CS583A: Course Project

Erika Deckter

December 2, 2019

1 Summary

I participated in a Kaggle competition to identify types of cloud organization in satellite images. The final model was a Mask R-CNN model using a ResNet101 backbone. The original 1,400 x 2,100 images were converted to 1,024 x 1,024 images (with zero padding for the height dimension). The model predicted either one of the four cloud type classes or a null class (background) for each pixel in the image. Each image could have one or more of the cloud types (up to all four types); however, no pixel could belong to more than one cloud type. The model was implemented using the Matterport implementation of the Mask R-CNN model, built in Python and Keras with a TensorFlow backend. All additional code was written in Python and Keras. The code was run on a Windows laptop with 2 CPUs and 16 GB of RAM and on Google Colab notebooks with an accelerated GPU backend. The competition metric was the mean Dice coefficient for each image and class pair:

$$\text{Dice coefficient} = \frac{2*|X \cap Y|}{|X|+|Y|}$$

By definition, if an image did not contain a particular cloud class in both the actual and prediction masks, the Dice coefficient for that class was defined as 1.0. My best Kaggle submission was a 0.136 public score and a 0.136 private score. This was based on the Fully Convolutional Network (FCN) model, not that Mask R-CNN model, which performed much better on the validation data. My Kaggle score is unfortunately not very good, not even beating the random baseline. I believe this is due to a combination of not having the best hyperparameters for the FCN model (due to very long training times, even using a GPU on Colab) and issues with the post-processing step that first transformed the model prediction masks to 350 x 525 sizes and encoded the pixel values into starting pixel and run length values for each class. I was not able to get the post-processing step to work for the Mask R-CNN output (which is why I do not have a Kaggle score for that superior model).

2 Problem Description

Problem. The problem was to identify types of cloud organization from satellite images. This was part of a Kaggle competition that can be found here: `https://www.kaggle.com/c/understanding_cloud_organization`. There were four cloud organization types to be identified in the images: fish, flower, gravel and sugar. Each image could have one or more types of cloud organization (up to all four), but there was no overlap in cloud organization types (i.e., a pixel could not have more than one label).

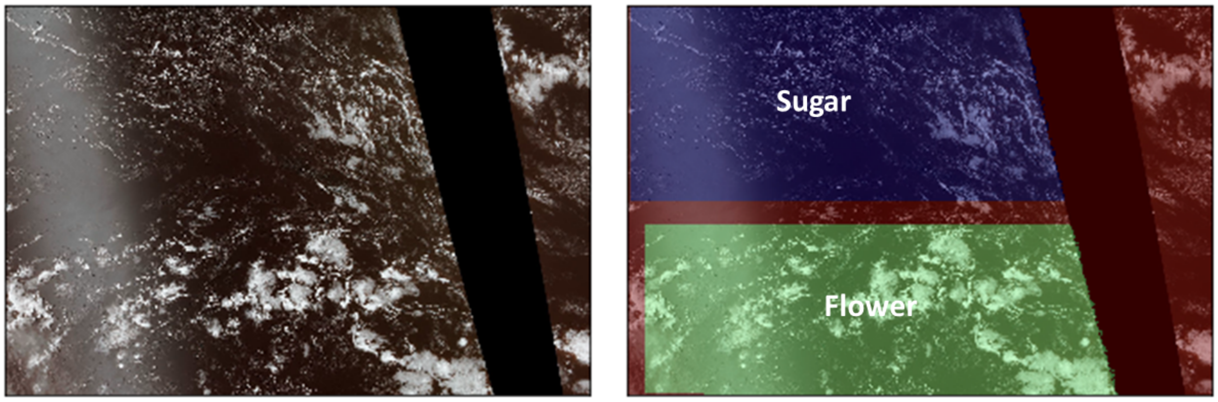


Figure 1: (Left) Satellite Image from the Training Dataset; (Right) Satellite Image Overlaid with Labeled Mask

