**Logo

Description automatically generated**

**GENERAL SIR JOHN KOTELAWALA DEFENCE UNIVERSITY**

**B.Sc. (Hons) Data Science and Business Analytics**

**Intake 39**

**Image Processing & Computer Vision**

**CS 4132**

**Advanced Face Recognition-based Attendance System**

|  |  |
| --- | --- |
| Report Details | |
| Student Name | TM Kahavidhana |
| Index Number | D/DBA/22/0017 |

**Submission Date: 07.04.2025**

Contents

[Chapter 1: Introduction 3](#_Toc194973444)

[1.1 Project Focus 3](#_Toc194973445)

[1.2 Why Face Recognition for Attendance? 4](#_Toc194973446)

[1.3 Objectives 4](#_Toc194973447)

[1.4 Scope and Limitations 4](#_Toc194973448)

[1.5 Future Implications 5](#_Toc194973449)

[1.6 Tools And Techniques 5](#_Toc194973450)

[Chapter 2: Dataset Preparation 5](#_Toc194973451)

[2.1 Data Collection 5](#_Toc194973452)

[2.2 Face Cropping and Cleaning 6](#_Toc194973453)

[2.3 Dataset Organization 8](#_Toc194973454)

[2.4 Dataset Summary 9](#_Toc194973455)

[Chapter 3: Data Preprocessing 9](#_Toc194973456)

[3.1 Initial Image Characteristics 9](#_Toc194973457)

[3.2 Resizing and Format Normalization 10](#_Toc194973458)

[3.3 Dataset Generator 11](#_Toc194973459)

[Chapter 4: Experiments & Results 13](#_Toc194973460)

[4.1 Basic CNN (Custom-built CNN from Scratch) 13](#_Toc194973461)

[4.2 Transfer Learning with MobileNetV2 (Frozen Base Model) 15](#_Toc194973462)

[4.3 MobileNetV2 Transfer Learning with Additional Fine-Tuned Custom Convolutional Layers 18](#_Toc194973463)

[Chapter 5: Model Testing 24](#_Toc194973464)

[Chapter 6: Conclusion 25](#_Toc194973465)

# Chapter 1: Introduction

In the era of digitization, automation and artificial intelligence (AI) have transformed how institutions handle routine tasks—one such task being attendance marking. Students' attendance has traditionally been a manual process that takes a significant amount of time and is prone to errors, proxy attendance, and inefficiencies. To bypass these limitations, biometric-based solutions have gained traction, with face recognition emerging as one of the most precise and non-intrusive biometric modalities for identity verification.

This project details the development of an advanced face recognition-based attendance system using Python. Rather than building a full web or mobile app, the emphasis of this assignment is to design, train, and test a machine learning model that has the ability to recognize faces of 10 different students in a class with high accuracy. It employs state-of-the-art deep learning architectures and image processing techniques to detect and classify faces, a non-contact and efficient means of recognizing students.

Through the use of transfer learning with pre-trained models such as MobileNetV2, this project gains better training efficiency and recognition accuracy. The dataset was created personally, consisting of labeled facial photographs for each student, and augmented for greater generalization. The model trained is now able to identify a student from an uploaded image, making it a solid foundation for future integration into real-time systems such as classroom cameras or mobile attendance apps.

This report documents the entire pipeline—from the preparation of the data, preprocessing, model design, training and fine-tuning, to testing on real student images. It also covers the techniques taken to prevent overfitting, boost performance, and ensure that the model generalizes well to new, unseen data. Although this system is currently limited to recognizing ten individuals, it provides a scalable basis for future extension to larger populations and more advanced features.

## 1.1 Project Focus

The primary goal of this project is not building a user-confronting product like an app or website but designing and testing a robust and scalable AI model that can accurately detect student faces. By using advanced tools such as TensorFlow, Keras, and transfer learning techniques (e.g., MobileNetV2), the project explores how modern AI pipelines can be leveraged in a practical educational use case.

To achieve this, a facial image dataset was collected and annotated, showing 10 students. Every student has a group of images with different lighting, facial expressions, and orientations to simulate classroom situations in real life. The images were augmented by applying different techniques—rotation, flipping, and contrast variation—to improve the generalization and robustness of the model.

The model was optimized using deep convolutional neural networks (CNNs) and MobileNetV2 as the feature extractor. With the pre-trained network layers freezing and fine-tuning, the system was successful in learning subtle differences between faces despite having fewer training data samples per class.

## 1.2 Why Face Recognition for Attendance?

Face recognition offers several advantages over biometric and conventional attendance systems:

* Non-intrusive: Face recognition is not a contact-intensive procedure, in contrast to card or fingerprint readers.
* Quick and Real-time: Once installed, face recognition systems can identify several individuals within seconds.
* Hard to Fake: Face recognition is harder to fake than ID cards or attendance signatures.
* Scalable: The system can be expanded to accommodate hundreds or even thousands of students with minimal adjustments.

These advantages position facial recognition as an attractive solution for schools seeking to computerize their administrative operations.

## 1.3 Objectives

* To explore the application of face recognition as a biometric approach for automating student attendance in an academic setting.
* To design and train a facial recognition model capable of accurately identifying 10 students from a labeled image dataset.
* To apply advanced AI methodologies, including transfer learning and image augmentation to improve model performance with limited data.
* To evaluate the system's effectiveness using performance metrics such as accuracy, prediction confidence, and misclassification analysis.
* To lay the foundation for future integration into a real-time attendance monitoring system, such as classroom camera feeds or automated check-in kiosks.

## 1.4 Scope and Limitations

This project has been focused on developing and testing a face recognition model for student identification. The following constraints are acknowledged:

* The system has been capped at 10 students, based on the training data.
* Dataset is not large data set, it contains only 100 images that collected from 10 students.
* Real-time webcam input and attendance logging features were not included due to scope and time constraints.
* The model was tested using static image uploads rather than live video feed.

Despite these limitations, the system is a good starting point for future extensions, such as being combined with a camera feed, classroom dashboard, or a mobile phone for real-time recognition and attendance marking.

## 1.5 Future Implications

This project not only demonstrates the technical feasibility of an attendance system using facial recognition but also offers avenues for numerous future developments and directions for research. These are:

* Interoperability with hardware modules such as IP cameras or Raspberry Pi modules for real-time identification.
* Security measures such as liveness detection to guard against spoofing using printed images or videos.
* Enhanced scalability to support multiple classes, departments, or even institutions.
* Optimization techniques to reduce latency and enhance deployment on low-resource devices.

As education institutions continue to embrace digitalization, AI-enabling solutions such as this will continue to play a key role in enabling administrative processes, enhancing security, and improving pupil accountability.

