Fully Convolutional Network in Land-Water Classification

Capstone Project

Feb 3, 2018

Main Author:Akhil Bhargava

Advisor: Vipin Kumar

Table of Contents

Introduction

Background

Within media and scientific literature, there have been reports of Earth undergoing through a climate change. Although there is media debate on the effect of human activity on climate change, the impact of climate change is on a global scale. One aspect of climate change would be the changes to bodies of waters and land globally. With the amount of water and land on our planet, it's impossible to observe in-person the changes to the bodies of land and water. Luckily, satellite images of the globe can be used to visualize the changes to water and land pixels of the globe. With billions of pixels spanning throughout years, it's a difficult task to classify what pixels of the globe are water and land manually. As a result, machine learning and predictive models can be used in order to automatically classify what pixels of the globe are water and land. If a model provides an accurate enough prediction, it will be allow researchers and other organizations to examine the changes to bodies of land and water for past and future years.

Dataset

The dataset used contains images of the globe taken by NASA's moderate-resolution imaging spectroradiometer (MODIS)[1]. These images were taken daily, and subsequently extracted to gather images that contained lakes and rivers from the dates between 2/18/2000 - 3/18/2000. Using the temporal context of an eleven day timeframe, the images at the same location are combined to generate a new cleaned image. MODIS collects 500 m sq. resolution data using seven bands.

Number	**Bandwidth (nm)**	**Name**
Band 1	620 - 670	Red
Band 2	841 - 876	Near Infrared (NIR)
Band 3	459 - 479	Blue
Band 4	545 - 565	Green
Band 5	1230 - 1250	Near Infrared (NIR)
Band 6	1628 - 1652	Shortwave Infrared (SWIR)
Band 7	2105 - 2155	Shortwave Infrared (SWIR)

Table 1: Band Description

Using the images, four datasets are procured: One containing a small subset of images of the lakes conserving its spatial pixel information (**ImageLevel_Dataset1**), and the other containing record's of a pixel's class and its corresponding 7 band values (**Pixelwise_Dataset1**). Another

[1] "MODIS Web." *Https://Modis.gsfc.nasa.gov/about/*, NASA, modis.gsfc.nasa.gov/about/

larger subset containing overlapping and disjoint images from the first subset were similarly created as **ImageLevel_Dataset2** and **Pixelwise_Dataset2**.

Dataset1 contains 3 possible values for the labels: 1 (*water*), 2 *(land*), 3 (*unclassified*). Unclassified labels indicate there is a disagreement with Kumar Lab's and NASA's classification of that pixel's label.[2] Dataset2 only has 2 possible labels, as these all pixels of these images have no class label conflict between Kumar Lab's and NASA's classification algorithms.

Dataset	N Images	N Pixels	N Land	N Water	N unclassified
1	135	3931252	16887717	520985	1722550
2	956	1852618	1736328	116290	N/A

