University of Trento

Fine-Grained Image Classification Project

on June 5, 2024

Team: Duck Duck Goose

Hafiz Muhammad Ahmed
Yishak Tadele Nigatu
Alberto Gabriele Scuderi
Julius Heiko Schmidt

Instructors

Paolo Rota
Yiming Wang
Benedetta Liberatori

Department of Sociology

Contents

1 Introduction **1**

2 Related Work **3**

3 Proposed Solutions **5**
3.1 Architecture of Song's Approach 5
3.2 Architecture of Zhang's Approach 7
3.3 Data Loader 8
3.4 Evaluation 9
3.5 Modifications 9
3.5.1 Drop-out 9
3.5.2 Transfer Learning 9

4 Comparative Experiments **10**
4.1 Model 1 10
4.1.1 Training phase 10
4.1.2 Testing Phase 12
4.2 Model 2 12
4.2.1 Training Phase 12
4.2.2 Testing Phase 14
4.3 Comparisons between the models 14

5 Results **15**

6 Contribution **16**

Bibliography **17**

Chapter 1

Introduction

Fine-grained image classification is a challenging task in computer vision that involves distinguishing between very similar categories within a broader class. Unlike general image classification, where the goal is to categorize images into high-level categories (e.g., dogs vs. cats), fine-grained classification requires identifying subtle differences between subcategories (e.g., different breeds of birds).

The motivation behind tackling fine-grained image classification stems from its significant potential is to enhance various real-world applications. In wildlife monitoring, accurately classifying species can aid in conservation efforts by providing precise data on biodiversity. Fine-grained classification can improve diagnostic accuracy in the medical field by distinguishing between similar-looking observation samples. Despite its importance, fine-grained classification remains challenging due to the high intra-class variability and low inter-class variance, making it an intriguing problem for further research and development. [6]

Our group project focused on adjusting two robust and efficient fine-grained image classification systems. The main contributions of our work are as follows:

Data Collection:
We opted for comprehensive datasets of images not only for our target category. We have experimented with End-to-End-Models and therefore have chosen no box or part annotation datasets. With different datasets, we were able to use techniques like transfer learning.

Model Development:
We adjusted existing models leveraging state-of-the-art convolutional neural networks (CNNs) like ResNet-36 or -50 [1]. We implemented the drop-out method for the first model and used transfer learning for the second model respectively. This technique allowed us to adapt the pre-trained models to focus specifically on the task of distinguishing fine-grained categories and enhance their performance.

The entire code base for both adjusted models can be found on GitHub:
https://github.com/unitn-machine-learning/fine-grained-image_classification

Chapter 2

Related Work

To introduce ourselves to the topic we considered the work of Wei et. al. [6] as a proper starting point. The paper gives a comprehensive overview of the current state as well as current techniques to address the issue of fine-grained image classification. The paper clusters image classification (or image recognition respectively) CNNs into two main categories with more subcategories: fine-grained recognition by localization-classification subnetworks (employing detection or segmentation techniques, utilizing deep filters, attention mechanisms, and others) and fine-grained recognition by end-to-end feature encoding (with high-order feature interactions, specific loss functions, and others). Following the paper's results and the knowledge that the test dataset in class will have no bounding box or part annotations we looked further to find options to achieve similar results without these annotations.

In exploring fine-grained image classification, the methods introduced by Song et al. [4] and Zhang et al. [7] present innovative approaches to handling the subtleties of this task. Song and his colleagues propose a Feature Boosting and Suppression Module (FBSM) and a Feature Diversification Module (FDM). The FBSM enhances crucial features while suppressing background noise, which is essential for accurately distinguishing between similar categories. Additionally, the FDM improves part-specific features by comparing complementary part-specific representations from other mined information, allowing for a more nuanced capture of unique aspects. The ability of these modules to integrate with traditional end-to-end CNNs makes them versatile and easily adaptable in existing frameworks. However, the complexity of integration and dependency on additional information for the FDM might limit its applicability in certain scenarios.

