ABSTRACT

Satellite imagery is important for many applications including disaster response, law enforcement, and environmental monitoring. These applications require the manual identification of objects and facilities in the imagery. Because the geographic expenses to be covered are great and the analysts available to conduct the searches are few, automation is required. Yet traditional object detection and classification algorithms are too inaccurate and unreliable to solve the problem. Deep learning is a family of machine learning algorithms that have shown promise for the automation of such tasks. It has achieved success in image understanding by means of convolutional neural networks (CNNs). Also, there has been a massive growth in Deep learning in many fields such as computer vision and natural language processing. But, still there exists a lack of deep review for the datasets and methods available for scene classification from the satellite imagery. This paper focuses on enlightening the concept and evolution of Deep Learning.

Contents

CHAPTER 1

INTRODUCTION

- Image classification is an important part of remote sensing, image analysis and pattern recognition. In some instances, the classification itself may be the object of the analysis. For example, classification of land use from remotely sensed data produces a map-like image as the final product. The image classification therefore forms an important tool for examination of digital images. At present, there are different image classification procedures used for different purposes by various researchers. Over the last decade, remote sensing technologies are available easily and in abundance. Recently remote sensing images are being used widely for urban area classification and change detection.

- Classification is used for various applications like Crop monitoring (crop condition, crop type, and crop production estimation), Soil mapping characteristics (soil type, soil erosion and soil moisture), Forest cover mapping differences(leaf density, canopy texture), Land cover change detection (seasonal and annual changes), Natural disaster assessment (flood, cyclone and heavy raining), Water resource applications, wetland mapping, and environment inventory, urban and Regional planning and similarly other objects of interest can be observed using the remote sensing data.

- The selection of a suitable classifier to process the satellite images has an important role with respect to the efficient and accurate classification results. In this work, we are creating a system to classify satellite images in order to extract information using image processing techniques. Classification of satellite images into used and unused areas and also sub-classing of each of the classes into four different classes has been carried out. Used satellite images are further classified into residential, industries, highways, crop lands, and unused images are classified further into forest, river, deserts, and beaches. Manual classification by using image interpretation techniques requires more time and field experts. So in our work, we focused on efficient automatic satellite image classification.

- Convolutional neural networks are used for feature extraction and classification of satellite images. CNN is a deep neural network which is most suitable when we deal with images. CNN will help to provide higher classification accuracy. Confusion matrix is used to estimate the overall classification accuracy. Deep learning is a class of machine learning models that represent data at different levels of abstraction by means of multiple processing layers. It has achieved astonishing success in object detection and classification by combining large neural network models, called CNN with powerful GPU.

- CNN-based algorithms have dominated the annual ImageNet Large Scale Visual Recognition Challenge for detecting and classifying objects in photographs. This success has caused a revolution in image understanding, and the major technology companies, including Google, Microsoft and Facebook, have already deployed CNN-based products and services. Satellite image classification is the most significant technique used in remote sensing for the computerized study and pattern recognition of satellite information, which is based on diversity structures of the image that involve rigorous validation of the training samples depending on the used classification algorithm. It is an extreme part of remote sensing that depends originally on the image resolution, which is the most important quality factor in images. Image Classification or segmentation is a partitioning of an image into sections or regions.

- The power of such algorithms depends on the way of extracting the information from the huge amount of data found in images. Then, according to this information, pixels are grouped into meaningful classes that enable to interpret, mining, and studying various types of regions that are included in the image. Many applications based on using Landsat imagery in a quantitative fashion require classification of image pixels into a number of relevant categories or distinguishable classes. Image classification or land use/land cover classes were identified using supervised/pixel-based/image classification techniques. Image classification is the process of creating thematic maps from satellite imagery. A thematic map is an information representation of an image that shows the spatial distribution of a particular theme.

CHAPTER 2

OVERVIEW

2.1 Problem Statement:

To segment and detect the images received from satellites by using a deep learning system and image processing which classifies objects and facilities in high-resolution multi spectral satellite imagery.

2.2 Research Questions:

- What is CNN?
- Why use CNN?
- What is the difference between ANN and CNN?

2.3 Architectural Overview:

- A CNN consists of a series of processing layers as shown in Fig1. Each layer is a family of convolution filters that detect image features. Near the end of the series, the CNN combines the detector outputs in fully connected “dense” layers, finally producing a set of predicted probabilities, one for each class. The network itself learns which features to detect, and how to detect them, as it trains.

