CHAPTER 1

INTRODUCTION

1.1 Motivation

Data science is getting more and more attraction in all the major industries. So, in this new, exciting and challenging field there are lots of opportunities. In order to use those opportunities basic understanding of the data analysis is required.

1.2 Objective

The objective of the project is to predict all the available labels in the image. It is a Multi-label image classification problem of the satellite images of the Amazon forest. The dataset has almost forty thousand images of the Amazon forest. Each image may have multiple labels. There are seventeen different types of labels. So the aim is to build a Convolutional Neural Network to process the images, label the images and test the accuracy achieved by the network.

CHAPTER 2

LITERATURE SURVEY

Various data mining techniques have been employed to predict image labels in recent years, such as artificial neural networks, decision trees, Bayesian method, logistic regression, and Support Vector Machine (SVM) and fuzzy methods. A review of the related literature showed that the choice of technique to a large extent depends on the parameters of the system. Supervised learning uses labeled data to train a model. Two types of taxonomy exists, regression, an algorithm that is meant for interval labels and, classification, an algorithm for class labels.

Considering all the facts, neural network gives the best technique for image classification. For this reason, the paper selected is about image classification of amazon rain forest using neural networks.

PAPERS VERIFIED:

The following papers were verified for learning Deep Learning.

1. Google cloud and Youtube-8M Video Understanding.
2. Self-Driving Car Steering Angle Prediction Based on Image Recognition
3. Video Understanding: From Video Classification to Captioning.
4. Automatic Sketch Colourization to generate acceptable colour images.
5. From 2D Sketch to 3D Shading and Multi-view Images.

CHAPTER 3

HARDWARE AND SOFTWARE SPECIFICATION

3.1 Hardware requirements:

For Developer:

- Processor : Intel Neon (8 CPUs)
- RAM : 16384 MB or higher
- Internal Memory : 8 GB or higher
- GPU : 8 GB
- CUDA Cores : 192

3.2 Software requirements:

For Developer:

- Development environment : Python 2.* or 3.*(Anaconda)
- Operating System : Windows,Linux
- Programming language : python.
- Software to be installed : CUDA, cuDNN, tensorflow gpu version .

3.3 Functional Features of Project:

- The classification can split the areas based on land terrain.
- The deforestation is monitored from the satellite.
- The specific land type favouring specific forestation can be determined.

CHAPTER 4

SYSTEM ANALYSIS

Deep Network:

Deep learning is a class of machine learning algorithms

- use a cascade of multiple layers of non linear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input.
- learn in supervised (e.g., classification) and/or unsupervised (e.g., pattern analysis) manners.
- learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.
- use some form of gradient descent for training via backpropagation.

Layers that have been used in deep learning include hidden layers of an artificial neural network and sets of propositional formulas. They may also include latent variables organized layer-wise in deep generative models such as the nodes in Deep Belief Networks and Deep Boltzmann Machines.

Types of neural networks:

- Radial Basis Network
- Deep Feed Forward
- Recurrent Neural Network
- Long/Short Term Memory
- Deep Belief Network
- Auto Encode

Convolutional Neural Network:

To create a network that is trained by satellite images with its corresponding tags.

1. To identify the tags(i.e., type of terrain) from the testing images.
2. For image classification, among all the neural networks, CNN forms the best option.

For all the above mentioned reasons, we select CNN for the multi label classification.

