Understanding the Amazon from Space with Deep Learning

Gejun Zhu, Yitao Zhai

August 04, 2017

Abstract

This document describes our team's 2nd place solution for the recent satellite image classification competition hosted by Planet [1]. The goal of the competition is to label satellite image chips with atmospheric conditions and various classes of land cover/land use. Our final result was based on ensembling 11 results from 7 different models. In the following sections, we will describe them in details.

Team member details

- Team Name: Plant
- Name: Gejun Zhu; Yitao Zhai
- Location: Texas, USA; Beijing, China
- Email: zhug3@miamioh.edu; longint2@163.com
- Competition: Planet - Understanding the Amazon from Space [2]

Contents

1 Data Preparation

There were two formats of images: JPG and TIF. We chose to use JPG format only because 1) we found some mismatched examples between JPG and TIF images with the same image id during the competition although this issue was fixed later by Robin; 2) our experiments showed that the performance of TIF images was worse than that of JPG images.

1.1 Data "Generation" and Augmentation

This was one of the most import parts in our solution. We generated images rotated by 90, 180, and 270 degrees, then treated them as new images in our training data. As can be seen from Figure 1, images rotated by these degrees would still have same labels. This meant that we had 4 times amount of images as of original images given. The reason why we did this was that the dataset was imbalanced. For example, 37513 images were labeled `primary`, while 98 images labeled `blow_down`. By generating rotated images, we can have more training data for rare labels. Also, we randomly flipped the image horizontally or vertically in some model. We didn't do any other augmentation to the images because cropping or shift augmentations will affect the images with labels as `partly_cloudy`, `road`, and `river`. This generated data will be considered as our training data in the following sections.

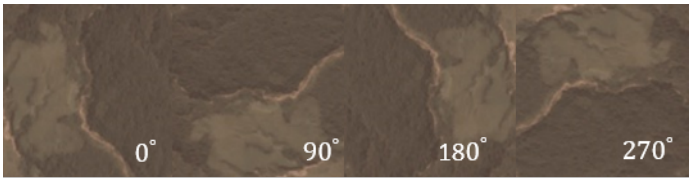

Figure 1: Generated image with 0, 90, 180, and 270 degree

1.2 Pre-processing

We didn't do anything special in pre-processing other than resizing images as required by different models and data normalization (divided by 255).

2 Models and Training

In this section, we will describe what pre-trained models we used and how we trained these models.

2.1 Network Architecture

Before we moved to pre-trained models, we tried multiple network architectures from scratch. But they didn't work as expected. The best f2 score from these models on the public leaderboard was 0.92441. Then we decided to stick to fine-tuning pre-trained models. Here is a list of network architectures we used in this competition:

- VGG[3]: VGG16, VGG19;
- ResNet[3]: ResNet50;
- Inception[3]: InceptionV3;
- Xception[3]: Xception;
- DenseNet[4]: DenseNet121, DenseNet169.

To fine tune these models, we removed the fully-connected layer from some pre-trained model and added our customized layers before the output layer. For models like Xception, Inception V3, DenseNet169, VGG16, VGG19, and ResNet50, we added two Dense layers (512 units and 1024 units) before the output layer. Also, the output layer was changed to 17 units Dense layer because we only had 17 labels in this competition.

As can be seen from Table 1, our best single model is DenseNet169 with image size 256*256. Note that we added the result from DenseNet121 with flip to the final ensemble without individual submission because of time limitation. So Public LB and Private LB scores were not available for this model.

