VRDL Homework 1

0716072 吳季嘉

GitHub link of my code

https://github.com/072jiajia/VRDL_HW1

Reference

https://github.com/PeiqinZhuang/API-Net
https://arxiv.org/pdf/2002.10191.pdf

Brief Introduction

In this homework, I used API-Net (Attentive Pairwise Interaction Network) to do the classification task. API-Net is a module which can attentively distinguish two fine-grained images via pairwise interaction.

I also used K-Fold to obtain K different training set and validation set, and trained K models on each of them. In the testing phase, I added up the K logits computed by the K models and chose the category with largest value as my prediction.

Methodology

In this section, I'd like to explain how API-Net works first, then describe some implementation details of my work.

API-Net. When learning to recognize some similar objects, humans usually focus on contrastive clues between image pairs.

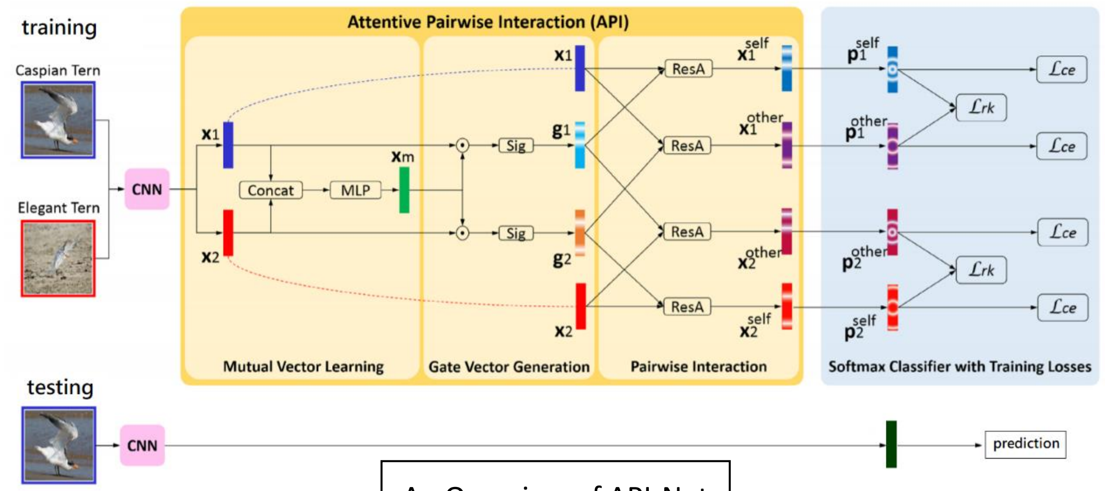
In the training phase, API-Net imitates this capacity of human.

First, it extracts the features of input images, and finds the most similar image of each image. It uses Euclidean distance in the feature space to determine the similarity of two images.

Gate Vector. Then, as we have the features of itself and its most similar image's features, it uses the features to compute a gate vector, which can be seen as the discriminative attention of a view.

$$\mathbf{x}_m = f_m([\mathbf{x}_1, \mathbf{x}_2])$$

$$\mathbf{g}_i = sigmoid(\mathbf{x}_m \odot \mathbf{x}_i), \quad i \in \{1, 2\}$$



An Overview of API-Net

Pairwise Interaction. Next, use the Gate Vector to highlight the discriminative clues via residual attention.

The last layer of this module is a fully-connected layer which can do 196-class classification.

$$\mathbf{x}_1^{self} = \mathbf{x}_1 + \mathbf{x}_1 \odot \mathbf{g}_1$$
$$\mathbf{x}_2^{self} = \mathbf{x}_2 + \mathbf{x}_2 \odot \mathbf{g}_2$$
$$\mathbf{x}_1^{other} = \mathbf{x}_1 + \mathbf{x}_1 \odot \mathbf{g}_2$$
$$\mathbf{x}_2^{other} = \mathbf{x}_2 + \mathbf{x}_2 \odot \mathbf{g}_1$$

Pair Construction. The above process will be done twice for each input image, one of it is pairing with the most similar image in the same class, and the other one is pairing with the most similar image in the other classes.

