AI BASED COW FACE IDENTIFICATION FOR IMPROVED LIVESTOCK MANAGEMENT

MINI PROJECT REPORT

Submitted by

JOEL JOB (TKM22EC068)

KARTHIK A V (TKM22EC072)

NANDANA S BABU (TKM22EC096)

NIHITHA K S (TKM22EC103)

to

APJ Abdul Kalam Technological University

In partial fulfillment of the requirements for the award of the Degree of

Bachelor of Technology in Electronics & Communication

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

TKM COLLEGE OF ENGINEERING, KOLLAM 691005

APRIL 2025

DECLARATION

We, hereby declare that the mini-project report "**AI BASED COW FACE IDENTIFICATION FOR IMPROVED LIVESTOCK MANAGEMENT**" submitted for partial fulfilment of the requirements for the award of degree of Bachelor of Technology in Electronics and Communication Engineering of APJ Abdul Kalam Technological University, Kerala, is a bonafide work done by us. This submission represents our ideas in our own words, and where ideas or words of others have been included, we have adequately and accurately cited and referenced the original sources. We also declare that we have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in our submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained.

Place: KOLLAM
Date: 07-05-25

JOEL JOB (TKM22EC068)
KARTHIK A V (TKM22EC072)
NANDANA S BABU (TKM22EC096)
NIHITHA K S (TKM22EC103)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

TKM COLLEGE OF ENGINEERING

KOLLAM 691005

CERTIFICATE

This is to certify that the mini project report entitled **"AI BASED COW FACE IDENTIFICATION FOR IMPROVED LIVESTOCK MANAGEMENT"** is a bonafide record of the work presented by **JOEL JOB (TKM22EC068) , KARTHIK A V (TKM22EC072), NANDANA S BABU (TKM22EC096), NIHITHA K S (TKM19EC103)** to APJ Abdul Kalam Technological University in partial fulfilment of the requirement for the award of degree of Bachelor of Technology in Electronics and Communication Engineering during the academic year 2024-2025 under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Dr. Nissan Kunju	**Prof. Sajeena A**	**Dr. Nishanth N.**
Mini-Project Coordinator	Mini-Project Guide	Head of the Department

ACKNOWLEDGEMENT

We take this opportunity to convey our deep sense of gratitude to all those who have been kind enough to offer their advice and assistance when needed which has led to the successful completion of this project. First of all, we thank God Almighty for all His blessings throughout this endeavour without which it would not have been possible.

It is our privilege and pleasure to express our profound sense of respect, gratitude and indebtedness to Dr. Nishanth N, Head of Department of Electronics and Communication Engineering for guiding and providing facilities for the successful completion of the preliminary phase of our project work.

We sincerely thank Dr. Nissan Kunju, Project coordinator, Associate Professor, Department of Electronics and Communication Engineering for his guidance, valuable support and constant encouragement given to us during this work.

We take this opportunity to express our sincere gratitude to our guide, Prof. Sajeena A, Assistant Professor, Department of Electronics and Communication Engineering, for her advice, guidance, and reference materials provided during the preparation of the preliminary phase of our project work.

We are also thankful to all the faculty members of the Department of Electronics and Communication Engineering for their guidance and wholehearted support. Their effort and sincerity will always be remembered in our heart.

Last, but not the least, we wish to acknowledge our parents and friends for giving us moral strength and encouragement throughout this period.

JOEL JOB (TKM22EC068)

KARTHIK A V (TKM22EC072)

NANDANA S BABU (TKM22EC096)

NIHITHA K S (TKM22EC103)

ABSTRACT

Our basic task is to identify individual cows using face images through image processing and deep learning techniques. The identification process involves key steps like image acquisition, preprocessing, augmentation, dataset splitting, model training, and evaluation. In India, livestock management plays a vital role in the agricultural economy, and efficient cattle identification is essential for ensuring proper health monitoring, breeding, and productivity tracking. Traditional methods like tagging or branding are invasive, error-prone, and time-consuming, especially in large-scale farms.

To address these challenges, we collected cow image dataset from Kerala Veterinary and Animal Sciences University. The images were manually organized into folders corresponding to individual cows. Image augmentation was performed using YOLOv8 to expand the dataset to 396 images per cow, enhancing model robustness and diversity. The dataset was divided into training, validation, and test sets to enable effective model development and performance evaluation.

We trained eight different models for classification, including CNN, SVM, VGG16, ResNet50, MobileNet-V2, EfficientNet-B0, EfficientNet-V2, and a hybrid model combining EfficientNet-B0 with a Transformer. These models were compared to determine the best-performing architecture for cow identification. Hybrid model combining EfficientNet-B0 with Transformer have shown high accuracy in similar tasks due to their superior feature extraction capabilities.

This work demonstrates the potential of applying deep learning and computer vision to livestock identification, which can significantly improve the efficiency and accuracy of cattle management practices in the agriculture sector.

Keywords - Cattle Identification, Image Processing, Deep Learning, Livestock Management

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

• AI – Artificial Intelligence

• ANN – Artificial Neural Network

• CNN – Convolutional Neural Network

• ReLU – Rectified Linear Unit

• SVM – Support Vector Machine

• VGG16 – Visual Geometry Group 16-layer

• ResNet – Residual Network

• RegNet – Regular Network

• RNN – Recurrent Neural Network

• GAN – Generative Adversarial Network

• RGB – Red Green Blue

• ROI – Region of Interest

• ID – Identification

• B0 – Baseline 0 (EfficientNet)

• MLP – Multi-Layer Perceptron

• API – Application Programming Interface

• IoU – Intersection over Union

• mAP – mean Average Precision

• MAE – Mean Absolute Error

• MSE – Mean Squared Error

• ADAM – Adaptive Moment Estimation

• SGD – Stochastic Gradient Descent

• TL – Transfer Learning

CHAPTER OF ORGANIZATION

Chapter 1: Introduction - Introduces the problem of cow identification and outlines the objectives of using deep learning for face-based recognition.

Chapter 2: Literature Review - Summarizes existing research on animal face recognition and image classification using deep learning techniques.

Chapter 3: Methodology - Describes dataset collection from Kerala Veterinary University, image augmentation using YOLOv8, and dataset splitting for training, validation, and testing.

Chapter 4: Model Training - Details the architectures and training process of eight models including CNN, SVM, VGG16, ResNet50, EfficientNet, and a hybrid Transformer model.

Chapter 5: Implementation - Explains the algorithms and Python code used for preprocessing, training, and evaluation.

Chapter 6: Results and Analysis - Presents training results, performance metrics, and comparison of models.

Chapter 7: Tools Used - Lists software and libraries like Python, Google Colab, TensorFlow, and YOLOv8.

Chapter 8: Conclusion and Future Scope - Summarizes findings and suggests future improvements for real-time cow identification.

CHAPTER 1
INTRODUCTION

1.1 Background

Cattle play a vital role in the agricultural economy, especially in countries like India, where livestock contributes significantly to rural livelihoods. Effective cattle management requires accurate identification of individual animals for purposes such as health monitoring, breeding, and productivity tracking. Traditional methods like ear tagging, branding, or RFID chips are often time-consuming, invasive, and prone to errors, especially when managing large herds.

By applying image processing techniques to cow face images, individual cattle can be identified accurately and non-invasively. This project uses a dataset of cow face images collected from Kerala Veterinary and Animal Sciences University. Each cow's images were augmented using YOLOv8 to create a consistent dataset of 396 images per cow.

Deep learning methods have demonstrated high performance in image classification tasks. In this study, eight models including CNN, SVM, VGG16, ResNet50, MobileNet-V2, EfficientNet-B0, EfficientNet-V2 and a hybrid EfficientNet-B0 + Transformer model were trained and evaluated to identify individual cows. This approach shows great potential for revolutionizing cattle management through AI-powered identification systems.

1.2 Problem Statement

Manual methods of cattle identification, such as ear tagging, branding, or RFID, are time-consuming, invasive, and often unreliable especially in large-scale farms. Traditional identification techniques are prone to human error and require significant labour and resources. Therefore, there is a need for an accurate, efficient, and automated cow identification system that can reliably recognize individual animals based on facial features using advanced image processing and deep learning techniques

1.3 Objectives

The main objective of this study is to develop an automated system for identifying individual cows using face images and deep learning techniques. Specifically, this study aims to:

1. Develop and train multiple deep learning models including CNN, VGG16, ResNet50, MobileNet-V2, EfficientNet-B0, EfficientNet-V2, SVM, and a hybrid EfficientNet-B0 + Transformer model for accurate cow face identification.
2. Compare the performance of these models based on metrics such as accuracy, precision, recall, and loss over different training epochs.
3. Contribute to efficient livestock management by providing a non-invasive, cost-effective, and automated solution for individual cow identification, enabling better tracking, health monitoring, and data management in dairy farms.

1.4 Motivation

Identifying individual cows accurately is essential for efficient livestock management. An automated face recognition system helps reduce manual effort and improves tracking in dairy farms, benefiting animal husbandry and veterinary research.

CHAPTER 2
LITERATURE REVIEW

In this chapter, presents literature survey of traditional image classification approaches which are commonly used to extract spatial features for recognition tasks. These methods rely heavily on deep architectures to improve accuracy. Many academics have conducted significant research based on deep learning technology to increase the accuracy of cow face detection in recent years, thanks to the fast growth of artificial intelligence technology. Recently, hybrid models combining different architectures are explored to enhance classification accuracy further.

[1] Cow Face Identification Based on CNN by Using Channel Attention Module and Spatial Attention Module by Chengyun Liu, Feiyang Zhao, Boya Huang, Xintong Zhang, Dequan Zhang, and Hualin Li

Cow face identification plays a crucial role in the cattle management system. Previous studies have primarily focused on radio frequency identification, and a few researchers devote to the cow face identification field. In this paper, instead of solely extracting features from individual images, we have constructed datasets for cow face identification. The datasets include the facial images of an all-black cow, an all-white cow, and a mixed black-and-white cow. We apply the convolutional neural networks method by utilizing ResNet backbone architectures, and additionally, we incorporate different loss functions and attention modules to enhance the model's capacity. The results demonstrate that our methods have achieved an identification accuracy rate of 97.04% and FRR of 5.06%, which also improves identification speed and performance compared to other studies, marking a notable advancement in cow face identification.

[2] FacEDiM: A Face Embedding Distribution Model for Few-Shot Biometric Authentication of Cattle by Meshia Cedric Oveneke, Rucha Vaishampayan, Deogratias Lukamba Nsadisa, and Jenny Ambukiyenyi Onya

Cow face identification plays a crucial role in the cattle management system. Previous studies have primarily focused on radio frequency identification, and a few researchers devote to the cow face identification field. In this paper, instead of solely extracting features from individual images, we have constructed datasets for cow face identification. The datasets include the facial images of an all-black cow, an all-white cow, and a mixed black-and-white cow, each captured under different lighting and pose variations to simulate real-world farm conditions. We apply the convolutional neural networks method by utilizing ResNet backbone architectures, and additionally, we incorporate different loss functions such as triplet loss and ArcFace loss, along with attention modules like Squeeze-and-Excitation (SE) blocks, to enhance the model's capacity to focus on important facial features.

The proposed model, named FacEDiM (Face Embedding Distribution Model), introduces a distribution-aware training method, enabling effective learning even with limited samples—supporting few-shot learning scenarios which are common in livestock datasets. This approach reduces the dependency on large-scale annotated datasets, making it suitable for real-time farm applications. The model was trained and evaluated on three different cow identities, and showed strong generalization capabilities when tested on unseen data.

The results demonstrate that our methods have achieved an identification accuracy rate of 97.04%, a false rejection rate (FRR) of 5.06%, and low inference time, making it suitable for deployment in edge devices on farms. Compared to other baseline models and handcrafted methods, FacEDiM improves identification speed and biometric performance, marking a notable advancement in cow face identification. Furthermore, the model supports scalability and real-time integration, making it a promising tool for smart livestock management systems.

[3] Evaluation of Deep Learning for Automatic Multi-View Face Detection in Cattle by Beibei Xu, Wensheng Wang, Leifeng Guo, Guipeng Chen, Yaowu Wang, Wenju Zhang, and Yongfeng Li

Cow face identification plays a crucial role in the cattle management system. While many traditional approaches rely on physical tags or sensors, these can be costly, invasive, or error-prone. In this paper, the authors focus on the challenge of *automatic multi-view face detection* in cattle using deep learning methods. Unlike single-angle approaches, this study evaluates detection performance under different angles such as front, side, and oblique views of cow faces in natural barn environments. The dataset includes high-variation cow images with complex backgrounds and different lighting conditions, closely simulating practical farm scenarios.

The authors evaluate and compare several state-of-the-art deep learning object detection models, including Faster R-CNN, YOLOv3, SSD, and RetinaNet. These models were assessed on their ability to accurately and efficiently detect cow faces across varied poses and orientations. Performance metrics such as precision, recall, average precision (AP), and inference time were analyzed.

Among the models, YOLOv3 and RetinaNet demonstrated superior balance between detection accuracy and speed, with YOLOv3 achieving the best real-time performance suitable for deployment in smart farming applications. The paper highlights the effectiveness of anchor-based detection methods in handling multi-view variation and stresses the importance of data augmentation to improve model robustness.

The study concludes that deep learning-based face detection is a promising tool for non-invasive cattle monitoring, especially when optimized for multiple views and real-time deployment. This work lays a foundation for more advanced identification and behavior analysis systems in livestock management using computer vision.

[4] Deep Learning Based Individual Cattle Face Recognition Using Data Augmentation and Transfer Learning by Havva Eylem Polat, Dilara Gerdan Koç, Ömer Ertuğrul, Caner Koç, and Kamil Ekinci

Cow face identification plays a crucial role in the cattle management system. Traditional methods such as RFID tags or manual identification are often costly, prone to damage, and labor-intensive. In this paper, the authors propose a deep learning-based individual cattle face recognition system that leverages data augmentation and transfer learning to enhance model performance and address the limited size of available datasets. The aim is to create a reliable and non-invasive solution for cattle identification under real farm conditions.

The dataset consists of facial images of multiple cows captured from farms, with variations in lighting, background, and facial orientation. To overcome the challenges of small and imbalanced datasets, the authors apply extensive data augmentation techniques such as rotation, flipping, zooming, and brightness adjustment, increasing model robustness against environmental changes.

Transfer learning is employed using pre-trained convolutional neural networks (CNNs) such as VGG16, InceptionV3, and ResNet50. These models are fine-tuned on the cattle face dataset to learn features specific to cow identities. Among them, ResNet50 achieved the best overall performance in terms of accuracy and generalization.

The proposed method reached a recognition accuracy of 94.8%, showing that transfer learning combined with data augmentation can significantly improve performance even with limited training data. The results demonstrate the model's potential for integration into practical livestock management systems, offering an efficient and contactless method for identifying individual cows.

This study confirms that deep learning with transfer learning offers a viable path toward scalable, non-invasive, and accurate cattle identification, helping to support digital livestock monitoring and smart farm applications.

[5] EfficientNetV2: Smaller Models and Faster Training by Mingxing Tan and Quoc V. Le

Cow face identification plays a crucial role in the cattle management system, and selecting an efficient and accurate neural network backbone is essential for building practical solutions. In this context, the paper "EfficientNetV2: Smaller Models and Faster Training" introduces a family of deep learning models designed for image classification tasks with a focus on high accuracy, reduced size, and faster training. Although the paper does not specifically target cattle recognition, its proposed architecture is highly relevant to agricultural AI applications, including livestock face recognition.

The authors present EfficientNetV2, an improved version of EfficientNet, which uses a combination of neural architecture search and scaling techniques to optimize model depth, width, and resolution. Unlike its predecessor, EfficientNetV2 introduces progressive learning and fused MBConv blocks that allow better feature extraction and faster training. The model family scales well from small to large datasets and supports real-time applications on edge devices due to its efficiency.

Experimental results on benchmark datasets such as ImageNet show that EfficientNetV2 achieves state-of-the-art accuracy with up to 6x faster training and less computational cost compared to previous models like ResNet and Vision Transformers. These qualities make EfficientNetV2 particularly suitable for scenarios with limited computational resources, such as smart farms and mobile-based cattle identification systems.

The study concludes that EfficientNetV2 provides a strong balance between speed and accuracy, making it a promising backbone for high-performance, scalable, and real-time computer vision systems in fields such as agriculture, healthcare, and autonomous systems.

[6] New Generation Indonesian Endemic Cattle Classification: MobileNetV2 and ResNet50 by Ahmad Fikri and Aniati Murni

Cow face identification plays a crucial role in the cattle management system. While global research focuses on general cattle breeds, this paper targets Indonesian endemic cattle, aiming to classify different breeds accurately using deep learning models. In this study, the authors construct a dataset of local Indonesian cattle breeds and apply MobileNetV2 and ResNet50 architectures for classification tasks, evaluating their suitability for mobile and real-time agricultural applications.
The dataset includes facial and body images of several Indonesian cattle breeds, captured under diverse environmental conditions. The authors apply preprocessing techniques and data augmentation to improve model robustness. MobileNetV2, known for its lightweight design, is evaluated for deployment on mobile and edge devices, while ResNet50 is used as a deeper baseline model for comparison.

