Enhancing Fine-Grained Image Classification

Yulin Zhou Kunsheng Liu Xinyu Chen

Abstract

Fine-Grained Image Classification (FGVC) is a challenging area in computer vision focused on categorizing images into highly specific and detailed categories, such as different species of birds or breeds of dogs. The primary challenge in FGVC lies in the small inter-class variance and large intra-class variance, making it difficult to distinguish between classes. For this task, we have tried the following methods: 1) Transfer learning, by fine-tuning several different ResNet models. 2) Generative Adversarial Networks (GANs), to generate additional data. 3) Vision Transformers (ViTs), using self-attention models to process images. 4) Visual Language Models (VLMs), attempting zero-shot transfer with a single VLM model. 5) Class Activation Mapping (CAM), to show how the model's weights focus. 6) Robustness Evaluation, using FGSM for attacks and employed adversarial training to enhance the model's robustness. 7) Using multiple datasets for experimental comparison to study the model's generalization ability.

1. Introduction

FINE-GRAINED visual classification (FGVC) has been an active area of research for quite a long time. The goal of FGVC is to accurately categorize images into very specific and detailed categories. In the real-world, FGVC enjoys a wide-range of applications in both industry and research societies, such as biodiversity monitoring, medical image analysis, product recognition and classification, and many more.

FGVC is a challenging task in computer vision that involves categorizing images into very specific and detailed categories, such as different species of birds, dogs. The primary challenge in fine-grained image classification lies in the small inter-class variance and large intra-class variance. Different categories often exhibit high similarity, while there can be significant variations among individuals within the same category. For example, different bird species might only differ slightly in feather color or shape, and individuals within the same species can show considerable differences due to factors such as pose, background, and lighting. Due to the fact that data annotation for fine-grained recognition tasks typically requires expert knowledge, is time-consuming, and incurs high costs, researchers are dedicated to developing semi-supervised learning, weakly supervised learning, and self-supervised learning methods. Additionally, techniques such as transfer learning and data augmentation are widely applied to fully utilize the limited labeled data and enhance the generalization ability of models.

Our study primarily utilized the CUB200-2011 dataset, a fine-grained classification dataset providing image-level annotations and keypoint locations. For foundational training, we evaluated the accuracy and convergence speed of ResNet models of different depths and batch sizes on validation and test sets. We analyzed the impact of model depth and batch size on post-training performance.

In transfer learning, we employed the ResNet101 32x8d model and ViT model, assessing performance by training different layers and analyzing the effects of layer-wise training on model performance.

Regarding training techniques, using ResNet50 as a baseline, we explored the effects of various techniques such as geometric transformations, CutOut, Mixup, Label Smoothing, and Dropout on model performance. We investigated their synergistic effects to maximize effectiveness.

Using GANs, we built generators and discriminators based on ResNet, achieving model stability and generating synthetic images from the original dataset. We analyzed differences between real and generated images and identified contributing factors.

For VLM, we utilized several CLIP models of different sizes, conducted zero-shot learning on the dataset, and analyzed learning outcomes.

Implementing CAM and Grad-CAM, we compared and analyzed the generated heatmaps to understand their differences in highlighting image features.

Assessing robustness, we tested models against FGSM attacks to evaluate their susceptibility and conducted adversarial training to enhance resistance to FGSM attacks. We then re-evaluated the models' resistance post-adversarial training.

To evaluate model generalization, we transferred models to another dataset, performed fine-tuning, and compared performance across various datasets.

The above is our main task.

2. Method

2.1. Residual Neural Network

At the beginning of our experiment, we will use the Residual Neural Network (ResNet) [1], a deep learning model widely used for image classification tasks, as a baseline for subsequent experiments. Its core innovation lies in allowing each layer of the network to learn residual functions relative to the input, rather than directly learning the desired output. This network design includes "skip connections" that bypass certain layers and merge with the layer outputs, facilitating identity mapping. This design makes it easier to train deep learning models with dozens or even hundreds of layers, alleviating the vanishing gradient problem and maintaining or improving accuracy as model depth increases. The overall network architecture is as follows:

- **Initial Convolutional Layer**: Performs spatial downsampling on the input image, reducing the spatial dimensions for subsequent layers, thereby lowering computational complexity and capturing image features. Using a large convolution kernel can effectively capture global information of the image.
- **Residual Block Groups**: Contains multiple residual blocks, with each block extracting higher-level features from the previous group's features. Through residual connections, each group can learn complex non-linear mappings between input and output while avoiding gradient vanishing and exploding.
- **Global Average Pooling**: The final convolutional layer of the network reduces dimensionality, lowers the number of model parameters and computation, and effectively prevents overfitting, leading to better generalization.
- **Fully Connected Layer**: Outputs a fixed-size feature vector for classification or regression tasks.

2.2. Various Training Tricks

2.2.1 Data Augmentation

Data augmentation is a technique used to increase the diversity of training data without actually collecting new data. By applying various transformations to the existing dataset, it helps improve the robustness and generalization capability of machine learning models, particularly deep learning models. The main goal of data augmentation is to create variations in the training data to prevent the model from overfitting and to enhance its ability to generalize to unseen data.