## 1.6 Tools and Techniques

* Programming Language: Python
* Libraries: OpenCV, TensorFlow/Keras, DeepFace, NumPy, Pandas, Matplotlib
* Models Used: MobileNetV2
* Platform: Google Colab

# Chapter 2: Dataset Preparation

The performance of any face recognition system also greatly relies on the representativeness and quality of the test and training dataset. For this project, a proprietary dataset was manually generated and carefully curated to ensure accuracy of recognizing the faces of ten specific students.

## 2.1 Data Collection

The initial step involved the collection of face images of 10 students from the university environment. 10 different images were taken for each student, amounting to 100 images in the dataset. Images were captured under varying lighting, expressions, and poses to encompass natural variations and make the model robust as shown in belove.

Nevertheless, some of the images collected were not well-centered or did not plainly reveal the student's face, example is shown in belove. These inconsistencies can be anticipated to worsen model performance. To rectify this, manual preprocessing steps were introduced.



## 2.2 Face Cropping and Cleaning

To ensure that the dataset included facial features alone and not extraneous background information, manual cropping was performed. Using image processing software such as default image viewers and basic cropping software:

* Areas that were not facial (background, body parts, distractions) were cropped out like shown in belove.



* Cropping placed the face area as close to the center as possible like shown in belove.



* Faces that were poorly lit, blurry, or blocked like shown in belove were retouched or replaced.



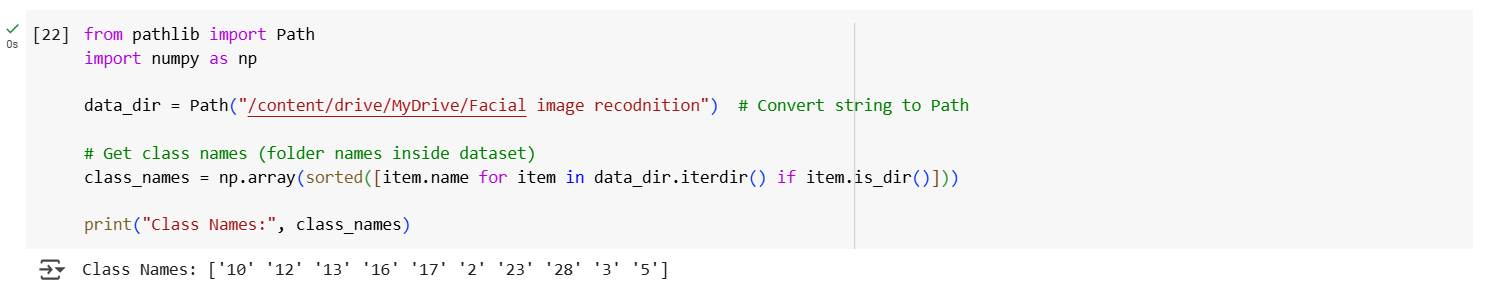
This manual yet crucial step significantly improved the overall quality of the dataset and prepared it for consumption by deep learning models.

## 2.3 Dataset Organization

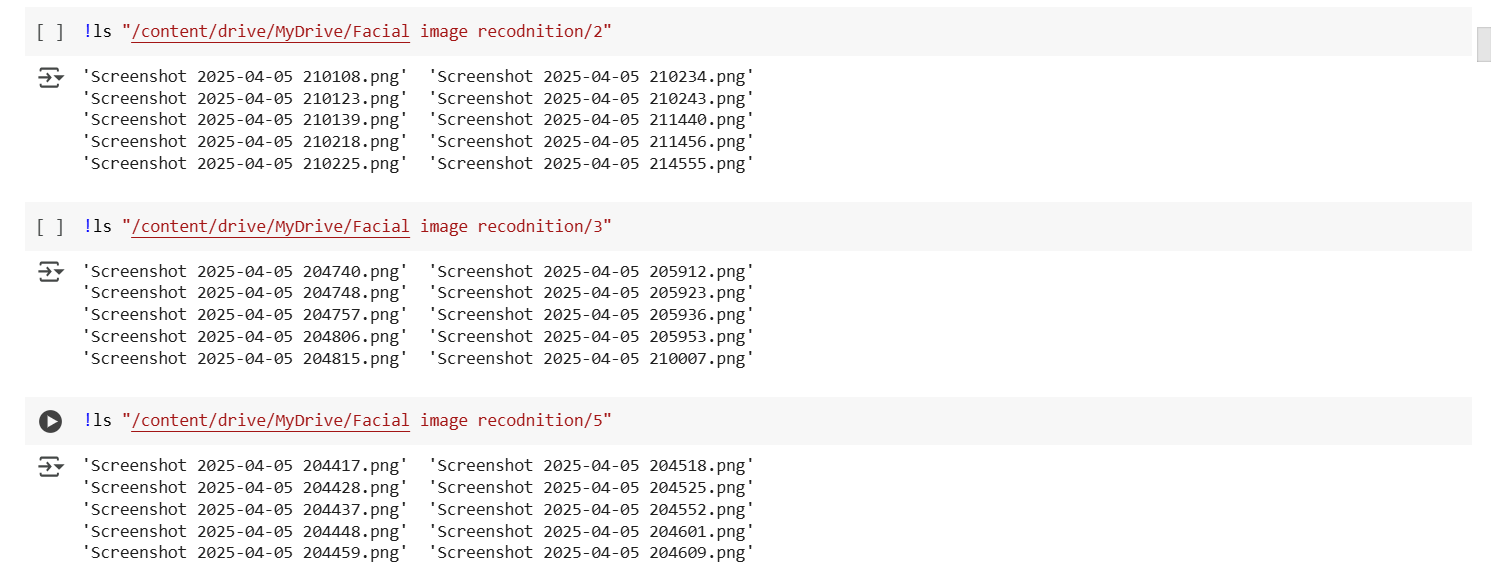
Once the face regions were manually cropped and cleaned, the next important task was to arrange the images in such a way that they are deep learning framework-compatible, such as TensorFlow and Keras. If the folder is properly organized, the framework itself can automatically infer which image corresponds to which class (student) without relying on another label file. Dataset folders are shown in belove.

* The root folder was named /Facial image recognition/.
* Each subfolder inside this directory represented a separate student (i.e., a class).
* Each subfolder contained exactly 10 images of that particular student, all in .jpg format.