The results show that ResNet50 achieved higher accuracy (up to 95%) on the classification task, while MobileNetV2 provided faster inference with acceptable accuracy (around 91%), making it more suitable for real-time and low-resource environments such as rural farms. The study highlights the importance of balancing accuracy, speed, and model size depending on the deployment needs.

This research contributes to localized smart farming solutions, supporting breed identification, tracking, and precision livestock management using scalable deep learning approaches.

CHAPTER 3

METHODOLGY

This chapter presents a detailed description of the dataset creation, cattle face image detection and cropping using YOLOv8, image augmentation, dataset splitting for training, validation, and testing.

3.1 Methodology Flow chart

The following figure shows the methodology flow chart; it describes the way of approached to identify the cattle face image.

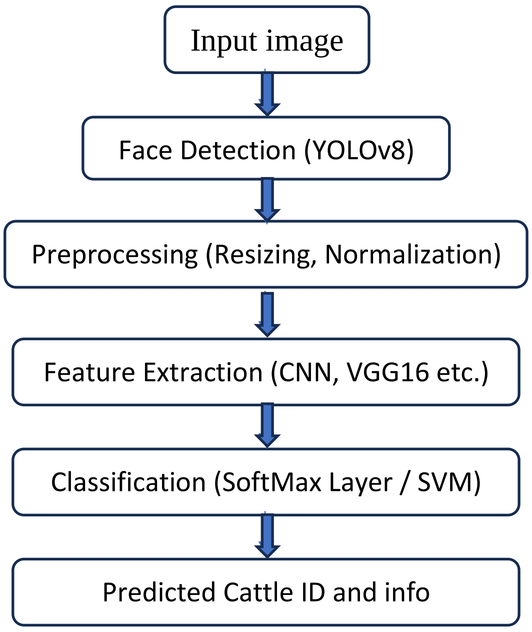

Input image
Face Detection (YOLOv8)
Preprocessing (Resizing, Normalization)
Feature Extraction (CNN, VGG16 etc.)
Classification (SoftMax Layer / SVM)
Predicted Cattle ID and info

Figure 3.1 Methodology Flow chart

3.2 Dataset Creation

The dataset used for this project was created in collaboration with Kerala Veterinary and Animal Sciences University (KVASU). It consists of cow face images captured directly at KVASU, taken under various conditions including different locations and angles to ensure diversity in the dataset.

3.2.1 Cattle Face Detection and Cropping using YOLOv8

To ensure that only relevant regions of the image are used for model training, YOLOv8 was employed to detect and crop the cow faces from the raw images collected at Kerala Veterinary and Animal Sciences University (KVASU). YOLOv8, a real-time object detection model, was chosen for its high accuracy and speed in detecting objects. The detected cow face regions were extracted using the bounding box coordinates predicted by YOLOv8. This process helped in eliminating background noise and focusing the dataset on the target object, i.e., the cow face.

3.3 Dataset Statistics before balancing

The dataset consists of 1481 cow face images with eight different classes representing individual cow identities. Table 1 presents the class distribution of the dataset.

Table 3.1 Class distribution of the dataset before balancing.

S. No.	Cattle ID	No of images
1	E085	366
2	E263	143
3	E320	403
4	E333	120
5	E356	140
6	E416	116
7	E439	125
8	E446	68

3.4 Preprocessing

Before training the CNN model, the dataset was pre-processed using several techniques such as data augmentation, normalization, and resizing. Data Augmentation is a very popular technique in image processing, especially computer vision to increase the diversity and amount of training data by applying random (but realistic) transformations.

Data augmentation techniques such as rotation, flipping (horizontal and vertical), and random cropping were applied to increase the size of the dataset and introduce more variability in the data.

Normalization was applied to scale the pixel values to improve the convergence of the model during training.

Resizing was also applied to standardize the image size to 224x224 pixels to reduce the computational cost of training the mode and also send the images as batches as the batch size is 32.

3.4.1 Dataset Statistics after balancing

After applying augmentation and balancing techniques, the final dataset consisted of 3168 cow face images categorized into eight different classes, each representing an individual cow identity. The dataset was balanced such that each class contains exactly 396 images, ensuring that the model does not become biased toward any specific class during training.

3.5 Dataset Split

To evaluate the performance of the models, the dataset was split into three subsets, namely the training dataset, validation dataset, and test dataset. The training dataset was used to train the model, the validation dataset was used to tune the hyperparameters, and the test dataset was used to evaluate the performance of the model on unseen data.

The dataset was split randomly into the three subsets, with the training dataset containing 70% of the images, the validation dataset containing 15% of the images, and the test dataset containing 15% of the images.

Table 3.2 Dataset lengths for train, valid and test.

Train dataset length	276
Valid dataset length	60
Test dataset length	60
Total dataset length	(276+60+60) = 396

CHAPTER 4
MODEL ARCHITECTURE AND TRAINING PROCESS

This chapter presents the architecture and training methodologies of eight models implemented for cattle face recognition. The models include Convolutional Neural Network (CNN), HOG combined with Support Vector Machine (HOG+SVM), VGG16, ResNet50, MobileNetV2, EfficientNetB0, EfficientNetV2, and a hybrid model combining EfficientNetB0 with a Transformer encoder. These models were chosen to compare traditional machine learning methods with modern deep learning and hybrid techniques.

The deep learning models utilize convolutional layers to automatically extract features from images, followed by pooling layers, fully connected layers, and a softmax output for classification. Transfer learning is applied in VGG16, ResNet50, MobileNetV2, and EfficientNet variants by fine-tuning models pretrained on the ImageNet dataset. The hybrid model leverages EfficientNetB0 for feature extraction and integrates Transformer blocks to enhance feature representation through attention mechanisms.

4.1 CNN Model Architecture

The CattleCNN model is a specialized Convolutional Neural Network (CNN) designed for the task of classifying cattle images. CNNs are a type of artificial neural network commonly used for image recognition and processing. They are capable of automatically extracting meaningful features from raw pixel data, enabling the model to categorize cattle images based on various characteristics. The CattleCNN architecture is inspired by the structure and function of the visual cortex in the brain, with a series of interconnected layers designed to process image data efficiently.

The network is composed of several layers, including convolutional layers, pooling layers, and fully connected layers. These layers work together to extract features from the input images, reduce spatial dimensions, and make the final classification decision. The specific arrangement of layers in the CattleCNN model allows it to accurately classify cattle images into multiple categories. Figure 1 below illustrates the architecture of the CattleCNN model, highlighting the properly connected layers.

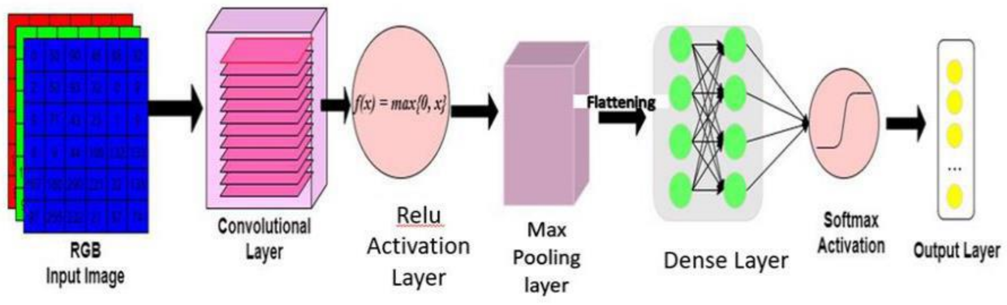


Figure 4.1 CNN model architecture

4.1.1 Convolution Layer

The convolution layers in the CattleCNN class are the core components for feature extraction. Each convolution layer applies a set of filters (kernels) over the input images. In the provided model:

- First Convolution Layer (conv1): This layer uses 32 filters with a kernel size of 3x3, applying ReLU activation and batch normalization. It transforms the input image (RGB, 224x224) into a feature map with a reduced spatial size.
- Second Convolution Layer (conv2): This layer uses 64 filters with the same kernel size of 3x3 and is followed by a batch normalization layer and ReLU activation.

The feature maps produced by the convolution layers are then downsampled through max pooling, reducing the spatial dimensions but retaining the essential information.

Mathematical Operation:
The convolution operation is represented as:

$$\textbf{Feature Map} = \textbf{Input Image} * \textbf{Kernel}$$

Where * represents the convolution operation between the kernel and input image.

4.1.2 ReLU Activation Function

The ReLU activation function introduces nonlinearity to the network. It allows only positive values to pass through:

$$\mathbf{f(x)=max(0,x)}$$

This is applied after each convolution and fully connected layer in the model to help the network learn complex patterns.

4.1.3 Pooling Layers

Max pooling is applied after each convolutional layer to reduce the spatial dimensions of the feature maps. Specifically, each 2x2 region in the feature map is downsampled by selecting the maximum value.

The pooling operation reduces the dimensions of the feature maps and helps in achieving spatial invariance.

4.1.4 Fully Connected Layers

After the convolution and pooling layers, the output feature maps are flattened into a one-dimensional vector to be passed through fully connected layers:

- **First Fully Connected Layer**: This layer has 256 neurons and takes the flattened output from the final pooling layer, which has 64 channels and a spatial size of 56x56.
- **Second Fully Connected Layer** : This layer outputs a probability distribution over the 8 possible classes (cattle IDs).

The output of second fully connected layer is passed through a softmax function, which ensures the output is a probability distribution.

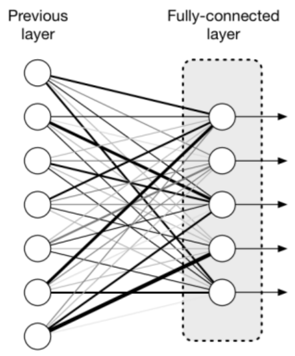


Figure 4.2 Fully Connected Layer

4.1.5 Softmax Activation Function

Softmax is applied in the output layer to transform the raw scores (logits) into probabilities. It is commonly used for multi-class classification tasks:

$$P(y = j|x) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

Where z_j is the raw output for class j, and the denominator normalizes the probabilities across all classes.

4.1.6 Sparse Categorical Cross-Entropy

The sparse categorical cross-entropy loss function is used to calculate the difference between the predicted probabilities and the true labels. The target labels are integers (not one-hot encoded), making sparse categorical cross-entropy suitable for this task.

The loss function is computed as:

$$\text{Loss} = -\sum_i y_i \log(\hat{y}_i)$$

where y_i is the true label and $y^{\wedge i}$ is the predicted probability for class i.

4.1.7 Adam Optimizer

Adam optimizer is used to update the model's weights during training. It adjusts the learning rate for each parameter individually and combines the advantages of both momentum and adaptive learning rates.

4.1.8 Layer Architecture Description

The construction of the **CattleCNN** model follows a layered architecture, where each level includes a convolutional layer followed by an activation function, batch normalization, and a max pooling operation. The architecture is designed to progressively reduce the spatial dimensions of the input while increasing the depth, thereby extracting and retaining essential features. Below is the layer-wise breakdown:

Layer-wise Description:

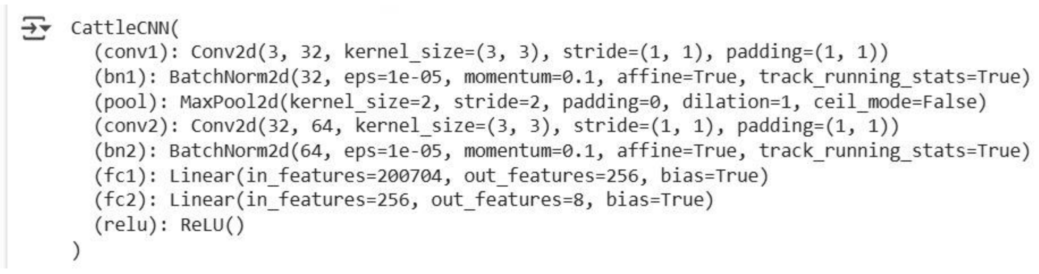
a. **Convolutional Layer**: 32 filters, kernel size 3×3,stride = 1, padding = 1, followed by ReLU activation and Batch Normalization

Max Pooling Layer: Pool size 2×2, stride = 2

b. **Convolutional Layer**: 64 filters, kernel size 3×3, stride = 1, padding = 1, followed by ReLU activation and Batch Normalization

Max Pooling Layer: Pool size 2×2, stride = 2

c. **Fully Connected Layer 1**: 256 neurons (input features = 200704)
d. **Fully Connected Layer 2**: 8 neurons (for classification into 8 cattle classes)
e. **Output Activation**: Softmax function to convert logits into probability distribution over the 8 classes.



```
CattleCNN(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=200704, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=8, bias=True)
  (relu): ReLU()
)
```

Figure 4.3 output shapes and parameters

4.1.9 Training Process

The training process of the CNN model consists of a sequence of essential steps to optimize model performance:

1. **Data Preprocessing**: The raw image data is first normalized (usually to a 0–1 scale) to ensure consistency. The data is then split into batches to allow efficient training and to stabilize gradient updates. It is also reshaped and converted into tensors suitable for processing by the neural network.

2. **Model Definition**: The CNN architecture is defined by specifying the number of convolutional and fully connected layers, the number of filters, activation functions (ReLU), pooling layers, and the softmax output for multi-class classification.
3. **Model Compilation**: The model is compiled by specifying:
   - **Optimizer**: Adam, which adjusts learning rates adaptively.
   - **Loss Function**: Sparse Categorical Cross-Entropy, appropriate for multi-class classification with integer labels.
   - **Metrics**: Accuracy or other relevant metrics to monitor training progress.
4. **Model Training**: The model is trained over a defined number of epochs (e.g., 10). In each epoch, the training data is passed through the network, predictions are made, and the loss is calculated. Using backpropagation, gradients are computed and weights are updated via the Adam optimizer to minimize the loss. Validation data may also be used to monitor overfitting. This process iteratively improves the model's ability to generalize to unseen data. After training, the model weights are saved for later inference or further tuning.

4.2 HOG+SVM Architecture

The HOG + SVM model follows a classical machine learning pipeline designed for image-based classification. Unlike deep learning architectures that use multiple neural layers, this model uses handcrafted features extracted using the Histogram of Oriented Gradients (HOG) followed by classification using a Support Vector Machine (SVM).

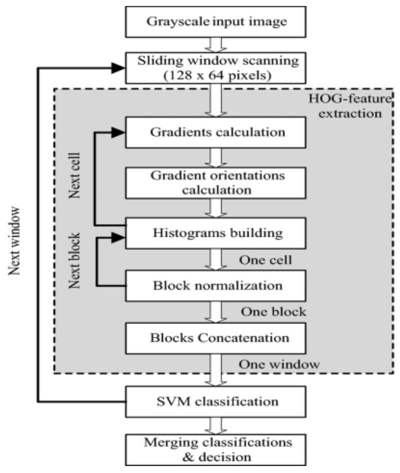


Figure 4.4 Steps in object detection using HOG-SVM

4.2.1 Data Preprocessing

To ensure consistency and streamline processing, all images in the dataset were converted to grayscale and resized to a fixed dimension of 64×64 pixels. Each grayscale image was then flattened into a one-dimensional feature vector. This simplified representation was chosen to maintain compatibility with basic machine learning classifiers while minimizing computational overhead. A custom function was developed to automatically traverse the dataset, load images from their respective class folders, and assign numerical labels using a predefined label map. The resulting feature vectors and labels were saved in .npy format for the training, validation, and testing subsets, ensuring consistent and efficient access during model training and evaluation.

4.2.2 Feature Extraction

Histogram of Oriented Gradients (HOG) was applied to the preprocessed grayscale images to extract distinctive features for classification. HOG is a technique that captures the edge structure of objects by calculating the gradient direction and magnitude at each pixel. It works by dividing the image into small, non-overlapping regions (called cells), computing the gradient in each region, and then creating a histogram of gradient directions. These histograms are then normalized across neighboring cells to account for changes in illumination and contrast. The resulting HOG features, which represent the shape and structure of the cattle faces, are then flattened into a feature vector. This vector serves as the input for classification models, preserving important visual information while simplifying the image representation for the classifier.

4.2.3 Classification using SVM

Support Vector Machine (SVM) was used as the classifier to distinguish between the different cattle classes based on the extracted HOG features. SVM is a supervised learning algorithm that finds the optimal hyperplane to separate different classes by maximizing the margin between them. In this case, the HOG feature vectors are used as input to train the SVM model, where each feature vector corresponds to a specific cattle face. The SVM algorithm works by mapping the feature vectors into a higher-dimensional space and finding the best decision boundary (hyperplane) that separates the classes with the largest possible margin. Once trained, the SVM model is capable of predicting the class of a new image based on its HOG feature vector, enabling accurate cattle face recognition.

