Waterbody detection using multispectral landsat surface reflectance data

1 Introduction

In the modern world, data is an ever more important notion. Data, in all it's different forms, is crucial for all industries. With the increase in data usage, the size of the datasets processed is also rising rapidly, meaning data processing and analysis need to keep up with the pace. More specifically, in the field of geoinformatics, data processing is a massive task. Data sets can contain countless amounts of data points in need of manipulation and interpreting. This report will focus on analyzing satellite data in raster format. Working with rasters usually consists of manipulating the data and performing calculations based on it. This is often a tedious process to do manually, which makes it a great candidate for being done by a machine learning model.

As for the structure, section 2 will describe the problem and the data selection, section 3 will discuss feature and label selection, section 4 will describe the data preprocessing, section 5 will outline the selected models, section 6 will analyze the results and finally section 7 will contain the conclusion.

2 Problem formulation & data selection

This report will outline an attempt at utilizing machine learning for detecting water bodies from satellite imagery. The satellite data selected is the landsat collection 2 level 2 imagery, displaying surface reflectance values of various bandwidths of light. The data was acquired from "USGS earth explorer", was gathered in june 2021 and the area in question is Espoo, Finland.

The task of detecting water bodies from surface reflectance data manually, can require data manipulation in the form of creating a water mask. A water mask is created by only assigning values for pixels with a surface reflectance value corresponding to water, thereafter displaying the resulting data where only water bodies are visible. By utilizing a machine learning model on the data before manipulation, the additional task of manually creating a water mask is avoided.

3 Feature & label selection

The landsat 2 level 2 satellite imagery chosen for this application includes a wide variety of bandwidths. In order to keep the data sets used in the machine learning model at a reasonable size, only 3 bandwidths were chosen to be incorporated into the feature matrix. Ideally more bandwidths would be incorporated to achieve more detailed data. The 3 chosen bandwidths are the RGB bandwidths, i.e. the red, green and blue light. Each band is in the form of a raster image (.tif file), which consists of pixels, each displaying a float value. This means that each data point in the feature matrix has three float values, displaying the respective bands surface reflectance values.

The label vector consists of only binary values. Where 1 is assigned to the pixels displaying water, and 0 for pixels not displaying water. The label vector is constructed from a manually created water mask.

4 Data preprocessing

In order to get the data from the satellite imagery, the GIS software ”QGIS”, and python were used. In QGIS, after cropping the imagery into smaller layers (453x666), a raster calculator was used to calculate the surface reflectance values for each band that is used as a feature. These bands were then exported as .tif files, after which they were converted into .csv files using python.

The label vector was created in a similar fashion, however an additional step was taken to convert the surface reflectance values into a water mask. Using a raster calculator in QGIS, rasters with a surface reflectance value of less than 0.02 were assigned the value 1, while all remaining values were assigned the value 0. The single band used for the water mask was band number 6, i.e. the shortwave-infrared band, as it is generally the best bandwidth for detecting water.

After creating the appropriate data structures, the data was split into training, validation and testing data. The chosen split of the data was an initial 85, 15 meaning that the training and validation data consists of 85% of the original data set and the testing 15%. Then the 85% of the data, i.e. the training and validation data was further split 85, 15 into specific training and validation data using scikit learns ”train_test_split” method. The reasoning for the more complicated split, is that the Random Forest Classifier uses the combined training and validation set for its training, as the hyperparameter tuning is done using scikit learns ”RandomizedSearchCV” method, that tries to find the optimal hyperparameters from a given set of parameter distributions. The Convolutional Neural Network uses the training and validation sets separately. The reason for allocating 85% of the data for training and validation, is that the models do not need a significant amount of data to be evaluated on, and the data is better used for training and hyperparameter tuning.

The training and validation, training, validation and testing sets have the following shapes: (256398, 3), (217938, 3), (38460, 3) and (45300, 3). Where the corresponding label vectors naturally have the same amount of rows, although only one column for the binary values. The testing set has retained its spatial order, thus the resulting predictions can be reconstructed into raster images for visual interpretation.

