Supervised Learning for Few-Shot Orchid types Classification with Prior Guided Feature

2022 AI CUP Competition, TEAM_142

Yu-Hsi Chen, Jia-Wei Liao, Kuok-Tong Ng

Department of Applied Mathematics,

National Yang Ming Chiao Tung University,

Taiwan

June 17, 2022

Contents

List of Figures

List of Tables

Abstract

Image classification has been widely used in engineering, agriculture, and medical applications. Nowadays, with the rapid development of deep neural network and the computing power of graphic cards, the performance of image classification has been greatly improved. In particular, pattern recognition plays an important role in image classification. Even though this field looks very mature, there are still some intractable problems, such as lack of labeled data and lack of understanding of species. In this paper, we introduce a pipeline that can find the most suitable process for each task systematically. Using this method, our final scores are 0.9115 and 0.8096 in public and private datasets, respectively. In addition, our overall ranking is 15th out of 743 teams.

Chapter 1

Introduction

Orchids are the most diverse variety of the plants, with more than $20,000$ species in the world. They can be found in almost every ecological environment on Earth, except the regions with extreme climates such as polar regions or deserts. The growth and adaptation of orchids to different environments results in differences in the size and appearance of the flowers. However, with the advancement of biotechnology, more and more similar new species appear one after another under the breeding technology, which requires orchid experts to distinguish. Therefore, how to accurately distinguish orchids with similar appearance in a non-artificial way is not an easy task. In the training model phase, we have $2,190$ images in our training data for a total of 219 categories. That is, we only have 10 training images per class at the current phase. Fig. 1.1 demonstrates the part of our training dataset. Due to the lack of training data, we decided to use Few-shot learning which aims to transfer information from one task in order to generalize to a new task given a small amount of data. Thus, we expect that few-shot learning systems should learn data representations that are invariant to common factors of object variation, while still representing features that allow different classes to be distinguished.

For contrastive learning, it applied to self-supervised representation learning for recent year, learning to SOTA performance in the unsupervised training of image models. Thus, we decided to combine instance discriminative contrastive learning and supervised learning in one framework. That is, contrastive learning in a fully supervised setting allows us to effectively utilize label information. The main reason why we want to combine contrastive learning in this task is still because there is only a small amount of data in each class, and we think that we may have better results by using contrastive learning to compare the representations in each data. Furthermore, we tried to use a variety of methods in the image preprocessing part to enhance the details between different orchid species, including image processing in the frequency domain [2], and embedding in different spaces

to analyze the image. In addition, we also used methods such as clustering to analyze the prior distribution of training images, expecting to focus on training the model through the prior features.

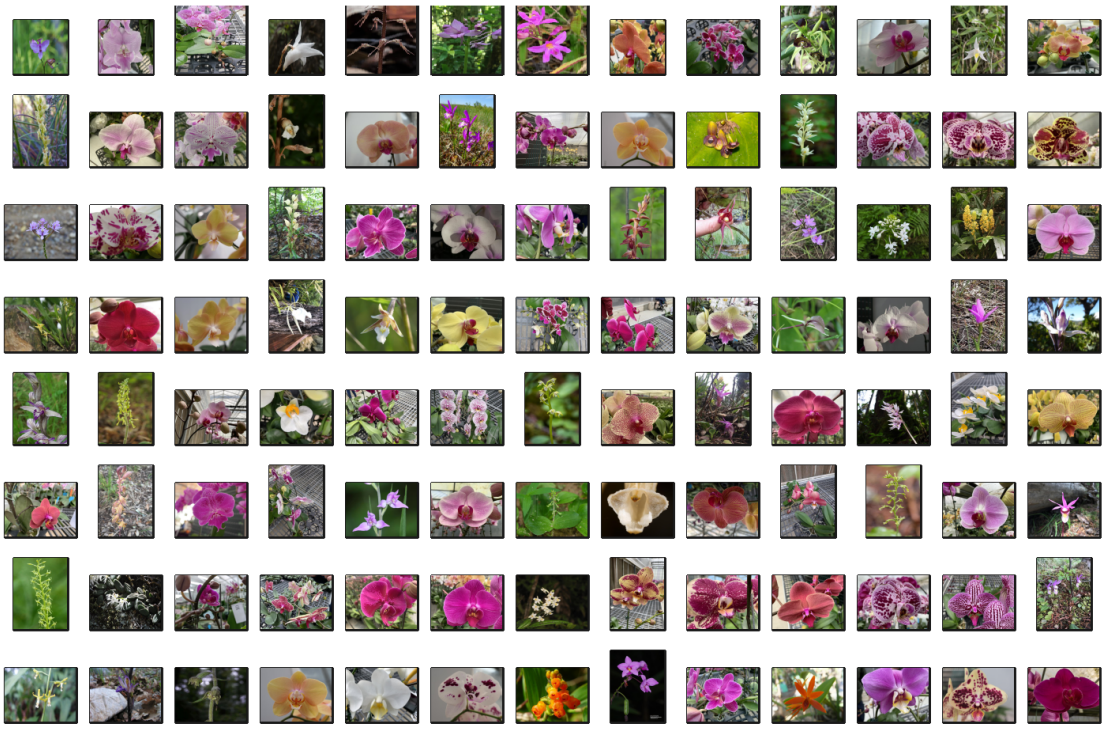

Figure 1.1: Orchid images in the training dataset.

In our experimental results, the best validation accuracy is about 91.1%, and the corresponding test accuracy is 91.55%. After using the ensemble technology, the best verification accuracy and test accuracy can be improved to 91.78% and 92.92%, respectively. We will open-source our checkpoints, training script, and configurations at: https://github.com/TW-yuhsi/ViT-Orchids-Classification and Google Drive.

Chapter 2

Related Works

2.1 Few-Shot Learning

Few-Shot Learning [21, 18, 15] can be regarded as a kind of meta-learning. In this method, since the learner is trained on several related tasks, during the meta-training phase, it can generalize well to unseen but related tasks with just few examples, during the meta-testing phase. An effective approach to the Few-Shot Learning problem is to learn a common representation for various tasks and train task specific classifiers on top of this representation.

2.2 Contrastive Learning

Contrastive learning [4, 13] is a self-supervised, task-independent deep learning technique that allows a model to learn about data, even without labels. Specifically, the concept of Contrastive loss proposed by Chopra et al. (2005) is one of the earliest training objectives used for deep metric learning in a contrastive fashion.

2.3 Fine-grained Classification

Fine-grained image classification that focuses on differentiating between hard-to-distinguish object classes and doing the detailed inference [22, 10, 24] are also highly relevant to this competition. Furthermore, there are also lots of literature related to orchid classification tasks including treat extracted feature and color as an important information for orchid types classification task [1] proposed by Pulung Nurtantio Andono et al in, Phylogeny and classification of the orchid family [9] and Features of pollinaria and orchid classification [8] proposed by RL Dressler in 1993 and 1986, respectively, etc.

Chapter 3

Proposed Method

3.1 Data Pre-processing

3.1.1 Data Splitting

First, we split our training data into 4-Fold as shown in Fig. 3.1. Then we ensemble the results to reduce the variance, so the performance of the model is less sensitive to the partitioning of the data.

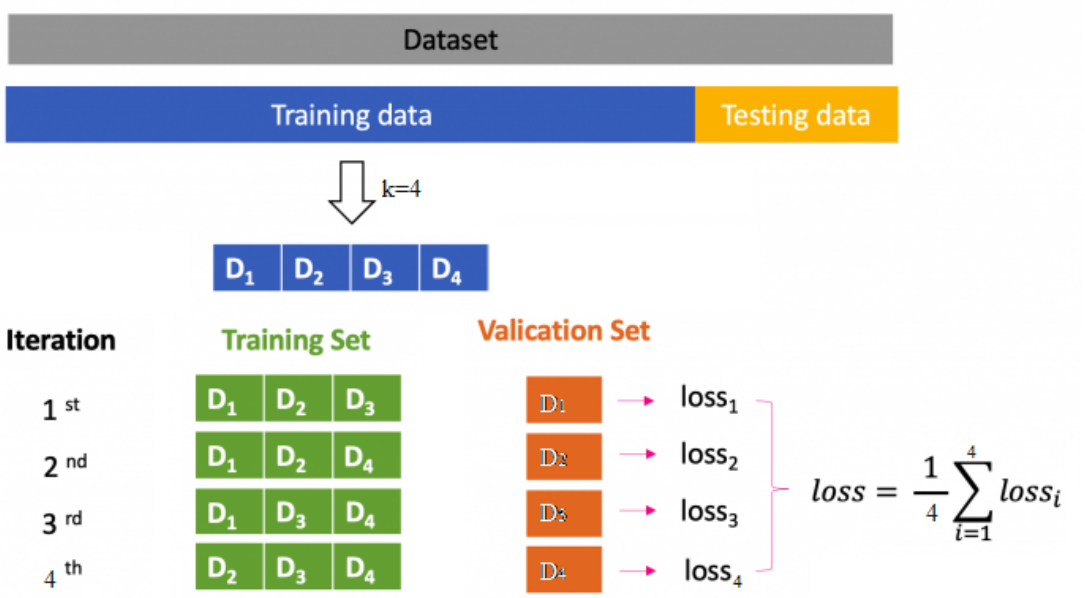

Figure 3.1: 4-Fold Cross-Validation.

