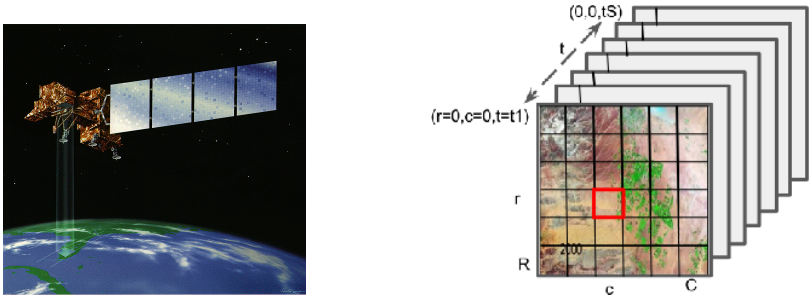
LULC-Net: Generalized convolution neural network classfier for detecting land use changes from satellite data

Rishu Saxena

Convolution neural networks (CNNs) have recently been applied in a wide variety of complex, previously intractable, tasks such as image recognition and semantic labeling. This report presents preliminary findings on a novel application of CNN — classifying locations on Earth's surface as those with change or no change (stable) using images obtained from satellite data. Land use change is described as changes in how humans use the surface of the Earth (e.g., for agriculture, plantations, pastures, managed woods, conservation, settlements, or leaving it alone as natural ecosystem). Changes in land use lead to changes in albedo, thereby directly affecting the temperatures of the surrounding area. Significant and lasting changes in land use and land cover (LULC) have more profound effects. The past century has seen an exponential growth in human activities such as deforestation and urbanization causing significant changes in land cover in several parts of the world [**?**]. Simultaneously, significant changes in the global climate have also been observed, driven in part by LULC change (LULCC) (e.g., [**?**]). LULCC also has impacts on a wide variety of other ecosystem services. Monitoring LULCC across the globe, therefore, has become the need *du jour*. Land use change detection comprises any methodology used for determining the occurance and nature of change in LULC.

Earth observation satellites (EOS) such as Landsat capture images of the Earth's surface at regular intervals using multiple spectral frequencies (Figure **??**(a)). These images hold valuable information that, if harnessed well, can be immensely helpful in understanding, monitoring, and managing our natural resources, as well as studying LULCC. An excellent way of analyzing these satellite images for LULCC studies is time series analysis (or, temporal trajectory analysis). For time series analysis, several images of the scene under consideration, taken over a period of time, are stacked together chronologically, and are subsequently analyzed (see figure **??**(b)). Commonly, the time series for each pixel is treated individually; the full image stack is thus a collection of many time series. The choice of spectral band(s) varies from application to application. The objective is to discover a 'trend' in how different relevant variables (indicators) evolve over time. Analysis is based on the behaviours of the time series of these variables. In change detection analysis, when the trajectory of one or more of the variables departs from the normal, a change is detected. Time series analysis for LULCC studies has been receiving increasing attention in the last decade, specifically, after the Landsat data became freely accessible in 2008 [**?**]. Several time series analysis algorithms have been proposed by different groups in the remote sensing community.

Despite a plethora of time series analysis algorithms available in remote sensing, design and selection of algorithms for LULCC detection in remote sensing appears to be almost always context specific. Most of the methods proposed to date seem to perform well on the type of data that they are designed for. Their performance on randomly picked datasets



(a) (b)

Figure 1: (a) Earth observation satellites take images of the Earth's surface in patches at regular interval (Source: https://www.nasa.gov/). (b) In TSA for LULCC, satellite images are stacked chronologically. The image stack yields a time series per pixel.

(a) before change (b) after change

Figure 2: Image of patch pre and post change.

from across the globe has not been studied. The onus of choosing an appropriate algorithm that will perform well on their particular dataset falls on the user. Unfortunately, no single algorithm designed so far seems to work for all datasets [**?**]. For example, the Western Antarctica as well as the Greenland Ice Sheets are beginning to collapse due to global warming, the melting leading to continually receding snow covers at the respective locations. For these regions, using LULC algorithms based on periodicity assumptions is expected to lead to incorrect predictions and/or false alarms, although the nature and extent of this has not been studied yet. Even if there were no global warming, mild shifts in the 'phase' and 'amplitude' of seasons are known to take place [**?**]. Time warping techniques [**?**] to deal with these issues may be helpful in some contexts, but their accuracy and scalability has not yet been satisfactorily investigated. Approaches based on periodicity and a moving window are possible, with additional computational costs.