2.2 Training

We used 5-folder cross validation strategy for each network architecture. This has been a popular strategy in kaggle competitions. The basic idea and major steps in our training part are shown as below:

1. Split training data into 5 folders;
2. Train each model on 4 folders of data and make predictions on the folder left and test data;
3. Average prediction probabilities for original image data;
4. Average different predictions for test data from each folder;

Models	Flip	Image size	Public LB	Private LB
Inception V3	No	156*156	0.93130	0.92983
Inception V3	No	256*256	0.93153	0.92961
Xception	No	128*128	0.93104	0.93014
Xception	No	256*256	0.93083	0.92936
VGG16	No	156*156	0.93162	0.93016
VGG19	No	196*196	0.93142	0.93025
DenseNet121	No	224*224	0.93137	0.92969
DenseNet121	Yes	224*224	×	×
DenseNet169	No	128*128	0.93111	0.92975
DenseNet169	No	256*256	0.93229	0.93129
ResNet50	No	208*208	0.93178	0.93027

Table 1: Performance overview for different models

5. Optimize f2 threshold based on predictions on original image data and generate test data labels.

Another important part in our solution is that we trained the networks with Adam [5] optimizer and customized learning rate scheduler. In the first 2 epochs, learning rate was set to 0.0001, and then decreased to 0.00005 in the next 20 epochs for all models except Inception V3. In Inception V3, the learning rate was set to 0.001 in the first 2 epochs, and then decreased to 0.0001 in the next 20 epochs. To combat with overfitting, we employed early stopping and model check point strategy based on the performance on the validation data during training. This was to say, if the model didn't improve in the last 2 epochs, the training stopped and saved the best model. In addition, `binary_crossentropy` loss function was used since this was a multi-label classification problem.

3 Model Ensembling

In this section, we will describe how we made predictions and combined results from different models.

3.1 Test-time Augmentation (TTA)

This was last important part in our solution. We learned about TTA strategy from one winning solution for National Data Science Bowl [6] by Deep Sea and also from the discussion [7] shared by Heng CherKeng [8]. Basically, we computed predictions across various augmented versions of input images and averaged them.

3.2 Threshold Selection

Since we generated 3 more rotated images for each original image, we would have 4 predictions for each original one. Therefore, to obtain the probabilities of each label of original image, we need to average these 4 predictions. Then we used these probability predictions to find best thresholds which maximize f2 score. We actually searched for best threshold of each label instead of one threshold for all labels. This idea was adopted from the discussion [9] by @anokas [10]. Here is the pseudo-code for threshold selection for each label.

Algorithm 1: Threshold Selection

```
OptimizeF2Threshold(y_true_prob, y_pred_prob, n):
for i ← 0 to 16 do
    initialize thresholds as [0.2]*17;
    initialize best_f2 as 0;
    for j ← 0 to n do
        thresholds[i] ← j/n;
        y_pred_label = y_pred_prob > thresholds;
        curr_f2 = f2_score(y_true_label, y_pred_label);
        if curr_f2 > best_f2 then
            thresholds[i] = j/n;
        end
    end
    return thresholds
end
```

3.3 Combining different models

After predicting probabilities and obtaining threshold of each label, we averaged predictions and thresholds across all models. We used equal weight when averaging since weighted averaging performed worse. Because different network architecture can capture different features of the image, the performance improved by quite a large margin.

4 Code Description

4.1 Yitao's model part

- `avg_all.py` - This file is used to average all probability predictions from multiple models.
- `custom_layer.py` - This file defines scale layer for DenseNet169.
- `densenet169.py` - This file defines network architecture for DenseNet169.
- `find_best_thre.py` - This file is used to find best threshold for each label to optimize f2 score.
- `gen_cv.py` - This file is used to generate data for 5 folder cross validation.
- `gen_test.py` - This file is used to generate testing data.
- `multi_gpu.py` - This file is used to make training process parallel with multiple GPUs.
- `predict_test.py` - This file is used to make predictions on testing data with TTA strategy.
- `run.sh` - This file is used to generate data, train model, find best thresholds, make predictions, and average all results.
- `train.py` - This file is used for model training.

It will take a week or so to run all the models. We provided a sample training process `run_sample.sh` which will require less than 24 hours to run for VGG19 with image size 156*156.