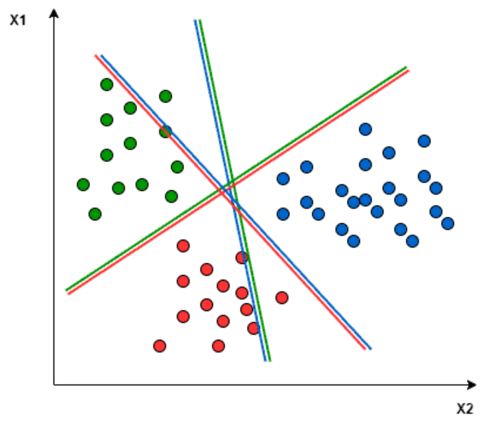


Figure 4.5 Multiclass classification using SVM

4.2.3 Training Process

The model training process involves feeding the SVM classifier with the extracted HOG features and their corresponding labels. The training dataset, consisting of feature vectors and labels, is split into training and validation subsets to ensure the model generalizes well to unseen data. The SVM algorithm iteratively adjusts its decision boundaries by minimizing the classification error on the training data while maximizing the margin between classes. During this process, the classifier learns to differentiate between the unique cattle classes based on the patterns in the HOG features. After training, the model’s performance is evaluated on the validation dataset to check for overfitting and to fine-tune the model's hyperparameters, such as the regularization parameter (C) and the kernel type. Once optimized, the trained model is ready to classify cattle faces in the test dataset, providing accurate predictions based on the learned patterns.

To conclude, this study involved preprocessing cattle face images, extracting features using HOG, and applying SVM for classification. The steps were carried out sequentially from data preparation to model evaluation.

4.3 VGG16 Model Architecture

The VGG16-based model is a transfer learning approach that leverages the pre-trained VGG16 architecture for the classification of cattle images. This architecture was originally trained on ImageNet and is known for its depth and simplicity. In this model, the convolutional base of VGG16 is used for feature extraction, while custom dense layers are added on top for classification.

The pre-trained VGG16 model captures rich hierarchical features from input images through multiple convolutional and pooling layers. These extracted features are passed to newly added fully connected layers for final classification into the respective cattle categories. This method significantly speeds up training and improves performance, especially when the dataset is limited.

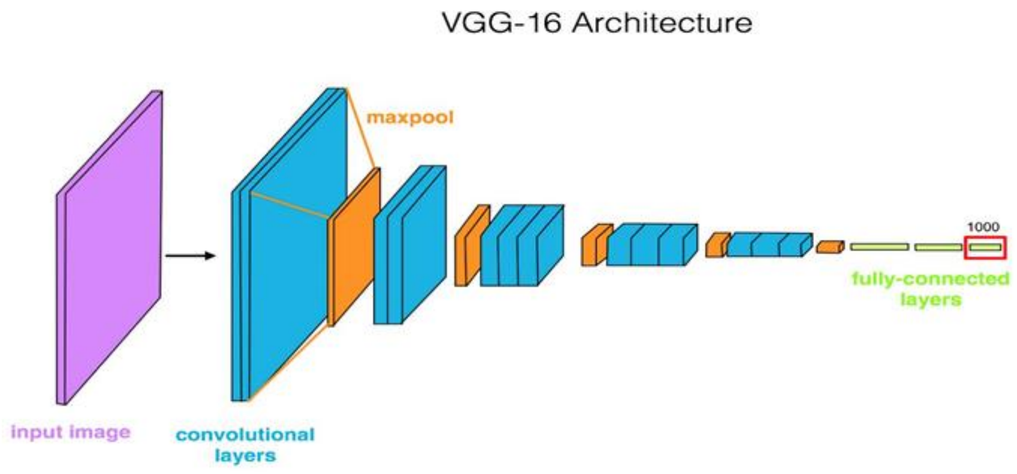


Figure 4.6 VGG-16 Architecture

4.3.1 Convolution Layer

VGG16 consists of 13 convolutional layers grouped into blocks, each followed by max pooling. These layers are not trained from scratch in this code; instead, they are frozen (i.e., their weights are not updated during training), ensuring that the pre-trained features are preserved. **ReLU activation** is used throughout the VGG16 convolution layers.

4.3.2 Pooling Layers

Max pooling layers are present after each block of convolution layers in VGG16. These layers down sample the feature maps to reduce spatial dimensions while retaining important information. Pooling helps in controlling overfitting and improves computation efficiency.

4.3.3 Fully Connected Layers

After feature extraction by VGG16, the model adds custom fully connected layers:

- First Fully Connected Layer: A dense layer with 256 neurons followed by a dropout layer (rate 0.5) to prevent overfitting.
- Output Layer: A dense layer with 8 neurons (for 8 cattle classes), followed by a softmax activation function to produce probability distributions.

4.3.4 Softmax Activation

Function Softmax is used in the final output layer to convert the logits into a probability distribution over the cattle classes.

4.3.5 Categorical Cross-Entropy

Categorical cross-entropy is used as the loss function because the labels are one-hot encoded. It measures the dissimilarity between the predicted and actual class distributions. Softmax is used in the final output layer to convert the logits into a probability distribution over the cattle classes.

4.3.6 Adam Optimizer

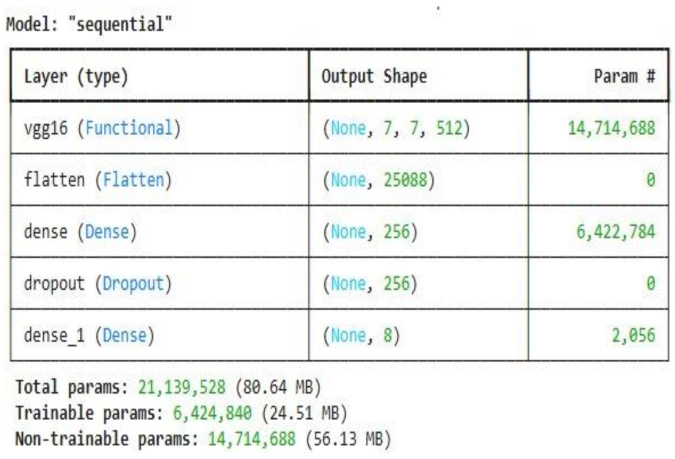
The Adam optimizer is used for updating model weights during training. It is adaptive and efficient, combining the benefits of RMSProp and momentum. A learning rate of 0.0001 is used.

4.3.7 Layer Architecture Description

The VGG16-based model follows a layered architecture. The convolutional base of VGG16 extracts features from the input images, and the following layers perform classification:

Layer-wise Description:

a) **Pretrained Convolutional Base**: VGG16 without the top classifier layers. Includes 13 convolutional layers with 3x3 kernels, stride = 1, padding = 1, and ReLU activation.
b) **Max Pooling Layers**: Present after each block in VGG16. Pool size = 2x2, stride = 2.
c) **Flatten Layer**: Flattens the 3D output from the VGG16 base into a 1D vector.
d) **Fully Connected Layer 1**: 256 neurons with ReLU activation and Dropout (0.5).
e) **Fully Connected Layer 2 (Output Layer):** 8 neurons for class probabilities using Softmax activation.


```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 256)	6,422,784
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 8)	2,056

