EYE IN THE SKY

7th INTER IIT TECH MEET 2018

TECHNICAL REPORT

IIT Guwahati

TEAM MEMBERS

1. Manideep Kolla manideepkolla@iitg.ac.in
2. Aniket Mandle aniketmandle11@iitg.ac.in
3. Apoorva Kumar kr17apoorva@iitg.ac.in

INTRODUCTION

1.1 REMOTE SENSING

Remote sensing is the science of obtaining information about objects or areas from a distance, typically from aircraft or satellites. Applications:

Hazard assessment: Track hurricanes, earthquakes, erosion, and flooding. Data can be used to assess the impacts of a natural disaster and create preparedness strategies to be used before and after a hazardous event.

Natural resource management: Monitor land use, map wetlands, and chart wildlife habitats. Data can be used to minimize the damage that urban growth has on the environment and help decide how to best protect natural resources.

Ocean applications: Monitor ocean circulation and current systems, measure ocean temperature and wave heights, and track sea ice. Data can be used to better understand the oceans and how to best manage ocean resources.

Coastal applications: Monitor shoreline changes, track sediment transport, and map coastal features. Data can be used for coastal mapping and erosion prevention.

1.2 SATELLITE IMAGE CLASSIFICATION

The currently available instruments (e.g., multi/hyperspectral , synthetic aperture radar, etc.) for earth observation generate more and more different types of airborne or satellite images with different resolutions (spatial resolution, spectral resolution, and temporal resolution). This raises an important demand for intelligent earth observation through remote sensing images, which allows the smart identification and classification of land use and land cover (LULC) scenes from airborne or space platforms.

1.2.2 METHODS

1) Pixel Based approach

Pixel sizes are typically coarser than, or at the best, similar in size to the objects of interest . Most of the methods for image analysis using remote sensing images developed since the early 1970s are based on per-pixel analysis, or even sub-pixel analysis for this conversion. With the advances of remote sensing technology, the spatial resolution is gradually finer than the typical object of interest and the objects are generally composed of many pixels, which has significantly increased the within class variability and single pixels do not come isolated but are knitted into an image full of spatial patterns

2) Object Based approach

The term "objects" represents meaningful semantic entities or scene components that are distinguishable in an image (e.g., a house, tree or vehicle in a 1:3000 scale color airphoto). The core task of is the production of a set of nonoverlapping segments (or polygons), that is, the partitioning of a scene image into meaningful geographically based objects or superpixels that share relatively homogeneous spectral, color, or texture information. Due to the superiority compared to pixel-level approaches, object-level methods have dominated the task of remote sensing image analysis for decades.

3) Semantic approach

Semantic-level remote sensing image scene classification which aims to label each scene image with a specific semantic class. Here, a scene image usually refers to a local image patch manually extracted from large scale remote sensing images that contain explicit semantic classes (e.g., commercial area, industrial area, and residential area).

1.2.3 DATASETS FOR SATELLITE IMAGE CLASSIFICATION

1)UC Merced Land-Use Dataset
2)WHU-RS19 Dataset
3)SIRI-WHU Dataset
4)RSSCN7 Dataset
5)RSC11 Dataset

1.2.4 DEEP LEARNING FOR REMOTE SENSING

In comparison with traditional handcrafted features that require a considerable amount of engineering skill and domain expertise, deep learning features are automatically learned from data using a general-purpose learning procedure via deep-architecture neural networks. This is the key advantage of deep learning methods. On the other hand, compared with aforementioned unsupervised feature learning methods that are generally shallow-structured models (e.g., sparse coding), deep learning models that are composed of multiple processing layers can learn more powerful feature representations of data with multiple levels of abstraction . In addition, deep feature learning methods have also turned out to be very good at discovering intricate structures and discriminative information hidden in high-dimensional data, and the features from toper layers of the deep neural network show semantic abstracting properties. All of these make deep features more applicable for semantic-level scene classification.

APPROACH

2.1 Motivation

We identified the problem as pixel to pixel mapping problem and their were two approaches to solving this[1].

Image Segmentation -

Segmentation refers to the process of partitioning a digital image into multiple segments. The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics

Image Classification-

Classification is an important task for all remote sensing applications, which partitions the images into homogenous regions, each of which corresponds to some particular landcover type.