Similarly, Zhang et al. present an Attention Object Location Module (AOLM) and an Attention Part Proposal Module (APPM) within a Multibranch Multiscale Learning Network (MMAL-Net). The AOLM autonomously locates the main object in the image, focusing the classification process on the relevant object without manual annotation. The APPM identifies informative parts of the object, which is particularly useful for fine-grained classification tasks. The multi-branch and multi-scale will first crop the image to the size that fits the projection of the AOLM and secondly divide and crop the image further to fit the proposal of the APPM. In the end the parts will go through a CNN for the third time to classify the image finally. This approach allows the network to handle different image scales effectively, focusing on critical details. Despite these advantages, the increased computational load from multiple image crops and the risk of overfitting due to deep focus on specific parts can be challenging. Furthermore, the complex architecture of the integration of multiple modules and branches may complicate the training and optimization processes.

For training and testing purposes, we used the following state-of-the-art benchmark datasets:

Dataset	**Object**	**# Classes**	**# Train Images**	**# Test Images**
CUB-200-2011 [5]	Birds	200	5994	5794
FGVC-Aircraft [3]	Aircrafts	100	6667	3333

Table 2.1: Dataset details

Chapter 3

Proposed Solutions

Both papers provided their code in a GitHub-Repository. We have transferred the two models in our own repository to work on them. We decided to use most of the provided code base and modified it to fit our requirements. We designed our own data loader for the provided dataset and did some slight debugging so that the model can be used for transfer learning because we wanted to train one model on the two datasets equally proposed in 2. Both papers provide a promising state-of-the-art solution to fine-grained image classification. Therefore we want to explain the architecture of both models as well as their modules and approaches in more detail. Starting with the first model from Wei and continuing with the model from Zhang.

3.1 Architecture of Song's Approach

The entire CNN uses ResNet-36 [1] as its foundation. The FBSM is integrated after the third, fourth, and fifth stage of the feature extractor of ResNet-36. The par-specific representations are then fed into the FDM where they will get diversified.

The FBSM works by splitting the feature maps into equal parts and evaluating the importance of each feature map after a 1x1 convolution. After scoring the importance a global average pooling and SoftMax layer will reduce the scoring maps to normalized importance factors. Therefore the important factors can be used to boost the most salient parts of the image through element-wise multiplication. The suppression factors can be obtained by subtracting the most salient importance factors from the feature maps. The

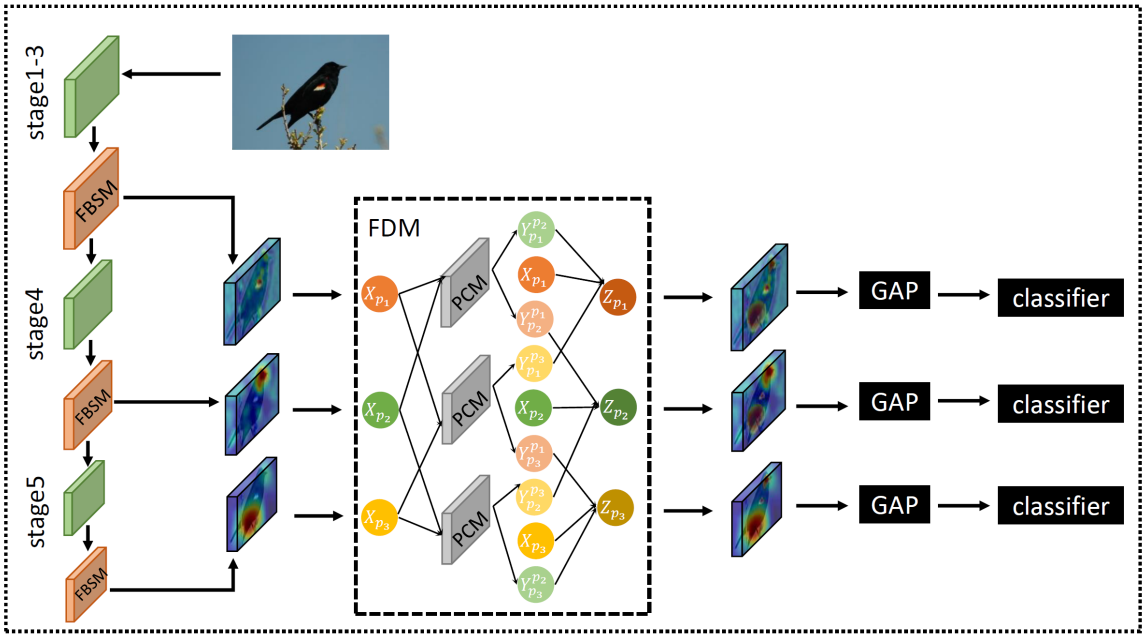
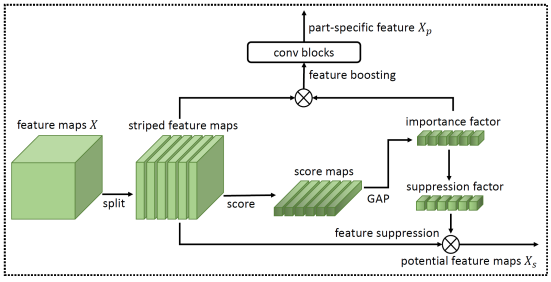


