EGE ÜNİVERSİTESİ

EÜ MÜHENDİSLİK FAKÜLTESİ

LİSANS TEZİ

Detection of Change in Satellite Photos with Change Detection Algorithm

Sinan Burak Filiz

Tez Danışmanı : Aybars Uğur

Bilgisayar Mühendisliği Anabilim Dalı

Sunuş Tarihi : 23. 06. 2023

Bornova-İZMİR

2023

TEŞEKKÜR

Tezimizin başından sonuna kadar bizi yönlendiren, her türlü sorumuza cevap verip bize destek veren, önerileri sayesinde tezimizin gelişmesinde büyük katkısı olan danışmanımız Prof. Dr. Aybars Uğur ve eğitim hayatımız boyunca hem maddi hem de manevi olarak bizi destekleyen ailelerimize teşekkürlerimizi borç biliriz.

Summary

Detection of Change in Satellite Photos with Change Detection Algorithm

Filiz, Sinan Burak

Lisans Tezi, Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Prof. Dr. Aybars UĞUR

Haziran 2023

This essay delves into the application of change detection algorithms in the analysis of satellite photos to identify and characterize changes occurring on the Earth's surface. The availability of satellite imagery has revolutionized our capacity to monitor and comprehend various aspects of our planet. However, the sheer volume of data generated by satellites poses a significant challenge when it comes to manual analysis and interpretation.

Change detection algorithms provide an automated solution to this challenge. By leveraging these algorithms, we can compare satellite images captured at different time points and detect areas where significant changes have occurred. These changes can range from alterations in land cover, such as urban expansion or deforestation, to variations in water bodies, geological features, or natural disasters. The objective is to identify and classify these changes accurately and efficiently, thereby enabling us to gain valuable insights into the dynamic nature of our environment.

The process of change detection involves several steps. Initially, two or more satellite images taken at different times are selected for comparison. The images are preprocessed to correct for radiometric and geometric distortions and to ensure proper alignment. Subsequently, change detection algorithms are applied to analyze the pixel-level differences between the images. These algorithms can employ various techniques, including spectral differencing, image thresholding, machine learning, or advanced deep learning models.

The output of the change detection process is a map highlighting areas of significant change. These areas can be further classified based on specific criteria, such as the magnitude or type of change. This information is invaluable for a wide range of applications, including land management, environmental monitoring, urban planning, disaster response, and natural resource assessment.

In conclusion, change detection algorithms provide an automated and efficient means of analyzing satellite imagery to detect and classify significant changes on

the Earth's surface. By harnessing the power of these algorithms, we can overcome the challenges posed by the vast amount of satellite data, thereby enabling us to monitor, understand, and manage our environment effectively.

Keywords: Change detection, Satellite photos, Change detection algorithms, Pixel-based change detection, Object-based change detection, Multispectral change detection, Urban monitoring, Environmental monitoring, Machine learning, Accuracy.

Abstract:

This essay explores the realm of change detection in satellite photos through the utilization of advanced change detection algorithms. As the volume of satellite data continues to exponentially increase, manual analysis of these images has become impractical and time-consuming. Consequently, automated change detection methods have emerged as a solution to effectively analyze and interpret the vast amount of satellite imagery. These algorithms enable the detection of significant changes by comparing images captured at different time points. By automating the process, change detection algorithms improve efficiency and accuracy, facilitating various applications in fields such as environmental monitoring, urban planning, and disaster management.

Table of Contents

Introduction

Satellite imagery has transformed our capacity to observe and comprehend the dynamic nature of the Earth's surface, providing a wealth of information about various phenomena over extensive spatial and temporal scales. However, the rapid increase in the volume of satellite data presents a significant obstacle to manual interpretation and analysis. As a result, the development of change detection algorithms has emerged as a crucial solution to efficiently and effectively analyze these vast collections of satellite images.

Change detection algorithms offer automated techniques to identify, analyze, and quantify changes that occur between multiple satellite photos taken at different time points. These algorithms utilize a range of computational methods and statistical models to compare the pixel values, spectral characteristics, and spatial patterns in the images. By detecting and quantifying changes automatically, these algorithms significantly enhance the efficiency and accuracy of satellite imagery analysis.[1]

Understanding Change Detection

Change detection entails the comparison of two or more satellite images to identify and quantify alterations that have occurred over time. By detecting changes, researchers and decision-makers gain valuable insights into various phenomena, including land cover dynamics, urban expansion, deforestation, agricultural practices, and natural disasters. It enables us to monitor trends, assess the impact of human activities and environmental changes, and formulate appropriate strategies for resource management and disaster response.[3]

Change Detection Algorithms

Several change detection algorithms have been developed to automatically analyze satellite imagery and identify changes effectively. These algorithms can be broadly classified into two categories: pixel-based and object-based change detection methods.

Pixel-based Change Detection

Pixel-based algorithms compare individual pixels between two images to identify changes. Commonly used techniques include:

a) Image differencing:

This method subtracts the pixel values of two images to create a difference image, where significant changes appear as distinct features. It is a simple yet effective approach for detecting changes, especially in urban areas.

b) Change vector analysis (CVA):

CVA compares the spectral characteristics of corresponding pixels in two images and measures the magnitude and direction of change. It considers both magnitude and direction of change, making it suitable for detecting subtle changes.

c)Principal Component Analysis (PCA):

PCA transforms multi-band satellite imagery into a reduced set of orthogonal components. By comparing the principal components of two images, changes can be identified. PCA is particularly useful when dealing with high-dimensional data.

Object-based Change Detection

Object-based change detection methods group pixels into meaningful objects or regions and analyze changes at a higher level. These algorithms consider spatial relationships and contextual information, resulting in more accurate change detection. Popular techniques include:

a) Image segmentation:

Image segmentation divides the satellite image into homogeneous regions based on pixel properties such as color, texture, and shape. Changes in the object boundaries or properties can indicate significant alterations.

b) Feature extraction and classification:

This method extracts features from objects or regions, such as shape, size, and texture, and uses machine learning algorithms to classify these features into change or non-change categories. Support Vector Machines (SVM) and Random Forests are commonly used classifiers.

c)Hybrid approaches:

Hybrid approaches combine pixel-based and object-based methods to take advantage of their respective strengths. They integrate pixel-level change detection with object-based analysis, resulting in improved accuracy and efficiency.[12]

Applications and Advancements

Change detection algorithms have had a transformative impact on the field of remote sensing and satellite imagery analysis. These algorithms have revolutionized various applications, ranging from urban planning and disaster response to climate change monitoring and ecosystem management. By identifying and quantifying changes that occur over time in satellite images, change detection techniques provide valuable insights for decision-making processes and resource management.

In recent years, the emergence of deep learning and artificial intelligence has further advanced the field of change detection. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are two notable examples of deep learning algorithms that have demonstrated exceptional capabilities in automatically learning intricate patterns and improving the accuracy of change detection tasks. By leveraging the power of these advanced algorithms, analysts and researchers can effectively detect and analyze subtle changes in satellite imagery that might otherwise be difficult to identify through traditional methods.