- **Geometric transformations**: Geometric transformations involve modifying the spatial characteristics of images while preserving their semantic content. These transformations include HorizontalFlip, Rotation, Cropping, etc. These transformations help the model generalize better.
- **Cutout**: CutOut is a data augmentation technique where random rectangular regions of an image are masked out, i.e., set to zero. This forces the model to focus on the less occluded parts of the image and helps improve its robustness.
- **Mixup**: Mixup is a data augmentation technique where two images and their corresponding labels are combined to create a new training example. Specifically, the images are linearly interpolated, and their labels are mixed proportionally. The only parameter in Mixup is alpha, which controls the strength of interpolation between two images.

2.2.2 Regularization

Regularization aims to prevent model overfitting. By adding constraints to the model parameters, regularization methods can improve the model's generalization ability.

- **Label Smoothing**: Label Smoothing is a regularization technique used to prevent the model from becoming overly confident on the training data. Label Smoothing achieves this by slightly "smoothing" the 1s in the label vector to other categories, thereby reducing the model's overconfidence.
- **Dropout**: Dropout is a regularization technique used during the training of neural networks, where a random subset of neurons are "dropped out" or ignored during each iteration. This technique aims to prevent the neural network from relying too much on specific neurons, thereby improving its generalization ability. This stochastic process forces the network to learn more robust features, as it cannot rely on the presence of particular neurons for making predictions.

2.3. Transfer Learning

Transfer learning focuses on leveraging pre-existing solution models and applying them to different but related problems. A typical scenario is when the data distribution of the source domain and target domain differ but exhibit some correlation or similarity. In such cases, transfer learning methods can enhance learning performance on the target domain.

Transfer learning is feasible due to the layered architecture of deep learning models, where different layers learn different features and are connected to a final layer, often a fully connected layer, to obtain the final output. This architecture allows us to utilize pre-trained networks, remove

their final layer, and use them as fixed feature extractors for other tasks. The most common method is Fine-tune, where we freeze part of the pre-trained model's layers (usually the majority of convolutional layers near the input) and train the remaining layers (usually the part of convolutional layers near the output and fully connected layers). In this paper, Fine-tune is mainly used to train pre-trained models. We perform fine-tuning training on ResNet and ViT pre-trained models and compare the results with training a network from scratch. We achieve good results in terms of training efficiency and final accuracy.

2.4. Generative Adversarial Network

GANs [2] are models that generate new data by learning the latent distribution of existing data. The core idea of GANs involves training two neural networks — the Generator and the Discriminator — in an adversarial manner. The Generator takes a random noise vector and attempts to generate synthetic data similar to the real data distribution to deceive the Discriminator.

Conditional GANs (cGANs) [3] are an extension of GANs that incorporate additional conditional information into both the Generator and the Discriminator, making the generated results more controllable and allowing for data generation based on specific conditions. Unlike traditional cGANs that simply concatenate the conditional information to the Discriminator's input, cGANs with Projection Discriminator [4] use a more effective method to integrate input data and conditional information to enhance the Discriminator's performance.

cGANs with Projection Discriminator integrate conditional information into the Discriminator's decision-making process through projection. Specifically, the Discriminator computes the inner product of the sample's feature representation and the conditional information, adding this result to the Discriminator's decision-making process to improve its discriminative ability.

We attempt to use cGANs with Projection Discriminator to generate labeled data under limited data conditions and incorporate the generated images into the input training data, thereby achieving data augmentation.

2.5. Vision Transformer

The Vision Transformer (ViT) model addresses fine-grained image classification by dividing an image into fixed-size patches and treating these patches as a sequence of inputs. By leveraging the self-attention mechanism of the Transformer model, initially developed for natural language processing (NLP), ViT captures intricate image features. The process involves flattening the patches and mapping them to a high-dimensional feature space. Positional encoding is added to retain spatial information of the image patches, enabling the self-attention model to handle image data effectively. This method overcomes the limitations of traditional convolutional neural networks (CNNs) and recurrent neural networks (RNNs) in capturing long-range dependencies, allowing ViT to process both local and global information effectively, making it well-suited for fine-grained image classification tasks.

When the training dataset is not sufficiently large, the performance of ViT tends to be inferior to that of ResNets. This is due to the lack of inductive bias in Transformers compared to Convolutional Neural Networks (CNNs). In contrast, the self-attention mechanism in ViT requires large amounts of training data to achieve acceptable visual representations. Starting from scratch with small datasets yields suboptimal results. Moreover, due to memory constraints, it is often impractical to train the entire ViT model's parameters from scratch. Therefore, in experiments, we utilized pre-trained ViT models and performed fine-tuning.

The specific fine-tuning method involves freezing most parameters and only training the last few self-attention layers of the encoder and the final classification head [5] . This approach minimizes the impact on accuracy while reducing GPU memory consumption.

When the resolution of the input image changes, the length of the input sequence also changes. Although ViT can handle sequences of arbitrary lengths, the pre-trained positional encodings may no longer be applicable. For instance, if the original patch configuration was 3×3 (yielding nine patches) with specific positional encodings, increasing the number of patches alters the positional information. One approach to address this is to use interpolation to resize the positional encoding table. However, significant changes in sequence length can degrade model performance due to interpolation inaccuracies, presenting a limitation for ViT during fine-tuning.

2.6. VLM

A Vision-Language Model (VLM) aims to simultaneously process visual and textual data, addressing a range of multimodal tasks involving both images and text. It tackles cross-modal understanding and generation tasks that traditional models struggle with, demonstrating innovation and superior performance across various applications. Current mainstream VLM models include CLIP, ALIGN, among others.

