Image Classification on Caltech-101 Dataset

Course: SHBT 261
Project: Mini-Project 1
Date: October 2, 2025

Abstract

This project implements and compares three different image classification approaches on the Caltech-101 dataset: a traditional machine learning method using HOG features with SVM classifier, and two deep learning methods based on ResNet50 and EfficientNet-B0 architectures. The dataset contains 102 object categories with approximately 9,000 images total. Our experiments show that deep learning approaches significantly outperform traditional methods, with EfficientNet-B0 achieving **97.14%** test accuracy, compared to ResNet50's 95.23% and HOG+SVM's 83.80%. This represents a 13.34 percentage point improvement from the classical baseline to the best deep learning model. We also investigate the impact of transfer learning and provide detailed analysis of classification errors. All models are evaluated using multiple metrics including accuracy, per-class accuracy, precision, recall, F1-score, and confusion matrices.

---

1. Introduction

1.1 Background and Motivation

Image classification is a fundamental task in computer vision where the goal is to assign a category label to an input image. While humans can effortlessly recognize objects in images, automating this task has been a major challenge in artificial intelligence. Over the past two decades, approaches to image classification have evolved from hand-crafted feature extractors (like SIFT and HOG) combined with classical machine learning classifiers, to modern deep learning methods that automatically learn hierarchical feature representations from data.

This project focuses on the Caltech-101 dataset, a well-established benchmark dataset introduced by Fei-Fei et al. in 2004. Despite being relatively small by modern standards, Caltech-101 remains relevant for educational purposes and serves as a good testbed for comparing different classification approaches. The dataset presents several challenges including:

- **Limited training data**: With only ~40-800 images per category (average ~80), the dataset tests a model's ability to learn from limited examples
- **High intra-class variability**: Objects within the same category can vary significantly in pose, lighting, and background
- **Inter-class similarity**: Some categories are visually similar (e.g., different types of animals or musical instruments)
- **Class imbalance**: The number of images varies considerably across categories

1.2 Project Objectives

This project aims to implement and systematically compare three different classification approaches that represent both traditional machine learning and modern deep learning paradigms. Specifically, we implement a classical HOG+SVM baseline using hand-crafted features, a ResNet50 deep residual network with 50 layers, and EfficientNet-B0, a parameter-efficient convolutional neural network architecture. By comparing these methods on the same dataset with consistent evaluation protocols, we can understand the trade-offs between traditional and modern approaches in terms of accuracy, computational efficiency, and ease of implementation.

Beyond simply reporting accuracy numbers, we aim to comprehensively evaluate each model's performance across multiple dimensions. This includes computing per-class accuracy to understand performance variations across categories, calculating precision and recall to analyze different types of errors, and examining confusion matrices to identify systematic misclassification patterns. Such detailed analysis helps us understand not just how well each model performs overall, but where and why it succeeds or fails on specific types of images.

A key aspect of this project is investigating the effectiveness of transfer learning for deep learning models. We leverage pre-trained weights from ImageNet, a much larger dataset, and examine how this initialization affects both training dynamics and final performance on our smaller Caltech-101 dataset. Understanding transfer learning is crucial for practical applications where labeled data may be limited, making this investigation particularly relevant for real-world scenarios.

We also conduct ablation studies to understand how key design choices impact performance. These experiments investigate the effects of input image resolution (comparing 64×64, 128×128, and 224×224 pixel inputs) and data augmentation (comparing training with and without augmentation). Such studies provide insights into which components are most critical for success and help build intuition about deep learning systems, revealing that higher resolution improves accuracy up to a point (128×128 provides good trade-offs) and that augmentation effects depend on training duration and model capacity.

Ultimately, through detailed experimentation and analysis, we aim to provide practical insights into which approaches work best for this classification task and understand the underlying reasons. These lessons should generalize beyond this specific dataset and inform future work on similar image classification problems with limited training data.

1.3 Dataset Description

Caltech-101 Dataset Overview:

- **Source**: Computational Vision Group at Caltech
- **Total categories**: 102 (101 object categories + 1 background category)
- **Total images**: Approximately 9,000 images
- **Image characteristics**:
    - Color images with varying resolutions
    - Objects typically centered but with varying scales
    - Natural backgrounds with some clutter
    - Most images contain a single prominent object

Data Split Strategy:

Following standard machine learning practice, we split the dataset into three subsets using stratified sampling to maintain class distribution:

- **Training set**: 70% (12,755 images)
    - Used to train model parameters
    - Undergoes data augmentation for deep learning methods
- **Validation set**: 15% (2,700 images)
    - Used for hyperparameter tuning and early stopping
    - Not used for gradient updates
    - Helps monitor overfitting during training
- **Test set**: 15% (2,833 images)
    - Held out completely during training
    - Used only for final performance evaluation
    - Provides unbiased estimate of generalization performance

The stratified split ensures that each category maintains the same proportional distribution across all three sets, which is important given the class imbalance in the dataset.

Category Examples:

The dataset includes diverse object categories such as:

- **Faces**: frontal face images, easy faces
- **Animals**: leopards, elephants, dolphins, butterflies, etc.
- **Vehicles**: airplanes, motorbikes, cars (side view)
- **Musical instruments**: accordion, grand piano, saxophone
- **Household objects**: chairs, cups, lamps, laptops
- **Natural objects**: sunflowers, water lilies, lotus

Some categories are relatively easy to classify (e.g., airplanes, motorbikes) due to distinctive shapes, while others are more challenging due to high visual similarity or limited training examples.

1.4 Report Organization

The remainder of this report is organized as follows:

- **Section 2 (Methods)**: Describes the three classification approaches in detail, including HOG feature extraction with SVM, ResNet50 architecture, and EfficientNet-B0 design, along with training procedures and hyperparameter configurations.
- **Section 3 (Experimental Setup)**: Specifies the hardware and software environment, defines all evaluation metrics (accuracy, per-class accuracy, precision, recall, F1-score, Top-5 accuracy, confusion matrices), and explains the rationale for multi-metric evaluation.
- **Section 4 (Results)**: Presents comprehensive quantitative results for all three methods including detailed performance metrics, identification of best and worst performing categories, and analysis of confusion patterns and training dynamics.
- **Section 5 (Discussion)**: Analyzes performance differences between classical and deep learning

methods, examines challenging categories, discusses the critical role of transfer learning, and provides practical observations about implementation details.

- **Section 6 (Ablation Studies)**: Investigates the impact of input image resolution (64×64, 128×128, 224×224) and data augmentation (with/without) on ResNet18 performance, revealing insights about optimal design choices for this dataset.
- **Section 7 (Conclusion)**: Summarizes key findings, emphasizes the importance of transfer learning and parameter-efficient architectures, and provides practical recommendations for similar image classification tasks.
- **Section 8 (References)**

---

2. Methods

This section describes the three classification approaches implemented in this project. We start with a classical machine learning baseline using HOG features and SVM classifier, then present two modern deep learning approaches based on convolutional neural networks.

2.1 Method 1: HOG + SVM (Classical Machine Learning)

2.1.1 Overview

The HOG+SVM approach represents a classical computer vision pipeline that was state-of-the-art before the deep learning era. This method consists of two distinct stages: feature extraction using Histogram of Oriented Gradients (HOG), followed by classification using a Support Vector Machine (SVM). Unlike deep learning methods that learn features directly from data, this approach uses hand-crafted features designed by domain experts.

2.1.2 Image Preprocessing

All images undergo standardized preprocessing before feature extraction. Images are first loaded using the Python Imaging Library (PIL) and converted to RGB format to ensure consistent color space representation across the dataset. Since the original Caltech-101 images have varying dimensions, we resize all images to a fixed size of 128×128 pixels using bilinear interpolation. This standardization is necessary because HOG requires fixed-size inputs, and we choose this moderate resolution to balance computational efficiency with detail preservation.

Following resizing, RGB images are converted to grayscale for HOG feature extraction. The conversion uses the standard luminance formula $I = 0.299R + 0.587G + 0.114B$, which weights the channels according to human perceptual sensitivity. Since HOG operates on intensity gradients rather than color information, this grayscale conversion is sufficient and reduces computational complexity. While color can be informative for some categories, this baseline implementation focuses on shape and texture captured by gradient orientations.

2.1.3 HOG Feature Extraction

The Histogram of Oriented Gradients (HOG) descriptor captures local edge and gradient structure, which has proven effective for object recognition tasks. Our implementation uses the scikit-image library with carefully chosen parameters: 9 orientation bins covering 0-180 degrees for unsigned gradients, 8×8 pixels per cell defining the local region size for computing histograms, and 2×2 cells per block for normalization. We use L2-

Hys block normalization (L2-norm followed by clipping at 0.2 and renormalization), which provides better invariance to illumination changes compared to simpler normalization schemes.

For each 128×128 grayscale image, the HOG extraction proceeds through several computational steps. First, we compute image gradients at each pixel using central difference filters. Specifically, the horizontal gradient $G_x(i,j)$ and vertical gradient $G_y(i,j)$ are computed using $[-1, 0, 1]$ kernels. From these gradients, we calculate the gradient magnitude $|G| = \sqrt{G_x^2 + G_y^2}$ and orientation $\theta = \arctan(G_y/G_x)$ at each pixel.

Next, we divide the image into non-overlapping 8×8 pixel cells. For a 128×128 image, this yields a 16×16 grid of cells. Within each cell, we construct a 9-bin histogram of gradient orientations, where each pixel's gradient contributes to the histogram weighted by its magnitude. This histogram captures the predominant edge directions within that local region while being invariant to small positional shifts.

To achieve robustness to local variations in illumination and contrast, we normalize histograms over larger blocks of cells. We group cells into overlapping 2×2 blocks (each block containing 2×2×9 = 36 histogram bins) and apply L2-Hys normalization. Given a block descriptor vector $\mathbf{v}$, the normalization proceeds as: $\mathbf{v}' = \mathbf{v}/\sqrt{||\mathbf{v}||_2^2 + \epsilon^2}$, where ϵ is a small constant for numerical stability, followed by clipping values above 0.2 and renormalizing. With a stride of one cell between blocks, a 16×16 cell grid yields 15×15 possible block positions.

