Chapter 1

Methodology

1.1 Methodology

1.1.1 Google Colab Platform

Google Colab was selected as the primary platform for developing and training deep learning models. As described by Anjum et al. [**?**], Google Colab offers significant advantages for machine learning research through its cloud-based environment with integrated GPU acceleration enabling fast model training. The platform's pre-installed libraries and integration with Google Drive provided an efficient workflow for model development, experimentation, and storage of datasets and trained models. This approach aligns with modern best practices in deep learning research where computational efficiency is crucial for iterative model development and refinement.

Despite its advantages, Google Colab presented a few challenges. The platform frequently disconnected during training sessions, interrupting the model training process before completing all epochs. These disconnections likely stemmed from limited RAM allocation, runtime timeouts, or resource constraints of the shared free GPU environment. As noted by Carneiro et al. [**?**], while Colab provides robust GPU resources that can match dedicated servers for certain tasks, these free resources "are far from enough to solve demanding real-world problems and are not scalable."

To mitigate these issues, two strategies were implemented. First, the relatively small size of our dataset helped minimize resource demands. Second, checkpoint saving was implemented throughout the training process, allowing training to resume from the last saved state if disconnections were encountered. This approach ensured that progress wasn't lost when disconnections occurred, though it introduced some workflow inefficiencies.

1.1.2 Python and PyTorch Framework

The implementation was carried out using Python as the primary programming language, chosen for its extensive library support and widespread adoption in the machine learning community. Python's simple syntax and powerful libraries make it particularly suitable for rapid prototyping and experimentation in deep learning research [**?**].

For the deep learning framework, PyTorch was selected over alternatives like TensorFlow or Keras due to its dynamic computational graph which allows for more flexible model development and easier debugging. PyTorch's intuitive design facilitates a more natural expression of deep learning algorithms while still providing the performance ben-

efits of GPU acceleration. The framework's robust ecosystem for computer vision tasks, including pre-trained models and transformation pipelines, was particularly valuable for this fine-grained classification task.

Advantages of PyTorch in Our Implementation

PyTorch offered several key advantages that were particularly beneficial for our transfer learning approach with pre-trained models:

- **Dynamic Computational Graph:** PyTorch's define-by-run approach allowed for more intuitive debugging and model modification during development. This was especially valuable when adapting pre-trained architectures like VGG16 for our specific classification task.
- **Flexible Model Customization:** The implementation benefited from PyTorch's object-oriented approach, which made it straightforward to modify pre-trained models, e.g., replacing classification layers while preserving feature extraction capabilities.
- **Efficient Data Loading and Augmentation:** PyTorch's DataLoader and transformation pipelines facilitated efficient batch processing and on-the-fly data augmentation, which was crucial for maximizing the utility of our limited dataset.
- **Gradient Visualization Tools:** PyTorch's native support for gradient computation and hooks made implementing Grad-CAM and other visualization techniques more straightforward, enabling better model interpretability.

Similar to approaches described by Raffel et al. [**?**], our implementation prioritized efficiency and optimization to work within the constraints of limited computational resources, allowing us to achieve high-quality results despite the limitations of the free cloud environment.

1.2 Dataset Preparation and Refinement

The dataset preparation followed a three-stage iterative refinement process, each addressing specific challenges identified during model development. This approach aligns with established methodologies in fine-grained bird classification research, where dataset quality has been shown to significantly impact model performance [**?**].

1.2.1 Stage 1: Initial Dataset Collection

The initial dataset was collected from public repositories including eBird and iNaturalist, comprising 451 images of Glaucous-winged Gulls and 486 images of Slaty-backed Gulls. This dataset included gulls of various ages (juveniles and adults) in different postures (sitting, standing, and flying). Initial model testing on this dataset yielded poor performance (below 50% accuracy), highlighting the need for dataset refinement. Similar challenges with diverse postures and class imbalance have been documented by Kahl et al. in their work on BirdNET systems [**?**].

