Project Report - ECE 176

Conner Hsu
Electrical & Computer Engineering
A16665092

Christian Alonzo
Electrical & Computer Engineering
A13659485

Abstract

In this project, we reimplement the U-net architecture originally presented in Ronneberger et al. [4]. Our implementation is tested on the MBRSC dataset [2], which contains aerial imagery of Dubai. Our experiments tested the performance of the architecture with varying approaches to implementation, initialization, and training.

1 Introduction

In this paper, we recreate the U-net architecture and try to determine what is the optimal implementation, initialization, and training approach for a semantic segmentation task. More than just achieving high accuracy on the dataset, our goal is to gain a qualitative intuition of how different hyper parameters in training, initialization and implementation of the U-net affect its final performance. In particular, we experiment with different input image formats, weight initialization methods, learning rate schedules, and batch normalizations.

2 Related Work

While the U-net architecture was originally presented for use in biomedical fields and research, it has since become widely used for other applications. In Katsamenis et al. [1], U-net is used to analyze the structural integrity of roads, bridges, and buildings by identifying cracks. In Neupane et al. [3], the U-net architecture is modified by adding skip connections in the contracting and expanding parts of the network. They then test this model on aerial images of cities to try to measure the ecological footprint of buildings.

Our implementation of U-net will be used in a similar context to that of Neupane et al., and will also be used in semantic segmentation of aerial imagery. However, our implementation will maintain mostly the same architecture as proposed in Ronneberger et al. for the sake of simplicity, ease of implementation, and faster training times.

3 Method

3.1 Data Preprocessing

Data preprocessing turned out to be fairly time consuming. Below is a list of things that needed to be done before using it for training.

- The MBRSC dataset had a few images with incomplete masks and had a few pixels on the edges that weren't classified but also not given the unlabeled class. A script was used to fill these in with the unlabeled class.
- Uploading the dataset to the servers gave rise to an issue with the image masks. For some reason, the image compression caused some of the labels to be distorted and their values

were changed. Luckily, all of the pixel values of the labels were relatively close to their original values. We ran a nearest neighbor classifier on the distorted images to fix the compression effects.

- Like a lot of other semantic segmentation datasets, the MBRSC does not have a lot of data. There are only about 72 images in the dataset. As a result, more data needs to be generated. Each image was broken up into patches and each patch is rotated 4 times to generate 4 images per patch. This allows us to have a much larger dataset and also helps the model become robust to rotation variance.
- Since the U-net architecture has a smaller output size than input, the ground truth images needed to be cropped according to this size.
- The mean and standard deviation for the red, green, and blue channels of the example images were calculated. This helps to normalize the data before it is used in training.
- The data was split into 70%, 10%, 20% for the training, validation, and testing data respectively.