CLIP (Contrastive Language-Image Pre-training) [6] is an open-sourced, large-scale text-image pre-training model developed by OpenAI based on contrastive learning principles. Its core idea is to use natural language supervision to guide the training of visual models. Specifically, the process involves extracting and encoding text features using modules such as Transformers, and extracting and encoding image features using similar modules. Positive and negative samples are constructed using these encoded text and

image features for contrastive learning. After training, category labels are used to construct prompt templates for sentence inputs, resulting in predicted category text encodings. For inference, an image is inputted to obtain its image encoding, which is then compared for similarity (e.g., cosine similarity) with the predicted category text encoding. The highest similarity score determines the predicted category for the image.

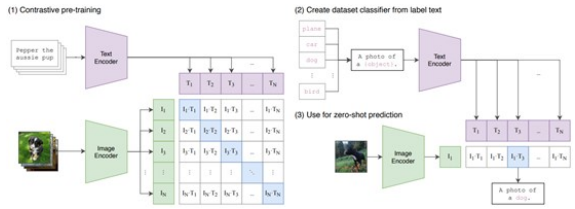

Figure 1. Summary of CLIP

2.7. CAM

To achieve better performance in complex tasks, we often use multi-layered and complex models. However, such models often lack interpretability. Hence, research on the interpretability of complex models has become increasingly important. Therefore, we employed CAM (Class Activation Mapping) [7] and Grad-CAM (Gradient-weighted Class Activation Mapping) [8] for interpretability studies.

Previous research has shown that deeper representations in CNNs capture higher-level visual structures, and convolutional layers naturally preserve spatial information lost in fully connected layers. Therefore, Grad-CAM uses gradient information flowing into the last convolutional layer of the CNN to assign importance values to each neuron for a specific region of interest in decision-making.

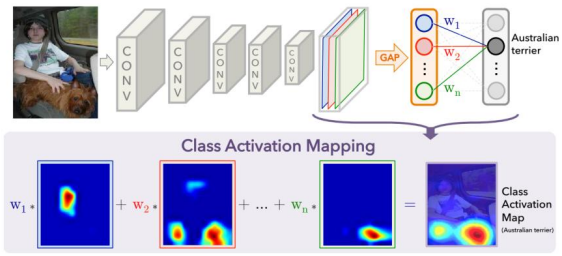

Figure 2. Workflow of CAM

CAM performs global average pooling on the output feature of the convolutional layer before the final output layer (softmax) to obtain input features for the fully connected layer. In this architecture, the weights of the fully connected layer correspond to the output features and reflect the importance of certain regions in the image. This method of projecting weights from the output layer back to the output features of the convolutional layer is called CAM. By linearly summing the output features weighted by the weights of the fully connected layer and overlaying them on the original image, a heatmap can be generated.

Grad-CAM calculates the gradient flowing into the final convolutional layer for any target concept to generate a rough localization map, highlighting the important areas of the image used for predicting the concept.

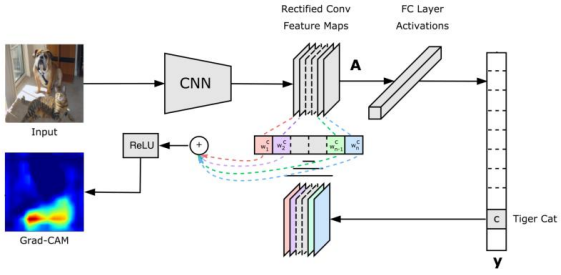

Figure 3. Diagram of CAM Principle

The process of Grad-CAM is divided into two steps.

1. Compute the network's prediction score for a specific category, then backpropagate to calculate the gradient of the prediction score with respect to the output features of the last convolutional layer. Finally, compute global average pooling to obtain importance weight values for each neuron, indicating the importance of a specific feature in the output features for the target category.

2. Use the obtained weights from step one to compute the linear weighted sum of the output features of the convolutional layer, and then apply the ReLU activation function to generate the heatmap. ReLU is applied because we are only interested in features that positively impact the target category, while negative pixels may belong to other categories in the image.

2.8. Robustness of the Model

The concept of model robustness is crucial in the context of machine learning due to its implications for real-world performance under various conditions. Model robustness refers to a model's ability to maintain stable and accurate predictions despite changes or disturbances in its input data, such as noise, errors, variations, or intentional adversarial attacks.

Adversarial attacks represent a significant challenge to model robustness. These attacks involve the deliberate introduction of small, often imperceptible perturbations to input data, with the goal of causing the model to misclassify or produce an incorrect output.

There are several kinds of assumptions of the attacker's knowledge, two of which are: white-box and black-box. A

white-box attack assumes the attacker has full knowledge and access to the model, including architecture, inputs, outputs, and weights. A black-box attack assumes the attacker only has access to the inputs and outputs of the model, and knows nothing about the underlying architecture or weights.

2.8.1 Adversarial Attack

- **FGSM**:

  The Fast Gradient Sign Attack (FGSM) [9] is a method used to generate adversarial examples. Adversarial examples are inputs that have been subtly perturbed to cause machine learning models to make incorrect predictions, despite appearing nearly identical to human observers.

  FGSM operates under the fundamental assumption that by adding small perturbations to input data, models can be induced to produce erroneous outputs. This approach leverages gradient information of the model to generate adversarial examples.