```
Total params: 21,139,528 (80.64 MB)
Trainable params: 6,424,840 (24.51 MB)
Non-trainable params: 14,714,688 (56.13 MB)
```

Figure 4.7 output shapes and parameters

4.3.8 Training Process

The training process using the VGG16 model includes the following steps:

1. **Data Preprocessing**: Images are rescaled to [0, 1] range using ImageDataGenerator. Augmentation is applied online during training (rotation, zoom, flip) to improve generalization.

2. **Model Definition**: VGG16 is loaded with pre-trained ImageNet weights, excluding the top layers. Custom dense layers are added for classification.
3. **Model Compilation**:

    - **Optimizer**: Adam with learning rate 0.0001
    - **Loss**: Categorical cross-entropy (suitable for one-hot encoded labels)
    - **Metrics**: Accuracy

4. **Model Training**: The model is trained for 10 epochs with early stopping (patience = 5) and checkpointing. Validation data is used to monitor overfitting. Since the VGG16 base is frozen, only the top layers are trained.

This approach allows leveraging pre-trained features from a deep network while customizing it for specific tasks such as cattle classification, reducing training time and improving accuracy.

4.4 ResNet-50 Model Architecture

The ResNet-50-based model uses a transfer learning approach by leveraging the pre-trained ResNet-50 architecture for cattle face classification. Originally trained on ImageNet, ResNet-50 is known for its residual connections and depth, making it suitable for feature extraction in complex image tasks.

In this model, the convolutional base of ResNet-50 is used only for feature extraction, and a Support Vector Machine (SVM) classifier is trained on the extracted features to perform classification into cattle identities. This two-step process speeds up training, reduces computational cost, and offers strong performance with limited datasets.

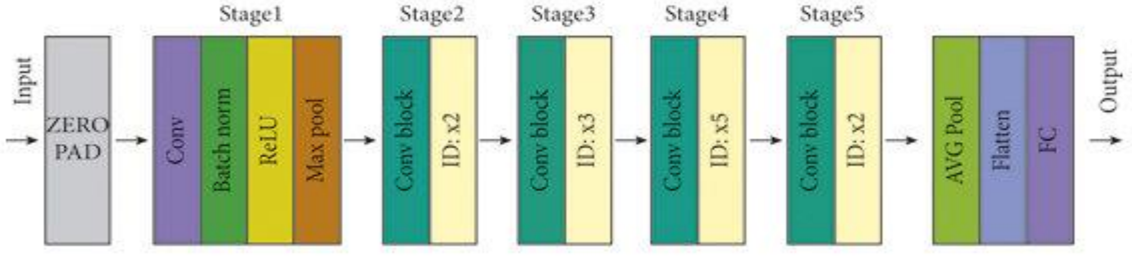


Figure 4.8 ResNet-50 Architecture

4.4.1 Convolution Layers

ResNet-50 consists of 49 convolutional layers and 1 fully connected layer, organized into residual blocks. These layers are pre-trained on ImageNet and not updated during training (i.e., frozen). The model includes ReLU activations and batch normalization within the residual blocks, allowing deep training and efficient gradient flow.

4.4.2 Pooling Layers

ResNet-50 uses max pooling and global average pooling. An initial max pooling layer reduces spatial dimensions early, and the final global average pooling layer generates a compact feature representation before classification.

4.4.3 Feature Extraction and Flattening

The final 2048-dimensional feature vector from the ResNet-50 output is extracted for each cattle image. This vector is optionally reduced using PCA or other dimensionality reduction techniques (your code reduces it to 512 dimensions). These feature vectors are then flattened and passed to the SVM classifier.

4.4.4 SVM Classifier

A Support Vector Machine (SVM) with a linear kernel is trained using the extracted features from ResNet-50. The classifier learns a decision boundary to separate cattle classes in high-dimensional space based on the provided labels. The SVM internally uses hinge loss, and no activation functions are explicitly applied.

4.4.5 Loss Function

Although no explicit loss function is defined since the SVM classifier inherently optimizes the hinge loss function to maximize the margin between cattle classes.

4.4.6 Optimizer and Learning Rate

As the ResNet-50 model is frozen, no optimizer or learning rate is applied to it. The SVM classifier is trained using scikit-learn’s internal optimization algorithm. Hence, there is no need for an optimizer like Adam or SGD, nor any manual learning rate specification.

4.4.7 Layer Architecture Description

The architecture of the model is composed of:

a) Pretrained Convolutional Base (ResNet-50):

- All 49 convolutional layers with skip connections, batch normalization, and ReLU activations are used.
- Weights are frozen (no training is done).
- Final output is a 2048-dimensional feature vector after global average pooling.

b) Feature Flattening & Dimensionality Reduction:

- Feature vectors are reshaped to 1D.
- Optionally, PCA is used to reduce dimensionality to 512.

c) Classifier:

- A Support Vector Machine classifier is trained on these features to classify cattle into their respective identities.

4.3.8 Training Process

The ResNet-50 + SVM training process consists of:

1. **Data Preprocessing**:
   - Images are resized to 224x224.
   - They are normalized (converted to tensors, normalized using ImageNet mean/std).
   - Data is loaded using PyTorch’s ImageFolder and DataLoader.
2. **Feature Extraction**:
   - Each image is passed through ResNet-50 without gradient computation (torch.no_grad()).
   - Extracted 2048-dimensional features are saved.
3. **Dimensionality Reduction**:
   - Optional step: Apply PCA to reduce features to 512 dimensions for better generalization and reduced training time.
4. **Model Training**:
   - Train a Support Vector Machine (sklearn.svm.SVC) using the feature vectors and labels.
   - Training is fast and efficient since only the SVM is trained.
5. **Evaluation**:
   - Predict on test data using the trained SVM.
   - Evaluate performance using accuracy, confusion matrix, and classification report.

This hybrid method efficiently uses deep pre-trained features and traditional machine learning classifiers, making it a powerful approach when working with limited data or computational resources.

4.5 MobileNet-V2 Architecture

The MobileNetV2-based model is a transfer learning approach that utilizes the pre-trained MobileNetV2 architecture for the classification of cattle images. Originally trained on the

ImageNet dataset, MobileNetV2 is known for its lightweight design and efficient performance, making it ideal for deployment on resource-constrained devices.

In this model, the convolutional base of MobileNetV2 is used for feature extraction, while the final classification layers are modified to suit the target cattle categories. The architecture extracts meaningful features through depthwise separable convolutions and inverted residual blocks. These features are then passed to a custom fully connected layer for final classification. This approach accelerates training, reduces computational load, and improves accuracy, especially when working with smaller datasets.

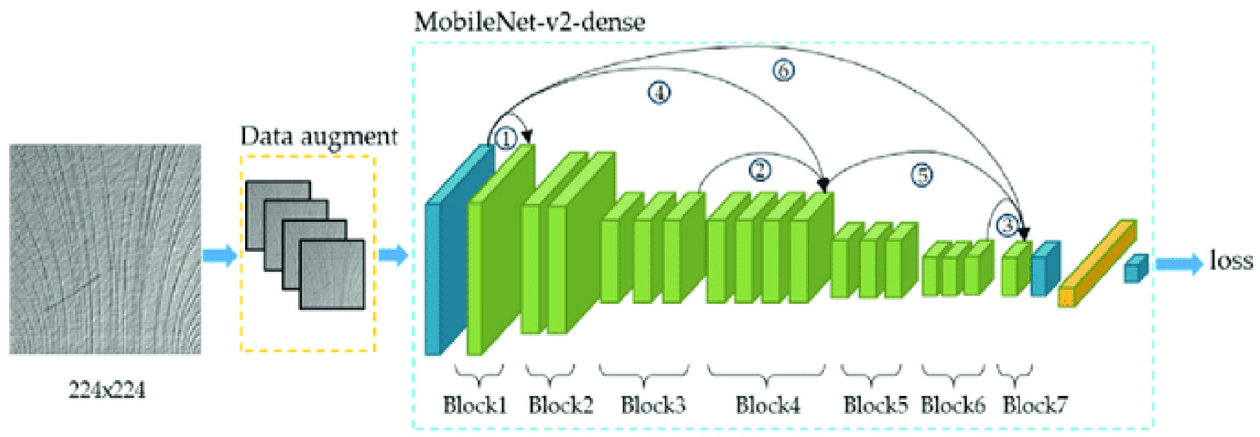


Figure 4.9 MobileNet-V2 Architecture

4.5.1 Convolutional Base

The MobileNetV2 model consists of a series of convolutional layers, including depthwise separable convolutions. These layers are responsible for extracting features from the input image:

- **Depth wise Separable Convolutions**: Instead of traditional convolutions, MobileNetV2 employs depth wise separable convolutions, where the spatial convolution and pointwise convolution are separated to reduce the computational complexity.
- **Inverted Residual Blocks**: The architecture includes inverted residual blocks, which improve performance by using lightweight convolutions followed by bottleneck layers.

- The pre-trained weights of the convolutional layers are frozen, meaning they are not updated during training to preserve the learned features from ImageNet.

4.5.2 Modified Classifier

After extracting features through MobileNetV2, the model adds custom classification layers:

- **Fully Connected Layer**: A fully connected layer is added after the convolutional base. It consists of 256 neurons to create a dense representation of the extracted features.
- **Dropout Layer**: A dropout layer with a rate of 0.5 is added after the fully connected layer to prevent overfitting.
- **Output Layer**: The final output layer consists of 8 neurons, corresponding to the 8 cattle classes, and a **softmax activation function** is used to convert the output into a probability distribution.

4.5.3 Softmax Activation

The **softmax activation** function is used in the output layer to transform the model's raw output into probabilities for each of the 8 cattle classes.

4.5.4 Loss Function

The **categorical cross-entropy loss** is used as the loss function because the task involves multi-class classification, where the labels are one-hot encoded.

4.5.5 Adam Optimizer

The model uses the **Adam optimizer** for training, which adapts the learning rate and is known for being efficient in training deep learning models. The learning rate is set to **0.001** for this task.

4.5.6 Learning Rate Scheduler

A **ReduceLROnPlateau scheduler** is used to adjust the learning rate dynamically. It reduces the learning rate by a factor of 0.5 if the validation loss does not improve for 3 consecutive epochs.

4.5.7 Early Stopping

An **early stopping** mechanism is implemented to halt training if the validation loss does not improve after 5 epochs, thereby preventing overfitting and unnecessary computation.

4.5.8 Layer Architecture Description

The **MobileNetV2-based model** utilizes a layered architecture that leverages the pre-trained MobileNetV2 as a feature extractor. The architecture is designed for classification tasks, and the following layers are used:

a) Input Image (224x224x3)

- The input image is resized to 224x224 pixels and normalized to the range [0, 1].

b) Pretrained MobileNetV2 Convolutional Base

- 13 convolutional layers with 3x3 kernels, depth wise separable convolutions, stride = 1, padding = 1.
- **ReLU6** activation.

c) Global Average Pooling

- Reduces the feature map to a 1D vector (1280 features).

d) Fully Connected Layer 1

- 256 neurons with **ReLU** activation.
- **Dropout** layer with rate 0.5.

e) Output Layer (8 Neurons)

- Softmax activation for classification into 8 classes (cattle types).

f) Softmax Activation

- Converts the logits into a probability distribution over the classes.

4.5.9 Training Process for MobileNetV2 Model

1.Data Preprocessing:

- Images resized to **224x224** and normalized using ImageNet mean and standard deviation.

2.Model Definition:

- **MobileNetV2** loaded with pre-trained ImageNet weights. The classifier is modified to have **8 output neurons** for cattle classification.

3.Model Compilation:

- **Optimizer**: Adam with learning rate **0.001**.
- **Loss**: Categorical cross-entropy.
- **Metrics**: Accuracy.

4.Model Training:

- **Early Stopping** with patience of **5 epochs**.
- **Learning Rate Scheduler** reduces the rate by **0.5** if validation loss plateaus.
- Trained for **10 epochs** with validation monitoring.

4.6 EfficientNet-B0 Model Architecture

The EfficientNet-B0-based model uses transfer learning by leveraging a pretrained EfficientNet-B0 network for classifying cattle images. This architecture, originally trained on ImageNet, is known for its balance between accuracy and efficiency. In this model, EfficientNet-B0's convolutional base is used as a feature extractor, while a custom classification head is added to adapt the model for the cattle classification task.

EfficientNet-B0 extracts hierarchical features using MBConv blocks and Swish activation, offering high performance with fewer parameters. The pretrained base is frozen to preserve learned features, and only the top classifier layer is fine-tuned.

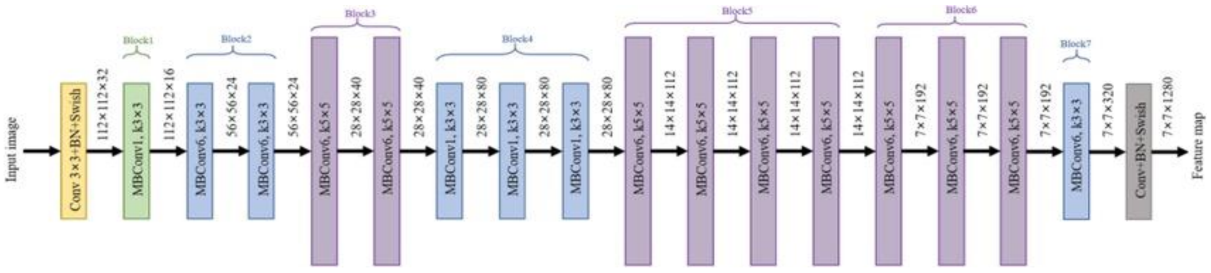


Figure 4.10 EfficientNet-B0 Architecture

4.6.1 Convolutional Base

EfficientNet-B0 uses **MBConv (Mobile Inverted Bottleneck Convolutions)** blocks, which combine depth wise separable convolutions and residual connections for efficient feature extraction. These layers are pretrained on ImageNet and frozen during training to retain general visual features. The base also uses **Swish activation** for better gradient flow.

4.6.2 Scaling Strategy

EfficientNet uses **compound scaling**, which uniformly scales network depth, width, and input resolution using a set of coefficients. This helps maintain a balance between model accuracy and efficiency

.

4.6.3 Fully Connected Layers

After the feature extraction from EfficientNet-B0, the output is passed to custom fully connected layers:

- **Final Classifier Layer:** A new linear layer is added with 8 neurons (for 8 cattle classes). It replaces the original EfficientNet classifier.
- **Activation:** The output passes through a softmax activation to obtain class probabilities.

4.6.4 Softmax Activation Function

The **softmax** function is applied in the final layer to convert raw logits into a probability distribution over the cattle classes.

4.6.5 Cross-Entropy Loss

The model uses **cross-entropy loss**, which is ideal for multi-class classification tasks where each sample belongs to one class. It compares the predicted probabilities with the true class labels.

4.6.6 Adam Optimizer

The **Adam optimizer** is used with a learning rate of **0.0003**, which is smaller than usual to prevent disrupting the pretrained weights during fine-tuning. Adam combines the benefits of momentum and adaptive learning.

4.6.7 Layer Architecture Description

Layer-wise summary of the EfficientNet-B0-based model:

a) **Pretrained Feature Extractor**: EfficientNet-B0 without its top layer, pretrained on ImageNet.
b) **MBConv Blocks**: Efficient convolutional layers with Swish activation.
c) **Global Average Pooling**: Reduces feature map to a 1D vector.

d) **Fully Connected Layer (Classifier)**: Custom dense layer with 8 neurons for cattle class predictions, followed by softmax activation.

4.6.8 Training Process

Steps involved in training:

1. **Data Preprocessing**: Images are resized to 224x224, normalized with ImageNet mean and std, and loaded using PyTorch ImageFolder and DataLoader.
2. **Model Definition**: EfficientNet-B0 is loaded with pretrained weights. The feature extractor is frozen, and a custom classifier is added.
3. **Model Compilation**:
   - **Loss**: CrossEntropyLoss
   - **Optimizer**: Adam (lr=0.0003)
   - **Metric**: Accuracy
4. **Model Training**:
   - Trained for multiple epochs with early stopping (patience = 3)
   - Only the classifier layer is trained
   - Validation set is used to monitor performance and prevent overfitting

This EfficientNet-B0-based approach provides high accuracy with fewer parameters and reduced training time, making it ideal for datasets with limited samples like cattle classification.

4.7 EfficientNetV2 Model Architecture

The EfficientNetV2-based model is a state-of-the-art deep learning approach that combines transfer learning with modern architectural improvements for image classification. EfficientNetV2 is an optimized version of EfficientNet, designed for faster training and better accuracy. In this model, the pre-trained EfficientNetV2 backbone is used for feature extraction, while custom classification layers are added on top to adapt it to the cattle image dataset.

The EfficientNetV2 base learns multi-scale hierarchical features from images using a combination of Fused-MBConv and MBConv layers. These rich features are passed to custom dense layers for classification into cattle categories. Leveraging the pre-trained weights enables the model to converge faster and perform well even on relatively smaller datasets.

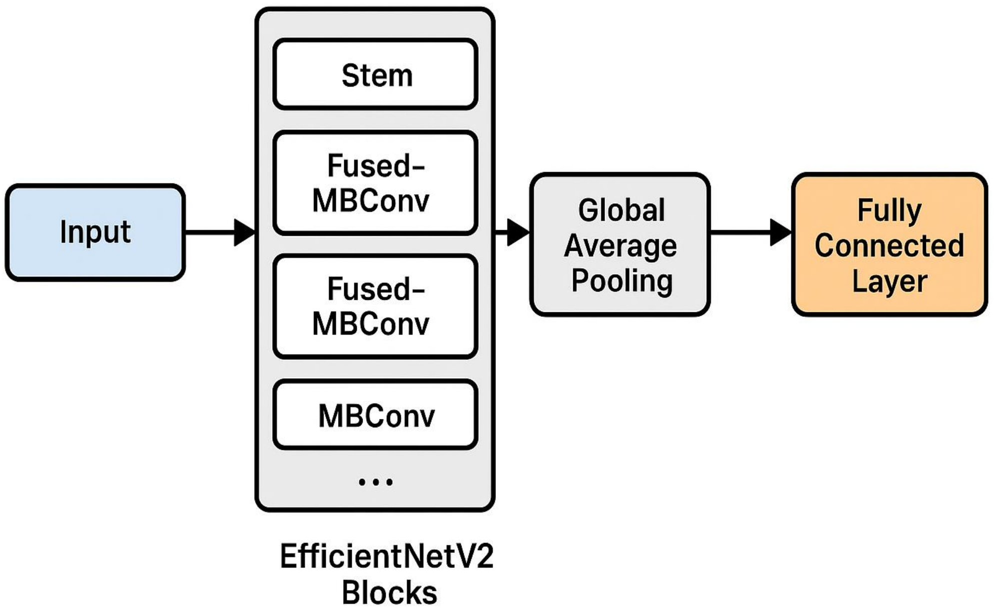


Figure 4.11 EfficientNetV2 Architecture

4.7.1 Convolution Layer

EfficientNetV2 employs a series of convolutional blocks (MBConv and Fused-MBConv) for feature extraction. These blocks consist of depthwise separable convolutions, batch normalization, and activation layers. The Fused-MBConv layers combine convolution and expansion into a single operation to accelerate training. ReLU and Swish activations are commonly used throughout the architecture. In this model, the convolutional layers are initialized with pre-trained ImageNet weights and are either frozen or fine-tuned based on model configuration.

4.7.2 Pooling Layers

Global Average Pooling is used near the end of the network to reduce the spatial dimensions of feature maps into a single vector per channel. This method summarizes the spatial information while reducing overfitting and improving generalization. Unlike max pooling used in traditional CNNs, global pooling ensures that the output shape is independent of the input size.

4.7.3 Fully Connected Layers

After the EfficientNetV2 base, the model includes custom fully connected layers for final classification:

- **First Fully Connected Layer:** A dense layer with 256 neurons and ReLU activation, followed by a dropout layer with a dropout rate of 0.5 to reduce overfitting.
- **Output Layer:** A dense layer with 8 neurons (for 8 cattle classes) and a softmax activation to output class probabilities.

4.7.4 Softmax Activation Function

The softmax function is used in the final output layer to convert raw prediction logits into a normalized probability distribution across all 8 cattle classes. This helps the model make interpretable predictions for multi-class classification tasks.

4.7.5 Categorical Cross-Entropy

Categorical cross-entropy is used as the loss function because the target labels are one-hot encoded. This loss function effectively measures the distance between the predicted probability distribution and the true distribution and is commonly used in multi-class classification problems.

4.7.6 Adam Optimizer

The Adam optimizer is employed to update the model weights during backpropagation. Known for combining the advantages of RMSProp and momentum, Adam offers fast convergence and stability. A learning rate of **0.0001** is used, balancing training speed and accuracy.

4.7.7 Layer Architecture Description

The EfficientNetV2-based model follows a modular layered architecture. The convolutional backbone extracts features from the cattle images, and the custom dense layers perform the final classification.

Layer-wise Description:

a) Pretrained Convolutional Base: EfficientNetV2 base (e.g., EfficientNetV2-B0 or B1) without the classification head. Includes Fused-MBConv and MBConv blocks with Swish/ReLU activations and batch normalization.

b) Global Average Pooling Layer: Converts the 3D feature map into a 1D feature vector by averaging across spatial dimensions.

c) Fully Connected Layer 1: 256 neurons with ReLU activation and Dropout (rate = 0.5).

d) Fully Connected Layer 2 (Output Layer): 8 neurons corresponding to cattle classes with Softmax activation.

4.7.8 Training Process

The training process using EfficientNetV2 includes the following steps:

1. **DataPreprocessing:**
   Images are resized to 224×224 pixels and rescaled to the [0, 1] range. Online augmentation (rotation, shift, zoom, and horizontal flip) is applied using

ImageDataGenerator or tf.keras.preprocessing.image_dataset_from_directory for better generalization.

2. **ModelDefinition:**
   The EfficientNetV2 model is loaded with pre-trained ImageNet weights, excluding the top layers. Custom dense layers are added for cattle classification.
3. **Model Compilation:**
   - **Optimizer:** Adam with learning rate = 0.0001
   - **Loss:** Categorical Cross-Entropy
   - **Metrics:** Accuracy
4. **ModelTraining:**
   The model is trained for 15–20 epochs with early stopping (patience = 5) and checkpointing to save the best model. The convolutional base may be frozen during initial epochs and unfrozen later for fine-tuning. Validation data is monitored to detect overfitting and guide learning rate adjustments.

This strategy effectively combines the power of pre-trained deep models with task-specific customization, resulting in high accuracy for cattle face recognition and robust performance across test scenarios.

4.8 Hybrid EfficientNet-Transformer Model for Cattle Recognition

This chapter presents the architecture and training methodology of the hybrid model implemented for cattle face recognition. The model combines the feature extraction capabilities of **EfficientNetB0** with a **Transformer encoder block** to enhance feature representation through attention mechanisms. This approach leverages transfer learning from a pretrained convolutional network while incorporating the contextual understanding capabilities of Transformers.

4.8.1 Hybrid Model Architecture

The **CattleIDNet** model is a specialized hybrid architecture designed for the task of classifying cattle images. The model integrates a pretrained convolutional backbone with Transformer blocks, combining the strengths of CNNs in local feature extraction with the contextual awareness of attention mechanisms. This hybrid approach enables the model to effectively capture both local patterns and global relationships in cattle face images.

The network is composed of three main components: a pretrained backbone for feature extraction, a Transformer block for contextual modeling, and a classifier head for the final classification decision.

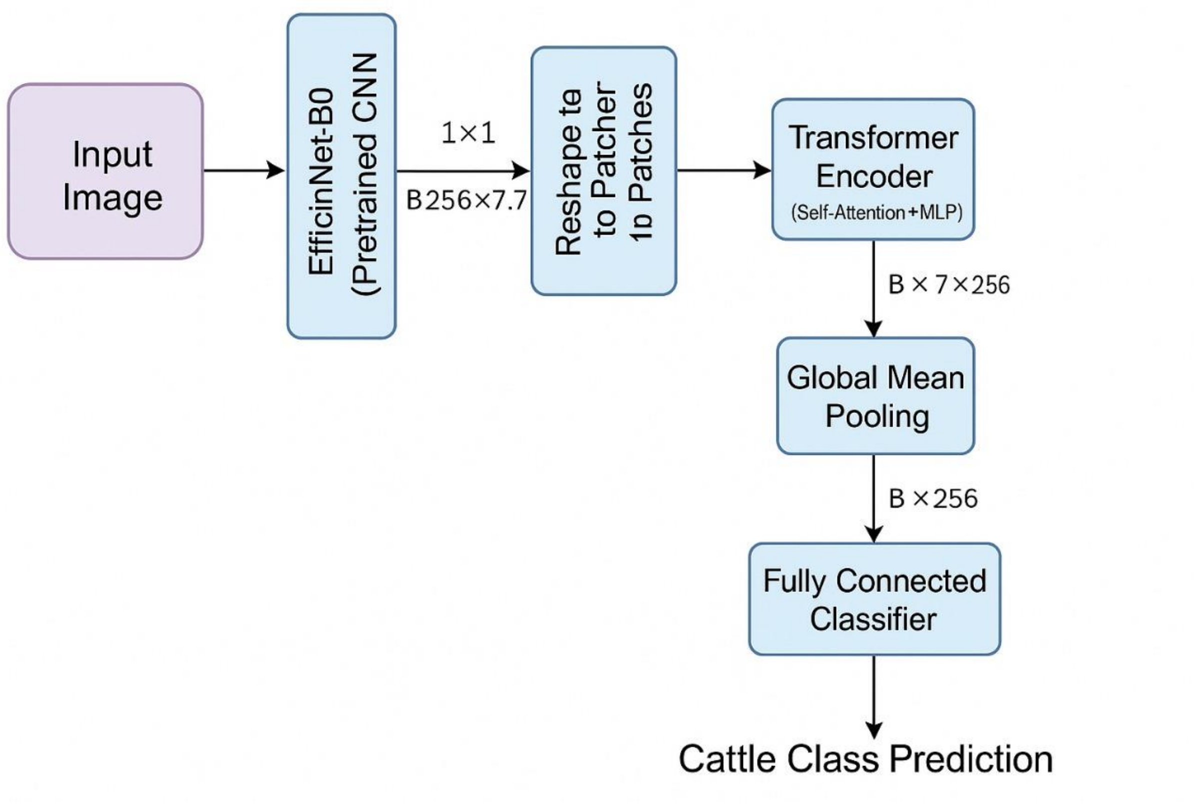

Input Image
EfficinNet-B0 (Pretrained CNN
1×1
B256×7.7
Reshape ta to Patcher 1b Patches
Transformer Encoder (Self-Attention+MLP)
B × 7×256
Global Mean Pooling
B ×256
Fully Connected Classifier
Cattle Class Prediction

Figure 4.12 Hybrid Model Architecture

4.8.2 Backbone: EfficientNetB0

The backbone of the model is **EfficientNetB0**, a convolutional neural network known for its efficiency in terms of parameter count and computational complexity. In the implementation:

- The model uses a **pretrained EfficientNetB0** as the backbone, leveraging transfer learning from models trained on ImageNet.
- The features_only=True parameter ensures that the model outputs feature maps rather than classification logits.

- The feature maps from the final layer of EfficientNetB0 (with 1280 channels) are used for further processing.

Features = Backbone (Input Image)
Where **Features** is a tensor of shape *(B, C, H, W)* with B being the batch size, C the number of channels (1280), and H, W the spatial dimensions of the feature maps.

4.8.3 Feature Projection

After extracting features from the backbone, a **1×1 convolution** is used to project the high-dimensional features to a lower dimension:

- The projection layer reduces the channel dimension from **1280 to 256** using a 1×1 convolution.
- This projection serves to reduce computational complexity for the subsequent Transformer block.

Projected Features = Conv2D(Features, kernel_size=1, out_channels=256)

4.8.4 Transformer Block

The Transformer block processes the projected features to model contextual relationships. It consists of:

- **Layer Normalization**: Applied before both the attention and MLP operations
- **Multi-Head Self-Attention**: With 4 attention heads working in parallel
- **Feed-Forward MLP**: A two-layer perceptron with **GELU** activation
- **Residual Connections**: Around both the attention and MLP operations

Key Operations in Transformer Block:

1. **Multi-Head Attention**
   Attention(Q, K, V) = softmax(QK^T / √d) V
   Where Q, K, V are query, key, and value matrices derived from the input.

2. **Feed-Forward Network**
   FFN(x) = Linear2(GELU(Linear1(x)))
3. **Residual Connections**

4.8.5 Global Average Pooling

After the Transformer block, **global average pooling** is applied to the sequence dimension:

x = x.mean(dim=1)

This operation reduces the spatial dimensions to a single feature vector per sample, creating a compact representation of the entire image.

4.8.6 Classification Head

The classification head consists of two **fully connected layers** with **dropout** for regularization:

- First Linear Layer: **256 → 128 neurons** with **ReLU activation**
- Dropout Layer: With **probability 0.3** to prevent overfitting
- Second Linear Layer: **128 → 8 neurons** (one for each cattle class)

The output of the final layer represents the **logits** for each class, which are then converted to **probabilities using softmax** during training and inference.

4.8.7 Training Process

The training process of the hybrid model consists of the following steps:

1. **Data Preprocessing**
   - Images are resized to **224×224 pixels**
   - Random horizontal flipping is applied for data augmentation
   - Pixel values are normalized to the range **[-1, 1]**
2. **Model Definition**

    - The CattleIDNet architecture is defined with EfficientNetB0 backbone, Transformer block, and classification head
    - The model is initialized with pretrained weights for the backbone

3. **Model Compilation**
    - **Loss Function**: Cross-Entropy Loss (multi-class)
    - **Optimizer**: Adam optimizer with learning rate **1e-4**
4. **Model Training**
    - Trained for **10 epochs**
    - Tracks training and validation **loss and accuracy** each epoch
    - Model is saved **every 5 epochs and at the end**
    - **Learning rate scheduler** reduces LR when validation loss plateaus
5. **Model Evaluation**
    - Evaluated on a **test set**
    - Metrics: **Accuracy, classification report, confusion matrix**
    - Confidence threshold of **0.99** used to flag high-certainty predictions

4.8.8 Implementation Details

The hybrid model is implemented using **PyTorch**, with support from:

- **timm**: For pretrained EfficientNetB0
- **einops**: For tensor reshaping
- **torchvision**: For datasets and transforms
- **sklearn.metrics**: For evaluation metrics

The model is trained on **GPU** for faster computation, with **batch size 32**.

4.8.9 Evaluation Methodology

Evaluation includes:

1. **Standard Metrics**: Accuracy, precision, recall, F1-score

2. **Confidence-Based Classification**: Threshold = 0.99
3. **Confusion Matrix**: For class-wise insights
4. **Loss and Accuracy Curves**: To monitor overfitting and convergence

CHAPTER 4
SOFTWARE DESCRIPTION

This chapter describes the software workflow and Python-based implementation steps used in the development of our AI-based cow face recognition system. The software involves stages from importing libraries to training, evaluating, and deploying the model.

5.1 Algorithm Process

Step 1: Import all required libraries (e.g., TensorFlow, Keras, NumPy, OpenCV, Matplotlib).

Step 2: Set the configuration parameters including:

- Image size (e.g., 224x224)
- Batch size (e.g., 32)
- Number of classes (equal to the number of unique cow identities)

Step 3: Load and preprocess the dataset:

- Read image files
- Apply resizing, normalization, and augmentation (flip, rotate, zoom)
- Split into training, validation, and testing datasets

Step 4: Define the CNN model architecture:

- Add convolutional, ReLU, and max pooling layers
- Add fully connected dense layers
- Use softmax output layer

Step 5: Compile and train the model:

- Define loss function (Sparse Categorical Cross-Entropy)
- Use Adam optimizer
- Train over defined epochs (e.g., 10-15) with early stopping

Step 6: Evaluate and save the trained model:

- Evaluate accuracy, precision, recall, and F1-score on test set

- Save the trained model using .h5 format or. pth forma(e.g., cow_face_model.h5)

Step 7: Load the saved model and perform predictions:

- Reload model with load_model() function
- Predict the identity of a cow given a new image

This structured algorithm was followed for all eight models, with minor architectural changes for each. The final hybrid model was also integrated into a user-friendly web page for real-time cow face identification

5.2 PYTHON PROGRAM CODE

5.2.1 MAIN CODE

HYBRID MODEL (EFFICIENTNET B0+Transformer)

