NYCU Introduction to Machine Learning, Final Project

110550167，侯博軒

This assignment is dedicated to exploring and fine tuning a classification model for wild bird classification, trained and verified on the bird class dataset CUB200-2011, consisting 200 different kinds of bird. I adapted the model HERBS as the base line model of my project, credit of the model design is from the paper "Fine-grained Visual Classification with High-temperature Refinement and Background Suppression". Below I will discuss training details and prediction performance.

Part 1. Environmental Details

    a. Python version

       --Python 3.8.0

    b. Framework
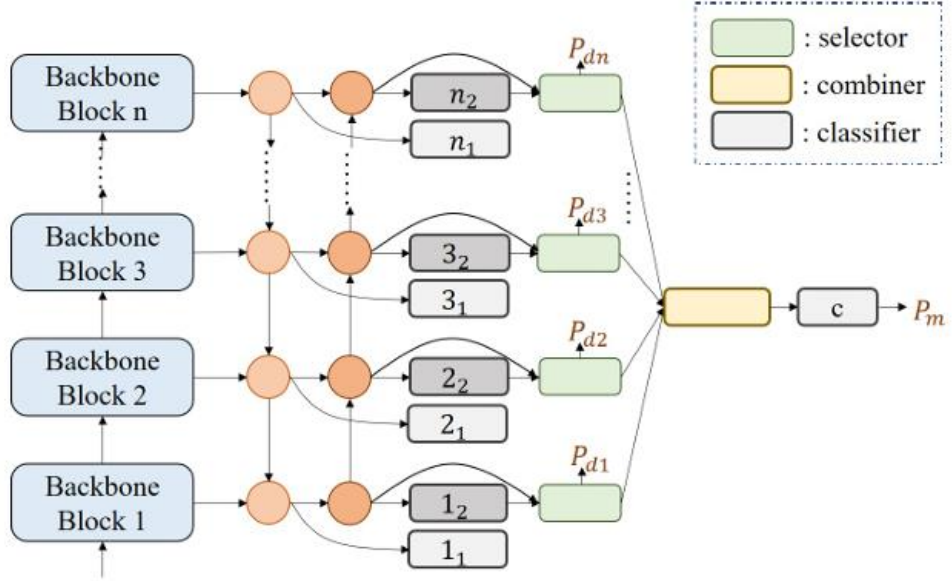
       --PyTorch

    c. Hardware

       All experiments are trained on a single Nvidia GeForce RTX 2080. It takes approximately 40 hours to complete a full training session.

    d. Inference reproduction

1. Download pre-trained weight to "training\records \FGVC-HERBS \basline_10percent_warmup_update_freq_8"
2. Install the requirements in "requirements.txt"
3. Put "inference.py" under the directory train
4. Put testing images under the directory "training/CUB200-2011/test" the layout should be:

   Train/CUB200-2011/test

            --image1.png

            --image2.png

            …

5. Run inference.py
6. The result will be in "training/prediction.csv"

Part 2. Implementation Details

a. Model Architecture

The model is composed of a back bone swing transformer to extract features from the input image. At different stage of the back bone block, these extracted features were passed to the top-down bottom-up feature fusion model (the orange dots above). We than train 8 different classifiers (dark gray and light gray block) to classify according to different stage of output features, and a selector combiner model to weight out all the predictions and make a final probability prediction of the classes. By this design we can learn information from different stages of feature, focusing on details in later layers and overall observations on earlier layers. The classifier benefits from top-down bottom-up feature fusion model because of its ability to propagate different stage information to all classifiers.

To fit this specific model design, HERBS proposed two modules, Background Suppression, and High Temperature Refinement to supervise their loss.