## 2.4 Dataset Summary

|  |  |
| --- | --- |
| Metric | Value |
| Total Students (Classes) | 10 |
| Images per Student | 10 |
| Total Number of Images | 100 |
| Image Format | .jpeg, .png |
| Image Dimensions (Intial) | 231×229×4 - 894×988×4 |
| Dataset Balance | Balanced |

# Chapter 3: Data Preprocessing

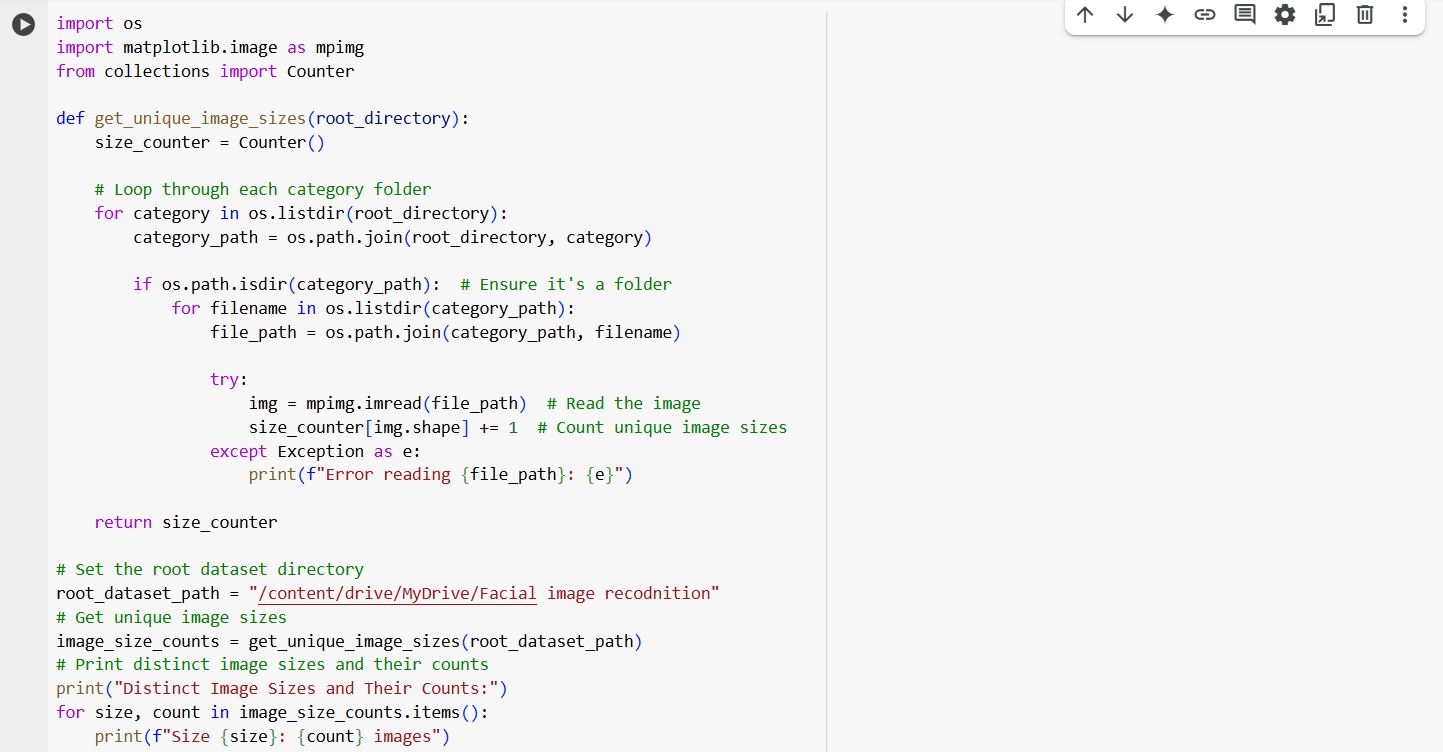
Preprocessing is always required in all computer vision-based machine learning tasks to ensure data consistency in format, enhance model performance, and eliminate noise. In the current project, preprocessing was crucial due to the broad range of images collected from diverse sources and devices, resulting in inconsistency in image dimensions as well as channels.

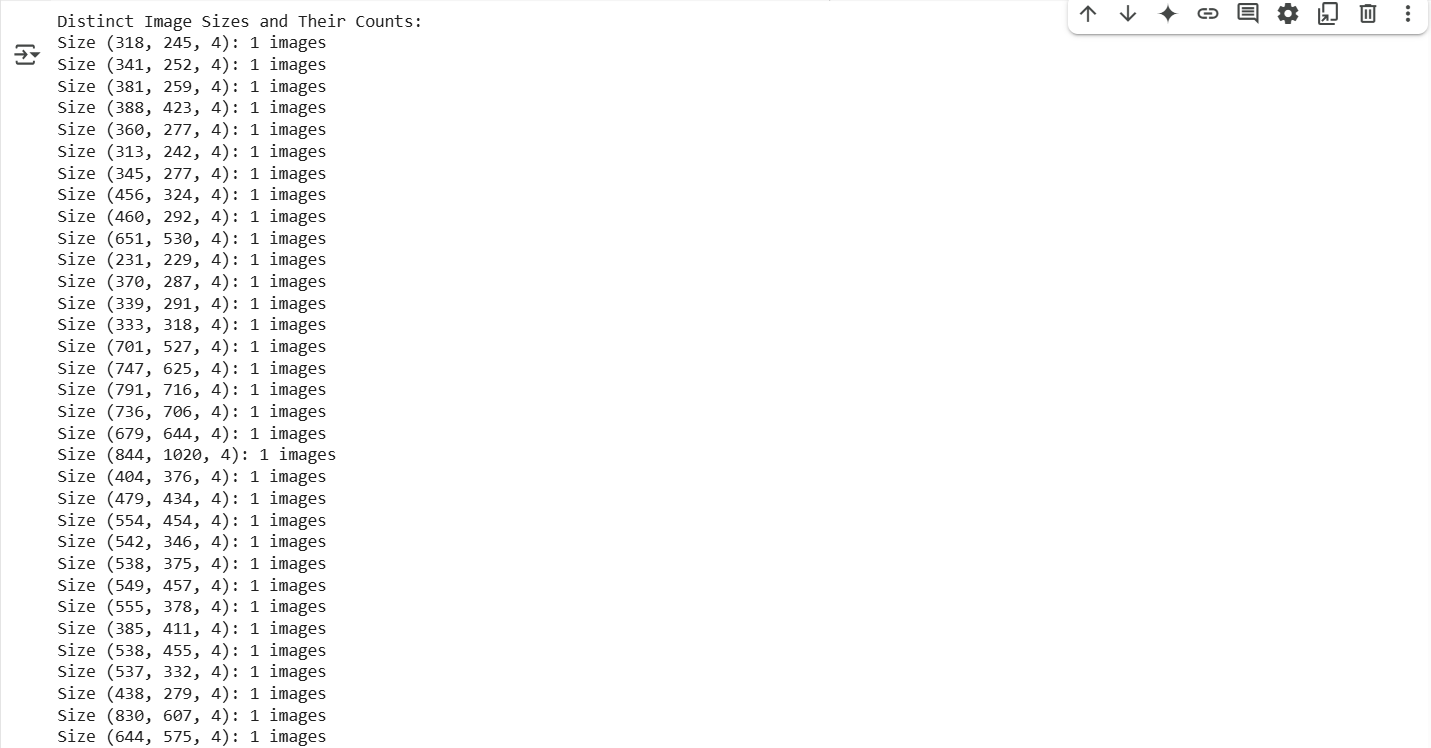
## 3.1 Initial Image Characteristics

The data set consisted of 10 hand-cropped face images of 10 students, and every student had 10 images, thus totaling 100. However, an investigation into the original data revealed the images were of different resolutions and dimensions, especially concerning height, width, and the number of channels.

