Activity- 01 (Image Classification)

Objective:

The goal is to build an image classification system that processes and categorizes images into specific classes, such as "cat" and "dog," using Python, OpenCV, and TensorFlow. This involves preparing the dataset, building and training a model, and testing its performance.

Step - 1 : Data Collection and Preparation:

Dataset Details

- Total number of images: **10,000**
  - Training set: **8,000 images** (80% of the dataset).
  - Validation set: **2,000 images** (20% of the dataset).
- Classes: **['cats', 'dogs']**

Dataset Directory:

dataset/

cats/

cat1.jpg

cat2.jpg

dogs/

dog1.jpg

dog2.jpg

Splits the dataset into training (80%) and validation (20%) subsets.

Resizes images to img_size=(128, 128) for consistency.

Groups images into batches of size 32 (batch_size=32) to process them efficiently

Training Dataset [16000 Files]: Used for model training.

Validation Dataset [4000 Files]: Used to evaluate the model's performance during training.

Class Names ['cats', 'dogs']: Automatically extracted from folder names.

Step 2: Preprocessing

Objective

Prepare images for input into the model and make the training process more robust.

Data Augmentation:

Introduces random transformations to training images, making the model more robust to variations.

Transformations applied:

RandomFlip("horizontal"): Randomly flips images horizontally.

RandomRotation(0.2): Rotates images randomly by ±20%.

RandomZoom(0.2): Zooms in/out randomly by 20%.

This ensures the model generalizes well to unseen data.

Normalization:

Converts pixel values from the range [0, 255] to [0, 1].

Done using Rescaling(1.0 / 255).

Performance Optimization:

cache(): Stores preprocessed images in memory to avoid redundant computations during training.

prefetch(buffer_size=tf.data.AUTOTUNE): Asynchronously prepares the next batch while the model processes the current one.

Result:

Output: Optimized and augmented datasets (train_ds and val_ds) ready for training.

Step 3: Feature Extraction and Model Building

Objective

Build a classification model that can learn patterns from image features.

Transfer Learning:

Use MobileNetV2, a pretrained model designed for feature extraction.

Benefits:

Leverages knowledge from training on the large ImageNet dataset.

Reduces training time and improves accuracy with fewer data.

Adjustments:

include_top=False: Removes the final classification layers to customize for our classes.

trainable=False: Freezes the base model to retain its pretrained knowledge.

Custom Layers:

Add layers to tailor the model for our classification task:

GlobalAveragePooling2D: Reduces the output of MobileNetV2 to a single feature vector per image.

Dense(128, activation="relu"): Fully connected layer with 128 neurons to learn task-specific features.

Dropout(0.6): Randomly disables 60% of neurons during training to prevent overfitting.

Dense(2, activation="softmax"): Final layer with two output nodes (one for each class) and softmax activation to output class probabilities.

Compilation:

Optimizer: Adam with a learning rate of 0.0001 for stable convergence.

Loss Function: sparse_categorical_crossentropy.

Metrics: Monitor accuracy during training.

Step 4: Training the Model

Objective

Train the model to learn from the training dataset while monitoring validation performance.

EarlyStopping:

Stops training if validation loss doesn't improve for 3 consecutive epochs.

Prevents overfitting and saves computational resources.

Training Process:

model.fit:

Feeds the training and validation datasets into the model for up to 25 epochs.

Updates model weights to minimize loss using the training data.

Validates on the reserved validation data after each epoch.

Output:

A trained model with weights optimized to classify images.

Step 5: Plotting Training History

Objective

Visualize the training process to understand model behavior.

Insights from Training History:

Accuracy and Loss Trends:

The model shows strong convergence, with minimal overfitting.

Validation accuracy closely follows training accuracy.

Step 6: Testing and Evaluation

Objective

Evaluate the model's performance on unseen data and calculate metrics.

```
Classification Report:
              precision    recall  f1-score   support

        cats       0.97      0.97      0.97       980
        dogs       0.97      0.97      0.97      1020

    accuracy                           0.97      2000
   macro avg       0.97      0.97      0.97      2000
weighted avg       0.97      0.97      0.97      2000

Accuracy: 0.9675
```

Precision:

- Cats: 97% of cat predictions were correct.
- Dogs: 96% of dog predictions were correct.