The integration of deep learning and artificial intelligence with change detection algorithms has opened up new avenues for remote sensing analysis, allowing for more efficient and accurate processing of large-scale satellite datasets. This, in turn, enables better monitoring of environmental changes, identification of urban development patterns, assessment of natural disasters, and understanding the impacts of climate change on our ecosystems. The continuous advancements in these technologies hold great promise for the future of satellite imagery analysis and its applications in addressing critical global challenges.[15][16]

The Change Detection Algorithm

a) Onera Satellite Change Detection Dataset:

The dataset provided for this project includes a total of 14 pairs of training images and 10 pairs of test images. All of these images are sourced from the SENTINEL-2 satellite. The SENTINEL-2 satellite, launched as part of the European Commission's Copernicus program, is equipped with advanced sensors capable of capturing high-resolution imagery in various spectral bands, including the visible, near-infrared (VNIR), and short-wave infrared (SWIR) ranges. The dataset, derived from the satellite's observations, offers valuable data for analysis and experimentation in the context of this project.

b) SENTINEL-2:

The SENTINEL-2 satellites were launched on June 23, 2015, as part of the Copernicus program initiated by the European Commission. These satellites are designed to capture high-resolution imagery for Earth observation purposes. The sensor on board the SENTINEL-2 satellites is capable of acquiring images at different spatial resolutions, including 10, 20, and 60 meters. What sets the SENTINEL-2 apart is its ability to capture images not only in the visible spectrum but also in the near-infrared (VNIR) and short-wave infrared (SWIR) spectral ranges. As a result, the images obtained from the SENTINEL-2 satellites consist of 13 channels, rather than the traditional three channels (RGB).

Figure 1 provides information about the resolution and central wavelengths of these 13 channels. With an average orbit height of 785 km, the SENTINEL-2 satellites can cover the Earth's surface with repeated surveys every 5 days at the equator and every 2-3 days at middle latitudes [10].

In recent years, advancements in artificial intelligence, particularly in the field of computer vision, have led to the development of sophisticated change detection algorithms. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are examples of such algorithms that can automatically learn intricate patterns and significantly enhance the accuracy of change detection processes. These AI-based techniques enable automated and efficient analysis of the vast amount of data collected by satellites like SENTINEL-2, enabling us to monitor and detect changes on Earth's surface in a more precise and timely manner.

Sentinel-2 Bands	Central Wavelength (µm)	Resolution (m)
Band 1 - Coastal aerosol	0.443	60
Band 2 - Blue	0.490	10
Band 3 - Green	0.560	10
Band 4 - Red	0.665	10
Band 5 - Vegetation Red Edge	0.705	20
Band 6 - Vegetation Red Edge	0.740	20
Band 7 - Vegetation Red Edge	0.783	20
Band 8 - NIR	0.842	10
Band 8A - Vegetation Red Edge	0.865	20
Band 9 - Water vapour	0.945	60
Band 10 - SWIR - Cirrus	1.375	60
Band 11 - SWIR	1.610	20
Band 12 - SWIR	2.190	20

Figure 1 SENTINEL-2 Band Specifications

c) The Data:

I have been given access to the Onera Satellite Change Detection Dataset, which comprises a total of 14 pairs of training images and 10 pairs of test images. One of the image pairs is visually represented in Figure 3 for reference. What makes this dataset unique is that within the set of training images, there is a corresponding ground truth image specifically annotated to highlight only the man-made differences present in the image. For Figure 2, the corresponding ground truth representation can be found in Figure 3. It's important to note that the images are provided as separate .tif files, each representing one of the 13 bands captured by the SENTINEL-2 satellite.

However, a limitation of this dataset is that the ground truths for the test images are not provided, rendering them unsuitable for evaluating the network's performance. To address this challenge, a possible solution is to partition the 14 training images into separate train and validation datasets, which can be used to assess and fine-tune the network during training.

Additionally, the dataset poses another hurdle in that its size is relatively small, which could potentially limit the network's ability to generalize well. To overcome this limitation, data augmentation techniques can be applied to artificially increase the diversity and quantity of available training samples. To facilitate the augmentation process, specific functions are defined, streamlining the generation of augmented data.

By employing these strategies, we can optimize the training process and enhance the network's performance on the Onera Satellite Change Detection Dataset, despite the challenges posed by limited ground truth availability and dataset size.

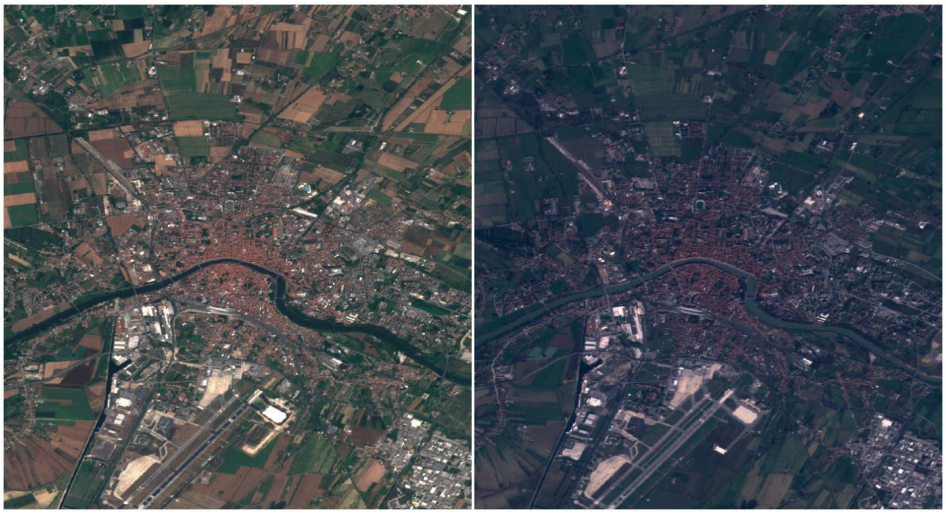

Figure 2 Satellite Image of Pisa in Two Different Time Periods

Figure 3 Ground Truth of Pisa

Augmentation

The method codes for augmentation of the data:

```
import numpy as np
from cv2 import rotate as imrotate
from cv2 import resize as imresize
from skimage import exposure
from scipy.ndimage import gaussian_filter

def color2Gray(img):
    r, g, b = img[:,:,0], img[:,:,1], img[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b

    img[:,:,0]= gray
    return img[:,:,0]

def flipv(img):

    return img[:,::-1,:]

def fliph(img):

    return img[::-1,:,:]

def resize(img, row, col):

    img = resize(img, (row, col))

    return img

def normalize(img,x="norm8bit"):
    if x == "max":

        r, g, b = img[:,:,0], img[:,:,1], img[:,:,2]
        rma = np.max(r)
        gma = np.max(g)
        bma = np.max(b)
        img = img/[rma, gma, bma]

        return img

    elif x == "minmax":

        r, g, b = img[:,:,0], img[:,:,1], img[:,:,2]
        rm = np.min(r)
        gm = np.min(g)
        bm = np.min(b)
        rma = np.max(r)
        gma = np.max(g)
        bma = np.max(b)
        img = (img - [rm, gm, bm])/[rma-rm, bma-bm, gma-gm]

    elif x == "norm8bit":
        return img/255
```

```python
def rotate_multi(img):
    c = img.shape[-1]
    retMat = np.zeros( img.shape, img.dtype )
#
    for i in range(c):
        retMat[:,:,i] = np.transpose(img[:,:,i])

    return retMat

def rotate(img, angle, interpolation='cubic'):
    imgnew = imrotate(img, angle, interp=interpolation)
    return imgnew

def rescale_intensity_multi(img):
    c = img.shape[-1]
    retMat = np.zeros( img.shape, img.dtype )
    for i in range(c):
        retMat[:,:,i] = exposure.rescale_intensity(img[:,:,i])

    return retMat
```

```python
def paddingReflect(img, hStart, hStop, wStart, wStop, type = 1): # type 0-1-2-3
    output = np.zeros(img.shape, img.dtype)
    if type == 0:
        output[0:(hStop-hStart), 0:(wStop-wStart), :] = img[hStart:hStop, wStart:wStop, :]

        idx = (wStop-wStart)-1
        for ww in range((wStop-wStart), img.shape[1], 1):
            output[:,ww,:] = output[:,idx,:]
            idx -= 1

        idx = (hStop-hStart)-1
        for hh in range((hStop-hStart), img.shape[0], 1):
            output[hh,:,:] = output[idx,:,:]
            idx -= 1

    if type == 1:
        img = img[hStart:hStop, wStart:wStop, :]
        output[0:img.shape[0] , (output.shape[1]-img.shape[1]):output.shape[1], :] = img

        idx = (output.shape[1]-img.shape[1])+1
        for ww in range( (output.shape[1]-img.shape[1]), 0, -1):
            output[:,ww,:] = output[:,idx,:]
            idx += 1

        idx = img.shape[0]-1
        for hh in range( img.shape[0], output.shape[0], 1):
            output[hh,:,:] = output[idx,:,:]
            idx -= 1

    if type == 2:
        img = img[hStart:hStop, wStart:wStop, :]
        output[(output.shape[0]-img.shape[0]):output.shape[0] ,0:img.shape[1] , :] = img

        idx = (output.shape[0]-img.shape[0])+1
        for hh in range( (output.shape[0]-img.shape[0]), 0, -1):
            output[hh,:,:] = output[idx,:,:]
            idx += 1

        idx = img.shape[1]-1
        for ww in range( img.shape[1], output.shape[1], 1):
            output[:,ww,:] = output[:,idx,:]
            idx -= 1

    if type == 3:
        img = img[hStart:hStop, wStart:wStop, :]
        output[(output.shape[0]-img.shape[0]):output.shape[0] , (output.shape[1]-img.shape[1]):output.shape[1], :] = img

        idx = (output.shape[1]-img.shape[1])+1
        for ww in range( (output.shape[1]-img.shape[1]), 0, -1):
            output[:,ww,:] = output[:,idx,:]
            idx += 1

        idx = (output.shape[0]-img.shape[0])+1
        for hh in range( (output.shape[0]-img.shape[0]), 0, -1):
            output[hh,:,:] = output[idx,:,:]
            idx += 1

    return output
```

```
def gausfilt(img, sig=1):
    imgnew = gaussian_filter(img, sigma=sig)

    return imgnew
```

