) which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited & its authors credited.

Abstract— Convolutional Neural Networks (CNN) have transformed the field of computer vision through their exceptional performance in image classification, face recognition, and object detection. Much of its success is due to the development of transfer learning, where pre-trained models trained on large datasets such as ImageNet are fine-tuned (adapted) for new tasks, even when only limited labeled data is available. Fine-tuning involves gradually unfreezing and retraining a pre-trained model to obtain optimal accuracy. However, determining the optimal number of layers to unfreeze remains a critical challenge. Unfreezing too few layers may limit the model's ability to adapt to task-specific features, leading to under-fitting, while fine-tuning too many layers risks over-fitting, thereby compromising generalization on unseen data. This research aims to systematically determine the number of layers to fine-tune in a pre-trained CNN to achieve optimal performance. Grad-CAM and experimental approach, which involves fine-tuning ResNet152, EfficientNetB0, and VGG16 networks, are used. Findings show that unfreezing one-fifth (20% to 25%) of the top layers gives optimal performance and unfreezing too many layers leads to overfitting. However, the VGG16 network requires unfreezing the entire layers for optimal performance because of its few layers (18 layers). Future research can consider other pre-trained networks to ascertain the findings of this study. This research is significant to AI researchers, AI engineers, data analysts, and individuals who build AI systems.

Keywords— Convolutional Neural Network, Fine-tuning, Transfer Learning, pre-train, Grad-CAM, Unfreezing.

Graphical Abstract



1. Introduction

Convolutional Neural Networks (CNNs) have transformed the field of computer vision through their exceptional performance in image recognition, semantic segmentation, object detection, and facial recognition [1]. These

unimaginable achievements are made possible through the development of transfer learning, where a pre-trained model trained on a large dataset such as ImageNet, is retrained for new tasks with limited labeled data. Transfer learning improves the learning process by leveraging knowledge from previous tasks to inform the target task, thereby enhancing efficiency and enabling faster, more cost-effective solutions. [2]. Furthermore, transfer learning is most applicable when there is not enough data to build a machine-learning model [3]. Researchers can leverage pre-trained models and fine-tune them to suit the constraints of a limited dataset. For instance, Huynh, Li, and Giger (2016) achieved good performance in the classification of mammographic tumors with just 607 digital mammographic images through transfer learning [4]. Also, Kermany et al (2018) utilized transfer learning to classify images for macular degeneration and diabetic retinopathy with limited data [5]. The approach not only accelerates model development but also enhances performance, especially in scenarios with constrained computational resources and data availability [6].

Transfer learning typically involves unfreezing certain layers of a pre-trained model and fine-tuning them on task-specific data [3]. For instance, the last few layers of an EfficientNetB0 model, a CNN model with 237 layers, can be fine-tuned (unfreeze) to update its weights in order to get acclimatized with the training dataset, to achieve better performance. However, determining the ideal number of layers to fine-tune remains a critical challenge [7]. Fine-tuning too few layers may limit the model's ability to adapt to task-specific features, leading to underfitting, while fine-tuning too many layers risks overfitting, thereby compromising generalization on unseen data. Currently, there is no established computational method to precisely determine or calculate the ideal number of layers for fine-tuning to achieve optimal performance. This process often involves iterative experimentation with different configurations of trainable layers to find the combination that yields the best results.

This research focuses on systematically determining the number of layers in a pre-trained CNN architecture that needs to be fine-tuned to obtain optimal performance. The research will investigate the impact of varying the number of trainable layers during fine-tuning of transfer learning models, specifically leveraging the EfficientNetB0, ResNet152, and VGG16 architectures. The goal is to identify the configuration that maximizes both accuracy and validation accuracy on a benchmark dataset, thereby establishing guidelines for optimal model adaptation in practical applications. By integrating computational experiments and Gradient-weighted Class Activation Mapping (Grad-CAM) machine learning principles, this research intends to provide empirical insights into optimizing the number of trainable layers in transfer learning architectures for enhanced performance across diverse visual recognition tasks.

1.1. Aims and objectives

This study aims to accomplish the following objective:

1. Assess how the number of trainable layers affects model performance: conduct empirical experiments to analyze how varying the number of trainable layers in transfer learning networks such as EfficientNetB0, ResNet152, and VGG16 architecture affects both accuracy and validation accuracy. Besides, the research aims to measure performance metrics across a benchmark dataset to quantify the influence of different layer configurations on model adaptability and generalization.
2. Establish guidelines for optimal fine-tuning: identify and document configurations of trainable layers that consistently yield optimal performance.
3. Derive insights from experimental and Grad-CAM analysis to define guidelines that inform scientists on selecting the appropriate number of layers for fine-tuning to obtain optimal training and validation accuracies.

2. Related Work

Training transfer learning models involve fine-tuning to obtain better performance. A critical decision in transfer learning involves selecting the number of layers to fine-tune (unfreeze) in the pre-trained model. This decision can significantly impact training accuracy, validation accuracy, and computational efficiency. This literature review examines several studies to determine the optimal number of layers to unfreeze to obtain the best accuracy. Recent studies consider fine-tuning under three sub-categories: fine-tuning approaches, evaluation of multiple pre-trained CNNs, and the impacts of fine-tuning depth on model performance.

2.1. Fine-Tuning Approaches

The fine-tuning approach has been very effective in improving the performance of pre-trained models. Several studies have investigated different approaches to determine the range of layers to unfreeze to maximize the accuracy of a pre-trained network. While the experimental approach is the most common [8], [9], [13], Vrbančič and Podgorelec (2020) employed the Differential Evolution-based Fine-Tuning (DEFT) method for determining the range of layers to unfreeze for identifying osteosarcoma from a medical imaging dataset [12]. This method was evaluated for osteosarcoma detection from medical images and compared with a conventionally trained CNN, a pre-trained model, and a fine-tuned model with manually selected layers. DEFT outperformed the other methods by margins ranging from 4.45% to 32.75% in classification accuracy.

Houlsby et al. (2020) introduced transfer learning with adapter modules, a compact and extensible approach that adds only a few trainable parameters per task while keeping the original network fixed [11]. This method enables the addition of new tasks without retraining previous ones, enabling extensive parameter sharing. Demonstrating this, adapter modules were used to transfer the BERT Transformer model to 26 diverse text classification tasks, including the GLUE benchmark. The adapters achieved near state-of-the-art performance within 0.4% of full fine-tuning on GLUE, while adding only 3.6% parameters per task, compared to 100% for full fine-tuning.

Masyitah et al. (2022) examined the performance of pre-trained CNN models (VGG-Net, MobileNet, ResNet, and DenseNet) in classifying visual field (VF) defects [2]. Their approach involved fine-tuning and hyperparameter tuning with a batch size of 32, 50 training epochs, and the ADAM optimizer. Findings show that VGG-16 achieved 97.63% accuracy. Bayesian optimization was employed for automated hyperparameter tuning and fine-tuning. DenseNet-121 model obtained an accuracy of 98.46% validation accuracy and 99.57% test accuracy, respectively.

2.2. Empirical Evaluation of Multiple Pre-trained Architectures

While the pre-trained model is widely used in medical imaging classification, their performance varies significantly

due to the fine-tuning strategy employed and the specific medical imaging domain, such as X-ray, MRI, Histology, Dermoscopy, and Endoscopic surgery. A few studies compared different CNN architectures across varying fine-tuning settings to identify architecture-specific behavior. Davila, Colan, and Hasegawa (2024) investigated the challenges of optimizing fine-tuning methods for pre-trained models in medical image analysis [29]. Eight fine-tuning approaches were employed with three pre-trained models: ResNet-50, DenseNet-121, and VGG-19. They found that the performance of the fine-tuning approach depends largely on the CNN architecture and the types of images. DenseNet-121 performed better than the traditional fine-tuning approach.

Kumar, Anuar, and Hassan (2022) investigated the performance of various pre-trained models (SqueezeNet, GoogleNet, ShuffleNet, Darknet-53, and Inception-V3) across different epochs, learning rates, and mini-batch sizes [10]. Using confusion matrices for evaluation, the experiments showed that Inception-V3 achieved the highest accuracy of 96.98%, along with precision of 92.63%, sensitivity of 92.46%, specificity of 98.12%, and an F1-score of 92.49%.

2.3. Impact of Fine-Tuning Depth on Model Performance

These studies explicitly examine how the number or depth of fine-tuned layers affects performance metrics such as accuracy, overfitting, and generalization. Karlsson and Runelöv (2021) researched a pre-trained AlexNet architecture and adopted fine-tuning to train the network to diagnose lung diseases from chest X-rays [8]. They investigated how deeply to fine-tune the architecture to achieve the best accuracy, sensitivity, and specificity. The network was divided into five blocks, resulting in five different fine-tuning depths. Although all models produced promising results, they failed to learn the intended features. Instead, the models resorted to shortcut learning by identifying the image origin rather than differences in the lungs. Consequently, the research question of this thesis remains unresolved.

The study done by Gupta and Gupta (2020) partly resolves the inconclusive result of Karlsson and Runelöv (2021). They studied the effects of fine-tuning a pre-trained image classification model on the accuracy of binary classification tasks [9]. Retraining the VGG-16 model with 640 medical images and 65 testing images over 100 epochs, they found that unfreezing the lower layers initially improved validation accuracy, followed by a decline.

Kandel and Castelli (2020) conducted experiments on two histopathology datasets using three state-of-the-art architectures to study the effect of block-wise fine-tuning of CNNs [1]. They found that fine-tuning the entire network does not result in the best validation accuracy, but, leads to overfitting and requires more computational resources during training. For shallow networks, in particular, fine-tuning only the top blocks can save time and computational resources while producing more robust classifiers.