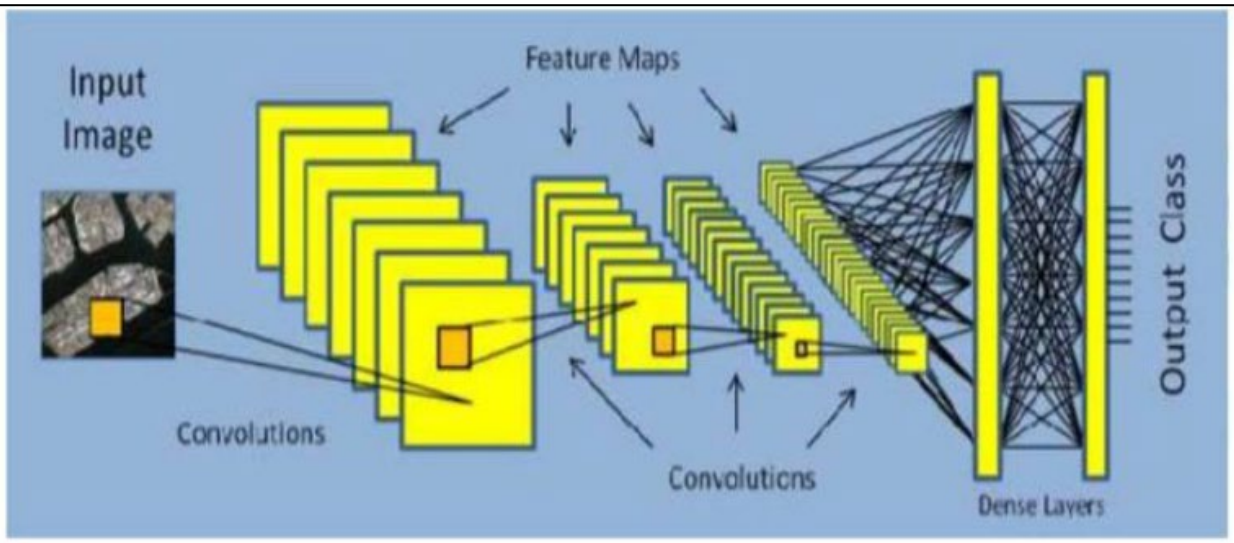


Figure.1. Architecture of Satellite Image classification.

General Framework:

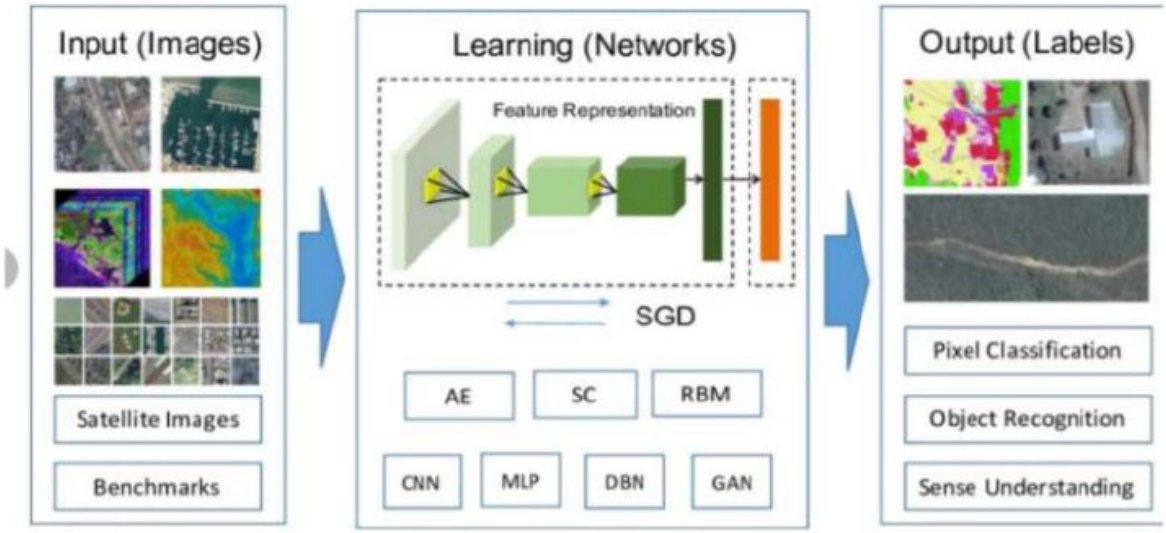


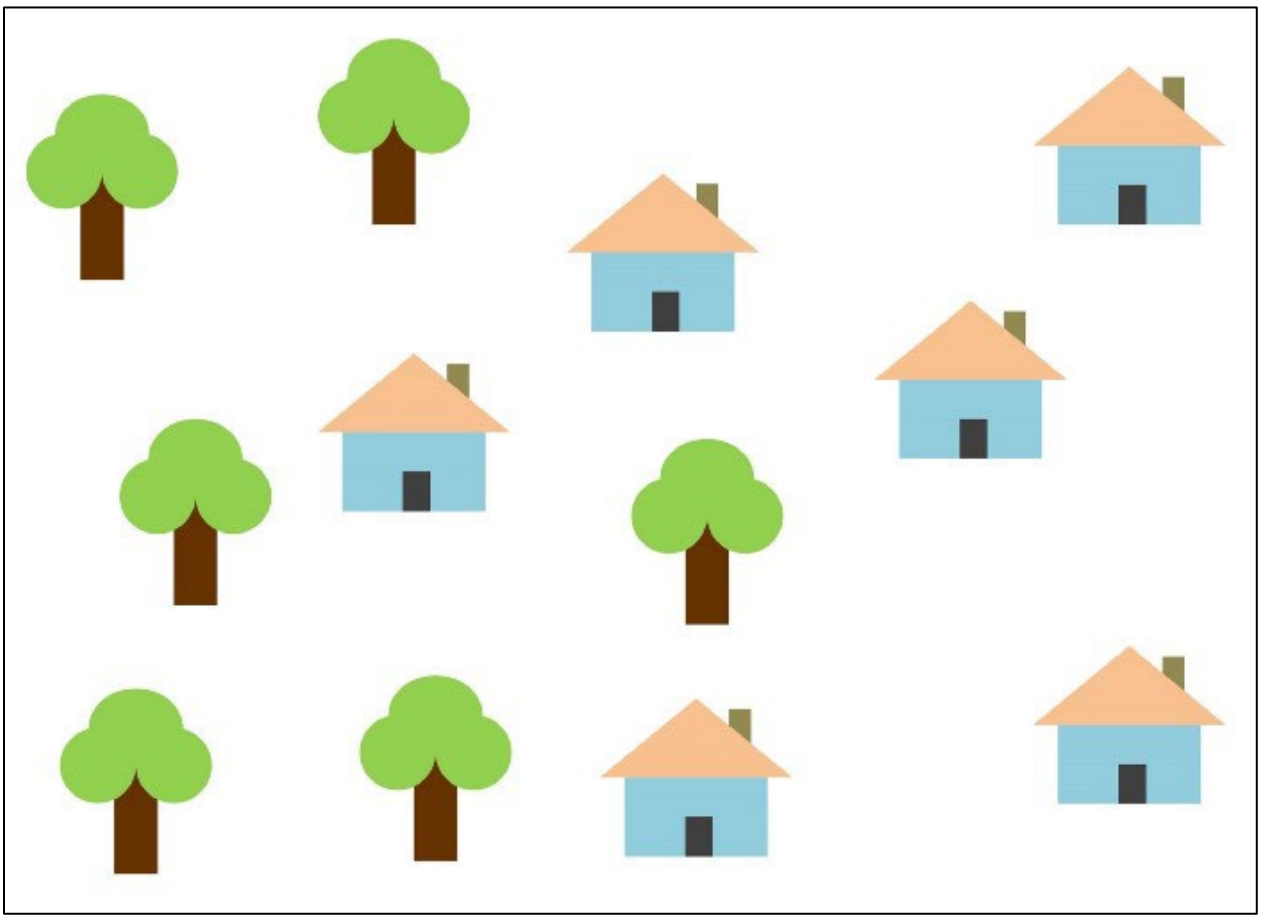
Figure.2. Architecture overview of Classification of satellite images