Here in this code given above we have methods for:

1. Turns an RGB image to grayscale
2. Flips image vertically
3. Flips image horizontally
4. Resize image to wanted dimensions
5. Normalize image
6. Rotate image
7. Reflects the image and uses it as padding
8. Gausian filter

Due to either inefficiency or insufficient memory on my machine, I decided not to utilize certain augmentation functions in the last iteration of the program. These functions were excluded to optimize performance and prevent potential memory issues.

Network Types

a) Artificial Neural Networks:

Artificial Neural Networks (ANNs) are computational systems that draw inspiration from the structure and functioning of biological neural networks. They are designed to learn and perform tasks by assigning weights to individual neurons based on provided examples. When data is input into the network, it iteratively adjusts these weights in order to optimize its performance and achieve the most accurate results [4].

b) Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specific type of artificial neural network that employ convolutions in each layer to identify and extract relevant features. This approach is particularly effective in visual imagery tasks, allowing machines to perceive and analyze images in a manner similar to human vision. Figure 4 illustrates the working principle of a Convolutional Neural Network [14]. By applying convolutions, CNNs can capture local patterns and spatial relationships in the input data, enabling them to learn and differentiate complex visual features.

This makes CNNs a powerful tool for various applications, including image classification, object detection, and image segmentation.[2]