1.2.2 Stage 2: Refined Dataset - Focus on Adult In-flight Images

Consultation with Professor Gibbins, an ornithological expert, revealed that adult wingtip patterns are the most reliable distinguishing features between these species, and these patterns are most visible in flight. This expert-guided refinement approach parallels methods described by Wang et al. in their work on avian dataset construction, where domain expertise significantly improved classification accuracy for visually similar species [**?**]. Consequently, the dataset was refined to focus exclusively on adult in-flight images, resulting in a curated collection of 124 Glaucous-winged Gull images and 127 Slaty-backed Gull images. This targeted approach significantly improved model performance, with accuracy increasing to approximately 70%.

1.2.3 Stage 3: High-Quality Dataset

To further enhance classification performance, 640 high-resolution images of in-flight Slaty-backed Gulls were obtained from Professor Gibbins. The Glaucous-winged Gull dataset was also carefully curated with expert guidance, reducing it to 135 high-quality images that clearly displayed critical wingtip features. Images showing birds in moulting stages, juveniles, or unclear wingtip patterns were systematically removed. This quality-focused approach aligns with findings from Zhou et al., who demonstrated that expert-curated datasets can achieve comparable or superior results with significantly smaller data volumes compared to larger uncurated collections [**?**].

For comparative analysis, an unrefined dataset containing 632 adult in-flight Glaucous-winged Gulls and 640 high-quality Slaty-backed Gull images was also tested. This multi-dataset evaluation approach follows best practices established in the BirdSet benchmark for avian classification studies [**?**].

1.3 Transfer Learning Methodology

1.3.1 Theoretical Framework and Rationale

Transfer learning is a powerful machine learning technique that involves reusing a pre-trained model developed for a specific task as a starting point for a new task. This approach significantly enhances learning efficiency by leveraging knowledge gained from solving previous problems, enabling a positive transfer learning effect and reducing the training time required. For fine-grained classification tasks like distinguishing between visually similar gull species, transfer learning is particularly valuable as it allows the model to build upon a foundation of general visual features already learned from diverse datasets.

As highlighted by Kahl et al. (2021), transfer learning addresses two critical challenges in specialized biological classification: data scarcity and feature abstraction [**?**]. First, data scarcity is a common issue in specialized domains like ornithological image classification, where large-scale annotated datasets are rare. Transfer learning mitigates this by leveraging models pre-trained on massive datasets like ImageNet. Second, these pre-trained models have learned to extract hierarchical features that capture important visual patterns, which can significantly enhance the accuracy of fine-grained classification tasks.

In our implementation, transfer learning was employed to leverage the robust feature extraction capabilities of pre-trained models on ImageNet. This approach aligns with best practices in fine-grained classification tasks, where lower-level features learned from diverse datasets can be effectively repurposed for specialized domains. The pre-training on ImageNet's 1.2 million images across 1,000 classes provides the model with a strong foundation for recognizing a wide range of visual patterns, which can then be fine-tuned for the specific task of distinguishing between Glaucous-winged and Slaty-backed Gulls.

Several pre-trained architectures were evaluated for this task, with VGG-16 demonstrating superior performance in our specific classification context. The effectiveness of transfer learning was evident in the rapid convergence and high accuracy achieved even with our relatively limited dataset of gull images, demonstrating the potential of this approach for specialized biological classification tasks.

1.4 Model Architectures and Implementation

article

1.4.1 VGG-16 Architecture

Theoretical Background

VGG-16 is a popular Convolutional Neural Network (CNN) architecture widely used in computer vision applications. Originally developed by Simonyan and Zisserman (2014), VGG-16 consists of 16 layers, including 13 convolutional layers arranged in blocks of increasing depth, followed by 3 fully connected layers, with a total of approximately 138 million parameters. The architecture follows a systematic approach of stacking convolutional layers with small 3×3 filters followed by max-pooling layers, creating a deep network capable of learning complex hierarchical features.

One of the main advantages of using VGG-16 as a pre-trained model for transfer learning is its ability to capture a wide range of features and patterns in images. This capability stems from the deep architecture of the VGG-16 model, which allows it to extract more complex features from images compared to shallower models. VGG-16 has been pre-trained on the ImageNet dataset, which contains over 1.2 million images across 1,000 classes, enabling it to recognize a wide range of features and patterns applicable to various computer vision tasks.

