Introduction to Machine Learning exam report: Fine-Grained Image Recognition

Filippo Costamagna, Giuseppe Curci, Damiano Zaccaron

June 2024

Abstract

Unlike traditional image classification, which deals with broad object categories, fine-grained classification aims to identify subtle differences between closely related classes (e.g., different bird species). Previous works have investigated different strategies, including architectural modifications, novel loss functions, and specialized neural network architectures. In our work, we employ four base neural networks as our foundation. We then try to improve their performance by experimenting with a different optimizer and incorporating plug-in networks specifically designed for fine-grained tasks. Our results demonstrate that these modifications lead to significant performance gains. However, the extent of improvement varies depending on the specific dataset and base architecture. Additionally, we attempt to interpret what drives the predictions of the models by generating heatmaps using a base Grad-CAM (Gradient-weighted Class Activation Mapping). This method provides insights into how different models make predictions, shedding light on their decision-making processes. Our code and experiments are available at https://github.com/giuseppecurci/Fine-Grained-Visual-Classification.

1 Introduction

Unlike conventional image classification, fine-grained classification addresses the challenge of distinguishing between visually similar subcategories. The goal is to retrieve and identify images belonging to multiple subordinate classes within a super-category (also known as a meta-category or basic-level category). Examples include distinguishing between different species of animals or plants, various car models, or diverse retail products.

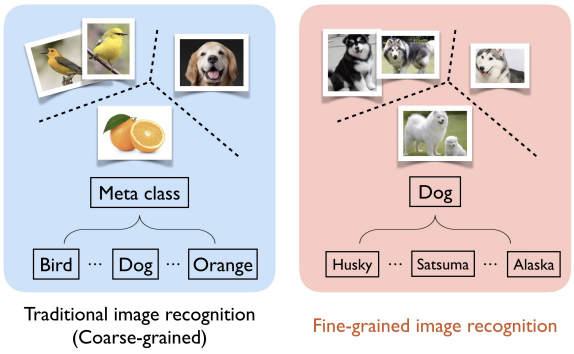


Figure 1: (image source)

Traditional image classification techniques often fall short due to the subtle distinctions required for accurate categorization. Therefore, task-specific adaptations are necessary, including customized loss functions, specialized neural network architectures, and tailored training strategies [1].

In our report, we evaluate some of the methods that have already been devised by other authors. We evaluate four distinct models, leveraging both convolutional neural networks (CNNs) and vision transformers. Specifically, we explore ResNet-50 and EfficientNet-B5 as CNN-based backbones, along with ViT (Vision Transformer) and Swin-Transformer. Moreover, we utilise two different optimizers: stochastic gradient descent (SGD) with momentum and Sharpness-Awareness Minimization (SAM). The latter was tested as it showed to achieve SOTA performance on different fine grained datasets such as "Food-101" and "Birdsnap" [2].

Beyond these baseline experiments, we also use the following plug-in task-specific architectures that can be seamlessly integrated with nearly any backbone. One is the Cross-Layer Mutual Attention Learning Network (CMAL-Net) [3] and the Plug-in Module (PIM) [4]. By combining these components, we aim to achieve superior fine-grained classification results.

To understand which image regions drive predictions, we use a popular tool in computer vision interpretability: Grad-CAM (Gradient-weighted Class Activation Mapping) [5]. Grad-CAM generates heatmaps that are subsequently overlaid on the original image, highlighting the crucial areas (depicted in red) that influence the model's decision-making process.

All the experiments were conducted on three datasets: Flowers102, FGVCAircraft and Mammalia. Flowers102 contains 102 flower categories, with each category having 40-258 images. FGVCAircraft consists of 10,000 images of aircraft across 100 different aircraft model variants. Finally, Mammalia is composed of 6000 images of different mammals divided in 100 balanced classes.

2 Related Works

2.1 Survey of Key Techniques

The task of Fine-Grained Image Recognition (FGIR) is challenging due to the subtle differences between classes and the high intra-class variability. Over the years, researchers have developed various techniques and models to address these challenges. In this subsection various methodologies and models used in the field of FGIR are surveyed.

The use of deep learning approaches has been predominant in the field of fine-grained image recognition, and remarkable progress has been made in deep learning powered FGIA. [6] Two primary architectures in this domain that have been considered in this research are Convolutional Neural Networks (CNNs) and Transformers.

2.1.1 Convolutional Neural Networks (CNNs)

A CNN is a type of deep learning model that is particularly well-suited for analyzing visual data. It is designed to recognize patterns in images by applying filters (learned weights of the convolutional layers) to the input data, which are then used to identify features and classify objects within the images[6]. It consists of multiple layers designed to automatically and adaptively learn spatial hierarchies of features from input images.

In fact, the hierarchical feature learning plays a fundamental part when it comes to the task of FGIR: CNNs capture local patterns (such as edges and textures) in the initial layers and more complex and abstract patterns in deeper layers[3]. This hierarchy, that allows for identification of more detailed features, is crucial for distinguishing subtle differences in fine-grained categories.
Furthermore, pooling layers in CNNs provide spatial invariance, making the models robust to translations and slight variations in object position.
However, training deep CNNs can be computationally expensive and time-consuming to train, and CNNs with a high number of layers and/or parameters are prone to overfitting. That said, they represent a more than valid approach for FGIR tasks, as demonstrated by the many state-of-the-art results in FGIR achieved using CNNs[7].

2.1.2 Transformers

Transformers, initially designed for natural language processing, have recently been adapted for image recognition tasks, demonstrating impressive results in various vision applications[8]. Unlike CNNs, which use convolutional layers to process images, transformers rely on self-attention mechanisms to model long-range dependencies.
Transformers use self-attention to weigh the importance of different parts of the image,

allowing them to focus on relevant features regardless of their spatial distance. They provide a powerful mechanism for capturing global dependencies and contextual information, which can enhance the model's ability to recognize fine-grained details spread across the image.

Compared to CNNs, Transformers tipically require larger datasets and more computational resources due to their high capacity and the need for extensive training to learn effective representations, but with sufficient data they might outperform Convolutional Networks.
Moreover, Transformers are highly flexible and can be adapted to various tasks without significant architectural changes.
As was the case for CNNs, Transformers too are computationally expensive and time-consuming to train. They are also prone to overfitting when trained on smaller datasets. Still, their ability to capture global features that may be overlooked by CNNs, and the great performances they can obtain with larger datasets, make them an excellent approach to be considered in this report.

2.2 Relevant Datasets

Many fine-grained benchmark datasets covering diverse domains have been released in the past few years[6]. Some of the most popular are listed below:

- **CUB-200-2011**: It is a widely used benchmark for fine-grained bird species recognition. It contains 11,788 images of 200 bird species[9].
- **FGVC-Aircraft**: The FGVC-Aircraft dataset contains 10,000 images of 100 aircraft variants, annotated with variant, family, and manufacturer labels[10].
- **Stanford Cars**: It comprises 16,185 images of 196 classes of cars, with each class corresponding to a specific make, model, and year[11].
- **Flowers102**: The Oxford Flowers 102 dataset consists of 8,189 images of 102 different flower species. Each class has between 40 and 258 images, and the dataset includes labels for each flower category[12].

These datasets have all been taken into consideration for the training and evaluation of the models, with the choice eventually falling on *FGVC-Aircraft* and *Flowers102*. This choice was made so that the performances of the models could be evaluated on both datasets with balanced (FGVC-Aircraft) and unbalanced (Flowers102) classes.

3 Proposed Solutions

Data Preprocessing

To improve the accuracy of the classifier and to create a larger and more diverse dataset, a choice has been made to utilise a data augmentation tool. Data augmentation is a process in which additional data is generated artificially by applying various transformations and modifications (such as flipping, cropping, ...) to the original data.
Beside exposing the model to a larger dataset, which addresses possible problems of limited training data, data augmentation helps reduce overfitting and improves robustness by introducing variations and perturbations to the training data.

In particular, a choice was made to utilise the AutoAugment procedure, which automatically searches for improved data augmentation policies [13].
While it is possible to train the AutoAugment function on any given dataset to optimise the policy, this is a time-consuming and computationally-costly operation. Therefore, we decided to opt for policies that were pre-trained on the *ImageNet1k* database, considering that it is a large and diverse dataset that includes images of the categories we tested our models on.