Ki-Sun Lee et al (2020) investigated the impacts of fine-tuning different numbers of convolutional blocks in VGG-16 and VGG-19 architectures for COVID-19 detection using chest X-ray images [13]. They found that fine-tuning up to three convolutional blocks improves accuracy while fine-tuning beyond three blocks decreases accuracy. Furthermore, the study by Amiri, Brooks, and Rivaz (2020) investigated the effects of fine-tuning different parts of the U-Net architecture for breast ultrasound image segmentation [14]. Their findings align with those of Ki-Sun Lee et al. (2020) which revealed fine-tuning the encoder part, while keeping the decoder part of the U-Net frozen, resulted in better segmentation performance. Also, including more layers from shallow to deep during fine-tuning led to improved results.

Taormina et al. (2020) investigated the impact of fine-tuning various layers of convolutional neural networks (CNNs), with a focus on AlexNet, for the classification of HEp-2 cell images [15]. Their study also included a comparative analysis of four widely used pre-trained models: AlexNet, SqueezeNet, ResNet18, and GoogLeNet. Using a public dataset, the models were evaluated based on accuracy and the area under the Receiver Operating Characteristic (ROC) curve (AUC). The findings highlighted the advantage of selective fine-tuning (retaining the early layers for general feature extraction while adapting the deeper layers to the specific classification task), demonstrating improved performance through task-specific layer adjustment.

Adepoju et al (2024) investigated how fine-tuning different layers of pre-trained models impacts classification performance, particularly in high-precision tasks such as medical diagnosis [30]. They used InceptionV3 and Xception architectures to classify breast cancer from mammographic images. Systematic fine-tuning approaches were employed to assess performance across varying layers and findings show that InceptionV3 achieved slightly better performance (0.65) than Xception (0.64) without fine-tuning. Accuracy increased to 0.66 when the last two block layers were fine-tuned and decreased to 0.61 upon fine-tuning all the layers.

2.4. Gaps in the literature

Despite the numerous studies on exploring the number of optimal layers to fine-tune in transfer learning, more studies are necessary to justify the range of layers to unfreeze in a pre-trained architecture to obtain optimal performance. The studies asserted that fine-tuning layers requires a balanced approach to prevent underfitting and overfitting the model. Many studies adopted various pre-trained models, fine-tuning strategies, and datasets to answer these research questions. However, the studies reveal varying findings. For instance, while some research supports shallow fine-tuning to avoid overfitting and reduce computational costs, others suggest that deeper fine-tuning can improve performance depending on the task. This inconsistency makes it challenging to draw an approximate conclusion on how deep to fine-tune across different domains or model architectures.

Even with the inconsistencies, much of the literature supports unfreezing the top few layers. However, more studies are

required to support this assertion and to provide an acceptable guideline on the number of layers to unfreeze to consistently achieve optimal performance.

Lastly, there is limited research that integrates explainable AI tools such as Grad-CAM to analyze and interpret the impact of fine-tuning depth on model decision-making and feature extraction. While performance metrics such as accuracy and

AUC are frequently used, they do not provide insight into why a particular fine-tuning configuration performs better. Few studies combine quantitative results with visual explanations to guide the selection of trainable layers. This creates a gap in practical, interpretable methodologies that can inform scientists not only on what layers to fine-tune but also on why those layers contribute to improved training and validation performance.

The table below summarizes the literature review and their corresponding findings

Table 1 summarizes the literature review

Author(s)	Problem definition	Methodology	Findings
Vrbančič & Podgorelec (2020) [12]	Determine optimal layers to unfreeze for medical image classification (osteosarcoma).	Differential Evolution-based Fine-Tuning (DEFT); compared with manual fine-tuning and conventional training.	DEFT outperformed others by 4.45%–32.75% in accuracy.
Houlsby et al. (2020) [11]	Proposed transfer learning with adapter modules Reduce resource cost of transfer learning while maintaining performance.	Adapter modules added to BERT; tested on 26 NLP tasks including GLUE benchmark.	Achieved 0.4% less than full fine-tuning with only 3.6% extra parameters vs 100%.
Masyitah et al. (2022) [2]	Classify visual field defects using pre-trained CNNs.	Fine-tuned VGG, MobileNet, ResNet, DenseNet; hyperparameter tuning with Bayesian optimization.	DenseNet-121 achieved 98.46% (val) and 99.57% (test) accuracy.
Davila, Colan & Hasegawa (2024) [29]	Compare fine-tuning strategies across architectures for medical imaging.	Used 8 fine-tuning strategies on ResNet-50, DenseNet-121, VGG-19.	DenseNet-121 outperformed other strategies; performance is architecture- and image-type-dependent.
Kumar, Anuar & Hassan (2022) [10]	Evaluate various pre-trained models across hyperparameter configurations.	Tested SqueezeNet, GoogleNet, ShuffleNet, etc., with varied learning rates, epochs.	Inception-V3 performed best: 96.98% accuracy, high F1-score and specificity.
Karlsson & Runelöv (2021) [8]	Determine fine-tuning depth for lung disease classification.	Fine-tuned AlexNet at 5 different block depths on chest X-rays.	Models used shortcut learning; failed to learn medical features.
Gupta & Gupta (2020) [9]	Examine how fine-tuning depth affects binary classification.	Fine-tuned VGG-16 using 640 train / 65 test images over 100 epochs.	Validation accuracy improved then declined with deeper unfreezing.
Kandel & Castelli (2020) [1]	Study block-wise fine-tuning on histopathology datasets.	Used 3 CNNs on 2 datasets; tested full vs partial fine-tuning.	Full fine-tuning led to overfitting; tuning top blocks was more efficient.
Ki-Sun Lee et al. (2020) [13]	Optimize fine-tuning depth in VGG-16/19 for COVID-19 detection.	Fine-tuned 1–5 blocks on chest X-rays.	Accuracy improved up to 3 blocks; decreased beyond that.
Amiri, Brooks & Rivaz (2020) [15]	Analyze effect of encoder vs decoder tuning in U-Net.	Partial fine-tuning on U-Net for breast ultrasound segmentation.	Encoder-only tuning improved segmentation results.
Taormina et al. (2020) [14]	Understand selective fine-tuning for HEp-2 cell image classification.	Compared AlexNet, SqueezeNet, ResNet18, GoogLeNet; varied fine-tuning layers.	Selective tuning of deeper layers improved performance (accuracy, AUC).
Adepoju et al. (2024) [30]	Investigate impact of layer-specific fine-tuning in cancer diagnosis.	Used InceptionV3 & Xception; systematically fine-tuned layers.	Accuracy rose from 0.65 to 0.66 (last 2 blocks), fell to 0.61 (full fine-tuning).

3. Fine-tuning in Transfer Learning

To understand fine-tuning and the processes involved in fine-tuning a transfer learning network, the CNN must be clearly understood. We define fine-tuning as the process of taking a pretrained model (that is, a model that has learned useful features from a large dataset) and adapting the model to learn

another often related task-specific dataset [7]. For instance, the EfficientNet series was trained on the ImageNet-1k dataset of cats. However, we can use the same model to classify medical images with few datasets. This is highly significant in model development as it allows researchers to leverage existing solutions to solve other related problems.

According to Karlsson and Runelöv (2021), transfer learning involves using existing knowledge to solve related tasks [8]. This is how the human brain solves problems. The human brain has learned from experiences over the years. At a time when people are required to make concrete decisions, the brain transfers the knowledge it has obtained from previous experiences to learn the pending situation [16]. This enables the brain to make informed decisions by drawing on prior knowledge and experience. In the same vein, transfer learning is based on the idea that knowledge gained from large data can be transferred to solve other related problems.

3.1. Cognitive Process of Fine-Tuning: Human Brain Analogy

The brain adapts and refines previously learned knowledge to learn new tasks through the process of refinement [16]. Drawing on cognitive science and neuroscience, the human brain learns by a dynamic interaction between prior knowledge (existing mental schemas) and new situations. According to the Cognitive Science Insight, Schema theory (Bartlett, 1932) suggests that the brain stores generalized information that helps in understanding new situations [17]. This process involves making small modifications to existing schemas or forming new sub-schemas that are more specific to the new context [17]. Carey (1991) argued that the concept of fine-tuning is similar to the conceptual change, where existing knowledge is modified [18]. Neuroplasticity (Hebb, 1949) also supports the idea that the brain can readjust or reorganize to adapt its neural connections, reinforcing useful patterns while discarding irrelevant or outdated ones [19]. Hatano and Inagaki (1986) supported the argument implying that the transfer process is akin to adaptive expertise, where individuals modify their approach and methodologies as they gain more expertise, revealing that the prefrontal cortex and hippocampus are involved in human memory management and long-term memory integration [20].

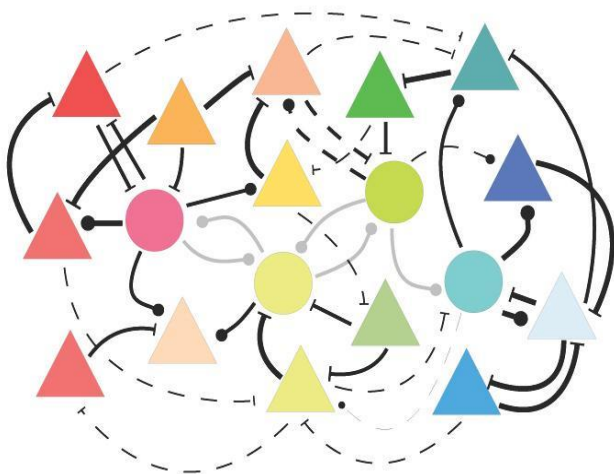


Figure 1 shows fine-tuning processes in the brain [16]

Figure 1 above explain the processes involved in knowledge refinement of the brain. The brain readjusts or refines existing knowledge to solve present situations [32].