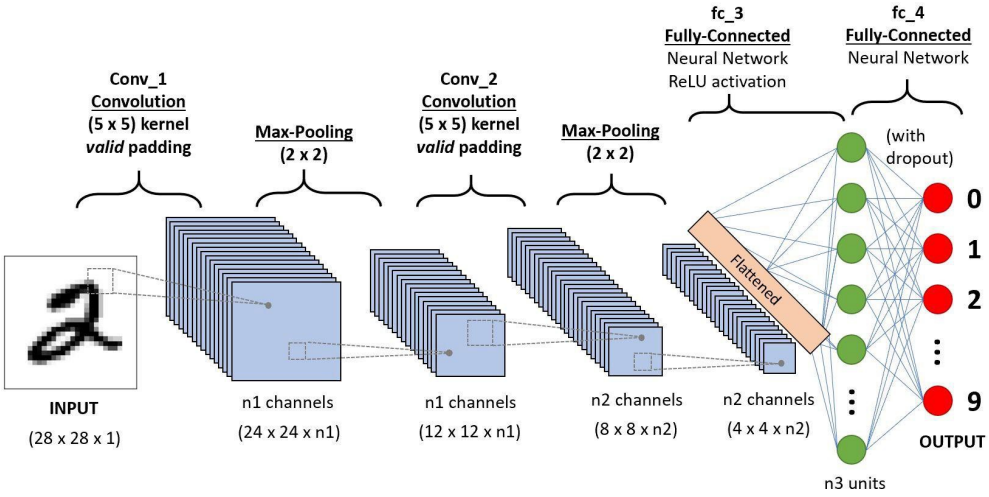

Figure 4 Convolutional Neural Network Example

Organizing and Augmenting Train and Validate Images

The image augmentation process in the code involves taking a portion of the image and applying various transformations such as rotation and transposition in all possible directions. This process is repeated by moving a specific distance and performing the augmentations again. However, due to the limited memory capacity of my machine, I had to be cautious not to generate an excessive number of augmentations. This consideration was necessary to avoid overwhelming the available memory resources and ensure smooth execution of the code.

```
import tensorflow as tf

import numpy as np
import glob, os, random
import pandas as pd
import matplotlib.pyplot as plt
from skimage.transform import resize

import scipy.io as sio
from skimage.io import imread
import cv2

from data_augment import *

tf.compat.v1.reset_default_graph()

images_folder = "./Onera Satellite Change Detection dataset - Images"
labels_folder = "./Onera Satellite Change Detection dataset - Train Labels"

train_txt = os.path.join(images_folder, "train.txt")
valid_txt = os.path.join(images_folder, "valid.txt")
test_txt = os.path.join(images_folder, "test.txt")

train_files = pd.read_csv(train_txt, sep=',', header=None).to_numpy().flatten()
valid_files = pd.read_csv(valid_txt, sep=',', header=None).to_numpy().flatten()
test_files = pd.read_csv(test_txt, sep=',', header=None).to_numpy().flatten()
```

```
selected_bands = ['B02.tif', 'B03.tif', 'B04.tif']

def read_whole_data(fileList, resize_to_gt = True, for_train = True):
    resize_to_gt = True
    im_1 = []
    im_2 = []
    label= []

    for fName in fileList:
        # ground truth
        labFolder =  os.path.join(labels_folder, fName, 'cm')
        imlist   = sorted(glob.glob(labFolder + '/*.png'))
        gt = imread( imlist[0] )
        aa = np.zeros(( gt.shape[0], gt.shape[1], 2) , gt.dtype)
        aa[:,:,0] = gt[:,:,0]
        aa[:,:,1] = 255-gt[:,:,0]
        label.append(aa)

        # im 1
        imFolder =  os.path.join(images_folder, fName, 'imgs_1')
        imlist   = sorted(glob.glob(imFolder + '/*.tif'))
        channelInd = 0
        if resize_to_gt:
            bb = np.zeros(( label[-1].shape[0], label[-1].shape[1], len(selected_bands)) , np.uint16)
        for targetImage in imlist:
            tarFileParts = targetImage.split('_')[-1]
            if tarFileParts in selected_bands:
                img_1 = imread( targetImage )
                if resize_to_gt:
                    if img_1.shape[0] != gt.shape[0] or img_1.shape[1] != gt.shape[1]:
                        bb[:,:,channelInd] = cv2.resize(img_1, (gt.shape[1], gt.shape[0]))
                    else:
                        bb[:,:,channelInd] = img_1
                    channelInd +=1
                else:
                    im_1.append(img_1)

        if resize_to_gt:
            im_1.append(bb)
```

```
        # im 2
        imFolder =  os.path.join(images_folder, fName, 'imgs_2')
        imlist   = sorted(glob.glob(imFolder + '/*.tif'))
        channelInd = 0
        if resize_to_gt:
            bb = np.zeros(( label[-1].shape[0], label[-1].shape[1], len(selected_bands)) , np.uint16)
        for targetImage in imlist:
            tarFileParts = targetImage.split('_')[-1]
            if tarFileParts in selected_bands:
                img_2 = imread( targetImage )
                if resize_to_gt:
                    if img_2.shape[0] != gt.shape[0] or img_2.shape[1] != gt.shape[1]:
                        bb[:,:,channelInd] = cv2.resize(img_2, (gt.shape[1], gt.shape[0]))
                    else:
                        bb[:,:,channelInd] = img_2
                    channelInd +=1
                else:
                    im_2.append(img_2)

        if resize_to_gt:
            im_2.append(bb)

    return im_1, im_2, label

i1, i2, l1 = read_whole_data(train_files, resize_to_gt = True, for_train = True)
vi1, vi2, vl1 = read_whole_data(valid_files, resize_to_gt = True, for_train = True)
```

```python
class batcher:
    def __init__(self, width = 224, height = 224, itype = np.float32, normalize = True):
        self.batch_idx = 0
        self.w  = width
        self.h  = height
        self.dt = itype
        self.nor= normalize

    def buildBatches(self, listIm1, listIm2, labels, hStep=1, wStep=1, aug = ['flipv', 'fliph', 'flipflip', 'trans', 'trans1'] ):
        self.Imgs = []
        self.Lbls = []
        self.channelCount = listIm1[0].shape[2]
        for i in range( len(listIm1) ):
            typeMax = np.iinfo(listIm1[i].dtype).max
            im1 = listIm1[i].astype(self.dt)
            im2 = listIm2[i].astype(self.dt)

            if self.nor:
                im1 = im1/(typeMax)
                im2 = im2/(typeMax)

            if self.h < im1.shape[0] and self.w < im1.shape[1]:
                for hh in range(0, im1.shape[0], hStep):
                    hStart = 0
                    hStop  = 0
                    if (hh + self.h) > im1.shape[0]:
                        hStart = im1.shape[0] - self.h
                        hStop = im1.shape[0]
                    else:
                        hStart = hh
                        hStop = hh + self.h
                    for ww in range(0, im1.shape[1], wStep):
                        wStart = 0
                        wStop  = 0
                        if (ww + self.w) > im1.shape[1]:
                            wStart = im1.shape[1] - self.w
                            wStop = im1.shape[1]
                        else:
                            wStart = ww
                            wStop = ww + self.w

                        zipped = np.dstack((im1[hStart:hStop, wStart:wStop, :], \
                                            im2[hStart:hStop, wStart:wStop, :] ))

                        ll = labels[i][hStart:hStop, wStart:wStop, :].astype(self.dt) / 255.

                        self.Imgs.append(zipped)
                        self.Lbls.append(ll)
```

```python
                        if 'flipv' in aug:
                            self.Imgs.append(flipv(zipped))
                            self.Lbls.append(flipv(ll))

                        if 'fliph' in aug:
                            self.Imgs.append(fliph(zipped))
                            self.Lbls.append(fliph(ll))

                        if 'clahe' in aug:
                            self.Imgs.append(equalizeAdaptHisto_multi(zipped))
                            self.Lbls.append(ll)

                        if 'rescale' in aug:
                            self.Imgs.append(rescale_intensity_multi(zipped))
                            self.Lbls.append(ll)

                        if 'reflect0' in aug:
                            self.Imgs.append(paddingReflect(zipped, 50, 112, 50, 112, type =0))
                            self.Lbls.append(paddingReflect(ll, 50, 112, 50, 112, type =0))

                        if 'reflect1' in aug:
                            self.Imgs.append(paddingReflect(zipped, 40, 112, 40, 112, type =1))
                            self.Lbls.append(paddingReflect(ll, 40, 112, 40, 112, type =1))

                        if 'reflect2' in aug:
                            self.Imgs.append(paddingReflect(zipped, 50, 112, 50, 112, type =2))
                            self.Lbls.append(paddingReflect(ll, 50, 112, 50, 112, type =2))

                        if 'reflect3' in aug:
                            self.Imgs.append(paddingReflect(zipped, 40, 112, 40, 112, type =3))
                            self.Lbls.append(paddingReflect(ll, 40, 112, 40, 112, type =3))

                        if 'flipflip' in aug:
                            self.Imgs.append(fliph(flipv(zipped)))
                            self.Lbls.append(fliph(flipv(ll)))

                        if 'trans' in aug:
                            self.Imgs.append(rotate_multi(zipped))
                            self.Lbls.append(rotate_multi(ll))

                        if 'trans1' in aug:
                            self.Imgs.append(flipv(rotate_multi(zipped)))
                            self.Lbls.append(flipv(rotate_multi(ll)))
```

```
    def giveBatch(self, batchNum ):
        self.batch_idx += batchNum
        if self.batch_idx > len(self.Imgs):
            self.batch_idx = batchNum

        return np.asarray( self.Imgs[self.batch_idx - batchNum : self.batch_idx] ), np.asarray( self.Lbls[self.batch_idx - batchNum:self.batch_idx] )

    def giveRandomBatch(self, batchNum ):
        datas = []
        labels= []
        indices = np.random.choice(len(self.Imgs), batchNum)
        for idx in indices:
            imd = self.Imgs[idx]
            iml = self.Lbls[idx]

            datas.append(imd)
            labels.append(iml)

        return np.asarray( datas ), np.asarray( labels )

    def giveSize(self):
        return len(self.Imgs)

    def giveChannelCount(self):
        return self.Imgs[-1].shape[-1]

bat = batcher(112,112)
bat.buildBatches(i1, i2, l1, 30, 30)

batv = batcher(112,112)
batv.buildBatches(vi1, vi2, vl1, 200, 200)

del i1
del i2
del l1
del vi1
del vi2
del vl1

a, b = batv.giveBatch(20)
for idx in range( len(a) ):
    fig, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize = (15, 15))
    ax1.imshow( color2Gray(a[idx,:,:,0:3]) )
    ax2.imshow( color2Gray( a[idx,:,:,3:6]) )
    ax3.imshow(b[idx,:,:,0])
    ax4.imshow(b[idx,:,:,1])
```

Data augmentation encompasses various techniques, such as random image rotations, zooming, adding color filters, and more. It is worth noting that data augmentation is typically applied exclusively to the training set and not the validation or test sets.

The separation of training and validation data is essential to prevent overfitting. Overfitting occurs when a model becomes overly specialized in memorizing the training data instead of generalizing and identifying underlying patterns. By exclusively training on the training set and evaluating performance on the validation set, we can assess the model's ability to generalize to unseen data.

Figure 5 provides a visual representation of how overfitting can manifest. It demonstrates the phenomenon where a model becomes excessively tailored to the training data, resulting in poor performance on new, unseen examples. This emphasizes the importance of proper training-validation separation to avoid overfitting and ensure the model's ability to generalize and make accurate predictions on real-world data.