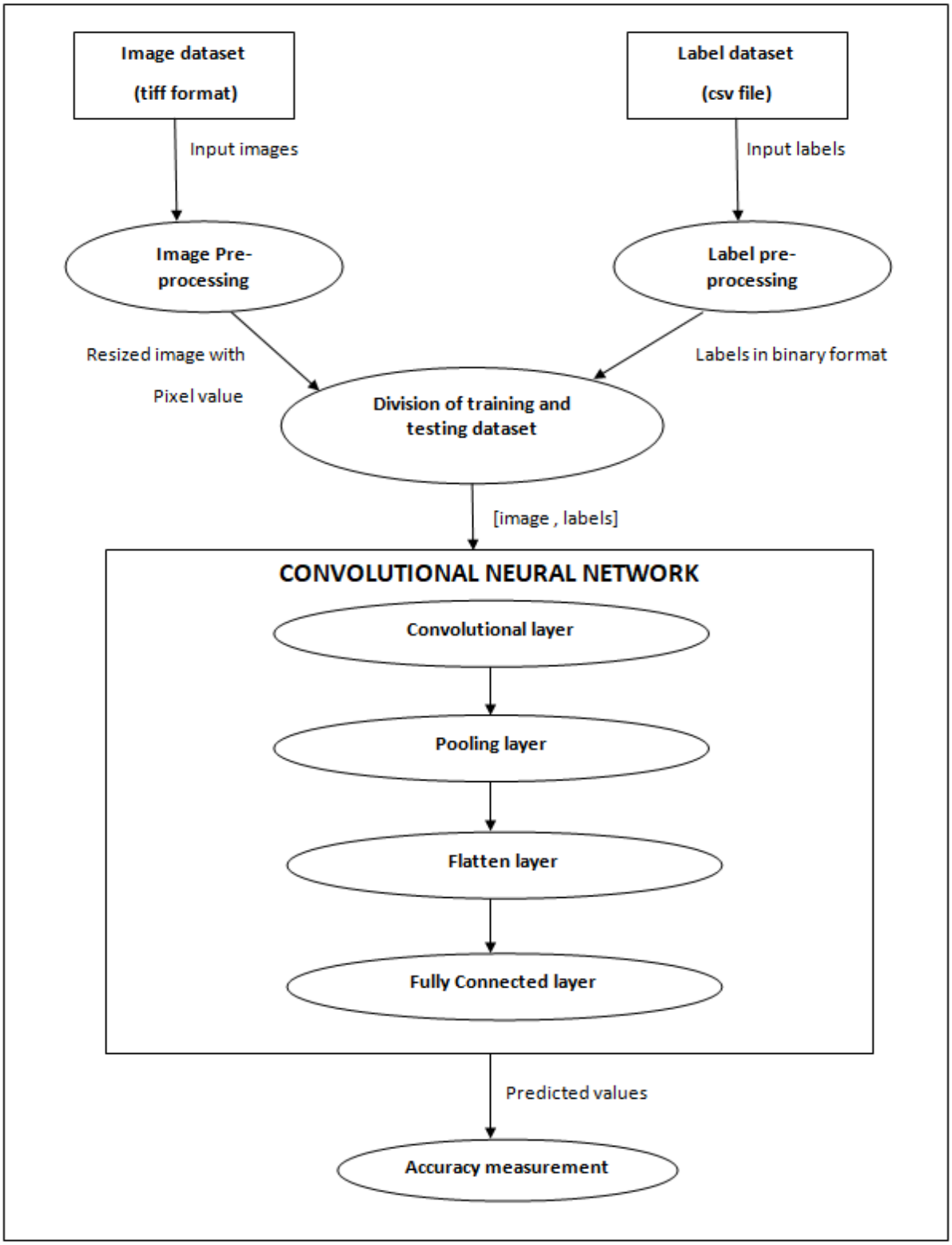

Figure 4.1

The figure 4.1 is the basic block diagram. The Block diagram involves five modules and in which one module has four sub modules. The modules are:

1) Image pre-processing
2) Label pre-processing
3) Division of training and testing dataset
4) Convolutional Neural Network
    a) Convolutional layer
    b) Pooling layer
    c) Flatten layer
    d) Fully connected layer
5) Accuracy Measurement

1) Image pre-processing:

This involves reading the image as a pixel value and reducing its size.

2) Label pre-processing:

This involves reading the image as a text and converting them into a binary value of 1 or 0.

3) Division of training and testing dataset:

Of all the images and labels read, it is divided into eighty percentage as training and twenty percentage as testing dataset.

4) Convolutional Neural Network:

This is used to process the image and identify the labels.

a) Convolutional layer:

Convolution operation is performed on the images to extract the features form the images.

b) Pooling layer:

Max pooling operation is performed to extract only the necessary features from the image.

c) Flatten layer:

The pixel values of the image are converted into a one dimensional array.

d) Fully Connected layer:

All the neurons are connected to the output layer which has seventeen neurons, one for each label.

5) Accuracy Measurement:

Of the predicted values for the test images, accuracy is measured (how similar is the predicted values to the actual value).

DATASET:

There are two files that contain the dataset. The first file is the image dataset that stores all the images to train and test. All the images are in tiff format to make processing easier. The second file is a csv file that holds all the labels for each individual image. It has two columns. The first column stores the image name and the second column stores the labels for that particular image.

There are 40479 images totally in the folder. Also for each image the corresponding tag is stored in the csv file.