This work explores the use of CNNs for LULCC. One of the key issues in using neural networks for LULCC via LandSat imagery is the lack of training data. Specifically, a ground truth/historical records dataset that provides information on the occurrance (timing and type) of change for any given location on the surface of the Earth (pixels, in LandSat image stacks) does not exist. The Landsat images themselves have significant level of noise,

missing data (for example, due to clouds), and incorrect and/or missing data (usually due to sensor malfunction). Such image quality, combined with the fact that remote sensing is still a niche field of study, makes it significantly challenging to build annotated datasets that can be used to train CNN (any change classifier, in general). This is in contrast with other major datasets/problems, such as object recognition and semantic labeling, where neural networks have been successfully used in state-of-the-art literature. To our knowledge, in most of these cases, high quality annotated images (datasets) are available on the internet (social media, Flickr, and the like). Researchers are able to build vast repositories of semantically annotated images which are then used to train the CNN.

Designing the neural network approach, validation, and scalability are the subsequent issues to be addressed.

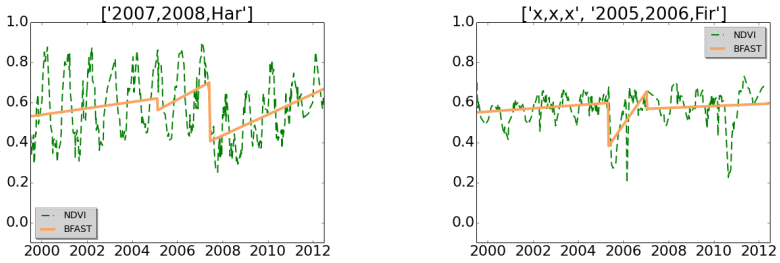
In this work, a polyalgorithmic approach is used to classify 4000 NDVI time series as stable or unstable. A canonical CNN consisting of three convolutional layers and 3 fully connected layers is the used for classifying 10000 time series as those with change or no change (stable).

The rest of this paper is organized as follows: Section 2 presents background on state-of-the-art change detection algorithms available in remote sensing. Section 3 presents the proposed neural networks based approach: training data generation, neural networks model, and the results. Conclusions and future work are presented in Section 4.

1 Generating Training data for CNN using Polyalgorithm and Timesync data

A polyalgorithm consisting of three component algorithms is for labeling 4000 pixels as 'stable' or those with 'change'. The component algorithms, viz., EWMACD, LandTrendR, and BFAST, are fundamentally unique to each other by constructions, and to some extent, in the phenomenon they capture. These algorithms were originally meant to identify the timing of change occurrence(s) in the input time series. However, in the present context, we only use them to determine whether a pixel had any change at all. Of these algorithms, BFAST is most exhaustive in its breakpoint search. We briefly discuss BFAST algorithm next.

Breaks For Additive and Seasonal Trend. Amongst the state-of-the-art LULCC detection algorithms in remote sensing, BFAST [**?**] is found to be most thorough and accurate in assessing change. This is a recuresive residual based algorithm that considers every single timepoint in the time series as a candidate breakpoint, computes the least squares residuals for fits on either side of that timepoint, and eventually selects the timepoints that yield lowest loeast squares residuals as the breakpoints. Specifically, BFAST decomposes the given time series iteratively into three components: trend, seasonal, and noise. BFAST computes and evaluates least squares fits in windows of increasing size. Qualitatively, (i) first the possibility of there being any structural change in the given time series is determined by computing the partial sums of residuals of least squares fits in windows (OLS-MOSUM). The limiting process of these partial sums is the increments of a Brownian bridge process [**?**]. If the observations do have a structural change, an ordinary linear least squares fit will result in large residuals and, hence, in large partial sums. Therefore, the occurence of large values in the process is an indication of the presence of a structural change — this probability being calculated from the Brownian bridge table. (ii) If a structural change is



(a) Harvest to Forest change

(b) Fire in non-forest vegetation

Figure 3: Figure illustrates result of BFAST algorithm on two patches that underwent land cover change.

