Kaggle TGS Salt Identification Challenge. First Place Solution

General Information

Competition Name: TGS Salt Identification Challenge
Team Name: b.e.s. & phalanx
Private Leaderboard Score: 0.896
Private Leaderboard Place: 1st

Team members:

b.e.s. (https://www.kaggle.com/ybabakhin)
Name: Yauhen Babakhin
Location: Minsk, Belarus
Email: y.babakhin@gmail.com

phalanx (https://www.kaggle.com/phalanx)
Name: Hirotoshi Kitamura
Location: Osaka, Japan
Email: ritskitamura@gmail.com

Previous Background

b.e.s.

I have got Master's Degree in Applied Data Analysis from Belarusian State University last year. Also, I have over 3 years working experience at Data Scientist positions: 2 years at Wargaming.net (working with classic ML) and 1 year at Profitero (primarily working on NLP problems). I'd already had Kaggle Competitions Master tier before this competition.

Before entering this competition I knew nothing about image segmentation. So, the reason to participate was to gain knowledge and experience in this type of problem. After fulfilling this goal, I found myself in the range of top-20 teams and decided to compete till the end of the competition.

In the early stage of the competition I was building pipeline from scratch, that's why it took about 20 hours per week. Further, I was just running experiments and implementing some new ideas (it took about 10 hours per week). Thus, overall time spent is about 180 hours.

phalanx

I have got Bachelor's Degree in Computer Science from Ritsumeikan University last year. And I am majoring in computer vision at the graduate school.

I have been studying action detection since last year, so I understand deep learning to a certain extent. Also, I have read papers on semantic segmentation for my research, so I had knowledge for this competition. But I had never worked on semantic segmentation problem. So, I decided to participate in this contest in order to gain experience and knowledge about semantic segmentation problem.

I spent 20 hours a week from the start of the competition. So, overall time is about 240 hours.

b.e.s. & phalanx

Two weeks before the competition deadline we realized that it would be hard to keep ourselves in the gold medals alone. At that moment we were both in top-10 and decided to team up. We were building own pipelines completely independently. Further, *phalanx* was responsible for building models based on two-stage pseudolabels. *b.e.s.* was responsible for generating pseudolabels and final postprocessing.

Solution Summary

Our solution is based on modifications of U-Net architecture with pretrained encoders (ResNet34 and ResNeXt50). The most important features, from our point of view, are custom U-Net architectures and multi-stage pseudolabeling scheme.

Our training process consists of 3 stages. On the first one we create models based on train data only, and images resized to the higher resolution (101x101 to 256x256). Second and third stages use both initial train data and pseudolabels generated based on test data.

As the final model, we use simple average of two models and apply postprocessing based on mosaics generated from 101x101 crops.

The overall time to train the final model from scratch on a single GTX1080Ti is about 16 days. Time to get predictions for 18,000 test images is about 3.5 hours. Considering frameworks, b.e.s. is using Keras, and phalanx is using PyTorch.

The next sections are organized as follows: Pipeline Overview gives the overall idea of our solution, Interesting Findings lists the most important facts used in the solution, Training Time describes the amount of time needed to utilize the model, Simpler Model shows simpler approach with a comparable quality, and Detailed Solution describes our solution in-depth.

Pipeline Overview

Our pipeline consists of three major stages. All the models are custom modifications of the vanilla U-Net architecture. During the prediction time for each of the models we're using test time augmentations (TTA) with Horizontal Flip: simple average of the original test predictions and their flipped copies.

Stage 1

We take 4,000 train images and each of us creates a single model predicting segmentation masks (input size for the first model is 224x224, for the second -- 256x256). Architecture and training methods are available in Detailed Solution. Simple average of these two models gives 0.885 in Private LB (about 45th place).

As long as test data contains 18,000 images (x4.5 times greater than train), we decide to use the idea of Pseudolabeling. It's semi-supervised learning method and it showed great results in past competitions as well as in production solutions, when one doesn't have much labeled data.

So, we treat the predictions of our ensemble on test images as the new training data.

Stage 2

Second stage models are trained from scratch, so they do not use any weights from Stage 1 except for pseudolabels data. Here we apply two most common strategies to utilize pseudolabels:

1) Pretrain model using pseudolabels only. Then use weights obtained and continue training the model only on the train data. Such strategy allows to create good weights initialization for our specific data, which implies faster convergence and better results.
2) Add confident pseudolabels to the train data. Such strategy allows to expand the size of the training data.

For this particular problem, the first strategy works better for us. Again, average of these two models gives 0.891 Private LB (about 15th place).

Similarly to the Stage 1, we create new pseudolabels using the model with better quality. Thus, improving the quality of pseudolabels.

Stage 3

Third stage model is also trained from scratch, so it do not use any weights from previous Stages except for pseudolabels from Stage 2.