3.2. CNN architecture

Since this research adopt image dataset, it is necessary to explain the CNN architecture to demonstrate the fine-tuning processes. There are so many CNN models that have been developed over the years. However, they almost follow similar architecture even though some architecture may differ slightly from others.

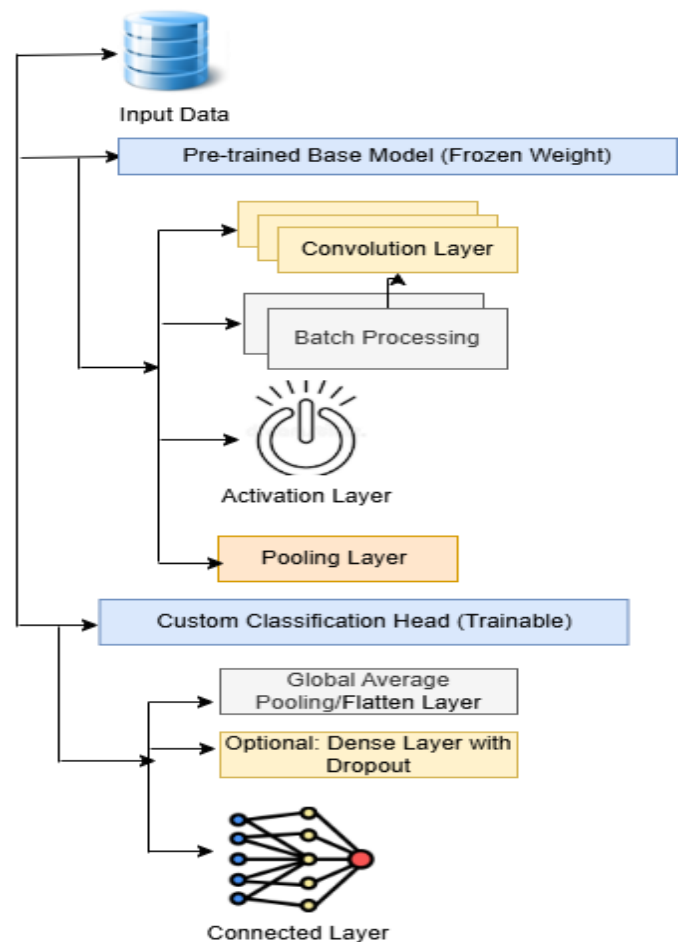


Figure 2 shows the CNN architecture

3.2.1. Input layer

The input layer of a CNN is the first layer that receives the raw image data for processing. It accepts images with specific characteristics, such as dimensions and channels. The dimension of the image is its size in pixels (height x width), while the channel indicates a colored image [Red, Green, Blue]. For instance, EfficientNetB0, a pre-trained CNN architecture, accepts images of dimensions [224 x 224 x 3]. 224 by 224 represents the height and width of the image, while 3 indicates the channels. Grey images are represented by [height x width x 1], which indicates that grey images require a single channel to indicate the intensity of the pixels.

3.2.2. Pre-trained Base Model (Frozen weights)

The next layer of the CNN network from Figure 2 is the pretrained layer. The pretrained model is also called the base model, and it has several sub-layers, which are frozen. The

pre-trained base model refers to the part of the architecture that has been pre-trained on a large dataset like ImageNet. Before fine-tuning, the base layers are frozen, enabling the architecture to maintain the features of the model trained with ImageNet. During training, the weights of the base model are kept constant (not updated), allowing the model to make predictions on a new dataset with the features of the trained model on ImageNet. Training the base model (with frozen layers) with new datasets offers several advantages, one of which is the ability to train effectively with a small image dataset [12]. Second, the trained model can leverage the feature extraction capabilities of the base model [12]. Third, when the base model is kept frozen, the model is only trained with the custom layer. This improves the speed of training and also reduces the likelihood of overfitting [12].

The pretrained base model, as shown in Figure 2, has four layers: convolutional layers, batch normalization layers, activation layers, and pooling layers, though the arrangement of the layers can vary slightly in different CNN architectures like EfficientNetB0 and VGG16 architectures.

I. Convolutional layers

The convolution layer is one of the most significant layers of the CNN architecture. Its function is to extract features from the images [21]. Convolution is a process of extracting features from the input images and it does this by computing the dot products of the image matrix with the kernel matrix as shown in Figure 3.

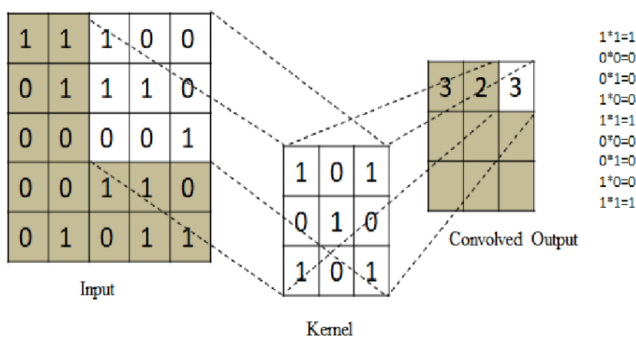


Figure 3 shows the convolution process [21]

The kernel is a 3x3 matrix, and it is used to multiply the image matrix to produce an activation map (convoluted output). The process performs a dot product, which is added to produce the activation map, as shown. This procedure is carried out for each of the image matrices. The stride allows the kernel matrix to move over the image matrix. A stride set as 1 means that the kernel moves over the image matrix one to the left, right, up, and down to calculate the convoluted output. Increasing the dimension of the kernel to 5x5 or 7x7 and the stride to 2 or more affects the depth of the features extracted negatively (more detailed features will be extracted when the kernel matrix and the stride are smaller), though the convolution process will be faster [22].

II. Batch normalization

To simplify batch normalization, it is necessary to consider each word separately. Normalization is a technique in machine learning that entails converting the pixel values of an image to a standard range, typically between 0 and 1 or -1 and 1. The essence of normalization is to enhance the convergence speed and stability of training algorithms [22]. It also ensures that the numeric input values of the images are uniform, mitigating gradient vanishing or explosion problems during the training phase [22]. This preprocessing step is particularly significant when using activation functions such as sigmoid or hyperbolic tangent (tanh), as they are sensitive to the input data scale.

The study done by Norhikmah, Afdhal, and Rumini (2022) reaffirms that normalization enhances overall performance and improves the training process [23]. They attributed the improved performance to the reduction in numerical instability and the promotion of faster gradient descent convergence. Krizhevsky, Sutskever, and Hinton (2012) found that normalization prevents CNNs from using a wide range of numerical data, thus promoting the model's ability to generalize across various datasets [22]. Normalization techniques include min-max normalization, which scales the pixel values within a specified minimum and maximum range, and z-score normalization, which scales the numeric values based on the mean and standard deviation of the dataset.

To simplify, the image dataset in EfficientNetB0 has a dimension of 225 x 225 pixels. In other words, each pixel in the dataset is within the range of 0 to 225. This dataset will not scale properly within this range during training, thus, a need for normalization [25]. Normalization is done by dividing the data by 225 to convert the data to within 0 and 1 [27]. For batch normalization, the data are converted to within 0 and 1 after the convolutional process. However, this process is done in a batch of 32 and not a single input.

III. Pooling layer

The pooling layer is a significant component of the CNN, which is used primarily for down sampling (reducing) the activation map (feature map) created by the convolution layer [24]. The essence is to focus on the most important features while reducing the image size. Imagine you have the image of a cat, and you want to shrink it, but you don't want to lose the most important part, which is the eyes, edges, and shapes. Pooling operations work by sliding a window across the input feature map and summarizing the most important features within that window [25]. There are several types of pooling layers but the most common are max pooling and average pooling. While max pooling selects the maximum values from the window, capturing the most important features, the average pooling calculates the average values, providing a more generalized representation of the features.

Figure 4 below shows the pooling process. Here, the pooling process selects the maximum number of the feature map.

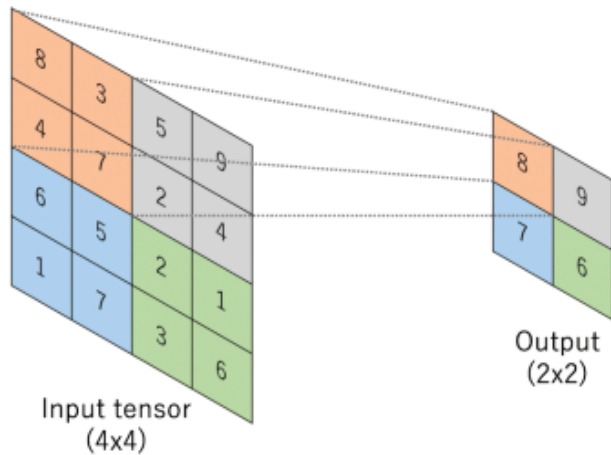


Figure 4 shows the max pooling process of a CNN [25]

Max pooling is widely used in CNNs due to its ability to retain the most important features while discarding less relevant information. For example, if a feature is highly activated within a certain region of the input image, max pooling will ensure that this activation is preserved in the down sampled output. This process reduces the spatial size of the feature maps and also helps to make the network more invariant to small translations and distortions in the input data. As a result, pooling layers play a crucial role in enhancing the robustness and efficiency of CNNs by enabling the network to focus on the most critical features and reducing the computational burden associated with large input dimensions [25].

Here's a detailed breakdown of the formula:

$$Output(i, j) = \max_{(m, n) \in window(i, j)} Input(m, n) \text{ ----- (1)}$$

Output (i, j): The value of the pooled output at position (i, j)

The table below summarizes some terms in the CNN sample architecture which are not preciously explained.