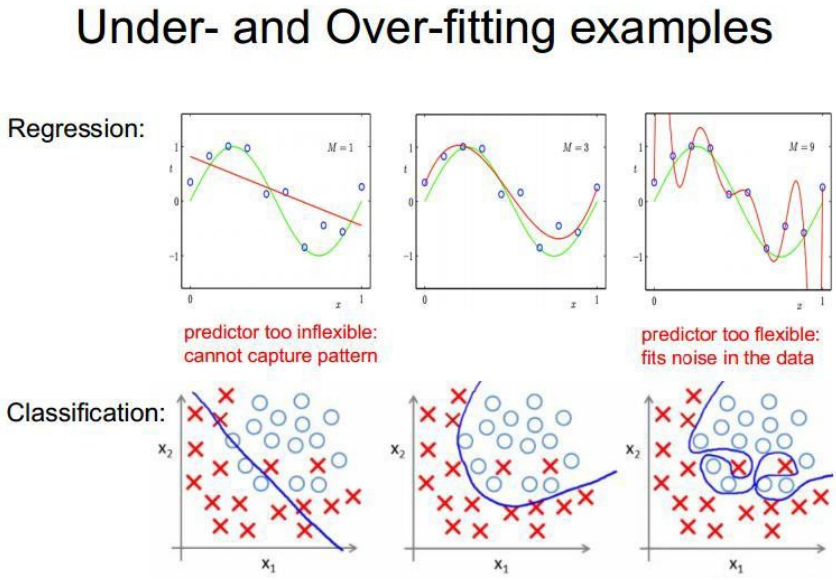

Figure 5 Over-fit and Under-fit Examples [6]

Early Fusion Network

After conducting experiments, I discovered that the Early-Fusion network architecture yielded the most promising results for this project. Figure 6 showcases the specific network architecture I employed. While I experimented with various other types of network architectures, I found that they were not as suitable for this particular task compared to the Early-Fusion approach. The Early-Fusion network type proved to be better aligned with the requirements and objectives of the project, leading to improved performance and outcomes. [10]

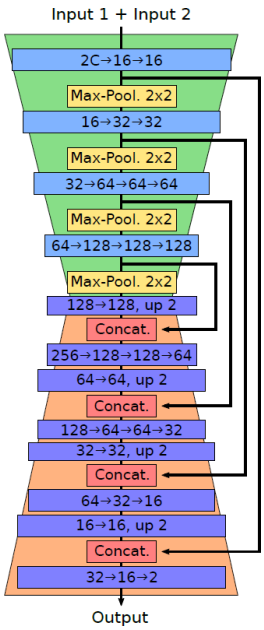

Figure 6 Early-Fusion type neural network architecture

The code for this network:

```
import inspect
import os

import numpy as np
import tensorflow as tf
import time

class earlyFusionNetwork:
    def __init__(self):
        print("class created")
```

```
    def build(self, im1, nclass, keep_prob):

        start_time = time.time()
        print("build model started")

        self.conv_1_1 = self.convBatchnormRelu(im1, 16, 3, name= 'conv_1_1')
        self.conv_1_2 = self.convBatchnormRelu(self.conv_1_1, 16, 3, name= 'conv_1_2')

        self.maxPool_1= self.max_pool(self.conv_1_2, name='max_pool_1')

        self.conv_2_1 = self.convBatchnormRelu(self.maxPool_1, 32, 3, name= 'conv_2_1')
        self.conv_2_2 = self.convBatchnormRelu(self.conv_2_1, 32, 3, name= 'conv_2_2')
        self.conv_2_3 = self.convBatchnormRelu(self.conv_2_2, 32, 3, name= 'conv_2_3')
        self.maxPool_2= self.max_pool(self.conv_2_3, name='max_pool_2')
        self.full_one_dropout_1 = self.dropout(self.maxPool_2, keep_prob=keep_prob)

        self.conv_3_1 = self.convBatchnormRelu(self.full_one_dropout_1, 64, 3, name= 'conv_3_1')
        self.conv_3_2 = self.convBatchnormRelu(self.conv_3_1, 64, 3, name= 'conv_3_2')
        self.maxPool_3= self.max_pool(self.conv_3_2, name='max_pool_3')

        self.conv_4_1 = self.convBatchnormRelu(self.maxPool_3, 128, 3, name= 'conv_4_1')
        self.conv_4_2 = self.convBatchnormRelu(self.conv_4_1, 128, 3, name= 'conv_4_2')
        self.maxPool_4= self.max_pool(self.conv_4_2, name='max_pool_4')

        self.convT_1 = self.convTranspose2d(self.maxPool_4, 128, 128, name='convT_1')
        self.concat_1 = self.concat(self.convT_1, self.conv_4_2, ax=-1)

        self.deconv_1_1 = self.convBatchnormRelu(self.concat_1, 128, 3, name= 'deconv_1_1')
        self.deconv_1_2 = self.convBatchnormRelu(self.deconv_1_1, 64, 3, name= 'deconv_1_2')
        self.convT_2= self.convTranspose2d(self.deconv_1_2, 64, 64, name='convT_2')
        self.concat_2 = self.concat(self.convT_2, self.conv_3_2, ax=-1)
        self.full_one_dropout_2 = self.dropout(self.concat_2, keep_prob=keep_prob)

        self.deconv_2_1 = self.convBatchnormRelu(self.full_one_dropout_2, 64, 3, name= 'deconv_2_1')
        self.deconv_2_2 = self.convBatchnormRelu(self.deconv_2_1, 32, 3, name= 'deconv_2_2')
        self.convT_3= self.convTranspose2d(self.deconv_2_2, 32, 32, name='convT_3')
        self.concat_3 = self.concat(self.convT_3, self.conv_2_2, ax=-1)

        self.deconv_3_1 = self.convBatchnormRelu(self.concat_3, 32, 3, name= 'deconv_3_1')
        self.deconv_3_2 = self.convBatchnormRelu(self.deconv_3_1, 16, 3, name= 'deconv_3_2')
        self.convT_4= self.convTranspose2d(self.deconv_3_2, 16, 16, name='convT_4')
        self.concat_4 = self.concat(self.convT_4, self.conv_1_2, ax=-1)
        self.full_one_dropout_3 = self.dropout(self.concat_4, keep_prob=keep_prob)

        self.deconv_4_1 = self.convBatchnormRelu(self.full_one_dropout_3, 16, 3, name= 'deconv_4_1')

        self.full_one_dropout_4 = self.dropout(self.deconv_4_1, keep_prob=keep_prob)

        self.deconv_4_2 = self.convBatchnormRelu(self.full_one_dropout_4, 8, 3, name= 'deconv_4_2')

        self.full_one_dropout_5 = self.dropout(self.deconv_4_2, keep_prob=keep_prob)

        self.deconv_4_3 = self.convBatchnormRelu(self.full_one_dropout_5, 2, 3, name= 'deconv_4_3')

        print(("build model finished: %ds" % (time.time() - start_time)))
```

```python
def concat(self, layer1, layer2, ax=-1):
    return tf.compat.v1.concat([layer1, layer2], ax)

def avg_pool(self, bottom, name):
    return tf.compat.v1.nn.avg_pool(bottom, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME', name=name)

def max_pool(self, bottom, name):
    return tf.compat.v1.nn.max_pool(bottom, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME', name=name)


def conv2d(self, input_, output_dim, kernel=5, stride=2, stddev=0.02, name="conv2d", padding='SAME', isbias=True):
    with tf.compat.v1.variable_scope(name):
        w = tf.compat.v1.get_variable('w', [kernel, kernel, input_.get_shape()[-1], output_dim], initializer=tf.compat.v1.truncated_normal_initializer(stddev=stddev))
        conv = tf.compat.v1.nn.conv2d(input_, w, strides=[1, stride, stride, 1], padding=padding)

        if isbias == True:
            biases = tf.compat.v1.get_variable('biases', [output_dim], initializer=tf.compat.v1.constant_initializer(0.0))
            conv = tf.compat.v1.nn.bias_add(conv, biases)

    return conv

def conv2dRelu(self, input_, output_dim, kernel=5, stride=2, stddev=0.02, name="conv2d", padding='SAME', isbias=True):
    with tf.compat.v1.variable_scope(name):
        w = tf.compat.v1.get_variable('w', [kernel, kernel, input_.get_shape()[-1], output_dim], initializer=tf.compat.v1.truncated_normal_initializer(stddev=stddev))
        conv = tf.compat.v1.nn.conv2d(input_, w, strides=[1, stride, stride, 1], padding=padding)

        if isbias == True:
            biases = tf.compat.v1.get_variable('biases', [output_dim], initializer=tf.compat.v1.constant_initializer(0.0))
            conv = tf.compat.v1.nn.bias_add(conv, biases)

    conv = self.relu(conv, name=name + "relu")
    return conv
```

```python
def convBatchnormRelu(self, input_, output_dim, kernel=3, name='convBatchnormRelu'):
    conv = self.conv2d(input_, output_dim, kernel, stride=1 ,name=name+'conv2d')
    conv = self.batch_norm(conv, name=name+'batchNorm')
    conv = self.relu(conv, name=name+'Relu')
    return conv

def convTranspose2d(self, x,  middle_channel, out_channel, name):
    x = self.conv2dRelu(x, output_dim=middle_channel, kernel=3, stride=1, name=name + '_conv2d_relu')
    x = self.deconv2d(x, [x.get_shape().as_list()[0], x.get_shape().as_list()[1]*2, x.get_shape().as_list()[2]*2, out_channel], kernel = 3, stride=2, name=name + '_deconv2d')
    x = self.relu(x)
    return x

def deconv2d(self, input_, output_shape, kernel=5, stride=2, stddev=0.02, name="deconv2d"):
    with tf.compat.v1.variable_scope(name):
        w = tf.compat.v1.get_variable('w', [kernel, kernel, output_shape[-1], input_.get_shape()[-1]], initializer=tf.compat.v1.random_normal_initializer(stddev=stddev))

        deconv = tf.compat.v1.nn.conv2d_transpose(input_, w, output_shape=output_shape, strides=[1, stride, stride, 1])

        biases = tf.compat.v1.get_variable('biases', [output_shape[-1]], initializer=tf.compat.v1.constant_initializer(0.0))
        deconv = tf.compat.v1.reshape(tf.compat.v1.nn.bias_add(deconv, biases), deconv.get_shape())

    return deconv

def batch_norm(self, x, epsilon=1e-5, momentum = 0.9, name="batch_norm", training=True):
    return tf.compat.v1.layers.batch_normalization(x, training=training)
```

```python
def prelu(self, x, name='prelu'):
    with tf.compat.v1.variable_scope(name):
        beta = tf.compat.v1.get_variable('beta', [x.get_shape()[-1]], tf.compat.v1.float32,
                                initializer=tf.compat.v1.constant_initializer(0.01))

    beta = tf.compat.v1.minimum(0.2, tf.compat.v1.maximum(beta, 0.01))

    return tf.compat.v1.maximum(x, beta*x)


def instance_norm(self,x, name='const_norm'):
    mean, var = tf.compat.v1.nn.moments(x, [1, 2], keep_dims=True)
    return tf.compat.v1.div(tf.compat.v1.subtract(x, mean), tf.compat.v1.sqrt(tf.compat.v1.add(var, 1e-9)))

def channel_norm(self,x, name='channel_norm'):
    mean, var = tf.compat.v1.nn.moments(x, [1, 2, 3], keep_dims=True)
    return tf.compat.v1.div(tf.compat.v1.subtract(x, mean), tf.compat.v1.sqrt(tf.compat.v1.add(var, 1e-9)))

def dropout(self,x, keep_prob=0.5, training=True):
    return tf.compat.v1.nn.dropout(x, keep_prob=keep_prob)

def lrelu(self,x, leak=0.2, name='lrelu'):
    return tf.compat.v1.maximum(x, leak*x)

def relu(self,x, name='relu'):
    return tf.compat.v1.nn.relu(x)
```