- The general Framework of Deep Learning based method for satellite image classification can usually be described as three components: prepared input data, deep networks and expected output, as shown in Figure. The input data include different kinds of satellite images datasets (i.e. benchmarks used for training), while the output is the manually defined label of the pixels per land corner type of upsell and the object type in the image tiles. The deep networks we stacked by multiple outliner neural layers,

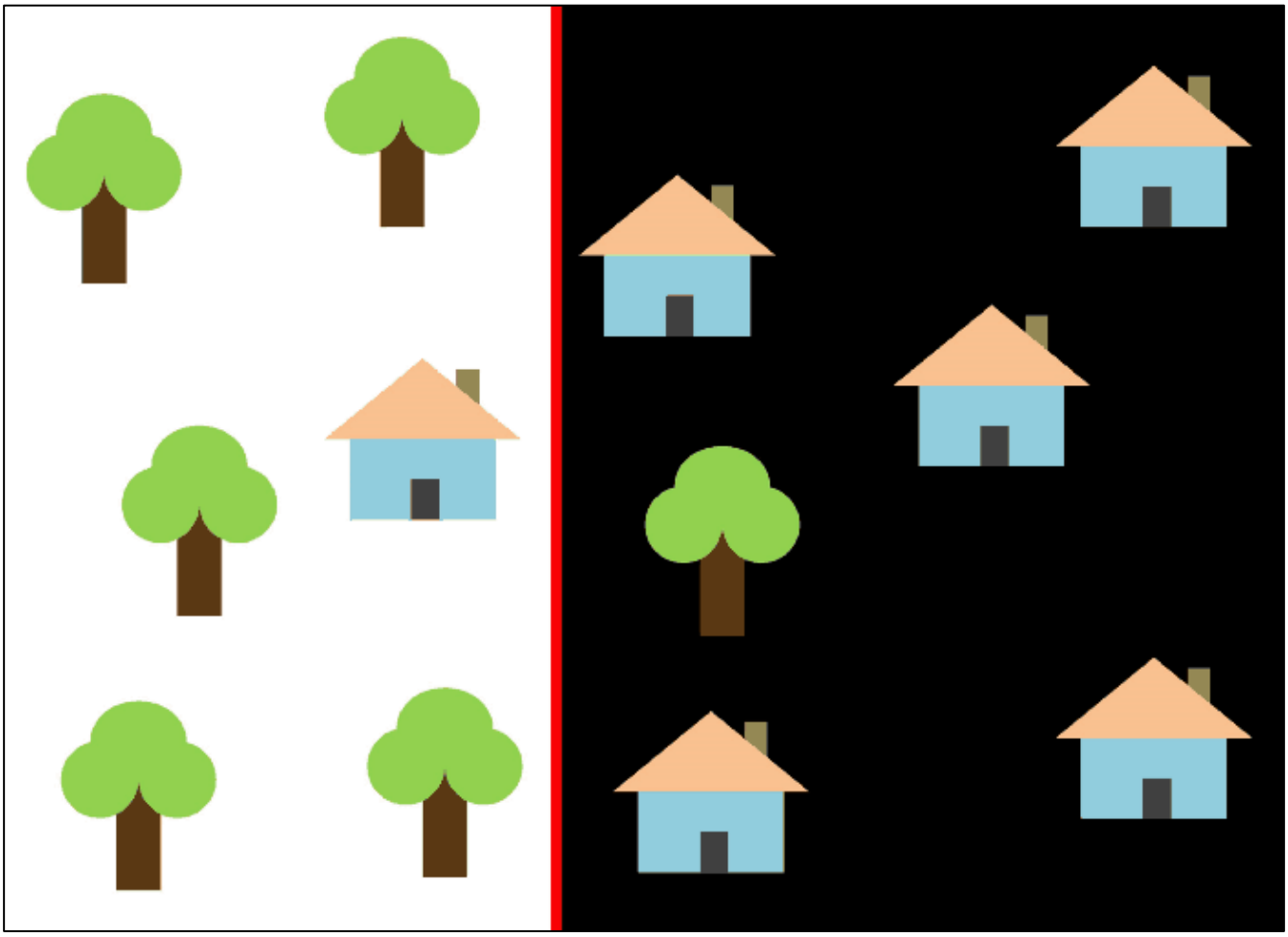
where such intermediate layer model input data to features encodes low level features to high-based features, the parameters of are usually learned by DL model like an Al unlabelled training images, while the overall parts are fine-tuned by algorithms like Stochastic Gradient Descent (SGD) with the supervision of label training. Satellite image classification with Deep Learning where classic deep networks like CNNs, Multiplayer Perceptions (MLPs) and Deep Belief Networks (DBNs) as well as feature learning algorithms like AEs, Restricted Boltzmann Machines (RBMs) and Sparse Coding (SC) are introduced and the studies are classified into four kinds according to the purpose, namely image pre-processing, pixel-based classification, target recognition and scene understanding. Except for the above deep networks, we supplement another unsupervised deep model named Generative Adversarial Networks (GANs) which have been widely explored.

