COMP3007 COMPUTER VISION COURSEWORK

Panagiotis Daniilidis

ABSTRACT

The abstract should appear at the top of the left-hand column of text, about 0.5 inch (12 mm) below the title area and no more than 3.125 inches (80 mm) in length. Leave a 0.5 inch (12 mm) space between the end of the abstract and the beginning of the main text. The abstract should contain about 100 to 150 words, and should be identical to the abstract text submitted electronically along with the paper cover sheet. All manuscripts must be in English, printed in black ink.

1. INTRODUCTION

This paper is within the context of the Computer Vision coursework and it presents a methodology for segmenting flowers from images. It attempted to create two convolutional neural networks, one by making use of an existing model and an original one. For the existing part, the architecture U-Net [1] was used for its deep layer system and complexity. The original part involved trying to improve the triangle segmentation example from the labs in order to try and achieve similar results to U-Net, albeit with fewer features, while attempting to improve training time. The whole project was a simplistic approach to CNN creation and training, heavily based on the lecture and lab examples due to strict time limitations.

2. METHODOLOGY

2.1. Structure

Both model iterations share the exact same code structure, with the same methodologies for loading the images and labels, ensuring that only annotated images are used, simplifying the class identification to only "flower" and "background" to normalize the data, using the same options for the training of the models and testing with the same additional image. The only thing that changes is the model used in each case, or more specifically, the amount and structure of the convolutional layers used.

2.2. Loading the Data

Loading the images is fairly straightforward. What is noteworthy is how the images were identified as annotated and unannotated so only the annotated ones could be used for training. Firstly, the "dir" Matlab function was used that returns the details of the files and directories in a structure array. This allowed for the extraction of the filenames in both the image and the label directories, and the subsequent loop through the names of the image files, where each image name is compared to every label file name and only the if there is an exact name match is the image then passed to a new "annotatedImages" array.

2.3. Creating Datastores

Knowing what classes each label file has identified each pixel into, along with knowing their names and IDs, we can specify which of them we need the model to train on during the use of "pixelLabelDatastore", which manages the mapping of each pixel in the annotation images to a specific class. Essentially, the model was told to focus on pixels with values '1' for 'flower' and '3' for 'background'. The values for boundaries, leaves and sky are ignored, effectively becoming a part of the background, since none of them can be on the flower.

It was also at this point that the data was split into two parts, one for training and one for testing. First, the split ratio was set to 0.8, ensuring that that 80% of the data would be set for training and the rest for testing. Secondly, the order of the input data was randomized, with Matlab's random number generator set to 'default'. This means that every time the code is ran, the data will have the exact same "randomized" state, which helps with reproducibility and code testing without affecting the results of the model.

2.4. U-Net

After a bit of research, U-Net was chosen as it described itself as being able to handle complex segmentation tasks due to its advanced feature reuse capabilities through skip connections. Having known uses in the agricultural industry, it would theoretically allow for a good benchmark to base the creation of the custom model around. As it will be described later, U-Net provided great results but at the cost of significantly longer training time. 23 lepta

It possesses extensive downsampling capabilities, with multiple encoder blocks containing 2 consecutive convolutional layers, utilizing ReLU activation. As the depth increases, the number of filters in each convolutional layer doubles, allowing the network to capture more complex features. It uses no padding and 2x2 max pooling

operations to quickly reduce the feature maps and perform operations on them at the bottleneck to receive maximum quality high-level features. Subsequently, it uses deconvolution methods, in conjunction with “skip connections”, which concatenates feature maps from the downsampling path with the upsampled feature maps to reintroduce the localized feature information lost during downsampling, to reconstruct the spatial dimensions of the original image.

2.5. Model Architecture

The model created in the premise of this project borrows heavily from both the U-Net architecture and the lab example one. Even though it would be ideal to describe this decision as the optimal solution, it was primarily forced through severe time constraints. Regardless, it provided valuable experience, and a satisfactory result.

The main concern of this model was provide a decreased training time compared to the U-Net model. It achieves this by three main ways. Firstly, and most importantly, it has a simplified architecture. Having fewer layers and utilizing fewer filters makes the data reach the output layer more quickly. Secondly, the model uses ‘same’ padding, as opposed to no padding for U-Net, which maintains the same spatial dimensions throughout the layers. This consistency also simplifies the computational requirements. Lastly, it uses batch normalization, which essentially normalizes all the data being passed from one activation layer to another convolutional layer.