Some key findings are as follows:

* Quite a variation between 231×229 pixels to 894×988 pixels.





* Channel inconsistency: all images had 4 channels (RGBA) instead of the standard 3 channels (RGB).
* Images that were poorly focused or taken were weeded out and manually cropped to retain just the face area.

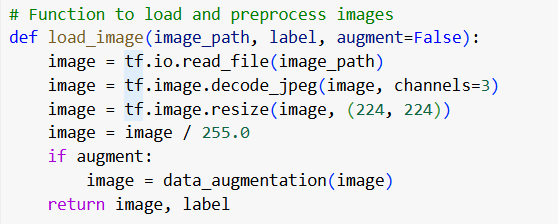
## 3.2 Resizing and Format Normalization

To ensure that all input images were of the same size and format, a preprocessing pipeline was utilized using TensorFlow's tf.image module. This allowed efficient image processing directly within the TensorFlow training pipeline.

Specifically:

* Resized to 224×224 pixels: All images were resized to 224×224 to meet the expected input size of standard CNN architectures such as MobileNet, ResNet, and VGG.
* Format Normalization: Normalized pixel values to range [0, 1] by dividing each pixel by 255.0, which stabilizes training and speeds up convergence.
* Automatic Decoding: Were read from JPEG files and decoded with three channels (RGB), making it format-independent irrespective of the source image format (even when there were alpha channels initially).

This was all done inside the load\_image() function using TensorFlow:



## 3.3 Dataset Generator

To prepare the dataset for training a facial recognition model, the pipeline used TensorFlow's tf.data.Dataset API instead of ImageDataGenerator. This modern approach offers greater efficiency, flexibility, and GPU compatibility.

For the division of the dataset, we used an 80% train and 20% validation split. Stratification ensures that all subsets have an equivalent proportion of classes (students). This ensures that there's no bias when training and testing to ensure learning occurs from an actual representative subset of the data. The 80/20 ratio is a popular threshold because it gives the model enough data (80%) to learn patterns and leaves 20% for quality testing. The ratio is well-fitting for small data sets, like the 100 images per class in this example, because it's enough to provide training data and enough for testing but not enough to risk overfitting or under-testing.

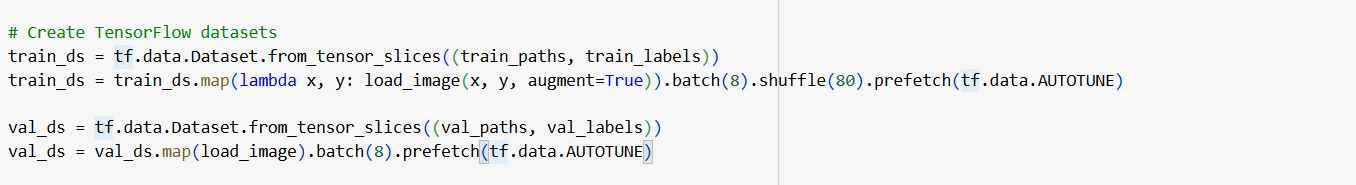
For augmentation of data, we employed on-the-fly transformations through keras.Sequential to run multiple augmentations solely on the training set. These include random horizontal flip, which simulates small face direction variations, and allows the model to learn invariant features notwithstanding pose variation. Random rotation (±10%) simulates head tilt or movement of the camera angle but capping it at 10% ensures the face features are not distorted. Random zoom (±10%) is employed to simulate changing distances from the camera so that the model does not rely on fixed face scales. Random contrast (±10%) completes by adjusting lighting conditions, improving the model's ability to cope with changes in real environments, e.g., changing weather or indoor lights.

Augmentation is essential in this case because each class has only 10 images. With data augmentation, we essentially have a bigger dataset size, introduce diversity, and allow the model to generalize better. This reduces overfitting and allows the model to learn more varieties of facial features from the small data. Augmentation allows the model to generalize better to real-world scenarios by exposing the model to varied inputs during training.

In terms of batching, the batch size was established as 8 to maximize training. Batching allows the model to train on multiple images concurrently, maximizing GPU usage and speeding up training. A small batch size of 8 was chosen for a variety of reasons. Firstly, the dataset is relatively small, and thus smaller batches prevent overfitting. Also, using a batch size of 8 enables the model to adjust weights more frequently after passing through each batch, which can improve generalization. The batch size is also appropriate for environments like Google Colab, where memory is limited compared to high-end hardware that can use larger batches (e.g., 32 or 64).

Finally, we introduced shuffling and prefetching to the training pipeline to maximize the training process. Randomizing the train set prevents the model from learning data order so that it does not incorporate biases such as memorization of class orders. Shuffling with the use of a buffer (e.g., 80) will provide randomization without having the entire dataset in memory. Prefetching with the use of AUTOTUNE will provide data pre-processing overlap during model execution. While working on one, the GPU gives the CPU time to compute the next and reduce idle times, increasing efficiency in training. AUTOTUNE is the feature that allows the prefetch buffer size to dynamically adjust based on system resources consumed, thereby conserving performance.

Example code used:



This approach ensures efficient loading, preprocessing, and augmentation of data directly within the TensorFlow training loop, making it highly suitable for large-scale or real-time model training scenarios.