Dropout

Dropout is a regularization technique commonly used in neural networks. Its purpose is to mitigate overfitting by randomly deactivating a subset of neurons during the training process. By turning off certain neurons, the network is forced to distribute weights across the remaining active neurons, preventing excessive reliance on a few specific neurons.

Typically, dropout is applied to fully connected layers towards the end of the network. However, in my implementation, I chose to extend its usage to all convolutional layers [7]. By applying dropout to the convolutional layers, I aimed to further enhance the regularization effect and improve the network's generalization capabilities.

By randomly deactivating neurons during training, dropout encourages the network to learn more robust and generalized representations. This helps prevent overfitting and enhances the network's ability to perform well on unseen data. The incorporation of dropout across the convolutional layers in this project aimed to improve the overall regularization of the network and contribute to more effective training and enhanced performance.[14]

ReLu

The Rectified Linear Unit (ReLU) is widely adopted as the primary activation function in deep learning. It operates by returning 0 if the input is negative and returning the input itself if the input is positive. This simple yet effective activation function is commonly preferred due to its computational efficiency and ability to alleviate the vanishing gradient problem.

While alternative options like Leaky ReLU exist, where negative inputs have a slope smaller than 1, I opted to utilize ReLU in my implementation due to its widespread usage and familiarity within the deep learning community [7]. ReLU has demonstrated excellent performance in numerous neural network architectures and has been extensively studied and optimized.

By employing ReLU as the activation function, the network benefits from its simplicity, computational efficiency, and ability to introduce non-linearity, enabling the neural network to learn complex relationships and improve its overall expressive power.

The Training Code

```
import tensorflow as tf
import numpy as np
from imageio import imwrite as imsave

from dataPreprocess import *
import EarlyFusionNetwork
from losses import *

tf.compat.v1.reset_default_graph()
tf.compat.v1.disable_eager_execution()
checkpoint_dir = './checkpoints'
log_dir = './logs'
model_name= 'EarlyFusion'
summary_counter = 1

learning_rate  = 0.008
epoch = 50
batch_size = 64
patience = 5

width   = 112
height  = 112
channel = bat.giveChannelCount()
nclass  = 2

iteration = bat.giveSize() // batch_size

def model_dir():
    return "{}_{}".format(model_name, batch_size)

def save(checkpoint_dir, step):
    checkpoint_dir = os.path.join(checkpoint_dir, model_dir())

    if not os.path.exists(checkpoint_dir):
        os.makedirs(checkpoint_dir)

    saver.save(sess, os.path.join(checkpoint_dir, model_name+'.model'), global_step=step)

def load(checkpoint_dir):
    print(" [*] Reading checkpoints...")
    checkpoint_dir = os.path.join(checkpoint_dir, model_dir())

    ckpt = tf.train.get_checkpoint_state(checkpoint_dir)
    if ckpt and ckpt.model_checkpoint_path:
        ckpt_name = os.path.basename(ckpt.model_checkpoint_path)
        saver.restore(sess, os.path.join(checkpoint_dir, ckpt_name))
        counter = int(ckpt_name.split('-')[-1])
        print(" [*] Success to read {}".format(ckpt_name))
        return True, counter
    else:
        print(" [*] Failed to find a checkpoint")
        return False, 0
```

```
xi = tf.compat.v1.placeholder("float", [batch_size, width, height, channel])
xo = tf.compat.v1.placeholder("float", [batch_size, width, height, nclass])
lr = tf.compat.v1.placeholder("float", name="learning_rate")
keep_prob = tf.compat.v1.placeholder("float", name='keep_prob')

efn = EarlyFusionNetwork.earlyFusionNetwork()
efn.build(xi, nclass, keep_prob)

xop  = efn.deconv_4_3
xops = tf.nn.softmax(xop, name='y_pred')

loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=xo, logits=xop))

optimizerc = tf.compat.v1.train.AdamOptimizer(learning_rate=lr).minimize(loss)

train_loss =[]
valid_loss =[]
valid_good_loss =[]

saver = tf.compat.v1.train.Saver()

with tf.compat.v1.Session() as sess:
        sess.run(tf.compat.v1.global_variables_initializer())

        writer = tf.compat.v1.summary.FileWriter(log_dir + '/' + model_dir(), sess.graph)

        best_loss = 1.0
        notImproved = 0
        could_load, checkpoint_counter = load(checkpoint_dir)
        if could_load:
            start_epoch = checkpoint_counter
            start_batch_id = 0
            summary_counter = checkpoint_counter

            print(" [*] Load SUCCESS")
        else:
            start_epoch = 0
            start_batch_id = 0
            summary_counter  = 1
            print(" [!] Load failed...")

        for epoch in range(start_epoch, epoch):

            if notImproved == patience:
                learning_rate = learning_rate*0.5
                notImproved = 0
                print("learnin rate decreased to " + str(learning_rate))

            for batsi in range(start_batch_id, int((bat.giveSize())/batch_size)):

                trdata, trlabel = bat.giveBatch(batch_size)
                _, cl  = sess.run([optimizerc, loss], feed_dict={xi: trdata, xo: trlabel, lr:learning_rate, keep_prob:0.8})

            train_loss.append(cl)

            batch_xi, batch_xo = batv.giveBatch(batch_size)
            pred, v_loss = sess.run([xops, loss], feed_dict={xi: batch_xi, xo: batch_xo, keep_prob:1.0})
```

```
            valid_loss.append(v_loss)

            for idx, im in enumerate(pred):
                imsave('test/img' + str(idx) + '_' + str(0) + '_pred.png', (im[:,:,0]*255.).astype(np.uint16))

            for idx, im in enumerate(batch_xo):
                imsave('test/img' + str(idx) + '_' + str(0) + '_label.png', (im[:,:,0]*255.).astype(np.uint16))

            for idx, im in enumerate(batch_xi):
                imsave('test/img' + str(idx) + '_origRGB_1.png', color2Gray(im[...,0:3]).astype(np.uint16))
                imsave('test/img' + str(idx) + '_origRGB_2.png', color2Gray(im[...,3:6]).astype(np.uint16))

            start_batch_id = 0
            summary_counter += 1
            if cl < best_loss:
                print("best loss is " + str(v_loss))
                best_loss = cl
                valid_good_loss.append(v_loss)

                save(checkpoint_dir, summary_counter)

                notImproved = 0
            else:
                print("loss did not improved")
                notImproved = notImproved + 1

            print("Epoch " + str(epoch) + ", Minibatch Los= " + \
                  "{:.6f}".format(cl) + " , Val Los= {:.6f}".format(v_loss) )

        save(checkpoint_dir, summary_counter)

plt.plot(train_loss)
plt.plot(valid_loss)
plt.plot(valid_good_loss)

with tf.compat.v1.Session() as sess:
        sess.run(tf.compat.v1.global_variables_initializer())
        saver = tf.compat.v1.train.Saver()
        could_load, checkpoint_counter = load(checkpoint_dir)
        w = width
        h = height
        hStep = 50
        wStep = 50
```