CHAPTER 5

SYSTEM DESIGN

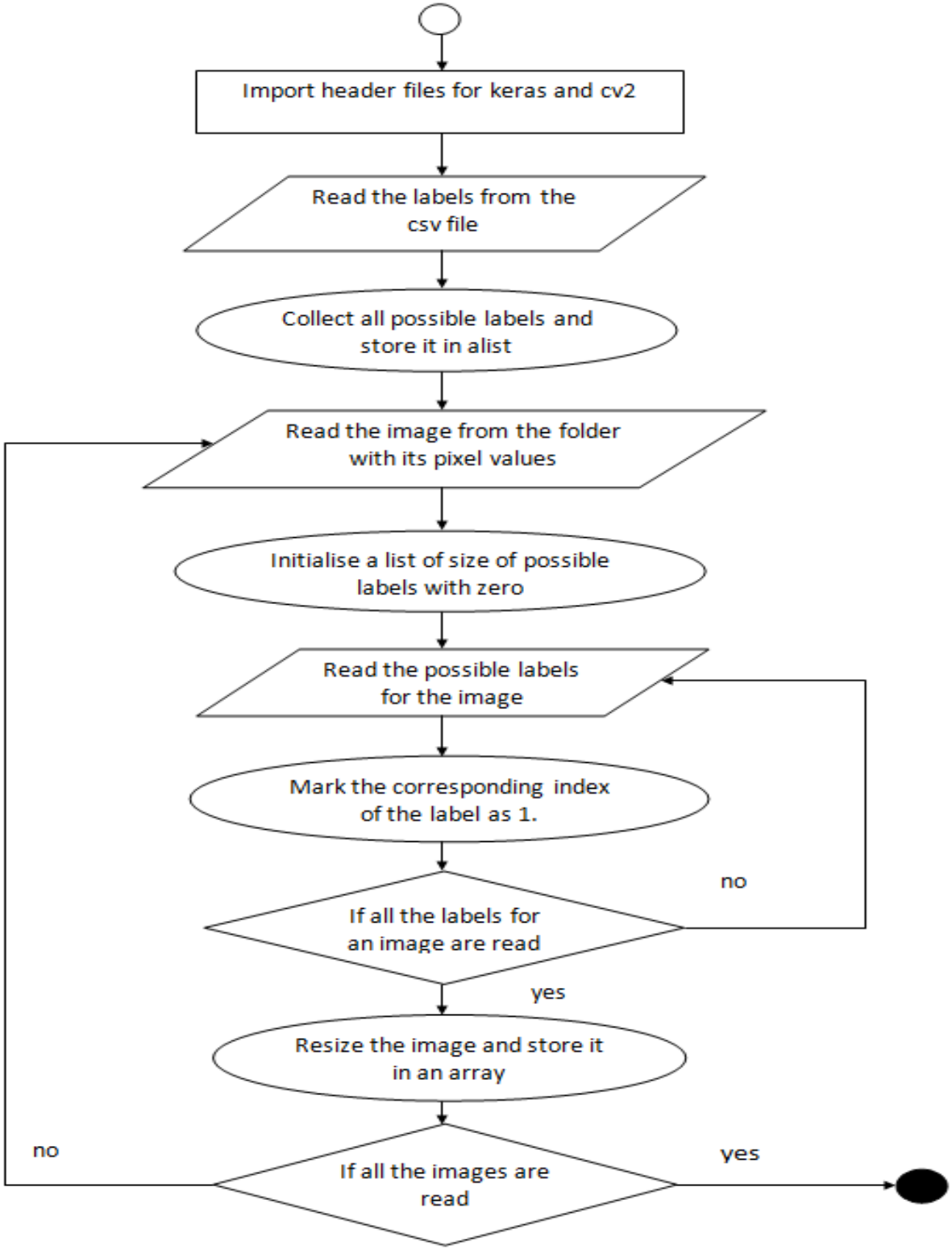

Figure 5.1

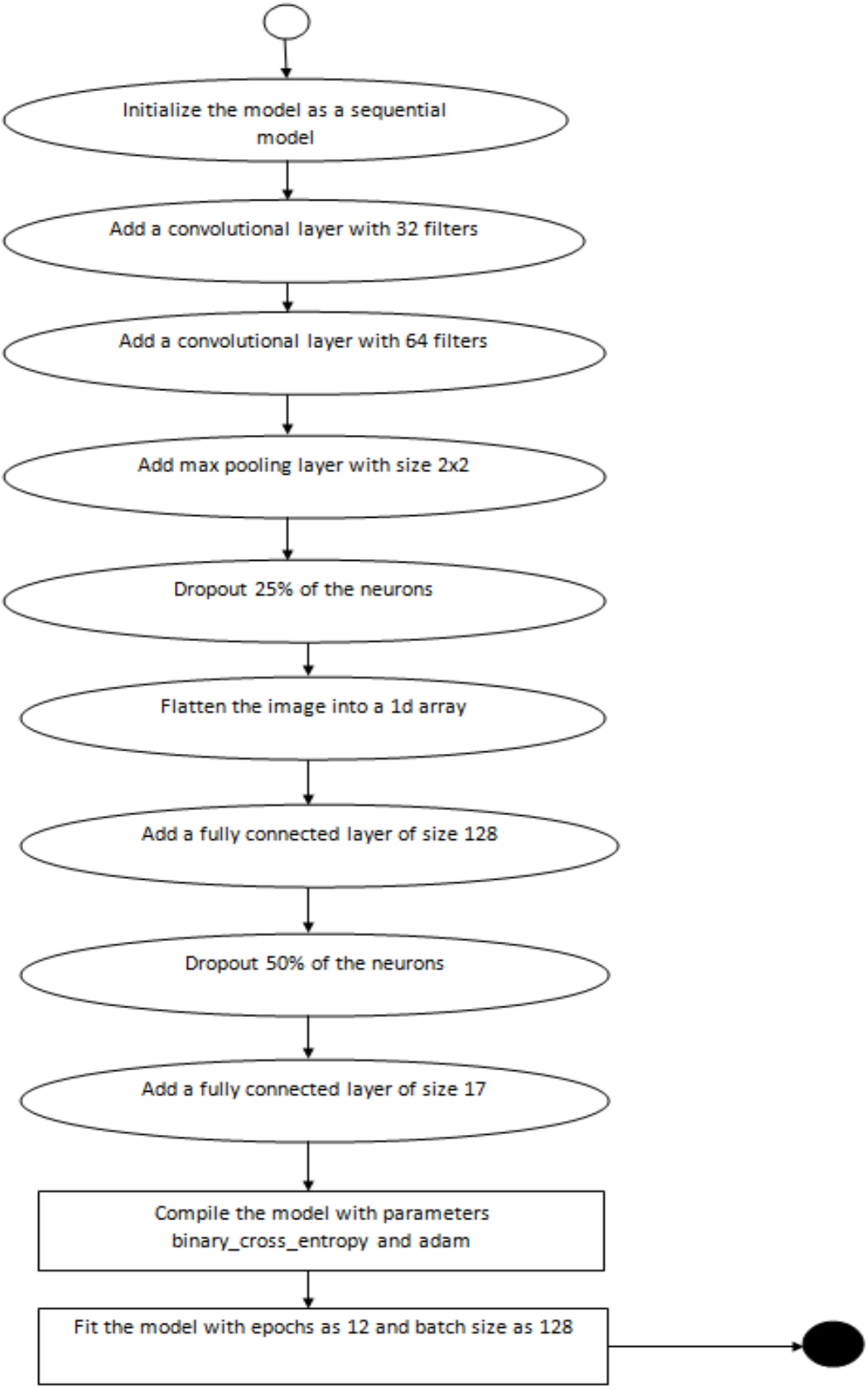
Initialize the model as a sequential model
Add a convolutional layer with 32 filters
Add a convolutional layer with 64 filters
Add max pooling layer with size 2x2
Dropout 25% of the neurons
Flatten the image into a 1d array
Add a fully connected layer of size 128
Dropout 50% of the neurons
Add a fully connected layer of size 17
Compile the model with parameters binary_cross_entropy and adam
Fit the model with epochs as 12 and batch size as 128

Figure 5.2

The figure 5.1 is the basic working of how the images and labels are processed before providing it into the neural network.

Step 1: Import the necessary header files for keras and cv2.
Step 2: Read the labels from csv file.
Step 3: Find all possible labels.
Step 4: Initialize an array of zeros with size as the number of possible labels.
Step 5: Read the pixel value of each image one by one.
Step 6: Read the possible labels for the image.
Step 7: For each image mark the index of the label as 1.
Step 8: Resize the image and store it in an array.

The figure 5.2 is the basic working of the neural network with its multiple layers.

Step 1: Initialize the network as a sequential network.
Step 2: Add a convolution2D layer with 32 filters.
Step 3: Add a convolution2D layer with 64 filters.
Step 4: Add a Max pooling layer of size 2x2.
Step 5: Dropout 25% of the neurons.
Step 6: Flatten the image into a 1-D array.
Step 7: Add a fully connected layer of size 128.
Step 8: Dropout 50% of the neurons.
Step 9: Add a fully connected layer of size 17.
Step 10: Compile the model.
Step 11: Fit the model

CHAPTER 6

IMPLEMENTATION

The project contains setting up environment, preprocessing, constructing model, fitting and training the model and test the trained model.

6.1 Setting up environment:

- Download the CUDA toolkit for NVIDIA through developer account.
- Install the tensorflow gpu version using conda command.

```
(tensorflow)C:> pip install --ignore-installed --upgrade tensorflow-gpu
```

Figure 6.1

- Download the cuDNN that forms the library for Neural Network to use CUDA.
- Set the environmental variable for the CUDA.
- Backend is tensorflow.