Input images have also been randomly cropped to a dimension of (224,224). They have also been resized to (260,260) for the *FGVCAircraft* and *Mammalia* datasets and to (300,323) for the *Flowers102* dataset.
The input tensors have been normalized using the *ImageNet1k* mean and standard deviation parameters.

3.1 Models

3.1.1 Residual Network (ResNet)

ResNet is a deep learning model designed in 2015 by Kaiming et al.[14]. It introduced, for the first time, the concept of residual connections, that aimed to ease the training of deep neural networks and reduce training error.

The architecture is based on the idea of residual mapping, defined as $F(x) := H(x) - x$, where $F(x)$ is the residual mapping, $H(x)$ is the desired output and x is the input. The original mapping is then recast so that the output of a residual block is $y = F(x) + x$. The idea is that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping.

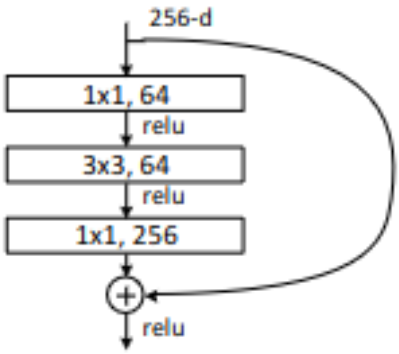


Figure 2: Bottleneck building block for ResNet-50

The ResNet architecture consists of several stages, each of which includes several residual blocks (Fig. 1). In a residual block, the input is passed through several 3x3 convolutional layer, after each of which a batch normalization is performed and the result is passed through a ReLU activation function. Before passing the output to the final activation function, the original input is added to the output of the block through a shortcut connection, which specifically performs an identity mapping. Identity shortcut connections add neither extra parameter nor computational complexity.
With regards to ResNet-50, the model used in this report, it utilises three convolutional blocks (Bottleneck Blocks) per layer.

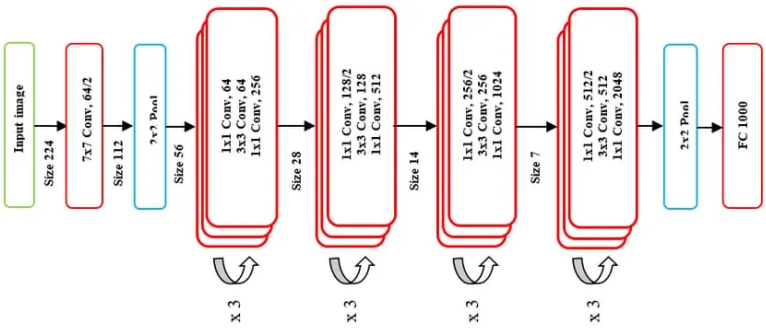


Figure 3: ResNet-50 architecture

ResNet has achieved state-of-the-art performances on several image recognition benchmarks[7] and has been widely used in computer vision tasks.

3.1.2 EfficientNet

Convolutional Neural Networks are commonly developed at a fixed resource budget, and then scaled up for better accuracy only if more resources are available. The main intuition and contribution that Efficient Net [15] brings to the world of deep learning is that

scaling the model without a sound criterion by trial and error is highly inefficient, time-consuming and detrimental in terms of performance. On this basis, they propose a new scaling method that uniformly scales all dimensions of depth/width/resolution using a compound coefficient ϕ. Specifically, the dimensions and the optimization problem is defined as follows

$$\begin{aligned} \text{depth}: &\quad d = \alpha^\phi \\ \text{width}: &\quad w = \beta^\phi \\ \text{resolution}: &\quad r = \gamma^\phi \\ \text{s.t. } &\alpha \cdot \beta^2 \gamma^2 \approx 2 \\ &\alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned} \tag{1}$$

where α, β, γ are constants that are determined with a small grid search and ϕ becomes a parameter specified by the user. The larger the selected ϕ is the larger the model will be.

To develop the baseline network EfficientNet-B0 the authors used a multi-objective neural architecture search that optimizes both accuracy and FLOPS. The model uses the following blocks and techniques:

- Mobile inverted bottleneck (MBConv): taken from [16], it consists of a residual block where the first step expands the number of channels, then applies a depthwise convolution and shrinks the output to match the initial number of channels and add the initial input through the skip connection

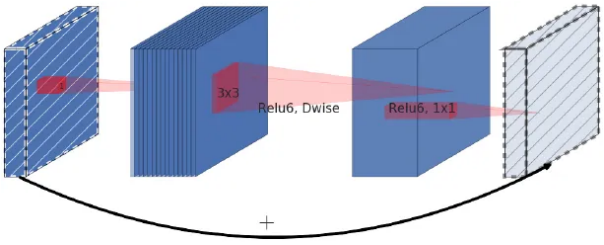


Figure 4: An inverted residual block connects narrow layers with a skip connection while layers in between are wide. Different activation functions can be applied and in Efficient Net SILU is used.

- linear bottleneck with Batch Normalization to alleviate the loss of information due to the convolutions
- squeeze and excitation between the convolutions
- stochastic depth which consists of randomly dropping some layers at each epoch

After finding the model, the authors applied the gridsearch to find the parameters α, β, γ and then use ϕ to create 7 models which are essentially the scaled version of EfficientNet-B0.

3.1.3 Vision Transformer (ViT)

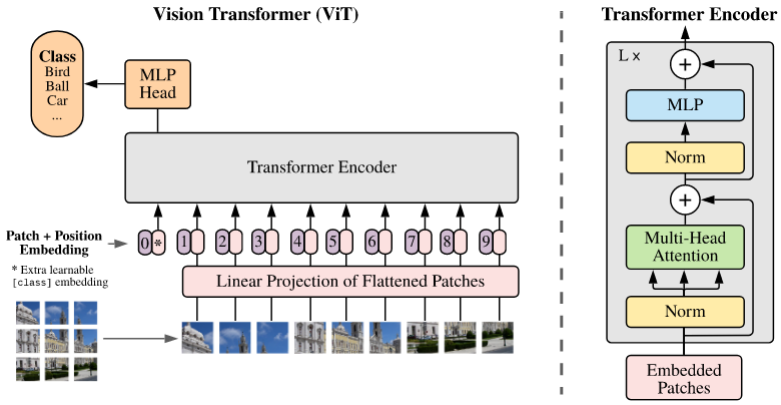


Figure 5: Vision Transformer Encoder Architecture

Vision Transformers introduced by [17] are the first successful attempt to use transformers in computer vision trying to modify the architecture as little as possible. To achieve this result, the authors adapt the work flow as follows:

- the input images are reshaped into a sequence of flattened 2D patches $x_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$ where (P, P) is the resolution of the image and N is the number of patches
- the patches are mapped to D dimensions with a trainable linear projection layer
- to encode the position of the patches, a positional embedding is added to the its patch embedding before feeding it to the standard Transfomer
- the Transformer is attached to an MLP head to produce the final classification

The smaller the patches, the more numerous they will be which will result in a higher computational cost. Depending on the patch size on the version of the model (ViT-B/L/H) the number of parameters is significantly different ranging from a minimum of 86M parameters up to a maximum of 633M.

3.1.4 Swin Transformer

The Swin Transformer [18] represents a significant advancement in the architecture of Vision Transformers, addressing the challenges posed by the attention mechanism when applied to image data. Traditional Vision Transformers encounter scalability issues because the computational complexity of the self-attention mechanism grows quadratically with the number of image patches. The Swin Transformer mitigates this by computing self-attention within local windows, rather than across the entire image. This local windowed approach reduces the computational complexity to linear in relation to the image size, as the number of patches within each window remains constant. The Swin

Transformer begins by dividing the input image into patches, typically of size 4x4 pixels. Each patch is flattened into a vector and treated as a token, analogous to how words are treated in natural language processing models. Afterwards, each token enters the core Swin transformer block, where it is initially passed trough a linear embedding which projects it into an higher dimension space, to then apply the Window Self-Attention (WSA), where self-attention is computed within local, non-overlapping windows. This confines the self-attention calculation to small, manageable regions of the image, thus reducing the overall computational burden. Afterwards, after another linear embedding and an MLP head, Shifted Window Self-Attention is applied, where the windows are shifted by a fixed amount (e.g., shifting by half the window size). This mechanism allows the model to capture interactions between patches that were in separate windows in the previous layer, facilitating the learning of cross-window relationships and ensuring global context aggregation across the entire image.

