CS542 Fall 2020 Class Challenge: Fine Grained Categorization of Plants

Khoa Tran
Boston University
kttran@bu.edu

1. Introduction

The CS542 Fall Class Challenge asks students to categorize 256x256 RGB images of plants into 1000 categories using around 250,000 data samples. There are two parts to the challenge: supervised and semi-supervised learning. My submissions use ImageNet pre-built convolutional neural network models to solve both problems.

2. Details of my approach

2.1. Supervised learning

The given baseline model is a simple model that cannot represent the data very well. The best the model could achieve was around 60% error after altering the model to include data augmentation using random horizontal flips, increasing the number of epochs to 10, and applying L2 regularization to the later dense layers. These approaches aimed to reduce overfitting so that I could train the model longer, but the simplicity of the model limited its performance.

My final submission uses EfficientNetB7 by Mingxing Tan and Quoc V. Le [1]. The choice to use an ImageNet pre-built model was because the models have more depth and thus can represent the inputs better. I chose EfficientNetB7 over other models because it produces the lowest top 5 categorical error among the three models I tested. The other two being VGG16 by Karen Simonyan and Andrew Zisserman [2] and DenseNet201 by Huang et al [3].

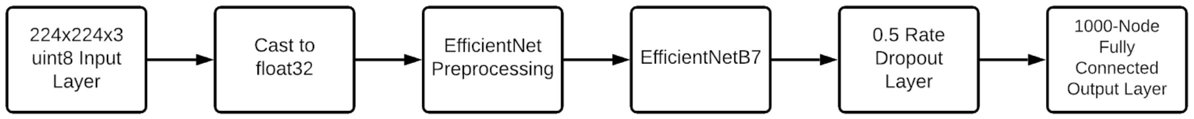
224x224x3 uint8 Input Layer
Cast to float32
EfficientNet Preprocessing
EfficientNetB7
0.5 Rate Dropout Layer
1000-Node Fully Connected Output Layer

Figure 1: Architecture of the supervised learning model for my final submission

The final model uses EfficientNetB7 without the top fully-connected layer and with global max pooling and initial ImageNet weights. I also added some input layers to fit the dimensions and number representation of the inputs, and a dropout layer and a fully-connected layer to properly output the results. With the Adam optimizer and amsgrad turned on, 0.001 learning rate, and sparse categorical cross entropy loss function, the model was able to achieve a 18.45% loss on the validation data after training for 20 epochs. These parameters are either standard or given. The only significant changes are the learning rate and the number of training epochs. The choice to increase the learning rate by an order of magnitude was to allow faster training. I allowed the model to train for 20 epochs to allow for convergence, although the model started to overfit at around epoch 13. I tried to combat overfitting by adding a data augmentation layer of random horizontal flips, translations, contrast alterations, rotations, and zooming, but the model performed worse with a 21.76% error. I believe the reason was the data augmentation changed the inputs too much, losing information. Due to time constraints, I wasn't able to apply other overfitting measures or reduce image alterations in the data augmentation layer.

2.2 Semi-supervised learning

The final semi-supervised model uses the DenseNet201 [3]. With the same thought process in the supervised learning task, I thought a pre-model dense model would be able to represent the images better than the given baseline model. I tested NasNetLarge by Zoph et al [4], InceptionResNetv2 by Szegedy et al [5], VGG16 [2], EfficientNetB7 [1], and DenseNet201 [3]. The DenseNet201 model performed the best with a baseline validation accuracy score of 58% while the other models had accuracy percentages in the low 50s.

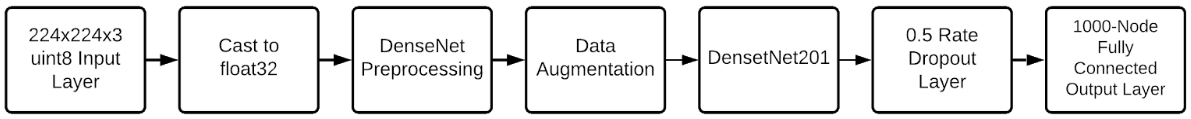

Figure 2: Core architecture of the semi-supervised learning model

The core model uses DenseNet201 without the top fully-connected layer and with global max pooling and initial ImageNet weights. Similar to the model in the first task, I added input layers and output layers to fit the data of the challenge. The final model also consists of a data augmentation layer of random horizontal flips, translation, contrast alteration, rotation, and zooming. The reasons for using these random image alterations and for choosing their parameters were more random than strategic. Unfortunately, I wasn't able to fine-tune these choices to determine the optimal implementation. However, these changes to the input images reduced overfitting and increased the validation accuracy of the model to about 62%. The model was trained for 100 epochs with the Adam optimizer and amsgrad turned on, learning rate of 0.0001, and the sparse categorical cross entropy loss function. The model's validation loss started to bounce in later training epochs, so I tried a decaying learning rate, but there seemed to be no changes in the performance of the model. I also tried to mitigate overfitting by adding L2 regularization to the dense layers, but the model worsened to an accuracy of 55%.

The final architecture uses the trained model to predict labels on the unlabeled datasets, adds the highly confident predicted labeled images to the training dataset, and trains the model for a second time. This self-training algorithm increased the accuracy of the model to 69%. The performance of the model can increase if I repeat the predicting and training steps in batches for many iterations, but I wasn't able to implement this feature due to time and resource constraints.

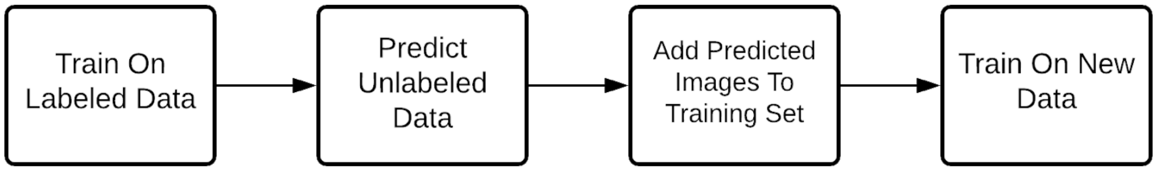

Figure 3: Self learning algorithm for my final submission

3. Conclusion and possible improvements

I was able to achieve 18.43% top 5 categorical error for the first task and 69.05% accuracy for the second task. The performance of my submission for the first task loses to the baseline performance by about 4 percentage points. My submission for the second task is worse, losing to the baseline performance by 10 percentage points. Without time and resource constraints, I believe I can improve my first task performance by trying overfitting measures such as applying L2 regularization, reduce the image changes in the data augmentation step, and increase the number of input images by data augmentation so the model may train on more images. For the second task, I can increase performance by using a generative adversarial network architecture or by changing the self learning algorithm to predict and train in batches.

References

[1] Mingxin.Tan *"EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks",* ICML 2019

[2] Karen.Simonyan *"Very Deep Convolutional Networks for Large-Scale Image Recognition",* ICLR 2015

[3] Gao.Huang *"Densely Connected Convolutional Networks",* CVPR 2017

[4] Barret.Zoph *"Learning Transferable Architectures for Scalable Image Recognition",* CVPR 2018

[5] Christian.Szegedy *"Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning",* AAAI 2017