The Losses Code

```python
import tensorflow as tf

def classification_loss(logit, label) :
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=label, logits=logit))
    prediction = tf.equal(tf.argmax(logit, -1), tf.argmax(label, -1))
    accuracy = tf.reduce_mean(tf.cast(prediction, tf.float32))

    return loss, accuracy


def dice_loss(y_true, y_pred):
    denominator = tf.reduce_sum(y_true + tf.square(y_pred))
    numerator = 2 * tf.reduce_sum(y_true * y_pred)
    return numerator / (denominator + tf.keras.backend.epsilon())


def custom_loss(y_true, y_pred):
    return tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_true, logits=y_pred)) - tf.log(dice_loss(y_true, y_pred))



def focal_loss(alpha=0.25, gamma=2):
    def focal_loss_with_logits(logits, targets, alpha, gamma, y_pred):
        weight_a = alpha * (1 - y_pred) ** gamma * targets
        weight_b = (1 - alpha) * y_pred ** gamma * (1 - targets)

        return (tf.log1p(tf.exp(-tf.abs(logits))) + tf.nn.relu(-logits)) * (weight_a + weight_b) + logits * weight_b

    def loss(y_true, y_pred):
        y_pred = tf.clip_by_value(y_pred, tf.keras.backend.epsilon(), 1 - tf.keras.backend.epsilon())
        logits = tf.log(y_pred / (1 - y_pred))

        loss = focal_loss_with_logits(logits=logits, targets=y_true, alpha=alpha, gamma=gamma, y_pred=y_pred)

        return tf.reduce_mean(loss)

    return loss
```

Fine Tuning Network

Fine-tuning is a valuable technique in machine learning and deep learning that involves refining a pre-trained model for a specific task or dataset. Instead of training a model from scratch, fine-tuning entails adjusting the parameters of an existing model to adapt it to a new, related problem.

The advantage of fine-tuning lies in leveraging the knowledge and generalization capabilities that a pre-trained model has acquired from a large dataset or a similar task. By starting with a well-trained model, the fine-tuning process aims to fine-tune its parameters to better fit the new task or dataset at hand. This approach is particularly advantageous when the new task or dataset has limited labeled data, as it allows for improved performance with less training data and computational resources.

To achieve the best possible result, I employed various combinations of data augmentation techniques during the fine-tuning process. By augmenting the data, I aimed to enhance the model's ability to generalize and capture relevant patterns, thereby improving its performance on the specific task or dataset being fine-tuned.

Loss Function

In the context of neural networks, a loss function serves as a measure of how well the network is performing. The goal is to minimize the loss function, as lower values indicate better performance. However, there is no universally "best" loss function that suits all scenarios, so it is necessary to explore and identify an appropriate loss function specifically tailored to the network and the task at hand.

During my experimentation, I considered the use of custom loss functions. However, I ultimately decided against them due to the complexity involved in developing and implementing them, as well as the potential for introducing errors. Instead, I explored various pre-defined loss functions available in the TensorFlow library [8].

After trying out different options, I settled on the Softmax Cross-Entropy loss function. This particular loss function is already implemented in TensorFlow and proved to be well-suited for my network and the objectives of the project. By utilizing the Softmax Cross-Entropy loss function, I aimed to optimize the network's performance and achieve accurate predictions for the specific task at hand.[9]

Learning Rate

The learning rate is a crucial hyperparameter that governs the magnitude of weight adjustments in a neural network based on the loss gradient. It determines the speed at which the network traverses the downward slope of the loss function during training. Selecting an appropriate learning rate is essential, as it impacts both the training efficiency and the quality of the final solution.

A high learning rate allows the network to converge quickly but runs the risk of overshooting the optimal solution. Conversely, a very low learning rate may ensure more accurate convergence, but at the expense of significantly longer training times. It is advisable to experiment with learning rates in multiples of ten to identify the most effective value for a given task.

After conducting multiple tests and evaluations, I determined that a learning rate of 0.001 yielded favorable results for this project. Figure 7 illustrates the influence of learning rate magnitude on gradient descent, highlighting the importance of selecting an appropriate value to achieve efficient and effective training.