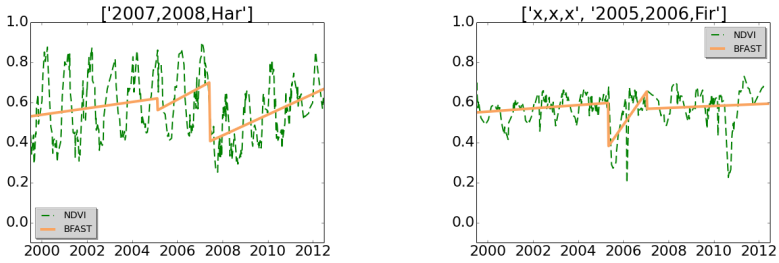
indicated, a search for change location is done. *Each* interior time point t is considered a breakpoint (change location) candidate. A recursive residual is the error at time t_j from the linear least squares fit over the window $[t_i, \ldots, t_{j-1}]$. The breakpoints (change locations) are chosen so as to minimize the sum of squared recursive residuals over all windows in between (omitting) the breakpoints. This is done for both trend and seasonal components of the time series, consecutively. A formal pseudocode for BFAST is presented in [**?**].

Timesync data. The state-of-the-art information on LULCC for a select set of pixels is provided by the TimeSync dataset [**?**]. This dataset was prepared using TimeSync Landsat time series visualization and change data collection tool [**?**]. This tool enables disturbance characterizations for pixel-level samples of Landsat time series data, relying on human interpretations of change as viewed in image chip series, spectral index trajectories, high spatial resolution image temporal snapshots from Google Earth, and other supporting products. Landsat image stacks spanning the years 1984 to 2014 and belonging to six different path/rows (scenes) are considered for preparing TimeSync dataset. From each scene, 300 pixels are chosen with random sampling, and without regard to land cover. Thus there are 1800 pixels in all included in the dataset. For each of these pixels, the following attributes are noted: occurence of disturbance, the first year of detection (a year between 1986 and 2011), the duration for gradual disturbances (in number of years), and the causal agent class (harvest, fire, mechanical, decline, wind, other). Of the 1800 pixels, 1303 pixels were forested at some point within the 26 year time period. The work presented in this article utilizes the time series of the pixels included in TimeSync data and the corresponding disturbance occurence information. The results presented are limited to the time period 2000–2012. Despite the tedious, conscentious efforts that TimeSync dataset has been prepared with, disagreement is sometimes found between the NDVI trajectory of a pixel and the Timesync change information.

2 LULC-Net: A neural network for land use change detection

We utilize a generic convolutional neural network (CNN) for our problem. Our network consists of five layers — first three convolutional, and next two fully connected.

For the convolutional layers, 32 filters each are used for the first two layers, 64 for the third layer. The convolution kernel has a receptive field of size 3×3. A stride of one pixel



(a) Harvest to Forest change

(b) Fire in non-forest vegetation

Figure 4: Figure illustrates result of BFAST algorithm on two patches that underwent land cover change.

Table 1: Runtime Configuration

software	version
GCC	5.2
CUDA	8.0.61
ATLAS	3.0.12
Python	2.7.13
PyCuda	2016.1.2
Theano	0.8.2
libgpuarray	3.0
PyGPU	0.7.5

is used. 'ReLu' (Rectified Linear Units) is used as the activation function. ReLu is defined as

$$f(x) = \max(0, x),$$

where x is the input to the neuron. Networks that train with ReLu activation allow efficient gradient propagation (no vanishing or exploding gradients) and efficient computation (only comparison, addition, and multiplication). ReLu based networks are several times faster than networks with other commonly used activation functions. Each activation is followed by a max-pooling layer. Overlapping pooling is used, with neighborhoods of size 2×2, and the centers of the neighborhoods 1 pixels apart.

The (pooled) output of the third layer is flattened out, and

Dropout is a technique used to prevent overfitting and co-adaptations of neurons by setting the output of any neuron to zero with probability p.

Keep a moving average of the squared gradient for each weight

Data augmentation and Dropout are used to reduce overfitting.

Figure 1 diplays two sample images from the training data.

2.1 Implementation in Keras

Header

```
import matplotlib
matplotlib.use("Agg")
```

```
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import backend as K
import numpy as np
```

Initialization

```
img_width, img_height = 204, 153
train_data_dir = 'rs_data/train/'
validation_data_dir = 'rs_data/validation'
test_data_dir = 'rs_data/test'
nb_train_samples = 100
nb_validation_samples = 50
epochs = 3
batch_size =1
K.set_image_data_format('channels_first')
input_shape = (3, img_width, img_height)
```