Here we only use the first strategy pretraining the model with pseudolabels. Moreover, now we're using 128x128 images for training. This model alone gives 0.895 Private LB (about 4th place).

Final Model

Final model is a simple average of one model from Stage 2 and one model from Stage 3. Such ensemble gives 0.896 Private LB (1st place). Also, we're using postprocessing based on jigsaw mosaics from 101x101 crops (initial code and idea are described here). The idea is to expand known vertical masks to the test predictions. It gives considerable improvement in Public LB, but almost no improvement in Private LB

Interesting Findings

As already said, the major factor to achieve the high score is gained through **multi-stage pseudolabeling** train scheme. However, there are also a couple of ideas made model better.

As long as images have small size of 101x101 pixels, bottom U-Net layers have very small spatial resolution. That's why **we resize and pad images to larger size** (e.g. 256x256) to increase the size of bottom layers. It gave a considerable improvement in the early stage of the competition. Another intuition about the padding is that is helps to increase the size of small masks near the borders.

Attention Mechanism is a very popular tool in modern architectures. In this competition we use **Concurrent Spatial and Channel Squeeze & Excitation** (scSE) approach. This module is built into both encoder and decoder. The idea is to calibrate the feature maps adaptively, to boost meaningful features, while suppressing weak ones.

Another attention module we're using is **Feature Pyramid Attention** (FPA). This module fuses features from under three different pyramid scales by implementing a U-shape structure like Feature Pyramid Network.

Another powerful idea of utilizing feature maps from different scales is Hypercolumns. Instead of using last layer as a feature representation, one could stack layers from multiple stages in the decoder. It allows to both make precise localization and capture semantics.

Considering the training process, we're actively using Snapshot Ensembling with Cosine Annealing LR (from 3 to 6 cycles). It allows to train a single model and then average multiple local minimas, which improves model quality and brings robustness.

Additionally, the best performing loss function was Lovasz loss. It helps to directly optimize the IoU measure. However, the best epochs of our models were chosen using the LeaderBoard metric.

Finally, a couple of days before the competition deadline we realized that we're predicting only the borders between earth and salt. So, **full-body masks are treated as empty in the train set**. Surprisingly, our models were pretty good at predicting such masks, so we didn't make any special tweaks to incorporate this knowledge.

Training Time

The overall time to train all three stages from scratch on a single GTX1080Ti is about 16 days. We have pretty limited computational power, that's why we do not use larger encoders:

1) b.e.s. has a single GTX1080
2) phalanx has a single GTX1080Ti and got another one only during the last week of the competition

Predicting 18,000 test images using final models takes about 3.5 hours (i.e. about 0.7s per image). Such long prediction time is explained by the fact that we're using 5-folds and multiple snapshots: there are 38 individual models in a final ensemble.

Simpler Model

As a simpler model with quality 0.885 in Private LB (99% of the final 0.896 score) we could take the Stage 1 ensemble. It doesn't use any pseudolables, and it's just a simple average of 2 models on 5 folds and multiple snapshots.

However, to train such a model one still needs about a week on a single GTX1080Ti. The prediction time is similar to the final model (about 3.5 hours on 18,000 images).

Detailed Solution

Local Validation

We use 5-fold cross validation stratified by depth. Score on local validation has pretty solid correlation with the Public LeaderBoard.

Training Process

Our training process consists of 3 stages. On the first one we create models based on train data only. Second and third stages use both initial train data and generated pseudolabels based on test data.

During the prediction time for each of the models we're using TTA with Horizontal Flip.

1st Stage Training

Each of us develop single model based on training data only. b.e.s. additionally excludes from training data 'black-square' images (80 images with all black pixels and empty masks).

Model 1:

- *Input size:* 101x101 -> resize to 192x192 -> pad to 224x224

- *Architecture:* U-Net with pretrained encoder. Encoder: ResNext50, weights are available here. Decoder: 3x3 transposed convolutions + Batch Normalization + scSE module for Attention
- *Training overview:* optimizer is RMSprop, batch size is 24.
    1. Loss: Binary Cross Entropy + Dice. Reduce Learning Rate on plateau starting from 0.0001
    2. Loss: Lovasz. Reduce Learning Rate on plateau starting from 0.00005
    3. Loss: Lovasz. 4 snapshots with cosine annealing LR, each snapshot consists of 80 epochs and LR starts from 0.0001

Model 2:

- *Input size:* 101x101 -> resize to 202x202 -> edge padding to 256x256
- *Architecture:* U-Net with pretrained encoder. Encoder: ResNet34 (remove first max pooling) + scSE, weights are available here. Middle block: Feature Pyramid Attention (remove 7x7 block). Decoder: 3x3 transposed convolutions + Batch Normalization + scSE module + Hypercolumns.
- *Training overview:* loss is Lovasz, optimizer is SGD, batch size is 18.
    1. 6 snapshots with cosine annealing LR, each snapshot consists of 50 epochs and LR from 0.01 to 0.001

Performance:

- 5-fold Model 1 has 0.864 Public LB (0.878 Private LB)
- 5-fold Model 2 has 0.863 (0.880 Private)
- Their ensemble scores 0.867 (0.885 Private)

As a result from Stage 1, we have test predictions for 18,000 images. We use these predictions as pseudolabels for the next-stage models. Moreover, we take only confident predictions.