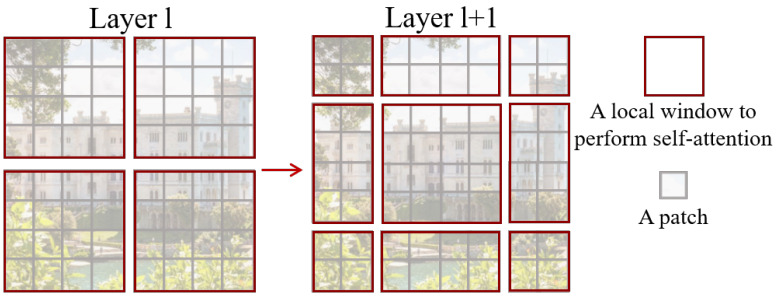


Figure 6: The mechanism of shifted window attention. By shifting the region over which attention is applied, the model can capture cross-region relations

After each Swin Transformer block the patches are merged. This involves concatenating the tokens from neighboring patches and then linearly projecting them back to a fixed size. This process effectively reduces the number of tokens by combining 2x2 patches into a single token, reducing the spatial resolution while increasing the feature dimension. The merged patches are treated as new tokens and undergo another set of Swin Transformer Blocks and merging operations. This process is repeated a total of 4 times, building a hierarchical structure that makes it easier to capture both local and global features.

3.2 Optimizers

An optimizer is an algorithm designed to adjust the parameters of a model to minimize the loss function, thereby improving the model's performance. We performed our experiments with two different optimizers. One is the common Stochastic Gradient Descent (SGD), which given its popularity is not described here, and the other is Sharpness-Awareness Minimization (SAM). As mentioned previously, the optimizer was taken into

consideration because it was demonstrated to significantly improve model performance on fine-grained datasets with little increase in computational cost. [2].

3.2.1 Sharpness-Awareness Minimization (SAM)

Unlike SGD and other optimizers that only minimize training loss, SAM also minimizes loss sharpness. In other words, SAM looks for parameters that are in uniformly low loss neighborhoods, which will result in better model generalization capability and improved robustness to label noise. The radius that defines the neighbourhoods is called ρ and is a hyper parameter. In the paper they found $\rho = 2$ to work well [2].

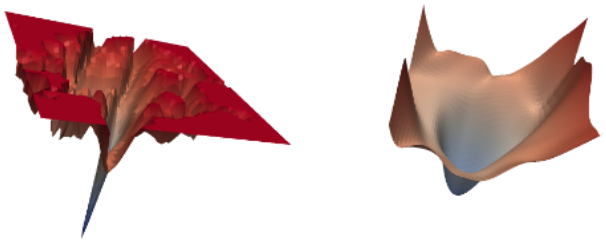


Figure 7: (left) A sharp minimum to which a ResNet trained with SGD converged. (right) A wide minimum to which the same ResNet trained with SAM converged. (taken from [2])

The formulation of the problem results in a min-max problem that can be solved a gradient descent algorithm. The SAM algorithm works as follows:

1. initialize weights w_0, $t = 0$
2. while not converged do
   (a) sample batch $\mathcal{B} = (x_1, y_1), \ldots, (x_b, y_b)$
   (b) first learning step: compute gradient $\nabla_w L_{\mathcal{B}}(w)$ and compute $\hat{\epsilon}$ (which is the perturbation to be added to w)
   (c) second learning step: compute the actual optimization step $g = \nabla_w L_{\mathcal{B}}(w)|_{w+\hat{\epsilon}(w)}$
   (d) update weights: $w_{t+1} = w_t - \eta g$
   (e) $t = t + 1$

Both learning steps require backpropagation so SAM requires twice the computations required for SGD. The reason why SAM takes the actual update step from $w + \hat{\epsilon}(w)$ is that that point corresponds to a local maximum in the neighbor ρ. Thus SAM not only minimizes the loss value, but also its sharpness.

3.3 Fine Grained Architectures

3.3.1 PIM

PIM (Plug-in module) [4]proposes to resolve one of the core challenges of fine-grained image classification, that is, as opposed to coarse-grained visual classification, the necessity to have field experts to correctly label the images, making data more expensive. To tackle this challenge, numerous strategies aim to automatically identify the most distinctive regions and utilize local features for more precise characterization. These strategies only need image-level annotations, which lowers annotation costs. The peculiarity of PIM resides in the fact that it is compatible with various common backbones, including both CNN-based and Transformer-based networks, helping them to highlight highly discriminative regions. This plug-in module generates pixel-level feature maps and integrates filtered features to enhance fine-grained visual classification. Experimental results indicate that the module had surpassed the state-of-the-art methods at the time of its release , achieving notable accuracy improvements of 92.77% on CUB200-2011 and 92.83% on NABirds. The idea behind this method is that the non-discriminative regions (such as the background patches of the image) will appear more widely across all the different classes of data, and using them as training data will result in a mostly flat predicted distribution of the test images while conversely preferring the distinctive regions will lead to more distinguishable predictions on the targets. This is due to the fact that very similar training data which is assigned to different classes can cause difficulties in convergence in the backpropagation. To do so, PIM adopts a "Division/Competition/Combination" approach, by dividing the image in patches and selecting the candidate background and object regions by looking at the predicted class score. The selected features from different scales will then be combined to give the prediction, reducing the noise coming from the background and focusing on the important regions. The backbone network, such as ResNet or Vision Transformer, is used to extract feature maps from the input image. These feature maps, denoted as $f_i \in \mathbb{R}^{C \times H \times W}$, are produced by different blocks in the backbone. Here, C is the number of channels, H is the height, and W is the width of the feature map. By inputting the feature map into a weakly supervised selector (WSS), which assigns a classification to each feature point using a linear classifier we can produce a modified feature map, labeled as $f_i \in \mathbb{R}^{C' \times H \times W}$, where C' represents the number of target classes. Following this, softmax is applied to determine the probability of each feature point belonging to a certain class, allowing the weakly supervised selector to choose the feature points with the highest confidence scores. The resulting selected points are then fused by a graph convolution, an operation in graph neural networks that generalizes the concept of convolution from traditional grid-structured data (like images) to graph-structured data. In particular, the selected feature points are treated as nodes of a graph relating together features across various spatial positions and scales.

This graph is fed into the graph convolutional network, enabling the model to learn the connections between different nodes. Subsequently, the feature points are grouped into several super nodes using a pooling layer. Finally, the features of these super nodes are averaged, and a linear classifier is applied to make predictions. This method enhances efficiency in integrating the features of individual points without significantly affecting the results generated by the backbone model. Additionally, a Feature Pyramid Network (FPN) architecture [19], widely used in object detection to extract semantic value from the higher-level feature maps to the lower-level ones, is added to the backbone network to fuse information of different scales and improve the recognition performance.

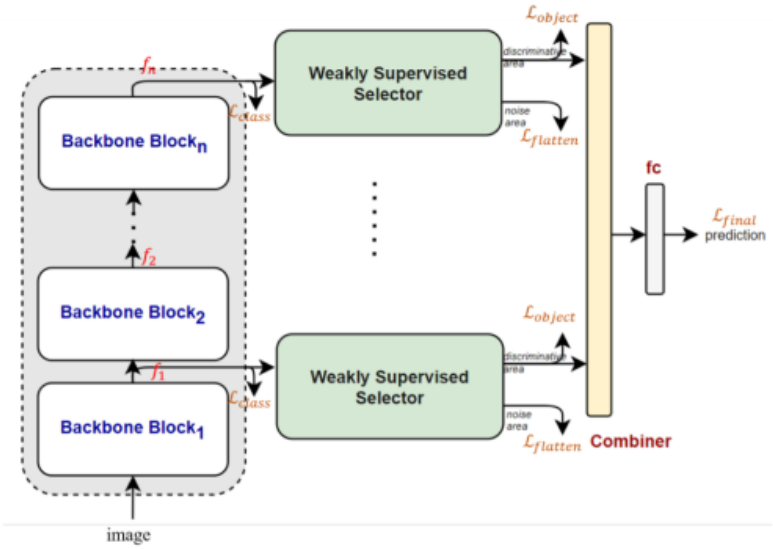


Figure 8: PIM architecture

To effectively train this architecture, we must utilize a loss function which takes into account all of the operations we perform, and not just the final prediction. In particular, we have 4 components that build the loss function: the first step is to make sure that the points of the feature maps f_i have the ability to classify. To do so, the predictions of all the feature points of a specific patch are averaged and a cross-entropy loss is applied. The resulting value is defined as L_b. Additionally, the original point in the feature map is characterized by a binary value, denoted as *Mask*, which indicates whether the said point was selected or not by the WSS. We then can calculate two loss functions, L_s for selected and L_n for dropped, which ensure that the WSS correctly identifies the object and background region. Lastly, a cross-entropy loss is applied to the predictions of the combiner, denoted ad L_c. The entire loss function is the weighted sum of those four components

$$L = \lambda_b \cdot L_b + \lambda_s \cdot L_s + \lambda_n \cdot L_n + \lambda_c \cdot L_c \tag{2}$$