The architecture's elegant simplicity, despite its depth, makes it particularly effective for fine-grained visual classification tasks like ours. Its performance on various computer vision benchmarks, including object detection, image segmentation, and image classification tasks, makes it a versatile choice for transfer learning in numerous applications. For our specific task of gull species classification, the hierarchical feature representation capabilities of VGG-16 proved particularly effective at capturing the subtle differences in wing patterns and morphological features that distinguish between the target species.

Model Adaptation for Gull Species Classification

The pre-trained VGG16 model was adapted for our binary classification task through targeted modifications to the final classification layer. The implementation approach follows best practices for fine-grained bird classification established in recent research

on transfer learning for avian species identification [**?**]. Specifically, the original 1000-class classifier was replaced with a custom binary classification head while preserving the feature extraction capabilities of the convolutional base. The model modification strategy followed this approach:

1. Loading the pre-trained VGG16 with ImageNet weights
2. Extracting the number of features from the original classifier (4096)
3. Replacing the final layer with a sequential block containing:
   (a) A dropout layer with rate 0.4 for regularization
   (b) A linear layer mapping from 4096 features to 2 output classes

This implementation maintained the complex feature hierarchy learned by VGG16 while adapting the final classification stage for our specific binary task. The relatively high dropout rate (0.4) was strategically implemented to address potential overfitting challenges common in fine-grained classification tasks with limited training data, particularly important given the visual similarities between the target gull species.

1.4.2 Data Preprocessing and Augmentation Strategy

Image Preprocessing

Images were preprocessed using a consistent pipeline to ensure compatibility with the VGG16 architecture. All images were resized to 224×224 pixels to match VGG16's expected input dimensions. Following resize operations, pixel values were normalized using ImageNet mean values [0.485, 0.456, 0.406] and standard deviations [0.229, 0.224, 0.225]. This normalization strategy ensures input distributions match those seen during pre-training, facilitating effective transfer learning.

Training Augmentation

A comprehensive data augmentation strategy was implemented to enhance model generalization capabilities and mitigate overfitting. The augmentation pipeline for training incorporated multiple techniques designed to preserve critical taxonomic features while introducing beneficial variability:

Geometric Transformations:

- Random horizontal flips (probability 0.5)
- Random vertical flips (probability 0.3)
- Small rotations (±10 degrees)
- Minor affine transformations (translations up to 5%)
- Random resized crops (scale 0.95-1.0 of original size)

Color and Appearance Transformations:

- Brightness, contrast, and saturation adjustments (±10%)
- Sharpness enhancement (factor 1.2, probability 0.3)

This augmentation strategy was carefully calibrated to maintain the integrity of critical morphological features (particularly wingtip patterns) while simulating natural variations in viewing conditions. The relatively conservative parameter choices reflect the importance of preserving diagnostic features in fine-grained classification tasks.

Validation and Testing Preprocessing

For validation and testing phases, a simplified transformation pipeline was employed, consisting only of resizing to 224×224 pixels, tensor conversion, and normalization. This approach ensures consistent evaluation conditions while maintaining the statistical properties expected by the pre-trained model.

1.4.3 Training Methodology

Dataset Organization and Splitting

The dataset was organized using the ImageFolder structure, with a 95:5 split between training and validation sets. This configuration provided substantial training data while maintaining a sufficient validation set for hyperparameter tuning and model selection. A separate test set was maintained for final performance evaluation.

Loss Function and Optimization

Cross-entropy loss was selected as the objective function for this binary classification task, providing appropriate gradients for optimization. This loss function effectively quantifies the discrepancy between predicted class probabilities and ground truth labels.

The AdamW optimizer was employed with carefully tuned hyperparameters to facilitate effective model training:

- Learning rate: 0.0001 (relatively low to enable stable fine-tuning)
- Weight decay: 0.001 (for L2 regularization to prevent overfitting)

This optimization configuration balances the need for fine-grained weight adjustments with regularization to maintain generalization capacity.

Learning Rate Scheduling and Training Stabilization

An adaptive learning rate schedule was implemented using ReduceLROnPlateau with a patience factor of 3 epochs and a reduction factor of 0.1. This approach automatically reduces the learning rate when validation performance plateaus, enabling finer weight adjustments as the model approaches optimal parameters.