Background Suppression module could be depicted as the selector and combiner above. It selects features and makes a class prediction based on the merge feature, and the none selected features are named drop maps. The BS loss is composed of three terms:

1. Cross-entropy loss: Supervise the final classification, y is the label and P is the predicted probability from the combiner.

$$loss_m = - \sum_{ci=1}^{C_{gt}} y_{ci} \log(P_{m,ci})$$

2. Drop loss: Increase the gap between foreground and background, suppress features in the drop map Y_d.

$$P_d = \tanh(Y_d)$$

$$loss_d = \sum_{i=1}^{C_{gt}} (P_{d,ci} + 1)^2$$

3. Average pooling loss: Prevent feature map from having same response in the same location.

$$P_{li} = \mathrm{Softmax}(W_i(\mathrm{Avgpool}(hs_i)) + b_i)$$

$$loss_l = - \sum_{i=1}^{n} \sum_{ci=1}^{C_{gt}} y_{ci} \log(P_{li,ci})$$

The background suppression module loss function is a weighted sum of the three above losses.

$$loss_{bs} = \lambda_m loss_m + \lambda_d loss_d + \lambda_l loss_l$$

High Temperature Refinement module is designed to help the model learn more widely in early layers and deeper in later layers. With the hyper parameter T_e and classification result of classifier n1, n2 as Y_{i1} and Y_{i2}, the HTR loss can be formulated as below:

$$P_{i1} = \mathrm{LogSoftmax}(Y_{i1}/T_e)$$

$$P_{i2} = \mathrm{Softmax}(Y_{i2}/T_e)$$

$$loss_r = P_{i2} \log(\frac{P_{i2}}{P_{i1}})$$

To sum up, the total loss for the HERBS model could be written as below:

$$loss_{herbs} = loss_{bs} + \lambda_r loss_r$$

Balancing the loss between the BS and HTR module with hyper parameter λ_r.

b. Hyperparameters

   Hyper parameters are saved in config.yaml under the same directory alongside the model weight. These parameters could be divided into two classes according to their usage.

Training parameters

- max_lr: Maxima learning rate, which is then mapped into a cosine dacay function.
- wdecay: Weight decay in the training process.
- max_epochs: Epochs in training process.
- warmup_batches: The amount of batch at the beginning where learning rate is scheduled linearly.
- batch_size: The batch size during training.
- update_freq: Counts the frequency that the module performs back propagation.
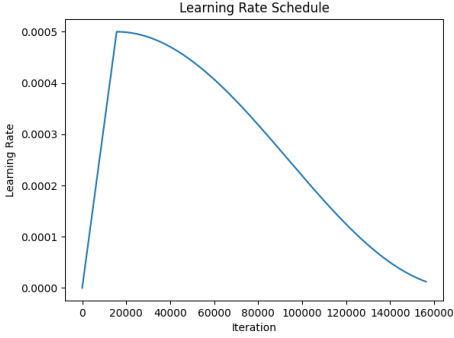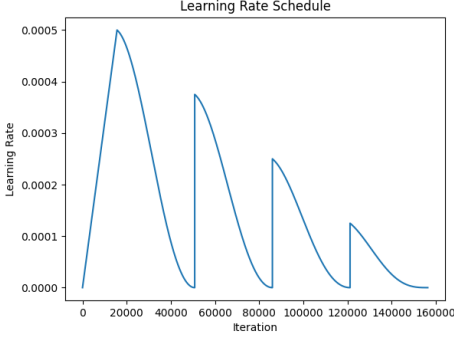
Objective function parameters

- lambda_b0: Balance the temperature and background suppression loss
- lambda_b: Balance the Average pool loss in background suppression module.
- lambda_n: Balance the drop loss in background suppression module.
- lambda_c: Balance the cross-entropy loss in background suppression module.
- temperature: Controls the temperature value used in High Temperature Refinement.

c. Training Strategy

   The model is guided by batches and a SGD optimizer.

   Learning rate during training is scheduled by cosine decay, which maps the learning rate by a cosine function throughout the training process except for the warm up batches. A simple graph of the mapped learning rate is given below. To further increase the performance of our model and not limit it to local maxims, I tried to implement a circular learning rate scheduling, the learning rate will go through 4 cycles and each time dividing the max learning

rate by its iteration number. This yields a better performance on the testing set in the kaggle contest, yet a less effective prediction on the local validation set.