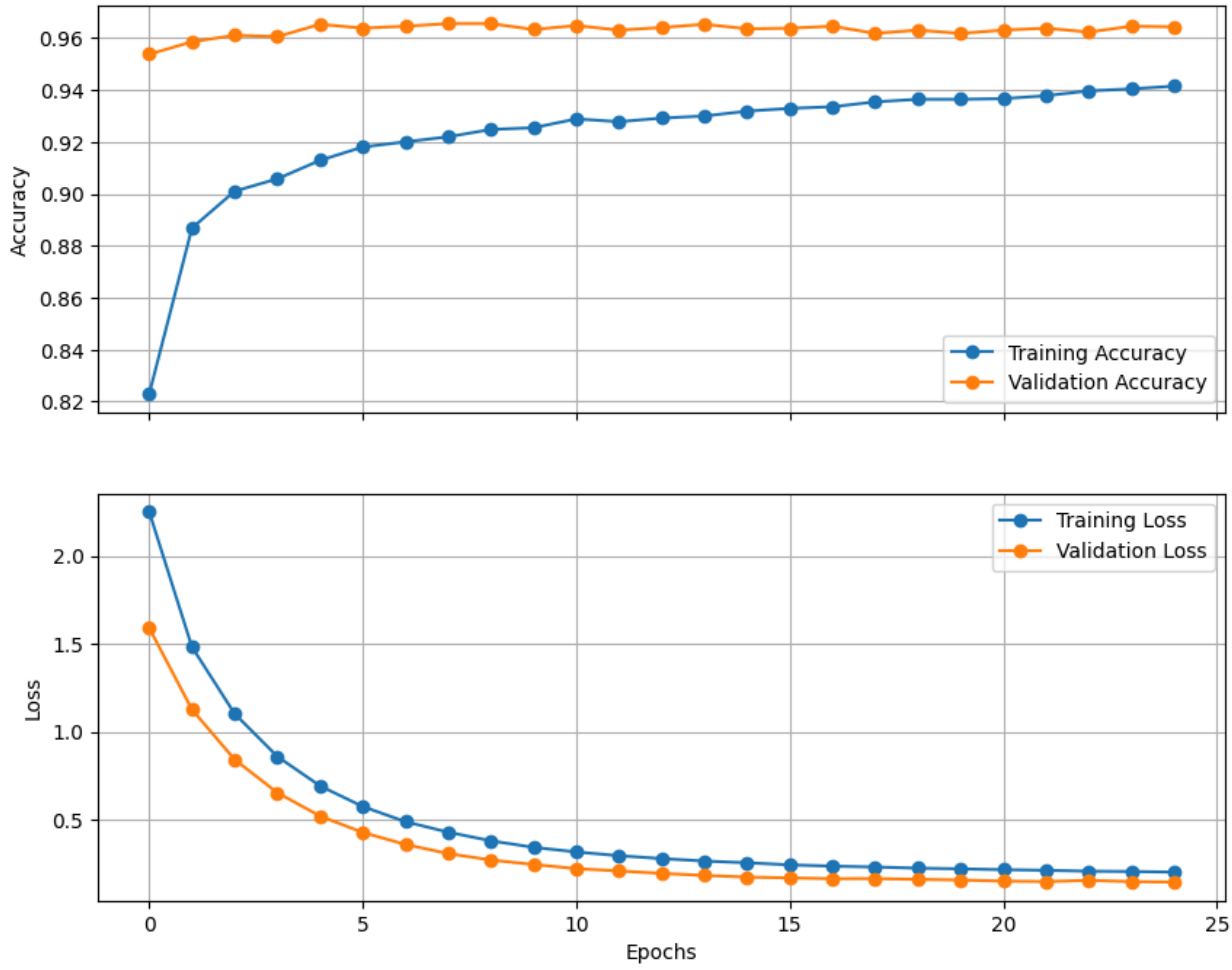
Recall:

- Cats: 95% of all actual cats were correctly identified.
- Dogs: 97% of all actual dogs were correctly identified.

F1-Score:

- A balanced measure of precision and recall, both at 96–97%.

Accuracy: 96.75%



Potting of the Training accuracy vs validation accuracy(Upper one)

Plotting of the Training loss vs validation loss (lower one)

Step 7: Classifying New Images

Objective

Test the trained model on unseen images and display results visually.

Implementation:

Prediction:

- Predicts the class probabilities of a given image.

- Displays the image with the predicted class label overlaid.

Step 8: Deploying the Model

Objective

Save the trained model and its training history for future use.

Implementation:

1. Save the Model:

 - `model.save` saves the trained model for reuse.