Semantic segmentation techniques -

Currently, the most successful state-of-the-art deep learning techniques for semantic segmentation stem from a common forerunner: the Fully Convolutional Network (FCN) by Long et al.. The insight of that approach was to take advantage of existing CNNs as powerful visual models that are able to learn hierarchies of features. They transformed those existing and well-known classification models – AlexNet , VGG (16-layer net) , GoogLeNet, and ResNet – into fully convolutional ones by replacing the fully connected layers with convolutional ones to output spatial maps instead of classification scores. Those maps are upsampled using fractionally strided convolutions (also named deconvolutions) to produce dense per-pixel labeled outputs. This work is considered a milestone since it showed how CNNs can be trained end-to-end for this problem, efficiently learning how to make dense predictions for semantic segmentation with inputs of arbitrary sizes. This approach achieved a significant improvement in

segmentation accuracy over traditional methods on standard datasets like PASCAL VOC, while preserving efficiency at inference [2]
Despite the power and flexibility of the FCN model, it still lacks various features which hinder its application to certain problems and situations: its inherent spatial invariance does not take into account useful global context information, no instance-awareness is present by default, efficiency is still far from real-time execution at high resolutions, and it is not completely suited for unstructured data such as 3D point clouds or models.
UNet -achieves state-of the-art performance on various datasets.

METHODOLOGY AND IMPLEMENTATION

Data Preprocessing:

Since the amount of training data is low as compared to traditional image segmentation datasets the individual images are of high resolution and this can be a tradeoff between the total no. of training images and the resolution of the training images.
On training of the UNet model on the given batch of 14 images with their corresponding ground truth values. The accuracy obtained is lesser when compared to an approach in which we have cropped the 14 images into smaller images using custom cropping technique to give 16k images.

The Cropping technique:

To have sufficient training data from the given high definition images cropping is required to train the classifier which has about 31M parameters.
The crop size of 64x64 we find under-representation of the individual classes and the geometry and continuity of the objects is lost, decreasing the field of view of the convolutions.
Using a cropping window of 128x128 pixels with a stride of 32 resultant of **15887 training 414** validation images.

Corner cases -

For the cases where the no. of crops is not the multiple of the image dimensions we initially tried zero padding , we realised that adding padding will add unwanted artifacts in the form of black pixels in training and test images leading to training on false data and image boundary.So we padded the difference from the start of the image to it's deficit end and similarly for the top and bottom of the image. For Ex For padding the right end of the image we will take the columns from the left end and replace it adjacent to the right end to give a "rounded" augmentation.

One hot encoding

To classify the ground truth into classes we one hot encoded the input ground truth values by first identifying the RGB values of the classes to be predicted according to this table:

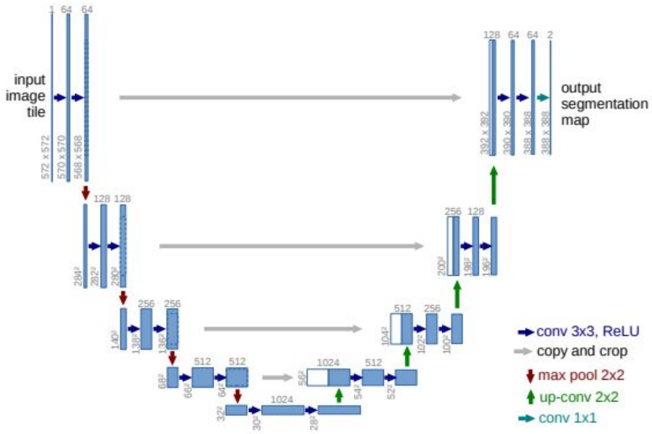
Class	Colour	RGB	Label	One hot
0	BLACK	(0,0,0)	Road	[1 0 0 0 0 0 0 0 0]
1	DARK GREEN	(0,125,0)	Tree	[0 1 0 0 0 0 0 0 0]
2	BROWN	(150,80,0)	Bare Soil	[0 0 1 0 0 0 0 0 0]
3	YELLOW	(255,255,0)	Rail	[0 0 0 1 0 0 0 0 0]
4	GREY	(100,100,100)	Building	[0 0 0 0 1 0 0 0 0]
5	GREEN	(0,255,0)	Field	[0 0 0 0 0 1 0 0 0]
6	BLUE	(0,0,150)	Water	[0 0 0 0 0 0 1 0 0]
7	PURPLE	(150,150,250)	Swimming pool	[0 0 0 0 0 0 0 1 0]
8	WHITE	(255,255,255)	Unclassified	[0 0 0 0 0 0 0 0 1]

Instead of training on the RGB values of the ground truth we have converted them into the one-hot values of different classes.
This approach yielded us a validation accuracy of 85% and training accuracy of 92% compared to 71% validation accuracy and 65% training accuracy when we were using the RGB ground truth values.