6.2 Preprocessing:

LANGUAGE USED: Python

Import the library files for manipulation on data.

Libraries imported:

- ✓ Numpy - number operations
- ✓ Cv2 - from image to pixel conversion(Opencv).
- ✓ Keras - for constructing the model.
- ✓ Sklearn- for measuring accuracy.

```
#keras model is imported for constructing the model
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from sklearn.metrics import accuracy_score
```

Figure 6.2

1. Import the tag containing csv file.
2. for each row in the csv file, the tags are seperated with space(' ').
3. Keeping this as delimiter, storing the all unique tags only once in possible labels

```
for z in df_train['tags'].values:
    a=z.split(' ')
    possible_labels=list(set(possible_labels)|set(a))
possible_labels=sorted(possible_labels)
print(possible_labels)
```

Figure 6.3

1. Read each image from the local folder and convert them to pixel values.
2. Creating an array of 17 zeros, since dataset contains only 17 unique labels.
3. Now for each image, the corresponding image label's index is set.
4. For example first image contains tags as primary and haze. Taking first zero label array, consisting of 17 zeros is now made to 1 in primary index and haze index only. Rest of the index in array is left as 0.
5. Resize the image to 32*32 for simplification.
6. Store the result in the training_image array.

```
k=0;
for f,tags in zip(df_train['image_name'].values,df_train['tags']):
    images = cv2.imread('F:/Sample/sample_train_tif/{}.tif'.format(f))
    label_array = np.zeros(17)
    for t in tags.split(' '):
        label_array[possible_labels.index(t)] = 1
    print('Image ',k,':',label_array)
    k+=1
    train_img.append(cv2.resize(images, (32, 32)))
    train_lab.append(label_array)
```

Figure 6.4

Setting the data type for labels as unsigned 8 bit and setting the data type for labels as float 16 bit.

Dividing the testing and training data using split variable where 80% is made as training set and 20% testing set out of total data set.

```
print(train_lab)
train_lab = np.array(train_lab, np.uint8)
print('array')
print(train_lab)

print(train_img)
train_img = np.array(train_img, np.float16) / 255.# Normalise data to [0, 1] ra
print(train_img)

print(train_lab)

split = 1600
train_img, test_img, train_lab, test_lab = train_img[:split], train_img[split:]
print(test_img)
print(test_lab)
```

Figure 6.5

6.3 Constructing Model:

1. Constructing the model using sequential (as the data is sent as input to the next layer).
2. Adding layers to the network.

- **Layer 1**:-(Convolutional Layer)
  - ✓ Filters :32 (number output of filters in the convolution)
  - ✓ kernel_size :3*3(specifying the width and height of the 2D convolution window).
  - ✓ Input_shape :(32,32,3)Dimension of the input image and channels
  - ✓ Activation :relu (A(x) = max(0,x)) to restrict the value in a range that produces the output from the layer.
- **Layer 2**:- (Convolutional Layer)
  - ✓ Activation is relu in order to remove negative values.
  - ✓ 64 filters in this convolutional layer.
- **Layer 3**:- (Pooling Layer)
  - ✓ To reduce the dimension of the previous layer by taking a stride of 2*2 with maximum value.
  - ✓ Pool size: 2*2 window within which the maximum value is picked.
- **Layer 4**:- (Dropout Layer=0.25)
  - ✓ Dropout is used to eliminate over fitting condition. Dropout randomly drops the neurons from activating.
- **Layer 5**:- (Flatten)
  - ✓ To produce the one dimensional array of the values from the previous layer.
- **Layer 6**:- (Dense Layer)
  - ✓ With same functionality as the previous layer and with 128 filters and relu activation function.
- **Layer 7**:- (Drop out Layer =0.5)
  - ✓ Dropout is used to eliminate over fitting condition. Dropout randomly drops the neurons from activating.
- **Layer 8**:- (Output Layer)
  - ✓ Sigmoid function is used to extract the features for small change in X-axis.

- ✓ Dense Layer: It is fully connected layer that decides the class based on pattern that has matched so far and the pattern is received from the previous layer.
- ✓ The filters are 17 because the number of unique tags is 17. This layer should fire the possible labels and for this reason sigmoid function is used.

```
cnn_net = Sequential()
cnn_net.add(Conv2D(32, kernel_size=(3, 3),
                   activation='relu',
                   input_shape=(32, 32, 3)))

cnn_net.add(Conv2D(64, (3, 3), activation='relu'))
cnn_net.add(MaxPooling2D(pool_size=(2, 2)))
cnn_net.add(Dropout(0.25))
cnn_net.add(Flatten())
cnn_net.add(Dense(128, activation='relu'))
cnn_net.add(Dropout(0.5))
cnn_net.add(Dense(17, activation='sigmoid'))
```

Figure 6.6

6.4 Fitting and training model:

6.4.1 Fitting:

The model is made to compile with loss as binary cross entropy.