Denote pixel prediction as confident if its probability is less than 0.2 or greater than 0.8. The confidence of the prediction is measured as the percentage of confident pixels in the image.

2nd Stage Training

Second stage models are trained from scratch, so they do not use any weights initialization from Stage 1 except for pseudolabels data.

Model 3:

- *Input size:* 101x101 -> resize to 192x192 -> pad to 224x224
- *Architecture:* U-Net with pretrained encoder. Encoder: ResNext50, weights available here. Decoder: 3x3 transposed convolutions + Batch Normalization + scSE module for Attention
- *Training overview:* optimizer is RMSprop, batch size is 24.
    1. Pretrain model with pseudolabels. Loss: Binary Cross Entropy + Dice. 1 snapshot with 80 epochs and cosine annealing LR starting from 0.0001

2. Continue training on pseudolabels only. Loss: Lovasz. 1 snapshot with 80 epochs and cosine annealing LR starting from 0.00005
3. Finetune only on training data. Loss: Lovasz. Reduce Learning Rate on plateau starting from 0.00005
4. Loss: Lovasz. 3 snapshots with cosine annealing LR, each snapshot consists of 40 epochs and LR starts from 0.00005

Model 4:

- *Input size:* 101x101 -> resize to 202x202 -> edge padding to 256x256
- *Architecture:* U-Net with pretrained encoder. Encoder: ResNet34 (remove first max pooling) + scSE, weights available here. Middle block: Feature Pyramid Attention (remove 7x7 block). Decoder: 3x3 transposed convolutions + Batch Normalization + scSE module + Hypercolumns + Deep Supervision.
- *Training overview:* The confident pseudolabels are added to each of the folds. Losses are Lovasz for segmentation and Binary Cross Entropy for classification, optimizer is SGD, batch size is 18.
    1. 4 snapshots with cosine annealing LR, each snapshot consists of 50 epochs and LR from 0.01 to 0.001

Performance:

- 5-fold Model 3 has 0.871 Public LB (0.890 Private LB)
- 5-fold Model 4 has 0.861 (0.883 Private)
- Ensemble of Model 3 and Model 4 scores 0.870 Public LB (0.891 Private)

As a result from Stage 2, we have test predictions for 18,000 images. We use all these predictions as pseudolabels for the 3rd stage model.

3nd Stage Training

Third stage model is trained from scratch, so they do not use any weights initialization from previous Stages except for pseudolabels data.

Model 5:

- *Input size:* 101x101 -> edge padding to 128x128
- *Architecture:* U-Net with pretrained encoder. Encoder: ResNet34 (conv7x7 to conv3x3 and remove first max pooling) + scSE, weights available here. Middle block: Feature Pyramid Attention (remove 7x7 block). Decoder: 3x3 transposed convolution + Batch Normalization + scSE module + Hypercolumns.
- *Training overview:* loss is Lovasz, optimizer is SGD, batch size is 64.
    1. 4 snapshots with cosine annealing LR, each snapshot consists of 50 epochs and LR from 0.01 to 0.001

Performance:

- 5-fold Model 5 has 0.874 Public LB (0.895 Private LB)

Final Model

Final model is a simple average of **Model 3** and **Model 5** with Horizontal Flip Test Time Augmentations: 0.876 Public LB (0.896 Private LB).

Augmentations

We are using the following list of augmentations with different parameters:

- Horizontal Flip
- Random Shift
- Random Scale
- Random Rotate
- Random Brightness
- Random Contrast

Moreover, during test time we're using Horizontal Flip augmentation: taking the average of the predictions on original test image and augmented test image.

Postprocessing

We use postrpocessing based on jigsaw mosaics available in this kernel: https://www.kaggle.com/vicensgaitan/salt-jigsaw-puzzle. Denote non-empty mask as 'vertical' if all its vertical stripes are the same. The idea of postprocessing is the following:

1. Find all vertical and half-vertical (bottom half of the mask is vertical) masks in train data
2. All test images below them in mosaics get the same mask
3. Only one test image above them get the same mask, and only if its depth in mosaic is greater or equal to 3

Also, we assign empty masks for the 'black-square' images (images with all black pixels).

It gives huge boost on Public LB and no boost on Private LB: 0.876 -> **0.884 on Public LB** and 0.896 -> **0.896 on Private LB**.