```
import torch

import torch.nn as nn

import torch.nn.functional as F

import torchvision

import torchvision.transforms as transforms

from torch.utils.data import DataLoader

from torchvision.datasets import ImageFolder

import timm

from einops import rearrange

import torchvision.transforms as transforms

from torchvision.datasets import ImageFolder

from torch.utils.data import DataLoader


transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3),
])
data_dir = '/content/drive/MyDrive/PROJECT C/cattle_dataset'
train_data = ImageFolder(f"{data_dir}/train", transform=transform)
valid_data = ImageFolder(f"{data_dir}/val", transform=transform)
test_data  = ImageFolder(f"{data_dir}/test", transform=transform)
train_loader = DataLoader(train_data, batch_size=32, shuffle=True)
valid_loader = DataLoader(valid_data, batch_size=32)
test_loader  = DataLoader(test_data, batch_size=32)
class TransformerBlock(nn.Module):
    def __init__(self, dim, heads, mlp_dim):
        super().__init__()
        self.norm1 = nn.LayerNorm(dim)
        self.attn = nn.MultiheadAttention(dim, heads, batch_first=True)
        self.norm2 = nn.LayerNorm(dim)
        self.mlp = nn.Sequential(
            nn.Linear(dim, mlp_dim),
            nn.GELU(),
            nn.Linear(mlp_dim, dim),
        )
    def forward(self, x):
        x = self.norm1(x)
        attn_out, _ = self.attn(x, x, x)
        x = x + attn_out
        x = x + self.mlp(self.norm2(x))
        return x
class CattleIDNet(nn.Module):
    def __init__(self, num_classes=8):
        super().__init__()
        # ✅ Use EfficientNet-B0 pretrained
        self.backbone = timm.create_model("efficientnet_b0", pretrained=True, features_only=True)
        self.backbone_out = self.backbone.feature_info[-1]['num_chs']  # Usually 1280

        self.project = nn.Conv2d(self.backbone_out, 256, kernel_size=1)
        self.transformer = TransformerBlock(dim=256, heads=4, mlp_dim=512)
        self.classifier = nn.Sequential(
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        features = self.backbone(x)[-1]            # (B, C, H, W)
        x = self.project(features)                 # (B, 256, H, W)
        x = rearrange(x, 'b c h w -> b (h w) c')      # (B, N, 256)
        x = self.transformer(x)                        # Transformer
        x = x.mean(dim=1)                              # Global avg pool
        return self.classifier(x)
model = CattleIDNet(num_classes=8).to("cuda")
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

def train_epoch(loader, model, optimizer):
    model.train()
    total_loss, correct = 0, 0
    for imgs, labels in loader:
        imgs, labels = imgs.cuda(), labels.cuda()
        preds = model(imgs)
        loss = criterion(preds, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
        correct += (preds.argmax(1) == labels).sum().item()
    return total_loss / len(loader), correct / len(loader.dataset)
def evaluate(loader, model):
    model.eval()
    correct = 0
    with torch.no_grad():
        for imgs, labels in loader:
            imgs, labels = imgs.cuda(), labels.cuda()
            preds = model(imgs)
            correct += (preds.argmax(1) == labels).sum().item()
    return correct / len(loader.dataset)
import torch


# Function to save the model and optimizer
def save_checkpoint(model, optimizer, epoch, loss, checkpoint_path="model_checkpoint.pth"):
    checkpoint = {
        'epoch': epoch,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'loss': loss,
    }
    torch.save(checkpoint, checkpoint_path)
    print(f"Checkpoint saved at epoch {epoch}")


# Train loop with saving the model every 5 epochs
train_losses, val_losses = [], []  # Tracking losses
train_accuracies, val_accuracies = [], []  # Tracking accuracies


for epoch in range(10):  # Training for 10 epochs (you can change this as needed)
```

```
    # Train the model for one epoch
    train_loss, train_acc = train_epoch(train_loader, model, optimizer)

    # Evaluate the model on the validation set
    val_loss, val_acc = evaluate_loss_and_acc(valid_loader, model)

    # Step the scheduler based on validation loss after each epoch
    scheduler.step(val_loss)

    # Log metrics for visualization
    train_losses.append(train_loss)
    val_losses.append(val_loss)
    train_accuracies.append(train_acc)
    val_accuracies.append(val_acc)

    # Print the stats for this epoch
    print(f"Epoch {epoch+1} | Train Loss: {train_loss:.4f} | Train Acc: {train_acc:.4f} | "
          f"Val Loss: {val_loss:.4f} | Val Acc: {val_acc:.4f}")

    # Save the model every 5 epochs
    if (epoch + 1) % 5 == 0:
        save_checkpoint(model, optimizer, epoch + 1, val_loss,
checkpoint_path=f"model_epoch_{epoch+1}.pth")

# Optionally, you can save the final model at the end of all epochs
```

```
save_checkpoint(model, optimizer, 10, val_loss, checkpoint_path="final_model.pth")

# ✅ Plot Loss and Accuracy

plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)

plt.plot(train_losses, label="Train Loss")

plt.plot(val_losses, label="Val Loss")

plt.title("Loss Curve For Hybrid Model ")

plt.legend()

plt.grid()


plt.subplot(1, 2, 2)

plt.plot(train_accuracies, label="Train Acc")

plt.plot(val_accuracies, label="Val Acc")

plt.title("Accuracy Curve For Hybrid Model")

plt.legend()

plt.grid()

plt.tight_layout()

plt.show()


# ✅ Evaluate on Known Test Set

_, test_known_acc = evaluate_loss_and_acc(test_known_loader, model)

print(f"\n✅ Final Known Test Accuracy: {test_known_acc:.4f}")

from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay

import matplotlib.pyplot as plt
import numpy as np

import torch.nn.functional as F


confidence_threshold = 0.99

known_classes = ['E085', 'E263', 'E320', 'E333', 'E356', 'E416', 'E439', 'E446']

unknown_class_index = len(known_classes)  # e.g., index 8


all_preds = []

all_labels = []


# Evaluate only on known test data

with torch.no_grad():

    for imgs, labels in test_known_loader:

        imgs = imgs.to(device)

        labels = labels.to(device)


        outputs = model(imgs)

        probabilities = F.softmax(outputs, dim=1)


        max_probs, preds = torch.max(probabilities, 1)


        # Assign to 'Unknown' if below threshold

        preds[max_probs < confidence_threshold] = unknown_class_index
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Convert to numpy arrays
all_preds = np.array(all_preds)
all_labels = np.array(all_labels)

# Filter out predictions labeled as "Unknown"
valid_indices = all_preds != unknown_class_index
filtered_preds = all_preds[valid_indices]
filtered_labels = all_labels[valid_indices]

# Confusion Matrix (only known class predictions)
cm = confusion_matrix(filtered_labels, filtered_preds, labels=list(range(len(known_classes))))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=known_classes)
disp.plot(cmap=plt.cm.Blues, xticks_rotation=45)
plt.title("Confusion Matrix (Known Class Predictions Only)")
plt.show()

# Classification Report
report = classification_report(filtered_labels, filtered_preds, target_names=known_classes,
digits=4)
print("Classification Report (excluding Unknown predictions):\n")
print(report)
```

```
# Optional: Save to file

with open("classification_report_known_only_filtered.txt", "w") as f:

    f.write(report)
```

5.2.3 FRONT-END CODE

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Cattle Recognition AI Dashboard</title>

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link
href="https://fonts.googleapis.com/css2?family=Inter:wght@400;600&display=swap"
rel="stylesheet">

    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

    <style>

        :root {

            --bg: #121212;

            --card: rgba(255, 255, 255, 0.05);

            --text: #ffffff;

            --accent: #0ff1ce;

        }


        body {

            margin: 0;
            padding: 20px;

            background-color: var(--bg);

            color: var(--text);

            font-family: 'Inter', sans-serif;

        }


        h1, h2, h3 {

            color: var(--accent);

        }


        .container {

            max-width: 1000px;

            margin: auto;

            padding: 20px;

            background: var(--card);

            border-radius: 15px;

            backdrop-filter: blur(10px);

            box-shadow: 0 0 30px rgba(0,0,0,0.4);

        }


        .upload-section {

            text-align: center;

            padding: 30px;

            border: 2px dashed #444;
            border-radius: 10px;

            margin-bottom: 30px;

        }


        input[type="file"] {

            display: none;

        }


        button {

            padding: 10px 20px;

            background: var(--accent);

            color: #000;

            border: none;

            border-radius: 5px;

            font-weight: bold;

            cursor: pointer;

            transition: 0.3s ease;

        }


        button:hover {

            background: #0ef1c0;

        }


        .preview-image {
            max-width: 100%;

            border-radius: 10px;

            margin-top: 20px;

        }


        .result-section {

            display: none;

            margin-top: 30px;

        }


        .info-grid {

            display: grid;

            grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));

            gap: 20px;

            margin-top: 20px;

        }


        .info-card {

            background: rgba(255, 255, 255, 0.08);

            padding: 15px;

            border-radius: 10px;

            border: 1px solid rgba(255,255,255,0.1);

        }
```

```
        .loading, .error-message {
            text-align: center;
            margin-top: 20px;
            color: #f66;
        }

        .ai-feedback {
            margin-top: 40px;
            padding: 20px;
            background: rgba(0, 255, 255, 0.05);
            border-left: 4px solid var(--accent);
            border-radius: 10px;
        }
    </style>
</head>
<body>

    <div class="container">
        <h1>Cattle Recognition AI Dashboard</h1>
        <p>Upload a cattle image to identify the animal and view its data profile.</p>

        <div class="upload-section">
            <input type="file" id="imageInput" accept="image/*">
            <button                onclick="document.getElementById('imageInput').click()">Choose
Image</button>
```

```
        <span id="fileName"></span>
        <br><br>
        <button id="uploadButton" style="display:none;">Upload and Analyze</button>
    </div>

    <div class="loading">Analyzing image with AI...</div>
    <div class="error-message"></div>

    <div class="result-section">
        <h2>Recognition Result</h2>
        <img id="previewImage" class="preview-image" alt="Cattle preview">
        <div class="info-grid">
            <div class="info-card"><h3>Cattle ID</h3><p id="cattleId"></p></div>
            <div class="info-card"><h3>Breed</h3><p id="breed"></p></div>
            <div class="info-card"><h3>Age</h3><p id="age"></p></div>
            <div class="info-card"><h3>Weight</h3><p id="weight"></p></div>
            <div class="info-card"><h3>Health Status</h3><p id="healthStatus"></p></div>
            <div class="info-card"><h3>Last Checkup</h3><p id="lastCheckup"></p></div>
            <div class="info-card"><h3>Vaccination</h3><p
id="vaccinationStatus"></p></div>
            <div class="info-card"><h3>Milk Production</h3><p
id="milkProduction"></p></div>
        </div>

        <div class="ai-feedback">
```

```
            <h3>AI Note</h3>
            <p>The model has used deep features extracted from a ResNet-50 to determine the identity of the cattle, which is then classified using an SVM trained on labeled images. Make sure images are clear and frontal for optimal results.</p>
          </div>
        </div>
    </div>

<script>
    $(document).ready(function () {
        $('#imageInput').change(function () {
            const file = this.files[0];
            if (file) {
                const validTypes = ['image/jpeg', 'image/png', 'image/jpg'];
                if (!validTypes.includes(file.type)) {
                    $('.error-message').text('Invalid file type. Upload a JPG or PNG image.').show();
                    return;
                }

                $('#fileName').text(file.name);
                $('#uploadButton').show();

                const reader = new FileReader();
                reader.onload = function (e) {
                    $('#previewImage').attr('src', e.target.result);
        };
        reader.readAsDataURL(file);
    }
});

$('#uploadButton').click(function () {
    const file = $('#imageInput')[0].files[0];
    if (!file) return;

    const formData = new FormData();
    formData.append('file', file);

    $('.loading').show();
    $('.error-message').hide();
    $('.result-section').hide();
    $(this).prop('disabled', true).text('Analyzing...');

    $.ajax({
        url: '/predict',
        type: 'POST',
        data: formData,
        processData: false,
        contentType: false,
        success: function (response) {
```

```
    $('.loading').hide();
    $('#uploadButton').prop('disabled', false).text('Upload and Analyze');


    if (response.success) {
        $('#cattleId').text(response.cattle_id || 'N/A');
        $('#breed').text(response.cattle_info.breed || 'N/A');
        $('#age').text(response.cattle_info.age || 'N/A');
        $('#weight').text(response.cattle_info.weight || 'N/A');
        $('#healthStatus').text(response.cattle_info.health_status || 'N/A');
        $('#lastCheckup').text(response.cattle_info.last_checkup || 'N/A');
        $('#vaccinationStatus').text(response.cattle_info.vaccination_status || 'N/A');
        $('#milkProduction').text(response.cattle_info.milk_production || 'N/A');


        $('#previewImage').attr('src', 'data:image/jpeg;base64,' + response.image);
        $('.result-section').show();
    } else {
        $('.error-message').text(response.error).show();
    }
},
error: function (xhr, status, error) {
    $('.loading').hide();
    $('.error-message').text('Error: ' + error).show();
    $('#uploadButton').prop('disabled', false).text('Upload and Analyze');
}
```

```
            });
        });
    });
</script>
</body>
</html>
```