During training phase , we set the values as indicated by the authors to $\lambda_b = 1$, $\lambda_s = 0$, $\lambda_n = 5$, $\lambda_c = 1$. This design of loss function ensures the module locates areas with strong

discrimination potential, and improve the prediction focusing on the points of such area. However, the result obtained experimenting with this module did not yield significant results compared to the other methods tested in our training setting (10 epochs, learning rate 0.01 and batch size of 16) as the authors advise for 50 epochs and a smaller batch size. For further information on the results of the module, please consult the appendix.

3.3.2 Cross-layer Mutual Attention Learning Network (CMAL-Net)

Dichao Liu et al.[3] bring forward the problem that, in the current state-of-the-art CNN designs used for Fine-Grained Image Classification, low-level information is more often than not neglected.
In fact, said networks normally add classifiers on top of the deepest layers within the CNNs, since deeper layers inherit the information learned by shallower layers through forward propagation and can provide better accuracy than shallower layers. However, important information for the task of FGIR is contained in the low-level information learned by the layers shallower than the deepest layer. Therefore, with CMAL-Net Liu *et al.*[3] aimed to build a model able to integrate such information into the output produced by the deepest layer of the CNN.

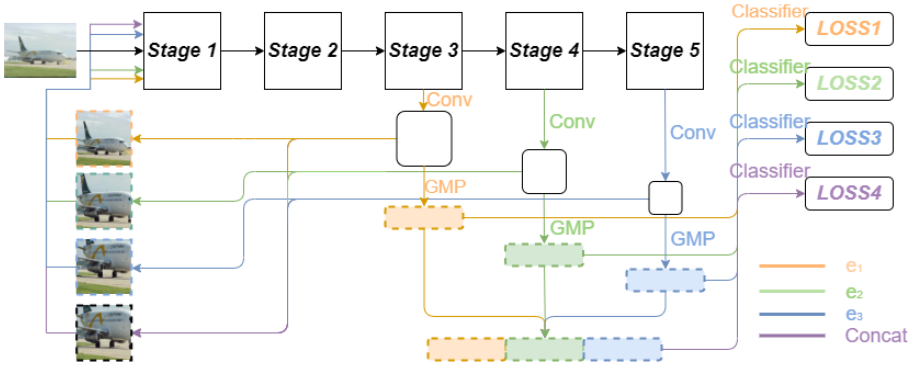


Figure 9: CMAL-Net architecture

The authors refer to the layers of a network as "experts" who have knowledge from different perspectives (low-level detailed information for shallow experts, high-level abstract information for deeper experts). From shallow to deep, each expert learns with the prior knowledge from the previous expert. While in a conventional CNN the prediction only comes from the deepest expert, CMAL-Net allows every expert to make a prediction of the categorical label and attention region. Concretely, the model starts from the deepest experts, obtaining the produced attention region, and move on to the shallower experts and, then, the fusion of all the experts. The attention regions are also passed as inputs to the shallower experts and treated as data augmentation to the original input.

CMAL-Net utilises existing pre-trained CNNs, such as ResNet50 or ResNeXt50, as a backbone. In particular, these networks are all formed by M layers $\{l_1, l_2, ..., l_M\}$, which are used as input for the training of the experts of the CMAL-Net. In fact, each expert $\{e_1, e_2, ..., e_N\}$ consists of the layers from the first layer l_1 to a certain layer of the M layers, with the deepest expert consisting of all the layers from l_1 to l_M.
Experts are trained in a progressive multi-step strategy, consisting of $N + 2$ steps:

- In the first N steps, experts are trained one by one from deep to shallow, so that they can learn the clues without being influenced by other experts;
- At step $N + 1$, all the experts and their concatenation are trained together with the overall attention region produced in the first N steps with information considered important by the collective of experts;
- At step $N + 2$, the concatenation of experts is trained on the raw input to ensure the parameters fit the resolution of the object in the original input.

CMAL-Net produces $N+1$ classifiers, each one with its own prediction score. For each image, both the raw input and the attention region are fed into CMAL to produce $2*(N+1)$ prediction scores. The final prediction is the average of those scores.

CMAL-Net is able to produce great results, reaching state-of-the-art accuracy on both the *FGVC-Aircraft* (94.7%, backbone ResNet-50) and the *StanfordCars* dataset (97.1%, backbone ResNeXt-50)[7].
However, the experts' training process can be time-consuming: training 10 epochs with the Azure Virtual Machine can take more than an hour (depending on input images' size and dataset used), compared to the 10-15 minutes used to fine-tune a simple CNN like ResNet50. In fact, CMAL-Net adds about 18 million parameters to the original backbone model, significantly impacting the output model's dimensions.

3.4 Grad-CAM

Gradient-weighted Class Activation Mapping (Grad-CAM), uses the gradients of any target concept flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept.

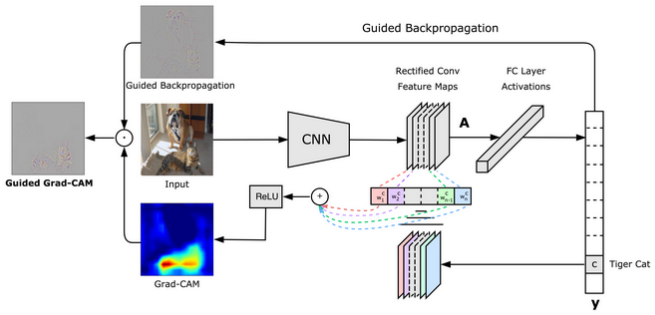


Figure 10: Grad-CAM architecture (image source)

To generate a localization map that highlights important regions for a specific class, Grad-CAM calculates the gradient of the score for the target class (y_c) with respect to the feature maps (A) of a chosen convolutional layer. These gradients are then global-average-pooled to obtain the importance weights (α_{ck}):

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y_c}{\partial A_{ij}^k}$$

The Grad-CAM heat-map is produced by computing a weighted combination of the feature maps using these importance weights and then applying a ReLU activation function to emphasize positive influences and completely discard the irrelevant ones, providing a clear, interpretable visualization of the areas in the input image that are most relevant for the prediction.

$$L^c_{\text{Grad-CAM}} = \text{ReLU}\left(\sum_k \alpha_k^c A^k\right)$$

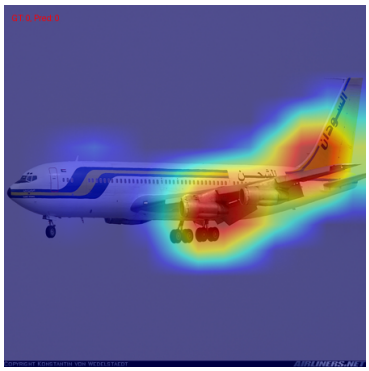


Figure 11: Grad-CAM for Resnet50 trained with SAM on an image from FGVCAircraft

4 Comparative Experiments

All models employed in our experiments were initially pre-trained on the *ImageNet1K* dataset.

All experiments have been conducted using a learning rate of 0.001 for the last layer of the model, while the other layers were trained with a learning rate of 0.0001. The momentum parameter was set to 0.9, and weight decay was configured to 0.005.
SGD optimizer has been used whenever the SAM optimizer was not employed.

Immediately below we present the plots depicting accuracy and loss on the test set. Additional graphs illustrating training accuracy and loss are available in Appendix A.