Table 2: Summary Statistics of Dataset 1 and Dataset 2

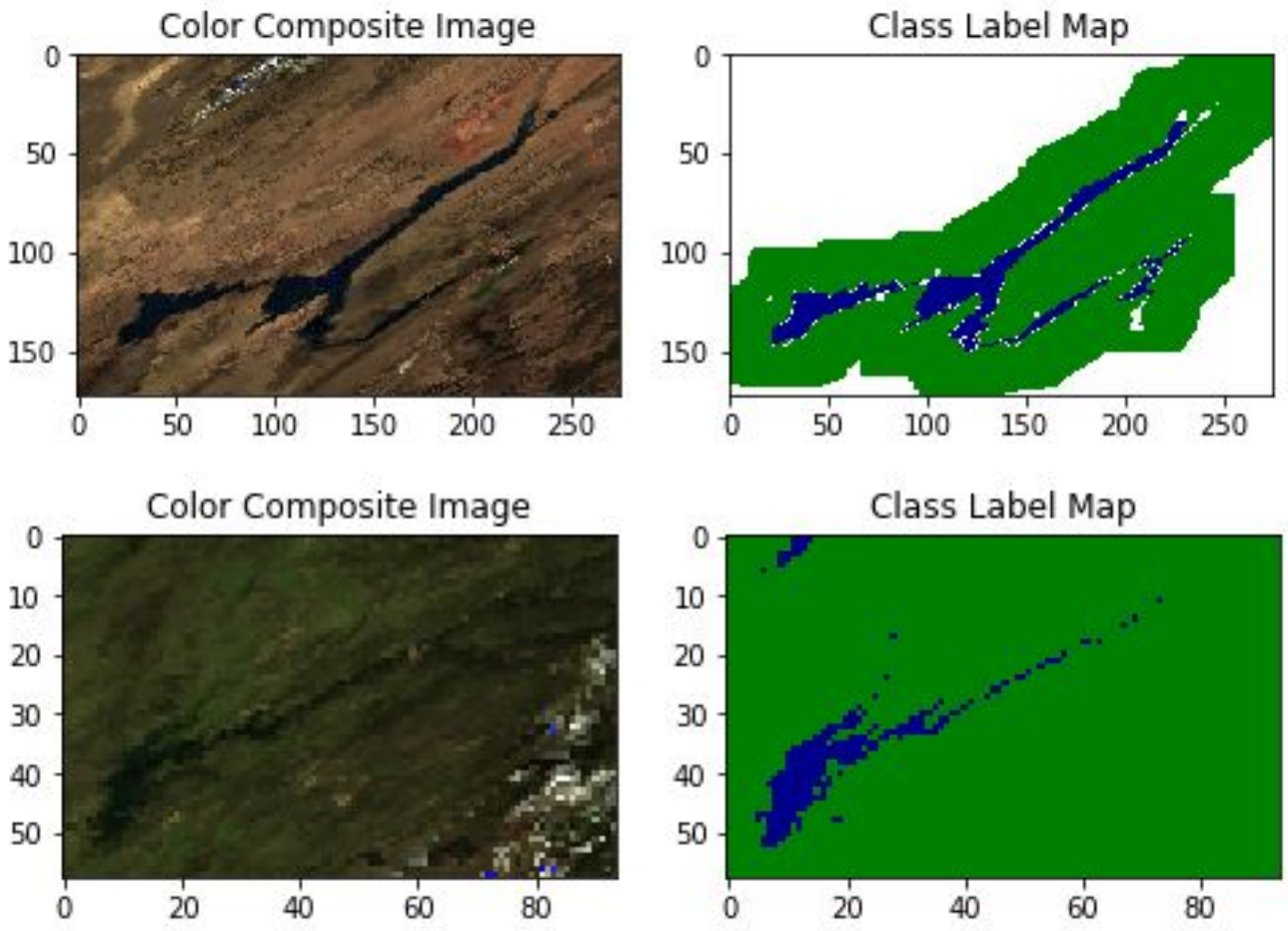


Figure 1 Lake and Labels: Using the RGB bands, the color composite image of the lake is shown along with the class label map. Dark blue indicates water, white indicates unclassified, and green indicates land. Lake from ImageLevel_Dataset1 is shown above lake from ImageLevel_Dataset2.

[2] "Vipin Kumar's Home Page." *Vipin Kumar's Home Page*, www-users.cs.umn.edu/~kumar001/.

Goal

The goal of the project focuses on utilizing a **Fully Convolutional Neural Network** (FCN) in order to perform dense pixelwise classification trained on the *ImageLevel_Dataset2* using only convolutional and deconvolutional layers[3]. Generation of the model will allow the ability to answer five main questions:

1). Will an FCN perform better than a naive classifier (choosing all pixels as land)?
For any model to be valid, it must be informative. The dataset contains two classes: land and water. The FCN should perform better than the naive classifier which would assume all pixels are land.

2). Is it necessary to use all 7 bands in a FCN?
It is quite possible that a few bands may provide a negligible increase to the explained variation of the class label for a pixel. It would be interesting to determine whether or not the inclusion of infrared and shortwave infrared bands will aid in the classification performance in the FCN.

3). Is spatial context a boon to pixelwise classification?
The purpose of the FCN is to capitalize on the spatial correlation that is inherent in land and water. In theory, if a pixel is surrounded by water, it is most likely of class water as well. In order to test this, a simple Logistic Regression model was implemented on the Pixelwise_Dataset2. If spatial context truly is a boon, the classification performance of the FCN should fare better than the Logistic Regression model.

4). Will the FCN perform better than popular deep learning framework UNET?
A comparison of how another deep learning model performs on the ImageLevel_Dataset2 will indicate if there is a benefit for utilizing an FCN for dense pixelwise classification[4]. The model was originally designed in Caffe for biomedical images.

5). How does the FCN perform on ImageLevel_Dataset1?
The “true” labels for the pixels are created from algorithms designed from NASA and the Kumar Laboratory, and it’s quite possible that the “true” labels may not be correct with reality. Although it’s impossible to quantify this via a metric, a side-by-side comparison of a RGB visualization of the image, the predicted labels of the FCN, and the “true” labels may provide some insight.

[3] Maggiori, Emmanuel, et al. “Fully Convolutional Neural Networks for Remote Sensing Image Classification.” *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 2016, doi:10.1109/igarss.2016.7730322.

[4] Ronneberger, Olaf, et al. “U-Net: Convolutional Networks for Biomedical Image Segmentation.” *Lecture Notes in Computer Science Medical Image Computing and Computer-Assisted Intervention MICCAI 2015*, 2015, pp. 234–241., doi:10.1007/978-3-319-24574-4_28.

Fully Convolutional Network Methodology

Input

Varying Image Size Issue

Utilizing Keras with tensorflow as its backend within Python, the required input shape for the features is: the *number of images, width of image, height of image, number of bands*. For example, if the dataset contained one-thousand 512 * 512 images, the corresponding numpy 4D array would be (1000, 512, 512, 7). The label array would be: *number of images, width * height, number of classes,* where the class is an encoded binarization. Following the original example, this would equate to a 3D array (1000, 262144, 2). The issue with the input requirements is the varying dimensions of the images within the ImageLevel_Datasets. The images can range from 12 * 28 to 132 * 132. As a result, it was decided to select patches of width *w* and length *l* from the images. One important note to consider is the training, validation, and test set split. It was determined that specific lakes should be first split into the training, validation and test sets, and then performing the patch-creation for each lake. The idea would ensure that there was no bias in training the FCN on a specific image of the lake or river and subsequently testing on the same image.

Patch Creation

A patch creation function was implemented in order to generate patches of a lake with a specified *w* and *l*. *Code can be found in the appendix.*

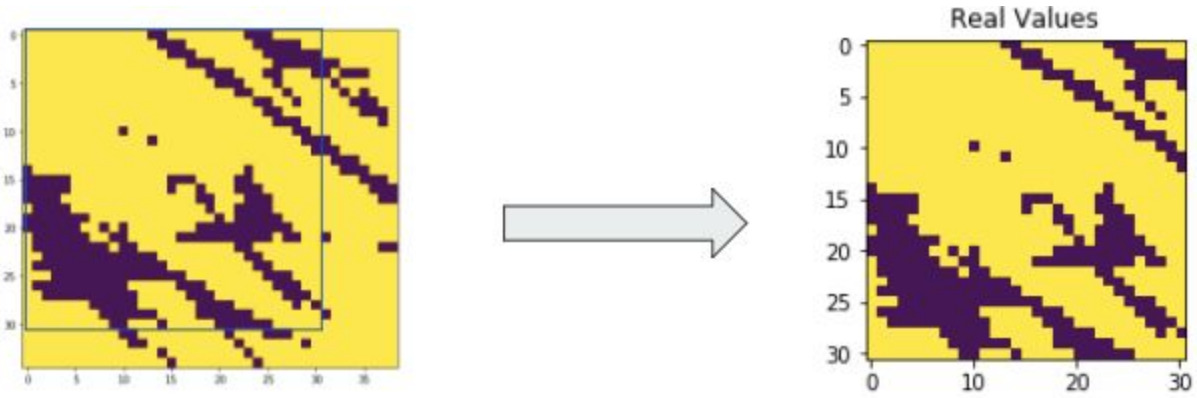


Figure 2 Patch Depiction: An image of size 38 * 36 was used as an input to output a single patch of size 31 * 31. Purple indicates water, yellow indicates land.