ADAM: Adam which is Adaptive Moment is used instead of gradient descent because ADAM has a learning rate for each weight whereas gradient descent has a single learning rate for all the weights.

BINARY_CROSSENTROPY: This is used because a label of 0 or 1(prediction) is given to each and very label.

Metrics: This parameter is used to judge the model and provides the information about the model in each epochs.

```
cnn_net.compile(loss='binary_crossentropy',
                optimizer='adam',
                metrics=['accuracy'])
```

Figure 6.7

6.4.2 Training:

1. Trains the model for a fixed number of epochs (iterations on a dataset).
2. X: train input images.
3. Y: Corresponding labels.
4. Batch_size: Number of samples taken to train per instance.
5. Epochs: the number of times to iterate over the training data arrays.
6. Verbose: To display the progress of epochs with bar.
7. Validation_data: Testing images and testing labels.

```
cnn_net.fit(train_img, train_lab,
          batch_size=128,
          epochs=4,
          verbose=1,
          validation_data=(test_img, test_lab))
```

Figure 6.8

6.4.3 Testing the model:

1. Predicting the model:
2. X: The testing images for prediction.
3. Batch_size: Number of samples taken to test per instance.

```
pred_test = cnn_net.predict(test_img, batch_size=128)
```

Figure 6.9

6.4.4 Threshold:

The result obtained from the prediction contains float values that is the probability of each tag to occur for that tested image. In order to find accuracy, the values should be in binary. For this reason, the value of threshold is set to 0.5. The value above 0.5 is set as 1 and values below the 0.5 is set as 0.

6.4.5 Accuracy:

The accuracy is determined by accuracy score of sklearn where the actual labels and predicted labels are passed that results the accuracy.

```
acc=accuracy_score(test_lab,pred_test)
print('Accuracy: ',acc*100,'%')
```

Figure 6.10

CHAPTER 7

SYSTEM TESTING

7.1 FUNCTIONALITY TESTING:

Test case 1:

Input:

Total data set: 10000

Training set : 8000

Testing set: 2000

```
split = 8000

cnn_net = Sequential()
cnn_net.add(Conv2D(32, kernel_size=(3, 3),
                   activation='relu',
                   input_shape=(32, 32, 3)))

cnn_net.add(Conv2D(64, (3, 3), activation='relu'))
cnn_net.add(MaxPooling2D(pool_size=(2, 2)))
cnn_net.add(Dropout(0.25))
cnn_net.add(Flatten())
cnn_net.add(Dense(128, activation='relu'))
cnn_net.add(Dropout(0.5))
cnn_net.add(Dense(17, activation='sigmoid'))

cnn_net.compile(loss='binary_crossentropy',
                optimizer='adam',
                metrics=['accuracy'])

cnn_net.fit(train_img, train_lab,
            batch_size=128,
            epochs=8,
            verbose=1,
            validation_data=(test_img, test_lab))


pred_test = cnn_net.predict(test_img, batch_size=128)

[[0 0 0 ..., 1 0 0]
 [0 0 0 ..., 0 0 0]
 [1 0 0 ..., 0 0 0]
 ...,
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 1]
 [0 0 0 ..., 0 0 0]] [[ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 ...,
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]]
Accuracy:  34.2828585707 %
```

Figure 7.1

Test case 2:

Total data set: 40479

Training set : 32384

Testing set: 8095

```
split = 32384   #     (80% of 40479)
```

```
cnn_net = Sequential()
cnn_net.add(Conv2D(32, kernel_size=(3, 3),
                   activation='relu',
                   input_shape=(32, 32, 3)))

cnn_net.add(Conv2D(32, (3, 3), activation='relu'))
cnn_net.add(MaxPooling2D(pool_size=(2, 2)))
cnn_net.add(Dropout(0.25))
cnn_net.add(Flatten())
cnn_net.add(Dense(128, activation='relu'))
cnn_net.add(Dropout(0.5))
cnn_net.add(Dense(17, activation='sigmoid'))

cnn_net.compile(loss='binary_crossentropy',
                optimizer='adam',
                metrics=['accuracy'])

cnn_net.fit(train_img, train_lab,
            batch_size=128,
            epochs=12,
            verbose=1,
            validation_data=(test_img, test_lab))


pred_test = cnn_net.predict(test_img, batch_size=128)
```

```
Accuracy:  43.6812847437 %
```

