3rd place Solution

posted in Planet: Understanding the Amazon from Space 5 months ago

30

Solution summary

Software

- Windows 10, Python 3.4 + 3.5
- Keras 1.2.1 - main framework for CNN training
- Theano 0.9.2 - main backend
- Tensorflow - additional backend for neural nets which is not supported by Theano (ResNet152 and XCeption)
- XGboost, LightGBM, Keras - main classificators for final ensemble
- OpenCV - image preprocessing

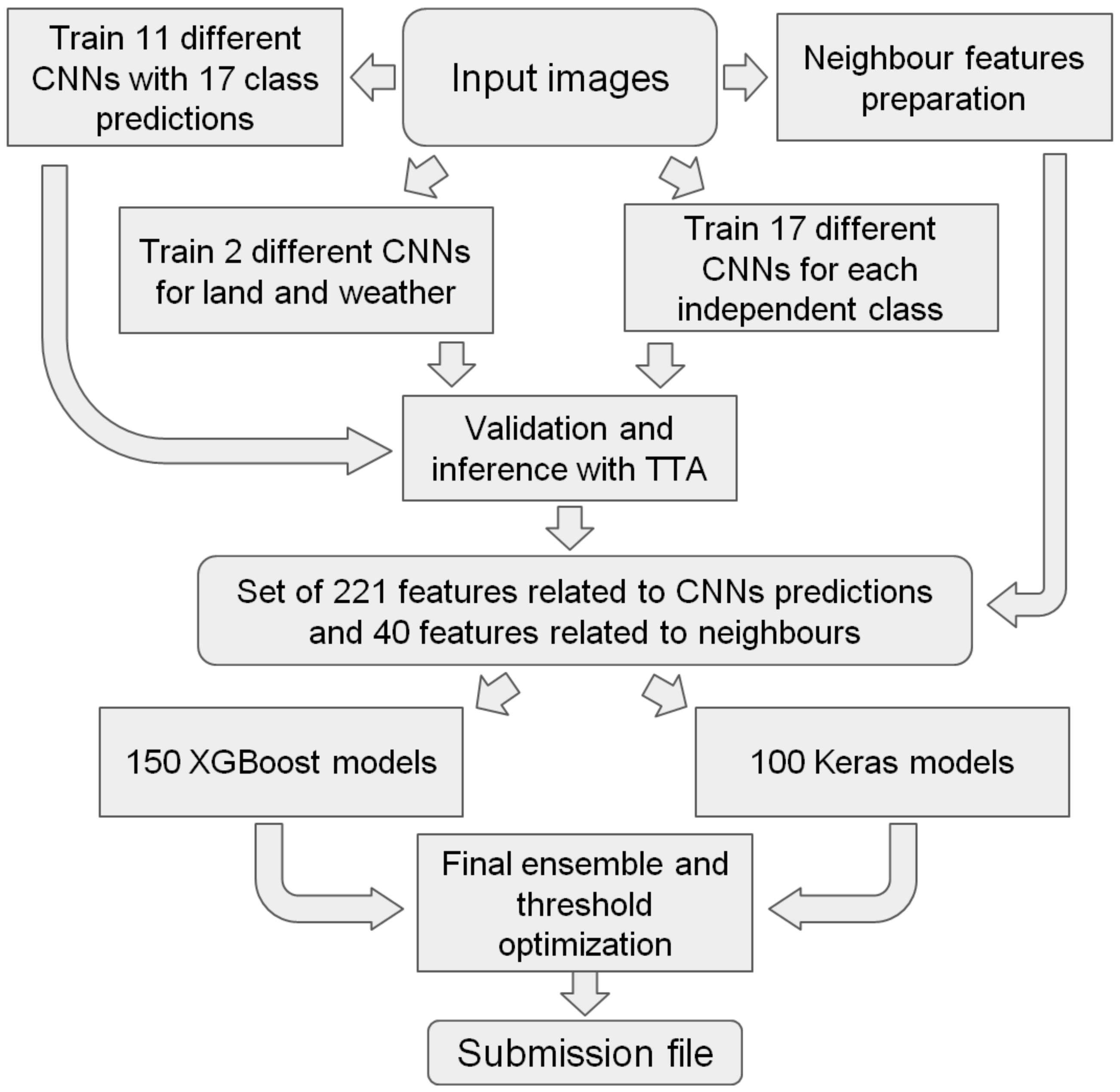
Hardware

- We used 6 GPUs: 2*1080Ti + 4*1080.
- Parallelization was made using either different fold on different GPUs, or different CNNs on different GPUs.

Training Time

- On single GPU around a month for full pipeline. On 6 GPUs it’s possible to finish in around 1 week.