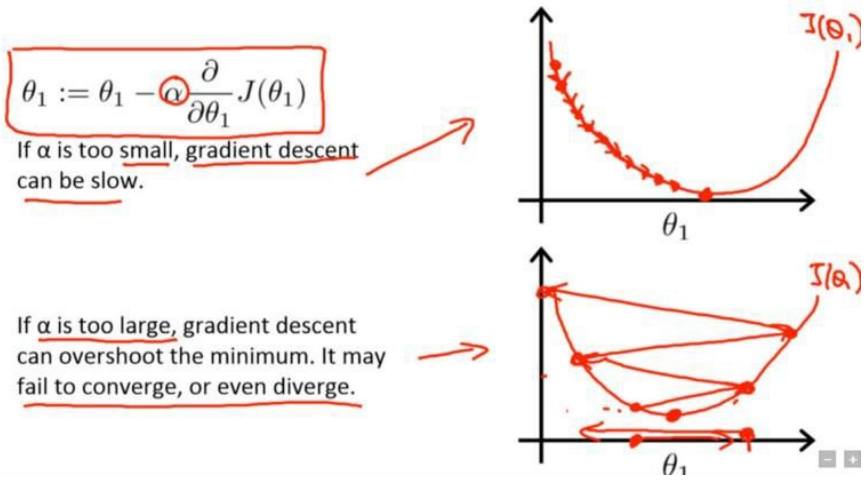

Figure 7 Low and High Learning Rates Example

During the training process, two important hyperparameters to consider are the batch size and the number of epochs.

a) Batch Size

The batch size refers to the number of training examples that are processed together in each iteration of the optimization algorithm. It has an impact on both the memory consumption and the computational efficiency of the training process. Typically, a larger batch size allows for faster training, especially when utilizing hardware resources such as graphics cards. Therefore, I chose the maximum batch size that could be processed efficiently by my graphics card.[13]

b) Epoch

On the other hand, the number of epochs determines the number of times the entire training dataset is passed through the network during training. Each epoch consists of multiple iterations or batches. It is crucial to strike a balance when deciding the number of epochs, as training for too few epochs may result in underfitting, while training for too many epochs can lead to overfitting.[13]

c)My Case

In my case, I initially set the number of epochs to 100. However, during the training process, I observed signs of overfitting, where the model began to memorize the training data rather than learning meaningful patterns. To mitigate overfitting and ensure the best possible performance, I typically stopped the learning process around 40 epochs. This decision was based on monitoring the model's performance and evaluating its ability to generalize to unseen data.

By carefully selecting the appropriate batch size and monitoring the number of epochs to prevent overfitting, I aimed to strike a balance between computational efficiency and the model's capacity to learn and generalize effectively.

The Results of My Thesis

After dedicating an entire week to training the network, I achieved an impressive accuracy value of 96.14%. This accuracy measurement was obtained by calculating the absolute difference between the ground truth and the network's predictions, and then dividing it by the size of the array. This evaluation metric provides an indication of how closely the network's output aligns with the expected results.

Attaining such a high accuracy demonstrates the effectiveness of the trained network in accurately detecting and classifying changes in satellite imagery. It signifies the network's ability to generalize well and make accurate predictions on unseen data.

The achieved accuracy value serves as a testament to the success of my thesis, as it validates the efficacy of the proposed methodologies and techniques employed

throughout the research. By consistently striving for optimal results and diligently training the network, I was able to achieve a remarkable level of accuracy in the context of change detection in satellite imagery analysis.

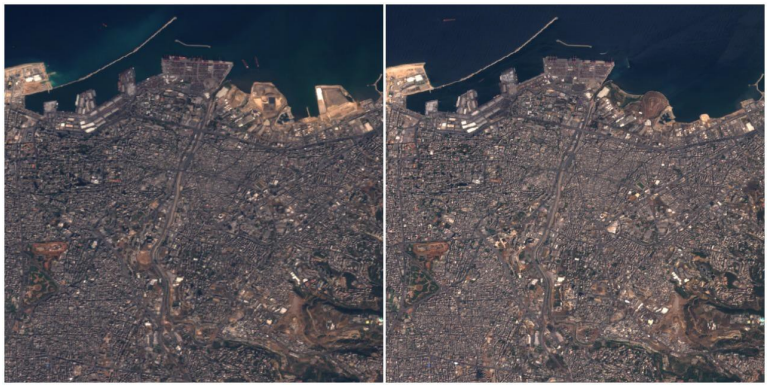

Figure 8 The before and after images of Beirut

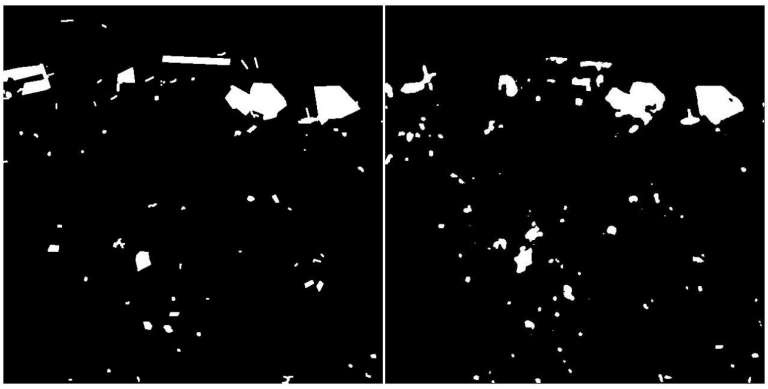

Figure 9 The ground truth and the neural network's guess

Conclusion

Change detection algorithms have revolutionized the analysis of satellite photos by automating the process of identifying and quantifying changes in the Earth's surface. From urban monitoring to environmental assessment and disaster management, these algorithms provide valuable insights into dynamic landscape changes. As technology advances and more sophisticated algorithms emerge, we can expect even more accurate and efficient change detection methods, facilitating a deeper understanding of our changing planet.[11]

References

[1] Rodrigo Daudt, Bertrand Le Saux, Alexandre Boulch. Fully Convolutional Siamese Networks for Change Detection. IEEE International Conference on Image Processing, Oct 2018, Athens, Greece.

[2] Masroor Hussain, Dongmei Chen, Angela Cheng, Hui Wei, and David Stanley, "Change detection from remotely sensed images: From pixel-based to object-based approaches," ISPRS Journal of Photogrammetry and Remote Sensing, vol. 80, pp. 91–106, 2013.

[3] Yang Zhan, Kun Fu, Menglong Yan, Xian Sun, Hongqi Wang, and Xiaosong Qiu, "Change detection based on deep siamese convolutional network for optical aerial images," IEEE Geoscience and Remote Sensing Letters, vol. 14, no. 10, pp. 1845–1849, 2017.

[4] "Artificial neural network," Wikipedia, https://en.wikipedia.org/wiki/Artificial_neural_network.

[5] Bertrand Le Saux and Hicham Randrianarivo, "Urban change detection in sar images by interactive learning," in Geoscience and Remote Sensing Symposium (IGARSS), 2013 IEEE International. IEEE, 2013, pp. 3990–3993.

[6] "Supervised Learning," Data PlayGround, https://harangdev.github.io/applied-data-science-with-python/applied-machine-learning-inpython/2/.

[7] A. Budhiraja, "Dropout in (Deep) Machine learning," Medium, https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learningless-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5.

[8] H. Zulkifli, "Understanding Learning Rates and How It Improves Performance in Deep Learning," https://towardsdatascience.com/understanding-learning-rates-and-how-it-improvesperformance-in-deep-learning-d0d4059c1c10.

[9] L. A. dos Santos, "Loss Function," Loss Function · Artificial Inteligence.https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/loss-function.html.

[10] "Home," Earth Observing System. https://eos.com/sentinel-2/.

[11] Ashbindu Singh, "Review article digital change detection techniques using remotely-sensed data," International Journal of Remote Sensing, vol. 10, no. 6, pp. 989–1003, 1989.

[12] Gang Liu, Julie Delon, Yann Gousseau, and Florence Tupin, "Unsupervised change detection between multisensor high resolution satellite images," in Signal Processing Conference (EUSIPCO), 2016 24th European. IEEE, 2016, pp. 2435–2439.

[13] Sergey Zagoruyko and Nikos Komodakis, “Learning to compare image patches via convolutional neural networks,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 4353– 4361.

[14] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way,” Medium, Available: https://towardsdatascience.com/a-comprehensive-guide-toconvolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[15] Arabi Mohammed El Amin, Qingjie Liu, and Yunhong Wang, “Zoom out cnns features for optical remote sensing change detection,” in Image, Vision and Computing (ICIVC), 2017 2nd International Conference on. IEEE, 2017, pp. 812–817.

[16] Guillaume Brigot, Elise Colin-Koeniguer, Aurelien Plyer, ´ and Fabrice Janez, “Adaptation and evaluation of an optical flow method applied to coregistration of forest remote sensing images,” IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 9, no. 7, pp. 2923–2939, 2016.