  Consider $\mathbf{x}$ as the original input, y as the corresponding label, θ as the model parameters, and $L(\theta, \mathbf{x}, y)$ as the loss function. The basic idea of FGSM (Fast Gradient Sign Method) is to add perturbations along the direction of the gradient of the loss function with respect to the input data. The specific formula is as follows:

$$\mathbf{x}_{\text{adv}} = \mathbf{x} + \epsilon \cdot \text{sign}\left(\nabla_{\mathbf{x}} L(\theta, \mathbf{x}, y)\right)$$

  - $\mathbf{x}_{\text{adv}}$ is the generated adversarial sample.
  - ϵ is a parameter that controls the magnitude of the perturbation (usually a small value, such as 0.01).
  - $\text{sign}(\cdot)$ is the sign function, which returns the sign of the input (positive is +1, negative is -1).

- **Iterative FGSM**:

  The Iterative Fast Gradient Sign Method (Iterative FGSM) [10] is an extension of the FGSM used for generating adversarial examples. Unlike FGSM, which applies a single-step perturbation, Iterative FGSM applies the perturbation iteratively, refining the adversarial example over multiple steps. This often results in more effective adversarial examples that can more reliably deceive machine learning models.

$$\mathbf{x}_{i+1} = \text{Clip}_{\epsilon,\mathbf{x}}\left\{\mathbf{x}_i + \alpha\, \text{sign}\left(\nabla_{\mathbf{x}} L(\theta, \mathbf{x}_i, y)\right)\right\}$$

  where $\text{Clip}_{\epsilon,\mathbf{x}}$ is an operator assuring element-wise difference between $\mathbf{x}_{i+1}$ and the original clean image $\mathbf{x}$ is within ϵ.

2.8.2 Adversarial Training

To enhance the robustness of a model, ensuring that it can maintain accurate and stable predictions when faced with various perturbations and attacks in the real world, we typically introduce adversarial training. The core idea of adversarial training is to augment data with adversarial examples. This involves training the model using both the original training data and the adversarial examples so that the model performs well not only on normal data but also on data with adversarial perturbations. [9]
Specifically, for FGSM, we aim to find the best θ by minimizing $\tilde{L}(\theta, \mathbf{x}, y)$ over all training data

$$\begin{aligned}\tilde{L}(\theta, \mathbf{x}, y) =& \alpha L(\theta, \mathbf{x}, y) \\ &+ (1-\alpha) L\left(\theta, \mathbf{x} + \epsilon\, \text{sign}\left(\nabla_{\mathbf{x}} L(\theta, \mathbf{x}, y)\right), y\right)\end{aligned} \tag{1}$$

3. Experiment

3.1. Dataset

The datasets used in this study are the CUB200-2011 [11], a fine-grained classification datasets. The CUB200-2011 dataset has a total of 200 bird categories, including 5,994 training images and 5,794 testing data. It provides image-level annotations and keypoint locations, but only image-level annotations will be used in this paper.

3.2. Basic Training

During training, the learning rate is set to 5e-5, with MultiStepLR scheduler and gamma set to 0.1. The optimizer used is AdamW, with weight decay set to 1e-3 and a batch size of 16. The accuracy of ResNet of different depths on the validation set and test set is shown in the table below.

Backbone	Train	Val	Test
ResNet34	0.9983	0.3389	0.3054
ResNet50	0.9881	0.3114	0.3016
ResNet101	0.9862	0.3063	0.2864

Table 1. Basic training results with ResNet of different depths

ResNet34 is a 34-layer deep ResNet with relatively fewer parameters and lower complexity. ResNet50, a 50-layer deep ResNet, introduces a bottleneck structure that increases the number of layers in each residual block. ResNet101 is a 101-layer deep ResNet, featuring more residual blocks and layers, resulting in a deeper network architecture.

In terms of convergence speed, theoretically, ResNet34 should converge faster than ResNet50 and ResNet101 because it has fewer layers, fewer parameters, and less computational load. However, based on the out experiment, all

three models converge in around 30 epochs, with ResNet34 showing a slightly steeper curve.

Regarding accuracy, theoretically, as the network depth increases, the model's capacity to fit the training data improves. Thus, ResNet50 and ResNet101 typically achieve higher accuracy than ResNet34, with ResNet101 usually reaching the highest accuracy due to its larger number of parameters and model capacity. However, the data shows that ResNet34 achieved a training accuracy of 0.9983, higher than ResNet50 (0.9881) and ResNet101 (0.9862). This might be due to the limited data volume or insufficient training time, causing the simpler ResNet34 to perform better. ResNet34's validation accuracy also surpasses the other two networks, supporting our hypothesis.

Batch Size	16	32	64
Val	0.3389	0.3114	0.3063
Test	0.3054	0.3016	0.2864

Table 2. Results with ResNet50 under different batch size

Using ResNet50 with `early_stopping.patience` set to 20, the accuracy for different batch sizes are shown in the table above.

In terms of convergence speed, theoretically, a smaller batch size typically results in more frequent parameter updates and faster gradient descent. Conversely, a larger batch size results in fewer updates and potentially slower convergence. In our experiment, both batch sizes (16, 32 and 64) converge in around 30 epochs. However, because a smaller batch size requires more iterations per epoch, the computation time will increase.

