Evaluate Greening Project in Northwest China using CNN and ResNet CNN

Zhuang Qian, Yichen Lei, Xingxing Shu

1. Introduction

The objective of this study is to use CNN and ResNet CNN analyzing the achievements of the project of conversion of dessert or barelands back to forests in northwest China. The greening project, which was launched in 1999, aims to stop the cultivation of farmland with severe soil erosion in a planned way and afforest and plant grass in accordance with local conditions to restore vegetation (Zhihu,2020). Shaanxi province is one of the provinces with serious desertification and desertification in China. It has a total area of 42,200 square kilometers of the Maowusu desert, which is an important target for project management (Zhihu,2020). It is reported that the green coverage of the Maowusu desert has increased from 100,000 acres and 0.9 percent in the mid-20th century to 3.5 million acres and 33 percent now, successfully stopping the southward trend of desertification and achieving the reverse movement of vegetation areas of more than 400 kilometers (Figure 1a)(Zhihu,2020). In addition, NASA reported in 2019 that since 2000, there is a 5 percent increase of the green in the world, which is the size of the Amazon Rainforest, and 25 percent of the increase has come from China (Figure 1b) (Nasa, 2019). In order to evaluate the results of green projects in China, remote sensing methods are the most efficient for such a large-scale space project. Through the classification of land use in satellite images, the proportion of vegetation, bare land, water and so on in the area can be quickly obtained, so as to make a quantitative conclusion.

Figure 1. (a) One Road boundary of the greening project(Zhihu, 2020) (b) Greening Increase Image of China (Nasa, 2019)

In this study, CNN and ResNet CNN will be used to realize automatic classification of satellite images of land use in northwest China, make statistics of land use in satellite images of a wide range of space, and evaluate the project of conversion of dessert or bare lands back to forests in China.

2. Data Description

The model training dataset used in this study was SAT-6 provided by Kaggle, whose original images were taken from the national agricultural imaging program (NAIP) dataset, consisting of 4 bands – red, green, blue and Near Infrared (NIR)(Chris, 2018). In order to maintain the high variance inherent in the entire NAIP dataset, the image patches were sampled from a multitude of scenes (a total of 1500 image tiles) covering different landscapes like rural areas, urban areas, densely forested, mountainous terrain, small to large water bodies, agricultural areas, etc. covering the whole state of California (Basu et al., 2015). Eventually, six classes were used to describe land cover categories, including barren land, trees, grassland, roads, buildings, and water bodies.

Then, the data of the model application is cropped from the sentinal2 satellite imagery in 2016, covering most areas of northwestern China. Both training data and application data are divided into contextual windows that have a maximum dimension of 28*28. This is because a window that is too large may be mixed with multiple land types, resulting in confusing classification, while a window that is too small is difficult to achieve accurately distinguish due to image resolution issues.

The exploratory analysis of datasets will be presented in the result chapter.

3. Methodology

The steps of this study are mainly divided into model construction and practical application. In the first step, both CNN and ResNet CNN will be trained based on the labeled SAT-6 data. After comparison and selection, the better model will be applied to the actual image of Sentinal2 to obtain the land use situation in northwest China. Using satellite imagery for land use classification is a classic challenge. It involves several terabytes of data and has undergone major changes due to data collection, preprocessing, and filtering conditions. Traditional supervised learning methods like Random Forests do not generalize well for such a large-scale learning problem (Ai et al., 2014). Literature (Mnih & Hinton, 2010) proposes a classification algorithm that uses deep neural networks to detect roads in aerial images. The problem of detecting various land cover classes in general is a difficult problem considering the significantly higher intra-class variability in land cover types such as trees, grasslands, barren lands, water bodies, etc. as compared to that of roads (Wikipedia, 2020). As for deep learning, CNN is a class of deep neural networks, having shared-weights architectures and translation invariance characteristics (Wikipedia, 2020). They are most often used to analyze visual images, so using CNN in satellite imagery is a good choice for land use classification. However, general CNN faces the vanishing gradient problem when training very deep neural networks. Increasing the network the depth does not work by simply stacking the layers together, as the gradient is backpropagated to earlier the layers, repeated multiplication may make the gradient extremely small. As a result, as the network goes deeper, its performance gets saturated or even starts degrading rapidly. Residual network (ResNet) is an advanced architecture used as the backbone of many computer vision tasks (Priya, 2019). The fundamental breakthrough of ResNet is that it enables us to successfully train ultra-deep neural networks with more than 150 layers, which helps to process massive amounts of satellite data (Priya, 2019). So, we choose CNN as the first choice for image classification. Generally speaking, CNN is adequate for image