# Chapter 4: Experiments & Results

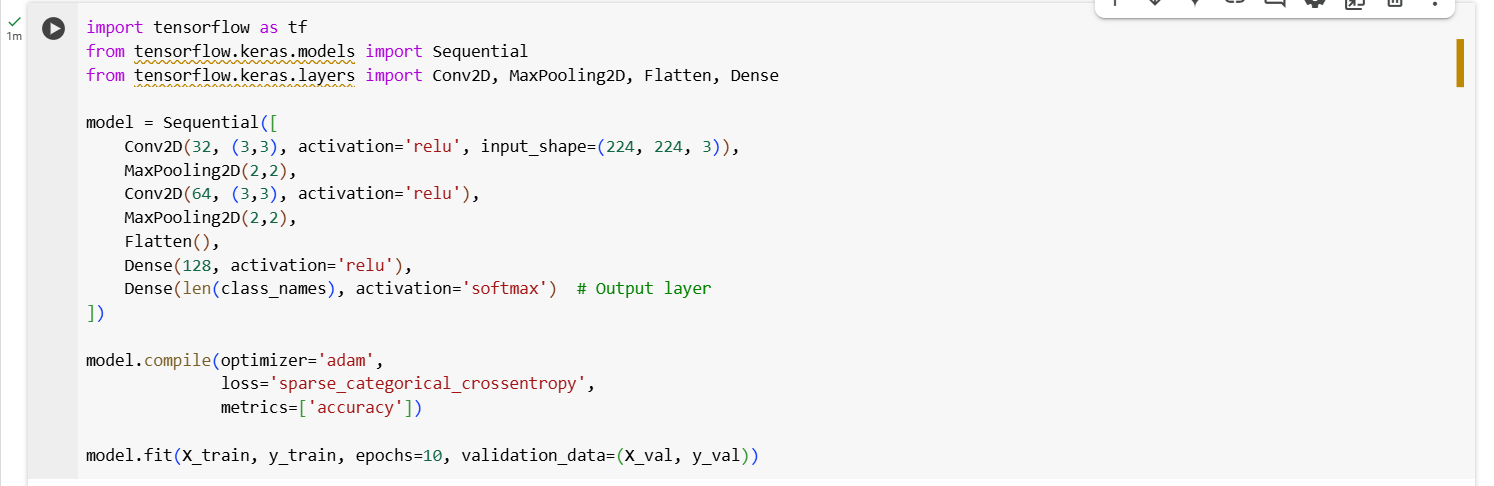
## 4.1 Basic CNN (Custom-built CNN from Scratch)

Architecture:

* 2x Conv2D + MaxPooling layers: It consists of two convolutional layers (Conv2D) with a pooling layer (MaxPooling) following each. The convolutional layers detect features such as edges, shapes, or textures in the images, and the pooling layers downsample the spatial dimensions, which helps to decrease the computational load and retain only the most important features.
* Flatten → Dense (128) → Output Dense layer (softmax): After the convolutional and pooling layers, the feature maps are flattened into a one-dimensional vector. This is then passed through a dense layer of 128 neurons that performs the task of combining learned features into a higher-level representation. Finally, the output layer with a softmax activation function categorizes the image into one of the 10 classes.

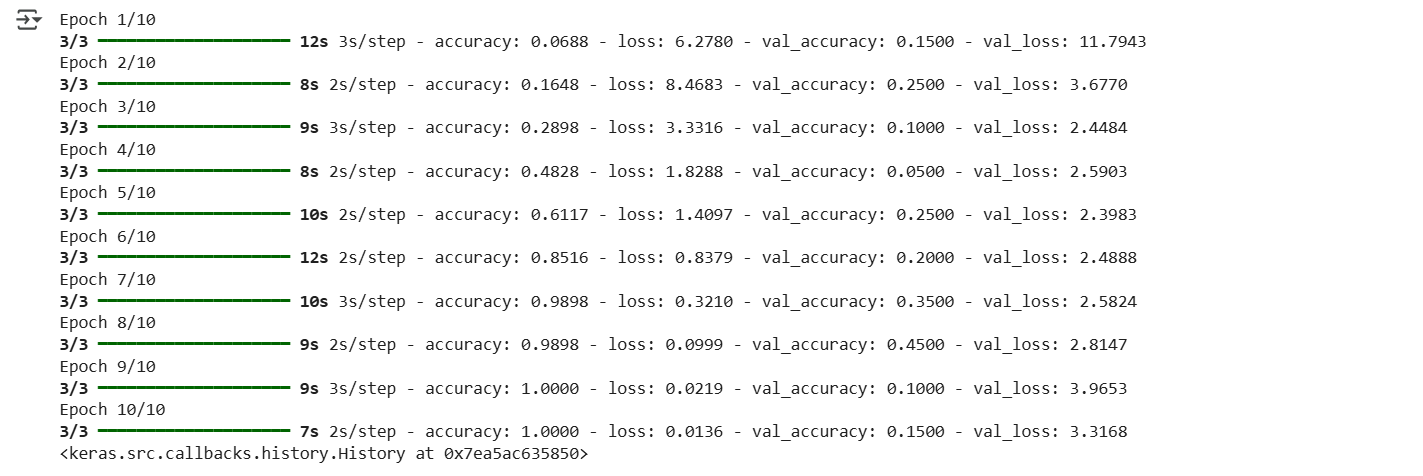
Parameters:

* Loss(sparse\_categorical\_crossentropy): This is the loss function for classification problems when the target labels are integers (not one-hot encoded). It's perfect for multi-class classification problems like this.
* Optimizer(adam): Adam is an adaptive optimization algorithm that adapts the learning rate during training. It's widely used because it's efficient and can deal with large datasets with sparse gradients.
* Epochs (10): The model was trained for 10 epochs, i.e., the entire dataset was passed through the model 10 times.



Training Results:

* Accuracy: Training accuracy of the model was perfect on the training data (100%), i.e., it learned the training examples really well. Validation Accuracy, the performance on new, unseen data (validation set) was significantly worse (15%). This shows that while the model memorized the training data, it did not generalize well to new, unseen data.
* Loss: Training Loss of the model is around 0. The model was successfully able to reduce the loss function on the training data, meaning that it fitted the training data well. Validation Loss of the model is Around 3.3. The loss on the validation data was much higher. This means that the model is not generalizing well to unseen data and is likely to be overfitting.



Observation:

* Overfitting occurred early: Overfitting happens when the model learns the training data too well, including noise and irrelevant patterns. As a result, the model performs well on the training data but not on the validation set because it does not generalize to new and unseen data.
* Model memorized the training data: The model's perfect accuracy on the training set indicates that it simply memorized the training data instead of learning generalizable patterns. This is a strong indication of overfitting.
* Failed to generalize: Even though it performed optimally on the training set, the model was unable to generalize to new, unseen validation data. The fact that the validation accuracy was much lower (15%) indicates this problem.

Conclusion:

* Small Dataset Size: When there is limited data, the model has fewer examples to learn from, and thus the model is more prone to overfit. More data would allow the model to learn more generalizable patterns.
* Complex Classification Problem: Image classification issues with complex features require models with strong generalization. A simple CNN might not be capable of capturing all of the subtleties of the data, especially if the issue is non-trivial.