Regarding accuracy, theoretically, a smaller batch size might lead to better adaptation to test data, yielding higher test accuracy and potentially better generalization. Larger batch sizes result in fewer parameter updates and lower accuracy. The actual data confirms this, with a test accuracy of 0.2864 for a batch size of 64, compared to 0.3016 for a batch size of 32 and 0.3054 for a batch size of 16.

Batch size significantly influences the convergence speed and accuracy of a model. A smaller batch size typically leads to faster convergence, better utilization of data, but increases the total training time (due to more batches). Conversely, a larger batch size usually results in fewer batches, shorter training time, but may underutilize the data and requires higher memory capacity. Choosing an appropriate batch size requires balancing convergence speed, hardware memory constraints, and the model's final generalization performance.

3.3. Transfer Learning

Using the ResNet101 32*8d model with `lr = 5e-5`, `batch_size = 32`, and `earlystop.patience = 20`, the results of training different layers are as follows.

Accuracy	FC	4+FC	3+4+FC	2+3+4+FC	All Layers
Train	0.7573	0.9998	0.9998	0.9998	0.9967
Val	0.3539	0.7922	0.8063	0.8306	0.8197
Test	0.3254	0.7745	0.7865	0.8124	0.8021
Loss	FC	4+FC	3+4+FC	2+3+4+FC	All Layers
Train	2.3648	0.0022	0.0029	0.0038	0.0200
Val	3.2990	1.0270	1.0113	0.7661	0.8136

Table 3. Training results with different layers

From the experimental data, comparing training with and without transfer learning shows that, when only training the FC layer and freezing other layers, the training accuracy is lower than without transfer learning (0.7 <0.9). The validation accuracy is similar to without transfer learning, around 0.3. However, when training layers including convolutional layers, both training and validation accuracy improved compared to without transfer learning, with training accuracy consistently around 0.99 and validation accuracy improving significantly from 0.2-0.3 to around 0.7-0.8. This is because transfer learning helps the model generalize better with limited data by leveraging features learned from large datasets.

From a training time perspective, models with transfer learning generally converge in about 10 epochs, faster than without transfer learning (around 30 epochs). This is because the pretrained model has already learned general features and only requires fine-tuning on the target task, significantly reducing training time.

Regarding training different layers, training only the FC layer yields the worst performance across both training and validation datasets, in terms of both accuracy and loss. Training 2+3+4+FC layers yields the best performance, with validation accuracy reaching a solid 0.8306. This suggests that while pretrained features are useful, they may not be optimal for the target task. Merely training the FC layer restricts the model to pre-extracted features, which may not be the best for the target task. Training more layers allows the model to adapt these features better, enhancing performance.

When training all layers, the slight decrease in validation accuracy compared to training 2+3+4+FC layers may be due to overfitting, where the model loses some of the general features captured by the pretrained weights. This indicates that fine-tuning all layers might disrupt these weights, leading to decreased generalization.

In summary, transfer learning leverages knowledge from source tasks to improve target task performance and reduce training time, especially useful when data is limited or training time is constrained. Its effectiveness is clearly demonstrated in this experiment.

3.4. Various Training Tricks

In this section, we will use ResNet50 as the baseline and explore the impact of various training techniques on the

model's performance.

Method	Accuracy
baseline (Pretrained Resnet50)	0.7485
Geometric Transformations	0.7814
CutOut	0.7356
Mixup	0.7654
Label Smoothing	0.7685
Dropout	0.7412
Geometric Transformations + Label Smoothing + Mixup	0.8054
Apply the Above Methods to Resnet101_32x8d	0.8351

Table 4. Training results with various training tricks

- **Geometric transformations**: Applying geometric transformations improves the accuracy to 0.7814. This significant increase suggests that data augmentation through geometric transformations is effective in enhancing the model's generalization.
- **Cutout**: The accuracy drops to 0.7356 when using the CutOut technique. This result indicates that CutOut may not be beneficial for this particular model and dataset, possibly due to the loss of critical information in the images.
- **Mixup**: The Mixup technique yields an accuracy of 0.7654, showing a moderate improvement over the baseline. Mixup helps in creating new training examples through linear combinations of existing ones, which can improve the robustness of the model.
- **Label Smoothing**: Label smoothing leads to an accuracy of 0.7685. This technique, which softens the target labels, helps in preventing the model from becoming overly confident, thus improving its performance.
- **Dropout**: Applying dropout results in a slight decrease in accuracy to 0.7412. This suggests that dropout might not be very effective for this specific configuration, possibly due to the already regularized nature of the pretrained ResNet50.

By combining the significantly effective training techniques from our experiments (geometric transformations, label smoothing, mixup), we achieved an accuracy of 0.8054 on the test set using ResNet50. This indicates a synergistic effect where the benefits of each technique complement each other, leading to a substantial performance gain.

Finally, we applied thecombined techniques to a more complex model, ResNet101_32x8d, the accuracy further increases to 0.8351. This demonstrates that a more powerful architecture can leverage these training tricks even more effectively, resulting in a considerable improvement in performance.

3.5. GAN

When using cGAN with a projection discriminator to generate images with specified labels, we employed a Generator and Discriminator based on ResNet and applied projection on the Discriminator. This projection mapped the conditional information to an embedding space and computed the inner product of this embedding with the image features to better combine conditional information and image features. For the loss function, we utilized a hinge version based on the standard GAN loss. In our experiments, we used the Adam optimizer with hyperparameters set to α = 2e-4, $\beta 1 = 0$, $\beta 2 = 0.9$. We updated the Discriminator five times for every Generator update.