3.1.2 Image Normalize

Images are represented as tensors with pixel values ranging from 0 to 255. Since the model's weights is pre-trained on ImageNet, we apply z-score transform to pixel values with mean = (0.485, 0.456, 0.406) and std = (0.229, 0.224, 0.224) the same as mean and std in ImageNet.

3.1.3 Data Augmentation

To avoid overfitting situations and enhance the robustness of the model, we use the following techniques, including

1. **Random rotate 10 degrees**
2. **Random flip with horizontal**
3. **Random add Gaussian noise:** Adding the noise with probability $I(i,j) + \sigma \cdot s$
4. **Auto-augmentation [5]:** Setting some policies consisting of rotation, sharpness, shearing, transla-tion, brightness, etc. We then trans-form the images by randomly choosing a policy.

3.2 Model Architecture

In the model part, besides ResNet family, we used ConvNext [17], EfficientNet [20], Vision Transformer (ViT) [7], Swin Transformer [16], CSWin [6], and Neighborhood Attention Transformer (NAT) [11] which are the SOTA in image classification task.

3.2.1 ConvNext

This is a pure convolutional model (ConvNet), inspired by the design of Vision Transformers, that claims to outperform them. Fig. 3.2 demonstrates that a standard ConvNet model can achieve the same level of scalability as hierarchical vision Transformers while being much simpler in design.

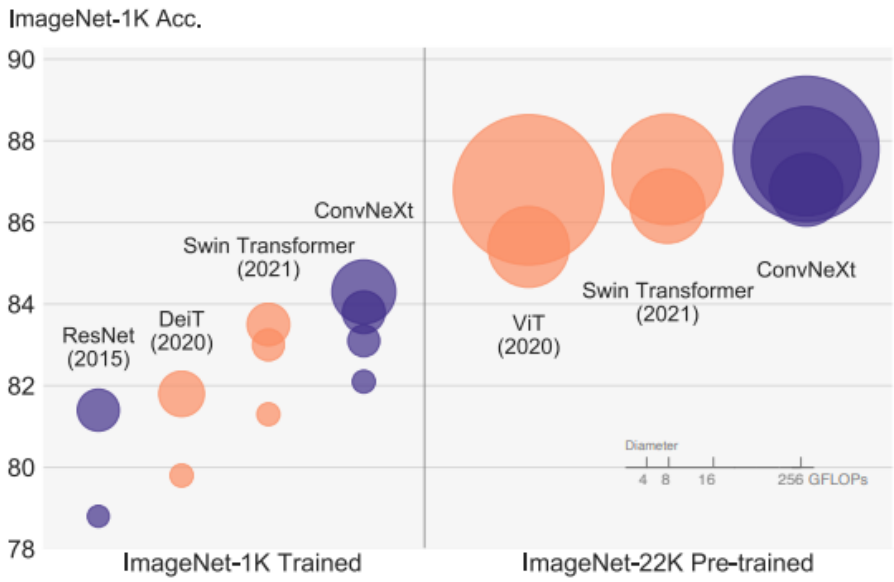

Figure 3.2: ImageNet-1K Classification.

3.2.2 EfficientNet

EfficientNet is proposed by Mingxing Tan, Quoc V. Le in 2019. Scaling up convolution networks is widely used to achieve better accuracy such as ResNet [12] scaling up the network's depth (ResNet18 to ResNet152). Wide Residual Network (WRN) [23] scaling up the network's width (increasing the number of output channels). Moreover, scaling up the image resolution also gives the better accuracy. The goal of EfficientNet is to scale up depth, width, and resolution simultaneously, as shown in Fig 3.3. The larger the input image, the deeper layer and more channels are model needs, which is to ensure that the receptive field is large enough to capture more features. And we can see EfficientNet gives a great accuracy, as shown in Fig 3.4.

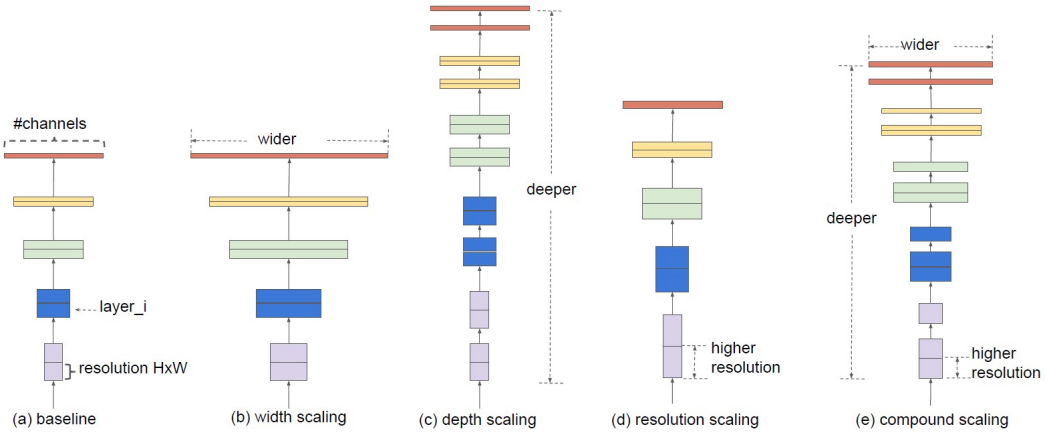

Figure 3.3: Model scaling.

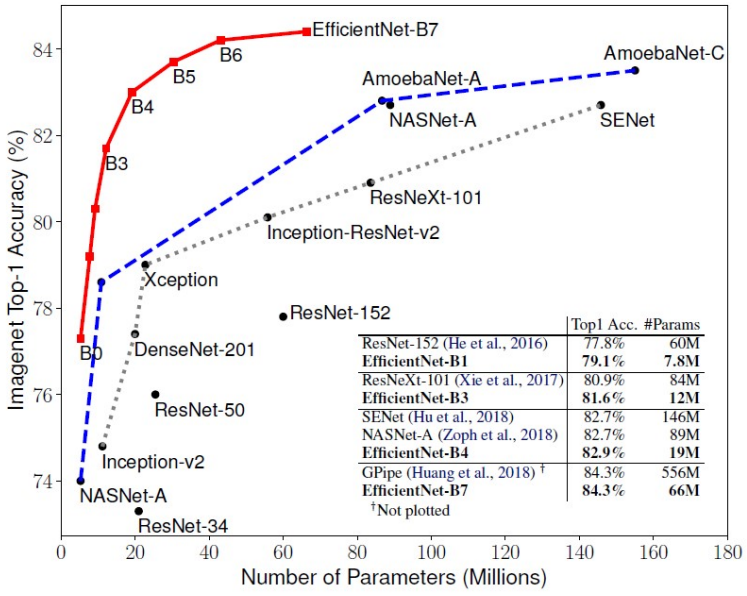

Figure 3.4: Model size versus ImageNet accuracy.

3.2.3 Vision Transformer

Vision Transformer is proposed by Alexey Dosovitskiy in 2020. It shows that reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Trans-former (ViT) attains excellent results compared to SOTA convolutional networks while requiring substantially fewer computational resources to train. Fig. 3.5 shows the overview for ViT, the procedure including split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, they use the standard approach of adding an extra learnable classification token to the sequence.

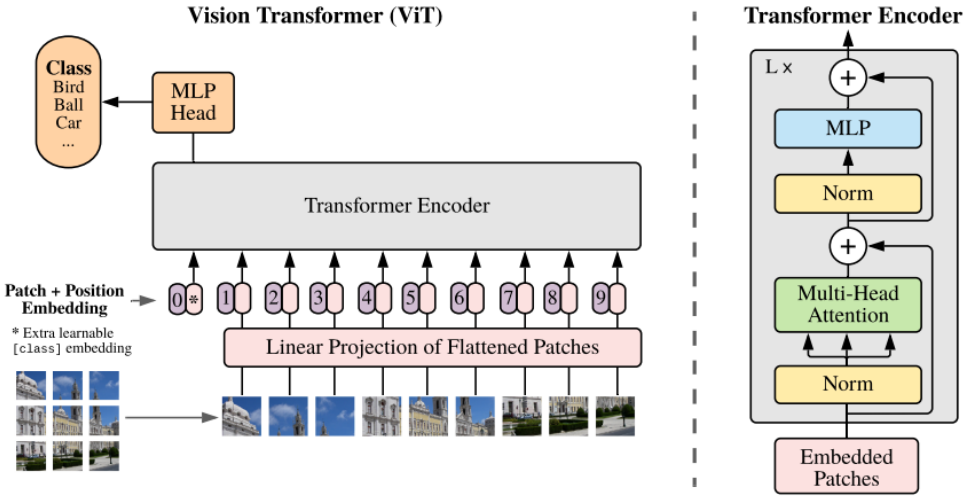

Figure 3.5: Model overview.