2.4 The general problem with satellite data:

- Two or more feature classes (e.g., built-up/ barren/ quarry) in the satellite data can have similar spectral values, which has made the classification a challenging task in the past couple of decades. The conventional supervised and unsupervised methods fail to be the perfect classifier due to the aforementioned issue, although they robustly perform the classification. But there are always related issues. Let us understand this with the example below:



- In the above figure, using a vertical line as a classifier and move it only alongthe x-axis in such a way that it classifies all the images to its right as houses, the answer might not be straight forward. This is because the distribution of data is in such a way that it is impossible to separate them with just one vertical line. However, this doesn't mean that the houses can't be classified at all!



- Let say the red line was used, as shown in the figure above, to separate the two features. In this instance, the majority of the houses were identified by the classifier but, a house was still left out, and a tree got misclassified as a house. To make sure that not even a single house is left behind, we might use the blue line. In that case, the classifier will cover all the house; this is called a high **recall**. However, not all the classified images are truly houses, this is called a low **precision**. Similarly, if we use the green line, all the images classified as houses are houses; therefore, the classifier possesses high precision. The recall will be lesser in this case because three houses were still left out. In the majority of cases, this **trade-off** between precision and recall holds.

- The house and tree problem demonstrated above is analogous to the built-up, quarry and barren land case. The classification priorities for satellite data can vary with the purpose. For example, if you want to make sure that all the built-up cells are classified as built-up, leaving none behind, and you care less about pixels of other classes with similar signatures being classified as built-up, then a model with a high recall is required. On the contrary, if the priority is to classify pure built-up pixels only without including any of the other class pixels, and you are okay to let go of mixed built-up pixels, then a high precision classifier is required. A generic model will use the red line in the case of the house and the tree to maintain the balance between precision and recall.

2.5 Data used in the current scope:

- Here, we will treat six bands (band 2 — band 7) of Landsat 5 TM as features and try to predict the binary built-up class. A multispectral Landsat 5 data acquired in the year2011 for Bangalore and its corresponding binary built-up layer will be used for training and testing. Finally, another multispectral Landsat 5 data acquired in the year 2011 for Hyderabad will be used for new predictions

CHAPTER 3

METHODOLOGY
and
OUTPUT ANALYSIS

3.1 Method:

- We will be using Google's Tensorflow library in Python to build a Neural Network (NN). The following other libraries will be required, please make sure you install them in advance:

    - Pyrsgis — to read and write GeoTIFF
    - scikit-learn — for data pre-processing and accuracy checks
    - NumPy — for basic array operations

- Placing all the three files in a directory — assign the path and input file names in thescript, and read the GeoTIFF files.

- The raster module of the pyrsgis package reads the GeoTIFF's geolocation information and the digital number (DN) values as a NumPy array separately. For details on this, please refer to the pyrsgis page.

- Printing the size of the data that we have read.