This might be due to decrease in variance and mean of the ground truth of training data as it acts as an effective normalization,
The architecture uses the input as cropped images (RGB) and after going through convolution layers with batch normalization the loss is calculated with one hot of the cropped ground truth.

ARCHITECTURE

Refrence Unet



Our Modified Unet with custom layers and Batch normalization

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, None, None, 4	0	
conv2d_1 (Conv2D)	(None, None, None, 6	2368	input_1[0][0]
conv2d_2 (Conv2D)	(None, None, None, 6	36928	conv2d_1[0][0]
batch_normalization_1 (BatchNor	(None, None, None, 6	256	conv2d_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, None, None, 6	0	batch_normalization_1[0][0]
conv2d_3 (Conv2D)	(None, None, None, 1	73856	max_pooling2d_1[0][0]
conv2d_4 (Conv2D)	(None, None, None, 1	147584	conv2d_3[0][0]
batch_normalization_2 (BatchNor	(None, None, None, 1	512	conv2d_4[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, None, None, 1	0	batch_normalization_2[0][0]
conv2d_5 (Conv2D)	(None, None, None, 2	295168	max_pooling2d_2[0][0]
conv2d_6 (Conv2D)	(None, None, None, 2	590080	conv2d_5[0][0]
batch_normalization_3 (BatchNor	(None, None, None, 2	1024	conv2d_6[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, None, None, 2	0	batch_normalization_3[0][0]
conv2d_7 (Conv2D)	(None, None, None, 5	1180160	max_pooling2d_3[0][0]
conv2d_8 (Conv2D)	(None, None, None, 5	2359808	conv2d_7[0][0]
batch_normalization_4 (BatchNor	(None, None, None, 5	2048	conv2d_8[0][0]
dropout_1 (Dropout)	(None, None, None, 5	0	batch_normalization_4[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, None, None, 5	0	dropout_1[0][0]
conv2d_9 (Conv2D)	(None, None, None, 1	4719616	max_pooling2d_4[0][0]
conv2d_10 (Conv2D)	(None, None, None, 1	9438208	conv2d_9[0][0]
batch_normalization_5 (BatchNor	(None, None, None, 1	4096	conv2d_10[0][0]
dropout_2 (Dropout)	(None, None, None, 1	0	batch_normalization_5[0][0]
up_sampling2d_1 (UpSampling2D)	(None, None, None, 1	0	dropout_2[0][0]
conv2d_11 (Conv2D)	(None, None, None, 5	2097664	up_sampling2d_1[0][0]
concatenate_1 (Concatenate)	(None, None, None, 1	0	dropout_1[0][0] conv2d_11[0][0]
conv2d_12 (Conv2D)	(None, None, None, 5	4719104	concatenate_1[0][0]
conv2d_13 (Conv2D)	(None, None, None, 5	2359808	conv2d_12[0][0]
batch_normalization_6 (BatchNor	(None, None, None, 5	2048	conv2d_13[0][0]
up_sampling2d_2 (UpSampling2D)	(None, None, None, 5	0	batch_normalization_6[0][0]
conv2d_14 (Conv2D)	(None, None, None, 2	524544	up_sampling2d_2[0][0]
concatenate_2 (Concatenate)	(None, None, None, 5	0	batch_normalization_3[0][0] conv2d_14[0][0]
conv2d_15 (Conv2D)	(None, None, None, 2	1179904	concatenate_2[0][0]
conv2d_16 (Conv2D)	(None, None, None, 2	590080	conv2d_15[0][0]
batch_normalization_7 (BatchNor	(None, None, None, 2	1024	conv2d_16[0][0]
up_sampling2d_3 (UpSampling2D)	(None, None, None, 2	0	batch_normalization_7[0][0]
conv2d_17 (Conv2D)	(None, None, None, 1	131200	up_sampling2d_3[0][0]
concatenate_3 (Concatenate)	(None, None, None, 2	0	batch_normalization_2[0][0] conv2d_17[0][0]
conv2d_18 (Conv2D)	(None, None, None, 1	295040	concatenate_3[0][0]
conv2d_19 (Conv2D)	(None, None, None, 1	147584	conv2d_18[0][0]
batch_normalization_8 (BatchNor	(None, None, None, 1	512	conv2d_19[0][0]
up_sampling2d_4 (UpSampling2D)	(None, None, None, 1	0	batch_normalization_8[0][0]
conv2d_20 (Conv2D)	(None, None, None, 6	32832	up_sampling2d_4[0][0]
concatenate_4 (Concatenate)	(None, None, None, 1	0	batch_normalization_1[0][0] conv2d_20[0][0]
conv2d_21 (Conv2D)	(None, None, None, 6	73792	concatenate_4[0][0]
conv2d_22 (Conv2D)	(None, None, None, 6	36928	conv2d_21[0][0]
conv2d_23 (Conv2D)	(None, None, None, 1	9232	conv2d_22[0][0]

