# COMPARATIVE ANALYSIS OF OPTIMIZERS USING MOBILENETV2 FOR BIRDS CLASSIFICATION

Project Submitted to the
SRM University AP, Andhra Pradesh
for the partial fulfillment of the requirements to award the degree of

**Master Of Technology in**
**Computer Science & Engineering**
**School of Engineering & Sciences**

submitted by

**Sandeep Reddy Panyala (AP24122060005)**
**Sri Harsha Vardhan Gogisetty (AP24122060007)**
**Maddi Eswar (AP24122060014)**

Under the Guidance of

**Dr. Kshira Sagar Sahoo**


**Department of Computer Science & Engineering**
SRM University-AP
Neerukonda, Mangalgiri, Guntur
Andhra Pradesh - 522 240
May2025

# ACKNOWLEDGMENT

I wish to record my indebtedness and thankfulness to all who helped me to prepare this Project Report titled Comparative Analysis Of Optimizers Using MobileNetV2 For Birds Classification and present it satisfactorily. I am especially thankful for my guide and supervisor Dr. Kshira Sagar Sahoo in the Department of Computer Science & Engineering for giving me valuable suggestions and critical inputs in the preparation of this report. My friends in my class have always been helpful and I am grateful to them for patiently listening to my presentations on my work related to the Project.

Sandeep Reddy, Sri Harsha Vardhan , Maddi Eswar
(Reg. No. AP24122060005, AP24122060007, AP24122060014)
M. TECH - DATA SCIENCE Department of Computer Science & Engineering SRM University-AP

# TABLE OF CONTENTS

## Abstract

The classification of bird species through automated systems has emerged as a vital tool in biodiversity conservation, ecological research, and environmental monitoring. Manual identification of birds from images is a time-intensive task that requires expert knowledge, making automation an attractive and practical alternative. In this project, we present a deep learning-based approach to the classification of bird species using transfer learning. Specifically, we utilize the MobileNetV2 architecture pre-trained on ImageNet and fine-tune it on a custom dataset containing 25 bird species. The limited size and imbalance of the dataset present challenges such as overfitting and insufficient generalization; these were mitigated through rigorous data augmentation strategies, including rotation, zooming, shearing, and horizontal flipping. Various optimization algorithms like Stochastic Gradient Descent (SGD), SGD with momentum, Adam, and Nadam were employed and compared in terms of their performance on training and validation accuracy.

The model training was carried out using TensorFlow and Keras frameworks, leveraging GPU acceleration for efficient computation. Through visual analysis of training/validation metrics, we identified the optimizer and parameter configurations that delivered the best classification performance. The project demonstrates how transfer learning with lightweight networks like MobileNetV2 can be an effective solution for fine-grained visual classification tasks in domains with limited labeled data.

## Introduction

Birds are vital indicators of ecological health and play diverse roles in maintaining ecosystem functionality. As pollinators, they contribute to plant reproduction; as seed dispersers, they promote forest regeneration and biodiversity; and as bioindicators, their presence, absence, or behavior can reflect environmental changes such as pollution, habitat loss, and climate shifts. Given their ecological importance, the ability to accurately monitor and classify bird species is essential not only for biodiversity conservation but also for guiding policies on land use, wildlife protection, and ecological restoration.

Traditionally, this process has depended on the expertise of ornithologists and experienced birdwatchers who rely on visual and auditory cues—such as plumage patterns, flight behavior, and bird calls—for identification. While effective, this manual identification is both time-consuming and labor-intensive, making it infeasible for large-scale monitoring or real-time applications, especially in remote or biodiverse regions where hundreds of species may coexist.

In response to this challenge, computer vision—particularly Convolutional Neural Networks (CNNs)—has emerged as a transformative tool for automating image-based classification tasks. CNNs have demonstrated impressive capabilities in learning hierarchical visual features, making them ideal for complex recognition problems. They have powered breakthroughs in fields ranging from medical imaging (e.g., tumor detection) to security systems (e.g., facial recognition) and wildlife tracking (e.g., species detection in camera trap images). However, applying CNNs to bird classification introduces domain-specific hurdles:

- Data Scarcity: High-quality, labeled datasets for bird species are often limited, especially for rare or region-specific species.
- Overfitting: With small datasets, CNNs may memorize training examples instead of learning general features.
- High Computational Demand: Deep networks require extensive GPU resources and long training times, which are often not accessible in conservation settings.

To mitigate these limitations, this project adopts transfer learning, a technique where knowledge gained from large-scale, pre-trained models (like those trained on ImageNet) is reused for related, smaller tasks. Transfer learning not only accelerates training but also improves generalization by leveraging features like edge detection, color gradients, and textures that are common across natural images.

We selected MobileNetV2—a modern, lightweight CNN architecture known for its efficiency and speed—as our base model. Designed specifically for mobile and embedded vision systems, MobileNetV2 balances performance with low computational overhead, making it suitable for deployment in real-world conservation tools, such as field monitoring devices, mobile apps, or automated camera traps.

To further enhance model generalization and combat overfitting, we applied aggressive data augmentation, introducing randomness in image orientation, scale, and positioning. This simulates real-world scenarios where birds appear under varying lighting conditions, backgrounds, or poses. Beyond the model architecture, we conducted a comparative evaluation of optimizers—the algorithms that guide how the model learns during training. Specifically, we assessed:

- SGD (Stochastic Gradient Descent): A baseline optimizer known for its simplicity but slower convergence.
- SGD with Momentum: An enhancement that helps navigate the loss landscape more efficiently.
- Adam (Adaptive Moment Estimation): A widely-used optimizer that adjusts learning rates dynamically and has shown strong empirical results.
- Nadam (Nesterov-accelerated Adam): A hybrid that incorporates the benefits of Adam with Nesterov momentum for more responsive updates.

This project demonstrates efficient bird species classification using transfer learning with MobileNetV2 and evaluates the impact of four optimizers on model performance. The resulting lightweight model is scalable and suitable for real-world ecological and conservation applications.

## Major Contributions

### 1. Dataset Preparation and Preprocessing

The dataset used in this project comprises high-resolution images from 25 distinct bird species, each representing a unique class in the classification task. These images were carefully organized into labeled directories, where each folder corresponds to a specific bird species. This structure was crucial for enabling supervised learning and facilitating automated image labeling during training.

Given the relatively limited size of the dataset, extensive data augmentation techniques were employed to artificially increase the size and variability of the training data. This not only mitigates the risks of overfitting but also improves the model's ability to generalize across unseen data. Using Keras's ImageDataGenerator, we applied a series of transformations including:

- Rotation: Random rotations to simulate birds captured at different angles.
- Width and Height Shifts: Slight translations to mimic changes in framing.
- Shearing: Applied shearing transformations to simulate natural image distortions.
- Zooming: Random zoom to emulate different distances from the subject.
- Horizontal Flipping: To generalize across different orientations.
- Resizing Pixel Values: Normalizing the pixel values to the [0, 1] range for stable training.

These preprocessing steps significantly enhanced the dataset's diversity and enabled the model to learn more robust features resilient to lighting changes, orientation, scale, and positioning.

### 2. Model Architecture and Transfer Learning

To address the limitations of training a deep model from scratch, we adopted a transfer learning approach using MobileNetV2—a highly efficient and lightweight convolutional neural network pre-trained on the ImageNet dataset. MobileNetV2 was selected due to its architectural efficiency and proven performance in resource-constrained environments, making it suitable for potential deployment on mobile or edge devices.

The top classification layer of MobileNetV2 was removed, and a custom classifier was appended to tailor the model to the bird classification task. The customized head consisted of:

- A GlobalAveragePooling2D layer to reduce spatial dimensions and flatten the feature maps.
- A Dropout layer to prevent overfitting by randomly deactivating neurons during training.
- A Dense output layer with 25 neurons (equal to the number of bird species), using softmax activation to produce class probabilities for multi-class classification.

The model was compiled using the categorical cross-entropy loss function, appropriate for multi-class problems, and various optimizers were tested to evaluate their effect on model performance.

**3. Optimizer Evaluation**

A critical part of the study involved evaluating the influence of different optimization algorithms on training dynamics and final accuracy. The following four optimizers were implemented and compared:

- **SGD (Stochastic Gradient Descent):** A basic optimizer that updates weights using the gradient of the loss function with respect to a mini-batch of training data. Though simple, its performance is highly sensitive to learning rate.
- **SGD with Momentum**: An improved version of SGD that incorporates momentum to accelerate convergence and reduce oscillations by considering the previous gradients' direction.
- **Adam (Adaptive Moment Estimation):** A popular optimizer that combines the benefits of momentum and adaptive learning rates by maintaining separate moving averages of the gradient and its square.
- **Nadam (Nesterov-accelerated Adaptive Moment Estimation):** A variant of Adam that integrates Nesterov momentum, allowing the optimizer to look ahead at future gradients, often leading to smoother and more responsive updates.

Each optimizer was used to train the model for 10 epochs, and both training and validation accuracy and loss were tracked to assess the optimizer's influence on convergence speed, stability, and generalization.