3.2.4 Swin Transformer

Swin Transformer is proposed by Ze Liu in 2021, which achieved the best paper award of ICCV2021. It uses the window-based multi-head self-attention (W-MSA) to re-duce linear time complexity. To reinforce the connectivity of different windows, it proposes a window shift mechanism. Figure 3 is the architecture of Swin Transformer, there are four stages in the Swin Transform-er and each stage is composed of a patch merging layer and two successive Swin Transformer blocks which are the feature extractor and maintains the output size as the input. The patch merging layer uses the inverse pixel shuffle to attain the dimension reduction. Its role is as a maximum poling in the CNN model. In Figure 3.7, we show the Swin Transformer block, which is composed of

layer normalization (LN), (shift) window-based multi-head self-attention ((S)W-MSA), and multilayer perceptron (MLP). Moreover, it also has the residual mechanism after (S)W-MSA and MLP layers.

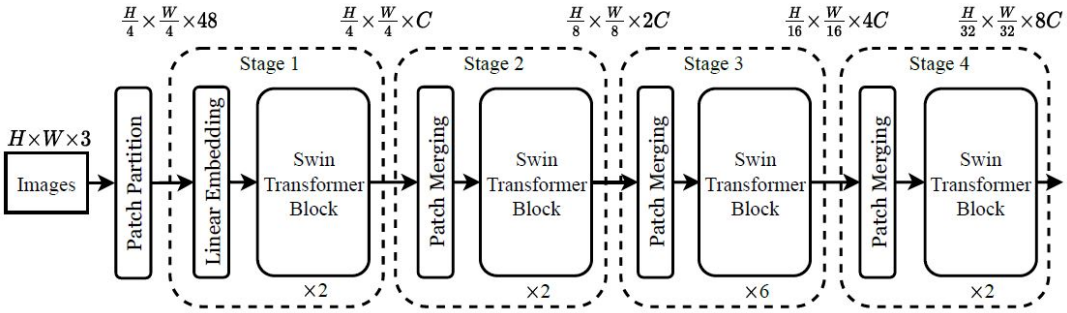

Figure 3.6: Architecture of Swin Transformer.

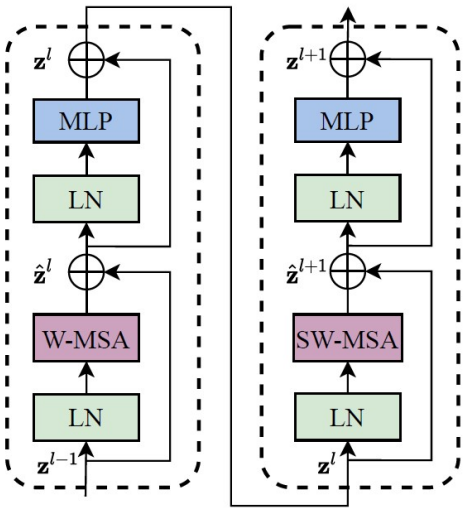

Figure 3.7: Two successive Swin Transformer block.

3.2.5 CSWin Transformer

A challenging issue in Transformer design is that global self-attention is very expensive to compute whereas local self-attention often limits the field of interactions of each token. To address this issue, authors developed the Cross-Shaped Window self-attention mechanism for computing self-attention in the horizontal and vertical stripes in parallel that form a cross-shaped window, with each stripe obtained by splitting the input feature into stripes of equal width. Fig. 3.8 demonstrates the different self-attention mechanisms. Firstly, authors split multi-heads into two groups and perform self-attention in horizontal and vertical tripe simultaneously. Secondly, they adjust the stripe width according to the depth network, which can achieve better trade-off between computation cost and capa-

bility. Fig. 3.10 shows the overall architecture of proposed CSWin Transformer (Left) and the illustration of CSWin Transformer block (Right).

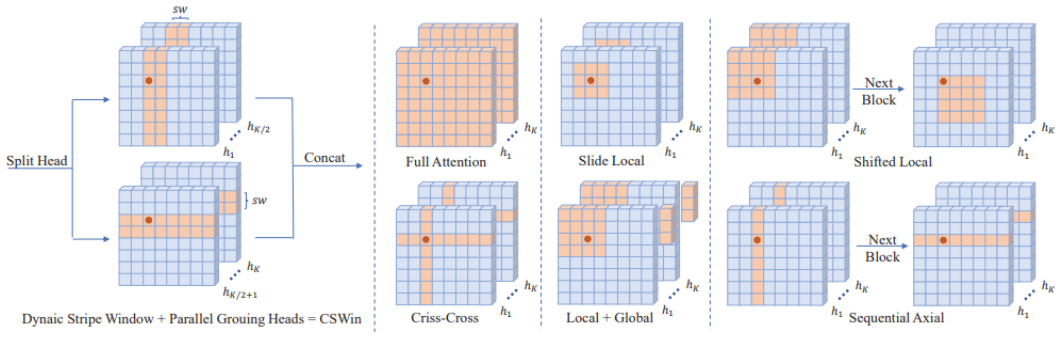

Figure 3.8: Different self-attention mechanisms.

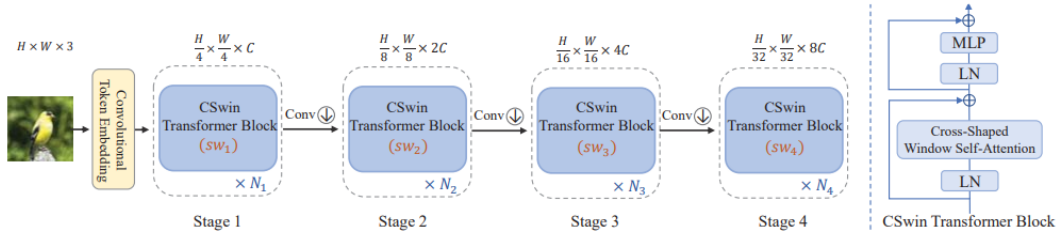

Figure 3.9: Overall architecture for CSWin.

3.2.6 Neighborhood Attention Transformer

Neighborhood Attention Transformer is proposed by Ali Hassani in 2022, which is an efficient, accurate and scalable hierarchical transformer that works well on both image classification and downstream vision tasks. It is built upon Neighborhood Attention (NA), a simple and flexible attention mechanism that localizes the receptive field for each query to its nearest neighboring pixels. NA is a localization of self-attention, and approaches it as the receptive field size increases. It is also equivalent in FLOPs and memory usage to Swin Transformer's shifted window attention given the same receptive field size, while being less constrained. Furthermore, NA includes local inductive biases, which eliminate the need for extra operations such as pixel shifts.

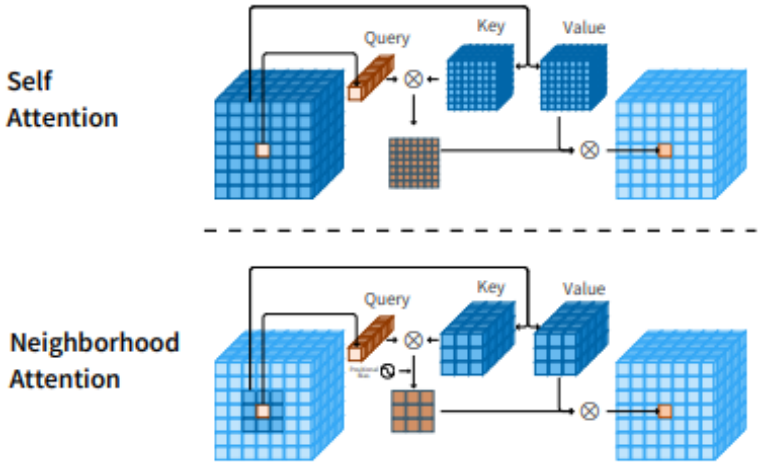

Figure 3.10: Neighborhood attention versus self attention.

3.3 Loss Funcion

For loss functions, we tried cross-entropy loss (CE), mutual channel with cross-entropy loss (MCCE) [3], focal loss (FL) [14], and adaptive focal loss (FLSD53) [19].

3.3.1 Cross-Entropy Loss (CE)

CE is the most popular loss function in classification tasks. It can be written in the form $\mathcal{L}_{CE}(y, y^{gt}) = -\sum_{i=1}^{C} y^{gt} \log y_i$, where y is the probability of prediction, y^{gt} is the one-hot label, and C is the number of categories.