Data. The training data consisted of 5,546 training images and 3,698 test images in JPEG format. Each image is 1,400 x 2,100 pixels. In addition, there was a file called "train.csv" which had encoded pixel labels for each training image. The "train.csv" file had two columns: Image_Label and EncodedPixels. The Image_Label column had the image name and the type of cloud formation for that row (e.g., 0011165.jpg_Sugar). The EncodedPixels column had a list of start pixel and run length pairs. Each pixel was encoded as an integer value from 1 to 2,9400,000. The pixels were numbered from top to bottom and then left to right, so pixel 1 was the first pixel in row 1, and pixel 2 was the first pixel in row 2, etc. As an example, a row that had values of "10 2 15 5" for the "Fish" label would mean that pixels 10 and 11 and pixels 15, 16, 17, 18 and 19 should all be labeled as "Fish." There were 4 possible labels for each pixel; however, I used 5 classes when building my model. The 5th class was a "null" class, which represented a background class (i.e., a pixel that did not belong to any of the cloud formation types).

Challenges. Image segmentation is a challenging problem. Before convolutional neural networks, most image segmentation methods were unsupervised (such as thresholding or k-means clustering), which grouped similar pixels together based on intensity or other metrics, but did not allow for image segmentation algorithms to differentiate based on pre-specified classes.

Specific challenges I ran into while working on this project are outlined below:

- Mask Pre-processing: The encoded pixel values needed to be translated into image label masks of the same size as the images. I solved this problem by writing a custom Python script to parse the csv file and create masks saved as numpy arrays.
- Image and Mask Alignment: The next challenge was to load the training images and masks so they could be used to train the model. The standard Keras ImageDataGenerator could not be used because I needed to load an image and its corresponding mask at the same time and perform the same data augmentation to both the image and the mask. In addition, I needed to convert the mask from flat labels (0 through 5) to a one-hot encoded tensor with

5 channels (one for each label). For the FCN model, I wrote a custom instance of the Keras Sequence class to load the data and perform the same augmentations on the image and mask. For the Mask R-CNN model, I wrote a custom instances of the Dataset and Config classes.

- Training Time: Due to the size of the convolutional neural networks being used, training took a significant amount of time. In addition, Google Colab sped up training when it worked, but often crashed or kicked me off the GPU servers during training. This unfortunately limited the amount of model variations I could realistically test and the number of epochs I could run for each model.

- Post-Processing for Kaggle Submission: The prediction masks needed to be converted back to the encoded pixel format for submission to the Kaggle competition. In addition, the predictions had to be sampled down to 1/4 of the original size of the images (325 x 525). In order to use the pre-trained models, the original images had to be resized, and then they needed to be resized again for the Kaggle submission. This continual resizing may have lead to translation problems in the actual results versus what was reported in the Kaggle submission file. In addition, creating this file for the nearly 4,000 test images took a significant amount of time for each iteration of the model.

3 Solution

Model. The final model was a Mask R-CNN as described in the 2018 paper by He et al. [7].

Implementation. All pre-processing, post-processing and modeling were implemented in Python. All code for this project can be found here: `https://github.com/edeckter/cs-583/tree/master/Project`. The model was run using Google Colab to take advantage of their GPU capability. A number of tutorials and references were used to help me write the code for this project.
Keras Data Generators and Data Augmentation:

- A Detailed Example of How to Use Data Generators with Keras [4]
- Custom Image Augmentation with Keras [10]
- Albumentations Read the Docs [1]

Fully Convolutional Network (FCN) Modeling:

- A Beginner's Guide to Deep Learning based Semantic Segmentation using Keras [8]
- How to do Semantic Segmentation using Deep Learning [9]

Mask R-CNN Modeling:

- How to Train an Object Detection Model with Keras [5]
- Mask R-CNN - Train on Shapes Dataset [2]

Settings. The learning rate used for the Mask R-CNN model was 0.001 and the model was trained with SGD with momentum (the momentum rate was set at 0.9). The images were resized to 1,024 x 1,024 with zero padding for the height dimension (since the original images had a 2:3 height to width ratio). The batch size was 2 images at a time (the default setting for the Mask R-CNN) trained for a single epoch.