**4. Training Monitoring and Visualization**

To analyze model behavior during training, performance metrics were continuously recorded and visualized using Matplotlib. This included:

- Plotting training and validation accuracy curves to observe how quickly the model learns and how well it generalizes.
- Plotting training and validation loss curves to identify issues such as underfitting, overfitting, or slow convergence.

These visualizations were crucial for diagnosing training challenges and comparing the effectiveness of each optimizer in guiding the model toward an optimal solution. The graphical insights helped validate the effectiveness of the transfer learning pipeline, data augmentation strategy, and optimizer selection in building a robust bird species classification model.

## Literature Survey

1. **Howard, A. G. et al. (2017) – MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications**

   This paper introduced the MobileNet architecture, which revolutionized lightweight deep learning by using depthwise separable convolutions to drastically reduce the number of parameters and computational cost without significantly compromising accuracy. It became a foundational architecture for deploying deep learning models on mobile and embedded devices. The use of MobileNet in this project is rooted in its efficiency and suitability for constrained computational environments, making it a robust backbone for bird species classification.

   arXiv:1704.04861

2. **Sandler, M. et al. (2018) – MobileNetV2: Inverted Residuals and Linear Bottlenecks**

   Building on the original MobileNet, MobileNetV2 incorporates new architectural features such as inverted residual blocks and linear bottlenecks that further enhance model performance while preserving efficiency. These innovations enable better gradient flow and reduced memory usage, which are crucial for training deep networks on limited hardware. The project uses MobileNetV2 as its core feature extractor, benefiting from its improved accuracy and speed, which are essential for high-performance classification of bird species.

   arXiv:1801.04381

3. **He, K. et al. (2016) – Deep Residual Learning for Image Recognition**

   This landmark work proposed the ResNet architecture, introducing residual connections that allow very deep neural networks to be trained effectively by mitigating the vanishing gradient problem. While ResNet is not directly used in this project, its conceptual innovations have deeply influenced the design of modern CNN architectures, including MobileNetV2. The idea of allowing gradients to pass more easily through the network has helped shape how deep learning models are structured for tasks such as fine-grained classification. arXiv:1512.03385

4. **Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012) – ImageNet Classification with Deep Convolutional Neural Networks**

   This groundbreaking paper introduced AlexNet, which was a turning point in the field of computer vision. By winning the 2012 ImageNet competition by a significant margin, it demonstrated the power of deep CNNs trained on large datasets. Techniques like ReLU activation and dropout regularization were popularized through this work. The pre-trained ImageNet weights used in MobileNetV2 in this project are a direct inheritance of the progress sparked by AlexNet.

5. **Zhou, B. et al. (2017) – Fine-Grained Image Classification by Exploring Bipartite-Graph Labels**

   This paper tackles the complex task of fine-grained image classification by using graph-based label structures to better capture inter-class relationships. Such techniques are especially useful in domains like bird species identification where subtle visual differences define class boundaries. Although this exact approach is not implemented in the project, the concept aligns with the challenges addressed—fine-grained bird classification using limited visual cues—thus emphasizing the importance of advanced representation learning techniques.

   arXiv:1702.08912

6. **Wah, C. et al. (2011) – The Caltech-UCSD Birds-200-2011 Dataset**

   The Caltech-UCSD Birds-200-2011 dataset is a standard benchmark in the domain of fine-grained image classification and has been used extensively in bird classification research. It includes over 11,000 images across 200 bird species, annotated with part locations and bounding boxes. This dataset has influenced many projects, including this one, by setting a standard for dataset structure and complexity in bird classification tasks. While this project uses a different dataset, it follows similar principles in terms of label granularity and class diversity.

7. **Kumar, N. et al. (2012) – Leafsnap: A Computer Vision System for Automatic Plant Species Identification**

   This paper presents Leafsnap, an automated system for identifying plant species based on leaf images. It leverages computer vision and machine learning techniques to tackle the challenge of fine-grained classification, a problem domain that closely parallels bird

species classification. Both applications require the system to distinguish between classes with subtle visual differences, often under varying lighting and background conditions. The methodologies and challenges discussed in this work provide valuable context and validation for applying CNN-based models to similarly structured problems in avian classification.

8. **Redmon, J., & Farhadi, A. (2018) – YOLOv3: An Incremental Improvement**
   YOLOv3 introduced a powerful and efficient approach to real-time object detection by predicting bounding boxes and class probabilities directly from full images in a single evaluation. Although YOLO focuses on object detection rather than classification, its core architectural strategies—such as multi-scale detection and residual connections—inspire efficient deep learning models, including those used in real-time bird species classification. The emphasis on speed and accuracy in YOLOv3 resonates with the goals of lightweight models like MobileNetV2 used in this project.
9. **Esteva, A. et al. (2017) – Dermatologist-level classification of skin cancer with deep neural networks.**
   This influential study highlights the ability of deep convolutional neural networks to match expert-level performance in fine-grained medical image classification. The use of transfer learning and large-scale image datasets demonstrated that CNNs can be powerful tools in domains where distinguishing between classes relies on nuanced visual cues. This directly supports the motivation for using CNNs in bird classification, where species may differ only by fine feather patterns or color tones, much like skin lesions differ subtly in medical imaging.
10. **Simonyan, K., & Zisserman, A. (2015) – Very Deep Convolutional Networks for Large-Scale Image Recognition.**
    VGGNet made a significant impact in the deep learning community by showing that very deep networks with small convolutional filters could achieve excellent performance in image classification tasks. Though more computationally expensive than MobileNet, VGG's design has been foundational in developing transfer learning strategies, often being used as a baseline for comparison. Its success in general image classification established the groundwork for using pre-trained models in fine-grained tasks like bird species recognition.

11. **Szegedy, C. et al. (2016) – Rethinking the Inception Architecture for Computer Vision.**

    This paper refined the Inception architecture to balance performance and computational cost, introducing ideas like factorized convolutions and residual connections. Inception-v3 and later versions became highly effective at both broad and fine-grained classification tasks. The innovations from this architecture contributed to the growing ecosystem of efficient CNNs and inspired the design of later models, including MobileNet variants. For this project, such architectures underscore the feasibility and efficiency of deploying deep networks in practical bird classification applications.

## Methodology

### DataSet Description

The dataset consists of 25 types of bird class images. The dataset is initially of two folders, one is train, in which each class in the folder has 600 images which we would be using for training and the other folder is valid which has 25 classes and with images of 300 utilized for the validation part. The image sizes are in different dimensions, not in a fixed specified format compatible with the model.

### Data Preprocessing

Rotation involves randomly rotating images within a specified range of degrees. This helps the model become invariant to orientation, so it can correctly classify objects regardless of how they are rotated in real-world scenarios. Width and height shift translate the image horizontally (width shift) or vertically (height shift) by a fraction of the total width or height. This makes the model robust to the position of objects within the frame, teaching it to focus on the object itself rather than its exact location.

Shearing applies a geometric transformation that slants the shape of the image, effectively "tilting" the object in the image. This augmentation helps the model learn to recognize skewed or perspective-distorted versions of the object. Zooming randomly zooms in or out of images. Zooming helps the model to generalize better to variations in object scale and distance, allowing it to detect both close-up and distant instances of the object.

Horizontal flipping mirrors the image along the vertical axis. This is especially useful when left-right orientation does not affect the object's identity (e.g., animals or vehicles), doubling the effective dataset size and improving robustness. Rescaling pixel values involves normalizing the image pixel intensities, typically by dividing by 255 so that pixel values fall within the [0, 1] range. This normalization helps the model converge faster during training and improves numerical stability.

**MobileNetV2**

MobileNetV2 is a lightweight and efficient convolutional neural network architecture designed for mobile and embedded vision applications. Developed by researchers at Google, MobileNetV2 builds on the success of its predecessor, MobileNetV1, and focuses on delivering high accuracy with low computational cost. It achieves this by utilizing depthwise separable convolutions and introducing two key innovations: inverted residual blocks and linear bottlenecks, which together enable faster inference and reduced model size while maintaining strong performance on visual recognition tasks.

The core idea behind inverted residuals is to reverse the typical residual connection used in ResNet architectures. Instead of expanding and then compressing the feature maps, MobileNetV2 first compresses the input using a bottleneck layer, applies depthwise convolution to learn spatial features, and then expands it again before adding the residual connection. This structure not only saves on computation but also allows for efficient gradient flow, improving training speed and performance. The use of linear bottlenecks instead of nonlinear activation functions in certain parts of the network helps preserve information and avoid loss of essential data, which is particularly important in low-dimensional feature spaces.

MobileNetV2 is widely used in real-time applications such as object detection, image classification, and face recognition on mobile devices due to its balance between accuracy and efficiency. It is compatible with TensorFlow Lite, making it easy to deploy on smartphones, IoT devices, and edge platforms. Because of its modularity and compact size, it has become a go-to choice for developers aiming to integrate deep learning models into resource-constrained environments without sacrificing much accuracy.