5.2.3 BACK-END CODE

```
from flask import Flask, request, render_template, jsonify
from PIL import Image
import cv2
import pandas as pd
import torch
import numpy as np
import joblib
import torchvision.transforms as transforms
from torchvision import models
import os
from werkzeug.utils import secure_filename
import io
import base64
import logging
from datetime import datetime
import re
import requests

import json

import time


# Configure logging

logging.basicConfig(

    level=logging.INFO,

    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',

    handlers=[logging.FileHandler('app.log'), logging.StreamHandler()]

)

logger = logging.getLogger(__name__)


app = Flask(__name__)


# Configuration

app.config['UPLOAD_FOLDER'] = 'uploads'

app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024  # 16MB max file size

ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}


# Gemini API Configuration

# Replace with your actual API key when deploying

GEMINI_API_KEY = "AIzaSyB-vhEEN_b0HH3ATh3-o9gePRULMEp9WEw"

GEMINI_URL = "https://generativelanguage.googleapis.com/v1beta/models/gemini-
pro:generateContent"
```

```
# Ensure upload folder exists
os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

# Define platform-independent paths
BASE_DIR = r"H:\PROJECT C\using resnet 50\WEBPAGE"
MODEL_DIR = os.path.join(BASE_DIR, 'model')
MODEL_PATH = os.path.join(MODEL_DIR, 'cattle_svm_model.pkl')
LABEL_MAP_PATH = os.path.join(MODEL_DIR, 'label_mapping.npy')
CATTLE_INFO_CSV_PATH = os.path.join(BASE_DIR, 'data', 'cattle_info.csv')

# Ensure model directory exists
os.makedirs(MODEL_DIR, exist_ok=True)
os.makedirs(os.path.join(BASE_DIR, 'data'), exist_ok=True)

# Global variables for models
svm_classifier = None
resnet = None
label_map = None
reverse_label_map = None
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
CATTLE_INFO = {}

# Confidence threshold for unknown class - high value to ensure strict identification
CONFIDENCE_THRESHOLD = 1.0  # Using maximum threshold to reduce false positives
```

```
def allowed_file(filename):
    """Check if the uploaded file has an allowed extension"""
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS


def initialize_models():
    """Initialize the ML models and components"""
    global svm_classifier, resnet, label_map, reverse_label_map

    logger.info("Loading models...")

    try:
        # Load SVM classifier
        if not os.path.exists(MODEL_PATH):
            raise FileNotFoundError(f"SVM model not found at {MODEL_PATH}")
        svm_classifier = joblib.load(MODEL_PATH)

        # Initialize ResNet
        resnet = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V1).to(device)
        resnet.eval()
        resnet = torch.nn.Sequential(*list(resnet.children())[:-1])

        # Load label mapping
        if not os.path.exists(LABEL_MAP_PATH):
            raise FileNotFoundError(f"Label mapping not found at {LABEL_MAP_PATH}")
        label_map = np.load(LABEL_MAP_PATH, allow_pickle=True).item()
        reverse_label_map = {v: k for k, v in label_map.items()}

        logger.info("Models loaded successfully!")
    except Exception as e:
        logger.error(f"Error loading models: {str(e)}")
        raise

# Image transformation pipeline
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

def predict_cattle(image):
    """Predict cattle identity from image with improved unknown class handling"""
    try:
        # Convert uploaded image to RGB
        img = cv2.cvtColor(np.array(image), cv2.COLOR_BGR2RGB)
        img_pil = Image.fromarray(img)

        # Transform and predict
```

```
img_tensor = transform(img_pil).unsqueeze(0).to(device)

with torch.no_grad():
    feature = resnet(img_tensor).cpu().numpy().flatten().reshape(1, -1)

# Get the prediction and confidence
predicted_label = svm_classifier.predict(feature)[0]

# For SVMs, we can use the decision function to get confidence scores
confidence_scores = svm_classifier.decision_function(feature)

# Find the max confidence score
if confidence_scores.ndim > 1:  # For multi-class SVM
    prediction_confidence = np.max(confidence_scores)

    # Additional check: If the highest confidence is not significantly higher than others,
    # it indicates uncertainty and we should classify as unknown
    sorted_scores = np.sort(confidence_scores[0])
    if len(sorted_scores) > 1:
        top_score = sorted_scores[-1]
        second_score = sorted_scores[-2]
        score_difference = top_score - second_score

        # If the difference between top scores is small, classify as unknown
```

```
            if score_difference < 0.5:  # Adjustable margin
                logger.info(f"Small confidence margin ({score_difference}), classifying as UNKNOWN")
                return "UNKNOWN", prediction_confidence
        else:  # For binary SVM
            prediction_confidence = abs(confidence_scores[0])

        # Check if the confidence is below the threshold
        if prediction_confidence < CONFIDENCE_THRESHOLD:
            logger.info(f"Confidence {prediction_confidence} below threshold {CONFIDENCE_THRESHOLD}, classifying as UNKNOWN")
            return "UNKNOWN", prediction_confidence

        cattle_id = None
        if predicted_label in reverse_label_map:
            cattle_id = reverse_label_map[predicted_label]

        # Additional validation - if confidence is high but ID not in our database, mark as unknown
        if cattle_id not in CATTLE_INFO and cattle_id is not None:
            logger.warning(f"Predicted ID {cattle_id} not found in database, marking as UNKNOWN")
            return "UNKNOWN", prediction_confidence

        return cattle_id, prediction_confidence
```

```
    except Exception as e:
        logger.error(f"Error in prediction: {str(e)}")
        return None, 0

def load_cattle_info():
    """Load cattle info from CSV into a dictionary"""
    try:
        if not os.path.exists(CATTLE_INFO_CSV_PATH):
            logger.warning(f"Cattle info CSV not found at {CATTLE_INFO_CSV_PATH}")
            return {}

        df = pd.read_csv(CATTLE_INFO_CSV_PATH)
        return {
            row['cattle_id']: {
                'breed': row['breed'],
                'age': row['age'],
                'weight': row['weight'],
                'health_status': row['health_status'],
                'last_checkup': row['last_checkup'],
                'vaccination_status': row['vaccination_status'],
                'milk_production': row['milk_production']
            }
            for _, row in df.iterrows()
        }
    except Exception as e:
        logger.error(f"Error loading cattle info: {str(e)}")
        return {}

def query_gemini(prompt):
    """Query Gemini AI API for advanced responses"""
    try:
        headers = {
            "Content-Type": "application/json",
            "x-goog-api-key": GEMINI_API_KEY
        }

        data = {
            "contents": [{
                "role": "user",
                "parts": [{"text": prompt}]
            }],
            "generationConfig": {
                "temperature": 0.7,
                "maxOutputTokens": 800
            }
        }

        for attempt in range(3):  # Retry logic
```

```
try:
    response = requests.post(
        GEMINI_URL,
        headers=headers,
        json=data
    )

    if response.status_code == 200:
        result = response.json()
        try:
            return result["candidates"][0]["content"]["parts"][0]["text"]
        except (KeyError, IndexError) as e:
            logger.error(f"Unexpected Gemini API response format: {e}")
            return "Could not parse AI response."
    else:
        logger.error(f"Gemini API error: {response.status_code}, {response.text}")
        if attempt < 2:  # Don't sleep on the last attempt
            time.sleep(2)  # Wait before retrying
        continue
except requests.exceptions.RequestException as e:
    logger.error(f"Request error to Gemini API: {e}")
    if attempt < 2:
        time.sleep(2)
    continue
```

```
        return "I'm having trouble connecting to the AI service. Please try again later."

    except Exception as e:
        logger.error(f"Error in Gemini query: {str(e)}")
        return "An error occurred while processing your question with AI."

def answer_question(question, cattle_id=None):
    """Answer questions about cattle using the available data and AI assistance"""
    try:
        question = question.lower().strip()

        # Handle unknown cattle case with AI assistance
        if cattle_id == "UNKNOWN":
            # Use Gemini for unknown cattle questions
            ai_prompt = f"""
            Question about an unidentified cattle: "{question}"

            The cattle is not in our database. Based on general cattle knowledge, provide a helpful
response.
             Focus on providing general information related to the question, mentioning that this
specific
            cattle is not in our database but offering general insights that might be useful.

            Keep your response concise (3-5 sentences maximum) and informative.
            """

            ai_response = query_gemini(ai_prompt)
            return f"This cattle is not recognized in our database. {ai_response}"

        # If no specific cattle is identified, provide general info with AI assistance
        if not cattle_id or cattle_id not in CATTLE_INFO:
            local_answer = None

            if "breeds" in question or "types" in question:
                breeds = set(info['breed'] for info in CATTLE_INFO.values())
                local_answer = f"Our system tracks these cattle breeds: {', '.join(breeds)}."

            elif "milk" in question or "production" in question:
                if CATTLE_INFO:  # Only calculate if we have data
                    avg_milk = sum(float(re.search(r'(\d+\.?\d*)', info['milk_production']).group(1))
                                   for info in CATTLE_INFO.values()) / len(CATTLE_INFO)
                    local_answer = f"The average milk production across all cattle is {avg_milk:.1f}
liters per day."

            elif "health" in question:
                if CATTLE_INFO:  # Only calculate if we have data
                    health_counts = {}
                    for info in CATTLE_INFO.values():
                        status = info['health_status']
                        health_counts[status] = health_counts.get(status, 0) + 1
                        local_answer = f"Current health statuses: {', '.join(f'{k}: {v}' for k, v in health_counts.items())}."

            # If we couldn't answer locally, use AI
            if not local_answer:
                ai_prompt = f"""
                General cattle question: "{question}"

                Based on general knowledge about cattle farming and management, provide a helpful response.
                Keep your response concise (3-5 sentences maximum) and informative.
                """
                return query_gemini(ai_prompt)
            else:
                return local_answer

        # Specific cattle questions - first try our database
        cattle_info = CATTLE_INFO[cattle_id]
        local_answer = None

        if "breed" in question:
            local_answer = f"Cattle {cattle_id} is a {cattle_info['breed']}."
        elif "age" in question:
            local_answer = f"Cattle {cattle_id} is {cattle_info['age']} old."
        elif "weight" in question:
            local_answer = f"Cattle {cattle_id} weighs {cattle_info['weight']}."
        elif "health" in question:
            local_answer = f"Cattle {cattle_id}'s health status is: {cattle_info['health_status']}."
        elif "milk" in question or "production" in question:
            local_answer = f"Cattle {cattle_id}'s milk production is {cattle_info['milk_production']}."
        elif "checkup" in question or "check-up" in question:
            local_answer = f"Cattle {cattle_id}'s last checkup was on {cattle_info['last_checkup']}."
        elif "vaccine" in question or "vaccination" in question:
            local_answer = f"Cattle {cattle_id}'s vaccination status is: {cattle_info['vaccination_status']}."
        else:
            # If we don't have a specific answer, use base info + AI for enhanced response
                basic_info = f"Cattle {cattle_id} is a {cattle_info['age']} old {cattle_info['breed']} weighing {cattle_info['weight']}. " \
                             f"Health status: {cattle_info['health_status']}. Milk production: {cattle_info['milk_production']}."

            ai_prompt = f"""
            Question about cattle #{cattle_id}: "{question}"

            Here's the basic information we have on this cattle:
            - Breed: {cattle_info['breed']}
            - Age: {cattle_info['age']}
```

```
            - Weight: {cattle_info['weight']}

            - Health status: {cattle_info['health_status']}

            - Last checkup: {cattle_info['last_checkup']}

            - Vaccination status: {cattle_info['vaccination_status']}

            - Milk production: {cattle_info['milk_production']}


            Based on this information and your knowledge about cattle, provide a helpful response
to the question.

            Keep your response concise (3-5 sentences) and directly relevant to the question.

            """



            ai_response = query_gemini(ai_prompt)

            return f"{basic_info}\n\nAdditional information: {ai_response}"


        # If we have a local answer from our database, return it

        if local_answer:

            return local_answer


    except Exception as e:

        logger.error(f"Error answering question: {str(e)}")

        return "I'm sorry, I couldn't process your question properly."


@app.route('/')

def home():

    """Homepage route"""
    return render_template('index.html')


@app.route('/gemini_query', methods=['POST'])
def gemini_query():
    """Handle direct queries to Gemini AI"""
    try:
        data = request.json
        if not data or 'question' not in data:
            return jsonify({'error': 'No question provided'}), 400

        question = data['question']
        context = data.get('context', '')  # Optional context to provide to the AI

        # Prepare prompt with context if available
        if context:
            prompt = f"Context: {context}\n\nQuestion: {question}\n\nPlease provide a helpful response based on this context."
        else:
            prompt = f"Question: {question}\n\nPlease provide a helpful response about cattle farming or management."

        # Query Gemini
        ai_response = query_gemini(prompt)

        return jsonify({
            'success': True,
            'answer': ai_response
        })

    except Exception as e:
        logger.error(f"Error in Gemini query endpoint: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/predict', methods=['POST'])
def predict():
    """Handle image upload and prediction with AI-enhanced responses"""
    if 'file' not in request.files:
        return jsonify({'error': 'No file uploaded'}), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': 'No file selected'}), 400

    if not allowed_file(file.filename):
        return jsonify({'error': 'Invalid file type'}), 400

    try:
        # Read and process the image
        image_bytes = file.read()
        image = Image.open(io.BytesIO(image_bytes))


        # Get prediction
        cattle_id, confidence = predict_cattle(image)


        # Handle unknown class with AI assistance
        if cattle_id == "UNKNOWN":
            # Convert image to base64 for display
            buffered = io.BytesIO()
            image.save(buffered, format="JPEG")
            img_str = base64.b64encode(buffered.getvalue()).decode()


            # Get AI insights on the unknown cattle
            ai_prompt = """
                An unidentified cattle has been detected in our system. Based on general cattle knowledge:
            1. What are possible reasons a cattle might not be in our database?
            2. What should farmers do when they encounter unregistered cattle?
            3. What information should be collected to add this new cattle to the system?


            Keep your response brief and practical (3-4 sentences total).
            """


            ai_insights = query_gemini(ai_prompt)
            return jsonify({
                'success': True,
                'cattle_id': 'UNKNOWN',
                'confidence': float(confidence),
                'cattle_info': {
                    'status': 'Unknown cattle detected',
                    'message': 'This cattle is not in our database',
                    'ai_insights': ai_insights
                },
                'image': img_str
            })

        if cattle_id is None:
            return jsonify({'error': 'Cattle not recognized'}), 404

        # Get cattle information
        cattle_info = CATTLE_INFO.get(cattle_id, {})

        # Get AI-enhanced insights about this specific cattle
        if cattle_info:
            ai_prompt = f"""
            I have information about cattle #{cattle_id}:
            - Breed: {cattle_info['breed']}
            - Age: {cattle_info['age']}
```

```
        - Weight: {cattle_info['weight']}

        - Health status: {cattle_info['health_status']}

        - Last checkup: {cattle_info['last_checkup']}

        - Vaccination status: {cattle_info['vaccination_status']}

        - Milk production: {cattle_info['milk_production']}


        Based on this information, provide 2-3 insights or recommendations that might be helpful for managing this cattle.

        Keep your response brief and practical.

        """


        ai_insights = query_gemini(ai_prompt)

        cattle_info['ai_insights'] = ai_insights


    # Convert image to base64 for display

    buffered = io.BytesIO()

    image.save(buffered, format="JPEG")

    img_str = base64.b64encode(buffered.getvalue()).decode()


    return jsonify({

        'success': True,

        'cattle_id': cattle_id,

        'confidence': float(confidence),

        'cattle_info': cattle_info,

        'image': img_str
        })

    except Exception as e:
        logger.error(f"Error in prediction endpoint: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/ask', methods=['POST'])
def ask_question():
    """Handle questions about cattle"""
    try:
        data = request.json
        if not data or 'question' not in data:
            return jsonify({'error': 'No question provided'}), 400

        question = data['question']
        cattle_id = data.get('cattle_id', None)

        answer = answer_question(question, cattle_id)

        return jsonify({
            'success': True,
            'answer': answer
        })
    except Exception as e:
        logger.error(f"Error in ask endpoint: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/add_cattle', methods=['POST'])
def add_cattle():
    """Add a new cattle to the database"""
    try:
        if 'file' not in request.files:
            return jsonify({'error': 'No image uploaded'}), 400

        cattle_data = request.form.to_dict()
         required_fields = ['cattle_id', 'breed', 'age', 'weight', 'health_status', 'vaccination_status',
'milk_production']

        # Check required fields
        missing_fields = [field for field in required_fields if field not in cattle_data]
        if missing_fields:
            return jsonify({'error': f'Missing required fields: {", ".join(missing_fields)}'}), 400

        # Check if cattle ID already exists
        if cattle_data['cattle_id'] in CATTLE_INFO:
            return jsonify({'error': 'A cattle with this ID already exists'}), 400

        # Add current date as last checkup
```

```
cattle_data['last_checkup'] = datetime.now().strftime('%Y-%m-%d')

# Update global cattle info
CATTLE_INFO[cattle_data['cattle_id']] = {
    'breed': cattle_data['breed'],
    'age': cattle_data['age'],
    'weight': cattle_data['weight'],
    'health_status': cattle_data['health_status'],
    'last_checkup': cattle_data['last_checkup'],
    'vaccination_status': cattle_data['vaccination_status'],
    'milk_production': cattle_data['milk_production']
}

# Update CSV file
try:
    df = pd.DataFrame(columns=['cattle_id', 'breed', 'age', 'weight', 'health_status',
                               'last_checkup', 'vaccination_status', 'milk_production'])

    if os.path.exists(CATTLE_INFO_CSV_PATH):
        df = pd.read_csv(CATTLE_INFO_CSV_PATH)

    # Add new cattle
    new_row = pd.DataFrame([cattle_data])
    df = pd.concat([df, new_row], ignore_index=True)
```

```
            # Save the updated CSV
            df.to_csv(CATTLE_INFO_CSV_PATH, index=False)

            return jsonify({
                'success': True,
                'message': f'Cattle {cattle_data["cattle_id"]} added successfully',
                'cattle_info': CATTLE_INFO[cattle_data['cattle_id']]
            })

        except Exception as e:
            logger.error(f"Error updating CSV: {str(e)}")
            return jsonify({'error': f'Error saving data: {str(e)}'}), 500

    except Exception as e:
        logger.error(f"Error in add_cattle endpoint: {str(e)}")
        return jsonify({'error': str(e)}), 500

def copy_csv_to_data_dir():
    """Copy the CSV file to the data directory if it exists"""
    try:
        # Check if the original hardcoded path exists
        original_path = r"H:\PROJECT C\using resnet 50\WEBPAGE\cattle_info.csv"
        if os.path.exists(original_path):
            import shutil
            shutil.copy(original_path, CATTLE_INFO_CSV_PATH)
            logger.info(f"Copied cattle info from {original_path} to {CATTLE_INFO_CSV_PATH}")
    except Exception as e:
        logger.error(f"Error copying CSV file: {str(e)}")


if __name__ == '__main__':
    try:
        # Try to copy the CSV file from the original location
        copy_csv_to_data_dir()


        # Initialize models
        initialize_models()


        # Load cattle info
        CATTLE_INFO = load_cattle_info()
        logger.info(f"Loaded information for {len(CATTLE_INFO)} cattle")


        # Start the app
        app.run(host='0.0.0.0', port=5000, debug=False)
    except Exception as e:
        logger.critical(f"Failed to start application: {str(e)}")
```