## 4.2 Transfer Learning with MobileNetV2 (Frozen Base Model)

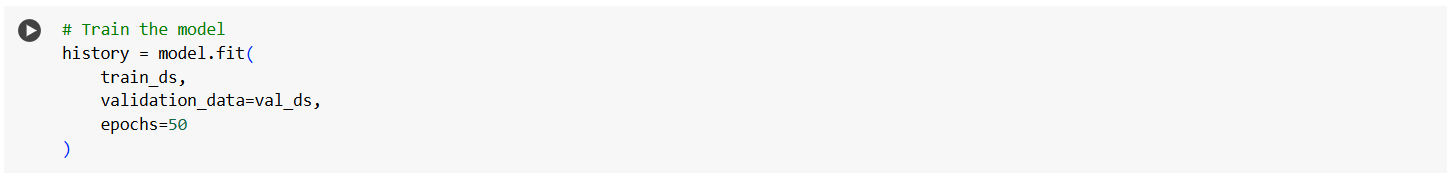
Architecture:

* MobileNetV2 (frozen, include\_top=False): The baseline model is MobileNetV2, which is a pre-trained network and can be used in general image classification applications. The model is "frozen" so that the pre-trained weights of the convolutional layers remain unchanged during training. The include\_top=False argument will make use of only the convolutional part of the model without using the last fully connected layers. This enables the model to focus on feature extraction using the knowledge gained from a huge dataset like ImageNet.
* GlobalAveragePooling2D → Dense (128, relu) → Dropout (0.5) → Output Dense (10, softmax): GlobalAveragePooling2D: This reduces the spatial dimension of the feature maps to a single average value per feature map, significantly reducing the number of parameters.
* Dense (128, relu): The input is then passed to a fully connected layer of 128 neurons with ReLU activation to add non-linearity.
* Dropout (0.5): Dropout is applied with a dropout rate of 0.5 for regularization against overfitting by randomly setting half of the neurons to zero during training, which forces the model to learn more stable features.
* Output Dense (10, softmax): The output layer consists of 10 units (for a 10-class classification problem) and uses softmax activation to output probabilities for each class.



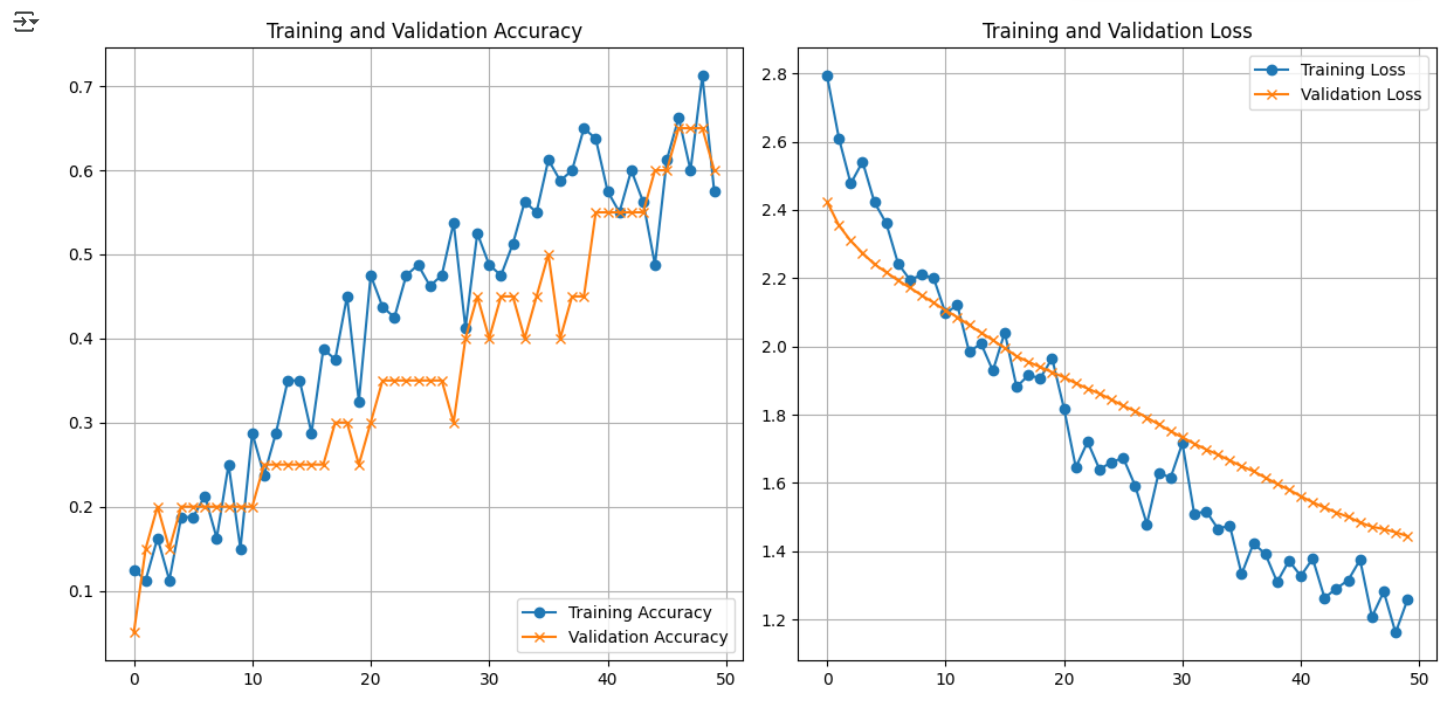
Parameters:

* Optimizer (Adam (lr=1e-4)): Adam is applied with a learning rate of 1×, Which is relatively low, so that sudden changes in the weights during training are avoided, allowing the model to converge smoother and avoid overfitting.
* Epochs (50): The model was trained for 50 epochs, providing more time for the model to learn and enhance its performance if possible.



Training Results:

* Accuracy: Training Accuracy of this model is 63%. The model performs decently on the training set but doesn’t achieve perfect accuracy, which is a sign that it's not overfitting the data too much. Validation Accuracy of this model is 65%. The performance on the validation set is slightly better than the training accuracy, indicating that the model generalizes well and doesn’t just memorize the training data.
* Validation Loss: The validation loss gradually reduced, but it fluctuated during training, which suggests some instability in the learning process. This can be due to a variety of reasons, including noisy data, learning rate settings.



Observation:

* No Overfitting: Unlike Model 1, Model 2 did not suffer from the problem of overfitting. The training accuracy and validation accuracy were nearer to one another, which indicated that the model generalized better. Transfer learning with MobileNetV2 being the base avoided overfitting through the utilization of pre-trained features.
* Generalization Improved: The MobileNetV2 model with transfer learning leveraged pre-learned features from a massive, diverse dataset (ImageNet) and applied them to the task in question. This likely helped it generalize better to new data.
* Accuracy Plateaued (60–65%): Despite not overfitting, the model did plateau around 60-65% accuracy. Perhaps because the initial base MobileNetV2 model had been frozen and not fine-tuned, and so was limited to only picking up on whatever the fixed model learned about the distinguishing characteristics of the current dataset.
* Training Curve Is Noisy: The noisy training curve implies that perhaps the model is underfitting or perhaps there's some imbalance in the data. The fluctuation in loss also means that the model learning process was not smooth, perhaps due to reasons such as the wrong learning rate or unbalanced classes.