Table 1: FGVCAircraft: Top-1 Accuracy by experiment.

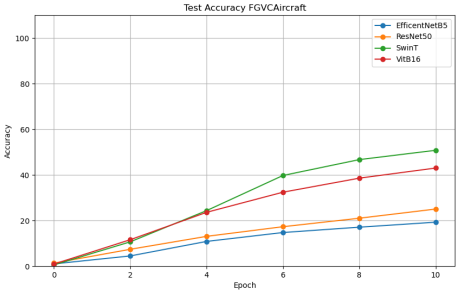
Model	Optimizer	FG plug-in	Accuracy	Parameters	Wall-Time (min)
ViT-B/16	SGD	–	43%	86M	33
ViT-B/16	SAM	–	23%	86M	51
ResNet-50	SGD	–	25%	25M	13
ResNet-50	SAM	–	65.5%	25M	39
ResNet-50	SGD	CMAL	72.7%	25M + 18M	36
ResNet-50	SGD	PIM	29.5%	25M+12M	26
ResNeXt-50	SGD	–	26.3%	25M	18
ResNeXt-50	SGD	CMAL	71.5%	25M + 18M	43
EfficientNet-B5	SGD	–	19.3%	30M	21
EfficientNet-B5	SAM	–	30.6%	30M	36
EfficientNet-B5	SGD	PIM	33.8%	30M+12M	27
Swin-t	SGD	–	50.8%	28M	18
Swin-t	SAM	–	54.8%	28M	25
Swin-t	SGD	PIM	24%	28M+12M	28

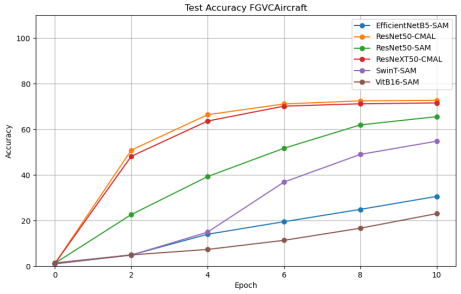
Table 2: Flowers102: Top-1 Accuracy by experiment.

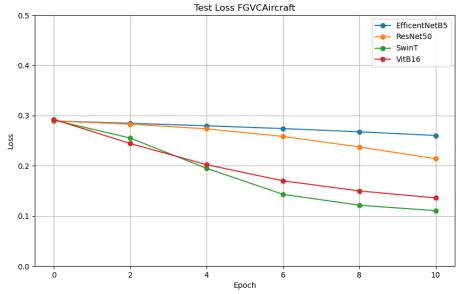
Model	Optimizer	FG plug-in	Accuracy	Parameters	Wall-Time (min)
ViT-B/16	SGD	–	92.6%	86M	42
ViT-B/16	SAM	–	93.5%	86M	84
ResNet-50	SGD	–	90.4%	25M	14
ResNet-50	SAM	–	94.3%	25M	21
ResNet-50	SGD	CMAL	97.2%	25M + 18M	53
ResNet-50	SGD	PIM	26.5%	25M+12M	24
ResNeXt-50	SGD	–	95.8%	25M	65
ResNeXt-50	SGD	CMAL	60.6%	25M + 18M	4
EfficientNet-B5	SGD	–	67.5%	30M	24
EfficientNet-B5	SAM	–	87.3%	30M	65
EfficientNet-B5	SGD	PIM	30.7%	30M+12M	28
Swin-t	SGD	–	95%	28M	16
Swin-t	SAM	–	95.7%	28M	29
Swin-t	SGD	PIM	36.3%	28M+12M	25

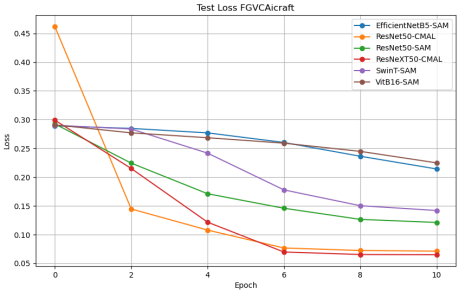
Table 3: Mammalia: Top-1 Accuracy by experiment.

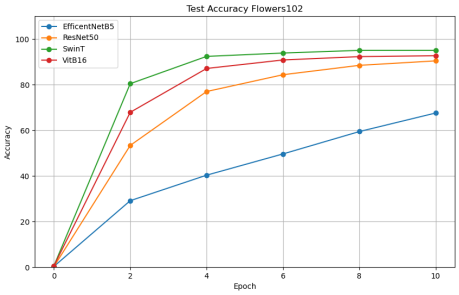
Model	Optimizer	FG plug-in	Accuracy	Parameters	Wall-Time (min)
ViT-B/16	SGD	–	53.9%	86M	55
ViT-B/16	SAM	–	52%	86M	32
ResNet-50	SGD	–	45.2%	25M	7
ResNet-50	SAM	–	45.5%	25M	12
ResNet-50	SGD	CMAL	46.9%	25M + 18M	43
ResNeXt-50	SGD	–	45.9%	25M	9
ResNeXt-50	SGD	CMAL	59.9%	25M + 18M	55
EfficientNet-B5	SGD	–	25.3%	30M	16
EfficientNet-B5	SAM	–	24.7%	30M	29
Swin-t	SGD	–	62.1%	28M	14
Swin-t	SAM	–	61.4%	28M	20

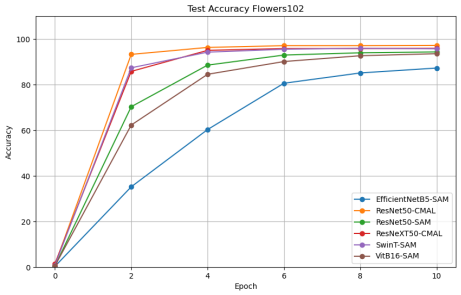


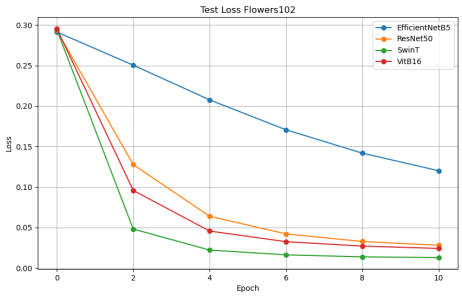


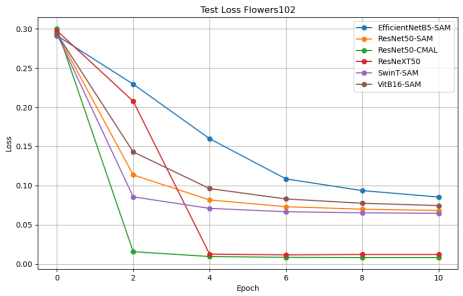


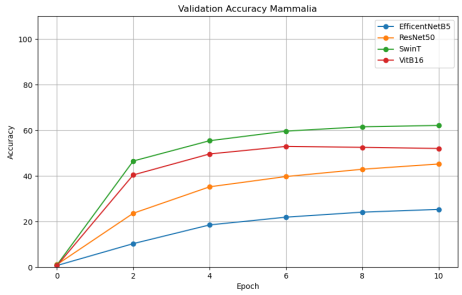


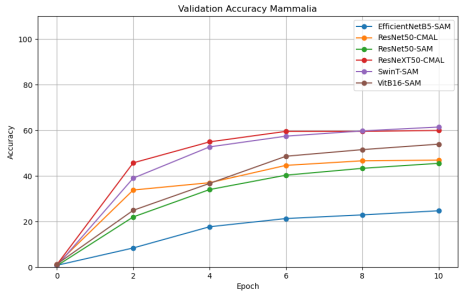


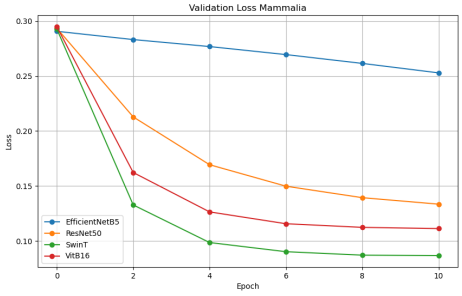


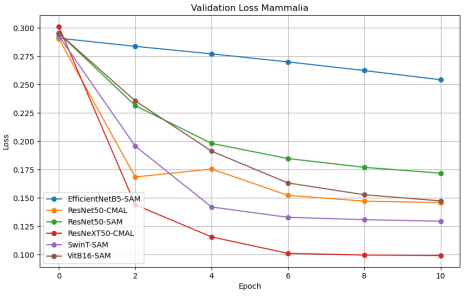

Test Loss Flowers102
EfficientNetB5
ResNet50
SwinT
VitB16
Loss
Epoch


Test Loss Flowers102
EfficientNetB5-SAM
ResNet50-SAM
ResNet50-CMAL
ResNeXT50
SwinT-SAM
VitB16-SAM
Loss
Epoch