5 Models

5.1 Random Forest Classifier

The random forest classifier is a supervised machine learning algorithm. The algorithms works by creating decision trees from random subsets of the training set, this means that it ideally helps prevent the dominance of any single feature, making the model more balanced. Using the created decision trees, the finished model can make predictions about data, by allowing each decision tree to cast a ”vote” about the prediction. The label type with the most ”votes” is then selected as the prediction. Therefore, the hypothesis space of a rfc is the collection of decision trees created from the training data.

The reason for choosing the model in question, is in part it’s ability to ”maintain high predictive accuracy while minimizing overfitting” [1]. Additionally, the model should have a high tolerance to outliers, as predictions are the average over many different decision trees. This is advantageous in the chosen application, as wrong predictions are easily visible.

A random forest classifier does not require a loss function, as it internally optimizes itself using an impurity measure. For sklearn:s random forest classifier, the default is gini impurity [2], and will be used in this model. Gini impurity ”calculates the amount of probability of a specific feature that is classified incorrectly when selected randomly” [3]. In other words, when building decision trees, the model actively calculates the best action, or ”split”, depending on the gini impurity.

5.2 Convolutional Neural Network

The convolutional neural network (CNN), is a supervised machine learning algorithm. The algorithm utilizes convolutional, pooling and fully-connected layers to classify the data provided. The CNN excels at image classification, as it is able to infer a lot of information from an image, by processing the data in multiple levels of abstraction in its different layers, and thereby extracting complex features from the data. The CNN is thus a great candidate for this application, as the data consists of pixel data from images.

The CNN model used in this application consists of two convolutional layers with ReLu activation functions to introduce non-linearity, 2 pooling layers to reduce the spatial dimension, one flattening layer to transform the data into 1D and finally 2 fully-connected layers to classify the data, where the last layer has the sigmoid activation function for binary classification. The hypothesis space is ultimately the mapping of the input values through the weights and configurations of the CNN to a probability of belonging to either binary label. Thus the outputted probalilistic values have to be "squeezed" to fit the desired output of binary values.

The loss function of the CNN used in this report is binary crossentropy, more commonly know as log loss. Log loss is a loss function for binary classification, where the output is the probability of a data point belonging to either label, making it ideal for this application. Log loss tries to minimize the loss by penalizing the model when the distance between the probabilities and true labels is high.

6 Results

The models will be evaluated based on accuracy, precision and recall. Accuracy assesses the overall accuracy of the model, without weighted distinguishing between positives and negatives, while precision measures the proportion of positive predictions that were correct. Recall measures the proportion of correct positive predictions out of the actual amount of positives, thus also taking into account the false negatives. The training and validation evaluation metrics are not fully comparable, due to the models differing hyperparameter tuning. Thus more weight will be put on the testing when choosing the better model.

Table 1: Evaluation metrics training

Model	**Accuracy Training**	**Precision Training**	**Recall Training**
RFC	0.992	0.975	0.941
CNN	0.984	0.930	0.898

Table 2: Evaluation metrics validation

Model	**Accuracy Validation**	**Precision Validation**	**Recall Validation**
CNN	0.983	0.916	0.898

Table 3: Evaluation metrics testing

Model	**Accuracy Testing**	**Precision Testing**	**Recall Testing**
RFC	0.962	0.918	0.544
CNN	0.986	0.898	0.917

The results from the evaluation metrics on the training and validation sets show very little difference between the two models, however direct comparison is limited, as the models differ in the approach of training and validation.

The results clearly show that the CNN is significantly outperforming the rfc in the recall metric of the testing set. This means that the CNN is significantly better at predicting the true positives, while simultaneously having a low amount of false negative predictions on unseen data. The rfc holds up well in the accuracy and precision on the testing set, even having slightly better precision, meaning the rfc predicts less false positives relative to true positives. As the CNN outperforms the rfc on the accuracy metric and significantly outperforms it on the recall metric on the testing set, the CNN is chosen to be the best model.