We tested our approach on CUB_200_2011, which contains 200 bird species images with approximately 50-60 images per category. We resized input images to 64*64 and generated images of the same size.

During experimentation, we conducted 5000 iterations. Towards the end of these iterations, the losses of the Generator and Discriminator had stabilized, indicating that the model was nearing convergence.

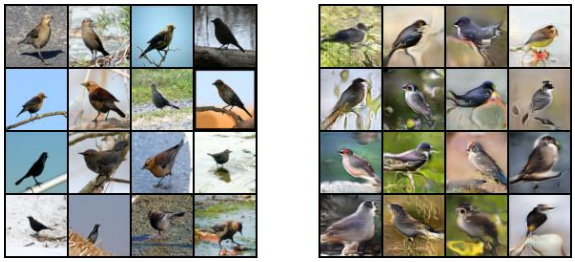

Figure 4. Comparison between Real Images (left) and Generated Images (Right) for class 11

Figure 4 shows a comparison between real and generated images for category 11 (Rusty_Blackbird). It can be observed that the generated images roughly capture the overall outlines accurately but lack in finer details. Additionally, due to the small scale of the dataset and limited data per category, the final generated model performed poorly across categories, failing to adequately represent the distinct characteristics of different categories in the generated images.

3.6. ViT

To ensure consistency with previous experiments, we used the AdamW optimizer, Cross-Entropy loss function, MultiStepLR training strategy with gamma set to 0.1 and milestones set to [30, 60, 90], learning rate set to 5e-5, batch size set to 32, and early stopping with patience set to 20. We loaded a pre-trained ViT model and, given the relatively small size of the CUB-200-2011 dataset, we initially used the base ViT model `vit_b_16`. We fine-tuned the pre-trained model, and the results of training with dif-

ferent numbers of frozen layers in `vit_b_16` are shown in Table 5.

Accuracy	only heads	heads + 6attn	heads+12attn
Train	0.8309	0.9985	0.9996
Val	0.6060	0.8105	0.8189
Test	0.6116	0.7972	0.7984
Loss	only heads	heads + 6attn	heads+12attn
Train	1.0576	0.0592	0.0088
Val	1.6118	0.7629	0.7274

Table 5. `vit_b_16` training results with different frozen layers

Next, we compared the training effects of using a larger model, `vit_l_16`, with those of `vit_b_16`, unfreezing all the self-attention layers and the classification head. The comparative training results are shown in Table 6.

Model	**Accuracy**
`vit_b_16`	0.7984
`vit_l_16`	0.7850

Table 6. Comparison of training results with different vits models

It can be observed that for the relatively small CUB-200-2011 dataset, the smaller `vit_b_16` model performs better. This is because simpler models tend to have better generalization abilities on smaller datasets, while more complex models require more training data to learn parameters effectively and to prevent overfitting.

Finally, we applied the training techniques from the previous experiments and compared the training effects on `vit_b_16`. The results are shown in Table 7.

Training Technique	**Accuracy**
Baseline	0.7984
add Mixup + Label Smoothing	0.8019
add drop out	0.8017

Table 7. training results using different training techniques

3.7. VLM

CLIP obtains supervision signals directly from textual descriptions, which significantly enhances its generalization capability. It exhibits strong zero-shot performance, achieving comparable results to ResNet-50 on ImageNet without fine-tuning. Therefore, our initial approach is to use the CLIP model without fine-tuning for the task of image classification on our dataset.

We employed multiple CLIP models, where category text is structured using the prompt template "a photo of a" followed by the names of 200 bird species. The specific process involves encoding the category text first. Subsequently, each bird image is encoded using the model, and cosine similarity is computed between the encoded image and the encoded category text for each bird species. The category with the highest similarity score is predicted, which is then compared against the actual category to compute accuracy.

Model	**Accuracy**
clip-vit-base-patch32	0.3882
ViT-B-32.pt	0.4502
ViT-B-16.pt	0.5002
ViT-L-14-336px.pt	0.5813
CLIP-ViT-g-14-laion2B-s12B-b42K	0.7408
clip-vit-base-patch32(fine-tuned)	0.7426

Table 8. Different training results of CLIP models

In this task, zero-shot CLIP achieved a prediction accuracy of approximately 50% to 60%. We believe this is because, although CLIP has been trained on a large amount of data (400 million data points), tasks that require specific domain knowledge, such as classifying bird images, demand additional prior knowledge to perform well. However, when the model scale increased to a certain extent (CLIP-ViT-g-14-laion2B-s12B-b42K, 5G), the model's prediction accuracy reached 70% to nearly 80%, indicating that as the model scale increases, its generalization ability and capability to handle specific domain tasks are significantly enhanced. Additionally, we fine-tuned the CLIP-ViT-base-patch32 model, and it also achieved an accuracy of over 70%, showing that the performance of CLIP models on specific domain tasks can be further improved through fine-tuning to better adapt to specific task requirements. In summary, although CLIP can effectively distinguish between broad categories (such as birds and dogs), identifying specific bird species requires more supplementary information to achieve better performance.

3.8. CAM

To generate heatmaps based on CAM for the last convolutional layer, it is necessary to register a forward hook function in the inference function to extract and save the input and output features of the model's convolutional layers. Then, you need to read the input features of the global average pooling layer (i.e., the output features of the last convolutional layer) and the weights of the fully connected layer to calculate the heatmap.