Loss Function. The loss function is composed of two parts, cross entropy loss for each image's prediction, and Score Ranking Regularization for each pair of predictions. The Score Ranking Regularization is thought that the feature highlighted by the object in the same class should lead to higher score than the one highlighted by the object in the other classes.

$$\mathcal{L} = \mathcal{L}_{ce} + \lambda \mathcal{L}_{rk}$$

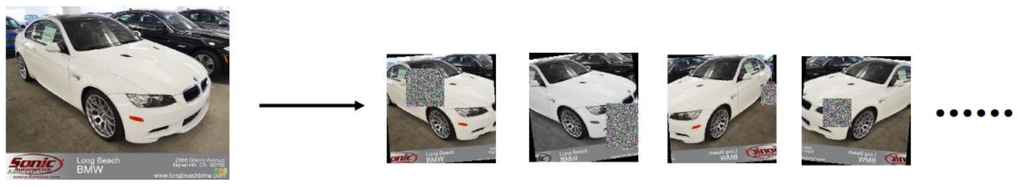
$$\mathcal{L}_{ce} = -\sum_{i\in\{1,2\}} \sum_{j\in\{self,other\}} \mathbf{y}_i^\top \log(\mathbf{p}_i^j)$$

$$\mathcal{L}_{rk} = \sum_{i\in\{1,2\}} \max(0, \mathbf{p}_i^{other}(c_i) - \mathbf{p}_i^{self}(c_i) + \epsilon)$$

Validation Phase and Testing Phase. In the training phase, API-Net has gradually generalized the discriminative power of CNN. Thus, in validation phase and testing phase, we can just input one single image without finding the discriminative clues between images. We can simply consider it as setting all the gate vectors to 0.

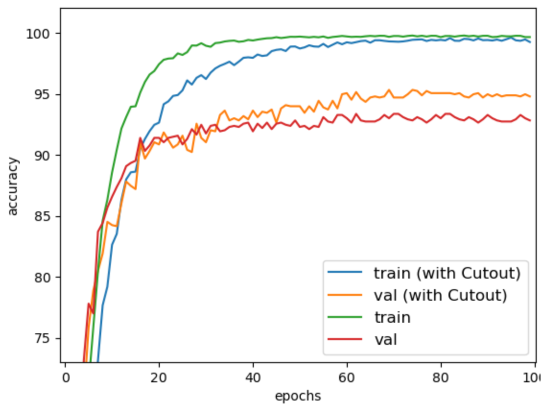
BackBone of API-Net. I tried ResNet-50, ResNet-101, DenseNet-121 as its backbone, ResNet-101 has the best performance. ResNet-152, DenseNet-169 or the other larger models have not been tried with the same batch size because of hardware limitation.

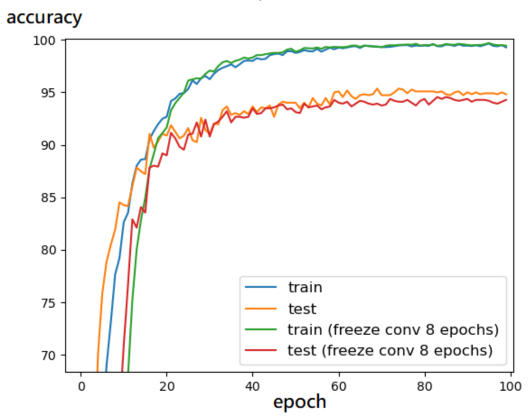
Data Augmentation. I resize all the image and make the shorter size of the images becomes 330 and randomly crop a 320*320 image from it. Then do random rotation, horizontal flipping, turn it to grayscale, and finally add a randomly generated mask on the image.



Training Detail. The total number of training epochs is 100. Every batch contains 30 categories, in each category, there are 4 images. I use SGD with momentum of 0.9, weight decay of 5e-4, initial learning rate of 0.01 and adopt cosine annealing to adjust it. In data augmentation, the degree of rotation is a random value between -15 to 15, the probability of changing an image into a grayscale is 10%, the proportion of the mask to the image is a random value between 0 to 0.25, and I do image normalization before training. I trained 10 models using different split of training set and validation set.

Experiments





Data Augmentation. I did an experiment about Cut Out data augmentation to check out if it still improve performance in fine-grained learning. I found that randomly apply a mask with random values on the image when training still improves the performance of the model.

Freeze Parameters. Many Papers have indicated that when we're doing transfer learning, we may freeze the pre-trained layers' parameters and start to update its value after several epochs, but I found that in this work, if I do not freeze the parameters, it can lead to better performance.

Summary

In this work, I adopt a module named API-Net to do fine-grained classification on the training dataset and make a prediction on the testing data set. I also adjusted the hyperparameters and tried several ways to improve the robustness of my models and show that it leads to better performence.