Advanced tricks. The Mask R-CNN model was pre-trained on the MS COCO dataset. In addition, data augmentation was used during training.

Cross-validation. The training images were split into training and validation datasets using 90% for the training dataset and 10% for the validation dataset, resulting in 4,991 training images and 555 validation images. A random seed was set to ensure the split would be the same each time the model was run.

4 Compared Methods

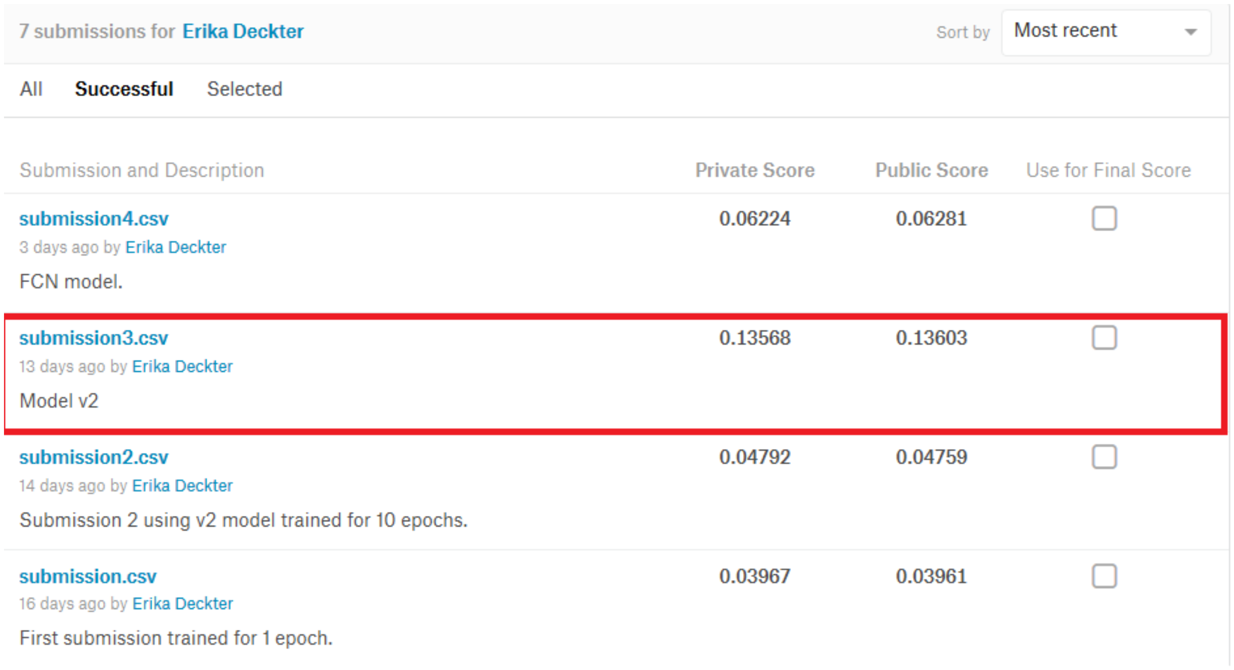
Baseline. As a baseline, I created a script that randomly assigned one of the 5 class labels for every image pixel and calculated the Dice coefficient comparing the random masks to the actual (ground truth) masks. This lead to a baseline Dice coefficient of 0.20.

Fully Convolutional Network. Next I implemented a version of the Fully Convolutional Network (FCN) model. My model architecture is a modification of the FCN-8 model proposed by Jonathan Long et al. in 2014 [6]. My model used a VGG16 CNN pre-trained on the ImageNet dataset as the encoder, and then built a decoder of deconvolution and upsampling layers. In addition, there were skip connections between each decoding block and the corresponding encoding block and from the original image to the final decoder output. A convolutional layer with a 1 x 1 kernel and a softmax activation function was used as the last layer to predict the label for each pixel. The validation Dice coefficients for this model is 0.00. This indicates a significant issue with the model. This low Dice score may also indicate that the model is overfitting. Further training and/or hyperparameter tuning may solve this problem.