Validation Accuracy Mammalia
EfficentNetB5
ResNet50
SwinT
VitB16
Accuracy
Epoch


Validation Accuracy Mammalia
EfficientNetB5-SAM
ResNet50-CMAL
ResNet50-SAM
ResNeXT50-CMAL
SwinT-SAM
VitB16-SAM
Accuracy
Epoch


Validation Loss Mammalia
EfficientNetB5
ResNet50
SwinT
VitB16
Loss
Epoch


Validation Loss Mammalia
EfficientNetB5-SAM
ResNet50-CMAL
ResNet50-SAM
ResNeXT50-CMAL
SwinT-SAM
VitB16-SAM
Loss
Epoch

4.1 Grad-CAM

For this part we used EfficientNet-B5 with SGD and SAM, ResNet50 with SAM, and Vision Transformer (ViT-B) with SAM. Due to limited computational resources on our Azure VM, not all combinations of models and optimizers were tested. We focused on FGVCAircraft and on classes where ResNet50 with SAM showed a significant performance margin compared to other models. Grad-CAM helped us understand the regions of the input images that the models were focusing on when making predictions. The class-wise accuracy for each model and optimizer combination is summarized as follows:

- Class 0: EfficientNet-B5 SGD (0.06) and SAM (0.51), Resnet50 SAM (0.72), ViT SAM (0.21)
- Class 99: EfficientNet-B5 SGD (0.7) and SAM (0.76), Resnet50 SAM (0.88), ViT SAM (0.97)

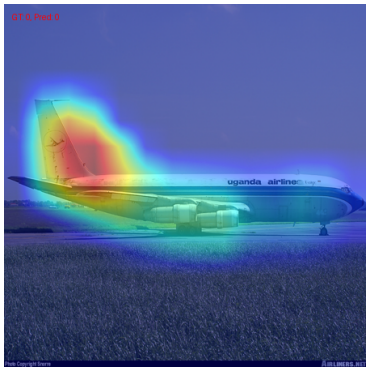


Figure 12: Resnet-SAM

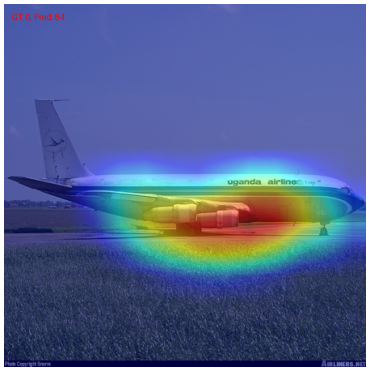


Figure 13: Effnet-SGD

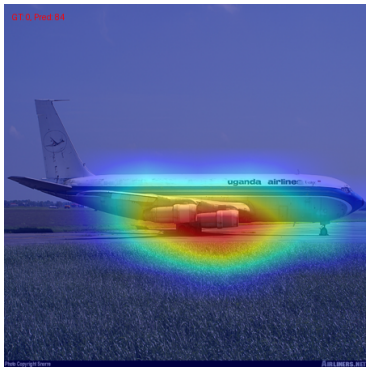


Figure 14: Effnet-SAM

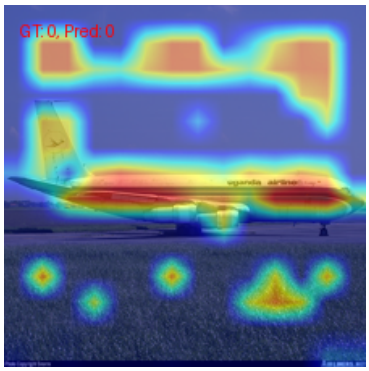


Figure 15: Vit-SAM

Following the creation of Grad-CAM visualizations for a few randomly selected images, we conducted occlusion tests. This involved omitting the parts of the image that were primarily used by the model to make predictions. The goal was to observe how the exclusion of these regions affected the model's predictions and to identify which parts of the image became relevant.

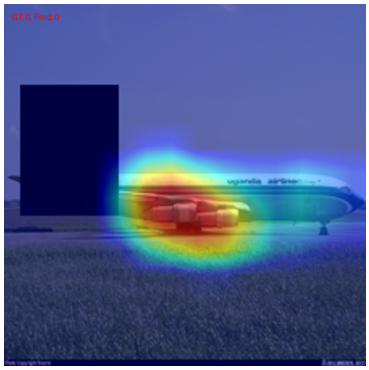


Figure 16: Resnet-SAM

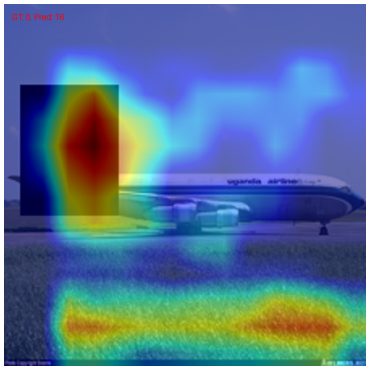


Figure 17: Effnet-SGD

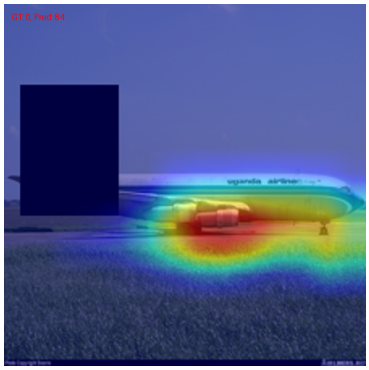


Figure 18: Effnet-SAM

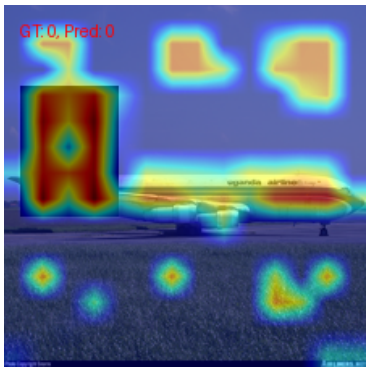


Figure 19: Vit-SAM

Although only a few images were considered, two patterns seem to emerge. First, ResNet50-SAM tends to focus on the tail of the plane. Second, when the tail is masked as in 16, ResNet is still able to make the correct prediction by focusing on other relevant parts of the planes (often the jet engines). Other examples are available in the appendix and all the examples are on our github repository.

5 Conclusion

One significant limitation was the restricted training and testing time (coupled with the unexpected instability of the Azure Virtual Machines), which made a compromise between the number of experiments and the variety of methods and datasets used necessary. Consequently, we prioritized variety by employing three datasets and multiple methods, although this limited each experiment to only 10 epochs of training. Additionally, we did not explore combinations such as SAM with PIM or SAM with CMAL. Notably, PIM did not demonstrate utility when constrained to the 10-epoch training period.

Despite these limitations, our findings reveal that SAM consistently enhances model performance across various models and datasets in fine-grained classification tasks. CMAL showed mixed results, significantly improving the performance of the backbone models on some datasets, but not always enough to justify the large increase in training time. Lastly,

our experiments confirmed that PIM is ineffective under the constraint of a 10-epoch fine-tuning period. These insights underscore the importance of selecting appropriate methods and training duration in fine-grained classification tasks within computer vision.

Grad-CAM was applied on different examples from the FGVCAircraft. Some patterns seem to emerge when comparing models performing overall well and others achieving much worse performance. It is important to bear in mind the limitations of this methodology. Specifically, if more computational resources had been available, it would have been appropriate to evaluate a much larger sample of images across more models. Nevertheless, the code and results might still be beneficial for future researches.

