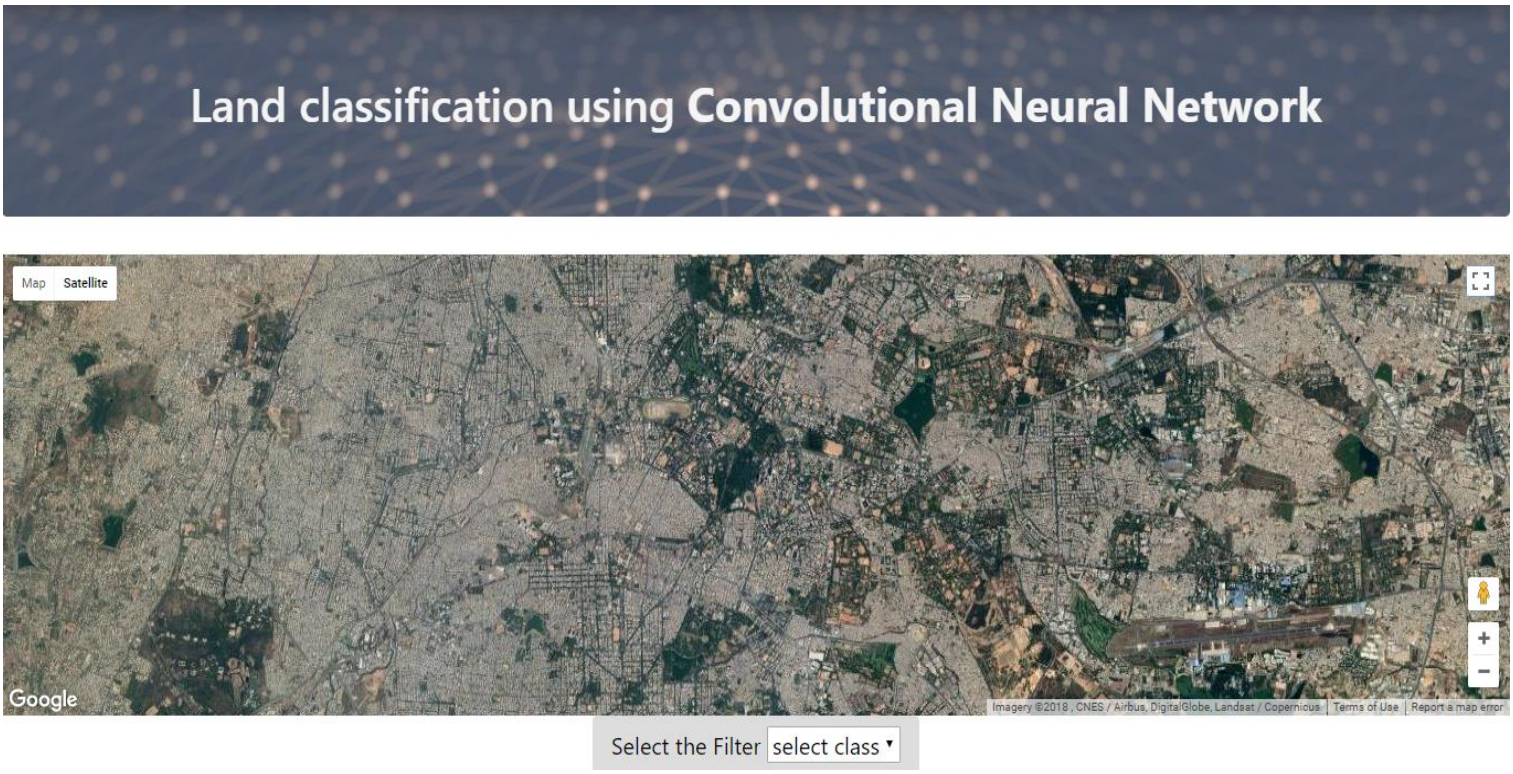


Deep Learning on Remotely Sensed Data

—Under the guidance of Prof. Uttam Kumar

Vivek Shukla (MT2017141)
Hatim Bohra (MT2017138)
Devesh Prasad (MT2017132)

This project was aimed at getting the basic land classification features from the maps data. Land classification has previously been done through various manual techniques at first. It took hundreds of day and a lot of manpower to generate the map of a small area. But Some areas are not trekkable by mankind therefore we had to find some ways to make to make .But this changed when we started sending satellite to monitor the earth. We can get very large remotely sensed data with the help of satellite. Machine learning helps in finding right patterns in that data efficiently.