To enhance training stability, gradient clipping was applied with a maximum norm of 2.0. This technique prevents exploding gradients by constraining the magnitude of parameter updates, which is particularly valuable when fine-tuning deep architectures like VGG16.

Batch Processing and Training Duration

The model was trained with a batch size of 16, striking a balance between computational efficiency and effective mini-batch gradient estimation. Training was configured to run for a maximum of 30 epochs with early stopping based on validation performance, ensuring optimal model selection while avoiding overfitting.

Model checkpoints were saved after each epoch, preserving the best-performing model configurations for subsequent evaluation and deployment.

1.4.4 Evaluation Metrics and Performance

The model's performance was assessed using multiple complementary metrics to ensure robust evaluation:

- Accuracy: Percentage of correctly classified images
- Precision: Proportion of true positive predictions among all positive predictions
- Recall: Proportion of true positives identified among all actual positives
- F1-Score: Harmonic mean of precision and recall

The final VGG16 model achieved exceptional performance with 98.80% validation accuracy and 100% test accuracy, demonstrating its effectiveness in distinguishing between the two gull species based on the refined dataset. This performance exceeds typical benchmarks for fine-grained bird classification tasks, highlighting the effectiveness of the implemented architecture, data preparation strategy, and training methodology.

A confusion matrix analysis confirmed the model's strong classification performance across both classes, with minimal misclassifications. This indicates the model successfully learned the discriminative morphological features necessary for distinguishing between Glaucous-winged and Slaty-backed Gulls.

article

1.5 Vision Transformer (ViT) Architecture

1.5.1 Theoretical Framework

Vision Transformers (ViT) represent a paradigm shift in computer vision, applying the self-attention mechanism from transformer models—originally developed for natural language processing—to image classification tasks. First introduced by Dosovitskiy et al. (2020), ViT processes images by dividing them into a sequence of fixed-size patches, which are then linearly embedded and processed through transformer encoder blocks[1].

Unlike CNNs that build hierarchical representations through local convolutional operations, ViT applies self-attention mechanisms to capture global relationships between image patches. This allows the model to attend to long-range dependencies within the image, potentially capturing more holistic patterns. As demonstrated by research from Liu et al. (2022), these attention mechanisms enable transformers to excel at detecting subtle features in biomedical images by focusing on the most discriminative regions[2].

The standard ViT architecture consists of:

- Patch embedding layer that converts image patches to token embeddings

- Position embedding to provide spatial information
- Multiple transformer encoder blocks with multi-head self-attention
- Layer normalization and MLP blocks within each transformer layer
- A classification head for prediction

This architecture's capacity to model global relationships makes it particularly promising for fine-grained classification tasks where relationships between distant parts of an image (e.g., wing patterns in relation to head features) may be important for accurate classification.

1.5.2 Standard Vision Transformer Implementation

My standard ViT implementation utilized the pre-trained 'vit_base_patch16_224' model from the TIMM library, which features a patch size of 16×16 pixels and was trained on the ImageNet dataset. The model adaptation process included:

- Loading the pre-trained ViT model with frozen weights
- Extracting the embedding dimension from the original model (768 features)
- Replacing the classification head with a binary classifier for our gull species task
- Maintaining the self-attention mechanisms and transformer blocks

This approach leverages the powerful feature extraction capabilities of ViT while customizing the final classification stage for our specific task. The implementation follows best practices established by Wightman et al. (2021) for adapting vision transformers to specialized classification tasks[3].

1.5.3 Enhanced Vision Transformer with Custom Attention

To further improve the model's ability to focus on taxonomically relevant features, we developed an Enhanced Vision Transformer (EnhancedViT) that incorporates a custom attention mechanism specifically designed for fine-grained classification tasks.

The key innovation in this implementation is an attention-based pooling layer that computes importance scores for each patch token, enabling the model to focus on the most discriminative regions of the input image. This approach draws inspiration from the work of Guan et al. (2022), who demonstrated that specialized attention mechanisms in vision transformers can significantly improve fine-grained classification by emphasizing taxonomically relevant features[4].