3.3.2 Mutual Channel Loss (MC-Loss)

Mutual-Channel Loss was proposed in 2020 and is mainly used for fine-grained classification problems. This method can be ap-plied directly to the ResNet50 model. MC-loss attempt to represent each class by several feature channels. MC-loss is composed of two mainly components below

1. Force all feature channels belonging to the same class to be discriminative.
2. Force the feature channels in the same class to discover different discriminative regions.

Thus, MC-loss can be represented by

$$\mathcal{L}_{MC} = \mathcal{L}_{CE} + \mu(\mathcal{L}_{dis} - \lambda\mathcal{L}_{div})$$

where μ, λ are hyper-parameters.

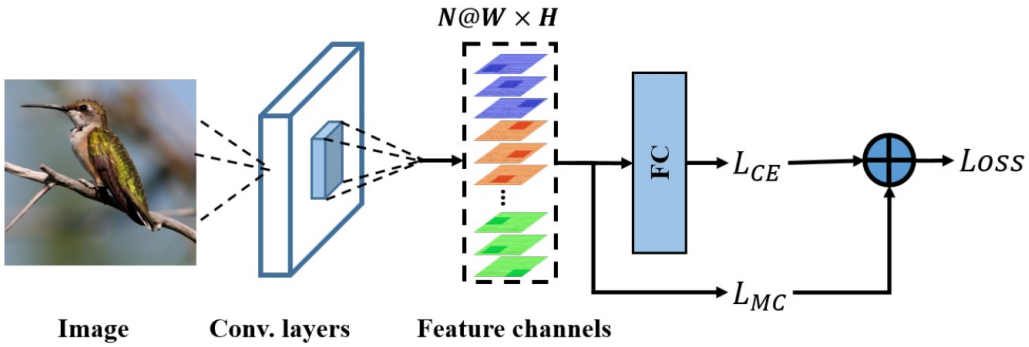

Figure 3.11: MC loss framework.

3.3.3 Focal Loss (FL)

Focal Loss is proposed in 2018, which is used to solve the data imbalance problem. Since the candidate object in a picture has most of the proportions of the background rather than the foreground, there will be an imbalance in calculating the loss. The hard example can be trained as much as possible during the training process and ignored those easy examples. So, it can solve the imbalance problem. FL is defined as

$$\mathcal{L}_{FL}(y, y^{gt}) = -(1-p^t)^{\gamma} \log p_t$$

where $p_t = \begin{cases} y, & \text{if } y^{gt} = 1 \\ 1-y, & \text{otherwise} \end{cases}$ and $\gamma \geq 0$ is called focusing parameter.

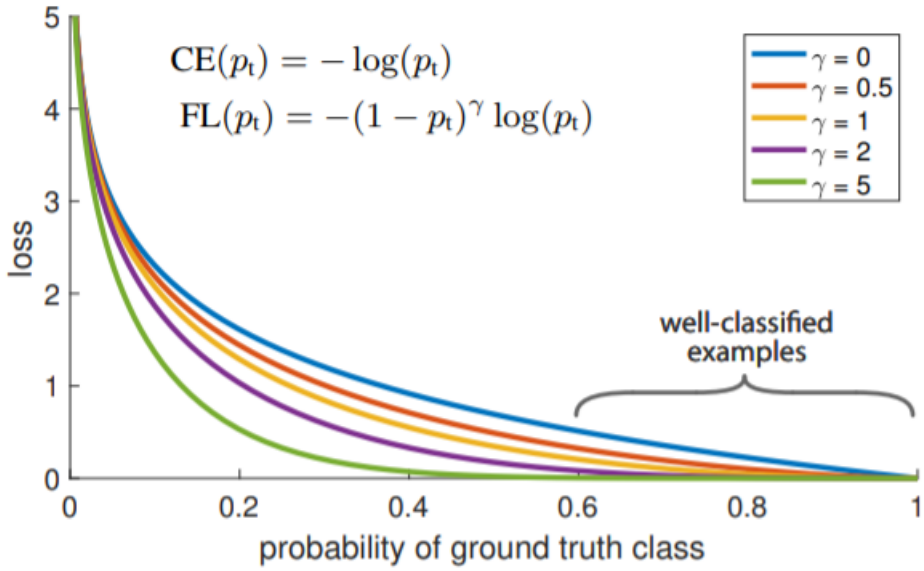

Figure 3.12: Focal loss with different γ.

3.4 Optimization

We mainly use SGD and AdamW as our optimizer, AdamW is a stochastic optimization method that modifies Adam's typical implementation of weight decay by decoupling weight decay from the gradient update. The algorithm is as follows

Algorithm AdamW

Input: $f(\theta)$ (objective), γ (lr), $\beta_1, \beta_2, \theta_0, \varepsilon, \lambda$ (weight decay)
Initial: $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$ (second moment),
for $t = 1$ to ... **do**
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  $\theta_t \leftarrow \theta_{t-1} - \gamma\lambda\theta_{t-1}$
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  $\hat{m}_t \leftarrow \dfrac{m_t}{1 - \beta_1^t}$
  $\hat{v}_t \leftarrow \dfrac{v_t}{1 - \beta_2^t}$
  $\theta_t \leftarrow \theta_{t-1} - \gamma \dfrac{\hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}$
end for
return θ_t

3.5 Learning Rate Scheduler

To further improve results and let the model converge to the global minimum, we adjust the learning rate by exponent decay. It represent by the following:

$$\text{Learning Rate} = (\text{Initial Learning Rate}) \times (\text{Decay Factor})^{\frac{\text{Epoch}}{\text{Decay Period}}}$$

Chapter 4

Experimental Results and Discussion

4.1 Performance Evaluation

Confusion matrix is a tool for visually predicting network performance. As shown in Fig. 4.1, it consists of True Positive (TP), False Positive (FP), False Negative (FN), and True Negative (TN). In particular, TP is the number of prediction which is predicted positive and the corresponding Ground Truth (GT) is also positive. TN is the number of prediction which is predicted negative and the corresponding GT is also negative. FP is the number of prediction which is predicted positive and the corresponding GT is negative. FN is the number of prediction which is predicted negative and the corresponding GT is positive.

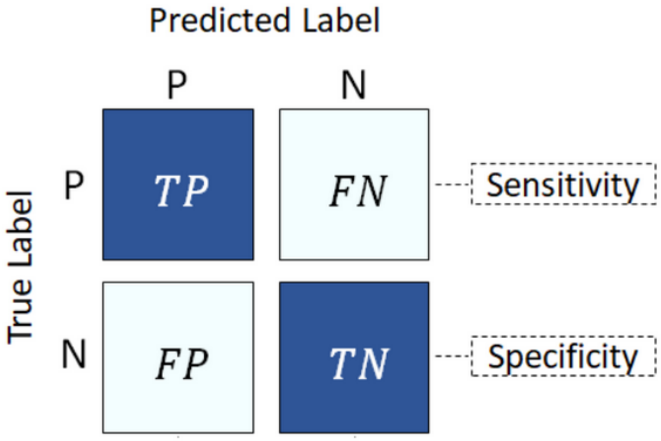

Figure 4.1: Confusion matrix.

After knowing the meaning of confusion matrix, we can calculate the Accuracy, Precision, and Recall by Eq. 4.1, Eq. 4.2, and Eq. 4.3, respectively. Further, we can calculate the F_1 score by using the formula as shown in Eq. 4.4.

$$Accuracy = \frac{\#\ of\ correctly\ classified\ samples}{\#\ number\ of\ total\ samples} \tag{4.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

$$F_1 score = 2 \times \frac{Precision \cdot Recall}{Precision + Recall} \tag{4.4}$$

The scoring standard is 50% for accuracy and 50% for Marco-F_1 score. The Macro-F_1 score counts the number of TP, FP, FN, and TN for each category and calculates the corresponding precision and recall, then take the average to obtain the Marco-F_1 score which can be presents in Eq. 4.5.

$$Macro\ F_1\ score = \frac{1}{C} \times \sum_{i=1}^{C} F_i^{(i)}, \tag{4.5}$$

where i is the index of ith class and C is the total number of class. As mentioned above, the final score is the weighted average for accuracy and Macro-F_1 score.

4.2 Validation Results

First, we use the ResNet101 architecture to set the baseline for this competition. With several combinations of optimizer and learning rate, we can get the best validation accuracy about 82.42% as shown in Table 4.1, which is a good start. After obtaining the baseline scores, we tried many augmentation methods, deep learning models, optimizers and loss functions and tried to find what combination would give the best performance.

model	batch size	image size	loss	optimizer	learning rate	val acc.
ResNet50	8	224	CE	SGD	$1e-3$	0.1758
ResNet50	8	224	CE	SGD	$1e-2$	0.6986
ResNet50	8	224	CE	SGD	$5e-2$	0.7968
ResNet101	8	224	CE	SGD	$1e-3$	0.2329
ResNet101	8	224	CE	SGD	$5e-2$	**0.8242**
ResNet101	8	224	CE	SGD	$4e-2$	0.8196
ResNet101	8	224	CE	Ranger	$5e-2$	0.4566
ResNeSt269	8	224	CE	SGD	$1e-2$	0.7009

Table 4.1: Validation accuracy for baseline models.