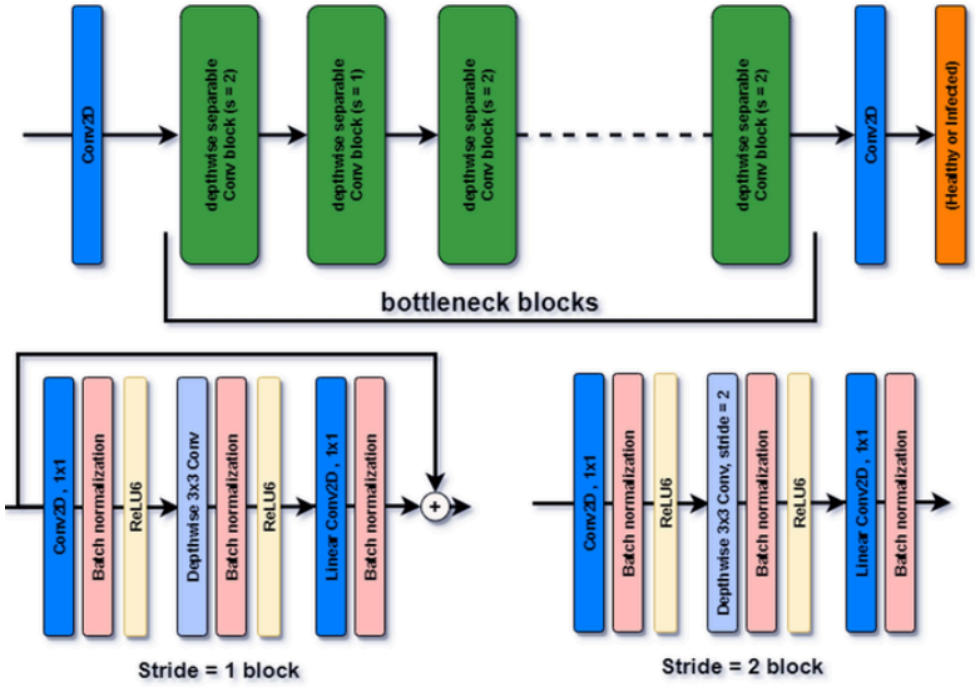



**Fig. MobileNetV2 Architecture**

## Optimizers

### 1. Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is one of the simplest and most widely used optimization algorithms in machine learning and deep learning. It updates model parameters iteratively by calculating the gradient of the loss function with respect to each parameter and moving in the opposite direction of the gradient. Unlike batch gradient descent, which uses the entire dataset, SGD updates weights using a single data point (or a mini-batch), which makes it computationally efficient and suitable for large-scale learning.

Despite its simplicity, SGD has some limitations. The updates can be noisy due to the random sampling of data, which may lead to a slower convergence or difficulty in navigating areas of the loss surface that require more stable updates. It may also get stuck in local minima or saddle points, especially in high-dimensional non-convex problems like deep neural networks.

To improve convergence, learning rate scheduling techniques like learning rate decay, step decay, or exponential decay are often applied with SGD. Although basic SGD can be effective in some scenarios, it is typically enhanced with additional strategies—such as momentum or adaptive

learning rates—to improve its stability and convergence speed, especially in deep learning applications.

**2. SGD with Momentum**

SGD with Momentum is an extension of standard SGD that helps accelerate the optimization process, especially in the presence of high curvature, noise, or saddle points. The momentum technique introduces a moving average of past gradients to smooth out the updates. This is similar to adding "inertia" to the gradient updates, allowing the optimizer to build up speed in directions with consistent gradients and reduce oscillations in noisy directions.

Mathematically, momentum updates involve a term that stores the exponentially decaying sum of previous gradients, which is then used in the parameter update step. The momentum parameter (commonly set between 0.9 and 0.99) determines how much of the past gradient contributes to the current update. As a result, the optimizer can "push through" small local minima and continue progressing towards a better solution.

The inclusion of momentum can significantly improve convergence speed and performance, particularly in problems where the loss surface contains narrow valleys or ravines. It helps reduce zig-zagging and makes learning smoother. Due to these benefits, SGD with Momentum is often preferred over plain SGD in training deep neural networks, especially for computer vision tasks like image classification.

**3. Adam (Adaptive Moment Estimation)**

Adam is a widely used optimization algorithm that combines the advantages of two other extensions of SGD—Momentum and RMSProp. It computes adaptive learning rates for each parameter by keeping track of both the first moment (mean) and second moment (uncentered variance) of the gradients. This allows Adam to adjust the step size individually for each parameter, making it particularly effective in handling sparse gradients and noisy objectives.

Adam maintains two moving averages: one for the gradients (momentum) and one for the squared gradients (adaptive scaling). These are bias-corrected during training to prevent initialization artifacts from skewing updates in the early stages. The learning rate and decay rates

for these moving averages are hyperparameters typically set to default values (learning rate = 0.001, beta1 = 0.9, beta2 = 0.999).

Due to its robustness and ease of use, Adam has become the default optimizer in many deep learning frameworks and architectures. It performs well in a wide range of tasks and typically requires less tuning of the learning rate. However, in some cases, especially in large-scale NLP or image models, researchers have noted that Adam may generalize worse than SGD with momentum, prompting careful evaluation depending on the task.

**4. Nadam (Nesterov-accelerated Adaptive Moment Estimation)**

Nadam is a variation of the Adam optimizer that incorporates Nesterov momentum, an advanced version of classical momentum. Like Adam, Nadam adapts learning rates for each parameter by using estimates of first and second moments of gradients. However, instead of applying standard momentum, it integrates Nesterov accelerated gradients into the adaptive framework to potentially achieve faster and more stable convergence.

In Nesterov momentum, the gradient is evaluated not at the current position but slightly ahead in the direction of the momentum. This "look-ahead" technique anticipates the next position, leading to more informed updates. Nadam modifies Adam's momentum term with this Nesterov correction, which often results in more responsive learning in practice, especially in models that require fine-grained convergence dynamics.

Nadam often performs better than Adam in some deep learning tasks, particularly where faster convergence or better generalization is needed. It shares Adam's strength in automatically adjusting learning rates and requires little manual tuning. However, like all optimizers, its effectiveness may vary depending on the dataset, model architecture, and task, so empirical testing is essential for best results.

**Activation Functions**

**1. ReLU (Rectified Linear Unit)**

ReLU (Rectified Linear Unit) is one of the most widely used activation functions in deep learning due to its simplicity and efficiency. It operates by outputting the input directly if it is positive; otherwise, it outputs zero. Mathematically, it is defined as: $f(x) = \max(0, x)$

Where x is the input value. This means that ReLU will return 0 for any negative input and pass positive values unchanged. This property makes ReLU computationally efficient, as it avoids expensive operations like exponentiation, which are required in other activation functions such as sigmoid or tanh.

The primary advantage of ReLU is that it helps mitigate the vanishing gradient problem, which often occurs in deep neural networks with activation functions like sigmoid or tanh. These functions suffer from gradients that shrink exponentially as the network depth increases, making it difficult to train. Since ReLU does not saturate for positive values, it allows for faster convergence during training, particularly in deep networks. However, ReLU has a drawback known as the "dying ReLU" problem, where some neurons may stop learning entirely if they always output zero (due to negative inputs), which can lead to inefficient training.
To address the dying ReLU problem, several variations of ReLU have been developed, such as Leaky ReLU (which allows a small negative slope for negative inputs) and Parametric ReLU (PReLU) (which learns the slope of negative values during training). Despite these issues, ReLU remains a go-to activation function in many neural networks because of its simplicity, effectiveness, and ease of implementation.

**2. Softmax**

Softmax is an activation function commonly used in the output layer of multi-class classification problems, where it transforms raw model outputs (also known as logits) into a probability distribution over all possible classes. The key feature of the softmax function is that it ensures the sum of the predicted probabilities is equal to 1, which is a requirement for classification problems. Softmax is defined as:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Where x_i is the input to the softmax function for class i, and the denominator is the sum of the exponentials of all input values (over all classes). This ensures that each output is a probability value between 0 and 1, representing the model's confidence in a particular class, with the highest probability corresponding to the predicted class.

Softmax is particularly useful when dealing with multi-class classification problems where the goal is to classify an input into one of several possible categories. It allows the network to make a probabilistic decision, outputting a vector where each element corresponds to the probability of the sample belonging to a given class. The class with the highest probability is typically chosen as the predicted class. Softmax is often combined with the categorical cross-entropy loss function, which quantifies the difference between the predicted probability distribution and the true labels in multi-class classification tasks.

Despite its widespread use, softmax can be computationally expensive for large datasets or many classes, as it requires calculating exponentials for each input and normalizing over the entire set of classes. To address this, optimizations such as log-sum-exp tricks are often used to stabilize the computation, especially when dealing with large values. Nevertheless, softmax remains the standard choice for classification tasks due to its clear interpretation and effectiveness in producing probabilistic outputs.

**LossFunction**

Categorical Cross-Entropy is a widely used loss function for multi-class classification problems, where each data point belongs to one of several possible classes. It measures the difference between the predicted probability distribution and the true distribution (the ground truth). In classification problems, the output is typically represented as a vector of probabilities, where each element corresponds to the probability of the sample belonging to a specific class. The true class is usually represented as a one-hot encoded vector, where all elements are 0 except for the index corresponding to the correct class, which is 1.