Mask R-CNN. The final model was the Mask R-CNN model as described in the 2018 paper by He et al. [7]. This was implemented using a pre-built package by Matterport [3] using Python 3, TensorFlow and Keras. The code uses a ResNet101 backbone and pre-trained on the MS COCO data. Custom modifications to the Dataset class and the Config class were made to load and configure the images and masks for the cloud classification problem. The learning rate used was 0.001 and the model was trained with SGD with momentum (the momentum rate was set at 0.9). The images were resized to 1,024 x 1,024 with zero padding for the height dimension (since the original images had a 2:3 height to width ratio). The validation Dice coefficient for this model was 0.63.

Advanced tricks. Both the FCN and the Mask R-CNN model used the following tricks:

- Data augmentation. The Compose, HorizontalFlip and ShiftScaleRotate functions from the albumentations package were used for the FCN model training. Affine rotation and scaling



7 submissions for Erika Deckter

Sort by Most recent

All **Successful** Selected

Submission and Description	Private Score	Public Score	Use for Final Score
submission4.csv 3 days ago by Erika Deckter FCN model.	0.06224	0.06281	☐
submission3.csv 13 days ago by Erika Deckter Model v2	0.13568	0.13603	☐
submission2.csv 14 days ago by Erika Deckter Submission 2 using v2 model trained for 10 epochs.	0.04792	0.04759	☐
submission.csv 16 days ago by Erika Deckter First submission trained for 1 epoch.	0.03967	0.03961	☐

Figure 2: My Unimpressive Kaggle Scores

and Fliplr (left/right flip) were applied from the imgaug package for the Mask R-CNN model training.

- Pre-Training. The FCN model was built on a VGG16 backbone (imported directly from Keras) pre-trained on the ImageNet dataset. The Mask R-CNN model was built on a ResNet101 backbone pre-trained on the MS COCO dataset.

5 Outcome

My best Kaggle submission was a 0.136 public score and a 0.136 private score. This was based on the FCN model. This is unfortunately not very good, not even beating the random baseline. I believe this is due to a combination of not having the best hyperparameters for my models (due to very long training times, even using GPUs on Colab) and issues with the post-processing step that first transformed the model prediction masks to 350 x 525 sizes and encoded the pixel values into starting pixel and run length values for each class. The Mask R-CNN model had a much better validation Dice score; however, I was not able to get the outcome from this model formatted in such a way that it could be submitted to Kaggle prior to the deadline. Despite my disappointing results, I feel I learned a lot from this project. First I learned that Kaggle competitions, especially computer vision problems, are not easy! I also learned that even if a model has already been implemented, either pre-loaded in Keras or in a nice package like the Mask R-CNN package, there is still a lot of code that needs to be written to make that model work for your particular problem.

References

[1] albumentations. *Read The Docs*.

[2] Waleed Abdulla. Mask r-cnn - train on shapes dataset. https://github.com/matterport/Mask_RCNN/blob/master/samples/shapes/train_shapes.ipynb, 2017.

[3] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. https://github.com/matterport/Mask_RCNN, 2017.

[4] Afshine Amidi and Shervine Amidi. A detailed example of how to use data generators with keras.

[5] Jason Brownlee. How to train an object detection model with keras. *Machine Learing Mastery*, 3 October 2019.

[6] Jonathan Long et al. Fully convolutional networks for semantic segmentation. *arXiv:1411.4038v2*, 2014.

[7] Kaiming He et al. Mask r-cnn. *arXiv:1703.06870v3*, 2018.

[8] Divam Gupta. A beginner's guide to deep learning based semantic segmentation using keras. 6 June 2019.

[9] James Le. How to do semantic segmentation using deep learning. *Nanonets*, 2018.

[10] Ceshine Lee. Custom image augmentation with keras. *Medium*, 4 April 2019.