The architecture consists of 3 encoder blocks, followed by 2 decoder blocks before reaching the Softmax and Classification layers. Each decoder block consists of a convolutional layer with batch normalization and the Rectified Linear Unit (ReLU) activation function, repeated twice and finished with 2x2 max pooling. The convolutional layers are set to use 64 different 3x3 filters for more complex and varied features to be captured from the input. The number of filters used is double from each encoder block to the next. The decoder blocks have the exact same setup as the encoder blocks, only with transposed convolutional layers, and starting with double the amount of filters used, only to be halved in the next block.

12 lepta

2.6. Training

Next, the training options for the model were set. It makes use of the Adaptive Moment Estimation optimization algorithm, which during training adapts the learning process to the parameters, ensuring that less frequent ones are allocated more resources and more regular updates than the more frequent ones. A leaning rate of 0.001 was chosen, which through testing was found to be ideal for a dataset of this size. The max number of epochs was set to one just in the context of this paper, so that efficiency data could be accumulated more quickly for demonstration purposes. This is also evident by having the data be shuffled each epoch with “'Shuffle', 'every-epoch'“. It could be set to any amount to increase model accuracy. Then, it was set so each epoch would be split into iterations, each containing a batch of data, with each batch containing 16 samples. Lastly, it enables the display of a plot showing training progress in real time, showcasing loss and accuracy, updating every two iterations.

2.7. Testing

After the training is completed, the testing data will undergo sematic segmentation using the newly trained model, classifying each pixel in an image into the category that it represents. Then, it will compare the results of this segmentation with the groundtruth data of the ‘labels_256’ folder with the ‘evaluateSemanticSegmentation’ function. This provides invaluable evaluation information such as Pixel Accuracy, which measures what percentage of pixels was correctly classified, and Intersection over Union, which measures the overlap between the predicted segmentation and groundtruth, providing more practical evaluation results.

3. RESULTS

This section includes the plots of the training process as well as the evaluation metrics of each model.

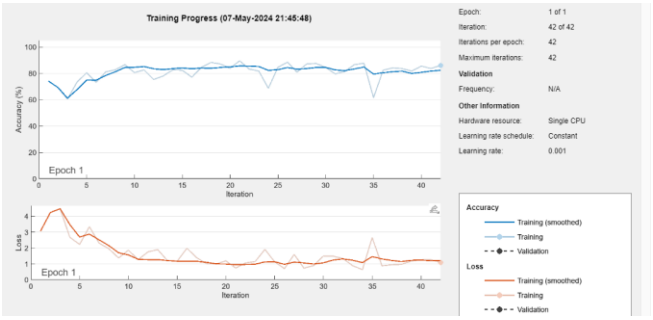

Figure 1: U-Net Model Training

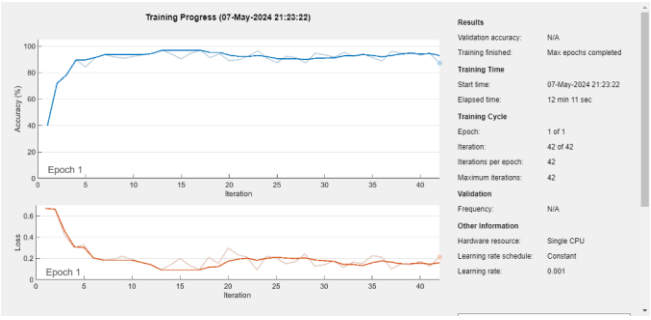

Figure 2: Custom Model Training

Dataset metrics:

GlobalAccuracy	MeanAccuracy	MeanIoU	WeightedIoU	MeanBFScore
0.87761	0.88004	0.76053	0.78832	0.55783

Class-wise IoU:

GlobalAccuracy	MeanAccuracy	MeanIoU	WeightedIoU	MeanBFScore
0.87761	0.88004	0.76053	0.78832	0.55783

Figure 3: U-Net Model Metrics

Dataset Metrics:

GlobalAccuracy	MeanAccuracy	MeanIoU	WeightedIoU	MeanBFScore
0.93558	0.92996	0.86176	0.88069	0.80469

Class-wise IoU:

GlobalAccuracy	MeanAccuracy	MeanIoU	WeightedIoU	MeanBFScore
0.93558	0.92996	0.86176	0.88069	0.80469

Figure 4: Custom Model Metrics

4. DISCUSSION

At first glance, it is apparent that the custom model vastly outperformed the U-Net model. It shows a smoother training process and higher percentages in the testing metrics. This is mainly due to the epochs for the showcasing purposes being set to 1, essentially depriving the more complex architecture time to completely comprehend the data. Regardless, however, when viewing the actual overlay performed on a testing image, the results can be quite interesting.