Mathematically, categorical cross-entropy is defined as the negative log-likelihood of the true class labels under the predicted probability distribution. For a single example, the formula is:

$$L = -\sum_{i=1}^{C} y_i \log(p_i)$$

Where C is the number of classes, yi is the true label (0 or 1), and pi is the predicted probability for class *i*. In this formula, the loss penalizes predictions that are far from the true labels, with

higher penalties for predictions that assign low probabilities to the correct class. The sum across all classes ensures that all classes are taken into account, even though the true label only corresponds to one class.

Categorical cross-entropy is most commonly used with models that output a probability distribution, such as softmax classifiers. It is effective when the classes are mutually exclusive, meaning each sample can belong to only one class. In deep learning, it is commonly used in combination with a softmax activation function in the final layer of a neural network, as softmax converts the raw outputs (logits) into a probability distribution. This loss function encourages the model to assign high probabilities to the correct class and minimize the probability assigned to incorrect classes, which makes it an essential tool for training classification models.

**Tools and Libraries**

**Numpy**

NumPy (Numerical Python) is a fundamental package for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions to operate on these arrays. At the core of NumPy is the ndarray object, which allows for efficient storage and manipulation of numerical data. Unlike regular Python lists, NumPy arrays are more compact, faster, and capable of performing complex mathematical operations using simple syntax.

One of the key advantages of NumPy is its performance. Many of its operations are implemented in C, making them significantly faster than equivalent Python code. NumPy also supports broadcasting, which allows arithmetic operations to be performed on arrays of different shapes without writing explicit loops. This makes vectorized programming possible, leading to more concise and readable code. Additionally, NumPy includes modules for linear algebra, Fourier transforms, random number generation, and integration with other scientific computing libraries.

NumPy is foundational to the scientific Python ecosystem. Libraries such as Pandas, SciPy, scikit-learn, TensorFlow, and PyTorch all rely on NumPy for underlying data operations. Its

compatibility with other tools and its ability to interface with C/C++ and Fortran code make it a powerful tool for scientific research, data analysis, and machine learning. Whether you're working with simple statistical calculations or building complex numerical models, NumPy provides the essential building blocks for efficient computation in Python.

**TensorFlow**

TensorFlow is an open-source platform developed by Google for machine learning and artificial intelligence tasks. It provides a comprehensive, flexible ecosystem of tools, libraries, and community resources that allows researchers and developers to build and deploy ML-powered applications. Originally released in 2015, TensorFlow supports a wide range of machine learning tasks, from simple linear regression models to complex deep learning architectures like convolutional neural networks (CNNs) and recurrent neural networks (RNNs). Its core is built on computational graphs that allow efficient execution of operations across CPUs, GPUs, and TPUs.

One of TensorFlow's key strengths is its scalability and versatility. Developers can build models in Python using a high-level API like Keras for ease of prototyping, while also accessing lower-level APIs for fine-tuned control and optimization. TensorFlow's data pipeline features, such as `tf.data`, allow efficient preprocessing and loading of large datasets. Furthermore, TensorFlow supports model training and evaluation in distributed environments, making it suitable for both academic research and production environments.

TensorFlow also excels in model deployment. It offers tools like TensorFlow Lite for mobile and embedded devices, TensorFlow.js for browser-based machine learning, and TensorFlow Serving for serving models in production. These capabilities make it possible to take models from research to real-world applications seamlessly. The TensorFlow ecosystem continues to grow, with active contributions from a large community of developers and researchers, ensuring its place as one of the leading frameworks in the machine learning landscape.

**Sklearn**

Scikit-learn (sklearn) is a powerful, open-source Python library used for machine learning and data mining tasks. Built on top of NumPy, SciPy, and matplotlib, it provides simple and efficient tools for data analysis and modeling. Scikit-learn includes a wide range of supervised and unsupervised learning algorithms, including classification, regression, clustering, dimensionality reduction, and model selection techniques. Its consistent and user-friendly API makes it accessible for beginners while remaining powerful enough for advanced users.

One of the standout features of Scikit-learn is its modularity and ease of use. Models can be trained, evaluated, and fine-tuned using a few lines of code, thanks to its clear and consistent structure. The library includes utilities for data preprocessing, feature selection, pipeline creation, and cross-validation, making it easy to build robust and reproducible workflows. Whether you're training a linear regression model, a decision tree, or an ensemble method like a random forest or gradient boosting, scikit-learn provides reliable implementations that are both fast and scalable.
Scikit-learn is widely used in industry and academia due to its strong community support and extensive documentation. It is ideal for prototyping and building traditional machine learning models, although it does not directly support deep learning (which is better handled by libraries like TensorFlow or PyTorch). Nonetheless, it remains a foundational tool in the data science toolkit, offering a solid framework for understanding and applying machine learning concepts in real-world applications.

**Seaborn:**

Seaborn is a popular open-source Python library for data visualization, built on top of matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn simplifies complex visualization tasks by offering easy-to-use functions for creating plots like bar charts, box plots, violin plots, scatter plots, heatmaps, and more. One of its key strengths is the ability to work seamlessly with pandas DataFrames, allowing users to plot data directly from structured datasets with minimal code.

What sets Seaborn apart is its focus on statistical relationships and its beautiful default styles. It comes with built-in themes and color palettes that enhance the readability and aesthetic appeal of

plots without needing much manual customization. Seaborn also makes it easy to add statistical elements such as regression lines, confidence intervals, and distributions to visualizations. Functions like `sns.pairplot()`, `sns.heatmap()`, and `sns.catplot()` allow users to quickly explore and understand patterns, trends, and correlations in their data.

Seaborn is widely used in data analysis, machine learning, and research because it enables faster exploratory data analysis (EDA) and better communication of results through visuals. It integrates well with matplotlib, meaning users can combine the simplicity of Seaborn with the fine-tuning options of matplotlib when needed. By making it easier to create complex visualizations with clear, concise code, Seaborn has become an essential tool for data scientists, analysts, and researchers working in Python.

**Matplotlib**

Matplotlib is a comprehensive and widely-used Python library for creating static, animated, and interactive visualizations. It provides a flexible framework for producing publication-quality plots and figures in a variety of formats. Originally developed to mimic the functionality of MATLAB's plotting capabilities, matplotlib gives users complete control over every aspect of a figure, from axis labels and ticks to line styles and colors. Its core component, `pyplot`, offers a simple interface similar to MATLAB, which makes it easy for beginners to get started with basic plotting tasks.

One of matplotlib’s greatest strengths is its versatility. It supports a wide range of plot types including line plots, scatter plots, bar charts, histograms, pie charts, 3D plots, and more. Users can create multi-panel figures, customize plot aesthetics, and even embed plots in graphical user interfaces or web applications. While it has a steeper learning curve compared to higher-level libraries like Seaborn, matplotlib allows for highly customized and fine-grained visualizations, making it suitable for both simple and complex projects.

Matplotlib plays a foundational role in the Python data visualization ecosystem. It integrates smoothly with NumPy and pandas, and many other libraries like Seaborn, Plotly, and pandas plotting functionality are built on top of it or are compatible with it. Because of its reliability,

maturity, and flexibility, matplotlib is often the go-to library for precise and detailed visualizations in scientific research, data analysis, and engineering applications.

**Pickle**

Pickle is a built-in Python module used for serializing and deserializing Python objects. Serialization, also known as “pickling,” refers to the process of converting a Python object into a byte stream that can be stored in a file or transmitted over a network. Deserialization, or “unpickling,” is the reverse process—reconstructing the original Python object from the stored byte stream. Pickle is particularly useful for saving complex data structures like dictionaries, lists, custom class instances, and machine learning models for later use.

One of the main advantages of using Pickle is its simplicity and compatibility with nearly all Python data types. With just a few lines of code, you can save and load objects using the `pickle.dump()` and `pickle.load()` functions. This is especially helpful in scenarios like machine learning, where you might train a model once and reuse it multiple times without retraining. It can also be used to store intermediate computation results or configuration settings during the execution of large programs.

However, Pickle has some limitations and security considerations. Since it is Python-specific, pickled files are not compatible with other programming languages. More importantly, unpickling data from an untrusted source can be a security risk, as malicious code can be executed during the unpickling process. Therefore, it's recommended to use Pickle only with trusted data. Despite these concerns, Pickle remains a popular and convenient tool for object serialization in Python-based applications.

**Evaluation Metrics**

**Accuracy**

Accuracy is one of the most commonly used performance metrics in classification tasks, representing the proportion of correct predictions made by a model out of all predictions. It is calculated as the ratio of the number of correct predictions (true positives and true negatives) to the total number of predictions. Mathematically, accuracy is expressed as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where TP represents true positives, TN represents true negatives, FP represents false positives, and FN represents false negatives. The simplicity of accuracy makes it an intuitive and easy-to-understand measure, and it is particularly effective when the dataset is balanced (i.e., the classes have roughly equal representation).

However, accuracy can be misleading in cases of class imbalance, where one class significantly outnumbers the other. For instance, in a dataset where 95% of the samples belong to class A and only 5% belong to class B, a model that predicts only class A for every sample would achieve 95% accuracy, despite not identifying any instances of class B. In such cases, accuracy fails to reflect the model's ability to correctly classify the minority class, making it unsuitable as a sole evaluation metric in imbalanced datasets.