3.2 U-net Implementation

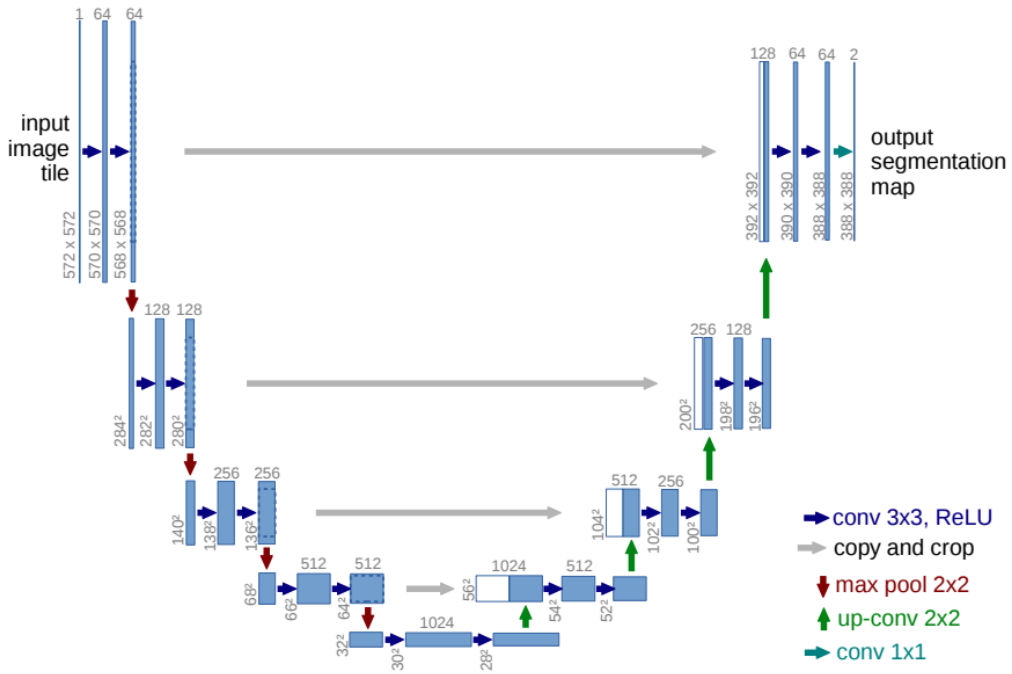

Figure 1: U-net architecture as shown in Ronneberger et al.

In this project we tried to stay true to the original U-net architecture as much as possible. However, a few modifications were made to fit our experiments better.

- The input image tile size was not always 572x572. The input image tile size serves as a hyper parameter for us to tune and is determined by the patch size decided in the data preprocessing.
- Different sizes of the U-net were tried as well. In the original implementation, the network starts with 64 channels then doubles the channels after every 2 convolutional layers in the contracting part of the network. We tried U-net with 16 channels in a few experiments to see if that would help the network generalize better by avoiding overfitting.
- The padding for the convolutional layers was set to "same" for a few trials.

- A sigmoid activation function was used for the 1x1 convolution at the end instead of a ReLU.
- A batch norm after each convolutional layer was also tried in some experiments.

3.3 Initialization, Training, and Evaluation

- For experiments that actually used a weight initialization, `kaiming_normal_` was used.
- For training, Adam optimization was used with a learning rate schedule. Cross entropy was used as the loss function.
- Intersection over union (IoU) was used to evaluate the perforamcne of the network.

4 Experiments

4.1 MBRSC Dataset

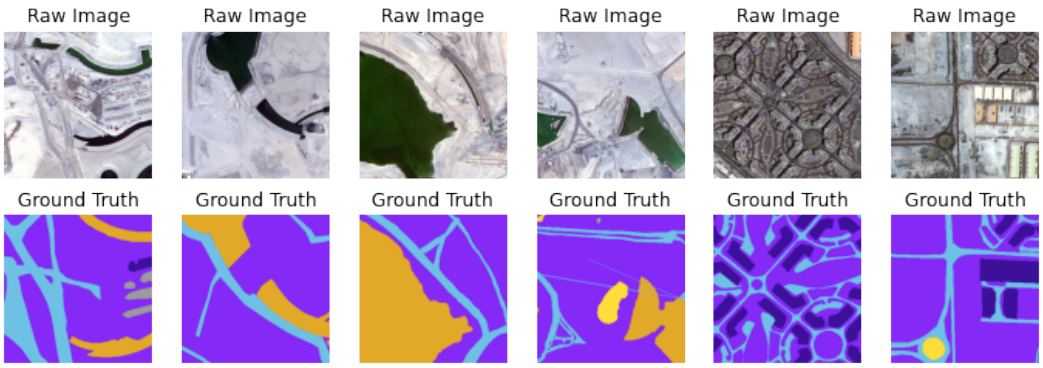

Figure 2: Examples of images in the MBRSC dataset.

In the following experiments, the MBRSC dataset was used to evaluate the performance of our U-net implementation. This dataset contains aerial imagery of Dubai. Aerial photos are paired with a corresponding semantically segmented mask. Masks are segmented into 6 classes:

- Buildings (dark purple)
- Land, unpaved (purple)
- Roads (light blue)
- Vegetations (yellow)
- Water (dark orange)
- Unlabeled (grey)

The dataset consists of 8 sets of 9 images. The images and masks for both formatted in RGB, however images are saved a `jpeg` while masks are saved as `png` files. Every image has different dimensions, however images within a set are relatively similar to one another. Dimensions range from roughly 500x500 pixels to 2000x2000 pixels.

Since the input to the U-net structure is fixed, these images were all split up into different patches in order form the dataset. From there, data augmentation such as rotations or allowing for overlap between different patches was used to generate more or less data.

4.2 Results and Discussion

This section will discuss each of the hyper parameters that were changed during experiments and how they affected the final training loss. Generally the network had a performance of about 30-50% mean IoU.

4.2.1 Input Image Size

The first hyper parameter is input image tile size. This can range from 188x188 to roughly 500x500 (minimum size is a restriction from the U-net architecture, the maximum size is related to the smallest image size in the dataset). Generally speaking, it was found that a larger input tile size improves performance since this gives the network more context for which to classify each pixel in.