batch_normalization_8 (BatchNor	(None, None, None, 1	512	conv2d_19[0][0]
up_sampling2d_4 (UpSampling2D)	(None, None, None, 1	0	batch_normalization_8[0][0]
conv2d_20 (Conv2D)	(None, None, None, 6	32832	up_sampling2d_4[0][0]
concatenate_4 (Concatenate)	(None, None, None, 1	0	batch_normalization_1[0][0] conv2d_20[0][0]
conv2d_21 (Conv2D)	(None, None, None, 6	73792	concatenate_4[0][0]
conv2d_22 (Conv2D)	(None, None, None, 6	36928	conv2d_21[0][0]
conv2d_23 (Conv2D)	(None, None, None, 1	9232	conv2d_22[0][0]
batch_normalization_9 (BatchNor	(None, None, None, 1	64	conv2d_23[0][0]
conv2d_24 (Conv2D)	(None, None, None, 3	51	batch_normalization_9[0][0]

Total params: 31,053,123
Trainable params: 31,047,331
Non-trainable params: 5,792

Comparison with Pyramid Scene Parsing net

We also implemented our own Pyramid scene parsing net(PSPnet) We have included the PSP net code in the source code of the project. PSPnet is a recent development from Unet giving state of architecture results. It came first in ImageNet scene parsing challenge 2016, PASCAL VOC 2012 benchmark and Cityscapes benchmark. A single PSPNet yields the new record of mIoU accuracy 85.4% on PASCAL VOC 2012 and accuracy 80.2% on Cityscapes.**But we found it to result in lower accuracy 49% training and 60% validation accuracy and also the learning rate was very slow (LR 1e-6) compared to our proposed unet solution (LR 1e-4).** The reason of PSP nets poor performance being less data and underfitting as the parameters were 46M because it uses resnet in it's structure and there wasn't enough data for it to train.