Table 4.2 shows the validation accuracy for our experiments with different models, batch size, loss function, and learning rate. In addition, we can see that ViT-B_16 model have the highest accuracy on test dataset. Also, training the ViT-B_16 model does not take a lot of time compared to other models. Thus, we trained lots of ViT-B_16 on 4-fold datasets. Furthermore, we use test-time augmentation while doing the inference.

model	batch size	image size	loss	optimizer	learning rate	val acc.
EfficientNet-B4	16	480	CE	AdamW	$1e-3$	0.8630
EfficientNet-B4	16	480	MCCE	AdamW	$1e-3$	0.8584
EfficientNet-B4	16	480	FL	AdamW	$1e-3$	0.8699
EfficientNet-B4	16	480	FLSD53	AdamW	$1e-3$	0.8470
EfficientNet-B4	32	480	CE	AdamW	$1e-3$	0.8402
EfficientNet-B4	32	480	FLSD53	AdamW	$1e-3$	0.8607
ConvNeXT-B	64	384	FL	AdamW	$3e-4$	0.8836
Swin-B	32	384	CE	AdamW	$3e-4$	0.8927
Swin-B	32	384	FL	AdamW	$3e-4$	0.9110
CSwin-B	32	384	FL	AdamW	$3e-5$	0.8858

Table 4.2: Validation accuracy for different models.

In Fig. 4.2, it demonstrates the different accuracy curves for the corresponding loss function while training the EfficientNet model. We can see that the accuracy will be lower if FL is used as the loss function. The possible reason is that the parameters are not adjusted properly.

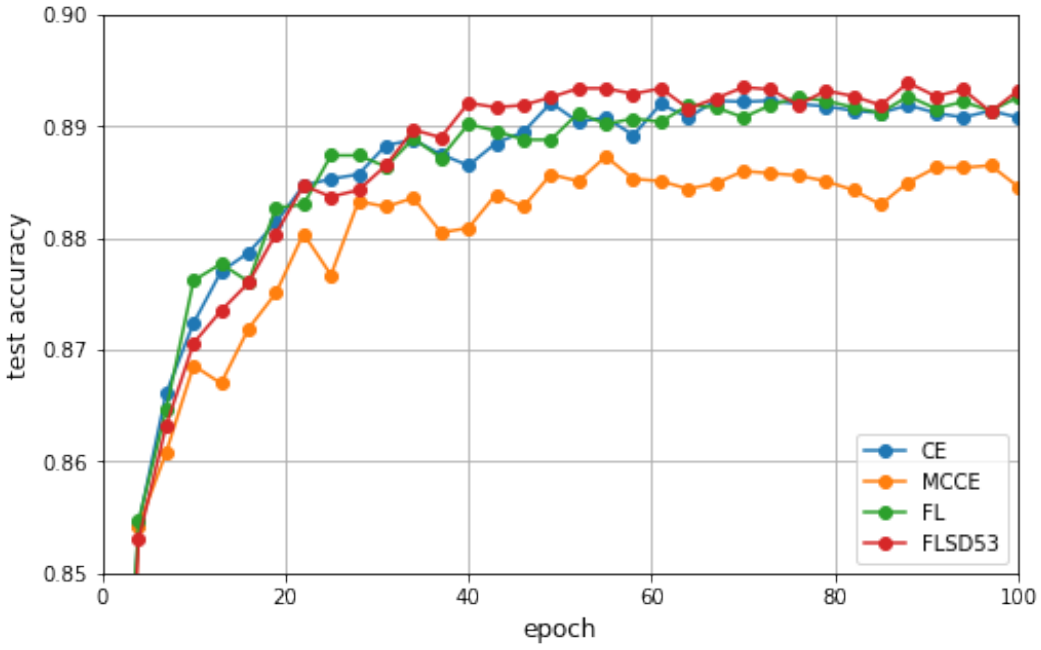

Figure 4.2: Test accuracy curves for EfficientNet.

4.3 4-Fold Results

Since ViT-based models are SOTA and they have attention mechanism while doing the inference, we mainly use ViT and Swin models for this competition. As shown in Table 4.3, it contains 32 results from ViT-B_16 models and we trained 8 ViT-B_16 for each fold. We also do the ensemble to combine each result at finals.

model	fold	batch size	image size	loss	optimizer	learning rate	test acc.
ViT-B_16	1	4	480	CE	SGD	$3e-2$	0.9110
ViT-B_16	1	4	480	CE	SGD	$3e-2$	0.9110
ViT-B_16	1	4	480	CE	SGD	$3e-2$	0.8973
ViT-B_16	1	4	480	CE	SGD	$3e-2$	0.9064
ViT-B_16	1	4	480	CE	SGD	$3e-2$	0.9041
ViT-B_16	1	4	480	CE	SGD	$3e-2$	0.8995
ViT-B_16	1	4	480	CE	SGD	$3e-2$	0.8881
ViT-B_16	1	4	480	CE	SGD	$3e-2$	0.9018
ViT-B_16	2	4	480	CE	SGD	$3e-2$	0.9087
ViT-B_16	2	4	480	CE	SGD	$3e-2$	0.9018
ViT-B_16	2	4	480	CE	SGD	$3e-2$	0.9064
ViT-B_16	2	4	480	CE	SGD	$3e-2$	0.9132
ViT-B_16	2	4	480	CE	SGD	$3e-2$	0.8973
ViT-B_16	2	4	480	CE	SGD	$3e-2$	0.8904
ViT-B_16	2	4	480	CE	SGD	$3e-2$	0.9087
ViT-B_16	2	4	480	CE	SGD	$3e-2$	0.9064
ViT-B_16	3	4	480	CE	SGD	$3e-2$	0.9132
ViT-B_16	3	4	480	CE	SGD	$3e-2$	0.9018
ViT-B_16	3	4	480	CE	SGD	$3e-2$	0.9178
ViT-B_16	3	4	480	CE	SGD	$3e-2$	0.9110
ViT-B_16	3	4	480	CE	SGD	$3e-2$	0.8995
ViT-B_16	3	4	480	CE	SGD	$3e-2$	0.9132
ViT-B_16	3	4	480	CE	SGD	$3e-2$	0.9155
ViT-B_16	3	4	480	CE	SGD	$3e-2$	0.9041
ViT-B_16	4	4	480	CE	SGD	$3e-2$	0.9087
ViT-B_16	4	4	480	CE	SGD	$3e-2$	0.9110
ViT-B_16	4	4	480	CE	SGD	$3e-2$	0.9178
ViT-B_16	4	4	480	CE	SGD	$3e-2$	0.8927
ViT-B_16	4	4	480	CE	SGD	$3e-2$	0.9087
ViT-B_16	4	4	480	CE	SGD	$3e-2$	0.9132
ViT-B_16	4	4	480	CE	SGD	$3e-2$	0.9110
ViT-B_16	4	4	480	CE	SGD	$3e-2$	0.9178

Table 4.3: 4-Fold test accuracy for ViT models.

Table 4.4, it contains 12 results from Swin-B models and we can see that most of test accuracy for Swin-B model on 4-fold datasets are higher than ViT models shown as above.

model	fold	batch size	image size	loss	optimizer	learning rate	test acc.
Swin-B	1	64	384	CE	AdamW	$3e-4$	0.9247
Swin-B	2	64	384	CE	AdamW	$3e-4$	0.9087
Swin-B	3	64	384	CE	AdamW	$3e-4$	0.9292
Swin-B	4	64	384	CE	AdamW	$3e-4$	0.9429
Swin-B	1	64	384	FL	AdamW	$3e-4$	0.9269
Swin-B	2	64	384	FL	AdamW	$3e-4$	0.9132
Swin-B	3	64	384	FL	AdamW	$3e-4$	0.9269
Swin-B	4	64	384	FL	AdamW	$3e-4$	0.9361
Swin-B	1	64	384	FLSD	AdamW	$3e-4$	0.9315
Swin-B	2	64	384	FLSD	AdamW	$3e-4$	0.9087
Swin-B	3	64	384	FLSD	AdamW	$3e-4$	0.9247
Swin-B	4	64	384	FLSD	AdamW	$3e-4$	0.9384

Table 4.4: 4-Fold test accuracy for Swin models.

4.4 Submitted Results

Let's look at the final scores. Table. 4.5 shows five results corresponding to our submitted files. Our best result consists of 0.911492 in public score and 0.809624582 in private score, and the corresponding public rank and private rank are 16 and 13, respectively. In addition, our final ranking is 15th out of 743 teams.