classification. Second, in order to understand the effects of deeper neural networks in experiments, ResNet CNN was implemented to see if our satellite data required more sophisticated neural networks. Therefore, both models are implemented in this experiment. Specifically, CNN is a regularized version of the multilayer perceptron. Multilayer perceptrons usually represent fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer (Wikipedia, 2020). CNN consists of an input layer and an output layer, as well as multiple hidden layers. CNN's hidden layer usually consists of a series of convolutional layers convolved with multiplication or other dot products (Wikipedia, 2020). The activation function is usually the RELU layer, followed by additional convolutions such as the pooling layer, the fully connected layer, and the normalization layer, known as the hidden layer, because their inputs and outputs are shielded by the activation function and the final convolution (Wikipedia, 2020).

ResNet CNN is an even more advanced architecture. ResNet first introduced the concept of skip connection (Priya, 2019). Figure 2 illustrates skip connection. The figure on the left is stacking convolution layers together one after the other. On the right we still stack convolution layers as before but we now also add the original input to the output of the convolution block. The reasons of using skip connects are that they mitigate the problem of vanishing gradient by allowing this alternate shortcut path for gradient to flow through, and let the identity function ensure that the higher layer will perform at least as well as the lower layer (Priya, 2019).

Figure 2.Skip Connection Image from DeepLearning.AI (Priya, 2019)

In this study, the ResNet model used is ResNet-50 model and its architecture is shown below. The ResNet-50 model consists of 5 stages, each stage has a convolution and identification block (Priya, 2019). Each convolutional block has 3 convolutional layers, and each identification block also has 3 convolutional layers (Priya, 2019). ResNet-50 has over 23 million trainable parameters, which is enough for our satellite data.

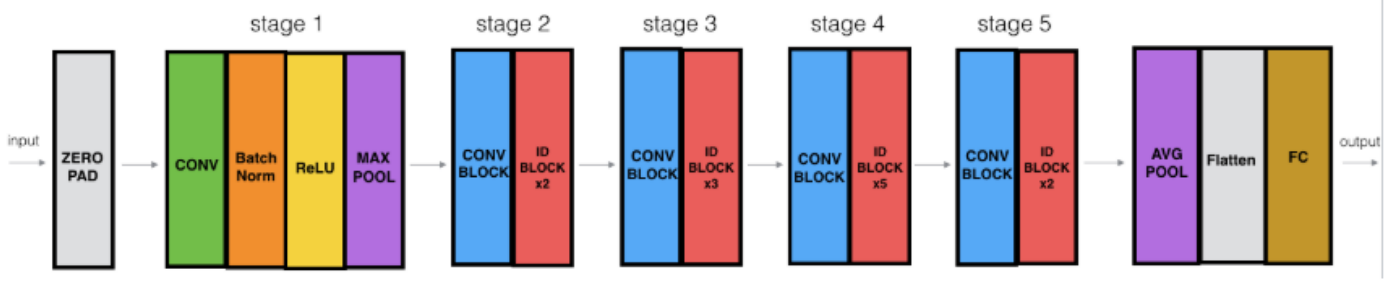


Figure 3.ResNet-50 Model (Priya, 2019)