The patch function takes an input image, and starting from the top left corner, it creates as many patches with the dimensions of the specified width and length with no overlap. As a result, if there are pixels that won't fit exactly in the specified patch dimensions, those pixels are discarded. It is possible to start from the opposite side of the image, and generate patches to conserve the discarded pixels, but there would be redundant patches. The redundancy could trigger a bias within classification performance of the FCN and was not pursued. Furthermore, patches that contained only pixels of one class were pruned to remove bias.

An interesting dual problem forms: larger patches result in the loss of data, but larger patch sizes also retain the most spatial context. Analysis on this dual problem was performed comparing patch sizes of 8 * 8, 16*16, 32*32, and 64* 64.

Model Architecture

Architecture

The fully convolutional model had four convolutional layers, followed by 3 convolutional layers.

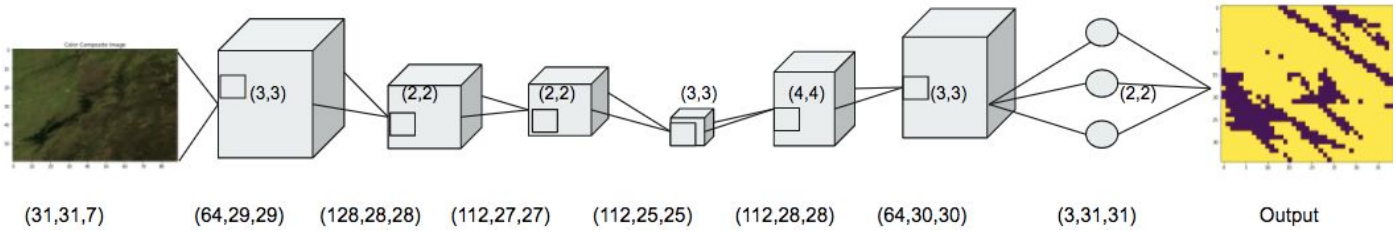


Figure 3 FCN architecture: An example patch size of 31 * 31 was used. Four convolutional layers with hidden units of 64, 128, 112, and 112 were used to learn specific neurons. Filter sizes used were (3,3), (2,2) , (2,2), and (3,3) These were followed by three deconvolutional layers containing 112, 64, and then 3 hidden units respectively to reform data to the original size using filters of size (4,4) , (3,3), (2,2). Each layer was activated using a relu.

After the first convolutional layer, a batch-normalization layer was introduced. Following the last deconvolutional layer, a reshape layer was added to get the 4D matrix (N, *w, l, 3*) converted into the 3D label matrix (N, *w*l*, 3) . Softmax was applied to determine the class a pixel belonged to after the reshape. The loss function used was categorical cross entropy.

Why Three Classes?

One important thing to note is the reshape layer reshapes the input to (Number of Patches, *w * l*, **3)**. Despite containing only two classes in ImageLevel_Dataset2 (land and water), the model determines the probability of a pixel belonging in three classes. The reasoning stems from ImageLevel_Dataset1 containing pixels that are of class “unclassified”. 3 classes are needed in the model in order to properly predict the labels of ImageLevel_Dataset1.

An inspection to determine if having an additional class would cause an issue in the classification of a pixel to be labeled as unclassified. It was deduced that the probability of a pixel being of this non-existent class was < 1%, indicating the impact on the presence of the class is negligible.

Experimental Results

Patch Dimension Dual Problem

Assigning the same 701 lakes to the training set, 109 lakes to the validation set, and 105 lakes in the test set using 30 epochs, an analysis on the impact of patch dimensions was performed. Each epoch took approximately 30 secs to train with a batch size of 16.

Patch Size	N train	N test	TN	FN	TP	FP	F1	Land %	Accuracy
8 * 8	4406	861	41629	2208	8585	2682	0.773	80.9%	91.27%
16 * 16	1592	324	69449	3206	8575	1714	0.7821	87.59%	94.39%
32 * 32	337	66	62248	1420	3144	772	0.7175	93.2%	96.62%
64 * 64	60	8	32562	206	0	0	NA	99.3%	99.3%

Table 3 FCN Patch Dimension Statistics: FCN metrics for various patch size dimensions are shown. TN indicates the number of land pixels that were correctly predicted as land. TP indicates the number of water pixels that were correctly predicted as water. FP indicates the number of water pixels that were incorrectly predicted as land. Conversely FN indicates the number of land pixels that were incorrectly predicted as water. F1 indicates the F1 score of the pixels.

Patch size dimensions of 16 * 16 contained the most pixels and had the highest F1 score. Furthermore, only patch size dimensions of 64 * 64 failed to perform better than a naive classifier choosing all pixels to be land. Lastly, the false negative to false positive ratio was approximately 2:1 for patch size dimensions 16 * 16, and 32 * 32. The 8 * 8 patch dimensions had a closer ratio of misclassification between land and water.

7 Bands Analysis

To look at the amount of variation in the class labels that is explained by the bands from the images, PCA was performed on the Pixelwise_Dataset2:

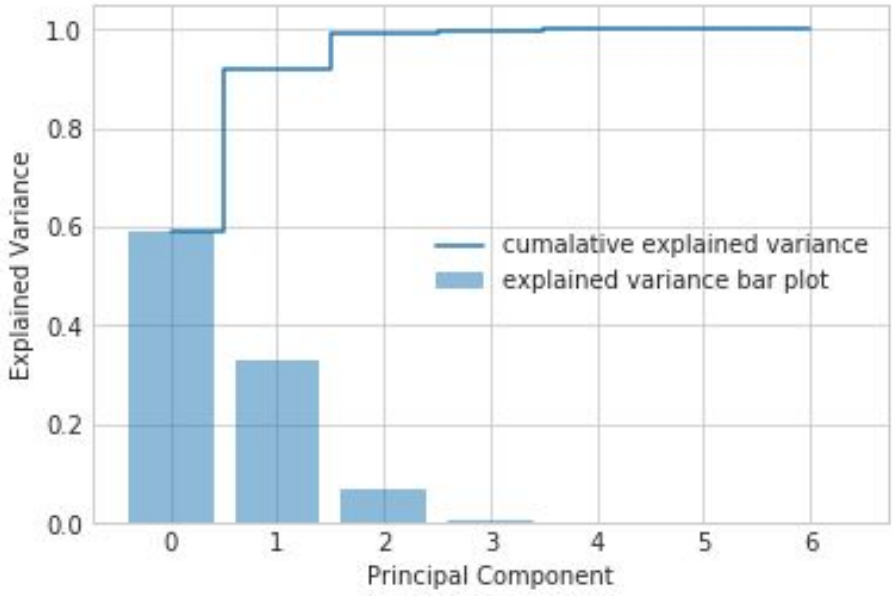


Figure 4: PCA Graph: Graph of the principal components used on the PixelWise_Dataset2 indicates that over 99% of the cumulative explained variance can be shown with 3 Principal Components.