7 Conclusion

This report compared and evaluated the use of a Random Forest Classifier and Convolutional Neural Network to classify water bodies from an input of three bands of reflectance values. The results clearly stated that the CNN was the superior model for this task, boasting a testing accuracy of 98.6% and consequently a test error of only 1.4%, and a great recall value of 0.917. Both models performed well, both having above 95% accuracy, indicating that the experiment as whole was successful, however there are areas of improvement.

The CNN outperforming the rfc was not a surprise, as the CNN is far more complex and able to capture more detail. The rfc fell short on the unseen data, even though the training errors were relatively small, possibly due to imbalanced data. Thus the rfc would likely perform better with more feature engineering. The main limitation of the rfc is, however, most likely the amount of data provided. I believe the rfc would benefit if it were to get more bands to work with, as the data would be more detailed, allowing the model to build more accurate decision trees to aid the classification. The CNN would most likely also benefit from an increased variety in the data, likely resulting in less outliers in the output. Below are the resulting predictions reconstructed for visual interpretation in reference to the y_test water mask used as the labels for the testing set.

Figure 1: y_test

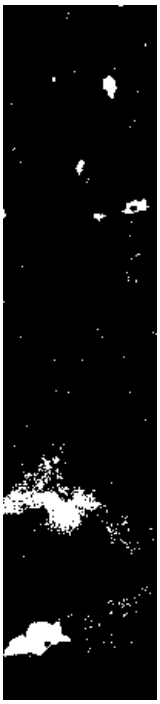

Figure 2: RFC prediction

Figure 3: CNN prediction

Sources:

[1]
Geeksforgeeks.org - Random Forest Classifier using Scikit-learn. 2024 (last updated)
https://www.geeksforgeeks.org/random-forest-classifier-using-scikit-learn/

[2]
Scikit-learn.org - RandomForestClassifier
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[3]
Neelam Tyagi. Understanding the Gini Index and Information Gain in Decision Trees. 2020.
https://medium.com/analytics-steps/understanding-the-gini-index-and-information-gain-in-decision-trees-ab4720518ba8

[4]
TensorFlow - Convolutional Neural Network (CNN)
https://www.tensorflow.org/tutorials/images/cnn

[5]
freeCodeCamp - Binary Classification with TensorFlow Tutorial
https://www.freecodecamp.org/news/binary-classification-made-simple-with-tensorflow/

[6]
kaggle - Binary Classification using CNN for beginners
https://www.kaggle.com/code/alifrahman/binary-classification-using-cnn-for-beginners

[7]
scikit-learn - RandomizedSearchCV
https://scikit-learn.org/1.5/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

[8]
rasterio - Profiles and Writing Files
https://rasterio.readthedocs.io/en/stable/topics/profiles.html

[9]
geeksforgeeks - Training of Convolutional Neural Network (CNN) in TensorFlow
https://www.geeksforgeeks.org/training-of-convolutional-neural-network-cnn-in-tensorflow/

A Code appendix

This section contains the code used in the project. Subsection A1 is the code for transforming the .tif files into .csv files, and subsection A2 is the code used for the models.

A.1

```
imagewatermask = Image.open(r"C:\Desktop\ML_GIS_Projects\B6_SR_watermask.tif")

image_red = Image.open(r"C:\Desktop\ML_GIS_Projects\ML_GIS_V2\B4_SR.tif")
image_red_2 = image_red.convert("L") #this is not needed for B6_SR. since it only has one band
image_green = Image.open(r"C:\Desktop\ML_GIS_Projects\ML_GIS_V2\B3_SR.tif")
image_green_2 = image_green.convert("L")
image_blue = Image.open(r"C:\Desktop\ML_GIS_Projects\ML_GIS_V2\B2_SR.tif")
image_blue_2 = image_blue.convert("L")

pixel_data_red = np.array(image_red_2)
pixel_data_green = np.array(image_green_2)
pixel_data_blue = np.array(image_blue_2)
pixel_data_labels = np.array(imagewatermask)

np.savetxt("featuresRED.csv", pixel_data_red, delimiter=",", fmt="%.8f")
np.savetxt("featuresGREEN.csv", pixel_data_green, delimiter=",", fmt="%.8f")
np.savetxt("featuresBLUE.csv", pixel_data_blue, delimiter=",", fmt="%.8f")
np.savetxt("labels.csv", pixel_data_labels, delimiter=",", fmt="%.8f")
```