The Model

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
convout_l1_act = Activation('relu')
model.add(convout_l1_act)
convout_l1_mp = MaxPooling2D()  #(pool_size=(2, 2))
model.add(convout_l1_mp)


model.add(Conv2D(32, (3, 3)))
convout_l2_act = Activation('relu')
model.add(convout_l2_act)
convout_l2_mp = MaxPooling2D()
model.add(convout_l2_mp)  #MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
convout_l3_act = Activation('relu')
model.add(convout_l3_act)
convout_l3_mp = MaxPooling2D()  #pool_size=(2,2)
model.add(convout_l3_mp)

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
```

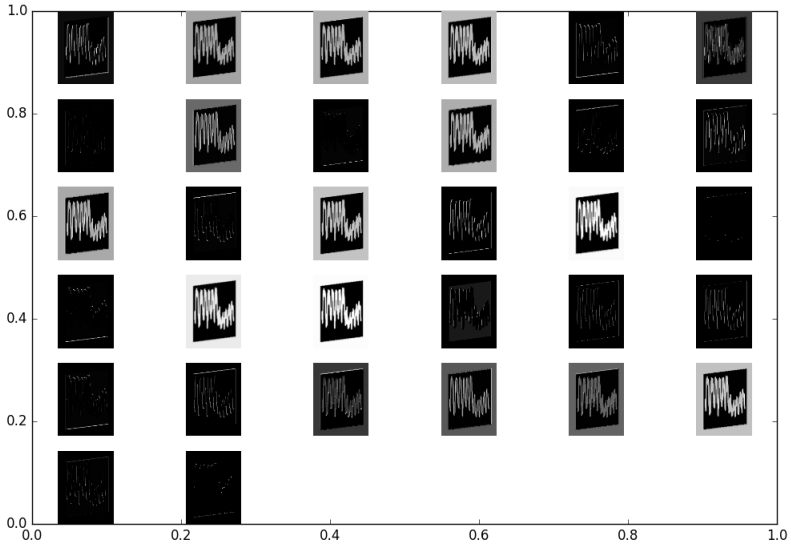
```
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop', metrics=['accuracy'])
model.summary()
```

Training and Validation

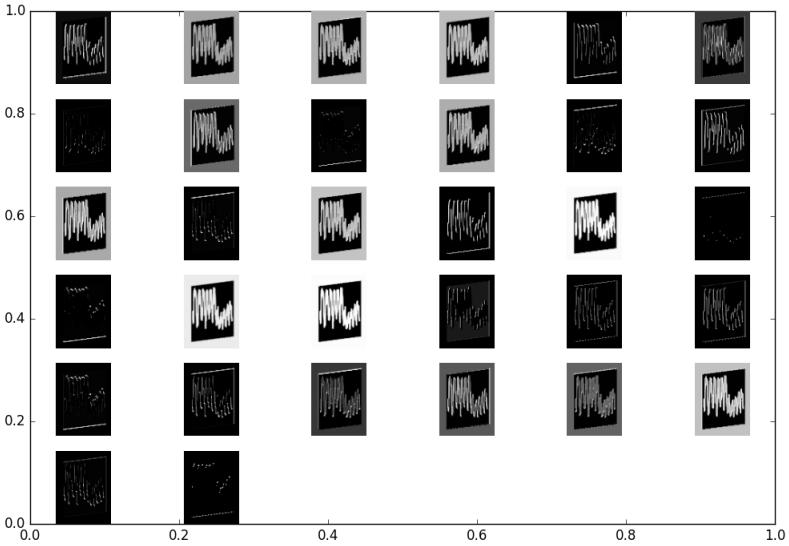
```
train_datagen = ImageDataGenerator(rescale=1./255, \
                                   shear_range=0.2,zoom_range=0.2, horizontal_flip=False)
test_datagen = ImageDataGenerator(rescale=1. / 255)
train_generator = train_datagen.flow_from_directory(train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size, shuffle=False, class_mode='binary')

validation_generator = test_datagen.flow_from_directory(validation_data_dir,
    target_size=(img_width, img_height), batch_size=batch_size, class_mode='binary')