4.Results

4.1 Exploratory Analysis

In [42]:

```
## Importing the libraries
import numpy as np
import time
import imageio
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import plotly.express as px
from skimage import color
from skimage.color import rgb2gray
import warnings
warnings.filterwarnings("ignore")
```

In [9]:

```
from sklearn.preprocessing import StandardScaler
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization, Activation
from keras.callbacks import EarlyStopping, ModelCheckpoint, TensorBoard
from time import time
from mlxtend.evaluate import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.layers import Dense, Flatten, Activation
from skimage.transform import resize
from keras.applications import resnet50
from keras import optimizers
```

```
Using TensorFlow backend.
```

Read Data

> Because the SAT-4 data has been stored in the csv, we can not read the image data directly. The way to import is read the csv file, then merge the df.value and set header = none.

In [2]:

```
# Input data files are available in the "../input/" directory.
import os
for dirname, _, filenames in os.walk('./deepsat-sat6/'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output.
```

```
./deepsat-sat6/sat-6-full.mat
./deepsat-sat6/sat6annotations.csv
./deepsat-sat6/X_test_sat6.csv
./deepsat-sat6/X_train_sat6.csv
./deepsat-sat6/y_test_sat6.csv
./deepsat-sat6/y_train_sat6.csv
```

In [3]:

```
# set path
train_data_path="./deepsat-sat6/X_train_sat6.csv"
train_label_path="./deepsat-sat6/y_train_sat6.csv"
test_data_path="./deepsat-sat6/X_test_sat6.csv"
test_lable_path="./deepsat-sat6/y_test_sat6.csv"
```

In [4]:

```
def data_read(data_path, nrows):
    data=pd.read_csv(data_path, header=None, nrows=nrows)
    data=data.values ## converting the data into numpy array
    return data
```

In [5]:

```
##Read training data
train_data=data_read(train_data_path, nrows=500)
print("Train data shape:" + str(train_data.shape))

##Read training data labels
train_data_label=data_read(train_label_path,nrows=500)
print("Train data label shape:" + str(train_data_label.shape))
print()

##Read test data
test_data=data_read(test_data_path, nrows=100)
print("Test data shape:" + str(test_data.shape))


##Read test data labels
test_data_label=data_read(test_lable_path,nrows=100)
print("Test data label shape:" + str(test_data_label.shape))

```

```
Train data shape:(500, 3136)
Train data label shape:(500, 6)

Test data shape:(100, 3136)
Test data label shape:(100, 6)
```

Dataset Visualization

In SAT-4 data, 6 classes land use is labeled from 0 to 5, as the sequence of building, barren land, trees, grassland, road, and water. In order to intuitively understand the labels of SAT-4 data and have a visual concept of the images in the small windows, we selected 16 images from the training set and the test set respectively for visualization. As shown, usually white buildings, barren land is yellow color, trees is green, gressland celadon, road is gray, the water is blue. Although pitures were a little fuzzy due to the resolution of satellite images, each small picture had typical land use characteristics, and the size of 28*28 we selected was good for only one type of land in each.