Table 2 shows some terminologies used in CNN network

Layers	Parameters	Meaning
1	Input layer	The input layer accept the images in dimension $[225 \times 225 \times 3]$, where $[225 \times 225]$ represent the image dimensions and the 3 represent the image channel [Red, Blue, and Green]
2	Conv2D (3x3, stride 2)	Con2D
		3 x 3
		Stride 2
		padding
3	BatchNorm	The batch normalization reduces the internal covariate shift (i.e. adjust the out of the activation map into values between 0 and 1), speed up convergence, and reduce overfitting.
4	Swish Activation	An activation function introduced by Google in 2017. It outperforms ReLU in deep networks (~0.5 – 1% improvement) and improves smooth gradients

3.4. Fine-Tuning (Unfreezing) the CNN Layers

Having understood the CNN network, we can unfreeze nearly all the layers in the transfer learning network though such act is not advisable because training all layers do not amount to better accuracy. Besides, the more the number of layer

Input (m, n): The value of the input feature map at position (m, n) within the window.

Window (i, j): The window of size (e.g., 2x2, 3x3) that is applied at position (i, j) on the input feature map.

Max: The max function, which selects the maximum value within the window.

3.2.3. The Custom Head Classification

The custom head is the third part of the transfer learning network. It represents the fully connected layer. The earlier layers (convolutional, normalization, and pooling layers) scan the images, spot edges, textures, and shapes, and transform the images into a list of features. The head classification looks at the features (numbers) and decides which images they are. If we have the images of dogs, cats, and rabbits at the input layers, the CNN learns from training data and classifies the test data into their corresponding image categories [33].

How does it work? After the features are flattened in the flatten layer into 1 1-dimensional vector, the fully connected layer processes this vector by applying a linear transformation followed by a non-linear transformation function. The transformation enables the network to learn complex representations and relationships, guiding the network to classify the input data more accurately. Typically, the fully connected layer is positioned at the end of the network, and is responsible for producing the final output, such as class scores in classification tasks [28]. The head classification layer can be customized. The researcher can decide the number of fully-connected layers, use dropout/batch-norm for regularization, the number of labels, and the activation function, which could be Softmax for single-label classification or Sigmoid for multi-label classification [34].

unfrozen, the more the computational resources required. There is no definitive method for determining the optimal number of layers to fine-tune in a transfer learning network to achieve maximum accuracy. This is the challenge that this study aims to unravel.

4. Methodology

In this study, systematic and empirical approach are used to obtain the optimal number of layers to unfreeze. The steps taken are summarized below:

1. Collect the image datasets
2. Set up the experiments
3. Training and fine-tune EfficientNetB0, ResNet152, and VGG16
4. Plot the graph and use regression to find the equation
5. Use the Grad-CAM method to also determine the range of layers to fine-tune to obtain optimal accuracy

4.1. Prepare Your Data

More than one dataset is used in the experiment. The essence is to have a variety of outcomes. Split your dataset into training, validation, and test sets. Ensure your data is properly preprocessed and augmented if necessary.

I. Food-101 dataset

The Food-101 dataset is a large-scale collection of food images categorized into 101 classes, each representing a different type of food. The dataset contains 101,000 images, with 1,000 images per class. 75% of the dataset make up the training data, while 25% make up the test data. 10% of the Food-101 dataset is used for the experiments to enhance faster processing times and reduce computational requirements. It also allows for quicker iterations and experimentation cycles. 10% of the Food-101 dataset amount to 10,100 images, 75% of whose is used for training and 25% for testing.



Figure 6 shows the Food-101 dataset

II. CIFAR-10 dataset

The CIFAR-10 dataset consists of 60,000 labeled images divided into 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each class contains 6,000 images. The dataset was created by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The images are split into 50,000 for training and 10,000 for testing.

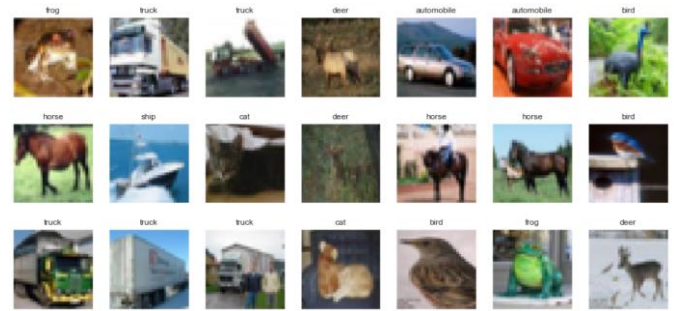


Figure 7 shows the CIFAR-10 dataset

III. Stanford Dogs Dataset

The Stanford Dogs dataset is a well-known dataset for fine-grained image classification, specifically focused on different breeds of dogs. The dataset has 20,580 images with 12,000 training images and 8,580 test images. It has a class of 120 different breeds of dog and each class has 120 images of the same type of dog. The images generally vary in resolution, often ranging from around 200x200 pixels to more than 500x500 pixels, and many images are larger than this range, often going up to 1024x1024 pixels or higher. Given the high-resolution nature of the images, the dataset is well-suited for tasks that require capturing fine details and textures, which is important for fine-grained classification tasks like distinguishing between different breeds of dogs.



Figure 8 shows the Stanford Dogs dataset

4.1.1. Reasons for different dataset

Training machine learning models with a wide variety of data types is crucial for improving their robustness and generalization capabilities, which helps the model to perform better on real-world tasks where input data can be highly variable and unpredictable [11]. By training with different data types, models can also handle noise and variability more effectively, leading to improved performance across various applications [29].

Besides, training a machine learning model with a wide variety of data types helps prevent over-fitting, where a model becomes accustomed to the training dataset, but performs badly in the testing dataset. In other words, when models are trained on a homogeneous dataset, they may fail to generalize to new and unseen data that differ from the training dataset. By integrating multiple data types, the model gains a deeper understanding of the underlying relationships within the data, leading to improved generalization and more accurate predictions. This approach is particularly beneficial in complex tasks, such as multi-modal learning and real-

world applications, where data is rarely uniform and often comes from multiple sources.

4.2. Set up Baseline Experiment

The experimental approach involves unfreezing layers by layers and training the architecture to determine the accuracy of each. In fact, most methods, such as statistical, computation, etc., rely on the experimental approach to obtain the optimal solutions. The experimental approach has the following steps, which are discussed in detail:

1. After obtaining the data, prepare the environment. It could be Anaconda Notebook, Python IDE, Google
 - Fine-Tune Top Layers: Start by unfreezing only the top few layers of the pre-trained model and train the network. Record the performance on the validation set.
 - Increase Fine-Tuning Depth: Incrementally unfreeze additional layers (e.g., unfreeze the next block of layers) and repeat the training process.
 - Monitor Performance: At each fine-tuning level, evaluate the model's performance on the validation set to assess how the depth of fine-tuning influences overall effectiveness.

4.3. Grad-CAM Approach

Machine learning models especially neural network, which has been successful in solving most dynamic human problem is not interpretable. In other words, managers and organizations find it challenging to explore AI in critical situations as they lack trust in the system due to its unexplainable decision-making processes [26]. However, Grad-CAM has been useful in the interpretability of complex neural network [27]. In this research, we utilize Grad-CAM as one of the techniques to help identify the optimal number of layers to fine-tune in order to achieve the best validation accuracy.

4.3.1. How does it work?

Grad-CAM helps to identify the most important features of an image learned by machine learning models. Grad-CAM identifies the regions of an image that are most influential in a convolutional neural network's final decision by highlighting the areas with the highest impact on the output prediction [27]. This is done by indicating the importance of each pixel in relation to the class by increasing or decreasing the intensity of the pixel [28]. For instance, if a Grad-CAM visualization is used to generate the image of a cat, the Grad-CAM can indicate the extent to which different parts of the pixels of the image correspond to a cat.

The class activation map (CAM) reveals the most significant parts of the image used for model prediction. It does this by combining the values from the last convolutional layer, using weights linked to the class the model predicted [29]. The result is a heatmap that highlights key areas the model focused on. This heatmap is then resized to match the original

Colab, Visual Studio, or any other platform where Python code can be written.

2. Import the data and create the code for any of the transfer learning CNN networks (EfficientNet, ResNet16, ResNet101, Inception, etc.)
3. Run the base model and display results.
4. Incremental Fine-Tuning Approach: Gradually unfreeze more layers from the pre-trained model and fine-tune them. For each configuration, keep other hyperparameters constant. The steps are as follows:

image size so we can see exactly what the model was looking at [30].

5. Result and Discussion

5.1. Results of the experimental method

Three categories of experiments were done: ResNet152 model with cifar10 dataset, VGG16 with Stanford dogs dataset, EfficientNetB0 model with Food-101 dataset,

ResNet152 experimental results with cifar10 dataset

The ResNet152 has 152 block layers but 514 layers. Table 3 below shows the result of the fine-tuning experiments over 5 epochs

Table 3 shows the outcome of the ResNet152 experiment

Model	Layers Unfreeze	Accuracy	Loss	Val-accuracy	Val-Loss
Base Model	0	0.2961	1.9466	0.2966	1.9569
1	207	0.5345	1.3091	0.5236	1.4021
2	310	0.3714	1.6755	0.3616	1.7522
3	180	0.5108	1.3929	0.5022	1.4012
4	100	0.4834	1.4553	0.4735	1.4756
5	20	0.4422	1.5742	0.4389	1.6109
6	230	0.3914	1.6773	0.3767	1.7843
7	210	0.5259	1.3491	0.5201	1.3569
8	10	0.4407	1.5885	0.3879	1.8210

ii. Fine-tuning the VGG16 with Stanford dogs dataset

The VGG16 has 18 layers. The table below shows the result of the fine-tuning experiments over 10 epochs

Table 4 shows the outcome of the VGG16 experiment

Model	Layers Unfreeze	Accuracy	Loss	Val-accuracy	Val-loss function
Base Model	0	0.7516	0.7041	0.6217	1.1544
1	18	0.9699	0.0918	0.8373	0.6029
2	10	0.9568	0.1308	0.8136	0.6844
3	14	0.9619	0.1118	0.8361	0.5994
4	17	0.9647	0.1049	0.8315	0.6264

iii. Fine-tuning the EfficientNetB0 with 10% of the Food-101 database

The EfficientNetB0 has 237 layers. Table 5 shows the result of fine-tuning experiments over 5 epochs.

