UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

UNIVERSITAT ROVIRA i VIRGILI

Artificial Vision and Pattern Recognition Final Project

Jason Klimack

Barcelona, Master in Artificial Intelligence

January 4, 2021

Contents

Figure 1: Images from DeepCrack dataset. Top) raw input data. Bottom) labelled images of crack locations.

1 Introduction

Cracks in structural objects have potential to pose a great risk to workers and citizens. When such structures become cracked, the integrity, and hence safety, of that structure is lessened, and can result in catastrophe. Thus, inspection of structures that can cause such harm is required. Typically, manual inspection is performed by humans in order to determine the location of any cracks. However, this is a time consuming task, and humans are prone to make mistakes.

Automatic semantic segmentation of cracks can reduce the amount of time required, as well as increase the overall result in finding cracks on structural components. In this work, I implement a deep learning network for the semantic segmentation of cracks in an image. The data used is from the DeepCrack dataset [1].

2 Implementation

In this section, I discuss the dataset used in section 2.1, and the network architecture in section 2.2. The goal of the network is to assign a label to each pixel in the image. The labels are designated either `crack` or `concrete`, for pixels that are not a part of any crack.

2.1 Data

The dataset used in this work for segmenting the cracks in an image is DeepCrack. The dataset consists of images of cracks on asphalt, and includes a labelled image associated with each raw data image. The data is pre-split into train and test images, with 300 and 237 samples in each respectively. Some sample images from the train data are shown in figure 1, along with the associated labelled image for each.

Each image in the dataset has dimensions of 384x544 pixels, with 3 color channels corresponding to the RGB values of each pixel. As shown in table 1, there are far more pixels corresponding to the `concrete` class, than the `crack` class. This means that the classes are not balanced, which can result in the class with more samples (`concrete`) dominating the `crack` class during training, resulting in the model not learning the crack data very well. Class weighting can be used to solve this issue, by

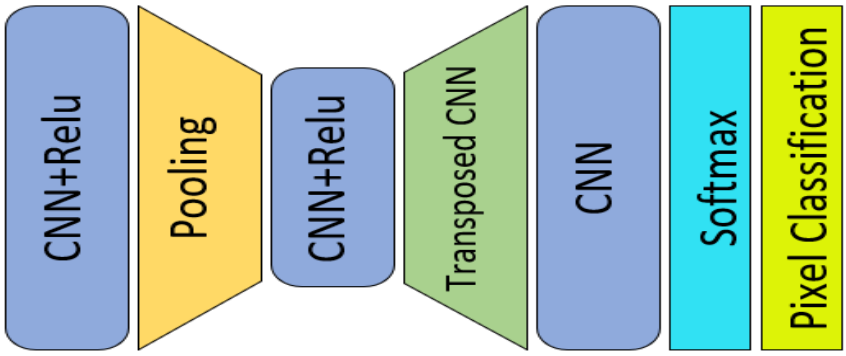

Figure 2: Network architecture. Input is a 2D image in RGB format, and the output is a 2D categorical label map with the same dimensions as the input image.

introducing a weight factor for the classification layer during training. The weights are computed as:

$$W_c = \frac{P}{P_c \cdot N} \tag{1}$$

where W_c is the weight of class c, P is the number of pixels in the images, P_c is the number of pixels of class c in the images, and N is the number of classes (2 in this case). The final values are shown in table 1.

Class Name	Pixel Count	Class Weight
crack	1.8254e+06	17.1660
concrete	6.0843e+07	0.5150

Table 1: The number of pixels corresponding to each class throughout all of the training samples, with the corresponding class weight that was used during training.

2.2 Architecture

The network architecture uses a hierarchical structure, shown in figure 2. The input to the network is a raw 2D image. The first 3 CNN layers each use a 3x3 filter, with 64 output features. Relu is used as the activation function. Pooling is used with a 2x2 kernel to reduce the size of the image, in order to learn lower level information from the image. The third CNN layer is a transposed CNN, layer, which combines upsampling and convolution function into a single layer. The final CNN layer uses a 1x1 filter and has 2 output features, used to reduce the number of features to the same as the number of classes. This allows the use of the softmax function for predicting the class of each pixel. Finally, the pixel classification layer is used to assign a class to each of the pixels in the image, and create a categorical label matrix as output with the same dimensions as the input image.

3 Results

In this section, I discuss the parameters used for training the network, the qualitative results, the statistical results, and finally some experiments that were performed on the network architecture.