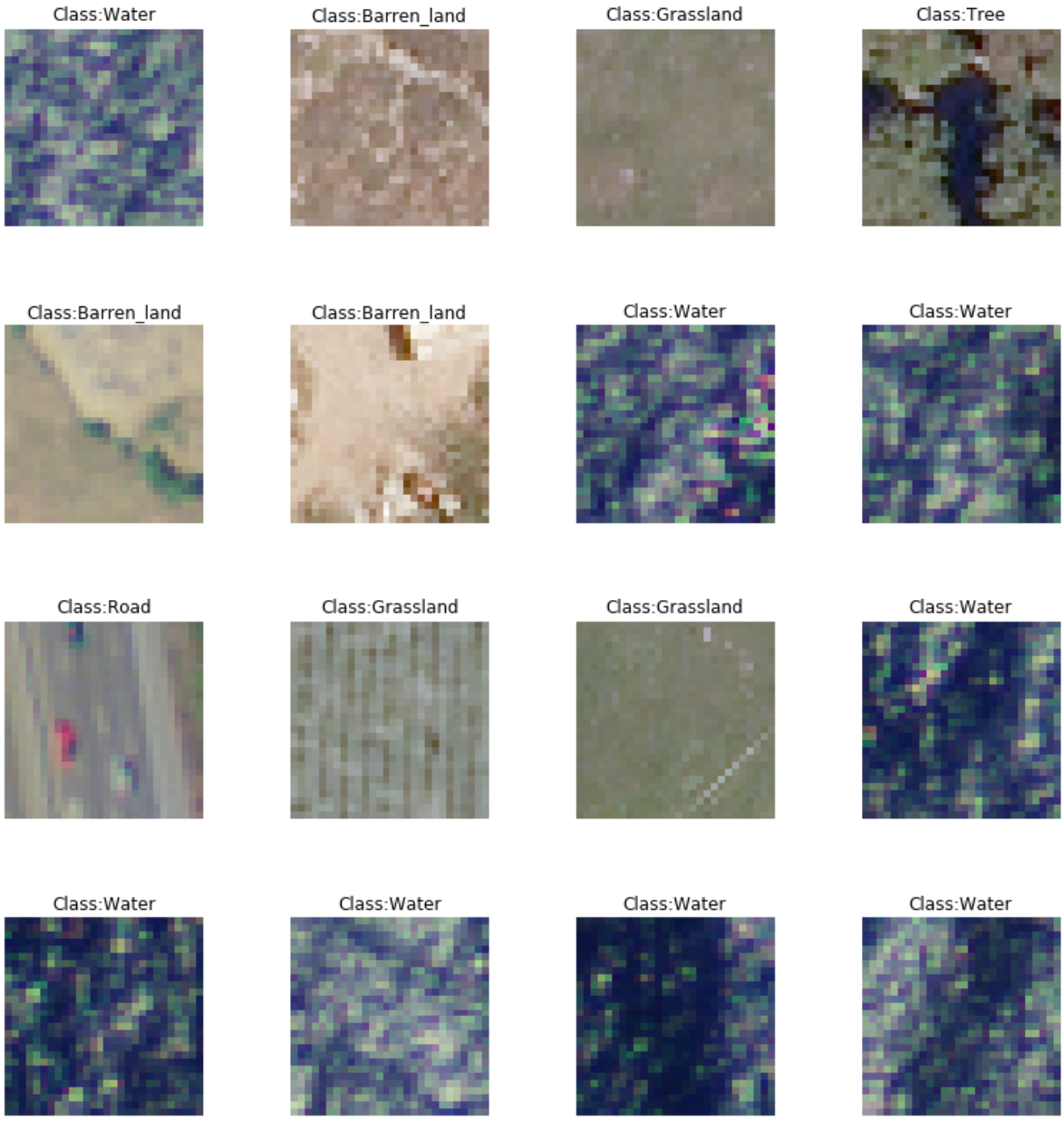
In [6]:

```
#label converter
# [1,0,0,0,0,0]=building
# [0,1,0,0,0,0]=barren_land
# [0,0,1,0,0,0]=trees
# [0,0,0,1,0,0]=grassland
# [0,0,0,0,1,0]=road
# [0,0,0,0,0,1]=water


def label_conv(label_arr):
    labels=[]
    for i in range(len(label_arr)):

        if (label_arr[i]==[1,0,0,0,0,0]).all():
            labels.append("Building")

        elif (label_arr[i]==[0,1,0,0,0,0]).all():
            labels.append("Barren_land")

        elif (label_arr[i]==[0,0,1,0,0,0]).all():
            labels.append("Tree")

        elif (label_arr[i]==[0,0,0,1,0,0]).all():
            labels.append("Grassland")

        elif (label_arr[i]==[0,0,0,0,1,0]).all():
            labels.append("Road")

        else:
            labels.append("Water")
    return labels
train_label_convert=label_conv(train_data_label)##train label conveter
test_label_convert=label_conv(test_data_label) ##test label converter


def data_visualization(data, label, n):
    ##data: training or test data
    ##lable: training or test labels
    ## n: number of data point, it should be less than or equal to no. of data points
    fig = plt.figure(figsize=(14, 14))
    ax = []  # ax enables access to manipulate each of subplots
    rows, columns=4,4
    for i in range(columns*rows):
        index=np.random.randint(1,n)
        img= data[index].reshape([28,28,4])[:,:,:3] ##reshape input data to rgb image
        ax.append( fig.add_subplot(rows, columns, i+1) ) # create subplot and append to
        ax[-1].set_title("Class:"+str(label[index]))  # set class
        plt.axis("off")
        plt.imshow(img)

    plt.subplots_adjust(wspace=0.1,hspace=0.5)
    plt.show()  # finally, render the plot
```