Table 5 shows the outcome of the EfficientNetB0 experiments

Model	Layers	Accuracy	Loss	Val_acc	Val_loss
Base Model	0	0.7278	1.1670	0.6243	1.4512
1	10	0.8797	0.5386	0.6635	1.2512
2	20	0.9040	0.4217	0.6662	1.2842
3	50	0.9380	0.2741	0.6642	1.3458
4	150	0.9315	0.2508	0.6491	1.4981
5	237	0.8879	0.4007	0.5867	1.7758
6	100	0.9494	0.2024	0.6452	1.5234
7	120	0.9378	0.2304	0.6352	1.5477

5.2. Interpreting the Experiments Using Graphical Method

Tables 3, 4, and 5 shows the outcome of the experiments. Based on these results, a key question arises: Can we identify a specific layer or a range of layers to fine-tune in order to achieve the best possible validation accuracy? To explore this question further, plotting a graph of the experimental results can help visualize the relationship between the layers fine-tuned and the validation accuracy. This graphical representation will provide insights that may help answer the research question more clearly.

5.2.1. Interpreting the ResNet152 results

The graphs in Figure 9 and 10 shows the validation accuracy and the validation of the ResNet152 experiments

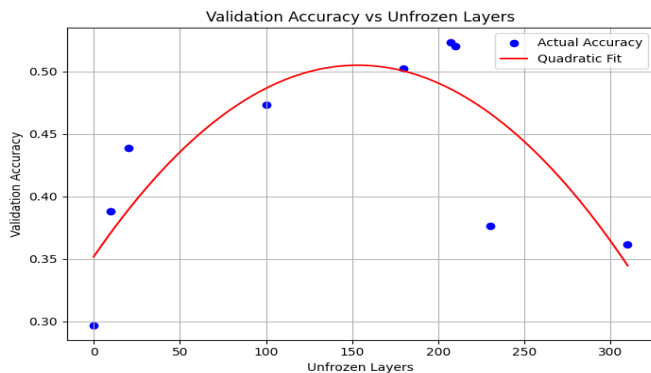


Figure 9 shows the validation accuracy against the unfrozen layer for the ResNet152 experiments

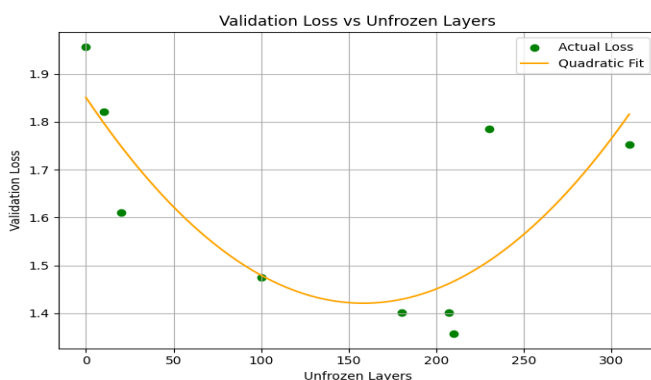


Figure 10 shows the validation loss against the unfrozen layer for the ResNet152 experiments

The curve is a **U-shape**, which confirms a **quadratic polynomial** of the form:

$$\text{Val-Accuracy} = -aL^2 + bL + c \text{ ----- (1)}$$

Where:

- L = number of unfrozen layers,
- The maximum point (vertex) seems to be somewhere around **150 layers**.

From visual estimation:

- At **0 layers**, accuracy ≈ 0.35
- At **~150 layers**, accuracy peaks at ≈ 0.51
- At **310 layers**, accuracy dips to ≈ 0.34

Using regression, this graph supports the equation:

$$\text{Val-Accuracy} \approx -0.00000673L^2 + 0.001554L + 0.35 \text{ ----- (2)}$$

(Check appendix I for how the equation is obtained)

The validation loss is a concave parabola with equation $\text{Val-Accuracy} = aL^2 + bL + c \text{ ----- (3)}$

Loss starts high at ~ 1.85 , reach the minimum at ~ 1.42 around **150 layers**, then rise again.

Supports earlier regression:

$$\text{Val-Loss} \approx 0.000017L^2 - 0.0054L + 1.85 \text{ ----- (4)}$$

Metrics

The Mean Square Error (MSE) and Mean Absolute Error (MAE) is used to determine the acceptance of the results.

Substituting the layers into the Val-accuracy equation $-0.00000673L^2 + 0.001554L + 0.35$, we obtain the table below:

Table 6 shows the actual and predicted val-accuracy of ResNet152

Layers	Actual Val-accuracy	Predicted Val-accuracy
0	0.2966	0.35
10	0.3879	0.3649
20	0.4389	0.3784
100	0.4735	0.4381
180	0.5022	0.4117
207	0.5236	0.3833
210	0.5201	0.3796
230	0.3767	0.3514
310	0.3616	0.1850

The actual Val-accuracy is obtained from the experimental results in Table 3. The MSE is calculated using the formula: , while the MAE is obtain using the formula:

The Mean Squared Error (MSE) is 0.00975 and the Mean Absolute Error (MAE) is 0.0828. A low MSE and MAE indicates that the polynomial regression model performs excellently well. A MAE of 0.0828 means that on average, the predicted values are only about 8.28% different from the actual values. A low MSE also confirms the model's good

performance, especially since MSE gives more weight to larger errors.

To determine the number of layer that gives the optimal solution, we substitute the layers from 0 to 514 into the Val-accuracy equation. The easiest way to do this is a run a Python code with input [0, 1, 2... 514] and obtain the corresponding Val-accuracy. Note, ResNet152 has 514 layers. **The outcome shows that the optimum accuracy is obtained at layer 107 at Val-accuracy of 0.439706. Fine-tuning the top 107 layers is 22% of the 514 layers of the ResNet152.**

5.2.2. Interpreting the EfficientNetB0 results

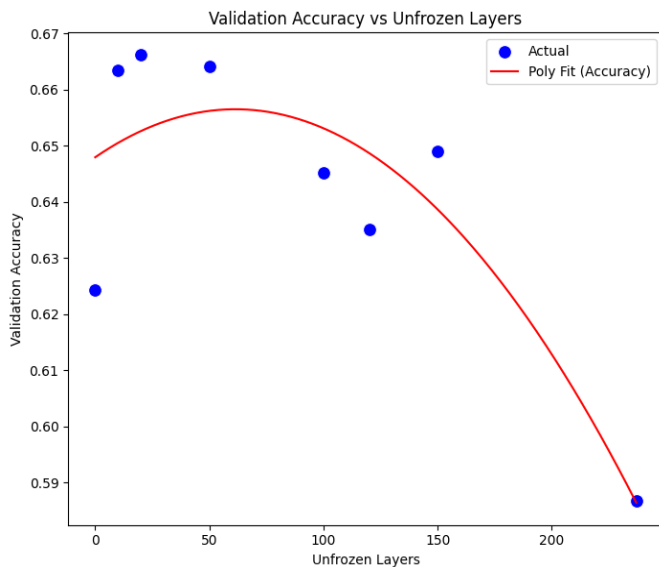


Figure 11 shows the validation accuracy against the unfrozen layer for the EfficientNetB0 experiments

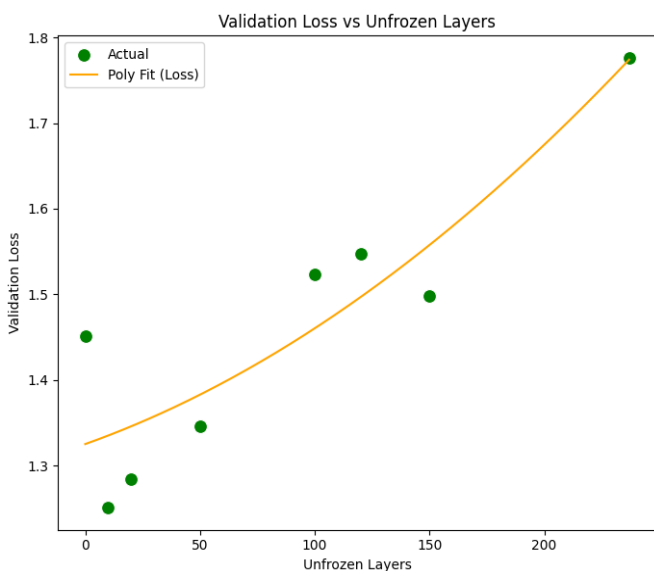


Figure 12 shows the validation loss against the unfrozen layer for the EfficientNetB0 experiments

This is a **convex (∩-shaped) parabola** → modelled with a **quadratic polynomial**.

The peak is around **50 layers**, and then accuracy declines.