3.2 Code Analysis:

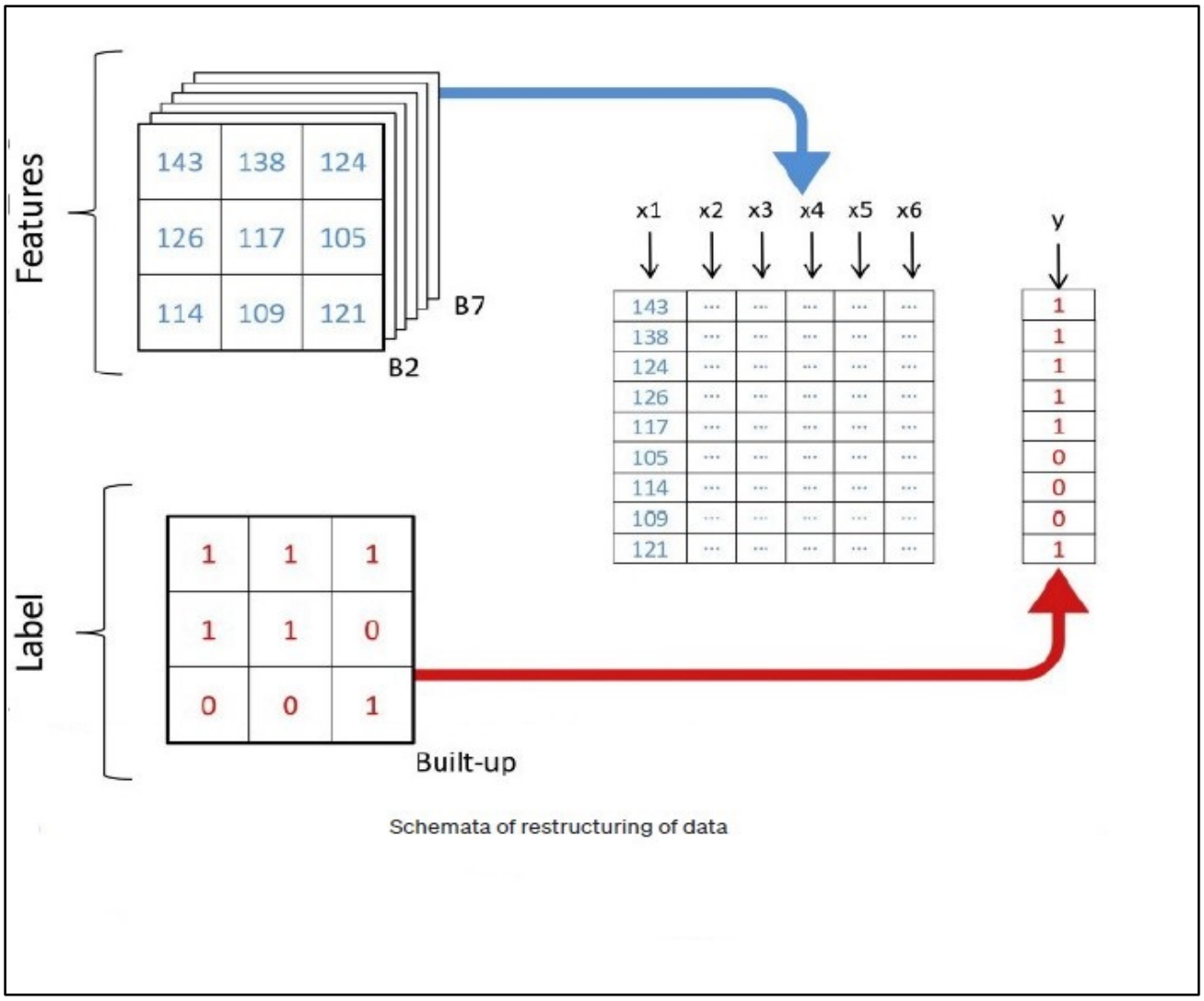
```
1  print("Bangalore multispectral image shape: ", featuresBangalore.shape)
2  print("Bangalore binary built-up image shape: ", labelBangalore.shape)
3  print("Hyderabad multispectral image shape: ", featuresHyderabad.shape)
```

Output:

```
Bangalore multispectral image shape: 6, 2054, 2044
Bangalore binary built-up image shape: 2054, 2044
Hyderabad multispectral image shape: 6, 1318, 1056
```

- As evident from the output, the number of rows and columns in the Bangalore images is the same, and the number of layers in the multispectral images are the same. The model will learn to decide whether a pixel is built-up or not based on the respective DN values across all the bands, and therefore, both the multispectral images should have the same number of features (bands) stacked in the same order.

- Changing the shape of the arrays to a two-dimensional array, which is expected by the majority of ML algorithms, where each row represents a pixel. The convert module of the pyrsgis package will do that for us.



Schemata of restructuring of data

```
featuresBangalore = changeDimension(featuresBangalore)
labelBangalore = changeDimension (labelBangalore)
featuresHyderabad = changeDimension(featuresHyderabad)
nBands = featuresBangalore.shape[1]
labelBangalore = (labelBangalore == 1).astype(int)

print("Bangalore multispectral image shape: ", featuresBangalore.shape)
print("Bangalore binary built-up image shape: ", labelBangalore.shape)
print("Hyderabad multispectral image shape: ", featuresHyderabad.shape)
```

Output:

```
Bangalore multispectral image shape: 4198376, 6
Bangalore binary built-up image shape: 4198376
Hyderabad multispectral image shape: 1391808, 6
```

- In the above code snippet above, we extract all the pixels with the value one. This is a fail-safe to avoid issues due to No Data pixels that often has extreme high and low values.

- Spliting the data for training and validation. This is done to make sure that the model has not seen the test data and it performs equally well on new data. Otherwise, the model will over fit and perform well only on training data.

```
from sklearn.model_selection import train_test_split

xTrain, xTest, yTrain, yTest = train_test_split(featuresBangalore, labelBangalore, test_size=0.4,

print(xTrain.shape)
print(yTrain.shape)

print(xTest.shape)
print(yTest.shape)
```