PCA on the Pixelwise_Dataset2 indicates that it may not be completely necessary to create a classifier using all 7 bands. This lead to the curiosity how the FCN would perform without the NIR and SWIR bands.

Using the same training parameters listed above and the same FCN architecture, patches were constructed using only the red, green and blue bands. A quick comparison on the F1 measure of the FCN using RGB bands only compared to the FCN using all bands is presented below:

Patch Dimensions	**RGB F1 Score**	**All Bands F1 Score**
64 x 64	NA	NA
32 x 32	.4756	0.7175
16 x 16	.7009	0.7821
8 x 8	0.716	0.773

Table 4 RGB vs 7 Band Accuracy

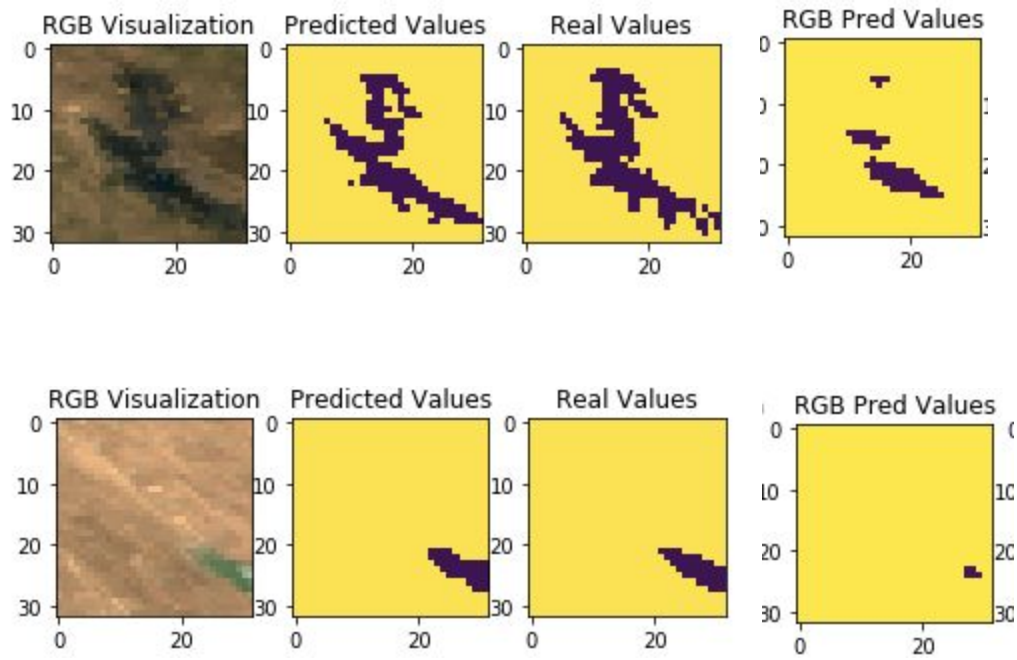


Figure 5 Visualization of RGB and 7 Band FCN performance: Patch size of 32 * 32 was used. Index of patches shown are 31 and 16. Purple indicates water, yellow indicates land.

Using only RGB confers a higher bias towards the FCN predicting a pixel is land.

Logistic Regression Comparison

The simple logistic regression was constructed on the PixelWise_Dataset2. Ideally, using the corresponding lakes that were used to train FCN was desired, however, the PixelWise_Dataset did not retain the information of its image counterpart. As a result, the training data was randomly split into 1389463 pixels, and was tested upon 463155 pixels.

	Predicted: Water	**Predicted: Land**
True Label: Water	18602	10618
True Label: Land	2957	430798

Table 5 Logistic Regression Confusion Matrix

The F1 score was 73.26%, and the percentage of land pixels in the test set is approximately 93.27%. Similar to the FCN, a logistic regression was able to learn a model that performed better than a naive classifier. The logistic regression was much quicker to train compared to the FCN.

U-NET vs FCN Comparison

U-NET, a popular deep-learning framework that was designed for biomedical images was used to compare classification performance.[5] The same images were used in the train-test-validation split from Imagelevel_Dataset2 as the FCN, along with using 30 epochs and a batch size of 16. Cross entropy was used for the loss function.

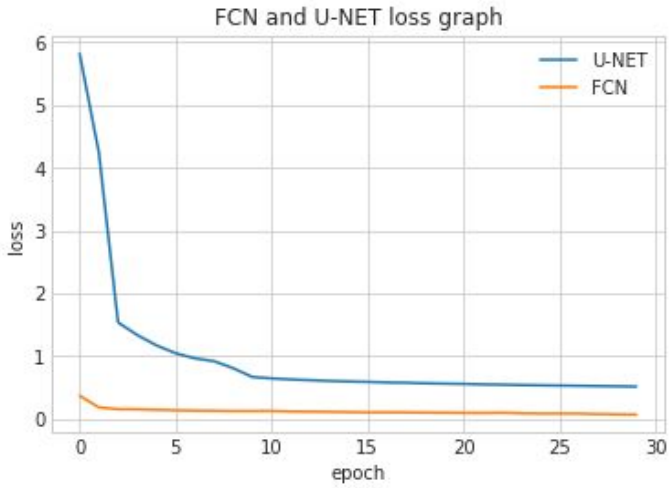


Figure 6: U-NET and FCN Loss Chart: Chart indicates the training loss during each epoch for both U-NET and FCN. Patch-Size used was 16 * 16.