A.2

Project

October 7, 2024

[ ]:
```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪ConfusionMatrixDisplay, precision_score, recall_score
from sklearn.model_selection import RandomizedSearchCV, train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
import rasterio
from rasterio.transform import from_origin
import tensorflow as tf
import keras
from keras import layers
from keras.models import Sequential
from keras.layers import Dense
from sklearn.utils import class_weight
from keras.metrics import Precision, Recall
```

[2]:
```
height, width = 453, 666

array_red = np.loadtxt('featuresRED.csv', delimiter=',')
array_green = np.loadtxt('featuresGREEN.csv', delimiter=',')
array_blue = np.loadtxt('featuresBLUE.csv', delimiter=',')
array_labels = np.loadtxt('labels.csv', delimiter=',')

train_val = slice(0, int(0.85 * width))
testing = slice(int(0.85 * width), width)
```

[3]:
```
#reshaping and sclicing the testing arrays

X_test_red = array_red[:, testing].reshape(-1, 1)

X_test_green = array_green[:, testing].reshape(-1, 1)

X_test_blue = array_blue[:, testing].reshape(-1, 1)

y_test = array_labels[:, testing].reshape(-1)
```

[4]:
```
#creating the training, validation and testing sets

X_test = np.hstack((X_test_red, X_test_green, X_test_blue))

train_val_red = array_red[:, train_val].reshape(-1, 1)
train_val_green = array_green[:, train_val].reshape(-1, 1)
train_val_blue = array_blue[:, train_val].reshape(-1, 1)

y_train_val = array_labels[:, train_val].reshape(-1)
X_train_val = np.hstack((train_val_red, train_val_green, train_val_blue))

#Creating the testing and validation sets with a 85, 15 split

X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,␣
 ↪test_size=0.15, random_state=40)
```

[6]:
```
#Tuning the hyperparameters using RandomizedSearhCV

n_estimators = np.linspace(10, 150, 20, dtype=int)
max_depth = [3, 10, 20, 30]
min_samples_split = np.linspace(2, 50, 10, dtype=int)
min_samples_leaf = [1, 2, 5, 9]
max_features = ['sqrt', 'log2']

param_distributions = dict(n_estimators=n_estimators, max_depth=max_depth,␣
 ↪min_samples_split=min_samples_split, min_samples_leaf=min_samples_leaf,␣
 ↪max_features=max_features)

rfc_tune = RandomForestClassifier(n_estimators=n_estimators,␣
 ↪max_depth=max_depth, min_samples_split=min_samples_split,␣
 ↪min_samples_leaf=min_samples_leaf, max_features=max_features)

grid_random = RandomizedSearchCV(estimator=rfc_tune,␣
 ↪param_distributions=param_distributions, n_iter=50, cv=2, n_jobs=-1)

grid_random_tuned_results = grid_random.fit(X_train_val, y_train_val)

best_rfc_random = grid_random_tuned_results.best_estimator_
```

[7]:
```
#fitting and predicting using the best rfc

rfc = best_rfc_random
rfc.fit(X_train_val, y_train_val)

y_pred_rfc = rfc.predict(X_test)
y_pred_rfc_training = rfc.predict(X_train_val)
```

[259]:
```
#Convolutional Neural Network
```

[8]:
```
#Initializing the CNN model

CNN = Sequential()

CNN.add(layers.Conv2D(32, (2, 1), activation='relu', input_shape=(3, 1, 1)))
CNN.add(layers.MaxPooling2D((1, 1)))
CNN.add(layers.Conv2D(64, (2, 1), activation='relu'))
CNN.add(layers.MaxPooling2D((1, 1)))
CNN.add(layers.Flatten())
CNN.add(layers.Dense(128, activation='relu'))
CNN.add(layers.Dense(1, activation='sigmoid'))

optimizer = keras.optimizers.Adam(learning_rate=0.001)

CNN.compile(loss='binary_crossentropy', optimizer=optimizer,␣
 ↪metrics=["accuracy", Precision(), Recall()])
```