In situations where class imbalance exists or when the costs of false positives and false negatives vary, other metrics such as precision, recall, F1-score, or the area under the ROC curve (AUC-ROC) are typically used alongside accuracy. These metrics provide a more nuanced view of model performance by evaluating different aspects of classification accuracy, like how well the model identifies positive instances (recall) or avoids misclassifying negative instances (precision). Therefore, while accuracy is a useful metric, it is important to consider other metrics in contexts where class imbalance or differing error costs exist.

**Precision**

Precision is a performance metric that measures the accuracy of positive predictions made by a model. It is defined as the ratio of true positive predictions to the total number of predicted positives (true positives + false positives). Mathematically, precision is calculated as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Where TP is the number of true positives and FP is the number of false positives. Precision focuses on the model's ability to avoid false positives, making it especially useful in situations

where the cost of a false positive is high, such as in spam detection or medical diagnostics. For instance, in a medical test, a false positive might mean diagnosing a healthy patient as sick, which can lead to unnecessary treatments or tests.

However, precision alone doesn't provide a complete picture of model performance. A model with high precision may still fail to identify many positive instances (i.e., have a low recall), which is a limitation in some applications. Therefore, precision is often used in conjunction with other metrics, like recall and F1-score, to provide a more balanced assessment of a model's effectiveness.

In some domains, the goal is to maximize precision, especially when false positives carry significant consequences. For example, in fraud detection, a high precision ensures that most transactions flagged as fraudulent are indeed frauds, reducing the risk of unnecessary investigations. However, it is important to balance precision with other metrics to ensure the model's performance is robust and generalizes well to various types of errors.

**Recall**

Recall, also known as sensitivity or true positive rate, is a metric that measures the ability of a model to identify all relevant positive instances in the data. It is defined as the ratio of true positives to the total number of actual positives (true positives + false negatives). Mathematically, recall is calculated as:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Where TP is the number of true positives and FN is the number of false negatives. Recall is particularly important in scenarios where failing to identify a positive case has serious consequences, such as in cancer diagnosis or fraud detection. In these cases, a false negative—missing a positive instance—could lead to catastrophic outcomes like undiagnosed illnesses or missed fraudulent activities.

While recall is a valuable metric for assessing how well the model captures positive instances, it does not account for false positives. A model could achieve high recall by predicting many positive instances, but this could lead to an excessive number of false positives, which would decrease precision. Thus, recall is typically balanced with precision to get a more comprehensive evaluation of the model's performance.

In situations where the cost of missing positive instances is more critical than incorrectly classifying negative cases as positive, recall is given more importance. For instance, in emergency medical screenings, the priority may be to catch every potential case, even if it results in a higher false positive rate. However, as with precision, recall is often used alongside other metrics like precision and F1-score to ensure that both false positives and false negatives are minimized.

**F1-Score**

The F1-score is a metric that combines both precision and recall into a single value to provide a balanced evaluation of a model's performance. It is the harmonic mean of precision and recall, defined as:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F1-score is particularly useful when there is an uneven class distribution, as it considers both false positives and false negatives. It is often used in scenarios where you want to balance the trade-off between precision and recall, ensuring that both are reasonably high without one dominating the other. The F1-score ranges from 0 (worst performance) to 1 (best performance), with higher values indicating better balance between precision and recall.

One of the main advantages of the F1-score is that it provides a single metric that takes both false positives and false negatives into account. This makes it ideal for problems where you need to optimize both precision and recall simultaneously, without favoring one over the other. For example, in tasks like information retrieval, spam filtering, or medical diagnoses, it's often critical to balance the risk of false positives and false negatives, and the F1-score offers a straightforward way to do that.

However, the F1-score does not distinguish between the types of errors (false positives vs false negatives), meaning that it may not provide enough detail in some cases where one type of error is more costly than the other. For instance, in a medical diagnostic setting where missing a positive case is far more harmful than having a false positive, recall might be prioritized over the F1-score. Nevertheless, the F1-score remains a widely used metric when dealing with imbalanced datasets or when a balanced approach to error handling is desired.

**Support**

Support refers to the number of actual occurrences of a class in the dataset. It represents the frequency of instances of a given class in the dataset and is used to calculate performance metrics such as precision, recall, and F1-score. Support is not a performance metric itself but serves as a contextualizing measure for these metrics. It shows how many samples of each class are present and helps in interpreting metrics like precision and recall, especially when there is an imbalance between classes.

For example, in a multi-class classification problem, you might find that the support for one class is significantly higher than another. In such cases, metrics like precision, recall, and F1-score could be disproportionately influenced by the class with higher support, so it is important to consider support when interpreting these metrics. A class with a higher support might naturally have better performance metrics simply due to its larger number of samples.

Support is especially useful when comparing performance across different classes in an imbalanced dataset. It helps to understand whether a model is overfitting to the more frequent classes and underperforming on rarer classes. For example, in a fraud detection model, the support for fraudulent transactions is typically very low compared to legitimate ones. In such cases, looking at the support alongside other metrics ensures that the performance evaluation takes into account the rare but important cases.

## Implementation

### Data Augmentation

In the execution phase, the key parameters for training the model are defined, including image size (224x224), batch size (32), and the number of epochs (10). Data augmentation is applied using ImageDataGenerator to enhance model generalization by increasing the diversity of the training data. This helps prevent overfitting by exposing the model to a wider range of variations in the images. The augmentations include:

**Rescaling:** Pixel values are divided by 255, normalizing them to a range between 0 and 1, which is commonly required for neural networks.
**Rotation:** A rotation range of 20 degrees randomly rotates images, simulating real-world rotations of objects.
**Width and Height Shifts:** Random horizontal and vertical shifts (up to 20% of the image size) allow the model to learn better spatial representations.
**Shear and Zoom:** A shear range of 0.2 and zoom range of 0.2 introduce slight distortions and zoom effects, helping the model adapt to varying object perspectives.
**Horizontal Flip:** Random flipping of images horizontally enhances robustness to left-right variations in the objects.
**Fill Mode**: The nearest option is used to fill newly created pixels during transformations with the nearest valid pixel, ensuring the image stays visually consistent.

The augmented images are loaded from the training and validation directories. The training data is shuffled to ensure randomness, while the validation data is not shuffled to maintain its integrity. The number of classes is determined by the number of subdirectories in the training directory, representing different bird species, which is used to define the output layer of the model.

The model architecture is defined using MobileNetV2 as the base model. MobileNetV2 is a pre-trained model on ImageNet, with the top classification layers removed (include_top=False) and input shape set to (224, 224, 3). The base model's weights are frozen (trainable=False) to retain its learned features. A GlobalAveragePooling2D layer is added to reduce the spatial

dimensions of the output from the base model, followed by a dense layer with 256 units and ReLU activation for learning complex features. A Dropout layer with a rate of 0.3 is applied to prevent overfitting. Finally, a dense output layer with softmax activation is used for multi-class classification, where num_classes represents the number of target categories. Different optimizers (SGD, SGD with momentum, Adam, and Nadam) are defined for training the model, allowing comparison of their performance during training. Each optimizer's effectiveness can be evaluated based on the model's accuracy, training time, and convergence. For each optimizer (SGD, SGD with momentum, Adam, and Nadam), the following steps are performed:

**Model Creation and Compilation**

For each optimizer, a new instance of the model is created. The model's architecture is predefined and is based on a MobileNetV2 backbone, which is a pre-trained model with the ImageNet weights. The final layers of the model are customized, including a Global Average Pooling layer to reduce the spatial dimensions of the feature maps, a fully connected Dense layer with 256 neurons, and a Dropout layer for regularization to prevent overfitting.

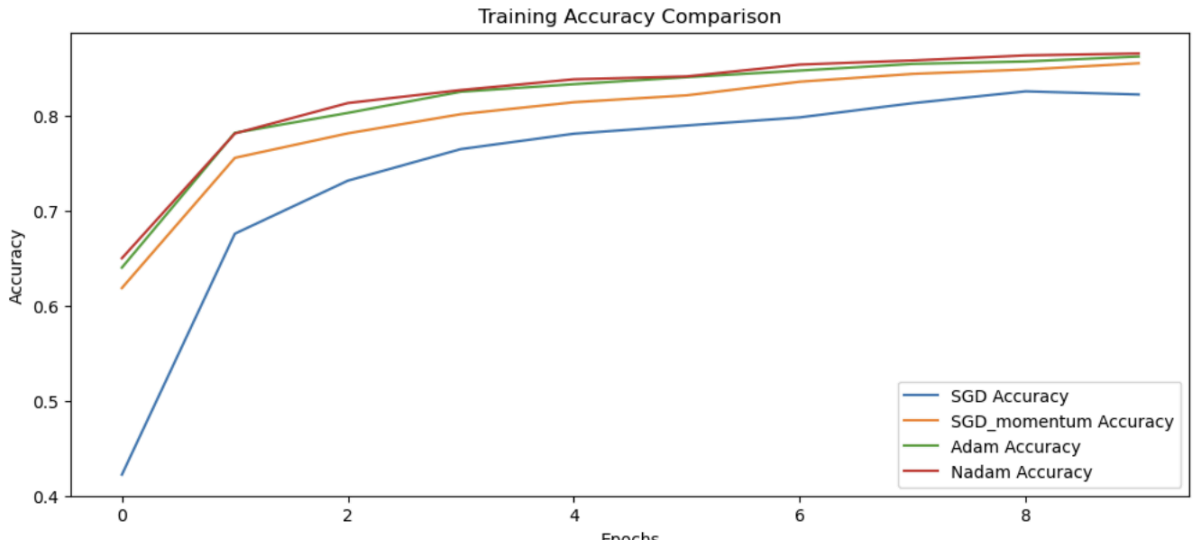
After constructing the model, it is compiled. Compilation is the process of configuring the model with necessary components to perform training. The optimizer (which will vary between SGD, Adam, etc.) is chosen for this step. The categorical cross-entropy loss function is selected, which is appropriate for multi-class classification tasks. The model also tracks accuracy as its evaluation metric during training to monitor performance.