Training data visualization

In [41]:

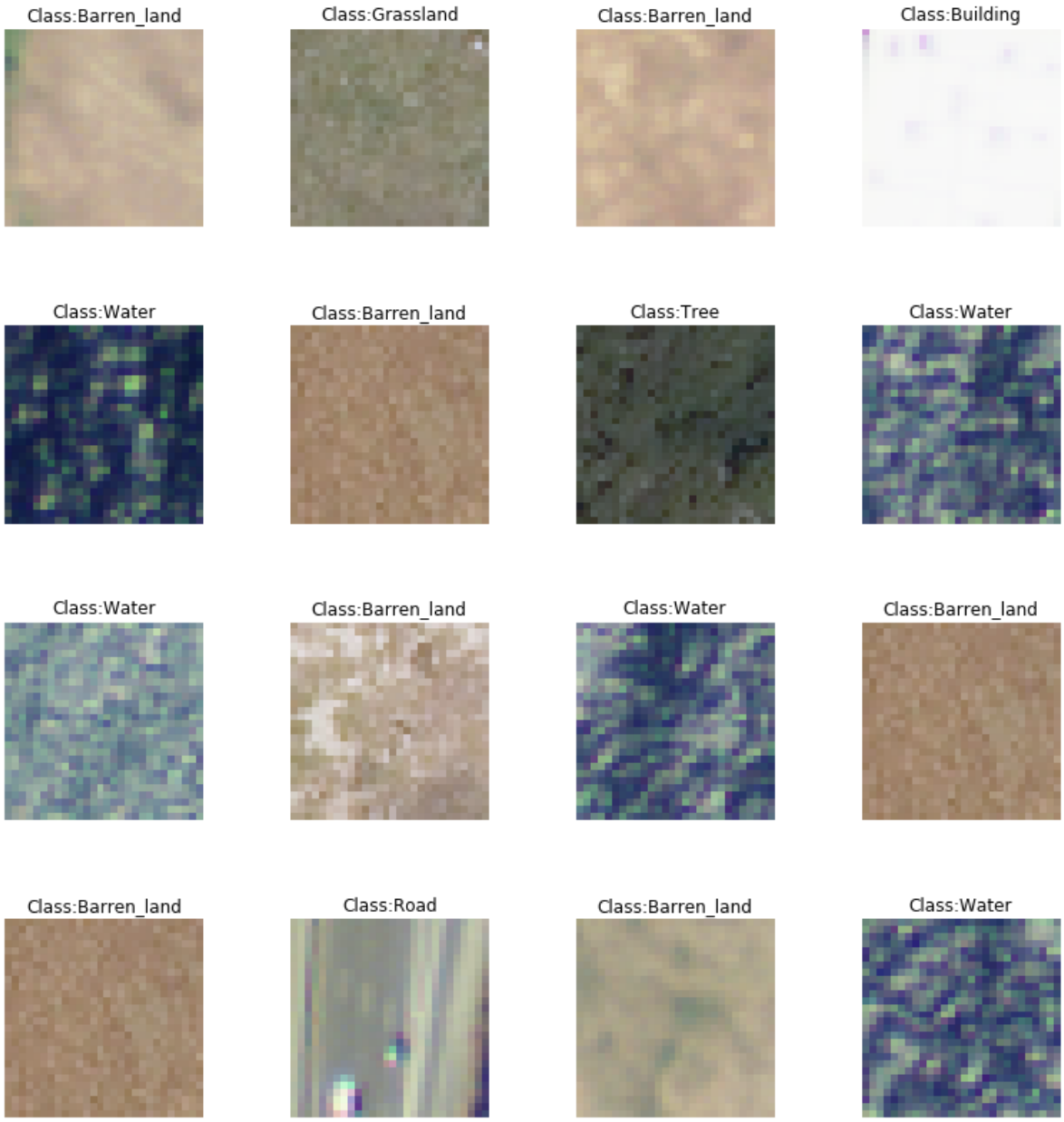
```
1 data_visualization(train_data,train_label_convert, n=500)
```