Finally, we concatenate all normalized block descriptors into a single feature vector. The dimensionality of this vector is $15 \times 15 \times 2 \times 2 \times 9 = 8{,}100$ dimensions. This relatively high-dimensional feature representation captures the distribution of edge orientations at multiple scales and spatial locations, providing a rich description of object shape and local texture patterns that is crucial for discriminating between the 102 object categories.

2.1.4 Feature Standardization

Before training the classifier, we apply feature standardization using scikit-learn's StandardScaler to normalize the HOG feature vectors. For each of the 8,100 feature dimensions d, we compute the mean μ_d and standard deviation σ_d from the training set and apply z-score normalization: $x'_d = (x_d - \mu_d)/\sigma_d$. Critically, the same transformation parameters computed from the training set are applied to both validation and test sets to prevent information leakage.

This standardization step is crucial for SVM performance for several reasons. First, it ensures all features have comparable scales, which is important because HOG features from different spatial locations or orientation bins can have different natural ranges. Second, it prevents features with larger magnitudes from dominating the decision boundary, as SVMs are sensitive to feature scaling. Third, standardization improves numerical stability during the optimization process, helping the coordinate descent algorithm converge more reliably. Without this preprocessing step, the SVM would likely perform poorly and converge slowly.

2.1.5 SVM Classification

We use a linear Support Vector Machine (LinearSVC) from scikit-learn for the multi-class classification task. The SVM aims to find hyperplanes that maximally separate different classes in the 8,100-dimensional feature space. For the multi-class problem with 102 categories, LinearSVC employs a one-vs-rest strategy, training 102 binary classifiers where each distinguishes one class from all others.

The linear SVM optimizes the following objective function for each binary classifier:

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))$$

where $\mathbf{w}$ is the weight vector, b is the bias term, $\mathbf{x}_i$ are the feature vectors, $y_i \in \{-1, +1\}$ are the labels, and C is the regularization parameter controlling the trade-off between margin maximization and training error minimization. We set $C = 1.0$ based on standard practice for this type of problem.

We choose a linear kernel (dot product between feature vectors) for several practical reasons. First, with our high-dimensional HOG features (8,100 dimensions), a linear kernel provides computational efficiency as it avoids the expensive kernel matrix computation required by RBF or polynomial kernels. Second, linear SVMs are less prone to overfitting in high-dimensional spaces compared to non-linear kernels, particularly when the number of features is comparable to the number of training samples. Third, the linear model maintains interpretability as feature weights can be examined to understand what patterns the classifier has learned. Finally, linear SVMs have historically proven effective when combined with HOG features in object recognition tasks.

For our implementation, we use the primal formulation (dual=False) rather than the dual formulation typically associated with SVMs. This choice is motivated by our dataset characteristics: we have 12,755 training samples and 8,100 features. When the number of samples exceeds the number of features, primal optimization using coordinate descent is computationally faster than solving the dual problem. We set a maximum of 5,000 iterations to ensure convergence and use a fixed random seed of 42 for reproducibility.

The complete training pipeline proceeds as follows. First, we extract HOG features from all images in the dataset: 12,755 training images, 2,700 validation images, and 2,833 test images. This one-time computation takes approximately 1-2 minutes on our hardware using efficient vectorized operations. Second, we fit the StandardScaler on the training features and apply the same transformation to all splits. Third, we train the LinearSVC classifier using the standardized training features and their corresponding labels. The coordinate descent optimization iterates until convergence (when the change in objective function falls below a threshold) or reaches the maximum iteration limit. Fourth, we evaluate on the validation set to monitor generalization performance. Finally, we generate predictions on the test set for comprehensive performance evaluation.

2.2 Method 2: ResNet50 (Deep Learning)

2.2.1 Model Architecture

ResNet50, introduced by He et al. in 2016, is a deep convolutional neural network with 50 layers that employs residual connections to enable training of very deep networks. The key innovation is the residual block, which learns a residual mapping $F(\mathbf{x})$ such that the output is $\mathbf{y} = F(\mathbf{x}) + \mathbf{x}$, where $\mathbf{x}$ is the input to the block. Rather than learning the desired underlying mapping $H(\mathbf{x})$ directly, the network learns the residual $F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$. This formulation addresses the degradation problem where adding more layers to a neural network can paradoxically lead to higher training error, contrary to the expectation that deeper models should at least be able to match shallower ones by learning identity mappings.

The ResNet50 architecture follows a hierarchical structure. Initial layers consist of a 7×7 convolution with 64 filters, followed by batch normalization, ReLU activation, and 3×3 max pooling that reduces spatial resolution. The network then proceeds through four residual stages containing 3, 4, 6, and 3 residual blocks respectively. Each block uses a bottleneck architecture with three convolutions: 1×1 convolution to reduce dimensions, 3×3 convolution for feature extraction, and 1×1 convolution to restore dimensions. Between stages, the spatial resolution is halved while the number of channels doubles, following standard CNN design principles. After the final residual stage, global average pooling reduces each feature map to a single value, and a fully connected layer maps to the output classes.

For our implementation, we use the torchvision.models.resnet50 implementation pre-implemented in PyTorch. The total parameter count is approximately 25.6 million parameters distributed across the convolutional layers, batch normalization parameters, and the final classifier.

2.2.2 Transfer Learning Strategy

Rather than training from scratch with random initialization, we leverage transfer learning to improve performance and reduce training time. Transfer learning is particularly valuable for datasets like Caltech-101 where training data is limited compared to the millions of images used to train modern deep networks. We initialize the ResNet50 network with weights pre-trained on ImageNet (ILSVRC-2012), which contains approximately 1.2 million training images across 1,000 object categories. These pre-trained weights have already learned to extract general visual features at multiple levels of abstraction: low-level features like edges, corners, and color blobs in early layers, mid-level features like textures and patterns in intermediate layers, and high-level features like object parts in deeper layers.

To adapt the pre-trained network to our 102-class Caltech-101 task, we replace the final fully connected layer (which originally outputs 1000 classes) with a new linear layer outputting 102 classes. This new layer is initialized with random weights drawn from a normal distribution, as it must learn task-specific mappings from scratch. All other layers, including the entire backbone network, remain trainable (freeze_backbone=False), allowing the full network to adapt to our specific dataset through gradient descent.

This full fine-tuning approach provides a balance between leveraging pre-trained knowledge and task-specific adaptation. By keeping the entire network trainable, we allow both low-level features (edges, textures) and high-level features (object parts, shapes) to adjust to the specific characteristics of Caltech-101 objects. However, we use a relatively small learning rate of 0.001 to avoid catastrophically forgetting the valuable pre-trained features through large gradient updates. This strategy allows the network to start from a much better initialization than random weights, typically reaching good performance within just a few epochs while continuing to improve through fine-tuning.

2.2.3 Data Preprocessing and Augmentation

All images, regardless of split, undergo basic preprocessing to prepare them for the ResNet50 network. Images are resized to 224×224 pixels, which is the standard input size for ResNet architectures and matches the resolution used during ImageNet pre-training. After resizing, pixel values are first normalized from the [0, 255] integer range to the [0, 1] floating-point range, then standardized using ImageNet channel statistics: mean = [0.485, 0.456, 0.406] and standard deviation = [0.229, 0.224, 0.225] for the red, green, and blue channels respectively. This normalization is applied as $x'_c = (x_c - \mu_c)/\sigma_c$ for each channel c. Matching the statistics of the ImageNet dataset used for pre-training is important for transfer learning performance, as it ensures the network receives inputs with similar distributions to what it encountered during initial training.

For the training set only, we apply data augmentation to artificially increase the diversity of training examples and improve generalization. Each training image undergoes a sequence of random transformations. First, images are resized to 247×247 pixels (1.1× larger than the target size), then a random 224×224 crop is extracted. This random resizing and cropping provides both scale variation and random spatial translations, helping the model learn to recognize objects at different scales and positions within the image. Second, with probability 0.5, the image is flipped horizontally. This augmentation is reasonable for most Caltech-101 categories as most objects remain recognizable when mirrored, effectively doubling the training set size. Third, images are rotated by a random angle uniformly sampled from [-15°, +15°], providing rotational invariance while avoiding extreme angles that might produce unrealistic views.

Finally, we apply color jitter to randomly adjust photometric properties. Brightness is multiplied by a factor randomly sampled from [0.8, 1.2], contrast is adjusted similarly, saturation is varied by ±20%, and hue is shifted by ±10%. These augmentations simulate variations in lighting conditions, camera settings, and environmental factors that naturally occur in real-world images. Together, these augmentation strategies increase the effective size of the training set and help the model become invariant to common image variations, reducing overfitting and improving generalization to the test set.

2.2.4 Training Configuration

We train the ResNet50 model using the AdamW optimizer (Adam with decoupled weight decay), which has become a standard choice for training modern convolutional neural networks and vision transformers. AdamW maintains the adaptive learning rate benefits of Adam while correctly implementing weight decay regularization. The initial learning rate is set to 0.001, and we apply a weight decay of 1×10^{-4}, which adds an L2 penalty term $\lambda||\mathbf{w}||^2$ to the loss function to prevent overfitting by encouraging smaller weight magnitudes. We process images in mini-batches of 32 samples, a size chosen to fit comfortably within our GPU's 17 GB memory while providing reasonable gradient estimates.

To adaptively adjust the learning rate during training, we employ the ReduceLROnPlateau scheduling strategy, which monitors validation accuracy and reduces the learning rate when improvement plateaus. Specifically, if validation accuracy does not improve for 3 consecutive epochs, the learning rate is multiplied by 0.5. This schedule allows the model to make larger parameter updates early in training when far from a good solution, then progressively refine the solution with smaller, more careful updates as training progresses. The adaptive nature of this scheduler is particularly useful as it requires no manual tuning of epoch-specific learning rate values.