**Training Process**

The model.fit() function starts the training loop. During training, the model learns from the training dataset, and its performance is validated periodically on the validation dataset. The model is trained for a specific number of epochs. An epoch refers to one complete pass through the entire training dataset. After each epoch, the model’s parameters are updated to minimize the loss function, and the validation performance is checked to see how well the model generalizes to unseen data. The training duration is measured by recording the start and end times for each optimizer. This allows for the calculation of the total training time for each optimizer, which can be compared across optimizers to evaluate efficiency.

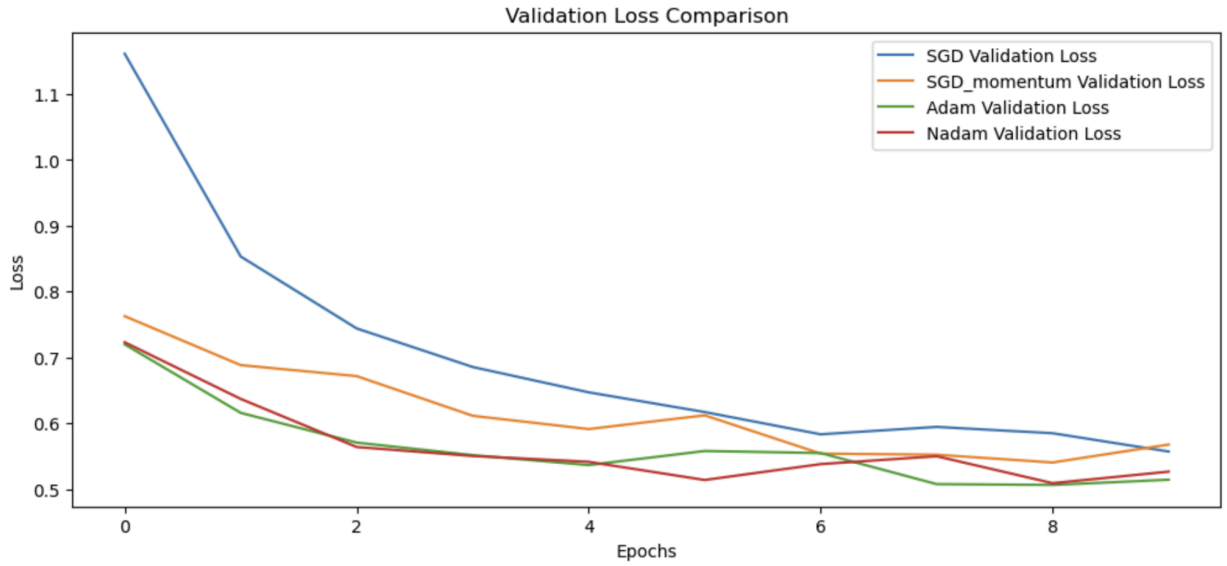
```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1176s 2s/step - accuracy: 0.2631 - loss: 2.6433 - val_accuracy: 0.7036 - val_loss: 1.1613
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1124s 2s/step - accuracy: 0.6518 - loss: 1.2207 - val_accuracy: 0.7620 - val_loss: 0.8533
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1134s 2s/step - accuracy: 0.7193 - loss: 0.9506 - val_accuracy: 0.7860 - val_loss: 0.7441
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1133s 2s/step - accuracy: 0.7565 - loss: 0.8016 - val_accuracy: 0.8032 - val_loss: 0.6855
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1118s 2s/step - accuracy: 0.7811 - loss: 0.7333 - val_accuracy: 0.8144 - val_loss: 0.6470
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 7438s 16s/step - accuracy: 0.7854 - loss: 0.7030 - val_accuracy: 0.8197 - val_loss: 0.6171
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 950s 2s/step - accuracy: 0.7951 - loss: 0.6692 - val_accuracy: 0.8291 - val_loss: 0.5833
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 462s 984ms/step - accuracy: 0.8128 - loss: 0.6143 - val_accuracy: 0.8199 - val_loss: 0.5945
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 487s 1s/step - accuracy: 0.8236 - loss: 0.5876 - val_accuracy: 0.8285 - val_loss: 0.5849
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 499s 1s/step - accuracy: 0.8255 - loss: 0.5474 - val_accuracy: 0.8340 - val_loss: 0.5571
```

**Model training using SGD optimizer**




**Training accuracy for all the models**

The validation evaluation is conducted after each optimizer's training, where the model is assessed on the validation dataset. The primary outcomes of this evaluation are validation accuracy and validation loss. Validation accuracy indicates how well the model generalizes to unseen data, while validation loss helps determine how closely the model's predictions align with the actual values. These results are crucial for comparing the effectiveness of the different optimizers in training models that not only fit well to the training data but also perform effectively on new, unseen data.




**Validation Loss Comparison for all the models**

**Model and History Saving**: After each optimizer completes training, the trained model is saved. It is saved in the HDF5 format with the name reflecting the optimizer used (e.g., model_SGD.h5), which allows for later use and deployment. Additionally, the training history—including metrics such as training and validation accuracy and loss at each epoch—is saved to a file (in pickle format). This history file can be loaded later to analyze the training process, plot graphs, and track the model's progress.

**Logging Metrics**: The time it took to train the model from start to finish for each optimizer. The time taken for one full pass over the training data (one epoch). This metric is helpful to evaluate the computational efficiency of different optimizers.

This is the epoch where the model first achieves its best validation performance (minimum validation loss). This gives insights into how quickly the model converges and whether one optimizer leads to faster convergence. These logs are saved in the train_time_log dictionary, which helps compare the performance of each optimizer in terms of efficiency and convergence.


**Time took for the model with respect to each optimizer in seconds**

## Results & Discussion

In this phase, the performance of the models trained with different optimizers (SGD, SGD with momentum, Adam, and Nadam) is evaluated using the classification report.

This is the summarized version for the training sessions

Optimizer Training Summary:

| Optimizer | Val Accuracy (%) | Val Loss | Total Time (s) | Time/Epoch (s) | Converged Epoch |
|---|---|---|---|---|---|
| SGD | 83.37 | 0.5578 | 15525.66 | 1552.57 | 10 |
| SGD_momentum | 83.45 | 0.5624 | 9210.53 | 921.05 | 9 |
| Adam | 83.60 | 0.5367 | 10893.32 | 1089.33 | 9 |
| Nadam | 84.84 | 0.4997 | 22267.72 | 2226.77 | 9 |

**Fig. Details explaining the summary of all the optimizers**

The classification report is generated by comparing the true labels of the validation set with the predicted labels from the trained models. This report provides a detailed evaluation of the model's performance by including precision, recall, F1-score, and support for each class. These metrics offer insights into how accurately the model identifies each class, how well it balances precision and recall, and how it performs overall for each individual class. The classification report is crucial in understanding not just the accuracy but also the robustness of the model across all classes, highlighting any areas where the model may struggle with certain categories. The model, when it is trained with the Nadam optimizer, has achieved the highest accuracy of 85% compared to any other optimizer.

Classification Report for Nadam:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Asian-Green-Bee-Eater | 0.87 | 0.88 | 0.87 | 300 |
| Brown-Headed-Barbet | 0.74 | 0.78 | 0.76 | 300 |
| Cattle-Egret | 0.91 | 0.93 | 0.92 | 300 |
| Common-Kingfisher | 0.93 | 0.84 | 0.88 | 300 |
| Common-Myna | 0.90 | 0.81 | 0.85 | 300 |
| Common-Rosefinch | 0.71 | 0.76 | 0.74 | 300 |
| Common-Tailorbird | 0.79 | 0.77 | 0.78 | 300 |
| Coppersmith-Barbet | 0.93 | 0.85 | 0.89 | 300 |
| Forest-Wagtail | 0.77 | 0.88 | 0.82 | 300 |
| Gray-Wagtail | 0.86 | 0.81 | 0.84 | 300 |
| Hoopoe | 0.88 | 0.94 | 0.91 | 300 |
| House-Crow | 0.76 | 0.81 | 0.78 | 300 |
| Indian-Grey-Hornbill | 0.71 | 0.76 | 0.73 | 300 |
| Indian-Peacock | 0.88 | 0.89 | 0.89 | 300 |
| Indian-Pitta | 0.91 | 0.82 | 0.86 | 300 |
| Indian-Roller | 0.85 | 0.78 | 0.81 | 300 |
| Jungle-Babbler | 0.89 | 0.83 | 0.86 | 300 |
| Northern-Lapwing | 0.81 | 0.86 | 0.83 | 300 |
| Red-Wattled-Lapwing | 0.89 | 0.89 | 0.89 | 300 |
| Ruddy-Shelduck | 0.96 | 0.91 | 0.93 | 300 |
| Rufous-Treepie | 0.87 | 0.86 | 0.86 | 300 |
| Sarus-Crane | 0.89 | 0.93 | 0.91 | 300 |
| White-Breasted-Kingfisher | 0.91 | 0.89 | 0.90 | 300 |
| White-Breasted-Waterhen | 0.90 | 0.78 | 0.83 | 300 |
| White-Wagtail | 0.77 | 0.94 | 0.85 | 300 |
| accuracy | | | 0.85 | 7500 |
| macro avg | 0.85 | 0.85 | 0.85 | 7500 |
| weighted avg | 0.85 | 0.85 | 0.85 | 7500 |