We can use following architectures of deep learning to examine the remotely sensed data:

1. **Convolutional neural network**

A **convolutional neural network** (**CNN**, or **ConvNet**) is a class of deep,feed-forward artificial NN that has successfully been applied to analyzing visual imagery.

CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

They have applications in image and video recognition, recommender systems and natural language processing.

2. **Long short term memory**

Long short-term memory (**LSTM**) units (or blocks) are a building unit for layers of a recurrent neural network (RNN). A RNN composed of LSTM units is often called an LSTM network. A common LSTM unit is composed of a **cell**, an **input gate**, an **output gate** and a **forget gate**. The cell is responsible for "remembering" values over arbitrary time intervals; hence the word "memory" in LSTM. Each of the three *gates* can be thought of as a "conventional" artificial neuron, as in a multi-layer (or feedforward) neural network: that is, they compute an activation (using an activation function) of a weighted sum. Intuitively, they can be thought as *regulators* of the flow of values that goes through the connections of the LSTM; hence the denotation "gate". There are connections between these gates and the cell.

3. **AutoEncoders**

An autoencoder learns to compress data from the input layer into a short code, and then uncompress that code into something that closely matches the original data. This forces the autoencoder to engage in dimensionality reduction, for example by learning how to ignore noise. Some architectures use stacked sparse autoencoder layers for image recognition. The first autoencoder might learn to encode easy features like corners, the second to analyze the first layer output and then encode less local features like the tip of a nose, the third might encode a whole nose, etc., until the final autoencoder encodes the whole image into a code that matches (for example) the concept of "cat".An alternative use is as a generative model: for example, if a system is manually fed the codes it has learned for "cat" and "flying", it may attempt to generate an image of a flying cat, even if it has never seen a flying cat before.

Basically, we want to analyse the remotely sensed data and classify the land data into 6 classes namely Lake, residential buildings, Aeroplanes, chaparral, Forest and Overpasses. We have used the data set for the training from https://www.ucmerced.edu. We have used VGG16 model for extracting the image features. VGG16 is open source architecture for image analysis.

Here the VGG-16 model is described-

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					

We extract the relevant features from our data set then feed it into our model and finally classify it into one of the six classes.

Our model is defined as below-

1) **Flatten layer** (size=7x7x512)
2) **Dense Layer**(Fully Connected Layer) (size=1000)
3) **Dense Layer** (size=6)

The last layer calculates the probabilities according to "**Softmax**" the loss according to "**Categorical Cross Entropy**" and updates according to "**AdaDelta**".