	16 * 16 F1 Score	**32 * 32 F1 Score**
FCN	0.7821	0.7175
U-NET	0.7535	0.1926

Table 6: F1 Scores for FCN and U-NET: F1 Scores for patch size 16 * 16 and 32 * 32 are listed. 8 * 8 was not used due to U-NET architecture unable to handle patch sizes less than 16.

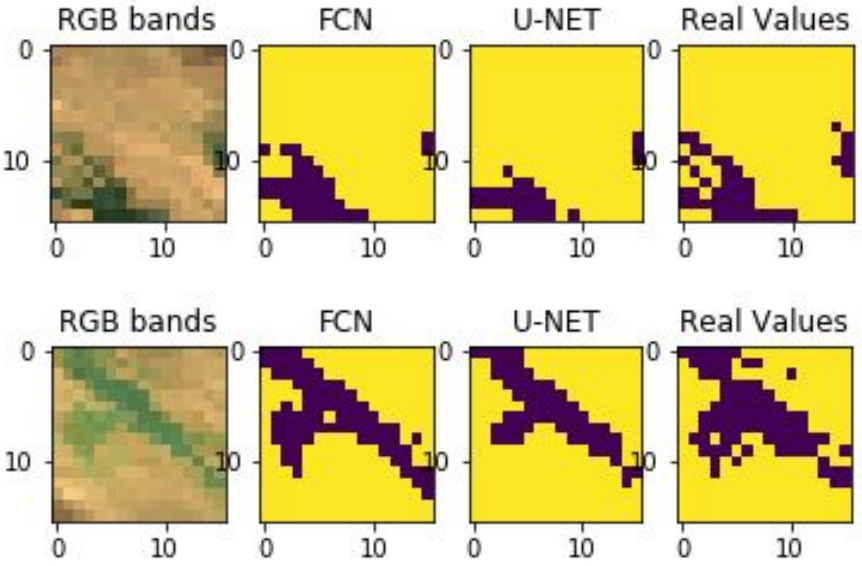


Figure 7: Images of FCN and UNET Prediction: Patch Sizes used were of size 16 * 16. Purple indicates water, yellow indicates land.

[5]Jocicmarko. "Jocicmarko/Ultrasound-Nerve-Segmentation." *GitHub*, 16 Aug. 2017, github.com/jocicmarko/ultrasound-nerve-segmentation.

Prediction on ImageLevel_Dataset1

ImageLevel_Dataset1 contained pixels possibly belonging to three classes: land, water, or unclassified. Using the same FCN model trained for 30 epochs and batch size of 16 on the ImageLevel_Dataset2, a prediction was performed on ImageLevel_Dataset1.

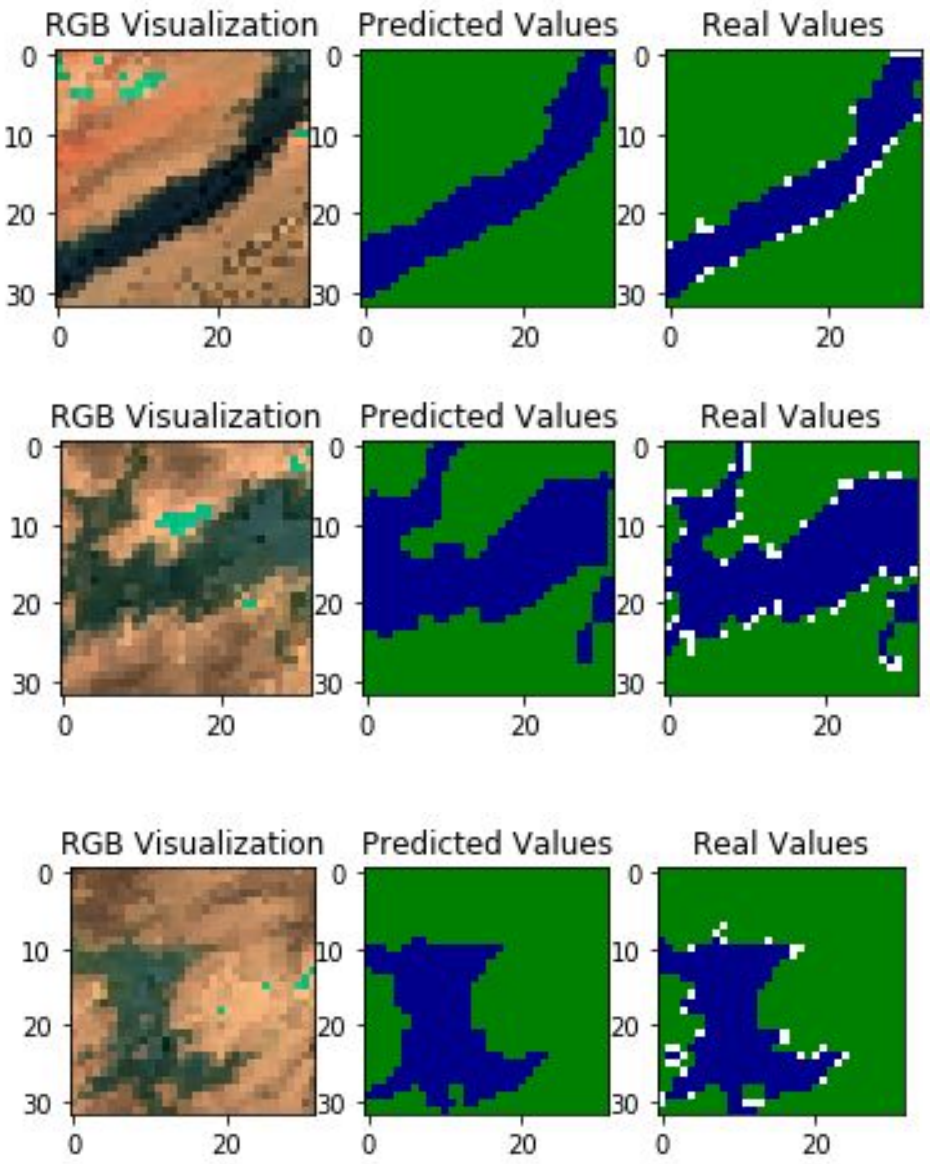


Figure 8: Image_Dataset1 Prediction: Predictions using FCN model of size 32 * 32 patch sizes was used. Green indicates land, blue indicates water, and white indicates “unclassified’. Unclassified is the result of NASA’s prediction and Kumar Lab’s prediction not syncing.