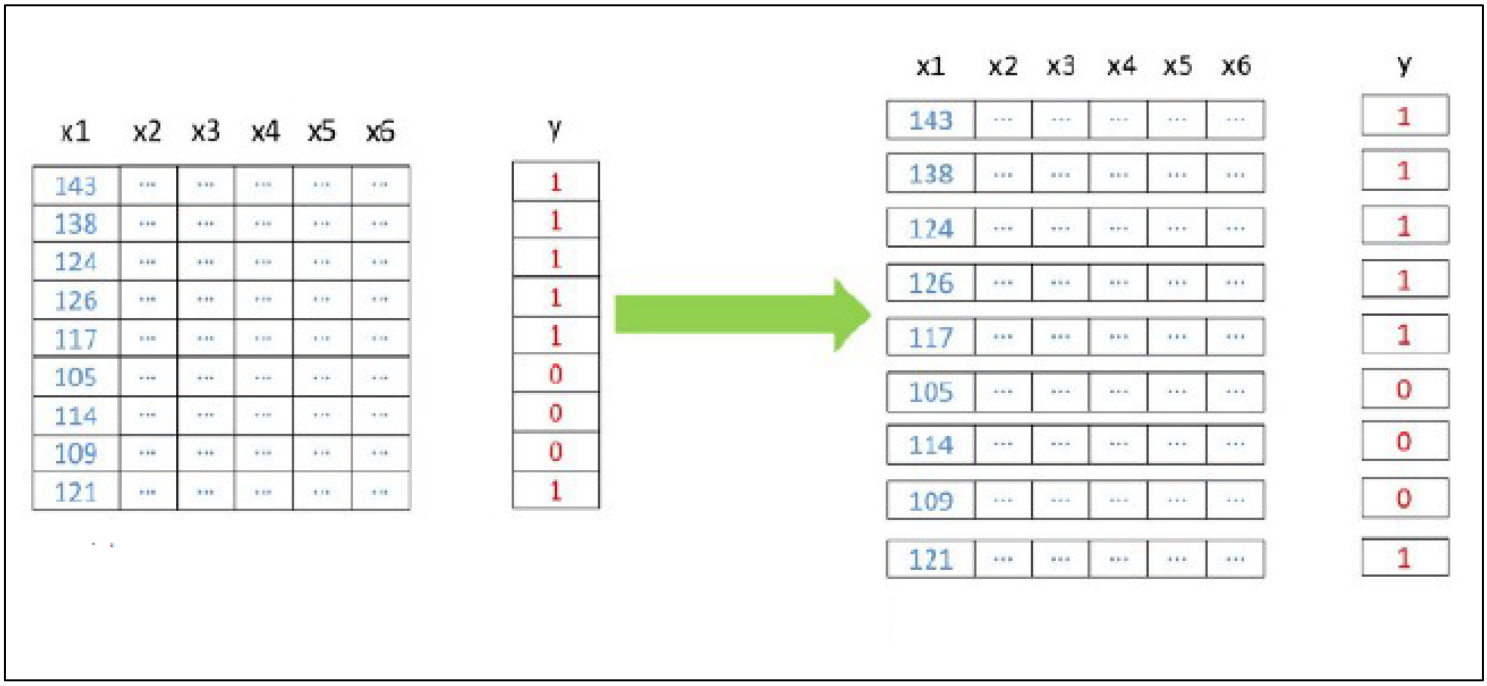
Output:

```
(2519025, 6)
(2519025,)
(1679351, 6)
(1679351,)
```

- The test size (0.4) in the code snippet above signifies that the training-testing proportion is 60/40.

- Many ML algorithms including NNs expect normalized data. This means that the histogram is stretched and scaled between a certain range (here, 0 to 1). We will normalize our features to suffice this requirement. Normalization can be achieved by subtracting the minimum value and dividing by range. Since the Landsat data is an 8-bitdata, the minimum and maximum values are 0 and 255 (2^8 = 256 values).

- Another additional pre-processing step is to reshape the features from two-dimensions to three-dimensions, such that each row represents an individual pixel.