**235/235** ———————— **261s** 1s/step - accuracy: 0.8388 - loss: 0.5263

**Fig. Classification report of the Nadam Optimizer**

Classification Report for Adam:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Asian-Green-Bee-Eater | 0.87 | 0.86 | 0.87 | 300 |
| Brown-Headed-Barbet | 0.84 | 0.62 | 0.72 | 300 |
| Cattle-Egret | 0.91 | 0.95 | 0.93 | 300 |
| Common-Kingfisher | 0.90 | 0.88 | 0.89 | 300 |
| Common-Myna | 0.84 | 0.82 | 0.83 | 300 |
| Common-Rosefinch | 0.70 | 0.75 | 0.72 | 300 |
| Common-Tailorbird | 0.74 | 0.76 | 0.75 | 300 |
| Coppersmith-Barbet | 0.78 | 0.87 | 0.82 | 300 |
| Forest-Wagtail | 0.82 | 0.85 | 0.83 | 300 |
| Gray-Wagtail | 0.93 | 0.75 | 0.83 | 300 |
| Hoopoe | 0.91 | 0.94 | 0.92 | 300 |
| House-Crow | 0.75 | 0.82 | 0.79 | 300 |
| Indian-Grey-Hornbill | 0.71 | 0.76 | 0.74 | 300 |
| Indian-Peacock | 0.96 | 0.80 | 0.87 | 300 |
| Indian-Pitta | 0.82 | 0.85 | 0.84 | 300 |
| Indian-Roller | 0.79 | 0.82 | 0.81 | 300 |
| Jungle-Babbler | 0.85 | 0.86 | 0.85 | 300 |
| Northern-Lapwing | 0.68 | 0.95 | 0.79 | 300 |
| Red-Wattled-Lapwing | 0.90 | 0.86 | 0.88 | 300 |
| Ruddy-Shelduck | 0.89 | 0.94 | 0.91 | 300 |
| Rufous-Treepie | 0.90 | 0.84 | 0.87 | 300 |
| Sarus-Crane | 0.92 | 0.90 | 0.91 | 300 |
| White-Breasted-Kingfisher | 0.95 | 0.88 | 0.92 | 300 |
| White-Breasted-Waterhen | 0.87 | 0.81 | 0.84 | 300 |
| White-Wagtail | 0.96 | 0.86 | 0.90 | 300 |
| accuracy | | | 0.84 | 7500 |
| macro avg | 0.85 | 0.84 | 0.84 | 7500 |
| weighted avg | 0.85 | 0.84 | 0.84 | 7500 |

**235/235** ━━━━━━━━━━━━━━━━━━━━ **307s** 1s/step - accuracy: 0.8241 - loss: 0.5733

**Fig. Classification report for the Adam Optimizer**

Classification Report for SGD_momentum:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Asian-Green-Bee-Eater | 0.89 | 0.76 | 0.82 | 300 |
| Brown-Headed-Barbet | 0.64 | 0.82 | 0.72 | 300 |
| Cattle-Egret | 0.83 | 0.97 | 0.89 | 300 |
| Common-Kingfisher | 0.91 | 0.87 | 0.89 | 300 |
| Common-Myna | 0.81 | 0.84 | 0.83 | 300 |
| Common-Rosefinch | 0.88 | 0.68 | 0.77 | 300 |
| Common-Tailorbird | 0.74 | 0.72 | 0.73 | 300 |
| Coppersmith-Barbet | 0.80 | 0.86 | 0.83 | 300 |
| Forest-Wagtail | 0.83 | 0.84 | 0.83 | 300 |
| Gray-Wagtail | 0.92 | 0.77 | 0.84 | 300 |
| Hoopoe | 0.93 | 0.93 | 0.93 | 300 |
| House-Crow | 0.84 | 0.67 | 0.75 | 300 |
| Indian-Grey-Hornbill | 0.73 | 0.72 | 0.72 | 300 |
| Indian-Peacock | 0.90 | 0.83 | 0.87 | 300 |
| Indian-Pitta | 0.72 | 0.91 | 0.80 | 300 |
| Indian-Roller | 0.74 | 0.85 | 0.79 | 300 |
| Jungle-Babbler | 0.88 | 0.79 | 0.83 | 300 |
| Northern-Lapwing | 0.79 | 0.84 | 0.81 | 300 |
| Red-Wattled-Lapwing | 0.81 | 0.89 | 0.85 | 300 |
| Ruddy-Shelduck | 0.95 | 0.93 | 0.94 | 300 |
| Rufous-Treepie | 0.90 | 0.84 | 0.87 | 300 |
| Sarus-Crane | 0.77 | 0.95 | 0.85 | 300 |
| White-Breasted-Kingfisher | 0.91 | 0.89 | 0.90 | 300 |
| White-Breasted-Waterhen | 0.84 | 0.80 | 0.82 | 300 |
| White-Wagtail | 0.98 | 0.79 | 0.87 | 300 |
| accuracy | | | 0.83 | 7500 |
| macro avg | 0.84 | 0.83 | 0.83 | 7500 |
| weighted avg | 0.84 | 0.83 | 0.83 | 7500 |

**235/235** ━━━━━━━━━━━━━━━━━━━━ **156s** 664ms/step - accuracy: 0.8213 - loss: 0.6027

**Fig Classification report for the SGD with momentum Optimizer**

Classification Report for SGD:

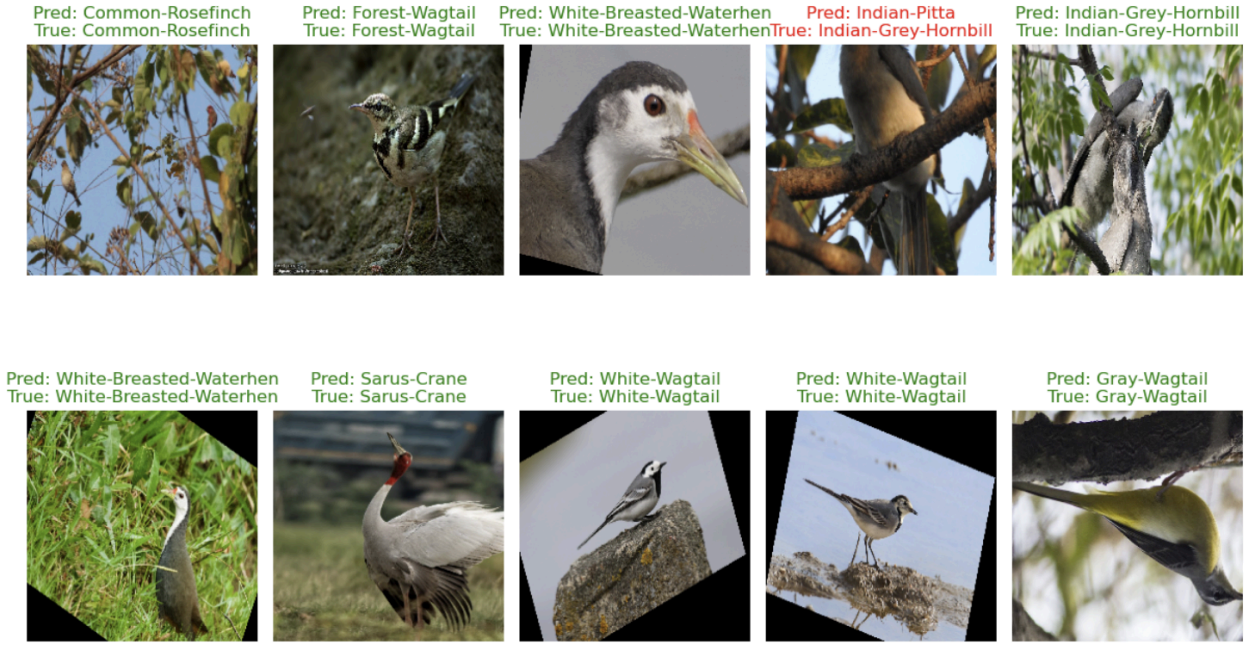
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Asian-Green-Bee-Eater | 0.82 | 0.87 | 0.84 | 300 |
| Brown-Headed-Barbet | 0.71 | 0.65 | 0.67 | 300 |
| Cattle-Egret | 0.89 | 0.92 | 0.91 | 300 |
| Common-Kingfisher | 0.85 | 0.90 | 0.88 | 300 |
| Common-Myna | 0.77 | 0.83 | 0.80 | 300 |
| Common-Rosefinch | 0.74 | 0.70 | 0.72 | 300 |
| Common-Tailorbird | 0.72 | 0.74 | 0.73 | 300 |
| Coppersmith-Barbet | 0.78 | 0.85 | 0.82 | 300 |
| Forest-Wagtail | 0.76 | 0.86 | 0.81 | 300 |
| Gray-Wagtail | 0.86 | 0.81 | 0.83 | 300 |
| Hoopoe | 0.96 | 0.87 | 0.91 | 300 |
| House-Crow | 0.76 | 0.80 | 0.78 | 300 |
| Indian-Grey-Hornbill | 0.72 | 0.72 | 0.72 | 300 |
| Indian-Peacock | 0.88 | 0.81 | 0.85 | 300 |
| Indian-Pitta | 0.87 | 0.81 | 0.84 | 300 |
| Indian-Roller | 0.85 | 0.78 | 0.81 | 300 |
| Jungle-Babbler | 0.72 | 0.86 | 0.79 | 300 |
| Northern-Lapwing | 0.80 | 0.83 | 0.81 | 300 |
| Red-Wattled-Lapwing | 0.92 | 0.85 | 0.89 | 300 |
| Ruddy-Shelduck | 0.92 | 0.94 | 0.93 | 300 |
| Rufous-Treepie | 0.87 | 0.81 | 0.84 | 300 |
| Sarus-Crane | 0.86 | 0.90 | 0.88 | 300 |
| White-Breasted-Kingfisher | 0.91 | 0.85 | 0.88 | 300 |
| White-Breasted-Waterhen | 0.84 | 0.78 | 0.81 | 300 |
| White-Wagtail | 0.91 | 0.91 | 0.91 | 300 |
| accuracy | | | 0.83 | 7500 |
| macro avg | 0.83 | 0.83 | 0.83 | 7500 |
| weighted avg | 0.83 | 0.83 | 0.83 | 7500 |