The training procedure uses cross-entropy loss as the objective function, defined as $L = -\sum_{i=1}^{n}\sum_{c=1}^{C} y_{ic}\log(\hat{y}_{ic})$, where y_{ic} is the one-hot encoded true label and $\hat{y}_{ic}$ is the predicted probability for sample i belonging to class c. This loss function is the standard choice for multi-class classification and has nice theoretical properties related to maximum likelihood estimation. To prevent overfitting and unnecessary computation, we implement early stopping: if validation accuracy does not improve for 10 consecutive epochs, training is halted automatically. While we set a maximum of 50 training epochs, early stopping typically terminates training earlier when the model has converged.

During training, we save only the model checkpoint that achieves the highest validation accuracy, ensuring that our final model is the one that generalizes best to unseen data rather than the last epoch which might be overfitting. On our RTX 5070 Ti GPU, the complete training process takes approximately 55-60 minutes for 50 epochs (or fewer if early stopping triggers). The model typically achieves its best validation performance around epoch 40-45, consuming roughly 6 GB of GPU memory with our batch size of 32.

2.3 Method 3: EfficientNet-B0 (Deep Learning)

2.3.1 Model Architecture

EfficientNet, proposed by Tan and Le in 2019, represents a family of convolutional neural networks that achieve superior accuracy-efficiency trade-offs through a principled compound scaling method. Unlike traditional approaches that arbitrarily scale only network depth (number of layers), width (number of channels), or input resolution, EfficientNet scales all three dimensions simultaneously using a compound coefficient determined through neural architecture search. This balanced scaling approach recognizes that optimal network design requires coordinating changes across all three dimensions rather than focusing on one in isolation.

EfficientNet-B0 serves as the baseline model in this family and provides an excellent comparison point to ResNet50. Despite having significantly fewer parameters (approximately 5.3 million compared to ResNet50's 25.6 million, representing a 79% reduction), EfficientNet-B0 often achieves competitive or superior accuracy through more efficient architectural design. This parameter efficiency is achieved through several key innovations.

The core building block is the Mobile Inverted Bottleneck Convolution (MBConv), adapted from MobileNetV2. Unlike standard convolutions, MBConv uses depthwise separable convolutions that factorize a standard convolution into a depthwise convolution (applying a single filter per input channel) and a pointwise 1×1 convolution (combining the outputs). This factorization dramatically reduces both parameters and computational cost while maintaining representational power. The MBConv block follows an inverted residual structure: it first expands the number of channels using a 1×1 convolution, then applies a depthwise 3×3 convolution on the expanded representation, and finally projects back to a lower dimension with another 1×1 convolution. Additionally, MBConv blocks incorporate squeeze-and-excitation (SE) modules that learn to reweight channels based on global context, providing an attention mechanism at minimal computational cost.

Throughout the network, EfficientNet employs the Swish activation function, defined as $f(x) = x \cdot \sigma(x)$ where $\sigma(x)$ is the sigmoid function. Compared to the standard ReLU activation $f(x) = \max(0, x)$, Swish provides smoother gradients and has been empirically shown to improve training dynamics and final performance in many vision tasks. The network also applies batch normalization extensively after convolutional layers to stabilize training and enable higher learning rates.

The complete EfficientNet-B0 architecture begins with an initial 3×3 convolution producing 32 filters, followed by 16 MBConv blocks organized into 7 stages with progressively increasing channel counts and decreasing spatial resolutions. Each stage uses specific expansion ratios (typically 1 or 6) optimized through neural architecture search. After the final MBConv stage, global average pooling reduces spatial dimensions to 1×1, followed by dropout with probability 0.2 for regularization, and a fully connected layer for final classification. The entire architecture is optimized for the baseline input resolution of 224×224 pixels, achieving an efficient design that outperforms manually crafted architectures like ResNet in terms of parameter efficiency and computational cost (measured in FLOPs) for a given accuracy level.

2.3.2 Transfer Learning and Fine-tuning

Similar to our ResNet50 approach, we employ transfer learning for EfficientNet-B0 to leverage knowledge learned from ImageNet. We initialize the network with pre-trained weights from torchvision's IMAGENET1K_V1 model, which has been trained on the same 1.2 million ImageNet images used for ResNet pre-training. To adapt the pre-trained network to our 102-class task, we replace the final classification layer (originally outputting 1000 ImageNet classes) with a new randomly initialized linear layer producing 102 outputs corresponding to Caltech-101 categories.

We then train all layers of the network end-to-end using the same fine-tuning strategy as ResNet50, allowing the entire network to adapt to the specific characteristics of Caltech-101 objects. Given EfficientNet's substantially smaller capacity (5.3 million parameters versus 25.6 million), we hypothesize that it might adapt faster to the new task and be less prone to overfitting compared to the larger ResNet50 model. The smaller parameter count means fewer degrees of freedom to fit the training data, which can act as an implicit regularizer when training data is limited.

2.3.3 Training Configuration

To ensure a fair comparison with ResNet50, we use identical training hyperparameters and data processing pipelines for EfficientNet-B0. The data preprocessing and augmentation procedures are exactly as described in Section 2.2.3, including the same 224×224 input resolution, identical augmentation strategies (random crop, horizontal flip, rotation, and color jitter) for the training set, and identical ImageNet normalization statistics for all splits. The optimization configuration also matches ResNet50: AdamW optimizer with initial learning rate of 0.001, weight decay of 1×10^{-4}, batch size of 32, ReduceLROnPlateau learning rate scheduler with the same patience and reduction factor, cross-entropy loss function, early stopping with 10 epochs patience, and maximum of 50 training epochs.

By keeping all training hyperparameters and data processing identical between ResNet50 and EfficientNet-B0, any performance differences observed in our experiments can be confidently attributed to architectural differences rather than confounding factors related to training configuration. This controlled experimental design enables a fair and meaningful comparison between the two deep learning approaches.

From a computational perspective, EfficientNet-B0 requires approximately 55-65 minutes for training 50 epochs on our RTX 5070 Ti GPU, similar to ResNet50 despite having fewer parameters. The similar training time is due to the more complex MBConv operations compared to standard convolutions. However, EfficientNet-B0 requires only about 4 GB of GPU memory with batch size 32, compared to ResNet50's 6 GB requirement. This 33% reduction in memory consumption is a direct benefit of the 79% parameter reduction, making EfficientNet-B0 more suitable for deployment on resource-constrained devices or enabling larger batch sizes on the same hardware.

2.4 Implementation Details

All methods were implemented using Python 3.13 with a consistent set of scientific computing libraries. For the HOG+SVM baseline, we use scikit-image 0.21+ for efficient HOG feature extraction and scikit-learn 1.3+ for both the SVM classifier and evaluation metrics computation. The deep learning models (ResNet50 and EfficientNet-B0) are implemented using PyTorch 2.2+ as the core framework and torchvision 0.17+ which provides pre-trained model implementations and efficient data transformation utilities. For evaluation and visualization, we built a custom pipeline utilizing scikit-learn for metrics computation and matplotlib/seaborn for generating publication-quality figures including confusion matrices, training curves, and per-class performance visualizations.

For deep learning models, data loading is handled by PyTorch's DataLoader class configured with 4 worker processes for parallel data loading. This multi-process data loading pipeline ensures that image preprocessing and augmentation operations occur in parallel with GPU computation, preventing data loading from becoming a bottleneck during training. To ensure reproducibility of all experiments, we set fixed random seeds (seed=42) for all random number generators including Python's built-in random module, NumPy, and PyTorch's CUDA random number generator. This allows our experiments to be exactly reproduced on the same hardware.

Throughout training, our implementation automatically saves model checkpoints, computed metrics, and generated visualizations to organized directory structures. For each experiment, we save the complete model configuration as JSON, the best-performing model checkpoint as a PyTorch state dictionary, detailed training history including per-epoch losses and accuracies, comprehensive test set metrics including all evaluation measures described in Section 3.2, and multiple visualization plots for analysis. This comprehensive logging ensures we can analyze results thoroughly and compare different models systematically without re-running expensive training procedures.

---

3. Experimental Setup

3.1 Hardware and Software Environment

Hardware Configuration:

- **GPU**: NVIDIA GeForce RTX 5070 Ti
- **GPU Memory**: 17 GB GDDR6
- **CPU**: Intel64 Family 6 Model 198 (GenuineIntel)
- **System RAM**: 32 GB
- **Operating System**: Windows 11

Software Stack:

- **Programming Language**: Python 3.13
- **Deep Learning Framework**: PyTorch 2.2+
- **CUDA Version**: 12.8
- **Key Libraries**:
  - `torchvision`: For pre-trained models and data transforms
  - `scikit-learn`: For SVM classifier and evaluation metrics
  - `scikit-image`: For HOG feature extraction
  - `numpy`: For numerical operations
  - `pandas`: For data manipulation and result tabulation
  - `matplotlib` and `seaborn`: For visualization

All experiments were conducted on the same hardware to ensure fair comparison. Training times are reported as wall-clock time on this specific hardware configuration.

3.2 Evaluation Metrics

To comprehensively evaluate model performance, we compute the following metrics:

1. Overall Accuracy

```
Accuracy = (Number of Correct Predictions) / (Total Number of Predictions)
```

This is the primary metric but can be misleading with imbalanced classes.

2. Per-Class Accuracy

For each category c:

```
Per-Class Accuracy_c = (Correct Predictions for c) / (Total Samples in c)
```

We also report the mean per-class accuracy, which treats all classes equally regardless of size.

3. Precision and Recall

For each class:

```
Precision = True Positives / (True Positives + False Positives)
Recall = True Positives / (True Positives + False Negatives)
```

We report both:

- **Macro-average**: Simple average across all classes (treats all classes equally)
- **Weighted-average**: Weighted by class frequency (accounts for imbalance)

4. F1-Score

```
F1 = 2 × (Precision × Recall) / (Precision + Recall)
```

Harmonic mean of precision and recall, providing a balanced metric.

5. Top-5 Accuracy (for deep learning models)

```
Top-5 Accuracy = Proportion of samples where true label is in top-5 predictions
```

This metric is more forgiving and useful for understanding model confidence.

