Image classification by FCN

Student: Yifan Bai

Abstract

Modern deep learning methods provide incredibly accurate results, however, they tend to be picky on quality and quantity of data. Oftentimes, we could have limited data available in hand and they could also be highly imbalanced, i.e. extreme difference between different label groups. This project aims to solve such a problem with a CNN with some data tricks.

1. Introduction

This project aims to solve a 3-class image classification problem, with all images of wood being assigned three labels: LargeKnots, NoDefect and SmallKnots. There is a fundamental issue that we need to deal with, which concerns with number of samples available per label. There are 1310 images for NoDefect, but only 80 for LargeKnots and 87 for SmallKnots. As such, two issues arise:

- LargeKnots and Small Knots have too little label to be directly trained.
- The sheer difference introduces high imbalance between data, as shown in Fig 1.

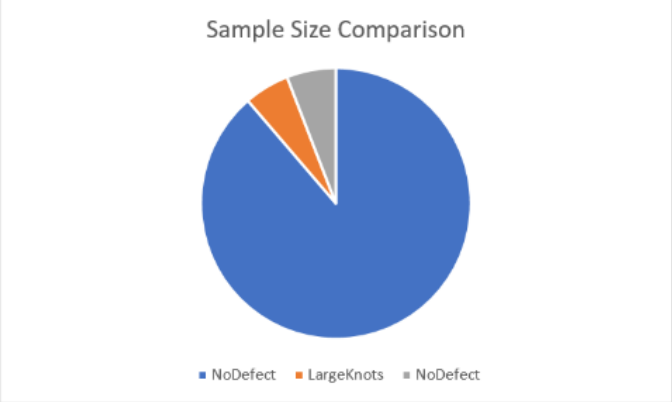

Fig 1: Pie Chart Visualization of Sample Sizes Between Labels

With small sample sizes, the model would easily overfit and perform poorly even during training, let alone testing. Moreover, even if there are enough data to train to predict all 3 labels, the model would be highly biased. That is, even fully trained, it would perform well for one label coming test time but generalize poorly for the two lesser sample classes [1].

2. Methodology

To tackle this for image classification tasks, one could rely on data augmentation to artificially boost sample sizes. Common approaches include but are not limited to: changing brightness, rotation, flipping. All these approaches are since CNN are shift-invariant at convolution layers, which serve to create feature maps and extract information. As a baseline, these three operations were executed to increase the size of the training samples available to the same level for each label. I also considered using cropping as did in my satellite imagery project, however, since the images are already small enough and cropping may get largely the same sub-samples as they are essentially pictures of wood, it would not be efficient nor effective.

2.1 Augmentation Details

We brightened the images by a factor of 1.5, and randomly rotate the image from counter-clock wise 180 degrees to clock wise 180 degrees. This is an arbitrary, baseline choice, inspired by a study done [2] to predict the rotation of images for self-supervised learning. For flipping, we simply used flip left and right by transpose.

2.2 Data Loading

After augmentation, we now have a set that is large and balanced enough for training and testing. In doing so, I used a standard 80/20 train/validation split for all images. For faster training, I also set the batch size to be 4. That is to have 4 randomly selected images per minibatch coming training time, and run through all images for 1 complete epoch. Fig 2 shows a sample of 4 images in one minibatch. As we can see, all 3 labels have at least one image. Particularly, the one from LargeKnots group is flipped, and the one from NoDefect and the second one from SmallKnots are brightened.

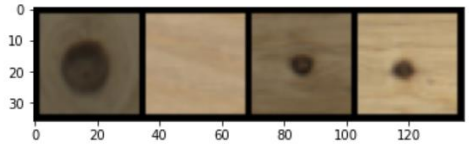

Fig 2: Sample minibatch.

2.3 Building FCN

Since we are required to build a fully convolutional network (FCN), there is no linear layers. FCNs have been known to perform well in semantic segmentation with its encoder-decoder structure. For the baseline model, I tried with encoder only and would add decoder should better performance be needed. The baseline model has 6 convolution blocks. For each block, there is a 2-D convolutional layer, followed by a ReLu layer, a 2-D batch normalization layer and a max pooling layer. At the final layer, we convert the tensor to batch * 3 tensor, and highest of 3 values stands for the prediction class.

2.4 Training

The loss function is the standard cross-entropy loss used by many classification problems. Training is sone via using standard SGD optimizer with parameters set to default, i.e. learning rate equals to 0.0001 and momentum equals to 0.9.