Figure 7.2

7.2 Accuracy for different test case:

S.NO	SAMPLE SIZE	BATCH SIZE	NO OF FILTERS	KERNAL SIZE	EPOCHS	THRESHOLD	ACCURACY
1.	200	128	64	3*3	4	0.5	25.49
2.	1000	128	64	3*3	4	0.5	32.33
3.	2000	128	64	3*3	4	0.5	33.91
4.	10000	128	64	3*3	15	0.5	32.33
5.	10000	128	64	3*3	8	0.5	35.22
6.	10000	128	64	3*3	5	0.5	33
7.	10000	128	64	3*3	9	0.5	34.2
8.	10000	128	64	3*3	8	0.5	34.28
9.	40479	128	64	3*3	8	0.5	37.41
10.	40479	128	64	3*3	8	0.3	33.42
11.	40479	128	64	3*3	8	0.7	31.27
12.	40479	128	64	3*3	12	0.5	41.24
13.	40479	128	64	5*5	10	0.5	43.01
14.	40479	128	64	5*5	12	0.5	41.55
15.	40479	128	32	3*3	12	0.5	43.68
16.	40479	128	32	3*3	8	0.5	34.28
17.	40479	128	32	3*3	10	0.5	36.47
18.	40479	128	64	3*3	10	0.5	42.24

Figure 7.3

7.3 Accuracy Graph with respect to other parameters:

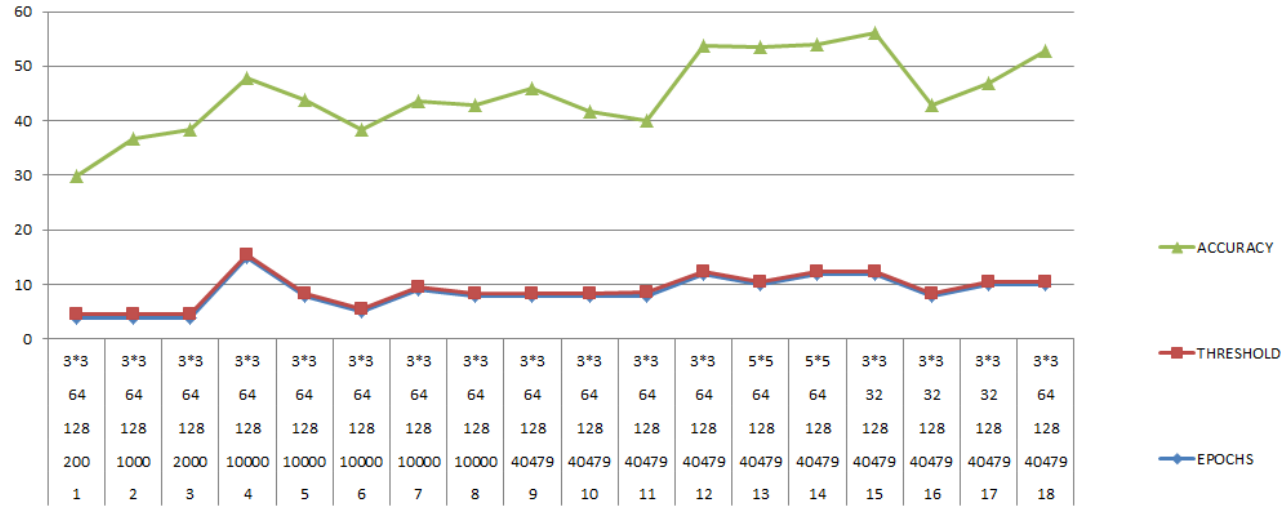

Figure 7.4

CHAPTER 8

CONCLUSION

The proposed application is very useful in identifying different terrains in the Amazon forest. It is also very helpful in identifying the deforested areas in the forest. This can help to identify the reasons for the deforestation. It is also noted that this application also helps in identifying the climatic conditions thus serving as a powerful tool for environmental issues.

The application can further be improved by changing the model structure by adding more layers, changing the layer size, training more data, adding data with variety of images appropriately.

REFERENCES

Installation Process: https://www.tensorflow.org/install/install_windows.