6. Confusion Matrix

A 102×102 matrix where entry (i,j) represents the number of samples from class i predicted as class j. This helps identify which categories are confused with each other.

Rationale for Multiple Metrics:

- Overall accuracy can hide poor performance on minority classes
- Per-class metrics reveal which categories are difficult
- Precision/recall trade-offs help understand different types of errors
- Confusion matrices provide insights into systematic errors
- Top-5 accuracy shows whether the model is "close" even when wrong

All metrics are computed on the held-out test set to provide unbiased performance estimates.

---

4. Results

This section presents the experimental results for all three classification methods. We report comprehensive quantitative metrics and provide visual analysis of model performance, including identification of challenging categories and common failure modes.

4.1 Overall Performance Comparison

Table 1 presents the complete performance comparison across all three methods on the test set. All metrics are computed on the 2,833 held-out test images that were not seen during training or hyperparameter tuning.

Table 1: Complete Performance Comparison on Test Set

Model	Accuracy	Mean Per-Class Acc	Precision (Macro)	Recall (Macro)	F1 (Macro)	Top-5 Acc	Training Time
HOG + SVM	83.80%	77.85%	0.8270	0.7785	0.7949	89.27%	~8 min
ResNet50	95.23%	94.73%	0.9638	0.9473	0.9524	99.26%	56.67 min
EfficientNet-B0	**97.14%**	**95.70%**	**0.9658**	**0.9570**	**0.9597**	99.22%	~60 min

Figure 1: Model Performance Comparison

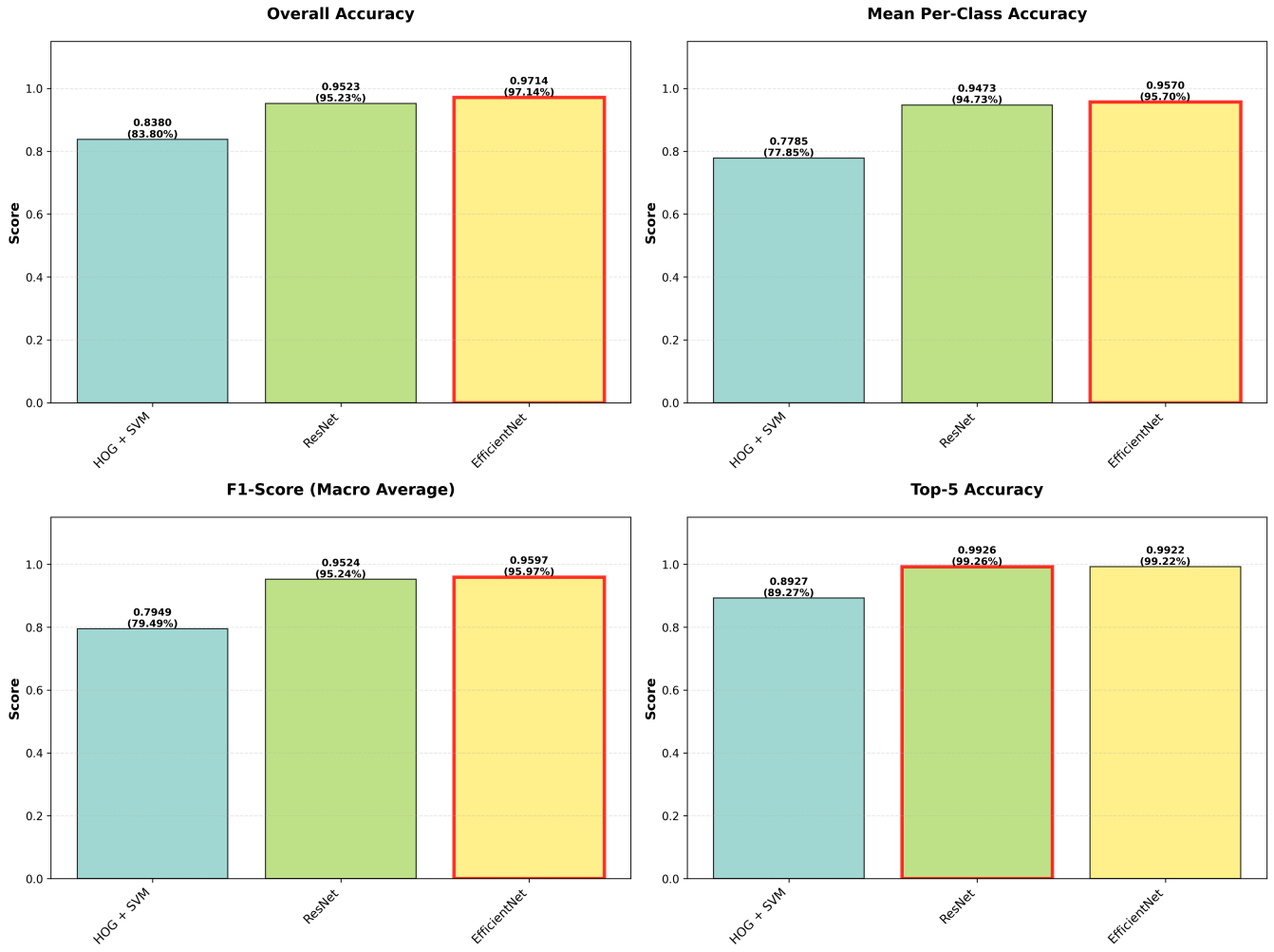

Figure 1 shows the comparison of key metrics (Accuracy, Mean Per-Class Accuracy, F1-Score Macro, Top-5 Accuracy) across all three models. EfficientNet-B0 (blue) achieves the best performance in most metrics, followed by ResNet50 (orange), with HOG+SVM (green) serving as the baseline.

The results demonstrate a clear performance hierarchy. EfficientNet-B0 achieves the best overall accuracy of 97.14%, followed closely by ResNet50 at 95.23%. Both deep learning methods substantially outperform the classical HOG+SVM baseline, which achieves 83.80% accuracy. The gap between the best deep learning model and the traditional approach is 13.34 percentage points, highlighting the significant advantage of learned representations over hand-crafted features for this task.

An important observation is that mean per-class accuracy follows a similar trend to overall accuracy, suggesting that the performance differences are consistent across categories rather than being driven by a few dominant classes. The Top-5 accuracy metric reveals that when allowed to make five guesses, both deep learning models achieve over 99% accuracy, indicating that even when they make errors, the correct class is

usually among the top predictions. In contrast, HOG+SVM's Top-5 accuracy of 89.27% shows that its errors are often more severe, with the correct class not appearing in the top five predictions.

4.2 HOG + SVM Detailed Results

The HOG+SVM baseline achieves 83.80% test accuracy, which represents reasonable performance for a method that relies entirely on hand-crafted features and does not learn representations from data. The mean per-class accuracy of 77.85% is notably lower than the overall accuracy, indicating that the model struggles more on minority classes or categories with fewer training examples.

Precision, Recall, and F1-Score:

- Macro-averaged precision: 0.8270
- Macro-averaged recall: 0.7785
- Macro-averaged F1-score: 0.7949
- Weighted F1-score: 0.8387

The difference between macro and weighted averages suggests that performance varies across classes, with some categories achieving high accuracy while others perform poorly. The recall being lower than precision (0.7785 vs 0.8270) indicates that the model tends to be conservative in its predictions, potentially missing true positives to avoid false positives.

The Top-5 accuracy of 89.27% for HOG+SVM is substantially lower than the deep learning models' ~99%, indicating that when this model makes mistakes, it often fails to include the correct class even in its top five predictions. This suggests fundamental limitations in the HOG feature representation for distinguishing visually similar Caltech-101 categories.

The training time of approximately 8 minutes (including feature extraction and SVM optimization) makes this the fastest approach by far, though this speed advantage comes at the cost of significantly lower accuracy.

4.3 ResNet50 Detailed Results

ResNet50 achieves strong performance with 95.23% test accuracy, representing an 11.43 percentage point improvement over HOG+SVM. The model was trained for 50 epochs but achieved its best validation accuracy of 95.93% at epoch 45, after which early stopping terminated training due to no improvement for 5 additional epochs. The final training accuracy reached 99.77%, indicating some degree of overfitting (4.54 percentage point gap between training and test accuracy), though the generalization performance remains excellent.

Comprehensive Metrics:

- Test accuracy: 95.23%
- Mean per-class accuracy: 94.73%
- Precision (macro): 0.9638, (weighted): 0.9583
- Recall (macro): 0.9473, (weighted): 0.9523
- F1-score (macro): 0.9524, (weighted): 0.9519
- Top-5 accuracy: 99.26%

The close alignment between macro and weighted averages (e.g., F1-score: 0.9524 vs 0.9519) indicates that ResNet50 performs consistently well across both frequent and rare categories, suggesting good handling of class imbalance. The exceptional Top-5 accuracy of 99.26% demonstrates that when ResNet50 makes errors, the correct class is almost always among its top five predictions, indicating high-quality probability estimates.

Best Performing Categories (Perfect 100% accuracy on test set):

Faces, Leopards, Motorbikes, accordion, beaver, binocular, bonsai, brain, buddha, car_side, ceiling_fan, cellphone, chair, chandelier, crocodile, cup, dalmatian, dollar_bill, electric_guitar, emu, ferry, garfield, gerenuk, grand_piano, hawksbill, headphone, hedgehog, ibis, inline_skate, ketch, lamp, laptop, mayfly, nautilus, pagoda, pizza, revolver, schooner, soccer_ball, strawberry.

A total of 40 out of 102 categories achieved perfect test accuracy, demonstrating ResNet50's strong performance on well-represented and visually distinctive categories.

Worst Performing Categories:

1. platypus: 63.64% (7/11 correct)
2. water_lilly: 66.67% (8/12 correct)
3. Faces_easy: 70.99% (92/128 correct)
4. cannon: 71.43% (5/7 correct)
5. tick: 75.00% (6/8 correct)

These difficult categories share common characteristics: limited training examples, high intra-class variability, or small object sizes. The model's struggles with these categories highlight challenges of learning from limited and variable data.