4.2.2 Data Augmentation and Quantity of Data

More data was produced by either rotating the image tiles or by decreasing the stride of the script that generates the data patches. Although we initially thought more data would help improve the performance of the network, past a certain point the final IoU wouldn't improve by much. Additionally, the training time would be significantly longer. Since our goal is mainly to experiment with different parameters, having shorter training time is desired.

4.2.3 U-net Size

We define the size of the network by the number of channels after the first convolutional layer of the U-net. In the original implementation, the network had a size of 64. We tried a size of 16 for a few trials and found that the network tends to learn faster and is less prone to overfitting.

4.2.4 Convolutional Layer Padding

In the original paper, the convolutional layers all had a padding of zero. This effectively reduces the size of the output image compared to the input image. As a result, the network only needs to classify the "center part" of each input image. The intuition behind this is that the network needs sufficient context for each pixel in order to accurately classify it; pixels too close to the edges don't have enough context to be accurately classified. However, we tried a few runs `padding="same"` and found that the performance actually improves. Perhaps when it comes to this dataset the surrounding pixels are less deterministic of what an individual pixel should be classified as. It could also be that the pixels that are ignored/trimmed off actually confuse the network by giving it more data than it actually needs.

4.2.5 Batch Normalization

Adding a batch norm after each convolutional layer generally increased performance especially in the first few training epochs. However, the network without batch norm would sometimes be able to match the performance of the network with batch norm if given enough epochs.

4.2.6 Activation layer for Conv 1x1 Layer

We originally tried the sigmoid function as the activation nonlinearity for the final convolution layer with a 1x1 kernel. Although using ReLU is standard as an activation function, using a sigmoid feels intuitive because its being used on a classification layer. If a channel has an output of 1 that would strongly indicate that pixel belongs to the class, and an output of -1 would indicate that pixel does not belong to that class. However, we tested both a ReLU and sigmoid on the final convolutional layer and didn't seem to notice much of a difference.

4.2.7 Weight Initialization

Not using a weight initialization seemed to have a positive effect on the final performance of the network.

4.2.8 Learning Rate Schedule

We ran two trials, one with a learning rate schedule and one without. The learning rate schedule will reduce the learning rate as the validation loss of the network begins to stabilize. The learning rate schedule tends to have a positive effect on the final IoU.

4.3 Conclusions

Based on our experiments, we can categorize each hyper parameter into different categories:

- **Strongly affects performance**: batch normalization, weight initialization, size of network, Conv2d padding, input image size
- **Moderately affects performance**: quantity of data, learning rate schedule (these parameters have diminishing returns once sufficiently large).
- **No effect on performance**: activation layer for final convolutional layer

In general the best networks used batch norm, no weight initialization, had a smaller network size than the original, used `padding="same"` and used a large input tile size.