model.fit_generator(train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs, validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size)
```

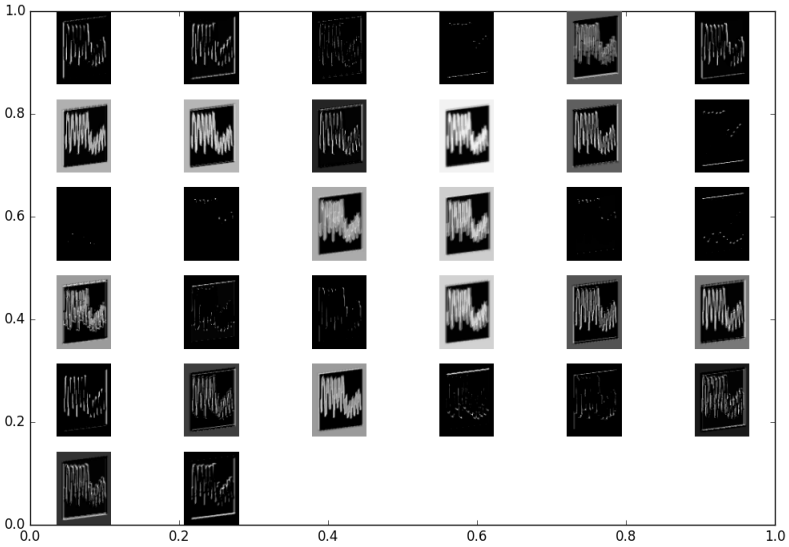
Testing

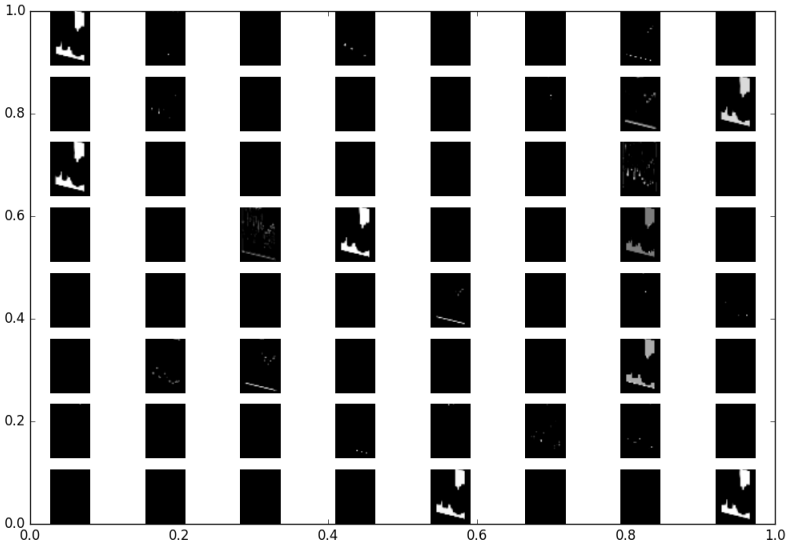
```
test_generator = test_datagen.flow_from_directory(
         test_data_dir,
         target_size=(img_width, img_height),
         batch_size=batch_size,
         class_mode=None,  # only data, no labels
         shuffle=False)  # keep data in same order as labels

probabilities = model.predict_generator(test_generator, 500)
#print probabilities

from sklearn.metrics import confusion_matrix
import numpy as np
from sklearn.metrics import classification_report

y_true = np.array([0] * 70 + [1] * 70)
y_pred = probabilities > 0.5
print(classification_report(y_true, y_pred))
tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()

print tn, fp
print fn, tp
```

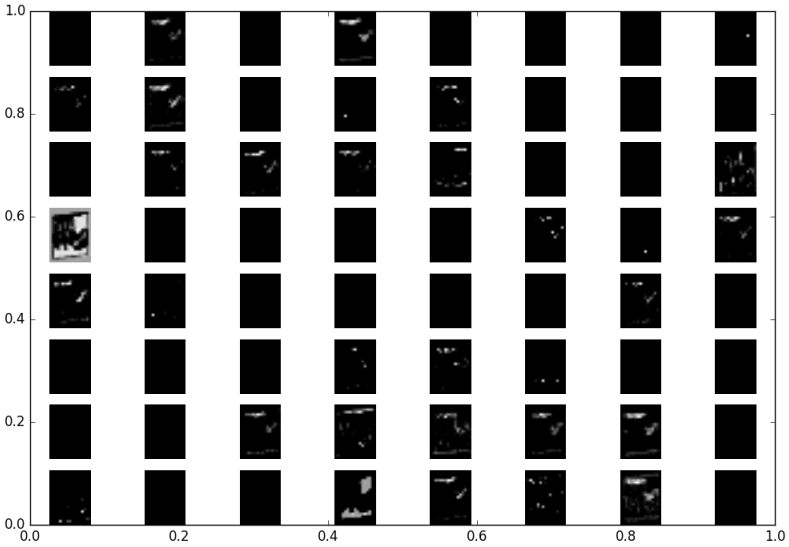


2.2 Visual interpretation of the generated model


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0