Test data visualization

In [40]:

```
1 data_visualization(test_data,test_label_convert, n=100)
```



4.2 CNN and ResNet CNN Models

Reading training data feature and training data label

> The images we used is 10000 of training set and testing set respectively.

In [10]:

```
# Method to load data and images
def load_data_and_labels(data, labels):
    data_df = pd.read_csv(data, header=None, nrows=10000)
    X = data_df.values.reshape((-1,28,28,4)).clip(0,255).astype(np.uint8)
    labels_df = pd.read_csv(labels, header=None, nrows=10000)
    Y = labels_df.values.getfield(dtype=np.int8)
    return X, Y

x_train, y_train = load_data_and_labels(data='./deepsat-sat6/X_train_sat6.csv',
                                        labels='./deepsat-sat6/y_train_sat6.csv')
x_test, y_test = load_data_and_labels(data='./deepsat-sat6/X_test_sat6.csv',
                                      labels='./deepsat-sat6/y_test_sat6.csv')

print(pd.read_csv('./deepsat-sat6/sat6annotations.csv', header=None))

# Print shape of all training, testing data and labels
# Labels are already loaded in one-hot encoded format
print(x_train.shape) # (10000, 28, 28, 4)
print(y_train.shape) # (10000, 6)
print(x_test.shape)  # (10000, 28, 28, 4)
print(y_test.shape)  # (10000, 6)
```

```
             0  1  2  3  4  5  6
0     building  1  0  0  0  0  0
1  barren_land  0  1  0  0  0  0
2        trees  0  0  1  0  0  0
3    grassland  0  0  0  1  0  0
4         road  0  0  0  0  1  0
5        water  0  0  0  0  0  1
(10000, 28, 28, 4)
(10000, 6)
(10000, 28, 28, 4)
(10000, 6)
```

CNN Model Building

In [88]:

```
# construct CNN
model = Sequential()

model.add(Conv2D(16, (3,3), activation='relu', input_shape=(28,28,4)))
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(6, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [12]:

```
# view summary of CNN
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 16)        592
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 32)        9248
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 64)          18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 4, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               131200
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 774
=================================================================
Total params: 164,950
Trainable params: 164,950
Non-trainable params: 0
_________________________________________________________________
```

CNN Model Training

> We adopted the training on CNN of 15 epochs. Finally, the accuracy of the test set is stable at 0.94, and there is no overfitting.

In [13]:

```
# fit CNN with 15 epochs
model.fit(x_train, y_train,
          batch_size=200,
          epochs=15,
          verbose=1,
          validation_data=(x_test, y_test))
```

```
WARNING:tensorflow:From C:\Users\Yichen\Anaconda3\envs\musa-620\lib\site-pac
kages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.