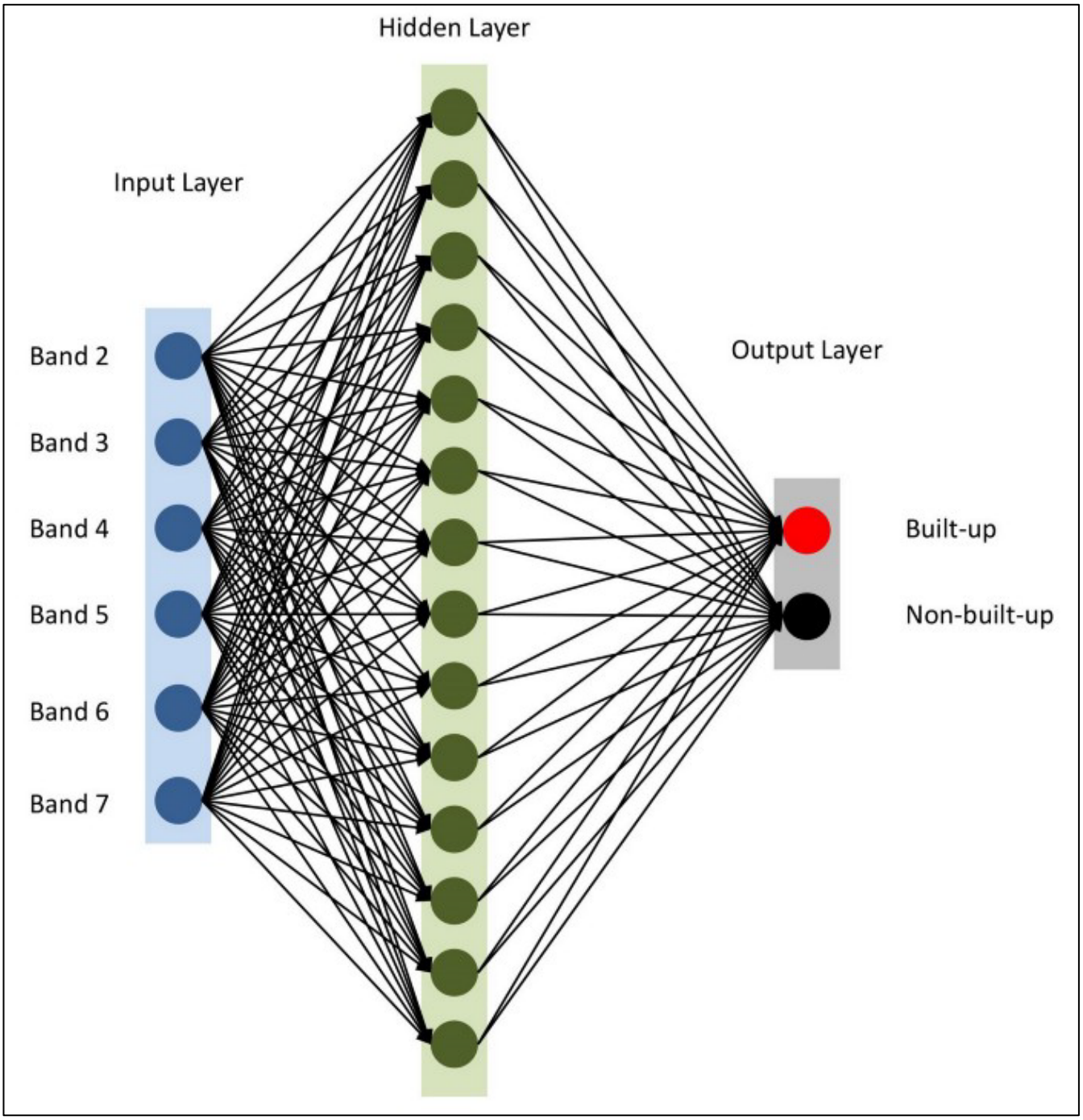



```
# Normalise the data
xTrain = xTrain / 255.0
xTest = xTest / 255.0
featuresHyderabad = featuresHyderabad / 255.0

# Reshape the data
xTrain = xTrain.reshape((xTrain.shape[0], 1, xTrain.shape[1]))
xTest = xTest.reshape((xTest.shape[0], 1, xTest.shape[1]))
featuresHyderabad = featuresHyderabad.reshape((featuresHyderabad.shape[0], 1, featuresHyderabad.

# Print the shape of reshaped data
print(xTrain.shape, xTest.shape, featuresHyderabad.shape)
```

Output:

```
(2519025, 1, 6) (1679351, 1, 6) (1391808, 1, 6)
```

- Building the model using **keras**. To start with, we will use the **sequential** model, to add the layers one after the other. There is one input layer with the number of nodes equal to n-Bands. One hidden layer with 14 nodes and 'relu' as the **activation function** is used. The final layer contains two nodes for the binary built-up class with 'softmax' activation function, which is suitable for categorical output.



```
from tensorflow import keras

# Define the parameters of the model
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(1, nBands)),
    keras.layers.Dense(14, activation='relu'),
    keras.layers.Dense(2, activation='softmax')])

# Define the accuracy metrics and parameters
model.compile(optimizer="adam", loss="sparse categorical crossentropy", metrics=["accuracy"])

# Run the model
model.fit(xTrain, yTrain, epochs=2)
```

- In the above code compile the model with ‘adam’ optimizer. (There are several others that you can check.) The loss type that we are using, for now, is the categorical-sparse-cross entropy. The metric for model performance evaluation is ‘accuracy’.

- Running the model on x-Train and y-Train with two **epochs** (or iterations). Fitting the model will take some time depending on the data size and computational power.

- The following can be seen after the model compilation:

```
2019-08-28 19:41:09.299466: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports ins
s TensorFlow binary was not compiled to use: AVX2
Epoch 1/2
2519025/2519025 [==============================] - 52s 21us/sample - loss: 0.1285 - accuracy: 0.9515
Epoch 2/2
2519025/2519025 [==============================] - 52s 21us/sample - loss: 0.1076 - accuracy: 0.9593
```

```
from sklearn.metrics import confusion_matrix, precision_score, recall_score

# Predict for test data
yTestPredicted = model.predict(xTest)
yTestPredicted = yTestPredicted[:,1]

# Calculate and display the error metrics
yTestPredicted = (yTestPredicted>0.5).astype(int)
cMatrix = confusion_matrix(yTest, yTestPredicted)
pScore = precision_score(yTest, yTestPredicted)
rScore = recall_score(yTest, yTestPredicted)

print("Confusion matrix: for 14 nodes\n", cMatrix)
print("\nP-Score: %.3f, R-Score: %.3f" % (pScore, rScore))
```

- Predicting values for the test and performing various accuracy checks. The SoftMax function generates separate columns for each class type probability values. We extract only for class one (built-up), as mentioned in the code snippet above. The models for geospatial-related analysis become tricky to evaluate because unlike other general ML problems, it would not be fair to rely on a generalized summed up error; the spatial location is the key to the winning model. Therefore, the confusion matrix, precision and recall can reflect a clearer picture of how well the model performs.

```
Confusion matrix: for 14 nodes
 [[1443164   37712]
 [  32062  166413]]

P-Score: 0.815, R-Score: 0.838
```