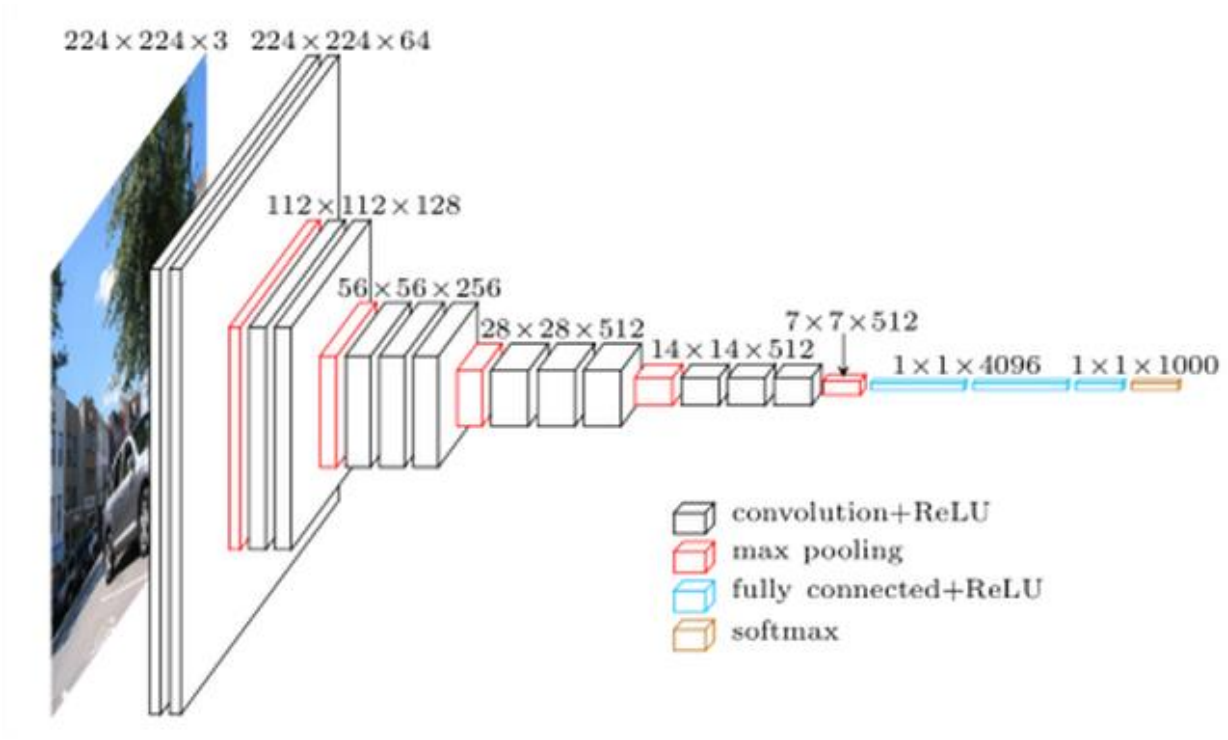
Flow of the Model-

The portion of the city is taken and divide these into grid in such a way that each sector contains image size at zoom level 19 or 20 and resize it into 224X224 resolution then for each sector in a grid it is feed into the model.

City

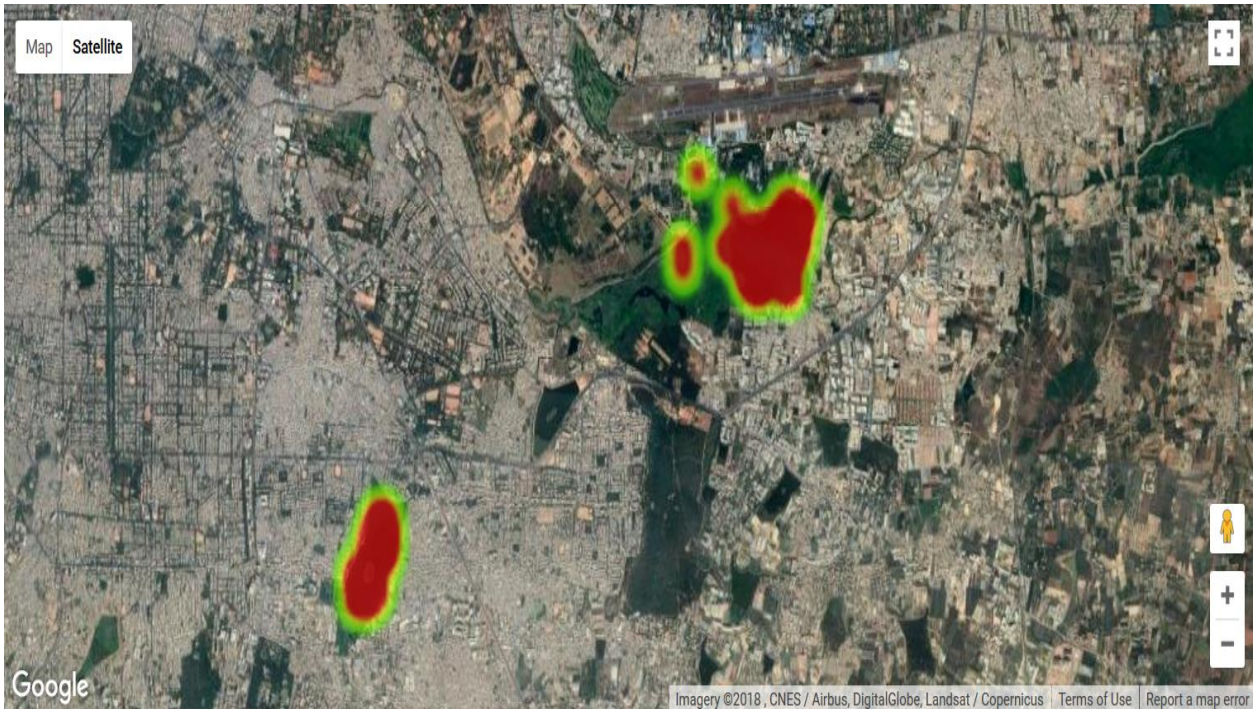
Portion of City:

The image of size (224x224x3) is passed through the VGG-16 network to get the features of size (7x7x512) which is then sent through our model to get the most probable class out of the six classes.



The coordinates of the most probable class are saved in the respective JSON file.

The coordinates are then retrieved by the WebApp at the runtime and they are sent to the Google Maps JavaScript API to generate the heatmap.



```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <link rel= "stylesheet" type="text/css" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css">

    <style>
       #map {

        height: 390px;
        width: 100%;
       }
        .jumbotron{
                    text-align:center;
                    color: #f2f2f2;
                    opacity: 0.8;
                    background-image: url("https://cdn-images-1.medium.com/max/1500/1*m2gDBT_nc-iE7R4AM3sHBQ.jpeg");
                }
                .container{
                    text-align:center;
                }

    </style>
    <title>Geoographic Information System</title>
  </head>
  <body >
      <div class="jumbotron " >
          <h1>Land classification using <strong>Convolutional Neural Network </strong></h1>
      </div>
    <div   id="map" > </div>
    <script>


var heatmap={};
var map={};
  function initMap(lat= 12.9716,long=77.5946,zoom=13) {

   var banglore = {lat: lat, lng:long};


         map = new google.maps.Map(document.getElementById('map'), {
          zoom: zoom,
          center: banglore,
          mapTypeId: 'satellite',


         });
```

HTML/CSS Code.

```
var data={};
$(document).ready(function(){

       $("#loadjson").change(function(){

       initMap(map.getCenter().lat(),map.getCenter().lng(),map.getZoom());
        $.ajax({url: "/loadjson?id="+this.value, success: function(data){


          len=Object.keys(data["lat"]).length;
          heatMapData=[];
         for(i=0;i<parseInt(len);i++){
          var s=""+i+"";
          heatMapData.push( {location: new google.maps.LatLng(data['lat'][s], data["long"][s]), weight: 1});

         }


        heatmap = new google.maps.visualization.HeatmapLayer({
        data:heatMapData,
        radius: 20

        });

        heatmap.setMap(map);

         }, error: function(xhr){
        alert('Request Status: ' + xhr.status + ' Status Text: ' + xhr.statusText + ' ' + xhr.responseText);
    }});

    });
});
</script>

    <script
    src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDLO-Q_SGZ7XWB99B3P_Rq_uBlc96Qu9BA&libraries=visualization&callback=initMap">
    async defer
    </script>
  <div class="container"  >
      <button class=  "btn btn-lg primary" > Select the Filter
        <select id="loadjson"  >
         <h3>Select the filter</h3>
          <option >select class</option>
           <option value="0">lake</option>
           <option value="1">airplane</option>
           <option value="2">buildings</option>
           <option value="3">chaparral</option>
```

HTML/CSS code.

Python Code as follows:

In [1]:
```
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
import numpy as np
from keras.models import Model,Sequential
from keras.layers import Dense,Activation,Dropout,Flatten
import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from keras.models import model_from_json
from io import BytesIO

from PIL import Image
from urllib import request
import pandas as pd
import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
```

In [2]:
```
lb=list(os.walk("../../datasets/UCMerced_LandUse/Images1"))[0][1]
base_model=VGG16(weights='imagenet',include_top=False,input_shape=(224,224,3))
base_model.summary()
```