For generating heatmaps of other convolutional layers based on CAM, I registered a hook for the last convolutional layer and then read the input features (i.e., the output features of the second-to-last convolutional layer) when extracting features. Since their sizes do not match (256 and 512), I resized the input features through a convolution operation to match the size of 512 before multiplying them

Figure 5. Heatmap for Convolutional Layers

Figure 6. Heatmap Generated by Grad-CAM

with the weights. However, this method does not provide strong interpretability.

Figure 7. Heatmap for Convolutional Layers

Figure 8. Heatmap Generated by Grad-CAM

In Grad-CAM, the heatmap is obtained by processing gradients and output features. To obtain gradient information, a backward hook function needs to be written and registered in the inference function. Then, execute inference in the inference function, calculate the cross-entropy loss, and perform backpropagation to obtain gradient information regarding the output features with respect to the inference score. After obtaining the output features and gradient information, the weights are calculated by averaging the gradients across channels. These weights are then multiplied by the corresponding channels of the output features to obtain the heatmap.

The CAM and Grad-CAM heatmaps shown above (Figures 9-12) were generated for the same model. It can be observed that the heatmaps generated by CAM are mostly focused on the birds, indicating that CAM is able to effectively capture bird features and concentrate on the target area. In contrast, the heatmaps generated by Grad-CAM are more dispersed, with only a small portion focusing on the birds, suggesting that Grad-CAM is less precise in identifying the target in this set of experiments. This could be due to the sparsity of gradient information or noise interference leading to dispersed focus points.

Comparing the two situations, we believe the reason is that CAM directly utilizes the output features of the convolutional layer and the weights of the fully connected layer, whereas Grad-CAM relies on gradient information for weighted averaging. During this process, the calculation of gradients may be affected by noise in the backpropagation path, causing the generated heatmap to be less focused. Additionally, the more complex calculation process of Grad-CAM may accumulate more errors due to the multiple layers involved. These factors collectively lead to the more dispersed focus points in Grad-CAM heatmaps.

In summary, while CAM is simpler to implement and can better concentrate on the target area, Grad-CAM uses gradient information, making it more versatile. Both have their advantages.

3.9. Robustness of the Model

To test the robustness of the model, we use FGSM to generate adversarial examples to attack the model. We set the ϵ in FGSM to 0, 0.05, 0.1, 0.15, 0.2, 0.25, and 0.3, respectively.

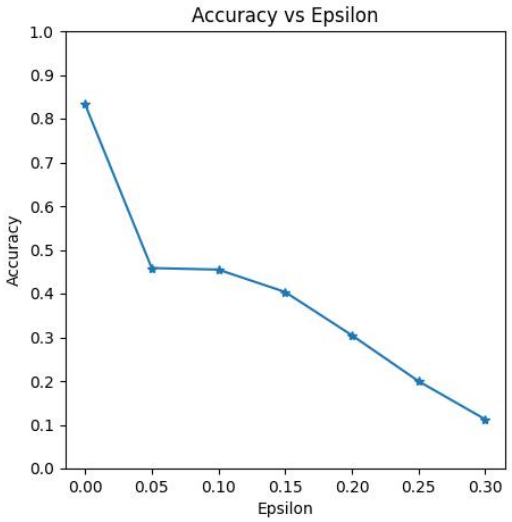

Figure 9. Model (backbone = resnext101_32x8d) performance under FGSM attacks with varying ϵ values.

From Figure 9, we can conclude that as the ϵ value in FGSM increases, the model's performance rapidly declines. This indicates that the original model has poor resistance to perturbations. We show some examples of successful adversarial examples at each epsilon value in Figure 10. Although as epsilon increases the test accuracy decreases, but the perturbations become more easily perceptible.

To enhance the robustness of the model, ensuring that it can maintain accurate and stable predictions whtn faced with various perturbations and attacks in the real world, we employed adversarial training against FGSM. We then tested the model's resistance to FGSM after using adversarial training.