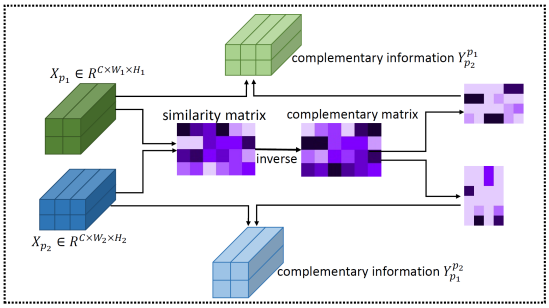
Figure 3.1: Overview of the framework [4]

FBSM outputs feature maps with highlighted parts and feature maps where the otherwise highlighted part is suppressed. This can be seen in 3.2a.

The obtained feature maps X_p and X_s will be fed into the FDM. Two different part-specific feature maps will be compared pixel-wise to find similarities. After applying a SoftMax row- and column-wise to the similarity matrix, the complementary matrix can be calculated. By multiplying the original X_p with the corresponding Y_p matrices, the complementary information is embedded in the new $Y_{p_1}^{p_2}$ or $Y_{p_2}^{p_1}$ respectively. The framework will use the part-specific feature map and the complementary information to create the part-specific feature Y_{p_i} and classify it subsequently. The diagram in 3.2b as well as the overview in 3.1 display this in a graphical form.



(a) Architecture of the FBS-Module



(b) Diagram of the PDM

Figure 3.2: Overview diagrams of FBS-Module and PDM [4]

3.2 Architecture of Zhang's Approach

The MMAL-Net is built on top of a ResNet-50 [1] backbone. The schematic structure can be seen in 3.3. The network consists of three branches: a raw, an object, and a part branch. The image will go through the same CNN and fully connected layer (FC) three times while the feature maps will be used to modify the input of the next branch. All branches use cross-entropy loss as the classification loss. Although not correctly shown in the picture every branch uses a SoftMax output layer at the end.