Conclusion

Patch Dual Problem

It's evident that patch dimensions play a role in FCN classification performance. The first trend noticed is the classification accuracy increased with larger patch sizes (**Table 3)**. This most likely stems from the datasets having a higher percentage of land pixels. As a result, it's a clear indicator that accuracy is a poor metric to evaluate classification performance.

Primarily, patch dimensions 16 * 16 contained the most pixels for training, as well as the highest F1 measure at 0.7821, making it the best choice. The reasoning most likely stems from a combination of *number of training patches, number of pixels,* and ***patch quality***.

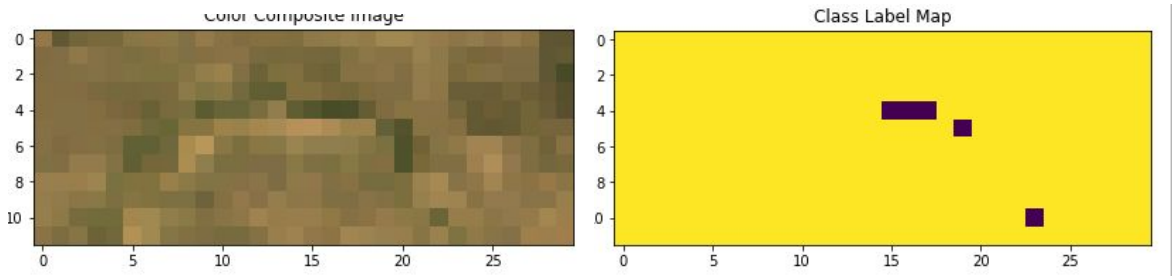


Figure 9: Smallest Image Visualization

An examination of smaller images in the dataset indicates a very coarse image **(Figure 9)**. These images are extremely difficult to correctly classify visually. 16 * 16 patches in theory, will include less images of poor quality, which may negatively impact the parameters learned in the FCN.

Secondly, patch dimensions 8 * 8 contains some interesting characteristics. This dimension should contain the most pixels in the entire dataset, however, it was only ahead of patch dimension 64 * 64 (**Table 3)**. This is the result of the pruning of patches that are only land. Furthermore, it's the only dimensions that had a ratio that was closest to 1 : 1 for having false positives to false negatives. A possible explanation may be explained as the impact of pruning only land patches. The 8 * 8 patches most likely contained patches containing only land, and as these patches were removed, there was a reduction in the bias of the model predicting a pixel to be of class land. Conversely, it's possible that there may be images that contained only one or two water pixels in the larger patch dimensions, conferring the increased bias for incorrectly classifying water pixels as land.

Lastly, patch dimension 64 * 64 failed to be of any meaningful value. With the accuracy matching the percent of pixels that are land, it performed as well as a naive classifier. This may not be an issue of patch dimensions, as there was an extremely limited amount of training patches. Overall, more images of high quality lakes would be desired to further expand on the analysis.

7 Bands Analysis

Initially, PCA indicates most of the explained variance can be represented in 3 principal components (**Figure 4)**. Upon further inspection, however, the visualization shown as well as the F1 scores indicates that using only the RGB bands tends to have a bias in indicating a pixel is land (**Figure 5 & Table 4)**. On the contrary, using all 7 bands seems to generate a prediction more similar in shape compared to the true labels and the image visualization. As a result, the addition of the NIR and SWIR bands decreases the bias of the model to classify a pixel as land. With water being such a rare class, it may be more beneficial to include the NIR and SWIR bands to correctly identify water pixels and decrease the bias of the model.

The explanation of 3 principal components retaining over 99% of the explained variation could be explained by its definition. PCA is an orthogonal transformation of the dataset, and it's possible that the 3 principal components are reliant upon all of the bands. Many of these bands independently contain similar information that would indicate a pixel is water or land, however, It's highly probable that a band used conditional on the other bands provides an deeper explanation of the patterns discovered.

Is Spatial Context a Boon?

With the results of the logistic regression conferring a F1 score of 0.7326 compared to the 16 * 16 FCN's F1 score of 0.7821, it brings interesting considerations if spatial context is useful for pixelwise classification (**Table 5, Table 3)**. The first thing to note is the training speeds are quicker for the logistic regression model. The second consideration is the ability for the logistic regression model to utilize every individual pixel. With reliance on patches, the FCN confers loss of data as not all of the pixels will be able to fit in the patches. This imposes a constraint on images to be predicted to be larger and cleanly divisible by patch dimensions.

The F1 score of the FCN was far better than the logistic regression, however, it's also important to note that the FCN could not be trained on the exact same data as the Logistic Regression. It's possible in theory if one could acquire the same training, valid, test split set for the pixels, the logistic regression would have an extremely different F1 measure.

Overall, it's not conclusive if spatial context truly is a boon. A F1 score of 0.7821 is much more impressive than a score of 0.7326, but more data would be required to determine if spatial context provides a new dimension to predicting the labels of land or water pixels.

FCN vs U-NET Comparison

Experimental results indicate FCN is the better deep learning model for this dataset. Not only does FCN have a better F1 score, the number of epochs required to achieve a negligible decrease in loss is smaller (**Table 6, Figure 6 and Figure 7,).** The one benefit U-NET has over FCN is that an epoch takes approximately 23 secs rather than FCN's 30 secs per epoch to train the models.