Looking at figure 5 and figure 6, it is apparent that even though the U-Net model has some rough classifications around the edges, it has done a much better job at correctly classifying more of the flower. In figure 6, the custom model has missed quite a bit of the left side of the flower, although that can be attributed to the data preprocessing incorporating the border class into the background, but it even incorrectly goes over the edge at the bottom right petal.

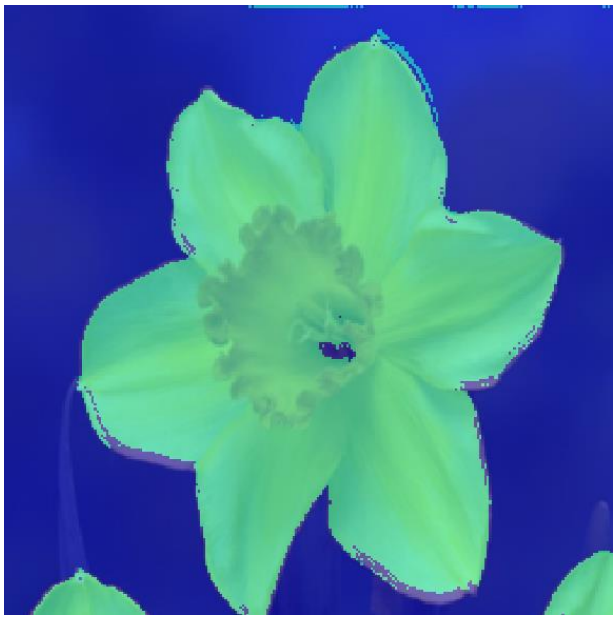

Figure 5: U-Net Model Overlay

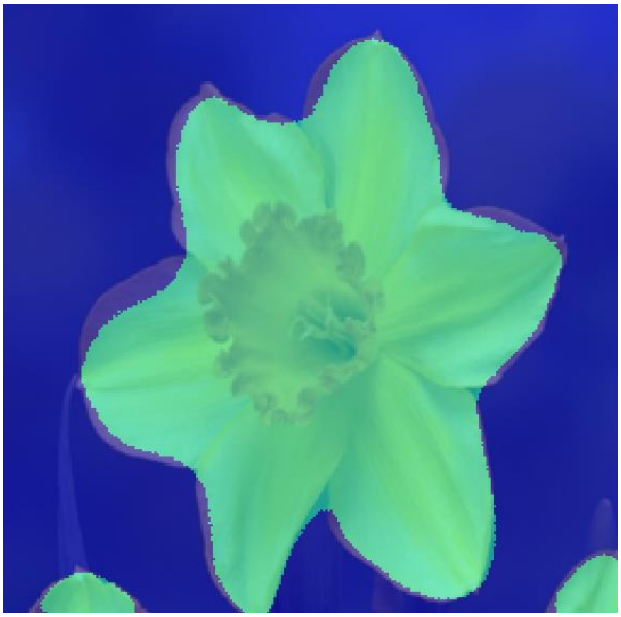

Figure 6: Custom Model Overlay

5. CONCLUSION

Although the data is not present here, and even though the average training time over one epoch for the U-Net takes double the amount of time it does for the custom model, given a higher amount of epochs vastly improves the accuracy of the U-Net model. An expected result, since it is a complex model used in biomedical imaging and satellite image analysis.

This whole project was completed in the span of two days. Given more time, the custom model could have been trained and tested with multiple different hyperparameters and over different amounts of time, producing more data to showcase in this paper.

6. REFERENCES

[1] **Taparia, A. (2023) 'U-Net Architecture Explained',** *GeeksforGeeks***, 8 June. Available at:** https://www.geeksforgeeks.org/u-net-architecture-explained/