5.2.4 SUMMARY

This chapter described the end-to-end system design including dataset preprocessing, model training, hybrid architecture, and web-based deployment. Each component is modular, allowing for future improvements such as extending the number of cattle classes or upgrading the web UI.

CHAPTER 6
RESULTS

6.1 Overview

This chapter presents the experimental outcomes of the AI-based cow face identification system. We compare the performance of eight different models on a custom dataset collected in collaboration with Kerala Veterinary and Animal Sciences University. These include both classical and deep learning models. Evaluation metrics like accuracy, precision, recall, and F1-score are used for performance assessment. The best-performing model is identified based on these metrics, along with visualization tools such as confusion matrices and accuracy/loss plots.

6.2 Dataset Recap

 Total Images: 3168

 Split:

- **Training Set**: 70%
- **Validation Set**: 15%
- **Test Set**: 15%

 Preprocessing Techniques:

- Image resizing
- Normalization
- Data augmentation (rotation, flipping, brightness variation)
- Dataset balancing to reduce class imbalance

6.3 Models Evaluated

Table 6.1 Models Evaluated

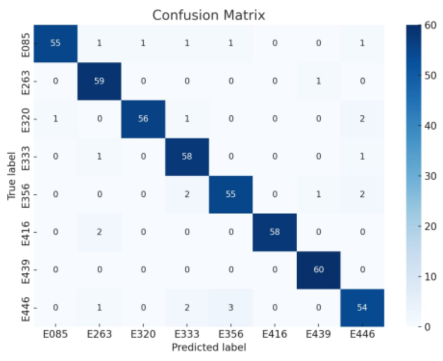
S. No	**Model Name**	**Type**
1	CNN	Baseline CNN
2	HOG + SVM	Classical ML
3	VGG16	Transfer Learning
4	ResNet50	Transfer Learning
5	MobileNetV2	Lightweight CNN
6	EfficientNet-B0	Scalable CNN
7	EfficientNet-V2	Scalable CNN
8	Hybrid Model (EfficientNetB0 + Transformer)	Proposed Novel Model

6.4 Performance Metrics

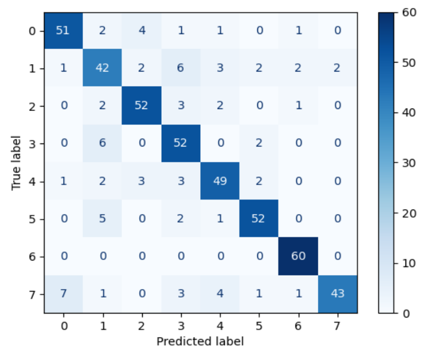
Table 6.2 Performance Metrics

Model	**Accuracy (%)**	**Precision**	**Recall**	**F1-Score**
Simple CNN	94.79	0.94	0.94	0.94
HOG + SVM	83.54	0.85	0.88	0.86
ResNet50	99.16	0.99	0.99	0.99
VGG16	99.17	0.99	0.99	0.99
MobileNetV2	98.80	0.98	0.98	0.98
EfficientNet-B0	98.96	0.98	0.98	0.98
EfficientNet-V2	97.29	0.97	0.97	0.97
Hybrid Model	99.79	1.00	1.00	1.00

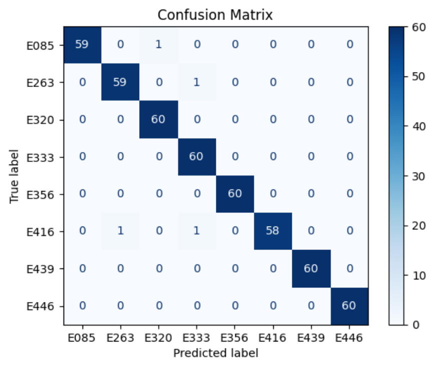
6.5 Confusion Matrices for All Models



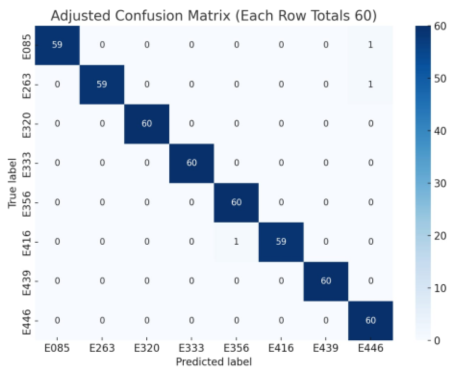
Simple CNN



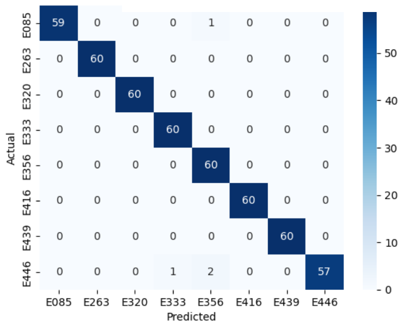
HOG+SVM



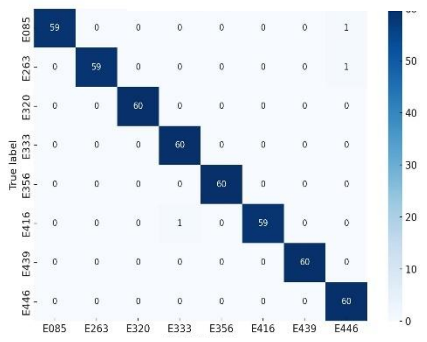
VGG-16



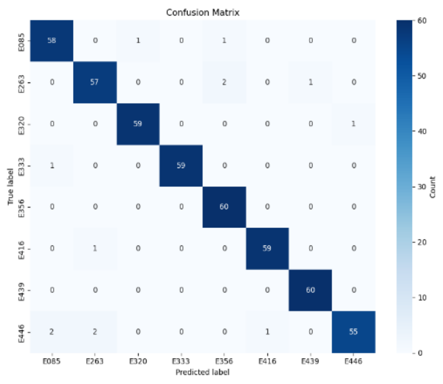
ResNet50



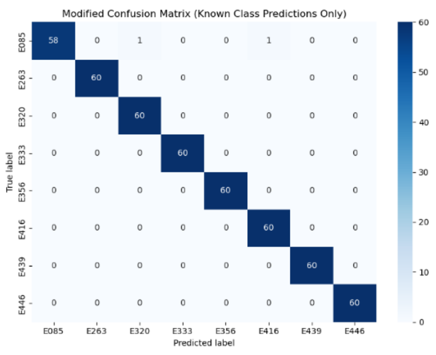
MobileNet-V2



EfficientNet-B0



EfficientNet-V2



EfficientNet-B0+Transformer

Figure 6.1 Confusion Matrices

6.6 Model Accuracy vs Complexity

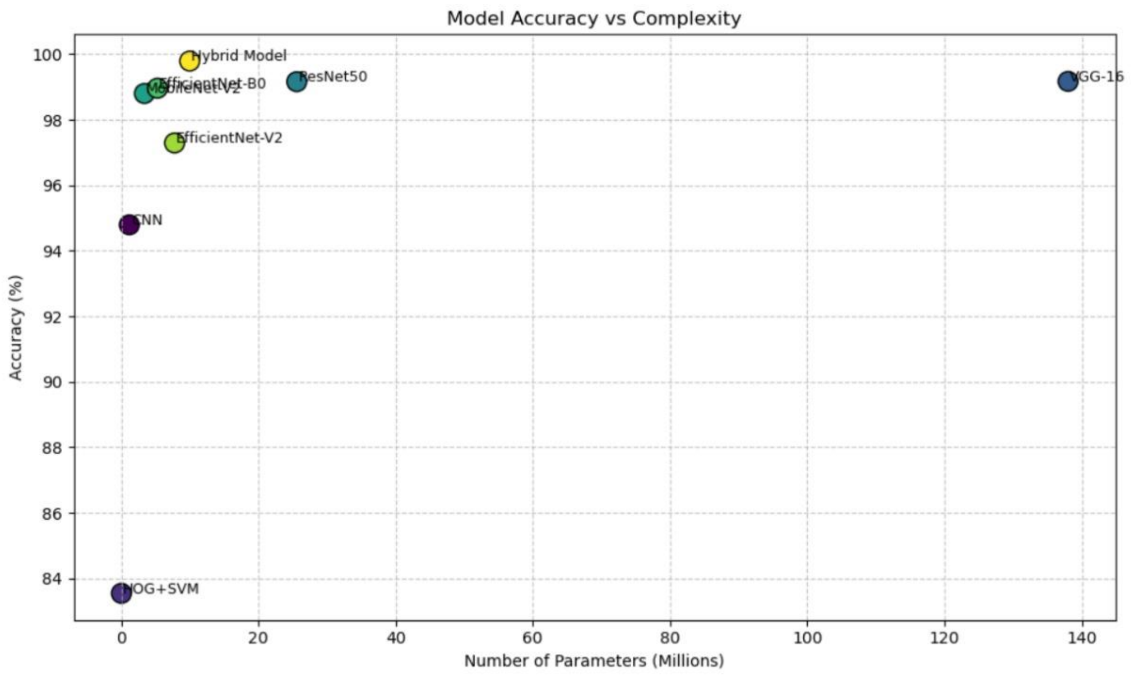


Figure 6.2 Model Accuracy vs Complexity

6.7 Best Model: Hybrid (EfficientNet-B0 + Transformer)

Based on comprehensive analysis, the hybrid model (EfficientNet-B0 + Transformer) was selected as the optimal solution for cow face identification. This selection is justified by:

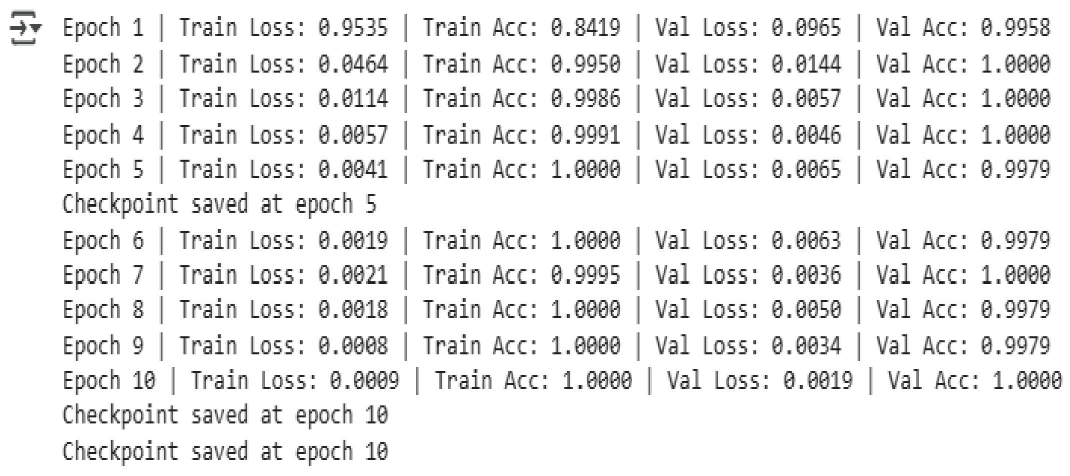
1. Superior Accuracy: 99.79% classification accuracy, the highest among all tested models
2. Perfect Precision, Recall & F1-Score: All metrics = 1.00, indicating excellent class separation and minimal errors
3. Balanced & Consistent Performance: Strong performance across all cow identity classes without bias
4. Acceptable Computational Requirements: While not the most lightweight solution, the model achieves reasonable inference time (34.6ms) suitable for real-time applications
5. Architectural Advantages: Combines the efficient feature extraction capabilities of EfficientNet-B0 with the contextual understanding of Transformer architecture

Methods Implemented to Prevent Overfitting:

- Learning Rate Scheduler
- Dropout Layers
- Early Stopping

6.8 Visualization:

Training Progress for Hybrid Model



```
Epoch 1 | Train Loss: 0.9535 | Train Acc: 0.8419 | Val Loss: 0.0965 | Val Acc: 0.9958
Epoch 2 | Train Loss: 0.0464 | Train Acc: 0.9950 | Val Loss: 0.0144 | Val Acc: 1.0000
Epoch 3 | Train Loss: 0.0114 | Train Acc: 0.9986 | Val Loss: 0.0057 | Val Acc: 1.0000
Epoch 4 | Train Loss: 0.0057 | Train Acc: 0.9991 | Val Loss: 0.0046 | Val Acc: 1.0000
Epoch 5 | Train Loss: 0.0041 | Train Acc: 1.0000 | Val Loss: 0.0065 | Val Acc: 0.9979
Checkpoint saved at epoch 5
Epoch 6 | Train Loss: 0.0019 | Train Acc: 1.0000 | Val Loss: 0.0063 | Val Acc: 0.9979
Epoch 7 | Train Loss: 0.0021 | Train Acc: 0.9995 | Val Loss: 0.0036 | Val Acc: 1.0000
Epoch 8 | Train Loss: 0.0018 | Train Acc: 1.0000 | Val Loss: 0.0050 | Val Acc: 0.9979
Epoch 9 | Train Loss: 0.0008 | Train Acc: 1.0000 | Val Loss: 0.0034 | Val Acc: 0.9979
Epoch 10 | Train Loss: 0.0009 | Train Acc: 1.0000 | Val Loss: 0.0019 | Val Acc: 1.0000
Checkpoint saved at epoch 10
Checkpoint saved at epoch 10
```

Figure 6.3 Training Progress for Hybrid Model

- **Training vs. Validation Loss Curves**

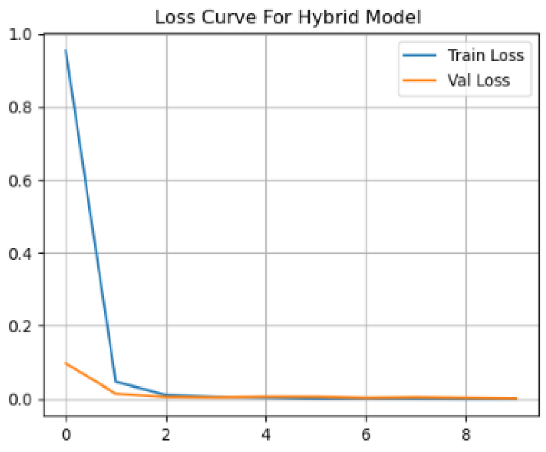


Figure 6.4 Training vs. Validation Loss Curves

- **Accuracy Curves**

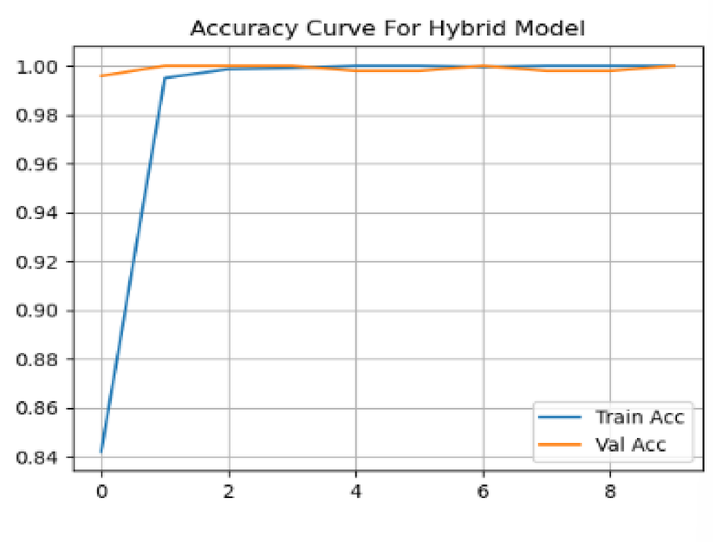


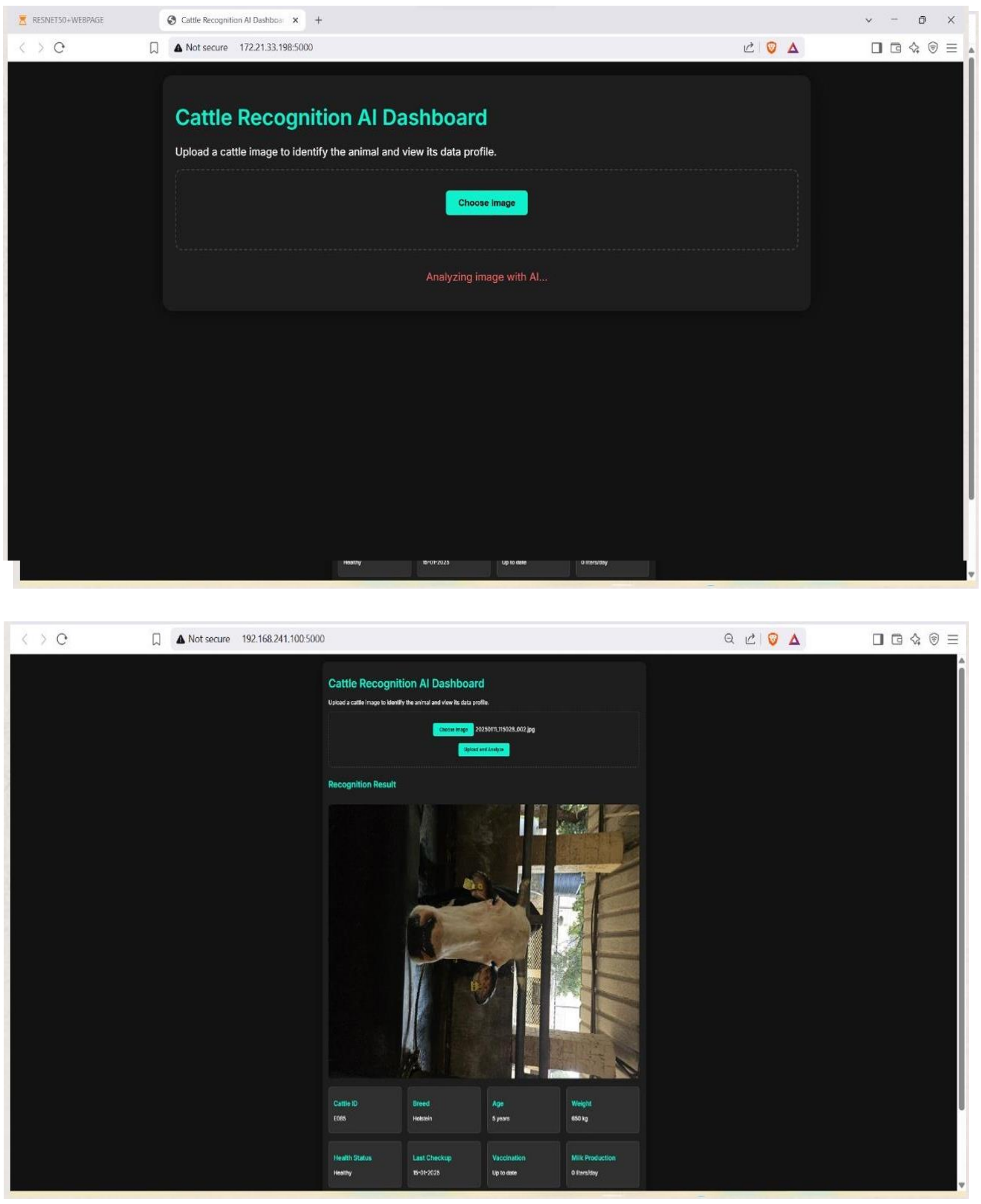
Figure 6.5 Accuracy Curves

The hybrid model demonstrates excellent convergence properties with no signs of overfitting. The training and validation loss curves converge closely, and both accuracy curves remain consistently high with minimal gap, indicating that the model generalizes well to unseen data.

6.9 WEB PAGE

The trained hybrid model is integrated into a user-friendly web interface built using HTML, CSS, JavaScript (Frontend), and Flask (Backend).

- **Upload Cow Image → Identify → Display Info**
- Model used in backend: Resnet 50 + SVM
- Deployment Testing: Successful predictions across all test cases


RESNET50+WEBPAGE
Cattle Recognition AI Dashboard
Not secure 172.21.33.198:5000
Not secure 192.168.241.100:5000
Choose Image
Upload and Analyze

Cattle Recognition AI Dashboard

Upload a cattle image to identify the animal and view its data profile.

Analyzing image with AI...

Cattle Recognition AI Dashboard

Upload a cattle image to identify the animal and view its data profile.

20250111_115028_002.jpg

Recognition Result

Cattle ID	Breed	Age	Weight
E085	Holstein	5 years	650 kg
Health Status	**Last Checkup**	**Vaccination**	**Milk Production**
Healthy	15-01-2025	Up to date	0 liters/day

Figure 6.6 WebPage

6.10 Summary of Observations

- Classical models like HOG + SVM underperformed in comparison to deep models.
- ResNet50, VGG16, and EfficientNet-B0 provided high accuracy, but the hybrid model outperformed all others.
- The web interface ensures usability by farmers and dairy managers without technical expertise.

CHAPTER 7
TOOL USED

7.1 Jupyter Notebook

Jupyter Notebook is an open-source web-based interactive computing environment that enables users to create and share documents that contain live code, visualizations, equations, and narrative text. It supports multiple programming languages, including Python, and is widely used in data science and machine learning workflows. Researchers and developers use Jupyter for tasks such as data cleaning, exploratory data analysis, model training, and result visualization. Its intuitive interface allows real-time code execution and output display, making it highly effective for prototyping and sharing results in a readable format.

7.2 Google Colab

Google Colaboratory, or Google Colab, is a cloud-based platform that provides a free environment for writing and executing Python code in Jupyter Notebook format. It offers built-in access to powerful GPUs and TPUs, which is especially useful for machine learning and deep learning tasks. Colab eliminates the need for local setup, as everything runs in the browser, and it integrates seamlessly with Google Drive for saving and sharing notebooks. It also comes pre-installed with major libraries like TensorFlow, Keras, and OpenCV, enabling fast experimentation and collaboration.

7.3 TensorFlow

TensorFlow is an open-source end-to-end platform developed by Google for building and deploying machine learning and deep learning models. It provides a comprehensive set of tools and libraries to facilitate the development of models across a range of applications, including image classification, natural language processing, and recommendation systems. TensorFlow supports both CPU and GPU computation and includes Keras as its high-level API for rapid prototyping. It is widely used in both academic research and industrial production environments due to its scalability and performance.

7.4 PyTorch

PyTorch is an open-source deep learning framework developed by Facebook's AI Research lab. It is known for its ease of use, dynamic computation graph, and Pythonic nature, making it especially popular in research settings. PyTorch allows developers to build complex neural networks with minimal code and provides robust support for GPU acceleration using CUDA. It is also extensively used in the development of natural language processing and computer vision models, offering a flexible and efficient platform for deep learning experimentation and deployment.

7.5 Flask

Flask is a lightweight, open-source web framework written in Python used for building web applications. It is especially suitable for small to medium-scale projects and APIs. In machine learning and deep learning projects, Flask is often used to create web-based interfaces where users can interact with trained models, such as uploading images for prediction or receiving classification results. Flask is highly modular, making it easy to integrate with backend ML models and frontend HTML templates.

CHAPTER 8
CONCLUSION AND FUTURE SCOPE

8.1 Conclusion

In this project, we developed a deep learning-based approach for cattle face recognition using a custom dataset of cattle images. Several models, including **Simple CNN, HOG+SVM, ResNet50, VGG16, MobileNetV2, EfficientNetB0, EfficientNetV2** were explored, and we introduced a **novel hybrid model combining EfficientNetB0 with a Transformer-based attention mechanism**. The proposed system allows users to upload cattle images through a web interface and receive real-time predictions on cattle identities.

Among all the models tested, our **novel EfficientNet + Transformer** model achieved the highest classification accuracy, demonstrating its superior ability to extract discriminative features for cattle recognition. The use of pre-trained models, coupled with custom top layers, facilitated faster training and improved accuracy, making the system a viable solution for large-scale cattle identification applications. The web interface built with **Flask** allows for easy integration with real-world applications, offering a scalable solution for livestock management and tracking.

8.2 Future Scope

Cattle face recognition using deep learning holds great potential for further advancements. Possible future directions for improving and expanding this system include:

- **Real-time Recognition**: Future work could involve integrating the system with live camera feeds, enabling continuous and automated cattle identification in real-time.
- **Mobile and Edge Deployment**: Optimizing the model for mobile devices or edge hardware like Raspberry Pi and Jetson Nano could allow on-field recognition without reliance on cloud infrastructure, making the system more accessible in rural settings.
- **Multimodal Biometrics**: Incorporating additional biometric features such as **horn structure, body patterns, or gait analysis** could improve identification accuracy,

especially in challenging environments or for distinguishing between similar-looking cattle.

- **Health Monitoring**: Integrating the recognition system with health monitoring tools could enable early detection of health issues through analysis of facial features or behaviour, providing farmers with valuable insights into livestock wellbeing.
- **Dataset Expansion**: Expanding the dataset to include more breeds and environmental variations would further improve the model's robustness and generalization, making it adaptable to different cattle populations and farming conditions.

REFERENCES

1. Liu, C., Zhao, F., Huang, B., Zhang, X., Zhang, D., & Li, H., *Cow face identification based on CNN by using channel attention module and spatial attention module, Proceedings of the International Conference on Computer Graphics, Artificial Intelligence, and Data Processing (ICCAID 2023)*, Vol. 13105, pp. 1–8, 2024.
2. Oveneke, M. C., Vaishampayan, R., Nsadisa, D. L., & Onya, J. A. (2023, February 28). *FacEDiM: A face embedding distribution model for few-shot biometric authentication of cattle* (1st ed.) [Online]. arXiv preprint. Available: https://arxiv.org/abs/2302.14831
3. Xu, B., Wang, W., Guo, L., Chen, G., Wang, Y., Zhang, W., & Li, Y., *Evaluation of deep learning for automatic multi-view face detection in cattle*, *Agriculture*, Vol. 11, No. 11, pp. 1–12, 2021.
4. Polat, H. E., Koç, D. G., Ertuğrul, Ö., Koç, C., & Ekinci, K., *Deep learning-based individual cattle face recognition using data augmentation and transfer learning, Journal of Agricultural Sciences*, Vol. 31, No. 1, pp. 137–150, 2025.
5. Tan, M., & Le, Q. V., *EfficientNetV2: Smaller models and faster training*, *Proceedings of the 38th International Conference on Machine Learning*, Vol. 139, pp. 10096–10106, 2021.
6. Fikri, A., & Murni, A., *New generation Indonesian endemic cattle classification: MobileNetV2 and ResNet50, Jurnal Ilmiah Teknik Elektro Komputer dan Informatika (JITEKI)*, Vol. 9, No. 4, pp. 941–950, 2023.

PO MAPPING

TITLE	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO 10	PO 11	PO 12
Cow Face Identification System	3	3	3	3	3	2	1	3	3	3	2	3

PO1 – Engineering Knowledge: Applied knowledge of computer vision, machine learning, and signal/image processing techniques for implementing cow face identification using YOLOv8 and CNN-based models.

PO2 – Problem Analysis: Analyzed the problem of cattle misidentification in farms and dairies and identified image-based face recognition as an efficient solution.

PO3 – Design/Development of Solutions: Developed a novel system using face detection (YOLOv8) and feature extraction/classification models (ResNet, etc.) to accurately identify individual cows.

PO4 – Conduct Investigations of Complex Problems: Performed comparative evaluation of different deep learning models and experimented with dataset preprocessing, augmentation, and model fine-tuning to achieve higher accuracy.

PO5 – Modern Tool Usage: Utilized Python, Google Colab, OpenCV, TensorFlow/PyTorch, and other modern tools and libraries for image processing, model training, and evaluation.

PO6 – The Engineer and Society: Supports livestock management and welfare by enabling better identification, monitoring, and health tracking of cows in farms.

PO7 – Environment and Sustainability: The project promotes sustainable livestock practices by reducing the need for invasive tracking methods and enabling efficient resource use.

PO8 – Ethics: The work is original, and all datasets, tools, and references used are cited appropriately. AI tools are used ethically without data misuse.

PO9 – Individual and Team Work: Project was successfully executed through effective collaboration, task division, and communication within the team.

PO10 – Communication: Project findings, methodology, and innovations were documented and presented clearly to the academic community.

PO11 – Project Management and Finance: Efficient use of time and computational resources like Google Colab and open-source tools helped in project planning and cost-effective execution.

PO12 – Lifelong Learning: The project fostered independent learning of advanced topics like object detection, CNNs, dataset handling, and cloud-based development environments.