	Cosine Decay	Circular Cosine Decay
Top1 accuracy		
validate	92.7%	92.5%
test	91%	91.1%

Part 3. Experimental Results

a. Evaluation Metrics

The model gets an accuracy score of 92.5% predicting the classes of the birds in the validation images. Considering that there are 200 classes in the dataset, which can be hard even for humans to distinguish bird classes at first guess, the top3 and top5 scores are also measured, giving the prediction a positive score if the correct bird class was within its third or fifth likely guess.

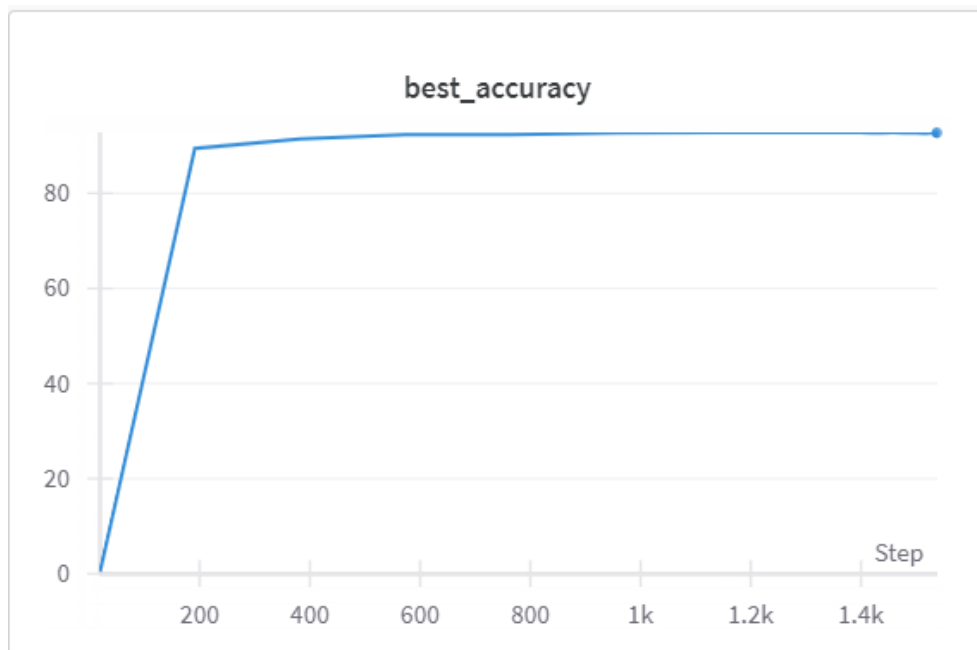|	Top 1	Top 3	Top 5
Accuracy score	92.503%	97.619%	98.328%

We also evaluate the model's classification ability on different bird sub classes. It can be observed that the model performance badly on certain sub classes, this may result from imbalanced data distribution, and less training samples on these classes.

	flycatcher	gull	kingfisher	sparrow	tern	warbler
Accuracy	92.5%	87.1%	75.6%	98.0%	91.8%	78.6%

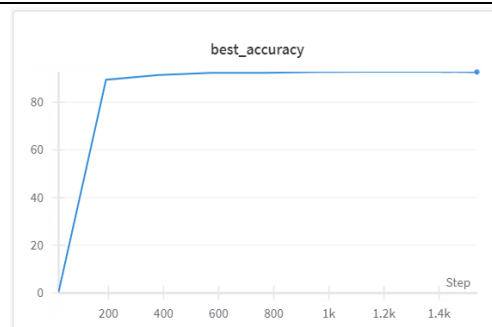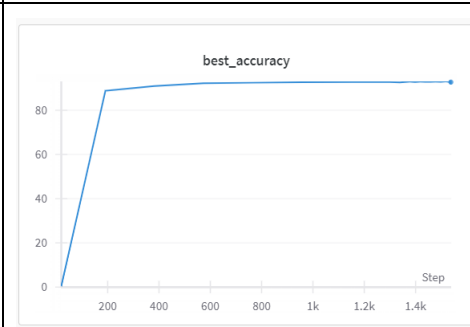|	woodpecker	wren
Accuracy	96.6%	95.7%

b. Learning Curve

Through the learning curve we can see that our model has reached a steady state after half of the training epochs. Indicating that we have reached a local or global minima of the objective function. Something worth noting is that the model has obtained over 80% accuracy after the first 10% of training, this is because of the pre-trained backbone network, and in our case am accurate swing transformer.