So, the general form is: Val-Accuracy = $-aL^2 + bL + c$ ----- (1), where $a < 0$

Using regression, the estimated equation of the curve is Val-Accuracy $\approx -0.000015L^2 + 0.0016L + 0.625$ ----- (5)

For the val-loss function, we use: Val-loss = $aL^2 + bL + c$ ----- (3), where $a > 0$

Estimated equation (fitting the curvature): Val-loss $\approx 0.00002L^2 - 0.002L + 1.28$ ----- (6)

Metrics

Table 7 shows the actual and predicted val-accuracy of EfficientNetB0

Layers	Actual val-accuracy	Predicted val-accuracy
0	0.6243	0.6250
10	0.6635	0.6395
20	0.6662	0.6510
50	0.6642	0.6675
100	0.6452	0.6350
120	0.6352	0.6010
150	0.6491	0.5275
237	0.5867	0.1617

The Mean Squared Error (MSE) is 0.0247, and the Mean Absolute Error (MAE) is 0.0793. This means the model's predictions are, on average, quite close to the actual values. MSE shows how far off the predictions are by squaring the errors, so it gives more weight to bigger mistakes. MAE is easier to understand, showing that the predictions are off by about 0.0793 on average.

To determine the number of layers that gives the optimal solution, we substitute the layers from 0 to 237 into the Val-accuracy equation. The easiest way to do this is a run a Python code with input [0, 1, 2... 237] and obtain the corresponding Val-accuracy. Note, EfficientNetB0 has 237 layers. **The outcome shows that the optimum accuracy is obtained at layer 54 at Val-accuracy of 0.667660. Unfreezing the top 54 layers is approximately 22.7% of the 237 layers**

5.2.3. Interpreting the VGG16 results

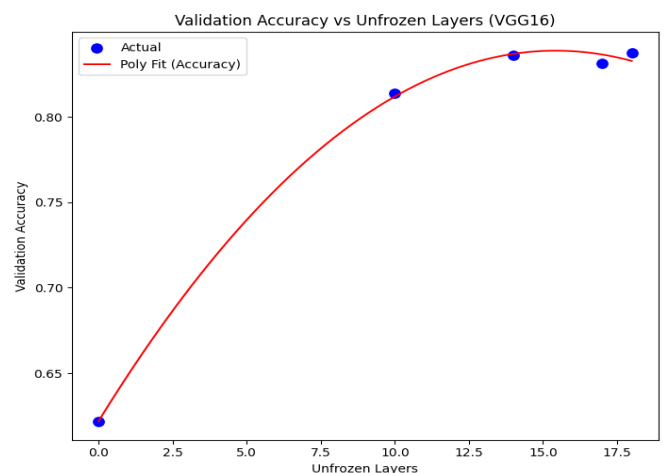


Figure 13 shows the validation accuracy against the unfrozen layer for the VGG16 experiments

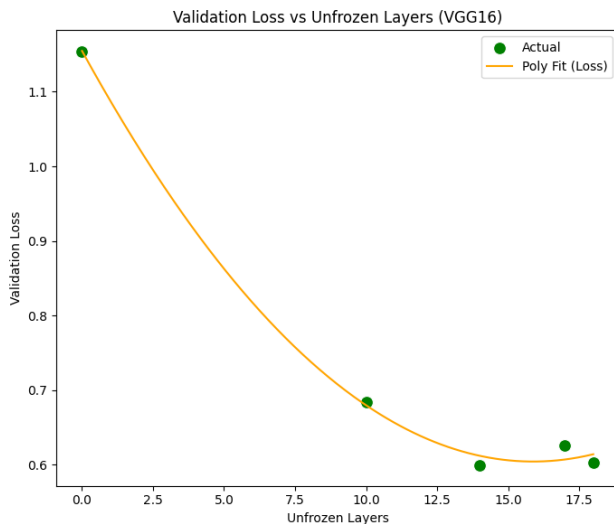


Figure 14 shows the validation loss against the unfrozen layer for the VGG16 experiments

Using polynomial regression, the

$$\text{Val-Accuracy} \approx -0.0007L^2 + 0.0306L + 0.62 \quad (7)$$

$$\text{The Val-Loss} \approx 0.0018L^2 - 0.056L + 1.15 \quad (8)$$

Metrics

Table 8 shows the actual and predicted Val-accuracy of VGG16

Layers	Actual Val-accuracy	Predicted Val-accuracy
0	0.6217	0.6200
10	0.8136	0.8560
14	0.8361	0.9112
17	0.8315	0.9379
18	0.8373	0.9440

The Mean Squared Error (MSE) is 0.00603 and the Mean Absolute Error (MAE) is 0.06646, indicating that the predicted Val-accuracy deviate from the actual values by about 0.603% in accuracy.

To obtain the number of layer that gives the optimal Val-accuracy, we substitute the 18 layers of the VGG16 into the Val-accuracy equation $-0.0007L^2 + 0.0306L + 0.62$. The easiest way to do this is to write a Python code with input [0, 1,2,3,4...18] and the Val-accuracy as the corresponding output. **The outcome shows that the optimal accuracy is obtained at layer 18, which gives a Val-accuracy of 0.944**

5.2.4. The General Equation

There are some challenges with the experimental approach. The nature, size, quantity, robustness of the dataset; and the hyperparameters enabled can impact the validation accuracy which in turn can affect the regression equation [30]. Therefore, it becomes quite challenging to have equations for different kinds of scenarios. One way to resolve this is to deduce a general equation which considers most of the parameters and hyperparameters that may affect the accuracy of the models. A simplified estimated equation for this problem must consider:

- Number of unfrozen layers (L)
- Model size (N, often total number of layers or parameters)
- Number of epochs (E)
- Sometimes, dataset size
- Constants or learnable coefficients like α (alpha), β (beta), γ (gamma), etc.

The general equation is given as

$$\text{Val} - \text{accuracy} = \alpha - \frac{\beta L^2}{N} + \gamma \log(E) \quad (9)$$

Where

- α = theoretical max accuracy (dataset-architecture upper bound)
- β = penalty for unfreezing layers (controls overfitting or instability)
- L = number of unfrozen layers
- N = total number of layers in the network (e.g., 152 for ResNet152)
- $\gamma \log(E)$ = gain from training epochs, with diminishing returns

This equation captures three important trends about fine-tuning processes

1. Performance improves with increasingly logarithmically time (increasing number of epochs)
2. Unfreezing too many layers hurts performance especially when processing limited and small sized dataset. This is reflected by the term $-\frac{L^2}{N}$
3. **Accuracy is capped** by an architecture- and data-dependent ceiling α

The equation assumes that:

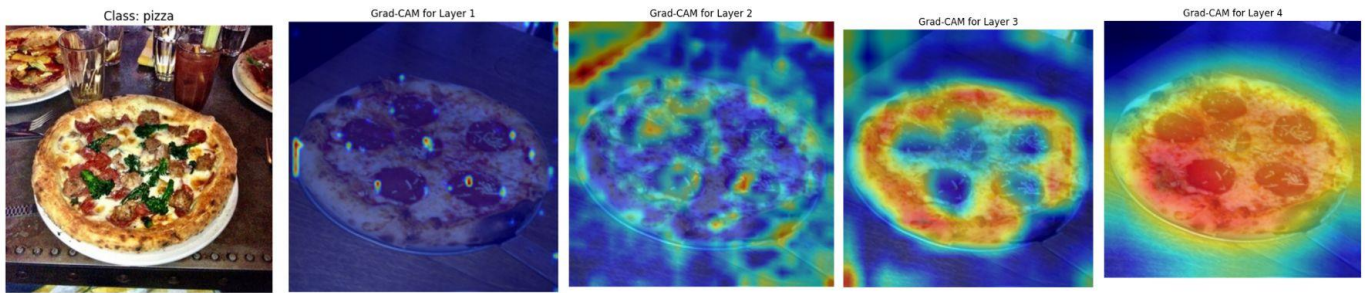
1. You're using transfer learning (pretrained models)
2. You're fine-tuning on a dataset with limited size
3. You want to control training time and overfitting

This formula can be extended by looking at the dataset complexity factor or the number of training samples S and the number of model parameters P

$$\text{Val} - \text{accuracy} = \alpha - \frac{\beta L^2}{N} + \gamma \log(E) + \delta \log(S) + \rho \log(P) \quad (9)$$

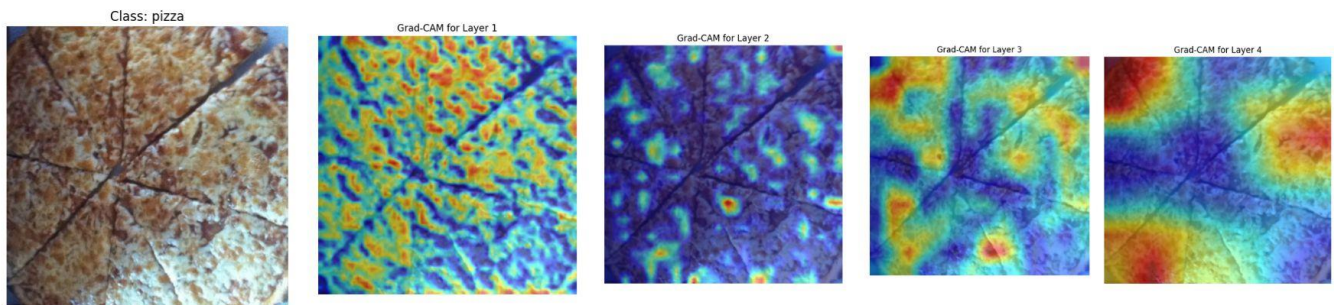
5.3. Grad-CAM Results and Interpretations

Figure 13 and 14 shows the Grad-CAM result for the EfficientNetB0 trained on 10% of the Food-101 database. It shows how Grad-CAM progressively focuses its attention on the relevant part of the images.