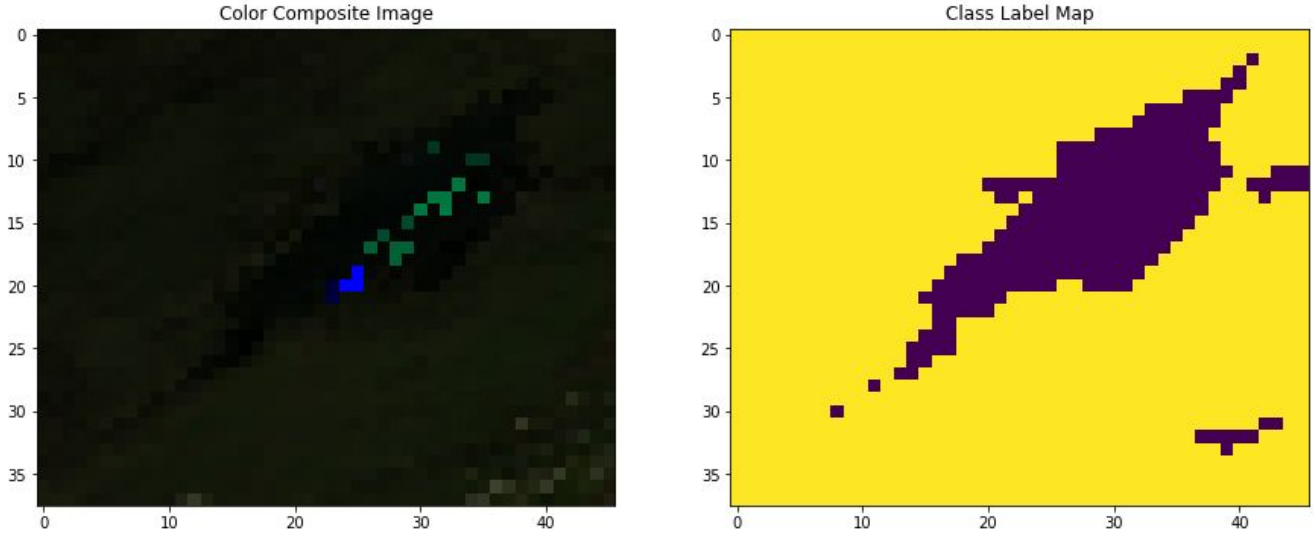
One possible reasoning why U-NET may perform worse than FCN's is the upsampling methods. FCN's utilize deconvoluational layers that use learned parameters to upsample the image. U-Net utilizes a simple upsampling technique, which is the opposite of a max-pooling layer. Secondly, U-NET also has an immense amount of parameters. U-NET contains over 2.8 million trainable parameters, whilst FCN uses 473,739 trainable parameters. This would clearly explain the training epochs required. It's possible U-NET may perform better if the number of hidden layers and units were changed. U-NET was initially designed for biomedical images which are larger than the MODIS images, however, the parameters were not changed in order to not deviate from the original paper's algorithm.

ImageLevel_Dataset1 Performance

The FCN does an extremely good job of gathering the overall shape, however, upon examination there are issues with the performance. The performance of the FCN on ImageLevel_Dataset1 provides a slightly coarser version of the classification results procured by the Kumar Laboratory and NASA (**Figure 8)**. This biggest issue lies upon the classification of pixels that neighbor large amounts of land and water. It's possible that more hidden layer may be able to identify this pattern, however, with only a limited amount of training instances, it's difficult to increase the amount of hidden layers and units of the FCN.

Domain Adaptation

One important consideration is the topic of domain adaptation. With the images stemming from all over the globe at various times, lakes will look different visually, causing a shift in the features values that would classify a pixel.



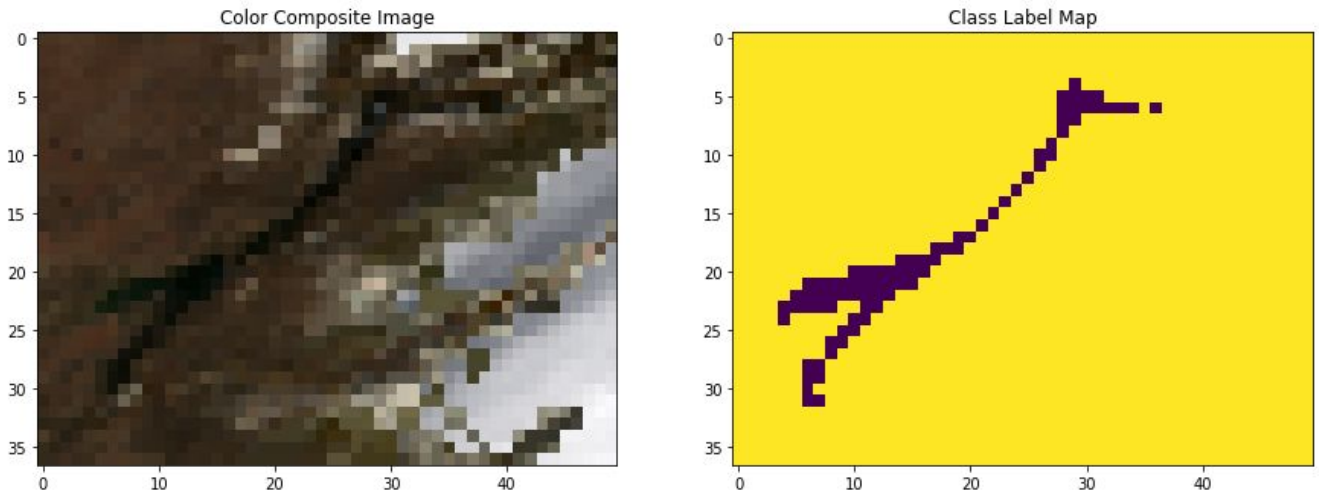


Figure 10 Lakes Present in Different Domains: Two images are presented using their RGB color spectrum, as well as the class labels. One lake is taken in the night time, while the other lake is taken what appears to be during a snowy weather. Purple indicates water, yellow indicates land.

The training set used to learn the features of what constitutes land or lake is highly dependent on how representative the training set is on the test set. Learning the features on lakes in the night time will cause a poor performance on the test set if the lakes are taken in the day. Various methodologies have been proposed for a model to learn the features in a domain invariant region which could be applied to further improve on the FCN's accuracy[6].