3. Evaluation

During evaluation, we also calculate recall and precision for each label, on top of accuracy across all the labels. This is to help evaluate the performance of the model in different dimensions. For this project, without data augmentation, we could still obtain high accuracy by having the model generalizing well on the NoDefect class accounts for 88.7% of the entire dataset. However, if we were to use this model for real life applications where the distribution between label classes change, the model would perform miserably. In essense, the former process would produce a model that 'overfits' badly into one class and the generalization to all three classes would be bad. In a Kaggle Competition setting, it would perform well on public tests but fail by large margin should the final private test set be different in terms of label distribution. To solve that, for instance, we could use recall, which for this case would produce a very low value and indicates poor performance. We would report results for all three metrics.

4. Results

After data augmentation, the problem essentially reduced to a simple supervised classification. The network was able to achieve convergence within 5 epochs. Fig 3 shows a plot of accuracy for the model, and after only 4 epochs with augmented dataset, the train and validation accuracy are already over 90%. For the sake of experimenting and understanding overfitting characteristics, I let it run until 30 epochs. We also observe that after stabilizing at 8th epoch, the validation accuracy stopped improving. This typically implies that the model is not learning, hence further training may result in overfitting. As it goes on, there are some fluctuations on validation accuracy and the training accuracy also stopped improving, suggesting saturation. Since our problem is rather simple at this point, we did not encounter serious overfitting, as some have when validation accuracy actually starts to drop. Techniques like early stopping is handy to deal with such issue and some libraries allow a 'patience' factor to stop training after few epochs where validation performance does not improve with training.

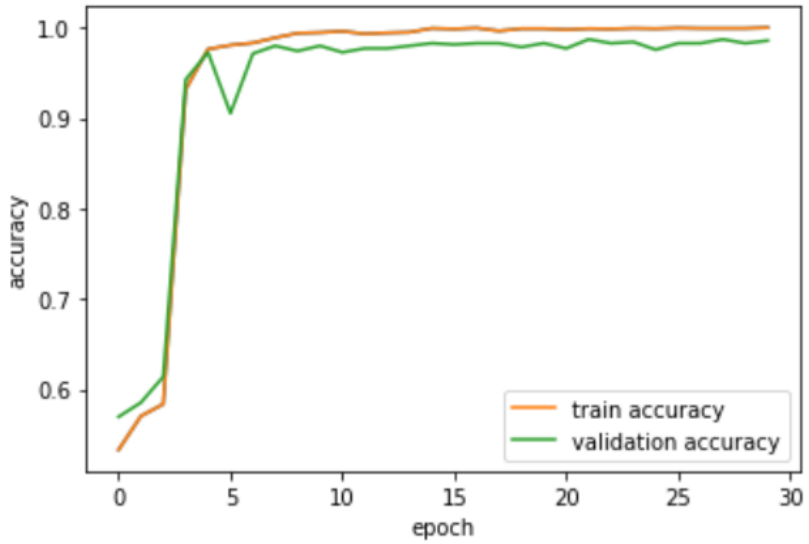

Fig 3: Train/Validation Accuracy Plot

From to Table 1, we can see that the model achieved great results. Note that these are taken at Epoch 10.

	Recall	Precision	Accuracy
BigKnots	0.903	0.939	
NoDefect	0.971	0.952	
SmallKnots	0.917	0.908	
Overall			0.933

Table 1: Metric after 10 Epochs

5. Discussions and Conclusion

To answer the follow up question, when working with small dataset for images, one would always be encouraged to deal with them using shift-invariant transformations, such as changing brightness, rotation and flipping used here. However, when we are using them, we should have the understanding of the task at hand as some techniques may not be the best suited, such as cropping which would only be valuable for larger images, such as satellite images. Another aspect is to deal with imbalanced dataset. Firstly, we shall use data augmentation discussed to boost sample size, and also use it to help balancing the dataset by making all labels

having the same amount of samples. The consequence for not doing so would cause the model to overfit and not able to generalize to the smaller lesser labels. To this end, we should also consider using metrics such as precision and recall, as in this case, a highly accurate model with low cross-entropy score would simply imply a model that classifies the dominant set well.

Notably, should time permit, we could further develop the model by incorporating the decoder, as well as explore other image data augmentation techniques and fine-tune the existing three. It also remains to be seen how would the model perform in a Kaggle competition setting and score on a private final test set with unknown label distributions.

Reference

[1] T. Boyle, "Dealing with Imbalanced Data," *towards data science*, 03-Feb-2019. [Online]. Available: https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18. [Accessed: 08-Nov-2020].

[2] Spyros Gidaris, Praveer Singh, Nikos Komodakis: "Unsupervised Representation Learning by Predicting Image Rotations", 2018; http://arxiv.org/abs/1803.07728 arXiv:1803.07728