6 Individual Contributions

The followings are the contribution of each member in the project:

- Curci G.
  - Created utility, train, test and log functions
  - Organized the code in the repository to seamlessly integrate the different methods used
  - Explained EfficientNet, ViT, SAM and Grad-CAM
  - Integrated SAM and Grad-CAM in the code
  - Trained various models
  - Conducted occlusion experiments
- Costamagna F.
  - Integrated PIM in the repo
  - Explained Swin Transformer and PIM
  - Trained various models
  - Generated all results visualizations
- Zaccaron D.
  - Integrated CMAL in the repo
  - Explained AutoAugment, Resnet and CMAL
  - Trained various models
  - Cured Related Works

References

[1] Xiu-Shen Wei et al. *Fine-Grained Image Analysis with Deep Learning: A Survey*. 2021. arXiv: 2111.06119 [cs.CV].

[2] Pierre Foret et al. *Sharpness-Aware Minimization for Efficiently Improving Generalization*. 2021. arXiv: 2010.01412 [cs.LG].

[3] Dichao Liu et al. "Learn from each other to Classify better: Cross-layer mutual attention learning for fine-grained visual classification". In: *Pattern Recognition* 140 (2023), p. 109550. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2023.109550. URL: https://www.sciencedirect.com/science/article/pii/S0031320323002509.

[4] Po-Yung Chou, Cheng-Hung Lin, and Wen-Chung Kao. *A Novel Plug-in Module for Fine-Grained Visual Classification*. 2022. arXiv: 2202.03822 [cs.CV].

[5] Ramprasaath R. Selvaraju et al. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization". In: *International Journal of Computer Vision* 128.2 (Oct. 2019), pp. 336–359. ISSN: 1573-1405. DOI: 10.1007/s11263-019-01228-7. URL: http://dx.doi.org/10.1007/s11263-019-01228-7.

[6] Xiu-Shen Wei et al. "Fine-grained image analysis with deep learning: A survey". In: *IEEE transactions on pattern analysis and machine intelligence* 44.12 (2021), pp. 8927–8948.

[7] *Fine-Grained Image Classification Benchmarks*. URL: https://paperswithcode.com/task/fine-grained-image-classification.

[8] Usman Ali et al. "Fine-Grained Image Recognition by Means of Integrating Transformer Encoder Blocks in a Robust Single-Stage Object Detector". In: *Applied Sciences* 13.13 (2023), p. 7589.

[9] *CUB-200-2011 Dataset*. URL: http://www.vision.caltech.edu/datasets/cub_200_2011/.

[10] *FGVC-Aircraft Dataset*. URL: https://www.robots.ox.ac.uk/~vgg/data/fgvc-aircraft/.

[11] *StanfordCars Dataset*. URL: https://datasets.activeloop.ai/docs/ml/datasets/stanford-cars-dataset/.

[12] *Flowers102 Dataset*. URL: https://www.robots.ox.ac.uk/~vgg/data/flowers/102/.

[13] Ekin D Cubuk et al. "Autoaugment: Learning augmentation policies from data". In: *arXiv preprint arXiv:1805.09501* (2018).

[14] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[15] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: 1905.11946 [cs.LG].

[16] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV].

[17] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].

[18] Ze Liu et al. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. arXiv: 2103.14030 [cs.CV].

[19] Tsung-Yi Lin et al. *Feature Pyramid Networks for Object Detection*. 2017. arXiv: 1612.03144 [cs.CV].

A Training and Testing Accuracy

A.1 Training accuracy

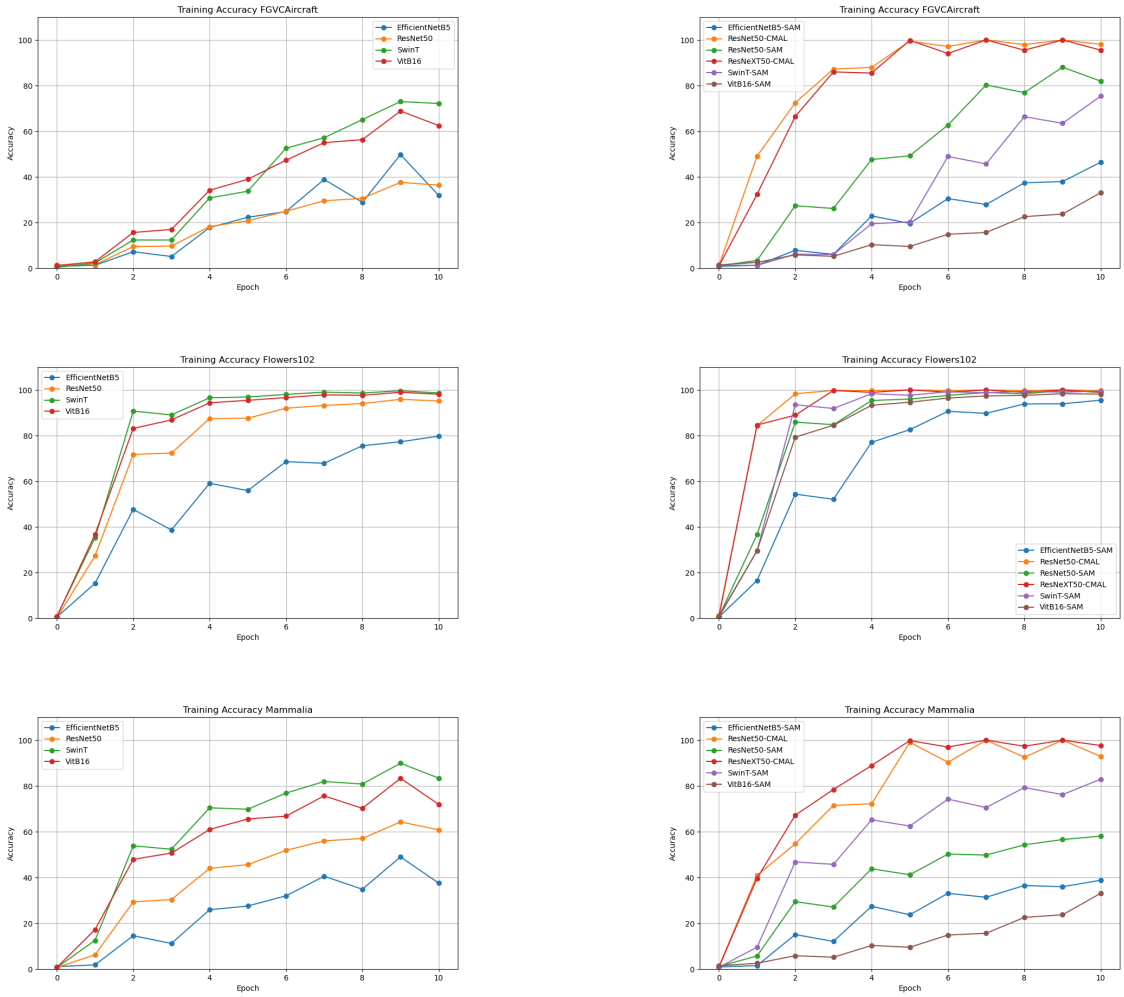


Figure 20: Training accuracy value of the conduced experiments.