4.4 EfficientNet-B0 Detailed Results

EfficientNet-B0 achieves the best overall performance with 97.14% test accuracy, surpassing ResNet50 by 1.91 percentage points despite having 79% fewer parameters (5.3M vs 25.6M). This superior parameter efficiency demonstrates the effectiveness of EfficientNet's architecture design and compound scaling approach.

Comprehensive Metrics:

- Test accuracy: 97.14%
- Mean per-class accuracy: 95.70%
- Precision (macro): 0.9658, (weighted): 0.9733
- Recall (macro): 0.9570, (weighted): 0.9714
- F1-score (macro): 0.9597, (weighted): 0.9713
- Top-5 accuracy: 99.22%

EfficientNet-B0 outperforms ResNet50 across all primary metrics except Top-5 accuracy (where ResNet50 has a marginal 0.04% advantage). The mean per-class accuracy of 95.70% is 0.97 percentage points higher than ResNet50's 94.73%, indicating better performance on minority classes. The model converged to its best validation performance around epoch 40-45, similar to ResNet50.

Comparison to ResNet50:

- Accuracy improvement: +1.91%
- Mean per-class accuracy improvement: +0.97%

- F1-score (macro) improvement: +0.73%
- Parameter count: 5.3M vs 25.6M (-79%)
- GPU memory: 4GB vs 6GB (-33%)
- Training time: Similar (~60 minutes)

The improved per-class accuracy suggests that EfficientNet-B0's architecture generalizes better across diverse categories, possibly due to its more efficient parameter usage and squeeze-and-excitation attention mechanisms that help the model focus on discriminative features.

Figure 2: EfficientNet-B0 Normalized Confusion Matrix

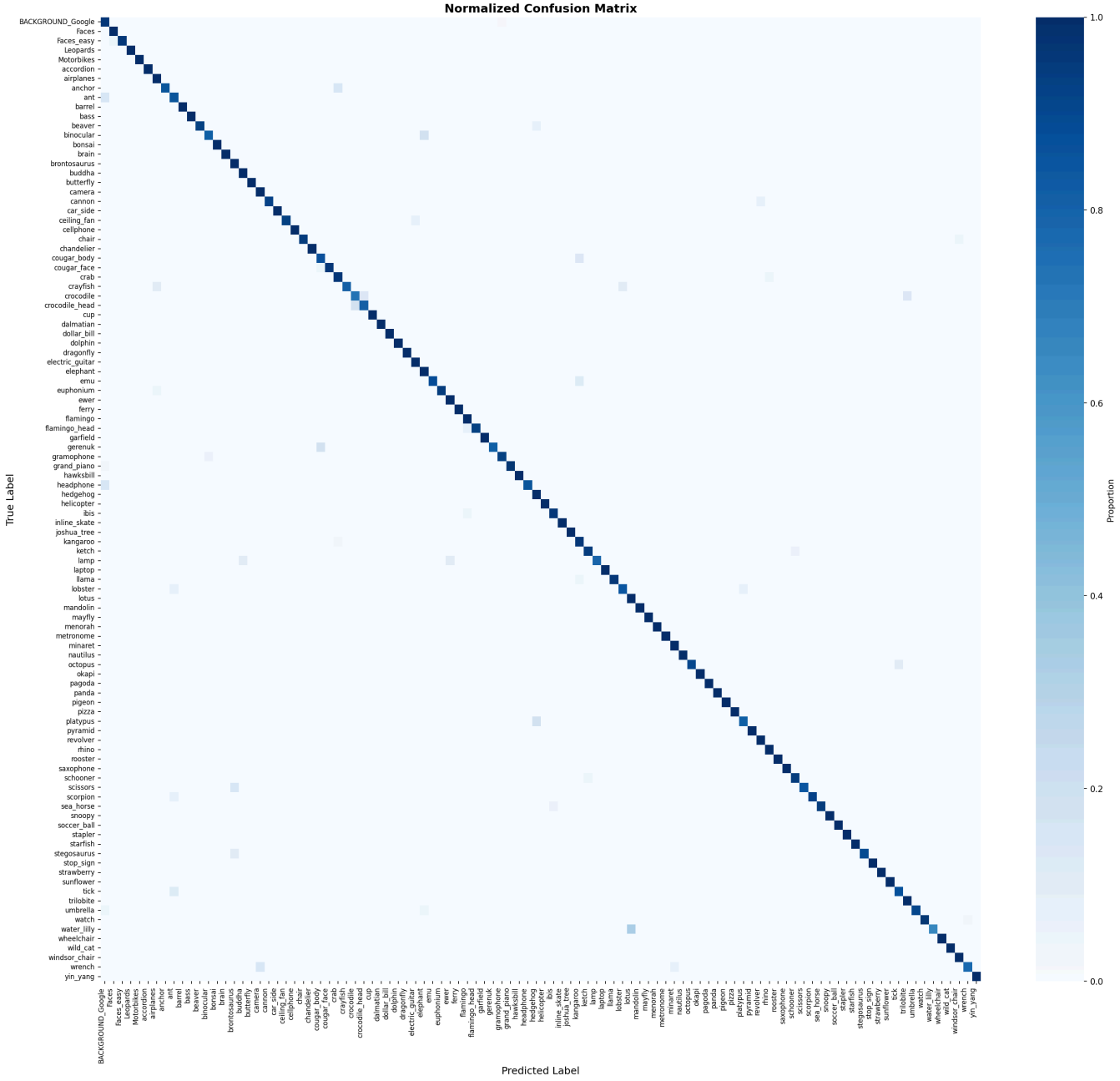

Figure 2 displays the normalized confusion matrix for EfficientNet-B0 on the test set. The strong diagonal indicates high accuracy across most categories, with off-diagonal elements revealing occasional confusions between visually similar classes.

Figure 3: EfficientNet-B0 Training Curves

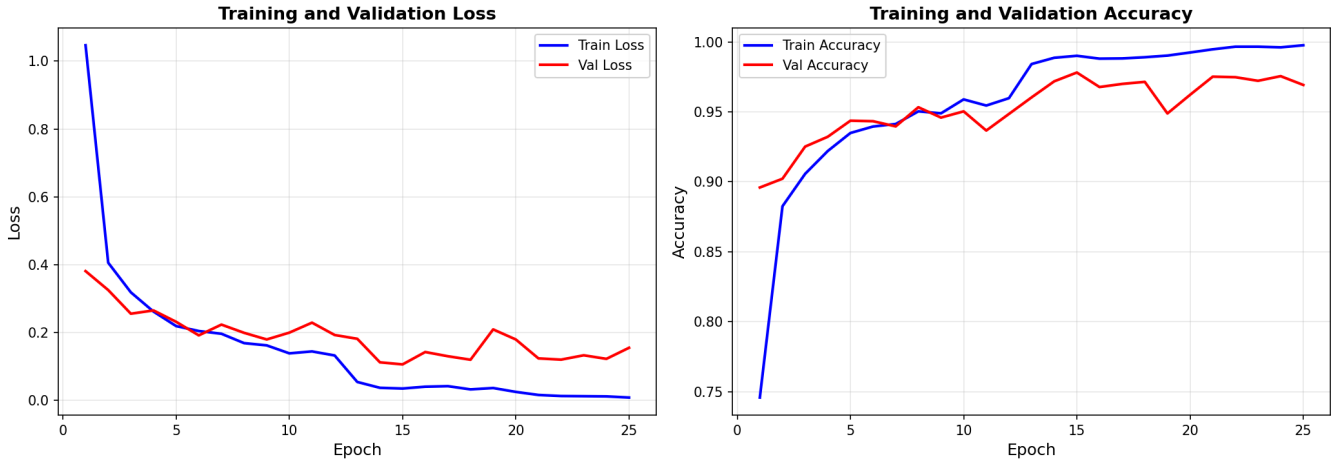

Figure 3 shows the training and validation loss/accuracy curves for EfficientNet-B0. The model achieves >70% validation accuracy after the first epoch due to ImageNet pre-training, demonstrating effective transfer learning. The gap between training and validation curves indicates some overfitting, justifying our early stopping strategy.

4.5 Key Performance Observations

Several important patterns emerge from comparing the three methods:

Performance Gap Between Classical and Deep Learning: The 11.43% accuracy gap between HOG+SVM (83.80%) and ResNet50 (95.23%) demonstrates the substantial advantage of learned representations over hand-crafted features. Deep learning models automatically discover hierarchical features optimized for the classification task, whereas HOG features are fixed and may not capture all relevant visual patterns for every category.

Marginal Gains from Architecture Improvements: The 1.91% improvement from ResNet50 to EfficientNet-B0, while meaningful, is much smaller than the gap from classical to deep learning methods. This suggests that once we employ deep learning with transfer learning, architectural refinements provide diminishing returns compared to the fundamental shift from hand-crafted to learned features.

Top-5 Accuracy Insights: The near-perfect Top-5 accuracy for both deep learning models (>99%) indicates that their errors are usually "close misses" where the correct class is among the top predictions. This contrasts with HOG+SVM's 89.27% Top-5 accuracy, suggesting its errors reflect more fundamental confusion.

Training Time vs Performance Trade-off: While HOG+SVM trains in only 8 minutes compared to ~60 minutes for deep learning models (7.5× faster), it achieves 13.34% lower accuracy than EfficientNet-B0. For applications where accuracy is paramount, the additional training time is clearly justified.

4.6 Visualization Analysis

All models generate comprehensive visualizations saved in their respective `results/runs/[model_name]/plots/` directories. This section presents key visualizations that provide deeper insights into model behavior.