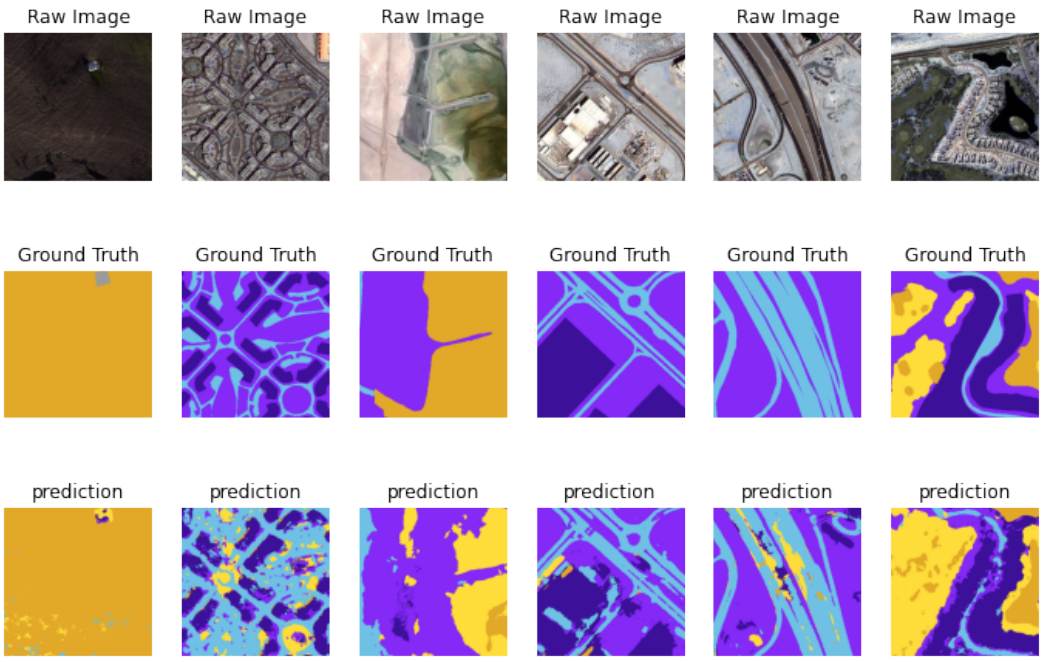

Figure 3: Examples of predicted masks using our U-net implementation.

Looking at some of the predictions made by the networks, a couple of observations can be made. The network is fairly good at identifying where there are roads (roads are labeled in light blue) and can sometimes overdo it a little (look at Figure 3, second prediction from the left). It also tends to confuse water and vegetation (water is dark orange, while vegetation is yellow). The aerial imagery is fairly tricky since the colors of vegetation can change depending on the location and season and the color of the landscape can vary greatly by time of the day or what kind of weather the images were taken in.

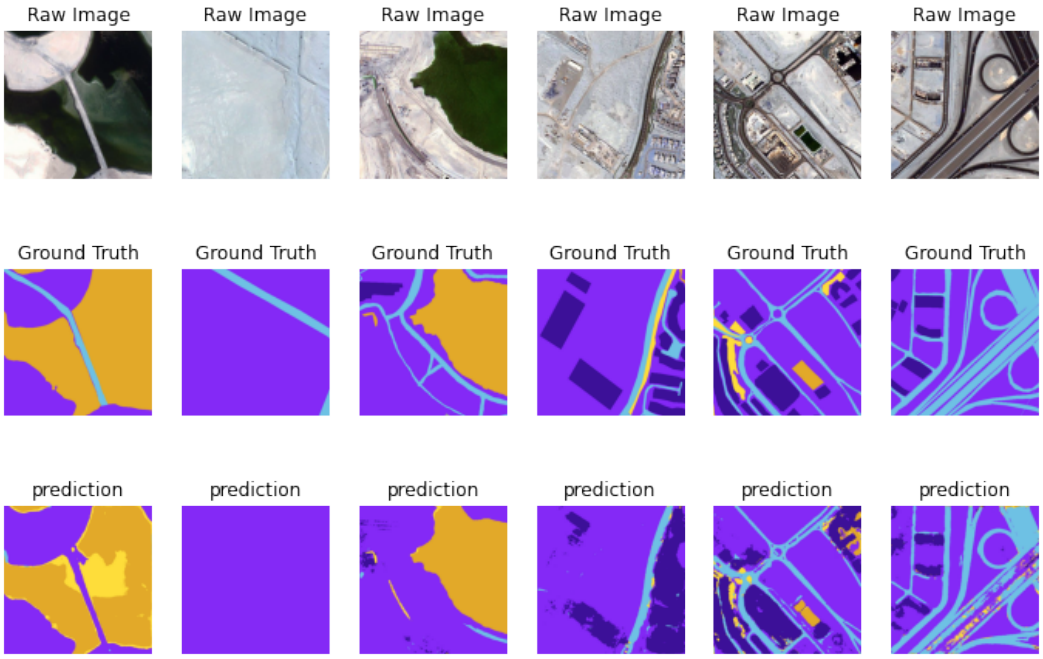

Figure 4: More examples of predicted masks using our U-net implementation.

The network also seems to have trouble with identifying nonpaved roads. Looking at Figure 4, second image from the left the unpaved road does not get detected at all by the U-net.

5 Supplementary Material

- Here is a video of us summarizing our paper:
  - `https://youtu.be/EyEu7xn1Iqo`
- Here is a link to a repository containing all of our code:
  - `https://github.com/khannuuuuur/ECE176_U-net`

References

[1] Iason Katsamenis, Eftychios Protopapadakis, Nikolaos Bakalos, Anastasios Doulamis, Nikolaos Doulamis, and Athanasios Voulodimos. A few-shot attention recurrent residual u-net for crack segmentation, 2023.

[2] Humans In The Loop. Semantic segmentation of aerial imagery, May 2020.

[3] Bipul Neupane, Jagannath Aryal, and Abbas Rajabifard. A novel dual skip connection mechanism in u-nets for building footprint extraction, 2023.

[4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.