The enhanced ViT architecture extends the standard implementation with:

- A custom attention layer that computes importance scores for each token
- An attention-weighted aggregation step that prioritizes informative regions
- A multi-layer perceptron classifier with dropout regularization
- Layer normalization for improved training stability

The attention mechanism was implemented as:

Attention-based Token Pooling

1: Compute attention scores for each token using a learned projection
2: Normalize scores using softmax to create attention weights
3: Perform weighted aggregation of tokens based on attention weights
4: Process the attention-weighted representation through the classifier

This approach allows the model to dynamically focus on the most relevant parts of the image for classification, such as distinctive wingtip patterns or other morphological features that differentiate between gull species.

1.5.4 Data Processing and Augmentation

Both ViT implementations used standardized preprocessing and augmentation pipelines:

- Resize operations to 224×224 pixels (the standard input size for ViT models)
- Normalization with mean [0.5, 0.5, 0.5] and standard deviation [0.5, 0.5, 0.5]
- Augmentation techniques including:
  - Random horizontal flipping
  - Random rotation (±15 degrees)
  - Color jittering (brightness, contrast, saturation)

The input normalization values specifically used [0.5, 0.5, 0.5] rather than ImageNet statistics, following recommendations from Touvron et al. (2021) for transfer learning with vision transformers[5].

1.5.5 Training Methodology

The training approach for both ViT variants included:

- AdamW optimizer with learning rate 0.0001 and weight decay 1e-4
- Learning rate scheduling with ReduceLROnPlateau (patience=3, factor=0.1)
- Batch size of 16 to balance computational efficiency and training stability
- Training for 20 epochs with early stopping based on validation performance

For the EnhancedViT, we employed additional training refinements:

- Layer-wise learning rate decay to fine-tune different components at appropriate rates
- Dropout regularization (p=0.3) in the custom classification head
- Checkpoint saving to preserve the best-performing model configuration

Both models were trained on the refined high-quality dataset (Stage 3), with an 80:20 split between training and validation sets to ensure robust performance evaluation during development.

article

1.6 Inception v3 Architecture

1.6.1 Theoretical Background

Inception v3, developed by Szegedy et al. (2016), represents a sophisticated CNN architecture designed to efficiently capture multi-scale features through parallel convolution paths with different kernel sizes. The key innovation in Inception architectures is the use of "Inception modules" that process the same input tensor through multiple convolutional paths with different receptive fields, and then concatenate the results. This enables the network to capture both fine-grained local patterns and broader contextual information simultaneously.

Inception v3 builds upon earlier versions with several important architectural improvements:

- Factorized convolutions to reduce computational cost
- Spatial factorization into asymmetric convolutions (e.g., 1×n followed by n×1)
- Auxiliary classifiers that inject additional gradient signals during training
- Batch normalization for improved training stability
- Label smoothing regularization to prevent overconfidence

These design elements collectively enable Inception v3 to achieve high accuracy while maintaining computational efficiency. As demonstrated by Shu et al. (2023), Inception architectures are particularly effective for tasks requiring multi-scale feature extraction, such as discriminating between visually similar biological specimens.

1.6.2 Model Adaptation for Gull Classification

Our implementation adapted the pre-trained Inception v3 model for gull species classification using the following approach:

1. Loading the pre-trained Inception v3 model with ImageNet weights
2. Extracting the feature dimension from the original classifier (2048)
3. Replacing the final classifier with a custom sequence:
   (a) Dropout layer (p=0.5) for regularization
   (b) Linear layer mapping 2048 features to 2 output classes

A distinctive aspect of our Inception v3 implementation was the utilization of auxiliary outputs during training. Inception v3's auxiliary classifier, which branches off from an intermediate layer, provides an additional gradient path during backpropagation. This approach helps combat the vanishing gradient problem and provides regularization, as noted by Szegedy et al. (2016) in their original paper.

The loss function was modified to incorporate both the main output and the auxiliary output during training:

$$\text{loss} = \text{main\_loss} + 0.3 \times \text{auxiliary\_loss} \tag{1.1}$$

where the auxiliary loss weight (0.3) was selected based on empirical optimization and aligns with recommendations in the literature for fine-tuning Inception architectures.