Suggestion:

* Enable Fine-Tuning: After training for a few epochs, fine-tuning the model by unfreezing the top layers of MobileNetV2 could help improve the model’s performance. Fine-tuning would allow the model to adjust the weights of the pre-trained layers slightly to better adapt to the current dataset, leading to improved accuracy.

Comparison with Model 1 (Custom-built CNN):

* Model Complexity: In Model 1 (Custom CNN), a simple model made up of two convolutional layers followed by dense layers. The model was trained from scratch, and the weights were all initialized randomly. It had to learn all the features from the training dataset. It thus overfitted and had poor generalization due to its simplicity and lack of data augmentation or regularization. In Model 2 (Transfer Learning with MobileNetV2), MobileNetV2 is a more complex and pre-trained model that benefits from learned features in a large dataset (ImageNet). Freezing the base model and fine-tuning after several epochs helps generalization, preventing overfitting. The transfer learning approach likely helped the model perform better on validation data.
* Overfitting: In Model 1, overfit, as shown by the perfect training accuracy and the immense gap between training and validation performance. In Model 2, prevented overfitting due to the pre-trained weights and frozen base model, but did suffer from some loss curve instability.
* Performance: Model 1 experienced perfect training accuracy but terrible generalization, with significantly lower validation accuracy (~55%). Model 2 experienced much more stable generalization with a validation accuracy of 65%, though it plateaued at a moderate level (~60–65%).

Recommendation:

* Model 1: Needs regularization techniques (including dropout, L2 regularization) and data augmentation to improve generalization. Even the model needs a more advanced architecture or transfer learning.
* Model 2: Along with fine-tuning, performance would even be improved further. Fine-tuning of the pre-trained MobileNetV2 model would make the model capable of learning task-specific features.

## 4.3 MobileNetV2 Transfer Learning with Additional Fine-Tuned Custom Convolutional Layers

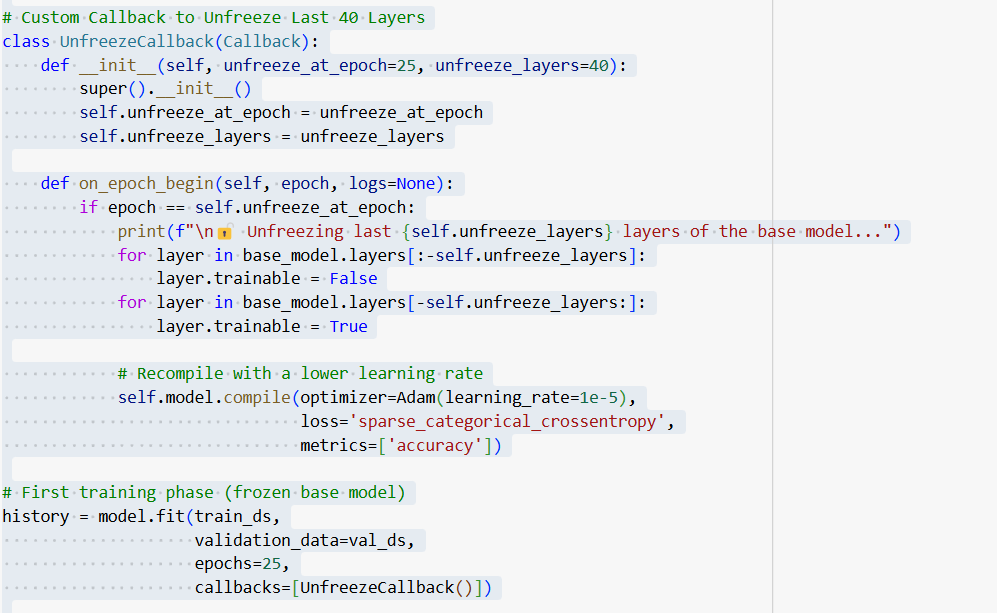
Architecture:

* MobileNetV2 (Pretrained, Partially Fine-Tuned): We begin this architecture with MobileNetV2 pre-trained on the ImageNet dataset used as a frozen feature extractor in the initial training stage. With include\_top=False, the final classification head of MobileNetV2 is removed, allowing for liberty to place custom layers on top. The model is enabled by transfer learning, leveraging rich low-level and mid-level image features learned from a massive dataset. This is used most effectively with the use of small or less diverse datasets. During the 25th epoch, the top 40 layers of MobileNetV2 were frozen and adapted to improve the ability of the model to learn the high-level representations to suit the specific target dataset.
* Custom Convolutional Layers: Conv2D (64, relu) → MaxPooling2D, adds a 64-filter convolutional layer with ReLU activation to learn dataset-specific features. MaxPooling reduces the spatial dimensions by downsampling, improving computational efficiency and robustness. Conv2D (32, relu) → MaxPooling2D, the second convolutional layer with fewer filters fine-tunes learned features and reduces the risk of overfitting via abstraction and pooling. These additional layers enhance the model's capacity to handle high-level target task-specific representations.
* Classification Head (Final Component That Makes the Prediction): GlobalAveragePooling2D takes every feature map and reduces it to a single value (average value). Reduces parameters, which makes the model smaller and less likely to overfit. Dense (128, relu) is a fully connected layer of 128 neurons. Adds more learning capacity so that the model can mix features and produce decision boundaries. Dropout (0.5) happens during training, it randomly "drops out" 50% of the neurons in the previous layer. This prevents overfitting by ensuring the model doesn't rely too much on any single neuron. Dense (10, softmax) is the last output layer with 10 neurons, one for each class. Softmax ensures outputs add up to 1 and can be interpreted as probabilities for each class.

Training Setup (How the Model Was Trained):

* Optimizer: Adam optimizer was used. It’s an adaptive optimizer that adjusts the learning rate during training automatically, making training more efficient.
* Learning Rate: Start with 0.0001 (1e-4) for the first 25 epochs. Then reduce to 0.00001 (1e-5) when fine-tuning. A lower learning rate prevents the pretrained weights from being changed too drastically during fine-tuning.
* Epochs: Total 50 epochs that 25 epochs with frozen MobileNetV2 and 25 more epochs with the last 40 layers of MobileNetV2 unfrozen for fine-tuning
* Loss Function: Sparse Categorical Crossentropy is used because you're dealing with multiclass classification and your labels are probably in integer form (not one-hot encoded).
* Metrics: Accuracy is the key performance metric during training.