Model	Public score	Public rank	Private score	Private rank
Swin (25) + ViT (6)	0.9098	-	-	-
ViT (6)	**0.911492**	**16**	**0.809625**	**13**
Swin (16)	0.9049	-	-	-
Swin (9)	0.9016	-	-	-
ViT(2)	0.8901	-	-	-

Table 4.5: Final submitted scores.

4.5 Discussion

4.5.1 Correlation Matrix

Single Result

Fig. 4.3 shows the inference results of the ViT-B_16 model on the test dataset, with an accuracy of about 0.9110, and trained on the fold 1 dataset.

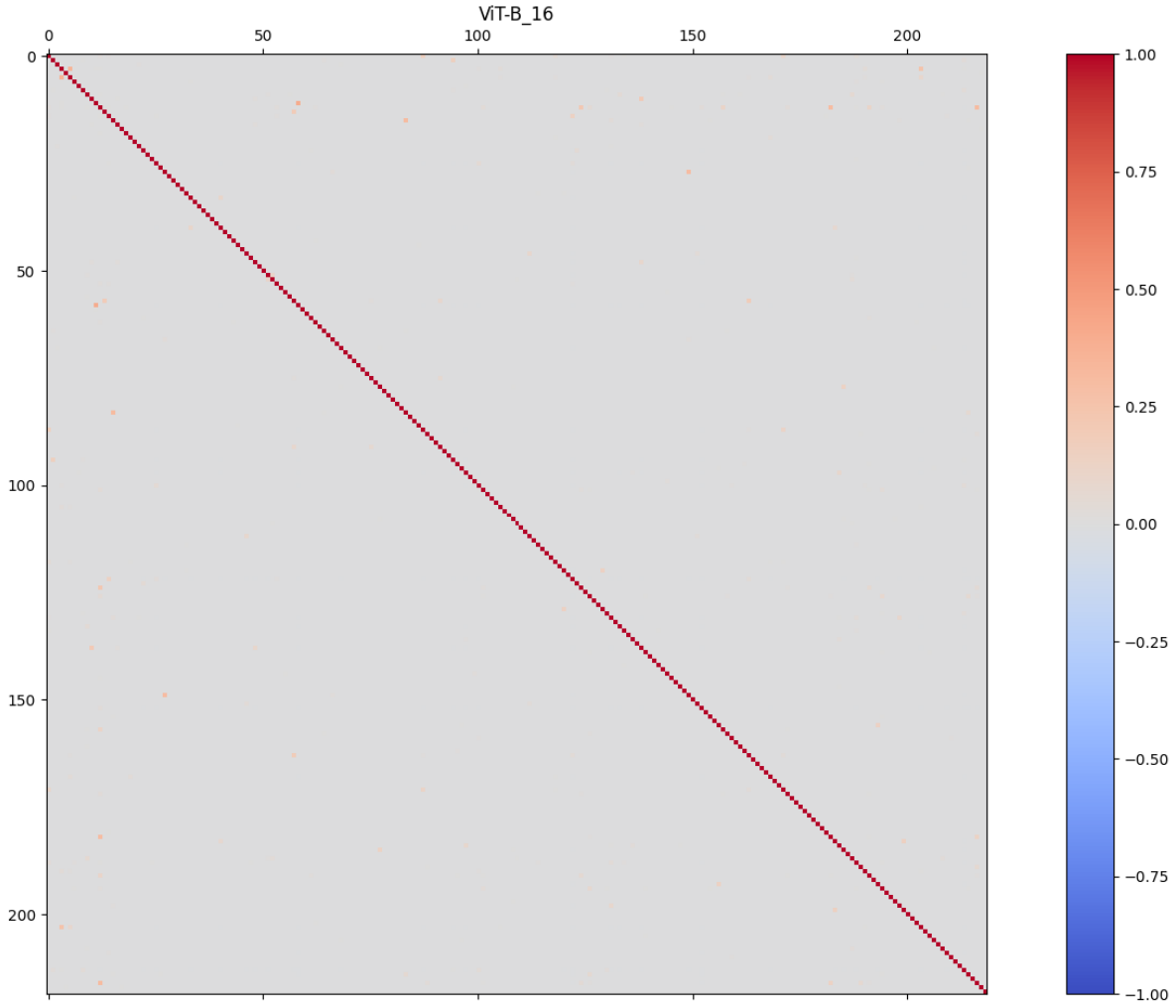

Figure 4.3: Correlation matrix for single model results.

Ensemble Result

Fig. 4.4 shows the correlation matrix of the ensemble results for the five models. Here, we use the mean ensemble method to ensemble each prediction by using the corresponding probability. The test accuracy can be raised to 0.93379 in our 4-fold test dataset.

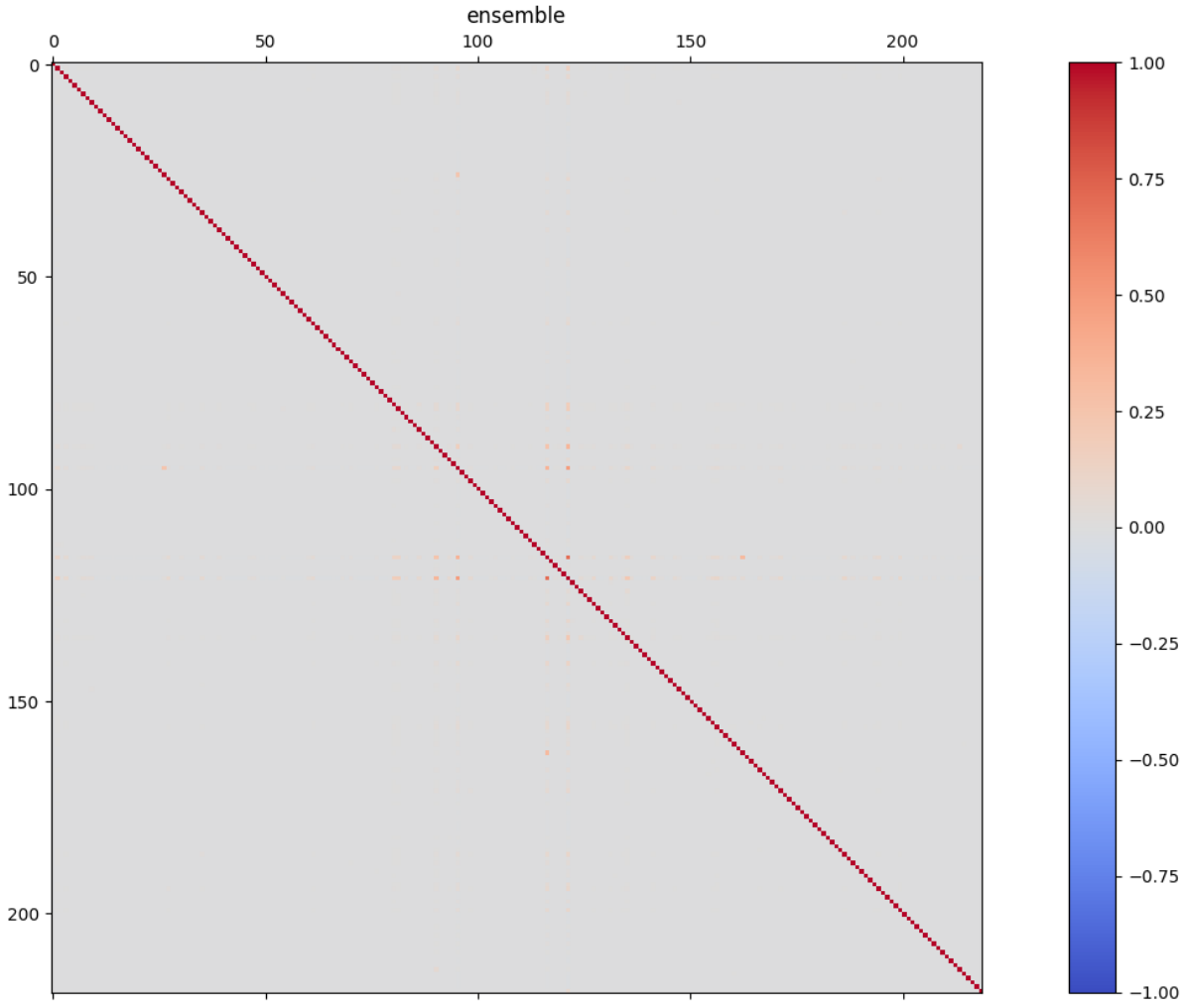

Figure 4.4: Correlation matrix for ensemble results.

Compare two correlation matrices above, we can find that the categories of prediction errors are not the same before and after doing the ensemble. Therefore, we think that the results can be improved a lot if the ensemble technique is handled well.

4.5.2 Attention Map

We also draw the attention map and observe whether the region for model doing the inference is reasonable. Fig. 4.5 shows an example of attention map which contains the top five possible categories and the corresponding confidence scores.