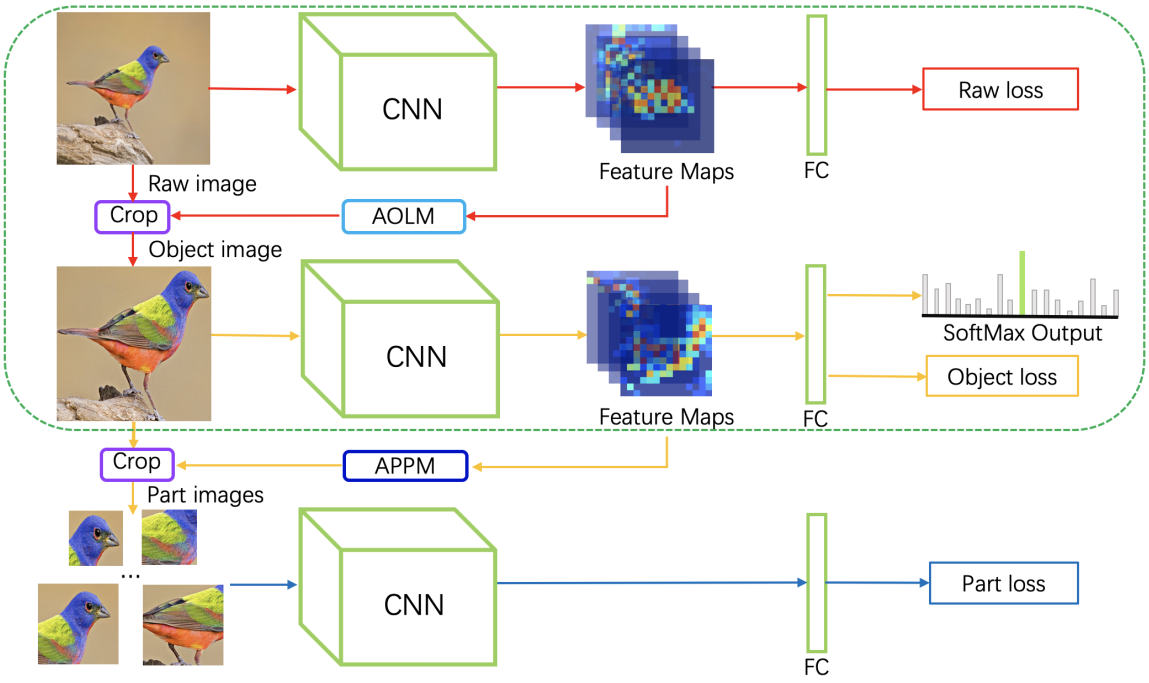


Figure 3.3: Overview of the MMAL-Net (picture on Github/Read.me) [7]

The AOLM will use the different feature maps from the convolutional layers. By aggregating the feature maps the AOLM will create an activation map. From the obtained activation map a coarse bound-box like rectangle can be calculated to localize and mask the object. This sequence is visualized in 3.4a.

(a) Pipeline of the AOLM

(b) Pipeline of the APPM

Figure 3.4: Overview diagrams of FBS-Module and PDM [7]

The APPM uses the sliding windows approach for object detection. By moving a window of a fixed size over each feature map the window will calculate the potential importance of an image region. Afterwards, the mean attention of each window through the different feature maps is calculated. To reduce the risk of redundancy the APPM will choose a fixed number of the highest-scoring windows to crop the image into parts. This is visualized in 3.4b.

3.3 Data Loader

To adapt to different datasets we needed to modify the data loader. Especially for the competition we needed to be able to ingest data from the following data structure:

```
$ tree competition_dataset
# competition_dataset/
#    train/
#        classID_classNAME/
#        ...
#    test/
#        imageID.jpg
#        ...
```

Figure 3.5: Structure of the competition dataset [2]

In the actual competition, we had an error coming up that pointed out that the CUDA device was not able to allocate enough memory. This issue was not easily addressed because the root cause was not directly pointed out by the code editors we used. After a lot of debugging the origin was found:

The issue originated from the data loader, which was designed to interpret the first segment of each folder's name as a class ID. However, the folders within the training directory contained class IDs that were not consistent with the predetermined class size. Specifically, there were instances where class IDs, such as 4780, exceeded the maximum class size of 100. This inconsistency was not identified during the challenge session because the system did not produce a descriptive error message.

We resolved this issue by creating a *class_labels.txt* file. This file established a mapping from class IDs ranging from 0 to 99 to the corresponding scripted class IDs in the directory.

This solution allowed the data loader to function correctly by ensuring that class IDs were within the expected range.

3.4 Evaluation

To evaluate the models we integrated *WandB*. We monitored different metrics like training loss, training accuracy, and testing accuracy throughout the epochs.

3.5 Modifications

3.5.1 Drop-out

For *Model 1* (Song's Approach) we used the drop-out method. The drop-out method is a regularization technique designed to prevent overfitting. It works by randomly "dropping out" (setting to zero) a fraction of the neurons during training in each iteration. This means that each iteration of training updates a different subset of neurons, which helps the network to learn more robust features that generalize better to unseen data. Dropout helps to make the CNN less sensitive to the specific weights of individual neurons, thereby improving the model's ability to generalize to new data.

3.5.2 Transfer Learning

For *Model 2* (Zhang's Approach) we used transfer learning. Transfer learning is a machine learning technique where a pre-trained model developed for a task is reused as the starting point for a model on a second task. It involves leveraging a pre-trained model's learned features and adapting them to a new, often related, task. This method is highly effective for improving model performance and efficiency, particularly when dealing with limited data.