c. Ablation Study

For the ablation study, we are especially interested in the contribution of the High Temperature Refinement (HTR) module in the HREBS' structure. We first compare the module trained without the HTR loss term with the base line module.
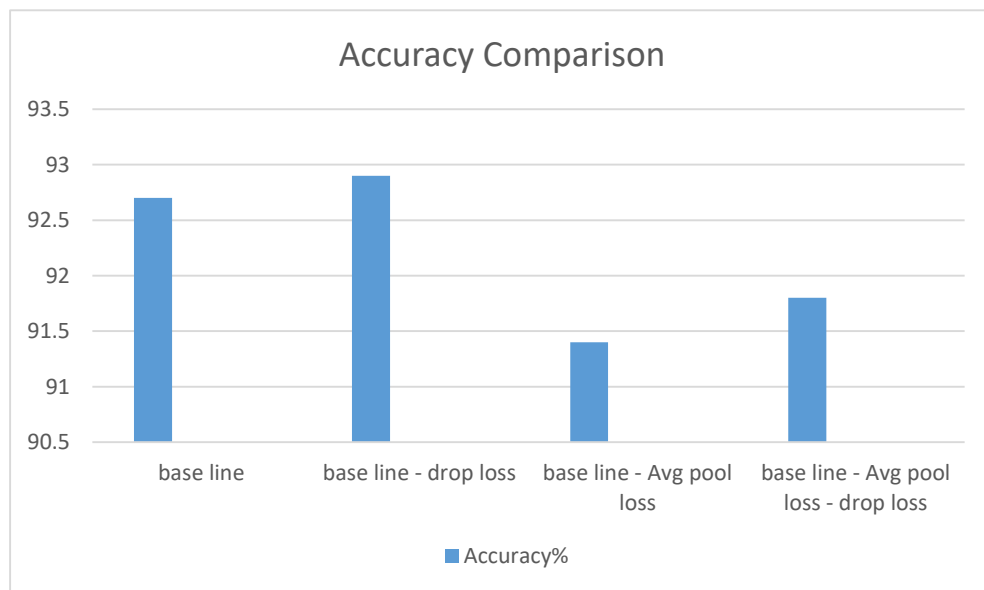
Base line	Base line - HTR
92.7%	92.7%

It was surprising to find that there was little performance gain by adding the HTR loss. Therefore, I further looked into the HTR module by testing different values for the hyper parameter "temperature", which the author of the original paper claims to help the model learn diverse representations in early layers and let later layers to focus on details. However, it seems that the temperature term fails to play an important role in guiding the model, indicated by the little performance influence of the multiplied temperature factor.

Model	Temperature = 64 (base line)	Temperature = 128
Accuracy	92.7%	92.6%

After examining the influence of HTR module, I decided to verify the other terms in the objection function. In the Background Suppression (BS) module, there are 3 different loss terms, namely, average pooling loss, drop loss, and cross-entropy loss. We know that cross-entropy loss is essential for classification problems, therefore I tested the module without average

pooling loss and drop loss independently. The result is graphed below.

Accuracy Comparison

A bar chart titled "Accuracy Comparison" with the y-axis ranging from 90.5 to 93.5. The bars show: base line ≈ 92.7, base line - drop loss ≈ 92.9, base line - Avg pool loss ≈ 91.4, base line - Avg pool loss - drop loss ≈ 91.8. Legend: Accuracy%.

From the result we can see that the Average pooling loss does indeed contribute greatly to the model's performance. On the other hand, drop loss seems to have negative impact on the model's performance. This may be due to the hyperbolic tangent term calculated for the drop loss, the original usage of the drop loss was to increase the gap between foreground and background, however this may result in losing important information from the input image. However, it could be also a temporary effect caused by the relatively small training set.