The Figure above shows the Grad-CAM visualization for the class pizza at the left hand-side

Figure 15 shows the Grad-CAM visualization learning from layer1 to layer 4



The Figure above shows the Grad-CAM visualization for the class pizza at the left hand-side

Figure 16 shows the Grad-CAM visualization learning from layer1 to layer 4

- i. *The pizza image*: The image on the far left displays an original sample from the Food-101 dataset. The image captures the pizza and its surrounding items.
- ii. *Layer 1*: At this level, the model output a broad and diffused image. The activation is minimal and scattered, with only small patches, showing random spots across the pizza. This reveals that the layer 1 of the EfficientNetB0 model captures the edges, textures and colors.
- iii. *Layer 2*: In this layer, the network concentrates more with the region of the pizza, but still captures the surroundings. This shows that the model could differentiate between the pizza and the background using more complex features such as shapes or small patterns.
- iv. *Layer 3*: At this level, the model focuses on the pizza. High activations are now clearly visible across the center of the pizza, ignoring most of the background. This indicates that the layer is beginning to recognize mid-level features relevant to the pizza class, such as crust edges or topping distribution.
- v. *Layer 4*: This layer intensely focusses on the pizza, especially its key components like toppings. The background is almost entirely ignored. This indicates that the model captures high-level, task-relevant features of the pizza at this level. These features are what the model use to make its final predictions

From the Grad-CAM image output, the model learns the features of the images at stages three and four. To obtain higher accuracy without overfitting, the deeper layers (those closer to the output, specifically Layer 3 and Layer 4) should be unfrozen as these contain mid-level and high-level class-specific features.

EfficientNetB0 has seven blocks (block1a, block2a,..., block7a) and the deeper blocks are the later blocks (6a, 6b, ..., 7a) [8]. The table below shows the number of blocks and the corresponding layer range

Table 8 shows the blocks and the corresponding number of layers of the EfficientNetB0

Block Name	Count of layers	Layer index range
Block1a	14 layers	2 – 15
Block 2a	16 layers	16 – 31
Block 2b	14 layers	32 – 45
Block 3a	17 layers	46 - 62
Block 3b	14 layers	63 – 76
Block 4a	16 layers	77 – 92
Block 4b	14 layers	93 – 106
Block 5a	16 layers	107 – 122
Block 5b	14 layers	123 – 136
Block 5c	14 layers	137 – 150
Block 6a	17 layers	151 – 167
Block 6b	16 layers	168 – 183
Block 6c	16 layers	184 – 199
Block 6d	16 layers	200 – 215
Block 7a	21 layers	217 – 236

How many layers fall within the block 6a, 6b and 7a? From the Table, we have 17 layers + 16 layers + 21 layers making a total of 54 layers. This shows that the 54 layers closest to the output should be unfrozen, which is approximately 23% of the 237 layers

To effectively unfreeze only 6a, 6b, and 7a blocks, the Python code below print out the exact layers and confirm the count. Then, the number of layers indicated can then be unfrozen to obtain the optimal accuracy.

```
from tensorflow.keras.applications import EfficientNetB0
```

```
model = EfficientNetB0(weights='imagenet',
include_top=False)
```

```
for i, layer in enumerate(model.layers):
```

```
    if 'block6a' in layer.name or 'block6b' in layer.name or
'block7a' in layer.name:
```

```
        print(f"{i}: {layer.name}")
```

Note that blocks 6c and 6d are not unfrozen. The reason is that deeper layers like Layer 3 and Layer 4 focused specifically on the pizza, especially in Layer 4, indicating high-level, class-specific feature detection, which happens at blocks 6a, 6b, and 7a. Fine-tuning more layers will increase training time, risk overfitting, and require more resource usage. Since Grad-CAM didn't show significant activation in 6c and 6d, unfreezing them may add unnecessary complexity, and may not have any impact on performance. Researchers should unfreeze blocks 6c and 6d when the Grad-CAM shows significant, task-specific activations in those layers, and when there is a large dataset. Note that this experiment only uses 10% of the dataset to reduce computational resources and save time. As a rule, when working with EfficientNetB0 with a small dataset, unfreeze the blocks 6a, 6b, and 7a; however for larger datasets, gradually unfreeze the 6a, 6b, 6c, 6d, and 7a blocks. For each block unfrozen, run the experiment to determine the validation accuracy.

5.4. Comparing the results in the Experimental and Grad-CAM approach

In the experiment involving ResNet152, unfreezing approximately the top 107 layers, which is approximately 21% of the 514 layers gives the optimal accuracy. In the same vein, in the EfficientNetB0 experiment, unfreezing the top 22% of the 237 layers gives optimal validation accuracy. However, a different result was obtained in the VGG 16 network, which requires the entire layers to be unfrozen to obtain an optimal validation result. The difference in the VGG 16 architecture is that it is an extremely lightweight architecture that contains only 18 layers.

The Grad-CAM experiment gives similar results to those of experimental results. Findings from the Grad-CAM approach show that approximately 23% of the EfficientNetB0 gives the optimal result. This is the same as the result obtained in EfficientNetB0 using an experimental approach (approximately 22%). Besides, the result obtained through

this study perfectly aligns with previous findings [9], [13], [14], [15], [30], etc. which established that fine-tuning a few top layers increases accuracy, however, unfreezing too many layers affects the accuracy.

6. Conclusion

This study investigated the optimal layers to unfreeze in a CNN network to obtain the best validation accuracy. The first part of the study conducted a comprehensive literature review to identify existing gaps and found that no explicit research has been done to determine the optimal number of layers to unfreeze in a CNN network for achieving the best validation accuracy. A few research established that unfreezing the few top layers of a CNN network is likely to increase accuracy. Research also established that unfreezing too many layers negatively affects validation accuracy, resulting in overfitting, but much has not been done to establish the range of optimal layers in a CNN network.

The second part of the study considers CNN architecture and expressly explains the fine-tuning processes. The methodology section explains the experimental processes and the Grad-CAM approach used to determine the optimal number of layers. The result sections derive regressive equations through the experimental approach and determine the MAE and MSE metrics to juxtapose the accuracy of the regressive equations. A general equation was deduced. This equation considers further hyperparameters that affect the optimal number of layers in a CNN network.

The results obtained through the experimental and Grad-CAM approaches closely align. The results show that fine-tuning the top one-fifth of pre-trained layers (between 20% and 25%) gives the optimal accuracy. However, VGG16 requires the entire layers to be unfrozen to obtain optimal solution. The reason is because it has only 18 layers. The outcome from both approaches also supports previous studies which implies that unfreezing a few top layers improves validation accuracy. This study expressly concludes that fine-tuning the top one-fourth layers of a pre-trained model will give an approximate training accuracy and validation accuracy. At these layers, the loss will also be minimal.

6.1. Significance of the research

It's been quite challenging to obtain the approximate number of layers to unfreeze to obtain an optimal solution. Much literature has proposed a general rule of thumb but could not specify in particular, the fine-tuning depth. This research is highly significant as it provides an answer to a long-awaited research gap. Besides, researchers, AI and ML experts, data analysts, and even computer science students, who work with transfer learning do not need to systematically unfreeze 514 layers in the case of ResNet152 to obtain a fairly accurate solution as the approach is computationally expensive and time-consuming.

6.2. Limitations

Even though this study deduces an equation for each CNN network experimented and a general equation to determine the number of layers to unfreeze for most CNN networks, the general equations do not consider all hyperparameters that might affect the number of layers to unfreeze to obtain the optimal solution. Besides, the quantity and quality of the data can change everything. For example, when the ResNet152 model is trained on a small, simple dataset like CIFAR-10, the validation accuracy is quite small. This isn't the same result when ResNet152 is trained with a large dataset such as Food-101. You'll get a different result than if you use a larger or more complex dataset. Parameters like how balanced the classes are, how many images you have, and how clear the labels are all affect how many layers you should unfreeze.

6.3. Future Research

The experimental method was used to obtain the range of layers to fine-tune to obtain optimal solutions. However, three CNN networks were only considered (ResNet152, VGG16, and EfficientNetB0). Further research can consider more CNN networks to corroborate the findings obtained in this study. Moreover, the Grad-CAM approach was only done on the EfficientNetB0 network. There is a need to investigate if the Grad-CAM approach would give similar findings in other architecture. Lastly, a more robust study can be done in the future to deduce general equations that consider all hyperparameters in a CNN network.

Author's Statement

The author declares that there is no conflict of interest related to this study and no financial support, sponsorship, or funding was received from any organization or institution for the conduct of this study. The author affirms that the research was conducted independently without influence from any third parties.

Acknowledgement – The author conducted the research independently, however, the author acknowledges all researchers who have conducted previous research on fine-tuning pre-trained models as their works contributed immensely to the success of this study. The author is most grateful for the valuable comments and suggestions provided by the reviewers, which helped improve the quality of the manuscript.

Funding Sources – No funding is received from any public, commercial, or not-for-profit agencies.

Author's Contribution – The author solely conceived the study, conducted the literature review, identified the gaps and research questions, investigated the approach to answer the research questions, collected and analyzed the data, and wrote the manuscript. The author also reviewed and approved the final version of the manuscript.

Conflict of Interest – The author declares that there is no conflict of interest regarding the publication of this research.

Data Availability – The datasets used in this study (Food-101, CIFAR-10, and the Stanford Dogs dataset) are readily available online and can be accessed from their respective official repositories. The Food-101 datasets can be accessed from https://www.vision.ee.ethz.ch/datasets_extra/food-101/, while the CIFAR-10 is available through the TensorFlow/Keras datasets module or at <https://www.cs.toronto.edu/~kriz/cifar.html>. The Stanford Dogs can be accessed at <http://vision.stanford.edu/aditya86/ImageNetDogs/>

The pre-trained architectures are meant to use data in a specific image resolution. For instance, EfficientNetB0 processes images in 224 by 224 resolution. However, some of the Food-101 datasets have higher resolution, resulting in distortions during training. This may affect the pre-trained models from capturing high-resolution features of certain datasets. Besides, 10% of the Food-101 dataset was used. This is because higher computational resources will be required to run the experiments on the full Food-101 dataset, which has 101,000 images.