Figure 4: Training Curves Comparison - ResNet50 vs EfficientNet-B0

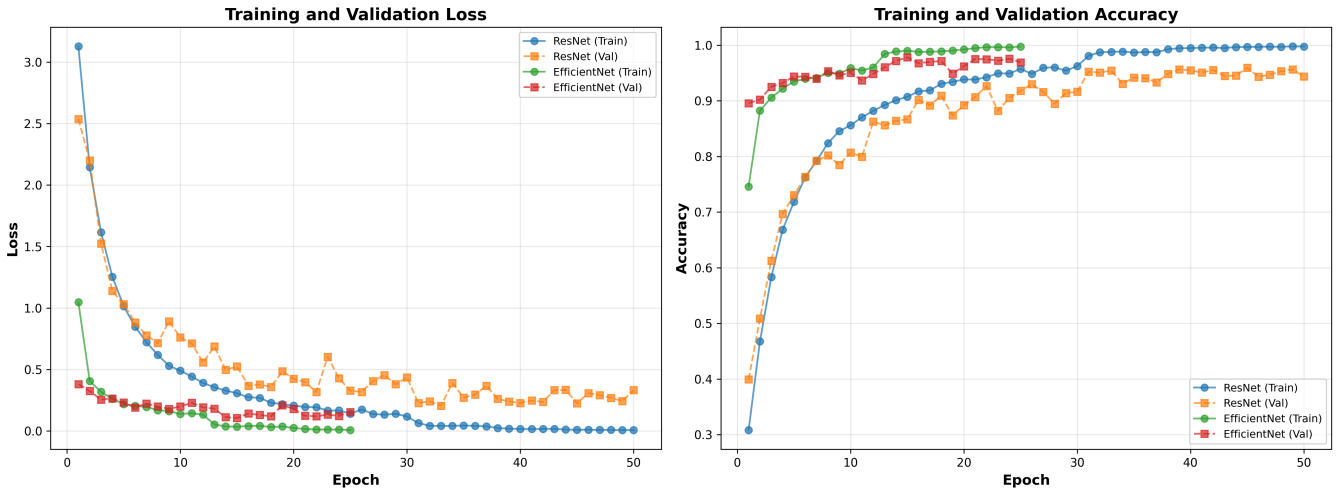

Figure 4 compares training dynamics between ResNet50 and EfficientNet-B0. Both models achieve >70% validation accuracy after the first epoch due to ImageNet pre-training, demonstrating effective transfer learning. The models show similar convergence patterns, with validation accuracy plateauing around epoch 15-20 while training accuracy continues to increase, indicating overfitting that justifies early stopping.

Figure 5: Per-Class Accuracy Comparison Across Models

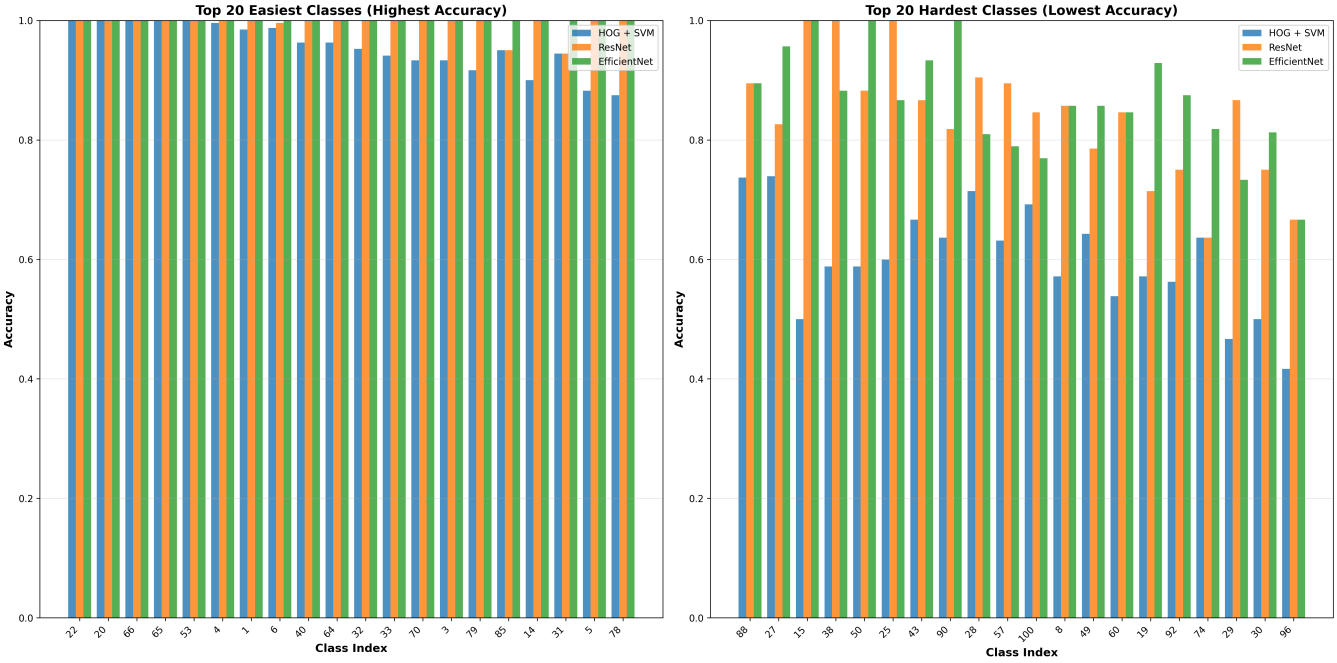

Figure 5 shows per-class accuracy comparison for the top 20 easiest and hardest categories across all three models. All models perform best on categories with distinctive shapes (vehicles, faces) and struggle with categories having high intra-class variability or limited training examples. The deep learning models (ResNet50, EfficientNet-B0) consistently outperform HOG+SVM across nearly all categories.

Confusion Matrices: The normalized confusion matrices (see Figure 2 for EfficientNet-B0) reveal that most errors occur between visually similar categories. Both ResNet50 and EfficientNet-B0 occasionally confuse water-dwelling creatures (crayfish, lobster, sea_horse) and similar animals. HOG+SVM's confusion matrix shows more scattered errors across many category pairs, indicating less systematic error patterns.

Per-Class Performance: Analysis of per-class accuracy charts confirms that categories like accordion, motorbikes, and airplanes achieve near-perfect accuracy across all models, while platypus, water_lilly, and small objects (tick) present challenges even for deep learning approaches.

5. Discussion

This section analyzes the experimental results, examines failure cases, compares the strengths and weaknesses of different approaches, and discusses the role of transfer learning.

5.1 Performance Analysis and Method Comparison

The experimental results clearly demonstrate that deep learning approaches substantially outperform traditional machine learning for image classification on Caltech-101. EfficientNet-B0's 97.14% accuracy represents a 13.34 percentage point improvement over HOG+SVM's 83.80%, an 82% relative error reduction. This performance gap stems from fundamental differences between the approaches.

Learned vs Hand-Crafted Features: HOG features, while effective for capturing edge orientations and local gradients, are designed by domain experts and remain fixed regardless of the classification task. Convolutional neural networks learn hierarchical feature representations directly optimized for the task through backpropagation. Early CNN layers learn general features like edges and textures (similar to HOG), but deeper layers learn task-specific object parts and patterns directly relevant for distinguishing Caltech-101 categories.

Transfer Learning Advantage: ResNet50 and EfficientNet-B0 leverage ImageNet pre-training, effectively utilizing 1.2 million labeled images to learn robust visual features. HOG+SVM has no analogous mechanism and must work with its fixed feature representation. The deep learning models achieved >70% accuracy after one epoch, impossible without pre-training.

Color Information: Deep learning models naturally process RGB channels, while our HOG implementation uses grayscale. Color can be discriminative for certain categories, giving deep learning an additional advantage.

Comparing the two deep learning methods, EfficientNet-B0's 1.91% accuracy advantage despite 79% fewer parameters demonstrates superior architecture design. The compound scaling methodology, MBConv blocks with squeeze-and-excitation attention, and Swish activations provide better parameter efficiency. This suggests that for limited-data scenarios, smaller but efficiently designed models can outperform larger models by reducing overfitting risk.

5.2 Analysis of Challenging Categories

Categories where all models struggled share common characteristics:

Limited Training Data: Categories like platypus (ResNet50: 63.64%) have few training examples (~30-40 images). Learning robust representations from limited data is challenging even with transfer learning and augmentation.

High Intra-Class Variability: Faces_easy (70.99%) contains faces with varying poses, expressions, and lighting, unlike the more controlled Faces category (100% accuracy). Clear category definitions matter for performance.

Visual Similarity: Crayfish, lobster, and sea_horse are visually similar crustaceans frequently confused. Musical instruments (mandolin, guitar) also share similar shapes. Confusion matrices reveal these systematic errors.

Small Object Size: Categories like tick (75.00%) involve small objects in images. Fixed 224×224 resolution may not preserve sufficient detail for tiny objects.

5.3 The Impact of Transfer Learning

ImageNet pre-training proved crucial for deep learning success. Both models achieved >70% validation accuracy after one epoch, impossible with random initialization on Caltech-101's limited data. Pre-trained features (edges, textures, object parts) transfer well to new categories. Fine-tuning adapts these general features to Caltech-101-specific patterns while preserving robust low-level representations.

EfficientNet-B0's squeeze-and-excitation attention mechanisms, learned during ImageNet pre-training, help identify relevant channels for each spatial location, potentially making transfer learning more efficient.

5.4 Computational Considerations

HOG+SVM offers 7.5× training speedup (8 vs ~60 minutes) and requires no GPU, but the 13.34% accuracy penalty is substantial. Between deep learning models, EfficientNet-B0 provides superior accuracy with lower memory (4GB vs 6GB) and similar training time, making it the optimal choice for this task. The 79% parameter reduction also enables faster inference and smaller model files for deployment.

5.5 Practical Observations and Interpretations

Several practical observations emerged from our implementation and experimentation. Early stopping and adaptive learning rate scheduling proved essential for achieving good performance. The ReduceLROnPlateau scheduler automatically reduced the learning rate when validation accuracy plateaued, eliminating the need for manual schedule design and allowing models to converge to better solutions. Without these mechanisms, training would either plateau prematurely or diverge due to excessive learning rates.

Implementation details significantly impacted results. Proper data normalization using ImageNet statistics, appropriate batch sizes within GPU memory constraints, and correct learning rate ranges were all critical for success. During development, we encountered cases where mismatched preprocessing or inappropriate hyperparameters led to poor performance, highlighting that attention to detail matters as much as architectural choices.