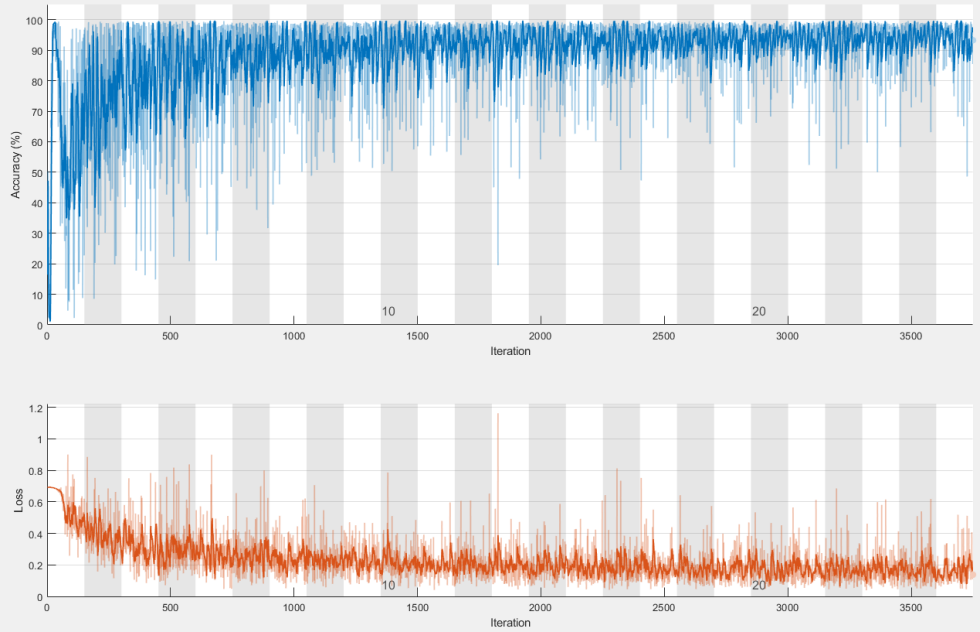

Figure 3: Training progress. Top) accuracy per iteration, bottom) loss per iteration.

3.1 Training Options

The network was trained for 25 epochs, with a learning rate of 0.001, and batch size of 2. The train data was shuffled every epoch. Figure 3 shows the training curves of the final model. By the tenth epoch, the model had more-or-less stabilized around the 90% accuracy mark. There was still fluctuation of between 5 and 10 percent, indicating that the model is not very stable.

3.2 Qualitative Results

The resulting model is able to achieve good qualitative results when there is little noise in the image, as shown in figure 4. However, when there is excess noise in the image, the model obtains additional False Positives. The noise may be able to be reduced either by performing some pre-processing on the input images, or by increasing the number of features/layers in the network.

3.3 Statistical Results

There were three different metrics used for determining the statistical results of the test set: accuracy, Jaccard distance, and F1-Score. Additionally, the number of pixels associated to each class are also counted, and are shown to have far more pixels associated to the `crack` class than the ground truth labels. The results per class are displayed in table 2, while the global results are displayed in table 3.

Class Name	Predicted Pixel Count	GT Pixel Count	Accuracy	Jaccard	F1-Score
crack	5.3228e+06	1.7702e+06	0.90798	0.29301	0.58425
concrete	4.4186e+07	4.7366e+07	0.92891	0.92526	0.66617

Table 2: Statistical results for per-class metrics.

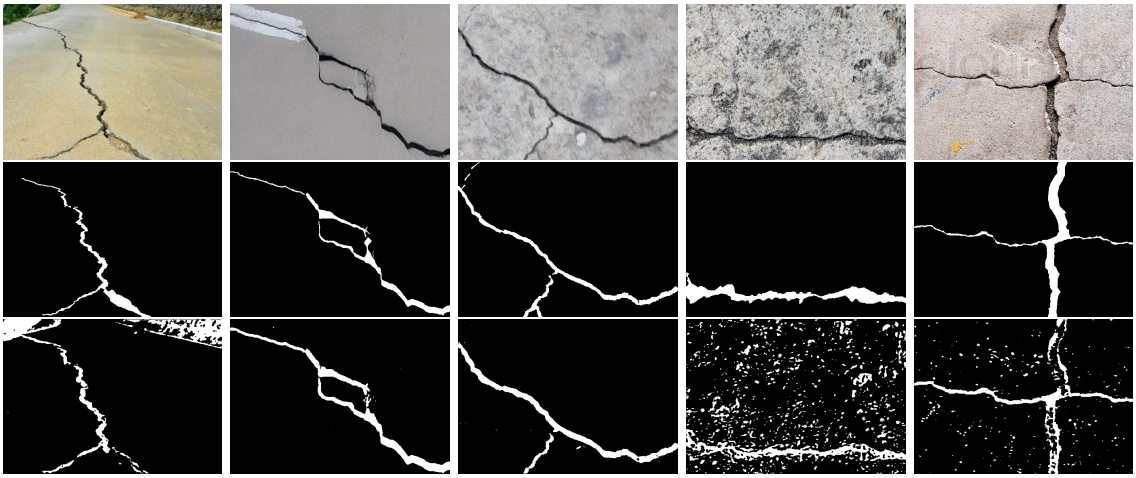

Figure 4: Segmentation results: top) original image, middle) ground truth labelled image, bottom) predicted labelled image.

Accuracy	Jaccard	F1-Score
0.91844	0.60913	0.62163

Table 3: Global statistical results for all classes.

3.4 Architecture

Alternative architectures were tested, by introducing more CNN and pooling layers, as well as adjusting the number of features in each layer. However, the changes only resulted in decreasing the performance of the model. Mainly, a frequent occurrence upon changing the network slightly was that all of the labels in the output matrix would correspond to a single class, and ignore the other. In other words, the network would become dominated by one class over the other, regardless of the class weighting that was introduced. Interestingly, with the majority of the pixels in the images belonging to the `concrete` class, if that class dominates the model in training such that all pixels in the image are labelled as `concrete`, then the accuracy of the model was very high, ranging from 95-98 percent.

4 Conclusion

In this work, I implemented a semantic segmentation network for the use of locating cracks in images of asphalt. The classes in the dataset DeepCrack are not at all balanced, which provides a slight challenge at achieving high quality results. The resulting network was able to properly locate the cracks, however in the case of noisy images, the model had many false positive results. Thus, more work is needed to handle the noise in the images, either by fine tuning the network, or by pre-processing the images before computing their labels.

References

[1] Yahui Liu et al. "DeepCrack: A Deep Hierarchical Feature Learning Architecture for Crack Segmentation". In: *Neurocomputing* 338 (2019), pp. 139–153. DOI: 10.1016/j.neucom.2019.01.036.