Chapter 4

Comparative Experiments

4.1 Model 1

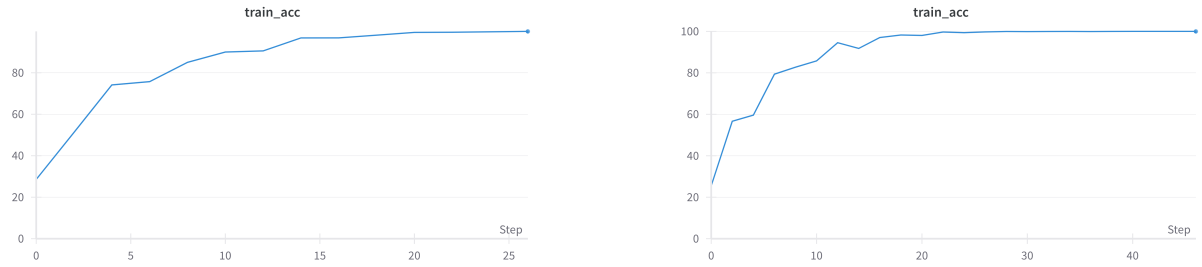
4.1.1 Training phase

We have monitored *Model 1* throughout its training steps. The training was done on the *competition challenge* dataset. The graph in 4.1a shows on the y-axis the accuracy percentage from 0 % to 100 % and on the x-axis represents the training steps. Initially, the training accuracy starts from around 30 % and reaches a stable 95+ % after about 15 epochs. With the drop-out method used the accuracy was not increasing as steadily as without drop-out but achieved the same accuracy early and a higher one overall, as can be seen in 4.1b. The high training accuracy suggests that the model could be overfitted.

The following hyperparameters were used for training Model 1 (Song's Approach):

Parameter	Value	Description
alpha	0.5	Weight for FBSM loss
beta	0.5	Weight for FDM loss
gamma	1	Weight for classification loss
kind	dog	Dataset category
bs	20	Batch size
epoch	200	Number of training epochs
arch	resnet50	Backbone architecture (ResNet-50)

Table 4.1: Hyperparameters of Model 1



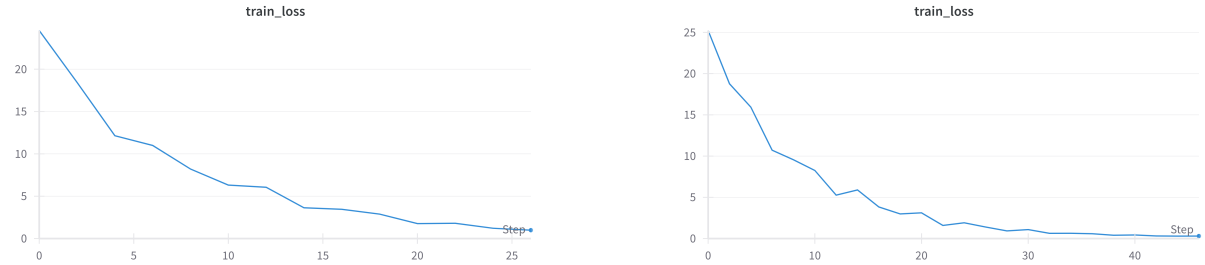
(a) without drop-out **(b)** with drop-out

Figure 4.1: Training accuracy of Model 1

The graph in 4.2a shows that training loss is steadily decreasing close to zero. This is expected as the model learns.

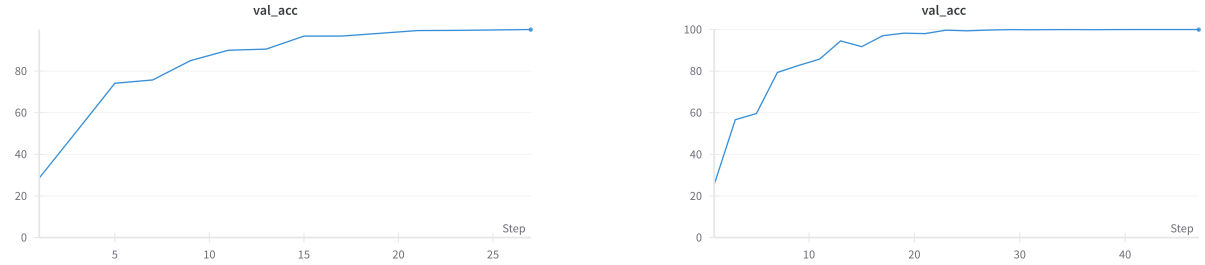
The graph in 4.3a shows a rapid increase in validation accuracy over the training process. The early steps show noticeable fluctuations in 4.3b as the model fine-tunes its parameters and copes with the eliminated neurons. Both graphs show a very high, close-to-perfect validation accuracy which is an indicator that the model is overfitted. This can be confirmed in 4.1.2.