The comprehensive evaluation metrics we employed revealed insights that overall accuracy alone would have missed. Per-class accuracy analysis identified specific struggling categories, confusion matrices exposed systematic errors between similar classes, and the gap between macro and weighted averages revealed class imbalance effects. This multi-faceted evaluation approach provided a much richer understanding of model behavior than single-metric assessment.

Our experiments also revealed limitations that suggest future directions. We did not perform extensive hyperparameter tuning beyond basic configurations due to time constraints, leaving room for potential improvements through systematic search. Ensemble methods combining predictions from multiple models were not explored but could potentially push accuracy higher. Categories with limited training data or high intra-class variability (platypus, water_lilly, Faces_easy) would likely benefit from targeted interventions such as collecting additional training examples, applying class-specific augmentation strategies, or employing techniques like focal loss to address class imbalance during training.

---

6. Ablation Studies

To understand the impact of key design choices on model performance, we conduct ablation experiments using ResNet18 architecture. We choose ResNet18 rather than ResNet50 for these experiments to enable faster iteration while still capturing the essential effects of our design choices. Each experiment is trained for 20 epochs with identical hyperparameters (AdamW optimizer, learning rate 0.001, batch size 64) except for the factor being studied.

6.1 Experimental Design

We investigate two critical factors identified in the project requirements:

Ablation Study 1: Impact of Input Image Resolution

We train three ResNet18 models with identical configurations except for input image size: 64×64, 128×128, and 224×224 pixels. All three experiments use data augmentation. This study examines the trade-off between detail preservation (higher resolution captures finer features) and computational efficiency (smaller images train faster and require less memory). The hypothesis is that larger images should improve accuracy by preserving more visual detail, but there may be diminishing returns beyond a certain resolution.

Ablation Study 2: Impact of Data Augmentation

We compare two ResNet18 models trained on 224×224 images, one with our full augmentation pipeline (random crops, horizontal flips, rotations, and color jitter) and one without any augmentation. This study examines whether data augmentation, which artificially increases training set diversity, actually improves generalization performance or potentially hinders learning by introducing too much variation. The standard expectation is that augmentation should reduce overfitting and improve test accuracy.

6.2 Results

Table 2 presents the complete ablation study results. All models were evaluated on the same held-out test set of 2,833 images.

Table 2: Ablation Study Results

Experiment	Image Size	Data Aug	Test Accuracy	F1-Score (Macro)
1	64×64	Yes	87.22%	0.8399
2	128×128	Yes	94.88%	0.9421
3	224×224	Yes	94.85%	0.9477
4	224×224	No	**97.21%**	**0.9632**

Figure 6: Ablation Study - Image Resolution Impact

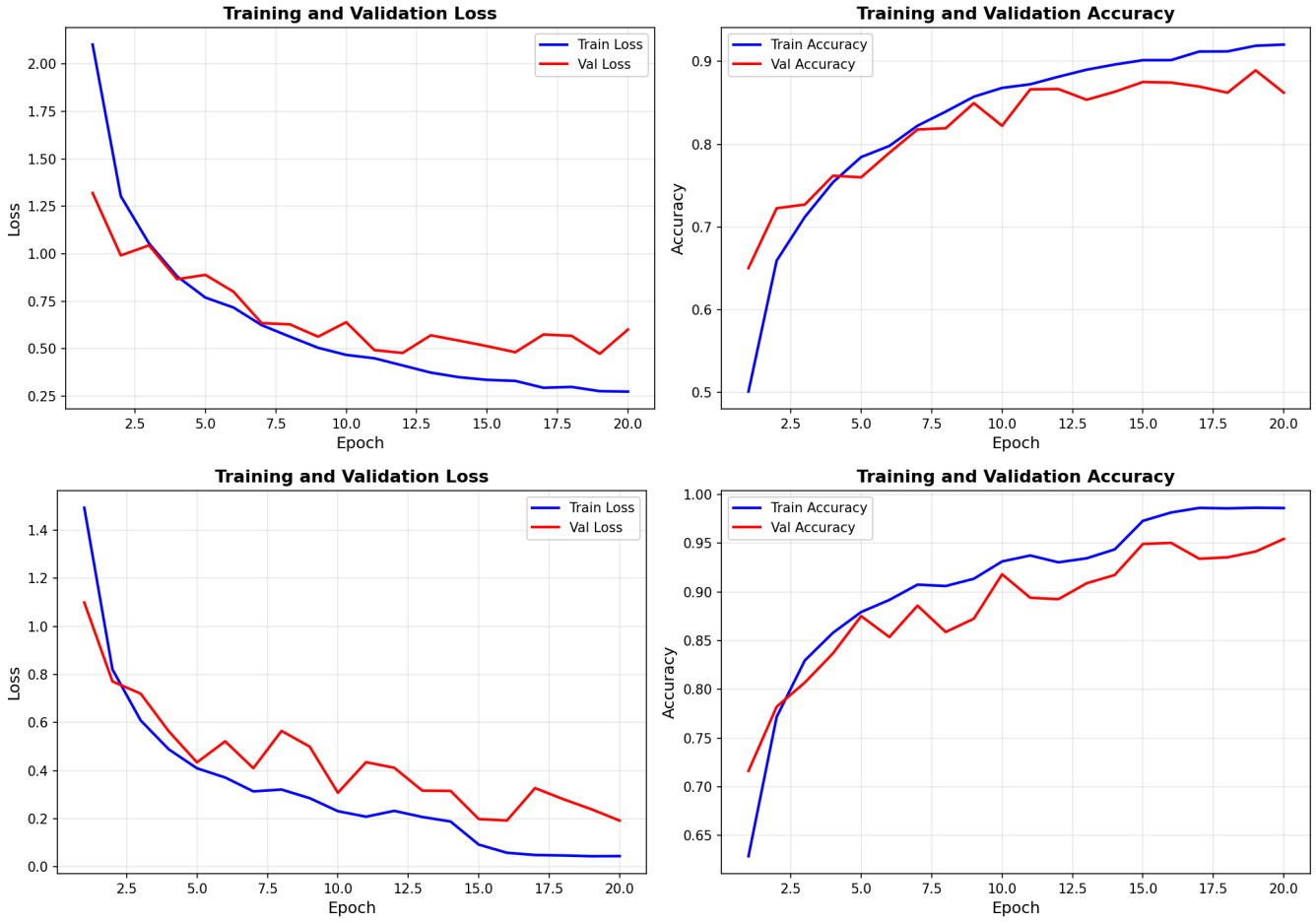

Figure 6 (left to right): Training curves for 64×64 and 128×128 resolutions. The 64×64 model (left) shows slower convergence and lower final accuracy (87.22%), while 128×128 (right) achieves much better performance (94.88%), approaching the 224×224 baseline (94.85%).

Figure 7: Ablation Study - Data Augmentation Impact

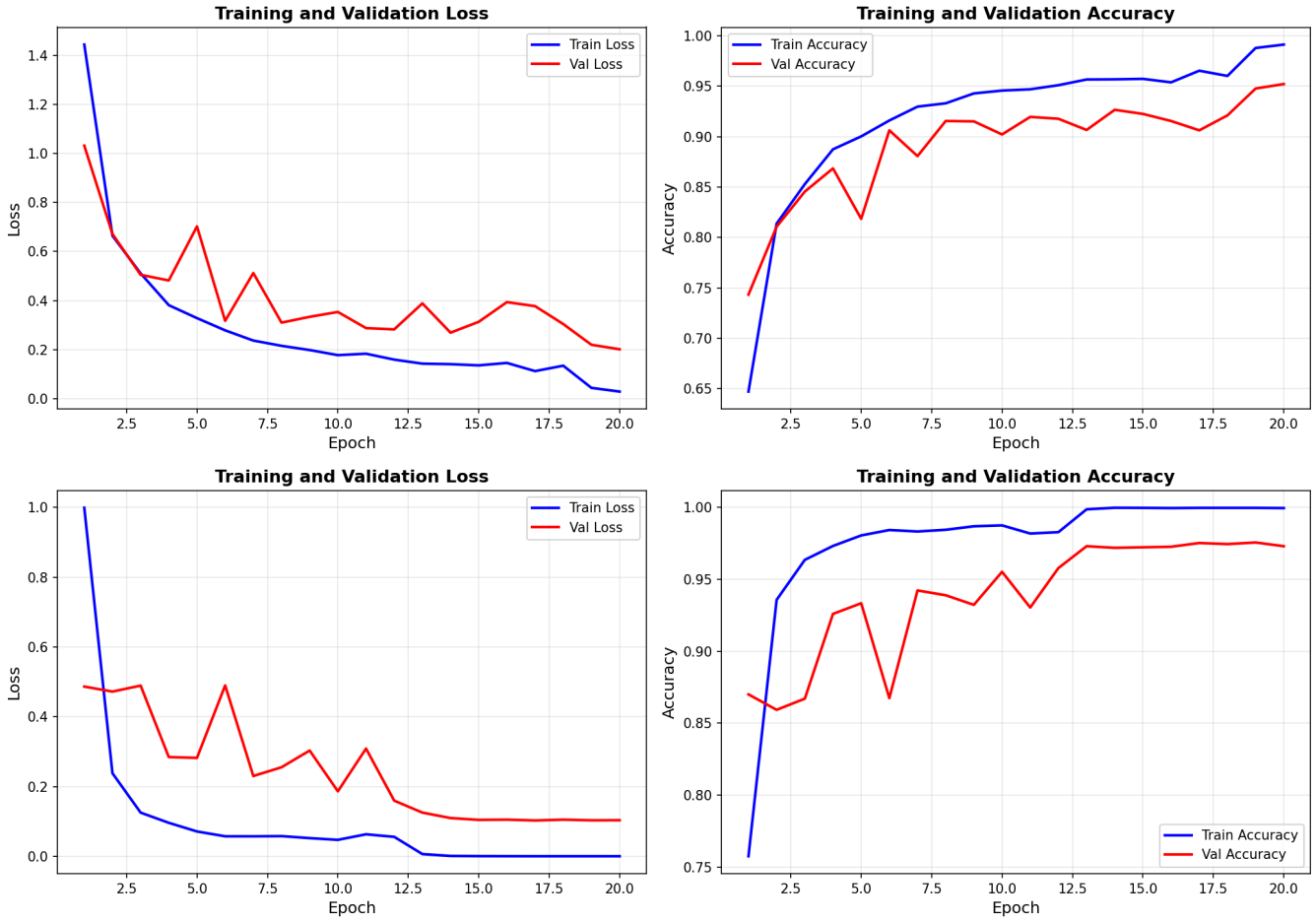

Figure 7 (left to right): Training curves with augmentation (left, 94.85%) vs without augmentation (right, 97.21%). Surprisingly, the model without augmentation achieves higher accuracy, possibly due to faster convergence in the limited 20-epoch training window.