Dataflow


Train 11 different CNNs with 17 class predictions
Input images
Neighbour features preparation
Train 2 different CNNs for land and weather
Train 17 different CNNs for each independent class
Validation and inference with TTA
Set of 221 features related to CNNs predictions and 40 features related to neighbours
150 XGBoost models
100 Keras models
Final ensemble and threshold optimization
Submission file

Set of CNN based models

Due to problem with TIFFs all CNN models used only JPEG images

- 11 models for different CNNs with 17 output neurons with sigmoid activation. Set of CNNs: INCEPTION_V3, INCEPTION_V4, DENSENET_121, DENSENET_169, DENSENET_161, RESNET50, RESNET101, VGG16, VGG19, RESNET152, XCEPTION
- 1 DENSENET_121 model with weather classes only: ['clear', 'partly_cloudy', 'haze', 'cloudy']. “Softmax” activation because classes is mutually exclusive - 4 output neurons.
- 1 DENSENET_121 model with land classes only - 13 output neurons.
- 17 DENSENET_121 models for single classe. Each model has 1 output neuron to predict single class. 50/50 batches, where half of images have class presented.

We used 5 KFold Cross Validation. So each model has 5 different weights set. 30 CNN models in total. 150 weight files

Training Process and Data Augmentation

- Keras module for Python with Theano or Tensorflow backend.
- Typical batch size around 20 images for small nets, 16 images for large nets due to GPU memory limitation
- Batches created on the fly, so no need to store all the data in memory
- As optimizer we used Adam, with learning rate ~0.00003
- We used “logloss” loss function, since direct usage of F2-score in training process gave worse results.
- Single class models (for rare classes) used 50/50 batches, where half of images have class presented

Data augmentation includes

- Random crops: chose some part of picture and resize it to CNN input shape (224x224 or 299x299)
- Random mirrors or 90 degrees rotations (8 possibilities in total)
- Random intensity changes

Validation and Test Time Augmentation (TTA)

- We used 5 KFold, so to process Train images for validation we process it one time with corresponding fold model. In total 40K images
- To process test images each image must be processed with each fold model. 5 times in total, so for test data we process 300K images.
- To increase the accuracy we used TTA. This mean we process each image several times with some changes. We used 32 variation of single image, predict each and use mean as final prediction: -- For Train images 1.2M predictions -- For Test images 9.6M predictions

After this we got 2 text CSV files with probabilities for each image to have predicted class. One CSV file for train images, second for test images. These files used later in second level classifiers.

Neighbors

More details in this thread: https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/discussion/36738

Based on neighbours IDs, panorama IDs and CNN predictions we created the following set of features:

- Average CNN predictions for 4 neighbours
- Average CNN predictions for 8 neighbours (including diagonal elements)
- Average CNN predictions for whole Panorama ID

Each image got “panorama ID” and “panorama Size“ as features for second level models or “N/A” depending on following rules:

- panorama must has more than one element
- panorama must contain at least one element from train and at least one element from test parts of data

2nd level models: XGBoost blender

We have large amount of CNN predictions and features. Now we need to some methods to join them altogether for more accurate predictions. One of methods is to use many random XGboost runs with random parameters.

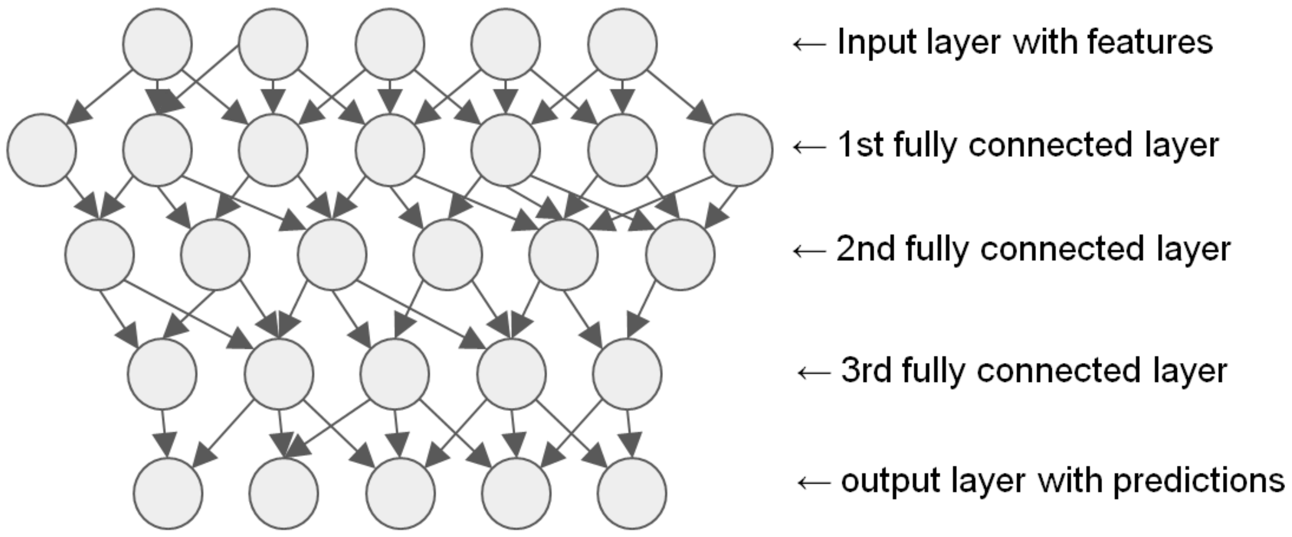
XGboost can predict only one class at the time. So each run create 17 different XGBoost models, each predict its own class. On the input we had ~300 features. What we variate from run to run:

- Folds number [4 - 10]
- Learning rate [0.06 - 0.45]
- Max Depth [2 - 5]
- Subsample [0.6 - 0.99]
- Colsample by tree [0.6 - 0.99]

2nd level models: Keras blender

Keras neural net classification models are good for following reasons:

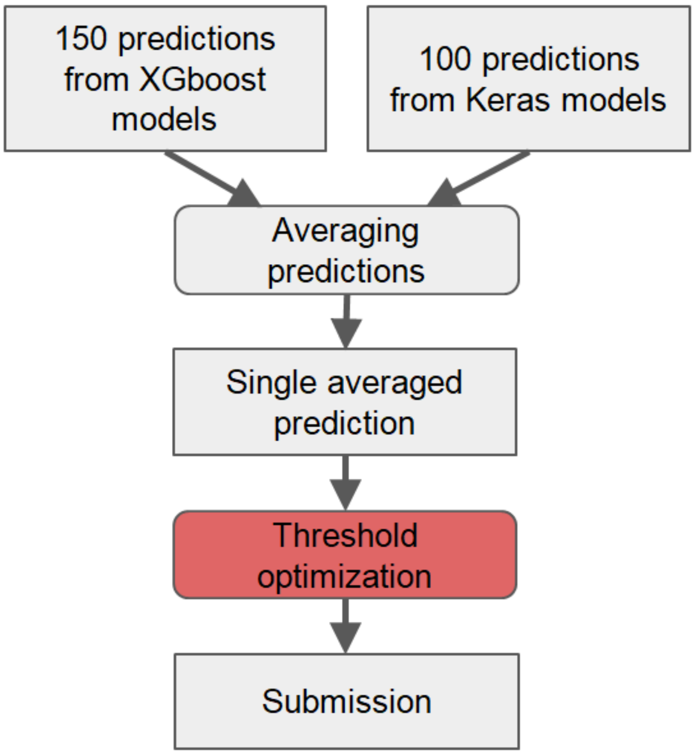
- They’re the very different from XGboost models
- They can predict all 17 classes at once, catching the mutual influence of classes The bad thing is that it in most cases has slightly worse accuracy than XGboost or LightGBM.