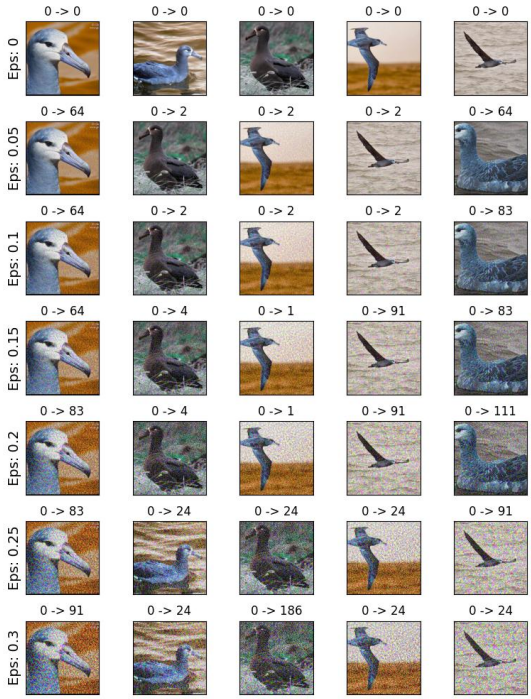

Figure 10. Adversarial Example

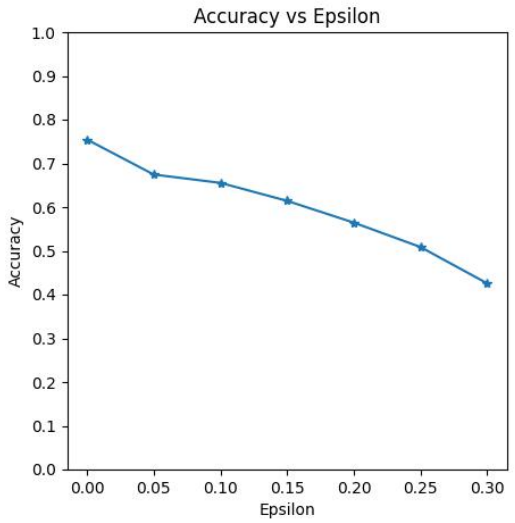

Figure 11. Model performance after adversarial training under FGSM attacks with varying ϵ values.

Shown in Figure 11, indicate a marked improvement in robustness. The model trained with adversarial examples shows a significantly slower decline in performance as ϵ increases, however, this comes at the cost of a decrease in accuracy on pure input, compared to the original model. This demonstrates that adversarial training effectively enhances the model's ability to withstand adversarial perturbations.

3.10. Empirical Evaluation

To investigate the generalization performance of the models, we transferred the models to another dataset, Stanford Dogs [12], including the complete fine-tuning training process, and compared the performance of different methods across various datasets.

Method	CUB_200	Stanford Dogs
ResNet101_32x8d	0.8351	0.8756
ViT_b_16	0.8019	0.8952
CLIP-ViT-G-14(zero-shot)	0.7408	0.7171

Table 9. Evaluating model performance on different datasets

ResNet101_32x8d performed relatively well on both the CUB_200 and Stanford Dogs datasets, achieving accuracy rates of 0.8351 and 0.8756, respectively. This indicates that ResNet101_32x8d has strong feature extraction capabilities and good generalization performance.

ViT_b_16 achieved the highest accuracy on the Stanford Dogs dataset (0.8952) and also performed well on the CUB_200 dataset (0.8019). This suggests that the Vision Transformer (ViT) has strong adaptability when processing image data, especially when dealing with more complex images.

CLIP-ViT-G-14 (zero-shot) performed worse than the other two models on both datasets, with an accuracy of only 0.7171 on the Stanford Dogs dataset. This may be because the CLIP model was primarily designed for cross-modal (vision-language) tasks, and its performance in purely image classification tasks was not as good as the specifically trained ResNet and ViT models due to the lack of fine-tuning.

4. Conclusion

Based on our extensive exploration and experimentation in fine-grained visual classification (FGVC), we have made significant strides in understanding and enhancing model performance across various facets of this challenging domain. Throughout our study, which primarily focused on the CUB200-2011 dataset, we thoroughly investigated foundational training using ResNet models of varying depths and batch sizes. We studied the impact of model architecture and training configurations on final accuracy and convergence speed, crucial for establishing a solid foundation for further research.

In the domain of transfer learning, our use of the ResNet101 32x8d model emphasized the importance of layer-wise training, revealing subtle performance improvements through selective training of model layers. Exploring diverse training techniques such as geometric transfor-

mations, CutOut, Mixup, Label Smoothing, and Dropout allowed us to uncover their individual impacts and synergistic effects on model performance, providing insights and effective strategies for enhancing classification accuracy in FGVC tasks.

Furthermore, our explorations into generative adversarial networks (GANs), Vision Transformers (ViT), and Vision-Language models (VLM) offered valuable perspectives, leveraging advanced architectures and learning paradigms to address FGVC challenges comprehensively. We observed improvements in model robustness, generalization capability, and interpretability through these approaches.

Additionally, the implementation of CAM and Grad-CAM techniques enriched our understanding of how models perceive and interpret fine-grained image features, facilitating more comprehensive analysis and visualization of classification decisions.

In addressing model robustness, our evaluations against FGSM attacks and subsequent adversarial training demonstrated effective strategies for enhancing model resilience against adversarial perturbations, crucial in real-world deployment scenarios where security and reliability are paramount.

Lastly, our assessment of model generalization across different datasets showcased the diversity and adaptability of our approaches. In conclusion, our comprehensive study not only delved deeply into various technologies in the FGVC field but also conducted a variety of experiments aimed at further improving the accuracy, robustness, and practical applicability of visual classification models in complex and dynamic real-world environments.

5. Contribution

- Yulin Zhou:40%
- Kunsheng Liu:30%
- Xinyu Chen:30%

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016. 2

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020. 3

[3] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014. 3

[4] T. Miyato and M. Koyama, "cgans with projection discriminator," *arXiv preprint arXiv:1802.05637*, 2018. 3

[5] S. Bhojanapalli, I. Schlag, and R. Salakhutdinov, "Three things everyone should know about vision transformers," *arXiv preprint arXiv:2203.09795*, 2022. 3

[6] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*, pp. 8748–8763, PMLR, 2021. 3

[7] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2921–2929, 2016. 4

[8] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, pp. 618–626, 2017. 4

[9] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014. 5

[10] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016. 5

[11] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The caltech-ucsd birds-200-2011 dataset," 2011. 5

[12] A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei, "Novel dataset for fine-grained image categorization," in *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, (Colorado Springs, CO), June 2011. 10

[13] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[14] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[15] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," in *International conference on machine learning*, pp. 2642–2651, PMLR, 2017.