**235/235** ━━━━━━━━━━━━━━━━━━━━ **161s** 687ms/step - accuracy: 0.8214 - loss: 0.5828

**Fig. Classification report for the SGD Optimizer**

The results from the classification report are printed for each optimizer, providing insights into how well each optimizer performs. This loop ensures the comparison of the optimizers based on various performance metrics.

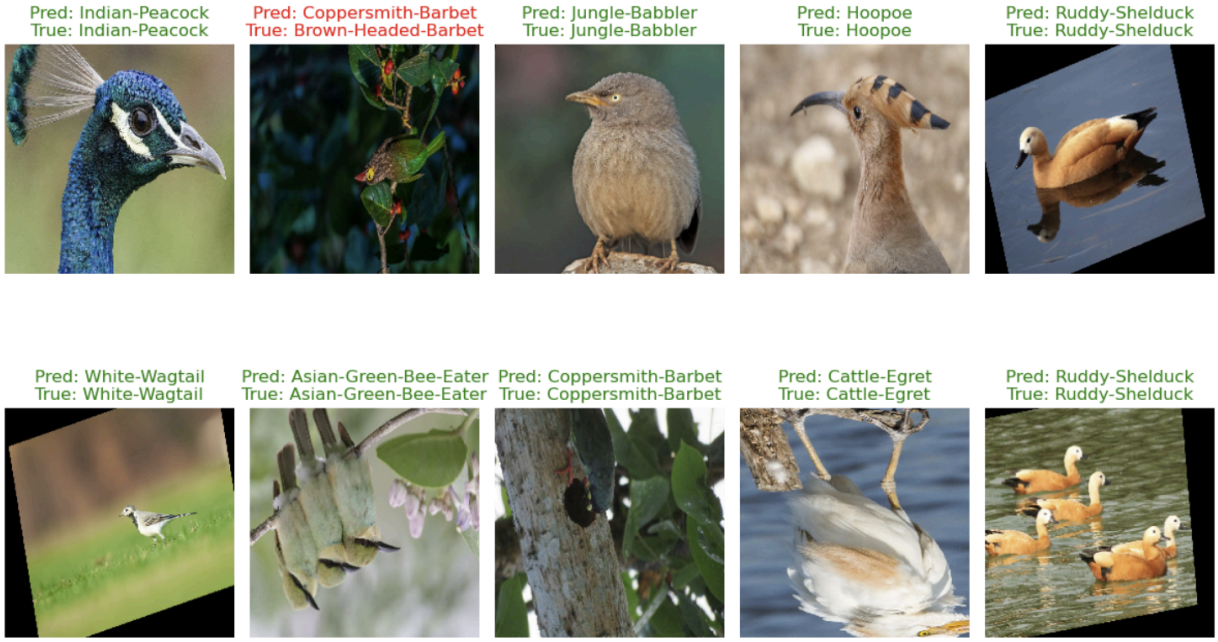
Later we visualized the predictions of the model on a random set of 10 images from the validation dataset. It loads the trained model (using every optimizer) and compares its predictions to the true labels of the images. Each image is displayed with its predicted and true labels, where correct predictions are highlighted in green, and incorrect ones are shown in red. This visualization helps to quickly assess the model's performance on individual images, offering insights into how accurately the model classifies different bird species in the validation set. The results provide an intuitive way to evaluate the model's generalization capabilities and identify any misclassifications.




**Predictions using the Nadam Optimizer**


**Predictions using the Adam optimizer**


**Predictions using the SGD with momentum optimizer**




**Predictions using the SGD optimizer**

## Conclusion

This project successfully demonstrates the practical implementation of transfer learning using MobileNetV2 for fine-grained bird species classification. The model's lightweight yet powerful architecture, designed specifically for efficiency on resource-constrained devices, allowed us to achieve high classification accuracy without requiring substantial computational resources. By leveraging data augmentation techniques, we were able to artificially increase the diversity of training samples, which not only improved generalization but also reduced overfitting, a common challenge in smaller datasets.

A comparative analysis of four optimizers Stochastic Gradient Descent (SGD), SGD with Momentum, Adam, and Nadam highlighted the critical role that optimizer choice plays in model convergence, training dynamics, and overall performance. Among these, Adam and Nadam demonstrated faster convergence and higher accuracy due to their adaptive learning rate properties, while SGD variants provided more stable learning curves in some configurations.

The project's results reinforce the suitability of MobileNetV2 for tasks involving limited datasets and constrained environments, striking an effective balance between speed and accuracy. The project further underscores the importance of fine-tuning pre-trained models, showing that even with a modest amount of labeled data, transfer learning can produce highly accurate classification systems.

Moreover, the approach taken in this project is not limited to avian classification; it is a generalizable framework that can be adopted for other fine-grained classification problems across different domains such as medical imaging, botany, and wildlife monitoring. It bridges the gap between theoretical deep learning concepts and their real-world deployment, contributing to a scalable solution for biodiversity research and conservation efforts.

Through detailed experimentation and thorough evaluation, this project not only achieved its technical objectives but also served as a strong educational exercise in model selection, optimization strategy comparison, and performance evaluation in a constrained deep learning scenario.

## Future Work

Increase dataset size: Incorporate more samples per class and introduce rare bird species for better generalization.

**Use ensemble models:** Combine predictions from multiple fine-tuned CNNs (e.g., ResNet50, InceptionV3) to improve robustness. Add attention mechanisms: Integrate visual attention layers to help the model focus on discriminative parts of the birds.
Incorporate real-time inference: Deploy the model on mobile or embedded systems using TensorFlow Lite.

**Evaluate with fine-grained metrics:** Add confusion matrix, class-wise F1 scores, and misclassification analysis in future experiments. Fine-Grained Localization and Part-Based Classification: Move beyond whole-image classification by implementing part-based models that detect specific bird features (e.g., wings, beak, plumage) for enhanced discriminative power. Techniques like Pose Normalization Networks or Keypoint R-CNN can be explored.

**Use of Transformer Architectures:** Evaluate the performance of Vision Transformers (ViT) or hybrid CNN-Transformer models on fine-grained bird classification. These architectures are showing promising results in image recognition and may outperform CNNs in certain settings.

## References


1. Howard, A.G., 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.
2. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. and Chen, L.C., 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4510-4520).
3. He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
4. Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.
5. Zhou, F. and Lin, Y., 2016. Fine-grained image classification by exploring bipartite-graph labels. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1124-1133).
6. Wah, C., Branson, S., Welinder, P., Perona, P. and Belongie, S., 2011. The caltech-ucsd birds-200-2011 dataset.
7. Kumar, N., Belhumeur, P.N., Biswas, A., Jacobs, D.W., Kress, W.J., Lopez, I.C. and Soares, J.V., 2012. Leafsnap: A computer vision system for automatic plant species identification. In *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part II 12* (pp. 502-516). Springer Berlin Heidelberg.
8. Redmon, J. and Farhadi, A., 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.
9. Esteva, A., Kuprel, B., Novoa, R.A., Ko, J., Swetter, S.M., Blau, H.M. and Thrun, S., 2017. Dermatologist-level classification of skin cancer with deep neural networks. *nature*, *542*(7639), pp.115-118.
10. Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556
11. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z., 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2818-2826)