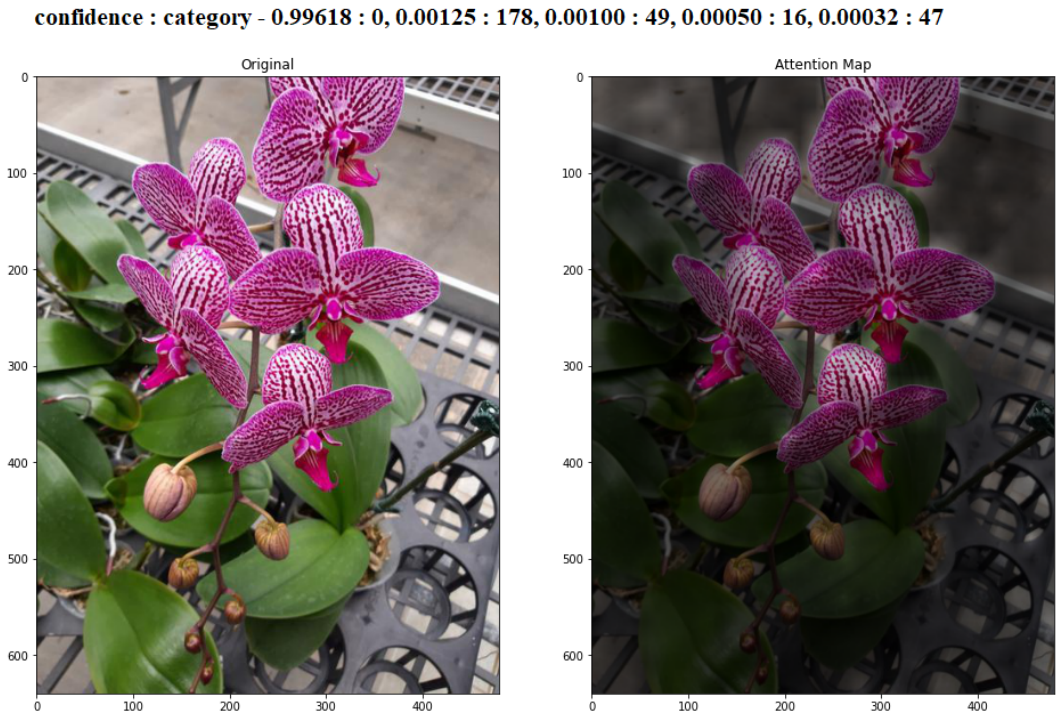

Figure 4.5: An example for the attention map.

4.5.3 Venn Diagrams for Submitted Files

As mentioned above, we trained lots of SOTA models, but due to the limitation of inference time, only some models with higher accuracy were selected for ensemble. Fig. 4.6 and Fig. 4.7 demonstrate the Venn Diagrams for our results and the corresponding submitted scores. From Fig. 4.6, we can see that the public scores for two ensembled ViT and a Swin are 0.890142 and 0.901620, respectively. But they only had $39,222$ predictions were same in $81,710$ images. This may indicate that the distribution of ViT and Swin are quite different. Thus, we guess that if we ensemble the ensembled ViT result and the ensembled Swin result by using vote ensemble would greatly improve the final score. However, Fig. 4.7 shows such operator does not make the result better. Hence, our best Marco-F_1 score still the ensemble result of the six ViTs.

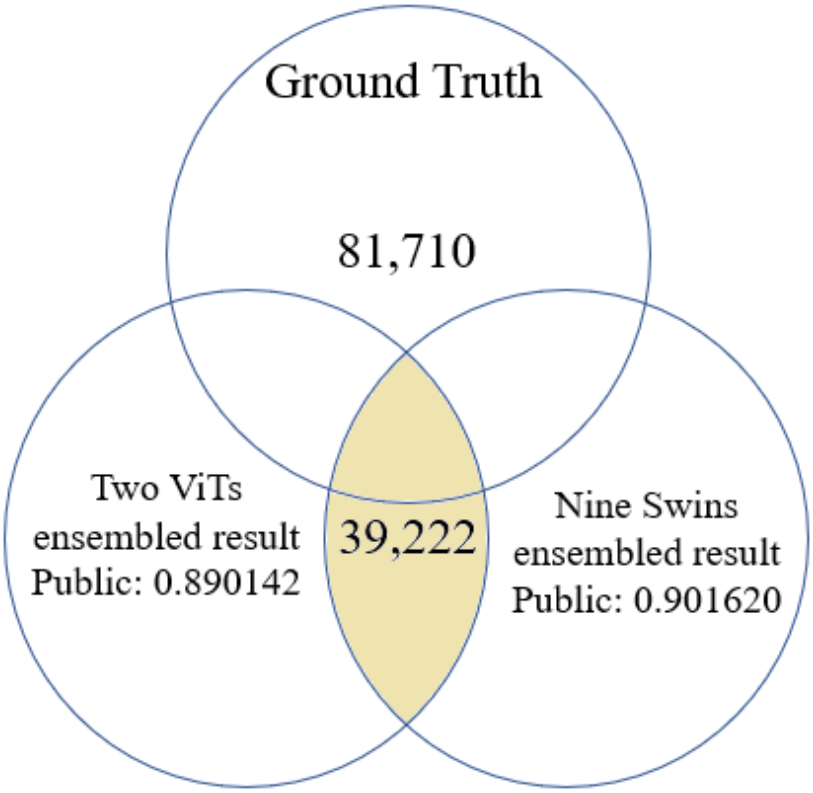

Figure 4.6: Venn Diagrams for ViTs and Swins.

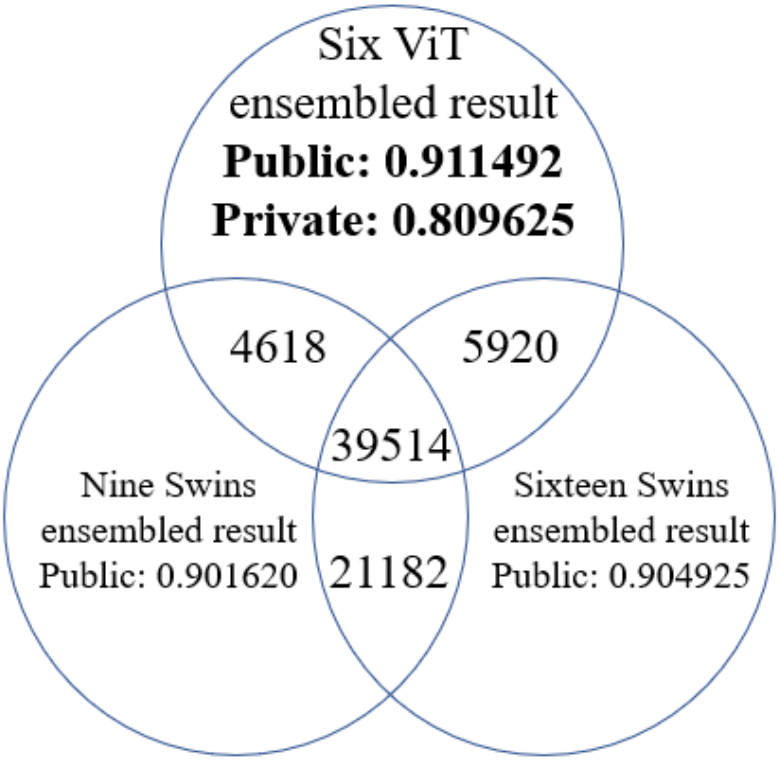

Figure 4.7: Public score of ensembles 6 ViTs, 9 Swins, and 16 Swins is 0.909891.

Chapter 5

Conclusion and Future Works

In this competition, we used many different models, data augmentation methods, training objective functions, optimization methods, and learning rate schedule. Then we doing lots of combination based on these modules and try to try to get as high Marco-F_1 coresas possible. After many trials, we extracted the six predictions with the highest accuracy and ensemble the results to get the final predictions. Under this system, our Marco-F_1 cores are 0.911493 and 0.809624 in public and private datasets, respectively. In addition, our overall ranking is $15th$ out of 743 teams.

In the future, if there are similar competitions with little amount of labeled data, except using more powerful network models and graphic cards, and few-shot learning method, we would like to explore external datasets and use self-supervised learning (SSL) methods.

Bibliography

[1] Pulung Nurtantio Andono, Eko Hari Rachmawanto, Nanna Suryana Herman, and Kunio Kondo. Orchid types classification using supervised learning algorithm based on feature and color extraction. *Bulletin of Electrical Engineering and Informatics*, 10(5):2530–2538, 2021.

[2] Diah Harnoni Apriyanti, Aniati Murni Arymurthy, and Laksana Tri Handoko. Identification of orchid species using content-based flower image retrieval. In *2013 International conference on computer, control, informatics and its applications (IC3INA)*, pages 53–57. IEEE, 2013.