Training Results:

* Accuracy (After Full 50 Epochs):Training Accuracy is 76% and validation Accuracy is 75%. This model shows a significant improvement over previous versions, achieving the highest validation accuracy so far and suggesting a robust generalization to unseen data.
* Loss Curves: Initially high training and validation loss gradually decreased. After fine-tuning, both training and validation loss showed steady decline, with minimal divergence. No signs of overfitting observed—validation accuracy improved after unfreezing, proving the value of fine-tuning.

Observations:

* Top Performance Among All Models: The model achieved 75% validation accuracy, overtaking previous architectures with the compounded strength of transfer learning combined with fine-tuned flexibility.
* Fine-Tuning Enhanced Learning: Fine-tuning and unfreezing the last 40 layers helped the model learn to map generic ImageNet features to specific domain, leading to enhanced accuracy as well as convergence stability.
* Deeper Customization Translates to Smarter Features: Custom convolutional layers added after the pretrained base allowed the network to acquire domain-specific hierarchies of features while being able to draw on pretrained knowledge.
* Training Stability: The accuracy and loss curves exhibited smooth convergence. While early epochs were of low accuracy, fine-tuning significantly enhanced performance, and the model stabilized with little oscillation.

Potential Improvements:

* Learning Rate Scheduling: A dynamic schedule would further enhance stability during fine-tuning.
* Unfreezing More Layers: As long as overfitting is avoided, unfreezing more than 40 layers may improve learning.

Comparison with Previous Models:

* Model Complexity: Model 1 was a purely custom CNN with a simple architecture—two convolutional layers followed by dense layers. Since it had no prior learning, it had to learn all visual patterns from scratch. This tends to lead to poor generalization, especially on small or less diverse datasets, since it easily memorizes training data but does not do well on new data. Model 2 used MobileNetV2 as a frozen backbone, i.e., its weights were not changed. This strong backbone offered decent feature extraction from prior learning on the ImageNet dataset. However, since no extra custom convolutional layers were added, its ability to specialize in new data was limited to the small dense layers appended on top. Model 3 expanded Model 2 by incorporating additional convolutional layers after MobileNetV2. This hybrid structure enabled the model to both reuse general features learned from ImageNet and learn new, task-specific patterns, making it more flexible and expressive.
* Transfer Learning: Model 1 did not use transfer learning, so it started from scratch—learning all patterns from the beginning. This led to inefficient training that was prone to overfitting. Model 2 and Model 3 each leveraged transfer learning by employing the pre-trained MobileNetV2. This gave them a good initial foothold in identifying basic image features. Model 3 took this a step further, however, by fine-tuning the top layers of MobileNetV2 and adding more convolutional blocks of its own. This allowed it to not just inherit pre-learned features, but also specialize them for the specific dataset—further improving adaptability and performance.
* Training and Validation Accuracy: Model 1 got a perfect 100% train accuracy but only 55% validation accuracy, a clear sign of overfitting and lack of generalization. Model 2 performed better in this aspect with 63% train accuracy and 65% validation accuracy, suggesting better generalization but still not by much. Model 3 worked best, with 76% train accuracy and 75% validation accuracy. This indicates the model learned beneficial patterns without overfitting, thanks to the simplicity of its architecture and the longer training.
* Loss Curve and Training Stability: The loss curve of Model 1 diverged after several epochs—training loss crashed while validation loss increased, the typical sign of overfitting and unstable training. Model 2's curve was more stable, albeit with minor fluctuations likely due to the frozen base's reduced flexibility. Model 3's loss curve was most stable, with both validation and training loss consistently decreasing. The curve was even smoother after fine-tuning commenced, with good convergence and successful learning.
* Overfitting: Model 1 experienced severe overfitting due to its vanilla architecture and lack of regularization or transfer learning. Model 2 avoided hard overfitting thanks to the frozen MobileNetV2 but nevertheless demonstrated some instability in performance. Model 3 demonstrated minimal overfitting, benefiting from both transfer learning and additional custom layers. Dropout and extended training provided harmony between learning and generalization.
* Generalization Ability: Model 1 generalized very badly; it memorized the training data but couldn't make good predictions with new data. Model 2 generalization was moderate, as it leveraged pre-trained features but without any flexibility to fine-tune them. Model 3 generalization was the best as it balanced pretrained knowledge and task-specific learning, resulting in a model that could generalize well to unseen examples.
* Training Duration (Epochs): Model 1have 10epochs and Model 2 were trained for 50 epochs, which wasn't long enough for Model 2 to adapt fully or Model 1 to generalize well. Model 3 was trained for 50 epochs in total, 25 with frozen layers and 25 with fine-tuning, giving it more time to stabilize and improve performance.

# Chapter 5: Model Testing





# Chapter 6: Conclusion

This project successfully demonstrated the power of transfer learning and model fine-tuning in achieving quality image classification on a relatively small dataset.

We began with our own CNN model (Model 1), and it overfitted early achieving high training accuracy but low validation accuracy. This highlighted the limitation of training from scratch using little prior knowledge, especially when working with sparse data.

Then, Model 2 introduced MobileNetV2 as a frozen feature extractor and experienced improved generalization and stable training. Nevertheless, the lack of ability to adjust the frozen base to the target dataset limited its maximum potential.

Finally, Model 3 gave us the best of both worlds—pretrained MobileNetV2 knowledge, task-specific custom convolutional layers, and fine-tuning the top 40 layers. This gave us the highest performance with the best validation accuracy (75%) and yet was stable, and overfitting was minimized.

Major takeaways:

* Transfer learning improves performance and convergence substantially, especially when data is limited.
* Fine-tuning existing pretrained models, even partially, allows them to adapt to new tasks better.
* The utilization of pretrained architecture with custom layers creates a more expressive and versatile model.
* A good training schedule like learning rate scheduling, regularization (e.g., dropout), and staged training is responsible for the success of the model.

Overall, this project explicitly demonstrates a trend of improving model performance and robustness, with a well-conceived and well-executed deep learning pipeline. The final model not only generalizes well but also demonstrates a practical application of state-of-the-art neural network methods.

**Appendices**

[**https://drive.google.com/drive/folders/1hlQ462G7K3ThLDolieP3OJ6QJa0anzFo**](https://drive.google.com/drive/folders/1hlQ462G7K3ThLDolieP3OJ6QJa0anzFo)

[**https://drive.google.com/drive/folders/17Gr3W7PGuuRLjwIW25iN26MVAVSz03sG**](https://drive.google.com/drive/folders/17Gr3W7PGuuRLjwIW25iN26MVAVSz03sG)