FCN Optimization

One important aspect not thoroughly investigated is the optimization of layers, hidden units, and filter sizes of the FCN. Larger patches would allow the ability to utilize kernels with bigger windows, providing features with more spatial context, and less parameters per layer. It's quite possible that each individual patch size has its own optimized machinery which composes of different number of hidden layers, hidden units per layer, and filter sizes. This was kept the same throughout experimentation of patch size dimensions in order to ensure the same machinery could be applied to every patch dimension.

Concluding Remarks

The Fully Convolutional Network performs reasonably well in determining the pixels that are land or water. The FCN relies upon using patches in order to correctly classify pixels. Using four convolutional layers, followed by three deconvolutional layers, the FCN performs better when utilizing all seven bands in comparison to only using the RGB bands. Thirdly, a logistic regression model performs worse than the FCN, however, there may be reasons to indicate spatial context may not be as big of a boon. FCN outperforms U-NET, a popular deep learning model. Lastly, FCN has visually performs similar, but slightly worse than the in-house models of the NASA and Kumar's lab. The most important consideration is that there are limited number of quality lakes in the dataset, and the FCN may prove to be more useful as more images are utilized.

[6] Tzeng, Eric, et al. "Adversarial Discriminative Domain Adaptation." *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, doi:10.1109/cvpr.2017.316.

Works Cited

Jocicmarko. “Jocicmarko/Ultrasound-Nerve-Segmentation.” *GitHub*, 16 Aug. 2017, github.com/jocicmarko/ultrasound-nerve-segmentation.

Maggiori, Emmanuel, et al. “Fully Convolutional Neural Networks for Remote Sensing Image Classification.” *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 2016, doi:10.1109/igarss.2016.7730322.

“MODIS Web.” *Https://Modis.gsfc.nasa.gov/about/*, NASA, modis.gsfc.nasa.gov/about/.

Ronneberger, Olaf, et al. “U-Net: Convolutional Networks for Biomedical Image Segmentation.” *Lecture Notes in Computer Science Medical Image Computing and Computer-Assisted Intervention MICCAI 2015*, 2015, pp. 234–241., doi:10.1007/978-3-319-24574-4_28.

Tzeng, Eric, et al. “Adversarial Discriminative Domain Adaptation.” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, doi:10.1109/cvpr.2017.316.

“Vipin Kumar's Home Page.” *Vipin Kumar's Home Page*, www-users.cs.umn.edu/~kumar001/.

Appendix

Github repository (UMN): https://github.umn.edu/bharg016/CSCI-8980-Project

FCN Code

```
def FCN_build(train_X):

    model = Sequential()
    model.add(Conv2D(64, kernel_size=(3, 3),
                     activation='relu',
                     input_shape= train_X.shape[1:]))
    model.add(BatchNormalization(axis = 1, momentum = 0.99, epsilon = .001))
    model.add(Conv2D(128, kernel_size = (2,2), activation = 'relu'))
    model.add(Conv2D(112, kernel_size = (2,2), activation = 'relu'))
    model.add(Conv2D(112, kernel_size = (3,3), activation = 'relu'))
    model.add(Conv2DTranspose(112,(4,4), activation ='relu'))
    model.add(Conv2DTranspose(64,(3,3), activation = 'relu'))
    model.add(Conv2DTranspose(3,(2,2), activation = 'relu'))
    model.add(Reshape(((train_X.shape[1] * train_X.shape[2]),3)))
    model.add(Activation('softmax') )


    model.compile(loss = 'categorical_crossentropy', optimizer= 'adam',
metrics=['accuracy'])

    return(model)
```

Patch Creation Code

```
def Patch_creator(image, X_array, Y_array, horizontal, vertical):

    #Get Features (pixels) and labels for the image
    pixels = image['X']
    labels = image['Y']

    #Used to encode
    lb = preprocessing.LabelBinarizer()
    lb.fit([1,2,3])

    #check to see if patch size is bigger than image
    if labels.shape > (horizontal, vertical):

        #Find how many patches can be achieved vertically and horizontally
        Num_Horizontal = math.floor(labels.shape[1] / horizontal)
        Num_Vert = math.floor(labels.shape[0] / vertical)

        #initalize vertical position
        vert_pos1 = 0
        vert_pos2 = vertical -1

        #initalize horizontal position
        horz_pos1 = horizontal * -1
```

```
        horz_pos2 = -1

        for vertical_shift in range (Num_Vert):

            #\Get all horizontal shifts for each vertical shift first
            for horizontal_shift in range(Num_Horizontal):


                horz_pos1 = horz_pos1 + horizontal
                horz_pos2 = horz_pos2 + horizontal

                #slices image on dimensions to get patch
                X_patch = pixels[vert_pos1:vert_pos2,
                                 horz_pos1 :horz_pos2, :].reshape(
                                 1,vertical-1,horizontal-1,pixels.shape[2])

                #Binzaration and Encode classes (3)
                Y_patch = labels[vert_pos1:vert_pos2,
                                 horz_pos1:horz_pos2].reshape((vertical -1) *
                                 (horizontal -1))
                Encoded_Y_patch = lb.transform(Y_patch)
                Encoded_Y_patch = Encoded_Y_patch.reshape(
                        1,Encoded_Y_patch.shape[0], Encoded_Y_patch.shape[1])

                if sum(Encoded_Y_patch[0,:])[1] != Encoded_Y_patch.shape[1] and
sum(Encoded_Y_patch[0,:])[0] != Encoded_Y_patch.shape[1] and
sum(Encoded_Y_patch[0,:])[2] != Encoded_Y_patch.shape[1]:
                    #If you have an empty array, create a new one,
                    #Else keep adding patches and info
                    if X_array == []:
                        X_array = np.array(X_patch)
                    elif X_array != []:
                        X_array = np.vstack( (X_array,X_patch))

                    if Y_array == []:
                        Y_array = np.array(Encoded_Y_patch)
                    elif Y_array != []:
                        Y_array = np.vstack( (Y_array,Encoded_Y_patch))

            #Shift vertical Patch
            vert_pos1 = vert_pos1 + vertical
            vert_pos2 = vert_pos2 + vertical

            #reset horizontal position
            horz_pos1 = (horizontal * -1)
            horz_pos2 = -1

    return (X_array, Y_array)
```