RESULTS

```
Total params: 31,053,225
Trainable params: 31,047,433
Non-trainable params: 5,792
________________________________________________________________________________
Train on 15887 samples, validate on 414 samples
Epoch 1/5
15887/15887 [==============================] - 615s 39ms/step - loss: 1.2578 - acc: 0.6633 - iou: 0.9881 - val_loss: 1.2270 - val_acc: 0.6964 - val_iou: 0.9908
Epoch 2/5
15887/15887 [==============================] - 587s 37ms/step - loss: 0.8799 - acc: 0.7836 - iou: 0.9907 - val_loss: 0.7549 - val_acc: 0.8013 - val_iou: 0.9930
Epoch 3/5
15887/15887 [==============================] - 587s 37ms/step - loss: 0.7072 - acc: 0.8168 - iou: 0.9921 - val_loss: 0.6823 - val_acc: 0.8020 - val_iou: 0.9933
Epoch 4/5
15887/15887 [==============================] - 587s 37ms/step - loss: 0.5861 - acc: 0.8409 - iou: 0.9933 - val_loss: 0.6975 - val_acc: 0.7932 - val_iou: 0.9938
Epoch 5/5
15887/15887 [==============================] - 586s 37ms/step - loss: 0.5029 - acc: 0.8562 - iou: 0.9941 - val_loss: 0.6680 - val_acc: 0.8115 - val_iou: 0.9945
```

```
Total params: 31,053,225
Trainable params: 31,047,433
Non-trainable params: 5,792
________________________________________________________________________________
Train on 15887 samples, validate on 414 samples
Epoch 1/15
15887/15887 [==============================] - 619s 39ms/step - loss: 0.4382 - acc: 0.8663 - iou: 0.9949 - val_loss: 0.6259 - val_acc: 0.8141 - val_iou: 0.9948
Epoch 2/15
15887/15887 [==============================] - 590s 37ms/step - loss: 0.3804 - acc: 0.8770 - iou: 0.9955 - val_loss: 0.6266 - val_acc: 0.8128 - val_iou: 0.9951
Epoch 3/15
15887/15887 [==============================] - 591s 37ms/step - loss: 0.3410 - acc: 0.8840 - iou: 0.9960 - val_loss: 0.5773 - val_acc: 0.8185 - val_iou: 0.9954
Epoch 4/15
15887/15887 [==============================] - 591s 37ms/step - loss: 0.3089 - acc: 0.8915 - iou: 0.9964 - val_loss: 0.6105 - val_acc: 0.8196 - val_iou: 0.9956
Epoch 5/15
15887/15887 [==============================] - 591s 37ms/step - loss: 0.2839 - acc: 0.8976 - iou: 0.9967 - val_loss: 0.5470 - val_acc: 0.8263 - val_iou: 0.9958
Epoch 6/15
15887/15887 [==============================] - 591s 37ms/step - loss: 0.2612 - acc: 0.9038 - iou: 0.9970 - val_loss: 0.6605 - val_acc: 0.8114 - val_iou: 0.9956
Epoch 7/15
15887/15887 [==============================] - 591s 37ms/step - loss: 0.2462 - acc: 0.9081 - iou: 0.9971 - val_loss: 0.5699 - val_acc: 0.8208 - val_iou: 0.9959
Epoch 8/15
15887/15887 [==============================] - 591s 37ms/step - loss: 0.2269 - acc: 0.9143 - iou: 0.9974 - val_loss: 0.5885 - val_acc: 0.8181 - val_iou: 0.9958
Epoch 9/15
15887/15887 [==============================] - 592s 37ms/step - loss: 0.2114 - acc: 0.9195 - iou: 0.9975 - val_loss: 0.6100 - val_acc: 0.8209 - val_iou: 0.9960
Epoch 10/15
15887/15887 [==============================] - 591s 37ms/step - loss: 0.2029 - acc: 0.9222 - iou: 0.9976 - val_loss: 0.6814 - val_acc: 0.8127 - val_iou: 0.9959
Epoch 11/15
15887/15887 [==============================] - 592s 37ms/step - loss: 0.1882 - acc: 0.9273 - iou: 0.9978 - val_loss: 0.6118 - val_acc: 0.8195 - val_iou: 0.9960
Epoch 12/15
15887/15887 [==============================] - 592s 37ms/step - loss: 0.1792 - acc: 0.9304 - iou: 0.9979 - val_loss: 0.7451 - val_acc: 0.8113 - val_iou: 0.9959
Epoch 13/15
15887/15887 [==============================] - 593s 37ms/step - loss: 0.1685 - acc: 0.9342 - iou: 0.9980 - val_loss: 0.6881 - val_acc: 0.8186 - val_iou: 0.9961
Epoch 14/15
15887/15887 [==============================] - 594s 37ms/step - loss: 0.1583 - acc: 0.9380 - iou: 0.9982 - val_loss: 0.6930 - val_acc: 0.8206 - val_iou: 0.9961
Epoch 15/15
15887/15887 [==============================] - 593s 37ms/step - loss: 0.1505 - acc: 0.9408 - iou: 0.9982 - val_loss: 0.6404 - val_acc: 0.8225 - val_iou: 0.9962
manideep2510@research-vm:~/interiit$ python test_unet.py
```

Training on 1st 13 images and testing on last image

Final Training Accuracy 94.08% [Images]

Validation accuracy 82.25%

CONCLUSION

The UNet architecture with one hot encoded ground truth images provides a higher accuracy model with training accuracy of 92% and validation accuracy of 82%. There can be further improvements in this architecture by adding data augumentation, tuning hyperparameters. We have also explored more complex architectures such as PSPnet which have failed to give good accuracy due to a shortage of data. More data (no. of images) will lead to the adoption of more complex techniques.