[3] Dongliang Chang, Yifeng Ding, Jiyang Xie, Ayan Kumar Bhunia, Xiaoxu Li, Zhanyu Ma, Ming Wu, Jun Guo, and Yi-Zhe Song. The devil is in the channels: Mutual-channel loss for fine-grained image classification. *IEEE Transactions on Image Processing*, 29:4683–4695, 2020.

[4] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[5] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

[6] Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. Cswin transformer: A general vision transformer backbone with cross-shaped windows. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12124–12134, 2022.

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[8] ROBERT L Dressler. Features of pollinaria and orchid classification. *Lindleyana*, 1(2):125–130, 1986.

[9] Robert L Dressler. *Phylogeny and classification of the orchid family*. Cambridge University Press, 1993.

[10] Michael Fleischman and Eduard Hovy. Fine grained classification of named entities. In *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002.

[11] Ali Hassani, Steven Walton, Jiachen Li, Shen Li, and Humphrey Shi. Neighborhood attention transformer. *arXiv preprint arXiv:2204.07143*, 2022.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[13] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in Neural Information Processing Systems*, 33:18661–18673, 2020.

[14] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, 2017.

[15] Chen Liu, Yanwei Fu, Chengming Xu, Siqian Yang, Jilin Li, Chengjie Wang, and Li Zhang. Learning a few-shot embedding model with contrastive learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8635–8643, 2021.

[16] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.

[17] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022.

[18] Orchid Majumder, Avinash Ravichandran, Subhransu Maji, Alessandro Achille, Marzia Polito, and Stefano Soatto. Supervised momentum contrastive learning for few-shot classification. *arXiv preprint arXiv:2101.11058*, 2021.

[19] Jishnu Mukhoti, Viveka Kulharia, Amartya Sanyal, Stuart Golodetz, Philip Torr, and Puneet Dokania. Calibrating deep neural networks using focal loss. In

H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15288–15299. Curran Associates, Inc., 2020.

[20] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[21] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34, 2020.

[22] Ze Yang, Tiange Luo, Dong Wang, Zhiqiang Hu, Jun Gao, and Liwei Wang. Learning to navigate for fine-grained classification. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 420–435, 2018.

[23] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[24] Peiqin Zhuang, Yali Wang, and Yu Qiao. Learning attentive pairwise interaction for fine-grained classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13130–13137, 2020.

Appendix A

Environment

Our system is Ubuntu which is a Linux distribution based on Debian and composed mostly of free and open-source software. Further, we use Python and Bash which is a Unix shell and command language written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell as our programming languages. Besides, the libraries we used including `logging`, `argparse`, `os`, `csv`, `numpy`, `random`, `pandas`, `tqdm`, `scipy`, `pandas`, `matplotlib`, `seaborn`, `scikit-learn`, `torch`, `torchvision`, `tensorboard`, `ml-collections`, `datetime`, `ttach`, `apex`, etc.

A.1 Hardware Information

The following list contains our hardware information.

- CPU: i7-11700F / GPU: GeForce GTX 1660 SUPER™ VENTUS XS OC (6G)
- CPU: i7-10700K / GPU: NVIDIA GeForce RTX 2070 SUPER (8G)
- TWCC GPU: NVIDIA V100 (32G)

A.2 Pre-trained Models

The following list contains the pre-trained models we used.

- **ResNet**
  - ResNet101
- **Vision Transformer**
  - **ImageNet-22K pre-train**
    - ViT-B_16
    - ViT-B_32

        - ViT-L_16
        - ViT-L_32
        - ViT-H_14
        - R50+ViT-B_16
- **Swin Transformer**
    - **ImageNet-22K pre-train**
        - Swin-B
- **CSWin Transformer**
    - **ImageNet-1k pre-train**
        - CSWin-B

A.3 External Resources

The following list contains the external resources we used.

- **GitHub Repository**
    - **Model**
        - Vision Transformer and MLP-Mixer Architectures
        - Swin Transformer
        - CSWin Transformer
    - **Augmentation**
        - AutoAugment - Learning Augmentation Policies from Data
        - TTAch
    - **Training Objective Function**
        - Focal Calibration
        - Mutual-Channel Loss for Fine-Grained Image Classification
    - **Optimizer**
        - Ranger-Deep-Learning-Optimizer
        - Ranger21 - integrating the latest deep learning components into a single optimizer
        - (Adaptive) SAM Optimizer
    - **Mixed Precision**
        - A PyTorch Extension: Tools for easy mixed precision and distributed training in Pytorch

Appendix B

Contact Information

- **Team**

Team name	Upload time	Public score	Public rank	Private score	Private rank
TEAM_142	2022-06-01 10:50:24	0.911492	16	0.809625	13

- **Team member**

Name	School name	Department	Phone number	Email address
Yu-Hsi Chen (陳育熙)	National Yang Ming Chiao Tung University (國立陽明交通大學)	Department of Applied Mathematics (應用數學系)	0905910509	yuhsi44165.sc09@nycu.edu.tw
Jia-Wei Liao (廖家緯)	National Yang Ming Chiao Tung University (國立陽明交通大學)	Department of Applied Mathematics (應用數學系)	0939580280	sam23582211@gmail.com
Kuok-Tong Ng (吳國棟)	National Yang Ming Chiao Tung University (國立陽明交通大學)	Department of Applied Mathematics (應用數學系)	0910306618	l19so18h13k10@gmail.com

- **Instructor**

Instructor name	Course name	Course code	School name	Department	Phone number	Email
Ching-Chun Huang (黃敬群)	Machine Learning for Signal Processing (機器學習之訊號處理應用)	IOC5207	National Yang Ming Chiao Tung University (國立陽明交通大學)	Institute of Computer Science and Engineering (資訊工程學系)	03-5712121 # 54736	chingchun@cs.nycu.edu.tw

Appendix C

Reproduce the Best Result

In this section, we will demonstrate how to reproduce the best result by two different approaches. On the one hand, we can reproduce the best result by using Google Colaboratory, on the other hand, we can reproduce the best result on our local machine.

C.1 Google Colaboratory Version

C.1.1 Folder Structure

Setup the Folder Structure as follows. The easiest way is to copy the entire folder, but be aware that there is a lot of weights in this folder.

```
尋找花中君子 - 蘭花種類辨識及分類競賽 [TBrain]/
|-- datasets/
    |-- test/
        |-- 0/   # orchid_public_set, 40285
        |-- 1/   # orchid_private_set, 41425
|-- Reproduce the Best Result/
    |-- ViT/
        |-- output/
            |-- A1.bin, A2.bin, ID_4.bin, ID_5.bin, ID12.bin, ID27.bin #
                ViT-B_16
        |-- Reproduce.ipynb
        |-- submit_meanEnsemble_convert.csv # after running
            Reproduce.ipynb
```

C.1.2 Execute Notebook

After the setup, ready to execute **Reproduce.ipynb**, no additional steps are needed.

Note: It took about six hours to complete inference on the test data.

C.1.3 Submission File

After running **Reproduce.ipynb**, we can get the file named **submit_meanEnsemble_convert.csv** which has the highest Macro-F_1 score.

C.2 Local Machine Version

C.2.1 Environment Setup

Follow 1. Environment Setup step by step to setup the required environment.

C.2.2 Folder Structure

Setup the Folder Structure as follows.

```
|-- test/
    |-- 0/   # orchid_public_set, 40285
    |-- 1/   # orchid_private_set, 41425
|-- ViT-Orchids-Classification-main/
    |-- apex/
    |-- checkpoint/
    |-- compare.py
    |-- convert.py
    |-- models/
    |-- output/
        |-- A1.bin, A2.bin, ID_4.bin, ID_5.bin, ID12.bin, ID27.bin #
            ViT-B_16
    |-- utils/
    |-- requirements.txt
    |-- test.py/
    |-- train.py/
    |-- submit.py/
```

C.2.3 Execute submit.py

After the setup, ready to execute **submit.py** by using the following command, no additional steps are needed.

Listing C.1: Commad for execute submit.py.

```
python submit.py --model_type
    ["ViT-B_16","ViT-B_16","ViT-B_16","ViT-B_16","ViT-B_16","ViT-B_16"]
    --checkpoint
    ["output/A1.bin","output/A2.bin","output/ID_4.bin","output/ID_5.bin","output/ID12.bin
    --img_size [480,480,480,480,480,480] --use_imagenet_mean_std
    [0,0,0,0,1,1]
```

C.2.4 Execute convert.py

After execute submit.py, we can get two files named submit_voteEnsemble.csv and submit_meanEnsemble.csv, respectively. Now, we are ready to execute convert.py.

Listing C.2: Commad for execute convert.py

```
python convert.py
```

C.2.5 Submission File

After executing **convert.py**, we can get the file named **submit_meanEnsemble_convert.csv** which has the highest Macro-F_1 score.