4.2 Gejun's model part

- `settings.py` - This file includes some basic path setup including `BASE_DIR`, `DATA_DIR`, `TRAIN_IMG_DIR`, `TEST_IMG_DIR`, `OOF_DIR`, `WEIGHT_DIR`.
- `scale_layer.py` - This file defines scale layer for DenseNet121.

- `create_n_folder.py` - This file is used to create training and validation data in `oof` folder. It will generate files like `tra_data_*.csv`, `val_data_*.csv` which will be used to determine which images with what rotation angle are used in training and which are in validation.
- `train_n_folder_w_flip.py` - This file is used to train the network DenseNet121, generate predictions for out of folder, and save the weights for best model to `weights` folder. When loading training images, it includes random flip augmentation.
- `train_n_folder_no_flip.py` - This file does the same thing as `train_n_folder_w_flip.py` except that there is no flip augmentation when loading the training images.
- `multiple_preds_for_test.py` - This file is used to load the weights saved from model training and make predictions on test images data.
- `run_model_gejun.sh` - This file is used to create cross validation data as needed, run all 5 folder cross validation for `train_n_folder_w_flip.py` and `train_n_folder_no_flip.py`, and prepare data for final ensemble.
- `utils.py` - This file defines `f2_score` function to calculate f2 score, `f2_socre_k` which was used to evaluate model performance when training, and `optimise_f2_thresholds` to find best threshold for each label.

This model will take 3 days around to run based on the hardware dependency provided.

5 Dependencies

5.1 Yitao's model part

5.1.1 Hardware

- 128G RAM
- 4 × Tesla K40c

5.1.2 Software

- CentOS Linux release 7.3.1611 (Core)
- Python 2.7
- Keras == 2.0.5
- tensorflow-gpu == 1.1.0
- opencv-python == 3.2.0.7
- numpy ==1.13.1
- pandas == 0.20.1

5.2 Gejun's model part

5.2.1 Hardware

- 64G RAM
- Nividia GTX 1080 Ti

5.2.2 Software

- Ubuntu 16.04
- Keras == 2.0.4
- tensorflow-gpu == 1.2.1
- opencv 3 == 3.1.0
- pandas == 0.20.1
- numpy == 1.13.1
- CUDA 8.0

6 How To Generate the Solution

6.1 Gejun's model part

1. Download data from competition website. Put all training images in `train-jpg` folder, and testing images in `test-jpg` folder. Also, put `train_v2.csv` in `data` folder.
2. Download pre-trained weight `densenet121_weights_tf.h5` from `https://github.com/flyyufelix/cnn_finetune`, and put them in `weights` folder.
3. Set working directory path in `settings.py`.
4. Run `./run_model_gejun.sh`.

This will generate two predictions from DenseNet121 in directory `BASE_DIR`. Move `densenet121_*_cv.csv` and `densenet121_*_fulltest.csv` to Yitao's `cv` folder for furtuer ensemble.

6.2 Yitao's model part

1. Download data from competition website into `input` folder. Put all training images in `train-jpg` folder, and testing images in `test-jpg` folder. Also, put `train_v2.csv` and `sample_submission_v2.csv` in `input` folder.
2. Download pre-trained weight `densenet169_weights_tf.h5` from `https://github.com/flyyufelix/cnn_finetune`, and put them in the work directory folder.
3. Go to `src` folder and run `./run.sh`.

It will generate a submission file named `ensemble11.csv` in `sub` folder.

7 Acknowledgment

We would like to thank Kaggle team and Planet for organizing this competition. Also, we would like to thank authors of Keras since Keras API was very easy to use.

References

[1] https://www.planet.com/
[2] https://www.kaggle.com/c/planet-understanding-the-amazon-from-space
[3] https://keras.io/applications/
[4] https://github.com/flyyufelix/DenseNet-Keras
[5] https://arxiv.org/pdf/1412.6980.pdf
[6] https://github.com/benanne/kaggle-ndsb
[7] https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/discussion/33559
[8] https://www.kaggle.com/hengck23
[9] https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/discussion/32475
[10] https://www.kaggle.com/anokas