Dataset: https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/

Understanding problem: https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/

APPENDIX

CODE:

```
import numpy as np
import pandas as pd

#import cv2 for reading the image in tif format
import cv2
import keras as k
images = []
train_img = []
test_img = []
train_lab = []

#keras model is imported for constructing the model
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from sklearn.metrics import accuracy_score


#speciy the path to get the tags for corresponding image. The file is in csv(Comma Separated value) file
df_train = pd.read_csv('F:/Sample/sample1.csv')

#getting all possible labels once
#Declaring the array to store the labels.
```

```
possible_labels=[]


#for each row in the csv file, the tags are seperated with space(' ').
#Keeping this as delimiter, storing the all unique tags only once in possible labels


for z in df_train['tags'].values:
    a=z.split(' ')
    possible_labels=list(set(possible_labels)|set(a))
possible_labels=sorted(possible_labels)
print(possible_labels)


#Reading the images and converting to array of pixel vakues
#store the image pixel values in the array.
k=0;
for f,tags in zip(df_train['image_name'].values,df_train['tags']):
    images = cv2.imread('F:/Sample/sample_50/{}.tif'.format(f))


#Creating an array of 17 zeros, this our dataset contains only 17 unique labels.
    label_array = np.zeros(17)
#Now for each image, the corresponding image label's index is set.
#for example first image contains tags as primary and haze.
# taking first zero label array consisting of 17 zeros is now made to 1 in primary index and haze index only
#Rest of the index in array is left as 0.
    for t in tags.split(' '):
        label_array[possible_labels.index(t)] = 1
```

```
    print('Image ',k,':',label_array)
    k+=1

#images are resized to 32*32.
    train_img.append(cv2.resize(images, (32, 32)))
    train_lab.append(label_array)

#Setting the datatype for labels as unsigned 8 bit.
print(train_lab)
train_lab = np.array(train_lab, np.uint8)
print('array')
print(train_lab)

#Setting the datatype for labels as float 16 bit.
print(train_img)
train_img = np.array(train_img, np.float16) / 255.
print(train_img)

print(train_lab)

split = 32384  #(80% of total dataset)



#Dividing the testing and training data using split variable
train_img, test_img, train_lab, test_lab = train_img[:split], train_img[split:], train_lab[:split], train_lab[split:]
print(test_img)
print(test_lab)
```

#Constructing the model using sequential (as the data is sent as input to the next layer).

cnn_net = Sequential()

#Adding layers to the network

#Filters :32 (number output of filters in the convolution)

#kernel_size :3*3(specifying the width and height of the 2D convolution window).

#input_shape :(32,32,3)Dimension of the input image and channels

#Activation :relu (A(x) = max(0,x)) to restrict the value in a range that produces the output from the layer.

```
cnn_net.add(Conv2D(32, kernel_size=(3, 3),
                   activation='relu',
                   input_shape=(32, 32, 3)))
```

#Activation is relu in order to remove negative values.

cnn_net.add(Conv2D(64, (3, 3), activation='relu'))

#To reduce the dimension of the previous layer by taking a stride of 2*2 with maximum value.

#Pool_size: 2*2 window within which the maximum value is picked.

cnn_net.add(MaxPooling2D(pool_size=(2, 2)))

#Dropout is used to eliminate overfitting condition.Dropout randomly drops the neurons from activating

```
cnn_net.add(Dropout(0.25))

#To produce the one dimensional array of the values from the previous layer.
cnn_net.add(Flatten())
cnn_net.add(Dense(128, activation='relu'))
cnn_net.add(Dropout(0.5))

#sigmoid function is used to extract the features for small cahnge in X-axis.
#Dense Layer: It is fully connected layer that decides the class based on pattern that has matched so far and the pattern is received from the previous layer.
cnn_net.add(Dense(17, activation='sigmoid'))


#ADAM:Adam which is Adaptive Moment is used instead of gradient descent
#because ADAM has a learning rate for each weight
#whereas gradient descent has a single learning rate for all the weights.

#BINARY_CROSSENTROPY:This is used because a label of 0 or 1(prediction) is given to each and very label.

#Metrics: This parameter is used to judge the model and provides the information about the model in each epochs.

cnn_net.compile(loss='binary_crossentropy',
                optimizer='adam',
                metrics=['accuracy'])
```

```
#Fitting the model:
#Trains the model for a fixed number of epochs (iterations on a dataset).
#X: train input images
#Y: Corresponding labels
#Batch_size: Number of samples taken to train per instance.
#Epochs: the number of times to iterate over the training data arrays.
#Verbose: To display the progress of epochs with bar
#Validation_data: Testing images and testing labels


cnn_net.fit(train_img, train_lab,
        batch_size=128,
        epochs=4,
        verbose=1,
        validation_data=(test_img, test_lab))

#Predicting the model:
#X: The testing images for prediction.
#Batch_size:Number of samples taken to test per instance.


pred_test = cnn_net.predict(test_img, batch_size=128)
print(test_lab)
print(pred_test)

#therhold is set to convert the floating numbers to binary values.
```

```
threshold=0.5
pred_test[pred_test>=threshold]=1
pred_test[pred_test<threshold]=0
print(test_lab,pred_test)
#Accuracy is used to know how well the model is developed to predict the testing data.
acc=accuracy_score(test_lab,pred_test)
print('Accuracy: ',acc*100,'%')
```