In [ ]:
```
#data load

n_classes=6
a = np.arange(n_classes)
b = np.zeros((n_classes, n_classes))
b[np.arange(n_classes), a] = 1
print(b)
features=[]
label=[]

count=0
for i in lb:
    #print(i)
    lbf=list(os.walk("../../datasets/UCMerced_LandUse/Images1/"+i))[0][2]
    for j in lbf:
        print(i," ",j)
        img_path='../../datasets/UCMerced_LandUse/Images1/'+i+'/'+j
        tmp=[]
        img=image.load_img(img_path,target_size=(224,224))
        x=image.img_to_array(img)
        x=np.expand_dims(x,axis=0)
        x=preprocess_input(x)
        x=base_model.predict(x)
        #print(x.shape)
        features.append(x.flatten())

        label.append(b[count])
    count+=1

features=np.array(features)
label=np.array(label)
print(features.shape,label.shape)
```

In [ ]:
```
# shuffle random
print(data[4][1])
np.random.shuffle(data)

print(data[4][1])

data[0][0].shape
```

In [ ]:
```
# extracting features
size=7
activations=np.zeros((n_classes*100,size*size*512))
label=np.zeros((n_classes*100,n_classes))
for i in range(n_classes*100):
    print(i)
    print(data[i,0].shape)
    activations[i]=data[i,0].tolist()
    label[i]=data[i,1].tolist()
activations

#save features in file
np.save('../../datasets/UCMerced_LandUse/vgg_activation.npy',activations)
np.save('../../datasets/UCMerced_LandUse/label.npy',label)
```

In [3]:
```
#Load features from file
features=np.load('../../datasets/UCMerced_LandUse/vgg_features.npy')
label=np.load('../../datasets/UCMerced_LandUse/labels.npy')
print(features.shape,label.shape)
```

In [ ]:
```
#define model

model=Sequential()

model.add(Dense(1000,input_shape=(features.shape[1],),name='fc',activation='sigmoid'))
model.add(Dropout(0.5))
model.add(Dense(label.shape[1],activation='softmax',name='prediction'))
model.compile(optimizer='adadelta',loss='categorical_crossentropy',metrics=['accuracy'])
model.fit(x=features,y=label,batch_size=100,epochs=4)
model.predict(features[0:1])
```

In [7]:
```

model1=Sequential()
model1.add(base_model)
model1.add(Flatten())
model1.add(model)
```

In [55]:
```
lat=[]
long=[]
```

In [ ]:
```
# select particular area

startlat=13111204
endlat=13119804
startlong=77699373
endlong=77713491
inc=500
steps=(int)(((int)((endlat-startlat)/inc)*(int)((endlong-startlong)/inc))/5)
print(steps)
```

```
# iterate over all area and classify image

def generator(batch_size):

    global startlat,endlat,startlong,endlong,inc
    m=startlat
    count=0
    t=0
    flag=False
    while(m<endlat):
        n=startlong
        while(n<endlong):
            print(str(m)+" "+str(n))
            center= ["{:.6f}".format(m/np.power(10,6)),"{:.6f}".format(n/np.power(10,6))]
            print(center[0]," ",center[1])
            while(True):
                try:

                    url = "http://maps.googleapis.com/maps/api/staticmap?maptype=satellite&key=AIzaSyCBrDFq12VimTpdCWQSBx82JC7G64

                    buffer = BytesIO(request.urlopen(url).read())
                    img = Image.open(buffer)
                    break
                except Exception as e:
                    print(e)
                    continue
                    #img.show()
            x=np.array(img.resize((224,224)).convert('RGB')).astype('float64')
            x=np.expand_dims(x,axis=0)
            x=preprocess_input(x)


            print(x.shape)


            if flag:
                images=np.concatenate((images,x),axis=0)

            else:
                flag=True
                images=x
```

```
            count+=1
            t+=1
            #print("count=",count," t=",t)
            if(count==batch_size):
                count=0
                print(t)
                print(images.shape)
                flag=False

                yield images



            n+=inc
        m+=inc

y=model1.predict_generator(generator(5),steps=steps)

m=startlat
x=0
while(m<endlat):
    n=startlong
    while(n<endlong):
        print(str(m)+" "+str(n))
        center= ["{:.6f}".format(m/np.power(10,6)),"{:.6f}".format(n/np.power(10,6))]

        k=np.argmax(y[x])
        x+=1

        print(k)

        print(center)

        if(k==3):
            lat.append((float)(center[0]))
            long.append((float)(center[1]))
        n+=inc
    m+=inc

# save coordinate to json files

data=pd.DataFrame({'lat':lat,'long':long})
data.to_json("GIS/json files/obj3.JSON")
```