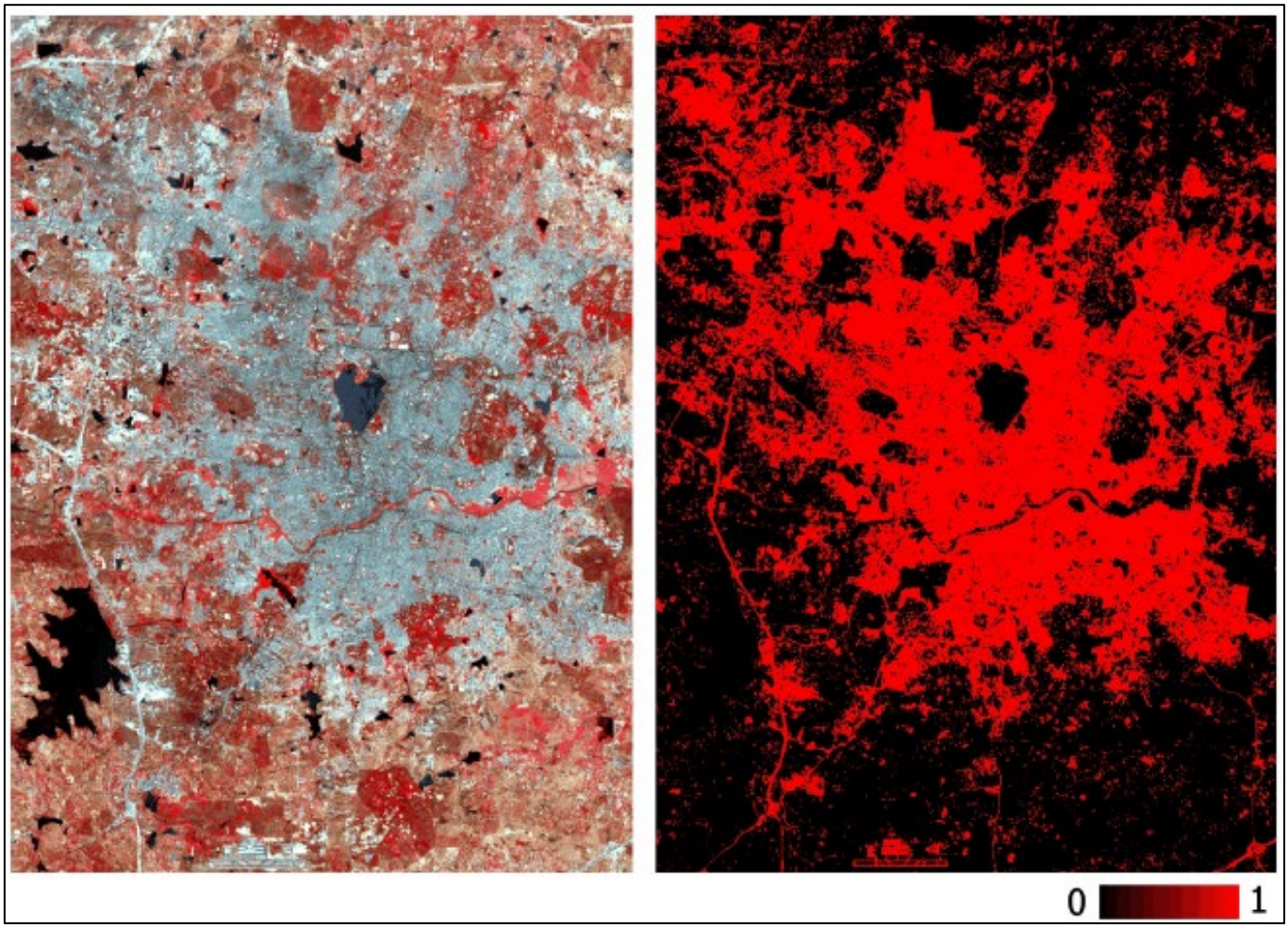
Confusion matrix, precision and recall as displayed in the terminal

- Confusion matrix above shows that, there are thousands of built-up pixels classified as non-built-up and vice versa, but the proportion to the total data size is less. The precision and recall as obtained on the test data are more than 0.8.

- Iterations can be performed to find the optimum number of hidden layers, the number of nodes in each hidden layer, and the number of epochs to get accuracy. Some commonly used remote sensing indices such as the NDBI or NDWI can also be used as features, as and when required. Once the desired accuracy is reached, using the model to predict for the new data and export the GeoTIFF. A similar model with minor tweaks can be applied for similar applications.

```
predicted = model.predict(featuresHyderabad)
predicted = predicted[:,1]

#Export raster
prediction = np.reshape(predicted, (ds.RasterYSize, ds.RasterXSize))
outFile = 'Hyderabad_2011_BuiltupNN_predicted.tif'
raster.export(prediction, ds3, filename=outFile, dtype='float')
```

- In this work we are exporting the GeoTIFF with the predicted probability values, and not its thresholded binary version. We can always threshold the float type layer in a GIS environment later, as shown in the image below.



Hyderabad built-up layer as predicted by the model using the multispectral data

- The accuracy of the model has been evaluated with precision and recall — the traditional checks can be done (e.g., kappa coefficient) on the new predicted raster. Apart from the aforementioned challenges of satellite data classification, other intuitive limitations include the inability of the model to predict on data acquired in different seasons and over different regions, due to variation in the spectral signatures.

REFERENCES

[1] Youtube - https://www.youtube.com/watch?v=PHJlZMwygHs

[2] USGS - https://earthexplorer.usgs.gov/

[3] GitHub - https://github.com/

[4] towardsdatascience.com