[ ]:
```
#Training and validating the CNN

X = X_train.reshape((217938, 3, 1))
X_v = X_val.reshape((38460, 3, 1))

training_history = CNN.fit(X, y_train, validation_data=(X_v, y_val), epochs=10,␣
 ↪batch_size=32)
```

[ ]:
```
#Predicting and squeezing the labels to obtain binary values

y_pred_CNN = CNN.predict(X_test.reshape((45300, 3, 1)))

y_pred_CNN = tf.squeeze(y_pred_CNN)
y_pred_CNN = np.array([1 if x >= 0.5 else 0 for x in y_pred_CNN])
```

[ ]:
```
#calculating accuracy and plotting the confusion matrices

print(training_history.history.keys())
CNN_train_accuracy = training_history.history['accuracy']
CNN_val_accuracy = training_history.history['val_accuracy']

CNN_train_precision = training_history.history['precision']
CNN_val_precision = training_history.history['val_precision']

CNN_train_recall = training_history.history["recall"]
CNN_val_recall = training_history.history["val_recall"]

conf_mat_rfc = confusion_matrix(y_test, y_pred_rfc)
```

```
ax = plt.subplot()
sns.heatmap(conf_mat_rfc,annot=True, fmt='g', ax=ax)
ax.set_xlabel('Predicted labels rfc',fontsize=15)
ax.set_ylabel('True labels',fontsize=15)
plt.show()

conf_mat_CNN = confusion_matrix(y_test, y_pred_CNN)

ax = plt.subplot()
sns.heatmap(conf_mat_CNN,annot=True, fmt='g', ax=ax)
ax.set_xlabel('Predicted labels CNN',fontsize=15)
ax.set_ylabel('True labels',fontsize=15)
plt.show()

#Printing the evaluation metrics

print(f"The training accuracy of rfc is {accuracy_score(y_train_val,␣
 ↪y_pred_rfc_training)}")
print(f"The training precision of rfc is {precision_score(y_train_val,␣
 ↪y_pred_rfc_training)}")
print(f"The training recall of rfc is {recall_score(y_train_val,␣
 ↪y_pred_rfc_training)}")
print(f"The training accuracy of CNN is {CNN_train_accuracy}")
print(f"The training precision of CNN is {CNN_train_precision}")
print(f"The training recall of CNN is {CNN_train_recall}")

print(f"The validation accuracy of CNN is {CNN_val_accuracy}")
print(f"The validation precision of CNN is {CNN_val_precision}")
print(f"The validation recall of CNN is {CNN_val_recall}")

print(f"The testing accuracy of rfc is {accuracy_score(y_test, y_pred_rfc)}")
print(f"The testing precision of rfc is {precision_score(y_test, y_pred_rfc)}")
print(f"The testing recall of rfc is {recall_score(y_test, y_pred_rfc)}")
print(f"The testing accuracy of CNN is {accuracy_score(y_test, y_pred_CNN)}")
print(f"The testing precision of CNN is {precision_score(y_test, y_pred_CNN)}")
print(f"The testing recall of CNN is {recall_score(y_test, y_pred_CNN)}")
```

[226]:
```
#reconstruct images

height, width = 453, 100

y_reshaped_rfc = y_pred_rfc.reshape((height, width))

metadata = {
    'driver': 'GTiff',
    'count': 1,
```

```
    'dtype': 'float32',
    'width': width,
    'height': height,
    'crs': 'EPSG:32635',
    'transform': from_origin(0, height, 1, 1)  #doesnt need coordinates

with rasterio.open("prediction_rfc.tif", "w", **metadata) as dst:
    dst.write(y_reshaped_rfc, 1)

y_reshaped_CNN = y_pred_CNN.reshape((height, width))

metadata = {
    'driver': 'GTiff',
    'count': 1,
    'dtype': 'float32',
    'width': width,
    'height': height,
    'crs': 'EPSG:32635',
    'transform': from_origin(0, height, 1, 1)  #doesnt need coordinates
}

with rasterio.open("prediction_CNN.tif", "w", **metadata) as dst:
    dst.write(y_reshaped_CNN, 1)
```