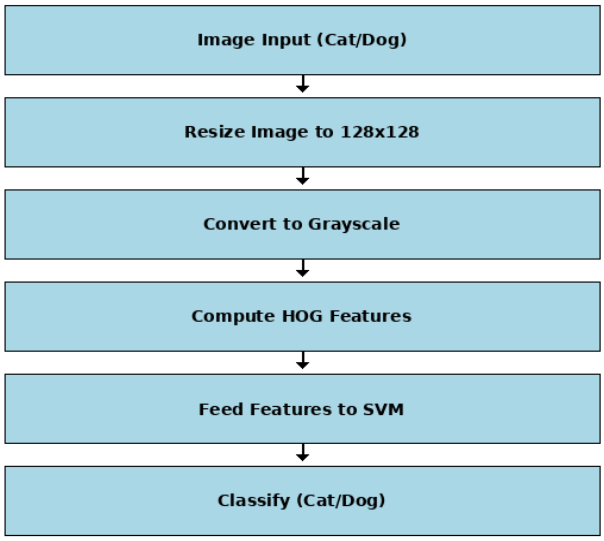
2. Save Training History:

 - `np.save` stores the accuracy and loss data for future analysis.

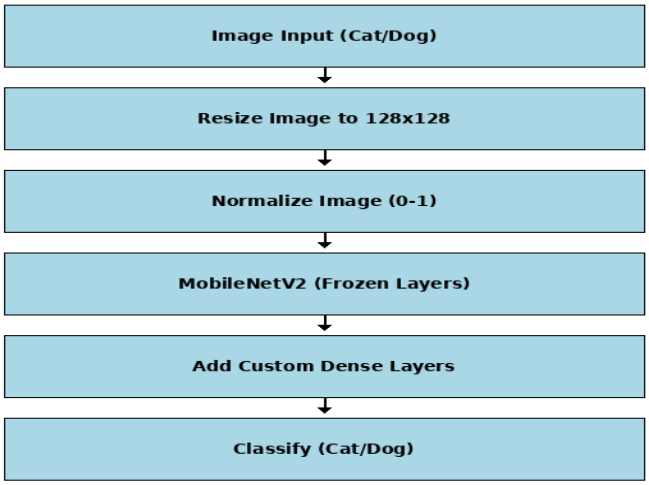
3. For deploying using streamlit

– saved the model in .h5 format

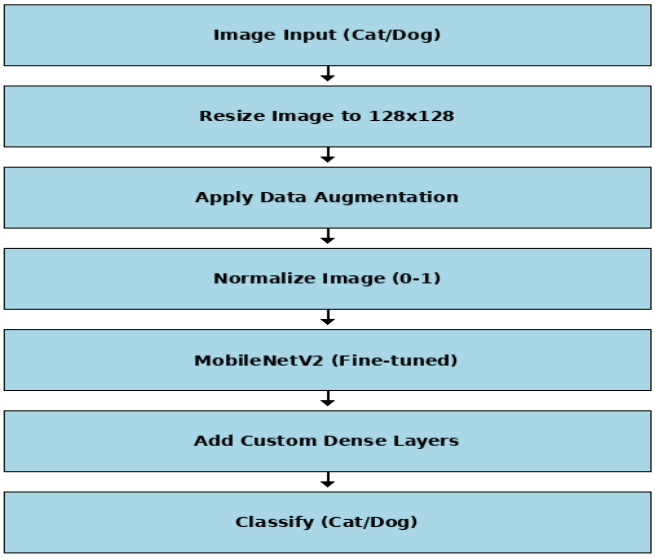
Block diagrams illustrating the feature extraction methods used in each iteration:



First Iteration



Second Iteration



Third Iteration

Key Takeaways

1. High Accuracy:

- The model achieves 96.75% accuracy, demonstrating excellent performance on the validation set.

2. Class Balance:

- High precision, recall, and F1-scores indicate balanced performance across both classes.

3. Transfer Learning Effectiveness:

- Using a pretrained MobileNetV2 significantly improved accuracy with limited data and training time.

This pipeline successfully classifies images into "cats" and "dogs" with high accuracy and robust generalization.

Note that :

- The transition from traditional HOG features to the pre-trained MobileNetV2 significantly improved both the accuracy and computational efficiency of the model.
- Data augmentation played a crucial role in enhancing the model's robustness by introducing diversity to the dataset, making it more adaptable to unseen data.
- Fine-tuning the pre-trained MobileNetV2 model, after freezing the initial layers, enabled better feature extraction, leading to a notable increase in performance.
- Additionally, optimizing the model with techniques such as adjusting learning rates, incorporating dropout layers, and using EarlyStopping contributed to achieving higher accuracy. This iterative approach underscores the effectiveness of leveraging pre-trained models and fine-tuning strategies for successful image classification tasks.