The gap between training and validation accuracy, along with the plateauing validation accuracy, strongly suggests that the model is not generalizing well to unseen data. The model achieves a nearly perfect level of accuracy and minimal loss which suggests that the model has learned the training data very well. Fluctuations in validation accuracy suggest instability in generalizing with new unseen data.



(a) without drop-out **(b)** with drop-out

Figure 4.2: Training loss of Model 1



(a) without drop-out **(b)** with drop-out

Figure 4.3: Validation accuracy of Model 1

4.1.2 Testing Phase

After sending our model to the provided server to test its performance we achieved an **test accuracy of 71 %**. The accuracy is not on par with current state-of-the-art models but is still good compared to the other teams. The "low" accuracy on the testing set and the high accuracy in training and validation suggests that *Model 1* is overfitted. This issue should be tackled in future work.

4.2 Model 2

4.2.1 Training Phase

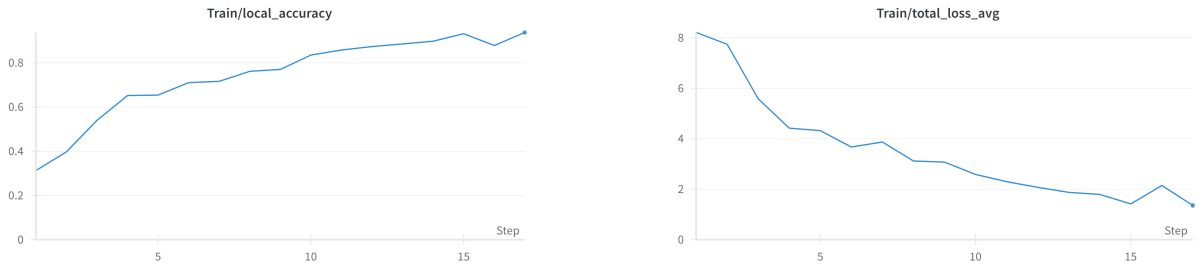
When testing *Model 2*, we can observe that, overall, it exhibits a lower accuracy on test data compared to *Model 1* accuracy rate, which suggests that it presents a better generalization. Decreasing loss values indicates an improved performance on test data as training progresses. From the graph in 4.5b, we have a hint at how bad Model 2

generalises unseen data. Accuracy on test data tends to fluctuate but generally trends upwards, indicating that the model is improving its performance on every training step. The test accuracy is lower compared to *Model 1* since it should perform better on training data. Similar to the local loss, the total average loss on the test dataset starts high and then gradually decreases close to a plateau with some fluctuation with the increase of the training steps.

For training Model 2 (Zhang's Approach), the following hyperparameters were used:

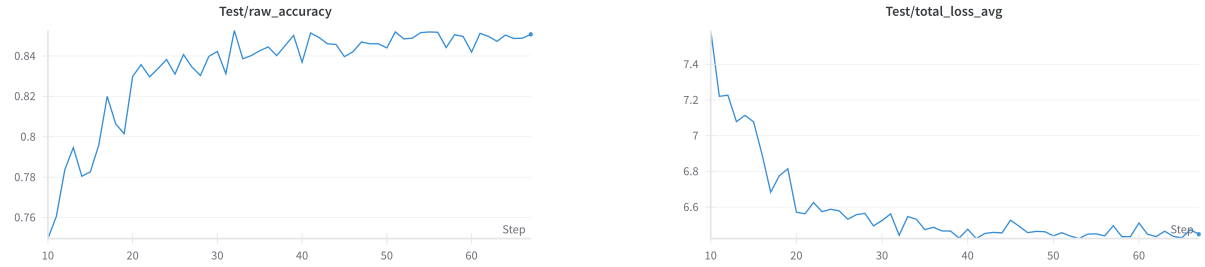
Parameter	**Value**	**Description**
batch_size	6	
vis_num	batch_size	Number of visualized images in TensorBoard
eval_trainset	False	Whether or not to evaluate on the training set
save_interval	1	Interval (in epochs) for saving checkpoint
max_checkpoint_num	200	Maximum number of checkpoints to keep
end_epoch	200	Total number of training epochs
init_lr	0.001	Initial learning rate
lr_milestones	[60, 100]	Epochs for learning rate decay
lr_decay_rate	0.1	Learning rate decay factor
weight_decay	1e-4	Weight decay regularization
stride	32	Stride for sliding window in APPM
channels	2048	Number of channels for feature maps
input_size	448	Input image size