1.6.3 Advanced Training Techniques

The Inception v3 implementation incorporated several advanced training techniques to optimize performance:

- Mixed-precision training using PyTorch's Automatic Mixed Precision (AMP) to accelerate computation while maintaining numerical stability
- Gradient clipping with a maximum norm of 2.0 to prevent explosive gradient updates
- Precisely tuned learning rate and weight decay parameters identified through hyperparameter optimization
- Layer-wise learning rate adjustment to fine-tune different parts of the network at appropriate rates

These techniques collectively enhanced training efficiency and model performance. The implementation of mixed-precision training was particularly valuable given the resource constraints of the Google Colab environment, as it reduced memory usage and accelerated computation without compromising model accuracy.

1.6.4 Data Processing Pipeline

The data processing pipeline for Inception v3 was adapted to the model's specific requirements:

- Resize operations to 299×299 pixels (the standard input size for Inception v3)
- Standard data augmentation techniques for training:
  - Random horizontal flipping
  - Random rotation (±15 degrees)
  - Color jittering
- Simple resizing and normalization for validation and testing

The larger input resolution (299×299 vs 224×224 used by VGG16 and ViT) provides the Inception architecture with more detailed information, potentially beneficial for capturing the subtle wing pattern differences between gull species.

[11pt]article

1.7 ResNet50 Architecture

1.7.1 Theoretical Background

Residual Networks (ResNet) represent a significant innovation in deep neural network architecture, introduced by He et al. (2016) to address the degradation problem that occurs when training very deep networks. The key innovation in ResNet is the introduction of skip connections or "shortcut connections" that bypass one or more layers, allowing gradients to flow more easily through the network during backpropagation[**?**].

This design enables the training of much deeper networks than was previously feasible, with ResNet-50 containing 50 layers organized in residual blocks[**?**].

ResNet-50 architecture consists of five stages, each containing multiple residual blocks. Each residual block contains a "shortcut" that skips over the main path and rejoins it later, allowing the network to learn residual functions with reference to the layer inputs rather than learning unreferenced functions[**?**][**?**]. This approach enables ResNet to achieve high performance on image classification tasks while mitigating the vanishing gradient problem common in very deep networks.

The architecture's ability to effectively extract hierarchical features through its deep structure makes it particularly well-suited for fine-grained classification tasks where subtle differences must be detected. As noted by Ghani et al. (2023), ResNet architectures have demonstrated strong performance in avian classification tasks due to their capacity to learn discriminative features at multiple scales and levels of abstraction.

1.7.2 Model Adaptation for Gull Species Classification

For our gull classification task, we adapted the pre-trained ResNet-50 model using a focused transfer learning approach. The model was initialized with weights pre-trained on the ImageNet dataset, providing a strong foundation of general visual features. The adaptation process involved:

1. Loading the pre-trained ResNet-50 model with ImageNet weights
2. Preserving the convolutional backbone to maintain feature extraction capabilities
3. Replacing the final fully connected layer (classifier) with a custom sequence:
   (a) Dropout layer with probability 0.5 to reduce overfitting
   (b) Linear layer mapping from 2048 features to 2 output classes

This adaptation strategy preserved ResNet-50's powerful feature extraction capabilities while customizing the classification head for our binary task. The relatively high dropout rate (0.5) was implemented to address potential overfitting, which is particularly important given the visual similarities between the target species and the limited size of our specialized dataset[**?**].

1.7.3 Image Preprocessing and Enhancement

A distinctive aspect of our ResNet-50 implementation was the incorporation of image sharpening as a preprocessing step. This approach was motivated by research from Zhou et al. (2021) showing that enhancing edge definition can improve the detection of subtle morphological features in avian classification tasks. The image enhancement process applied a 3×3 sharpening kernel through a custom preprocessing function:

$$\text{Sharpening Kernel: } \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

This technique enhanced the visibility of critical diagnostic features like wingtip patterns while preserving the overall image content. To ensure consistency, image sharpening was applied across both training and evaluation pipelines[**?**].