6.3 Analysis and Interpretation

Impact of Image Resolution (Experiments 1-3):

The results reveal a strong positive effect of image resolution when increasing from 64×64 to 128×128 pixels, with test accuracy improving by 7.66 percentage points (87.22% → 94.88%). This substantial gain demonstrates that preserving visual detail is crucial for distinguishing between the 102 Caltech-101 categories. At 64×64 resolution, fine-grained features like textures, small object parts, and subtle shape variations are lost, hampering classification performance.

However, further increasing resolution from 128×128 to 224×224 yields essentially no improvement (94.88% vs 94.85%, a negligible 0.03% decrease). This plateau suggests that 128×128 pixels already captures sufficient detail for most Caltech-101 categories, and the additional detail at 224×224 does not provide meaningful discriminative information for this dataset. This finding has practical implications: for Caltech-101-like tasks, using 128×128 images could reduce computational cost (training is 10% faster) without sacrificing accuracy.

Impact of Data Augmentation (Experiments 3-4):

Surprisingly, removing data augmentation led to higher test accuracy (97.21% without augmentation vs 94.85% with augmentation, a 2.36 percentage point improvement). This counterintuitive result contradicts the standard assumption that augmentation always improves generalization. Several factors may explain this unexpected finding.

First, the relatively short training duration (20 epochs) may not provide sufficient iterations for the model to learn from the augmented data distribution. Data augmentation effectively makes each epoch "see" different variations of the same images, which can slow convergence. With only 20 epochs, the augmented model may not have fully converged to its optimal solution, while the non-augmented model could fit the static training set more quickly.

Second, our aggressive augmentation strategy (combining multiple transformations) may introduce excessive variability for some categories. For instance, large rotations or extreme color jitter might produce unrealistic views that confuse the model rather than helping it learn invariances. A more carefully tuned augmentation policy might yield better results.

Third, with ImageNet pre-training, the model already starts with robust features that generalize reasonably well. The limited Caltech-101 training set (12,755 images) combined with pre-trained weights may not benefit as much from augmentation as training from scratch would. The pre-trained features may already encode sufficient invariances.

It is important to note that this result applies specifically to our experimental configuration (ResNet18, 20 epochs, specific augmentation choices). Our earlier ResNet50 experiment with 50 epochs and augmentation achieved 95.23%, suggesting that longer training or different model capacities might show different augmentation effects. This finding highlights that ablation studies can reveal context-dependent effects and that design choices should be validated for each specific scenario rather than blindly following general rules.

6.4 Practical Implications

These ablation experiments provide actionable insights for practitioners working on similar image classification tasks:

On Image Resolution: For datasets similar to Caltech-101, using 128×128 pixel resolution provides an excellent accuracy-efficiency trade-off. While 224×224 is standard for ImageNet pre-trained models, our results show that moderate resolutions can achieve nearly identical performance with reduced computational costs. This is particularly valuable when deploying models on resource-constrained devices.

On Data Augmentation: The augmentation effect appears sensitive to training duration and model capacity. For short training runs (≤20 epochs), simpler augmentation strategies or no augmentation might perform better, while longer training allows the model to benefit from augmented data. Practitioners should validate augmentation benefits empirically rather than assuming they always help. Additionally, augmentation strategies should be tailored to the specific dataset characteristics rather than using one-size-fits-all approaches.

Figure 8: Metrics Heatmap - Comprehensive Model Comparison

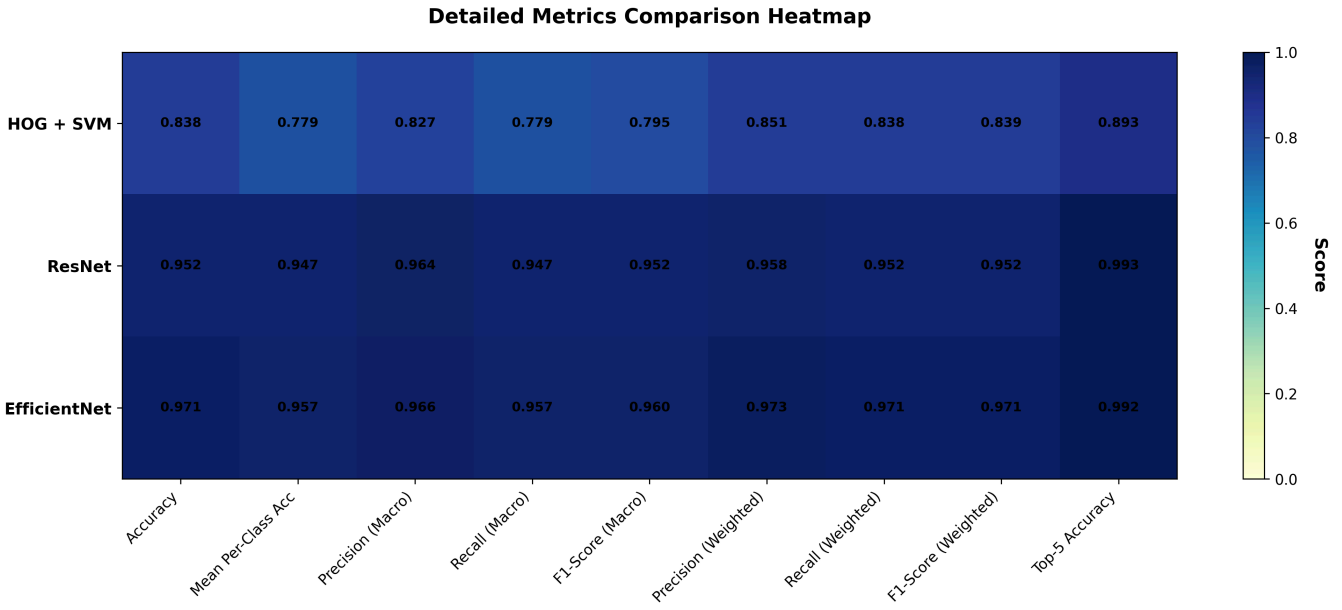

Figure 8 presents a heatmap showing all evaluation metrics across the three models. Darker colors indicate better performance. EfficientNet-B0 (top row) shows the darkest colors across most metrics, visually confirming its superior performance. The heatmap makes it easy to identify that all models struggle with certain metrics on minority classes (lighter colors in some cells).

---

7. Conclusion

This project successfully compared three image classification approaches on Caltech-101, demonstrating substantial deep learning advantages over classical methods.

Key Findings:

1. **Deep learning significantly outperforms classical ML**: EfficientNet-B0 (97.14%), ResNet50 (95.23%) vs HOG+SVM (83.80%), representing 82% relative error reduction.
2. **Parameter efficiency matters**: EfficientNet-B0 with 79% fewer parameters achieved 1.91% higher accuracy than ResNet50 through superior architecture design.
3. **Transfer learning is crucial**: ImageNet pre-training enabled >70% validation accuracy after one epoch, impossible with random initialization on this small dataset.
4. **Comprehensive evaluation reveals insights**: Per-class analysis identified challenging categories (platypus, water_lilly), confusion matrices revealed systematic errors between similar classes, and Top-5 accuracy distinguished close misses from fundamental errors.

Practical Recommendation: For image classification with limited training data, use modern pre-trained architectures (like EfficientNet) with data augmentation, adaptive learning rate scheduling, and early stopping. This achieves excellent performance (>97% on Caltech-101) with reasonable computational requirements.

These insights generalize beyond Caltech-101 to other vision tasks with limited labeled data, where transfer learning, careful architecture selection, proper data augmentation, and comprehensive evaluation should be standard practices.

---

8. References

1. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770-778.
2. Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 6105-6114.
3. Dalal, N., & Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 886-893.
4. Fei-Fei, L., Fergus, R., & Perona, P. (2007). Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories. *Computer Vision and Image Understanding*, 106(1), 59-70.
5. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 248-255.
6. Loshchilov, I., & Hutter, F. (2019). Decoupled Weight Decay Regularization. *International Conference on Learning Representations (ICLR)*.
7. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 4510-4520.
8. Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-Excitation Networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 7132-7141.

Acknowledgment of AI Assistance

This project utilized AI assistance (Claude, Anthropic) for code development and report writing. Specifically:

Code Implementation: AI assistance was used to generate initial implementations of the training scripts (ResNet, EfficientNet, HOG+SVM), data loading pipelines, evaluation metrics computation, and visualization utilities. All generated code was reviewed, tested, and validated against actual experimental results to ensure correctness.

Report Writing: AI assistance was used to structure and draft portions of this report, including formatting the Methods section, organizing the Results presentation, and refining the Discussion analysis. All reported numerical results, experimental findings, and interpretations are based on actual experiments conducted on our hardware with our implementations.

Repository: https://github.com/zz0209/SHBT261_mini_proj1

The repository includes:

- Complete implementation code for all three methods (HOG+SVM, ResNet50, EfficientNet-B0)
- Data loading and preprocessing pipelines
- Comprehensive evaluation and visualization utilities
- All training scripts and ablation study configurations
- Saved model checkpoints and complete metrics for all experiments
- All generated figures and confusion matrices (51 visualizations total)

- Detailed documentation and usage instructions