Table 4.2: Hyperparameters of Model 2



(a) accuracy **(b)** loss rate

Figure 4.4: Training evaluation on the competition dataset.



(a) accuracy **(b)** loss rate

Figure 4.5: Validation evaluation on the test data of the CUB-200-2011 dataset

4.2.2 Testing Phase

Because of a downtime of the test server, we couldn't test the performance in its current state. We reckon that it won't perform better, because of results in the early stages that only achieved values around 66 % after epoch number 14.

4.3 Comparisons between the models

Model 1 shows a better learning curve. Accuracy and loss during training are not rapidly changing but steadily increasing or decreasing respectively. While probably being overfitted it still accomplished a good accuracy on the test set. Although we haven't tracked the time per epoch, this model has shown better performance regarding computational load. Beating the time per epoch by almost half in regards to *Model 2*.

Model 2 shows a more rugged response to the testing of patterns on the *CUB-200-2011* dataset. Reasons for that could be a small batch size. It also shows that the MMAL-Net has issues adapting to new images. Perhaps this originated in the transfer learning process because the model was pre-trained on the *FGVC-Aircraft* dataset beforehand. But on the other side, the same model performed more smoothly on the training on the *competition challenge* dataset.

All in all, we would prefer *Model 1* because of better results and performance benefits. Also, it is a bit easier to understand and implement into other CNNs.

Chapter 5

Results

During the training phase, both models exhibited promising performance on benchmark datasets, such as CUB-200-2011 and FGVC-Aircraft. *Model 1* incorporated modules to enhance crucial features, suppress background noise and diversify part-specific representations. *Model 2* adopted a region-based approach, generating multi-scale proposal windows and employing attention modules to locate objects and identify informative parts without manual annotations. Both models demonstrated promising performance on benchmark datasets but *Model 1* would be recommended.

However, limitations were identified, including increased model complexity, overfitting and generalization issues and computational overhead. Future work could focus on exploring new techniques to improve efficient architectures and explore alternative loss functions or regularization methods to enhance generalization capabilities.

Chapter 6

Contribution

In the table we have listed the work everyone has done during the project:

Name	**Contribution**
Hafiz Muhammad Nigatu	Adjusting and training Model 1
Yishak Tadele Nigatu	Code Organization, Adjustment of the models, Training and evaluation of both models
Alberto Gabriele Scuderi	Writing of the Report
Julius Heiko Schmidt	Training Model 2, Writing of the Report

Table 6.1: Dataset details

Bibliography

[1] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[2] Benedetta Liberatori. *dataset structure for the competition*. 2024.

[3] Subhransu Maji et al. *Fine-Grained Visual Classification of Aircraft*. 2013. arXiv: 1306.5151 [cs.CV].

[4] Jianwei Song and Ruoyu Yang. *Feature Boosting, Suppression, and Diversification for Fine-Grained Visual Classification*. 2021. arXiv: 2103.02782 [cs.CV].

[5] Catherine Wah et al. *CUB-200-2011*. Apr. 2022. DOI: 10.22002/D1.20098.

[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 (2022), pp. 8927–8948. DOI: 10.1109/TPAMI.2021.3126648.

[7] Fan Zhang et al. *Multi-branch and Multi-scale Attention Learning for Fine-Grained Visual Categorization*. 2020. arXiv: 2003.09150 [cs.CV].