1.7.4 Data Augmentation Strategy

The data augmentation pipeline for ResNet-50 was structured to enhance model robustness while preserving class-discriminative features:

- Resize operations (300×300 pixels) followed by sharpening
- Random horizontal flipping to simulate viewpoint variation
- Random rotation (±15 degrees) to account for flight angle variability
- Color jittering (brightness, contrast, saturation adjusted by ±20%)
- Random cropping with padding to vary focus regions

For validation and testing, a more conservative approach was employed with resizing, center cropping (256×256 pixels), and the same sharpening preprocessing to maintain feature clarity without introducing variability[**?**].

1.7.5 Training Approach and Optimization

The ResNet-50 model was trained using the following methodological approach:

- Adam optimizer with learning rate 0.001 and weight decay 1e-4 for regularization
- Adaptive learning rate scheduling using ReduceLROnPlateau with patience=3
- Early stopping with patience=5 to prevent overfitting
- Batch size of 16 for efficient GPU utilization

The implementation of early stopping was particularly valuable for the ResNet model, as it helped prevent overfitting to the training data while ensuring the model retained its generalization capabilities. As demonstrated by Huang et al. (2022), early stopping acts as an effective regularization technique for deep networks when working with specialized datasets of limited size[**?**].

article

1.8 Custom CNN with Squeeze-and-Excitation Blocks

1.8.1 Architectural Innovation

To complement the transfer learning approach with pre-trained models, we developed a custom CNN architecture specifically designed for our gull classification task. The architecture incorporates Squeeze-and-Excitation (SE) blocks, an attention mechanism introduced by Hu et al. (2018) that adaptively recalibrates channel-wise feature responses by explicitly modeling interdependencies between channels.

The SE mechanism enhances standard convolutional operations by adding two operations:

- A "squeeze" operation that aggregates feature maps across spatial dimensions to produce a channel descriptor

- An "excitation" operation that produces per-channel modulation weights

This channel-wise attention mechanism allows the network to emphasize informative features and suppress less useful ones, improving the representational power of the network. As demonstrated by Hu et al. (2018), the SE mechanism yields significant performance improvements while adding minimal computational overhead.

Our custom CNN implementation follows this architectural pattern:

1.8.2 Addressing Class Imbalance

An important methodological consideration in our custom CNN implementation was addressing potential class imbalance in the dataset. To ensure balanced learning despite the uneven distribution of examples between classes, we implemented a weighted sampling approach based on class frequencies.

The implementation calculated inverse class weights to prioritize examples from underrepresented classes:

Weighted Sampling for Class Balance

1. Count examples per class in the training dataset
2. Calculate inverse class frequencies: weights = 1 / class_counts
3. Assign a sampling weight to each training example based on its class
4. Create a WeightedRandomSampler using these weights
5. Use the sampler in the DataLoader to achieve balanced mini-batches

This approach ensures that the model receives a balanced distribution of examples during training, preventing bias toward the majority class. The effectiveness of this technique for handling class imbalance has been demonstrated in fine-grained classification research by Buda et al. (2018), who showed that sampling strategies can significantly improve model performance on imbalanced datasets.

1.8.3 Training Methodology

The custom CNN was trained using the following approach:

- Adam optimizer with learning rate 0.001 and weight decay 0.0005
- Cosine Annealing learning rate scheduler for cyclical learning rate adjustment
- Cross-entropy loss function
- Batch size of 32 (larger than the pre-trained models due to lower memory requirements)
- Training for 20 epochs with checkpoint saving for best-performing models

The use of Cosine Annealing for learning rate scheduling represents a different approach compared to the ReduceLROnPlateau used with the pre-trained models. This scheduler cyclically varies the learning rate between a maximum and minimum value following a cosine function, helping the model escape local minima and potentially converge to better solutions. This approach aligns with research by Loshchilov and Hutter (2017) demonstrating the effectiveness of cyclical learning rates for CNN training.

Chapter 2

Summary and Reflections

Including a discussion of results in a wider context (considering other work).

2.1 Project management

Covering the tasks as a part of your work plan and progress as well as how time and resources are managed.

2.2 Contributions and reflections

Providing the details of your achievements and contributions including innovation, creativity and novelty (if there is any) as well as a personal reflection on the plan and your experience of the project (a critical appraisal of how the project went).

Chapter 3

User Manuals

Chapter 4

User Evaluation Questionnaire