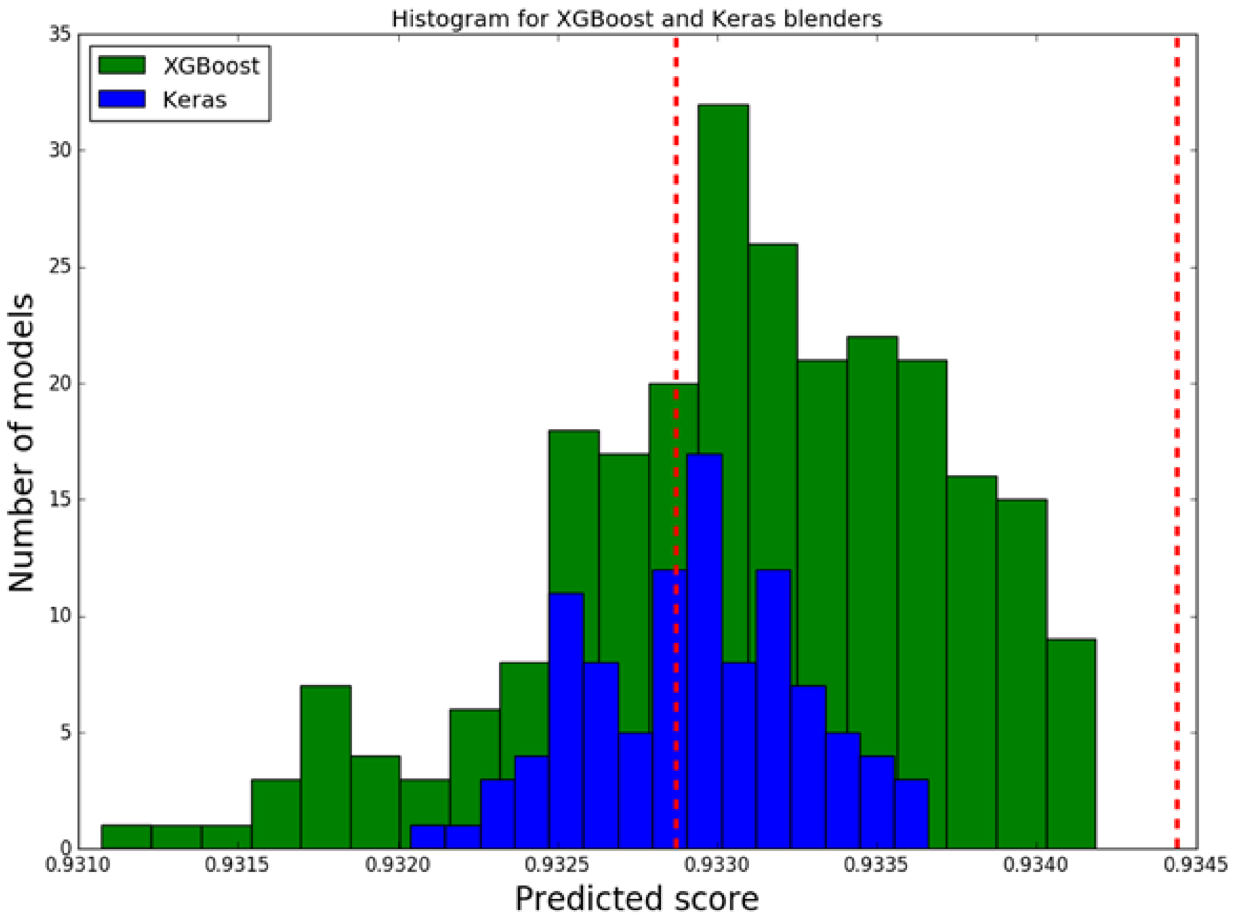

← Input layer with features
← 1st fully connected layer
← 2nd fully connected layer
← 3rd fully connected layer
← output layer with predictions

Variation parameters:

- Number of neurons on each level: 1st: 400 - 700, 2nd: 350 - 600, 3rd: 200 - 500
- Dropout value
- Random choice of activation: ’RELU’, ‘ELU’, ‘PRELU’
- Batch size: 200-1000
- Learning rate: 1e-5 - 1e-3
- Early stopping: 50 - 150
- Number of folds: 4 - 10

Final Ensemble


150 predictions from XGboost models
100 predictions from Keras models
Averaging predictions
Single averaged prediction
Threshold optimization
Submission



It's possible to get 2nd place with this solution:

Submission and Description	Private Score	Public Score
merger_final_0.9347088497635784.csv a month ago by ZFTurbo XGBoost 150 + Keras 80 Blenders - standard ensemble	0.93302	0.93435

Code

Github repo

Dataflow.png (72.07 KB)
Score1.png (35.02 KB)
Ensemble.png (23.08 KB)
Keras-blender.png (46.45 KB)
2nd-place.png (9.63 KB)

Comments (6) Sort by Oldest

MosMK • 5 months ago • Options • Reply 0

Thank you. Can I ask how did you choose the learning rates in a02_zoo.py?

ZFTurbo • 5 months ago • Options • Reply 1

Trial and error. Also based on previous experience. I think LR can be tuned further for faster performance with same accuracy.

MosMK • 5 months ago • Options • Reply 0

Just another question, why do you still use keras 1 and not 2?

ZFTurbo • 5 months ago • Options • Reply 1

There are many syntax and parameters changes in Keras 2.0. Not all the networks worked fine in 2.0 version while I solve Amazon problem. But I plan to move on Keras 2.0 in next projects, since all CNNs now fixed to work with 2.0.

TomaszGrel • (10th in this Competition) • 5 months ago • Options • Reply 0

Thanks for the detailed description. What kind of threshold optimization did you use?

ZFTurbo • 5 months ago • Options • Reply 0

There was some modification of bisection method which was performed on validation data. Obtained thresholds then used for test data.

On first step we search for single threshold for all classes. On the second step search optimal threshold independently for each class, while fix all other thresholds. Step 2 can be repeat several times since it keep increasing accuracy.