References

- [1] I. Kandel, M. Castelli, "How Deeply To Fine-Tune A Convolutional Neural Network: A Case Study Using A Histopathology Dataset," *Appl. Sci.*, Vol.10, No.10, pp. 3359, 2020. <https://doi.org/10.3390/app10103359>
- [2] Maszytiah et al., "A Comprehensive Performance Analysis Of Transfer Learning Optimization In Visual Field Defect Classification," *Diagnostics*, Vol.12, No.5, pp.1258, 2022, <http://dx.doi.org/10.3390/diagnostics12051258>
- [3] S. J. Pan, Q. Yang, "A Survey On Transfer Learning," *IEEE Trans. Knowl. Data Eng.*, Vol. 22, No.10, pp. 1345–1359, 2009.
- [4] B. Q. Huynh, H. Li, M. L. Giger, "Digital Mammographic Tumor Classification Using Transfer Learning From Deep Convolutional Neural Networks," *J. Med. Imaging*, Vol.3, No.3, pp.034501, 2016.
- [5] D. S. Kermany et al., "Identifying Medical Diagnoses And Treatable Diseases By Image-Based Deep Learning," *Cell*, Vol.172, No.5, pp.1122–1131, 2018.
- [6] A. Hosna, E. Merry, J. Gyalmo et al., "Transfer Learning: A Friendly Introduction," *J. Big Data*, Vol.9, pp.102, 2022, <https://doi.org/10.1186/s40537-022-00652-w>
- [7] M. E. Taylor, P. Stone, "Transfer Learning For Reinforcement Learning Domains: A Survey," *J. Mach. Learn. Res.*, Vol.10, pp.1633–1685, 2009.
- [8] A. Karlsson, I. Runelöv, "The Effects Of Fine-Tuning Depth On A Pre-Trained Alexnet Architecture, Applied To Chest X-Rays," *KTH Royal Institute Of Technology, School Of Electrical Engineering and Computer Science*, 2021.
- [9] A. Gupta, M. Gupta, "Transfer Learning For Small And Different Datasets: Fine-Tuning A Pre-Trained Model Affects Performance," *J. Emerg. Investig.*, 2020,
- [10] J. S. Kumar, S. Anuar, N. H. Hassan, "Transfer Learning-Based Performance Comparison Of The Pre-Trained Deep Neural Networks," *Int. J. Adv. Comput. Sci. Appl.*, Vol. 13, No.1, 2022.
- [11] N. Houlsby et al., "Parameter-Efficient Transfer Learning For NLP," 2020.
- [12] G. Vrbanić and V. Podgorelec, "Transfer Learning With Adaptive Fine-Tuning," *IEEE Access*, Vol.8, pp.196217–196230, 2020, <http://dx.doi.org/10.1109/ACCESS.2020.3034343>
- [13] K. S. Lee, J. Y. Kim, E. T. Jeon, W. S. Choi, N. H. Kim, K. Y. Lee, "Evaluation of Scalability and Degree of Fine-Tuning of Deep Convolutional Neural Networks for COVID-19 Screening on Chest X-ray Images Using Explainable Deep-Learning Algorithm," *J Pers Med*, Vol.7, No.4, pp.213–225, 2020, doi:

- 10.3390/jpm10040213.
- [14] V. Taormina, D. Cascio, L. Abbene, G. Raso, "Performance of Fine-Tuning Convolutional Neural Networks For Hep-2 Image Classification," *Appl. Sci.*, **10**, **6940**, **2020**, <https://doi.org/10.3390/App10196940>
 - [15] M. Amiri, R. Brooks, H. Rivaz, "Fine Tuning U-Net for Ultrasound Image Segmentation: Which Layers," *Electrical Engineering and Systems Science*, **2020**, <https://doi.org/10.48550/Arxiv.2002.08438>
 - [16] M. Kardinal, "Fine Tuning in the Brain," *Plos Computational Biology*, *Bernstein Coordination Site*, **2015**.
 - [17] F. C. Bartlett, "Remembering: a Study in Experimental and Social Psychology," *1st Ed. Cambridge, U.K, Cambridge Univ. Press*, **1932**.
 - [18] S. Carey, "Conceptual Change in Childhood," *1st Ed. Cambridge, Ma: Mit Press*, **1991**.
 - [19] D. O. Hebb, "The Organization of Behavior: A Neuropsychological Theory," *1st Ed. New York, Ny: Wiley*, **1949**.
 - [20] G. Hatano, K. Inagaki, "Two Courses of Expertise. Cognition and Instruction," *Vol.3, No.4*, pp.**271-291**, **1986**.
 - [21] R. Dhiman, G. Joshi, R. K. Challa, "A Deep Learning Approach for Indian Sign Language Gestures Classification With Different Backgrounds," *In Proc. J. Phys.: Conf. Ser.*, *Vol. 1950*, pp.**012020**, **2021**, <http://dx.doi.org/10.1088/1742-6596/1950/1/012020>
 - [22] A. Krizhevsky, I. Sutskever, G. E. Hinton, "Imagenet Classification with Deep Convolutional Neural Networks," *In Adv. Neural Inf. Process. Syst.*, *Vol.25*, pp.**1097-1105**, **2012**.
 - [23] R. Norhikmah, A. Lutfhi, C. Rumini, "The Effect of Layer Batch Normalization and Dropout on CNN Model Performance for Facial Expression Classification," *International Journal on Informatics Visualization*, **2022**, <https://doi.org/10.30630/Joiv.6.2-2.921>
 - [24] R. Yamashita, M. Nishio, R. Do, K. Togashi, "Convolutional Neural Networks: An Overview And Application In Radiology," *Insights Imaging*, *Vol.9*, pp.**611-629**, **2018**, <https://doi.org/10.1007/s13244-018-0639-9>
 - [25] C. Szegedy E. T. Al., "Going Deeper with Convolutions," *In Proc. Ieee Conf. Comput. Vis. Pattern Recognit. (Cvpr)*, pp. **1-9**, **2015**.
 - [26] B. A. Alejandro, N. Díaz-Rodríguez, J. Del Ser, "Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities, And Challenges Toward Responsible AI," *Information Fusion*, *Vol. 58*, No. **82**, **2020**.
 - [27] C. Van-Zyl, X. Ye, R. Naidoo, "Harnessing Explainable Artificial Intelligence for Feature Selection in Time Series Energy Forecasting: A Comparative Analysis of Grad-Cam and Shap," *Applied Energy*, *Vol.353*, **122079**, **2024**, <https://doi.org/10.1016/j.apenergy.2023.122079>
 - [28] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, D. Batra, "Grad-CAM: Visual Explanations From Deep Networks Via Gradient-Based Localization," *IEEE International Conference on Computer Vision*, pp.**618-626**, **2017**, [10.1109/Iccv.2017.74](https://doi.org/10.1109/Iccv.2017.74)
 - [29] A. Davila, J. Colan, Y. Hasegawa, "Comparison of Fine-Tuning Strategies for Transfer Learning in Medical Image Classification. Image and Vision Computing," **146**, **105012**, **2024**.
 - [30] T. M. Adepoju, M. O. Oladele, M. O. Akintunde, A. M. Ogunleye, "Effect of Fine-Tuning Transfer Learning Layers: A Case Study of Breast Cancer Classification," *Fuoye Journal of Engineering and Technology*, *Vol.9, No.4*, pp.**660-668**, **2024**.
 - [31] X. Zhang, Y. Wang, N. Zhang, D. Dong, B. Chen, "Research on Scene Classification Method of High-Resolution Remote Sensing Images Based On RFONET," *Appl. Sci.*, *Vol.9, No.10*, pp.**2028**, **2019**, <http://dx.doi.org/10.3390/app9102028>
 - [32] D. Scott, "Cognitive Flexibility," *In Theories of Learning and Instruction*, *1st Ed., New York, NY: Springer*, pp.**151-182**, **1993**.
 - [33] B. B. Folajinmi, E. M. Eronu, S. J. Fanifosi, "Using Hybrid Convolutional Neural Network and Random Forest (CNNRF) Algorithms to Address Non-Technical Losses in Power Distribution," *World Academics Journal of Engineering Sciences*, *Vol.12, Issue.1*, pp.**01-08**, **2025**
 - [34] M. Deepanshu, K. Srinivas, A. Charan Kumari, "Convolutional Neural Network-Based Automated Acute Lymphoblastic Leukaemia Detection and Stage Classification from Peripheral Blood," *International Journal of Scientific Research in Computer*

Science and Engineering, Vol.12, Issue.3, pp.21-28, 2024

AUTHORS PROFILE

My name is Rotimi Philip Adebayo (R. P. Adebayo). I hold a Bachelor's degree in Computer Science and Mathematics (2008), a Master's degree in Computer Science (2013), and a second Master's in Operations Research (2017). I am currently pursuing a Master's degree in Artificial Intelligence and will begin a PhD program in September 2025 at the University of Portsmouth, UK.



With over a decade of teaching experience, I specialize in preparing students for standardized exams such as the GMAT, GRE, SAT, IGCSE, and A-Level Mathematics. Since 2018, I have also been working as a Data Analyst with Bluesilvers Consulting. I review articles, reports and literature for clients and organizations.

In addition to my teaching and industry experience, I actively contribute to academic research. I have published four papers in top U.S. journals, and one of my recent works—Explainable Artificial Intelligence (XAI) in Management and Entrepreneurship: Exploring the Applications and Implications—is currently under review. My project portfolio includes machine learning applications for cancer diagnosis and housing price prediction using Random Forest models, among others.