A.2 Effect of variation on ResNet50

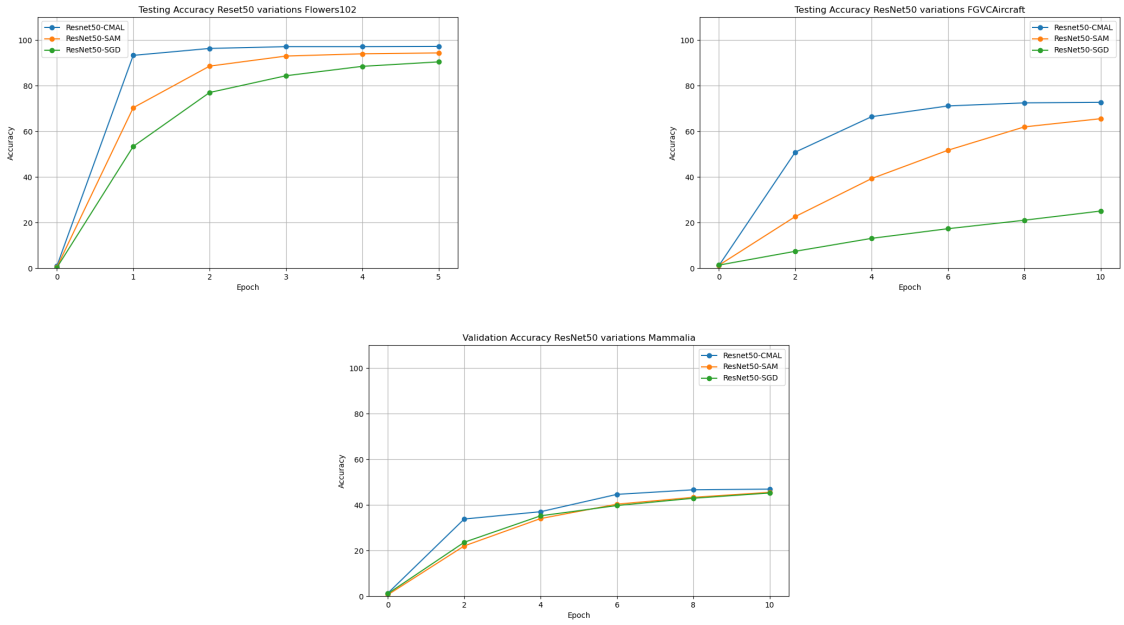


Figure 21: Comparison of the performances on the test sets by the various flavours of ResNet-50 trained.

A.3 PIM Test Result

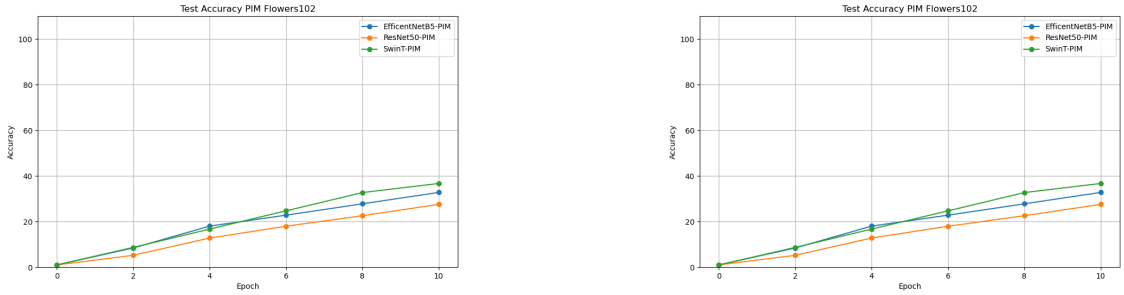


Figure 22: The training of PIM is much more slower and steady compared to the other models. In our training setting, the test accuracy is well below the other examples, as the authors suggest a 50 epoch training.

B Grad-CAM Examples

These are some of the tested examples with Grad-CAM. The others can all be found in the github repo.

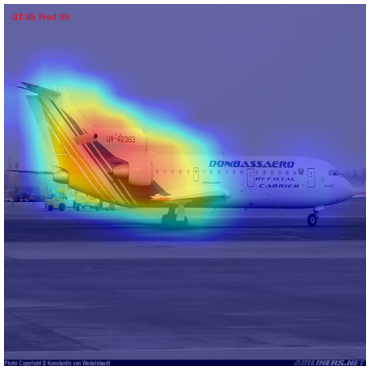


Figure 23: Resnet-SAM

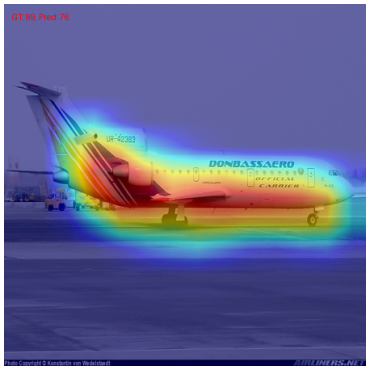


Figure 24: Effnet-SAM

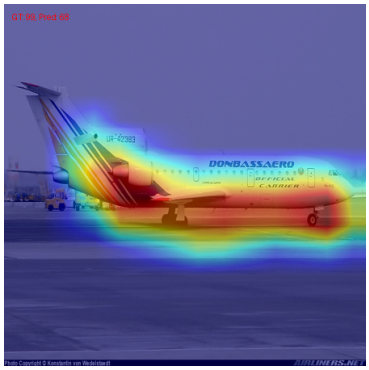


Figure 25: Effnet-SGD

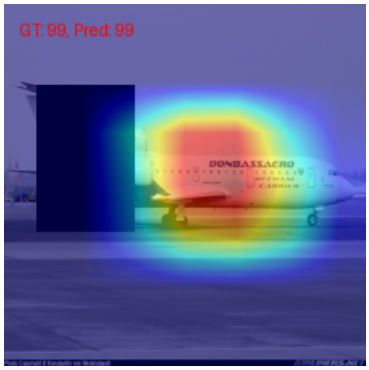


Figure 26: Resnet-SAM

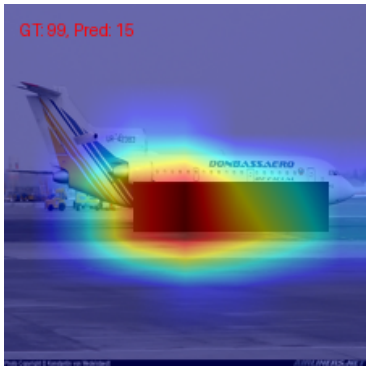


Figure 27: Effnet-SAM

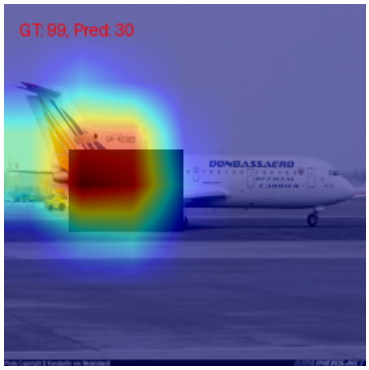


Figure 28: Effnet-SGD

Figure 29: Resnet-SAM

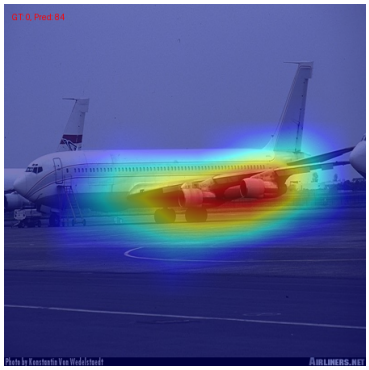


Figure 30: Effnet-SAM

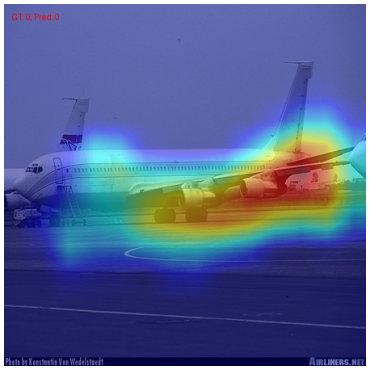


Figure 31: Effnet-SGD

Figure 32: Resnet-SAM

Figure 33: Effnet-SAM

Figure 34: Effnet-SGD

Figure 35: Resnet-SAM

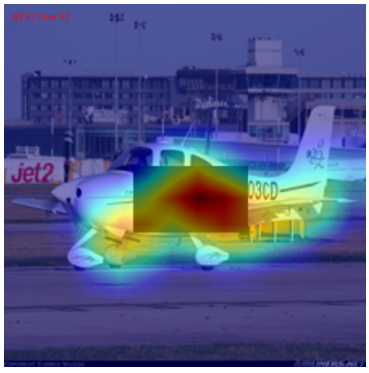


Figure 36: Effnet-SAM

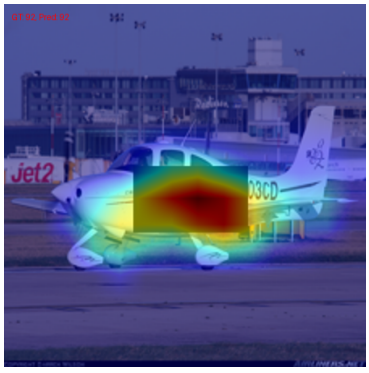


Figure 37: Effnet-SGD