Train on 10000 samples, validate on 10000 samples
Epoch 1/15
10000/10000 [==============================] - 2s 244us/step - loss: 6.7470
- accuracy: 0.4600 - val_loss: 0.8989 - val_accuracy: 0.5628
Epoch 2/15
10000/10000 [==============================] - 1s 127us/step - loss: 0.8626
- accuracy: 0.7015 - val_loss: 0.7775 - val_accuracy: 0.7639
Epoch 3/15
10000/10000 [==============================] - 1s 125us/step - loss: 0.7546
- accuracy: 0.7800 - val_loss: 0.6573 - val_accuracy: 0.8617
Epoch 4/15
10000/10000 [==============================] - 1s 133us/step - loss: 0.5999
- accuracy: 0.8494 - val_loss: 0.4585 - val_accuracy: 0.8806
Epoch 5/15
10000/10000 [==============================] - 1s 129us/step - loss: 0.4278
- accuracy: 0.8830 - val_loss: 0.2603 - val_accuracy: 0.9179
Epoch 6/15
10000/10000 [==============================] - 1s 130us/step - loss: 0.2992
- accuracy: 0.9043 - val_loss: 0.2116 - val_accuracy: 0.9210
Epoch 7/15
10000/10000 [==============================] - 1s 130us/step - loss: 0.2495
- accuracy: 0.9131 - val_loss: 0.2034 - val_accuracy: 0.9168
Epoch 8/15
10000/10000 [==============================] - 1s 131us/step - loss: 0.2399
- accuracy: 0.9154 - val_loss: 0.1962 - val_accuracy: 0.9193
Epoch 9/15
10000/10000 [==============================] - 1s 134us/step - loss: 0.2263
- accuracy: 0.9153 - val_loss: 0.2620 - val_accuracy: 0.8960
Epoch 10/15
10000/10000 [==============================] - 1s 130us/step - loss: 0.2312
- accuracy: 0.9143 - val_loss: 0.1924 - val_accuracy: 0.9270
Epoch 11/15
10000/10000 [==============================] - 1s 132us/step - loss: 0.1961
- accuracy: 0.9260 - val_loss: 0.1904 - val_accuracy: 0.9314
Epoch 12/15
10000/10000 [==============================] - 1s 133us/step - loss: 0.1945
- accuracy: 0.9288 - val_loss: 0.1572 - val_accuracy: 0.9406
Epoch 13/15
10000/10000 [==============================] - 1s 129us/step - loss: 0.1806
- accuracy: 0.9308 - val_loss: 0.1670 - val_accuracy: 0.9361
Epoch 14/15
10000/10000 [==============================] - 1s 134us/step - loss: 0.1710
- accuracy: 0.9375 - val_loss: 0.1388 - val_accuracy: 0.9437
Epoch 15/15
10000/10000 [==============================] - 1s 131us/step - loss: 0.1629
- accuracy: 0.9391 - val_loss: 0.1266 - val_accuracy: 0.9478
```

Out[13]:

```
<keras.callbacks.callbacks.History at 0x157c1a45438>
```

CNN Confusion Matrix

> It can be seen from the confusion matrix that the overall results of six types of land use are acceptable. Among them, road and building, grassland and barren land classification effect is a bit poor. This is predictable, because both the road and the building are artificial buildings, which use similar materials, resulting in similar reflection of each wave band, which is easy to be misclassified. At the same time, the barren land and grassland is a gradual change in real, the relationship between division mainly depends on how much the amount of grass, and it is difficult to give an objective shedshod.

In [14]:

```
#Label converter
# [1,0,0,0,0,0]=building
# [0,1,0,0,0]=barren_land
# [0,0,1,0,0,0]=tree
# [0,0,0,1,0,0]=grassland
# [0,0,0,0,1,0]=road
# [0,0,0,0,0,1]=water


##Building confusion matrix

y_pred=model.predict_classes(x_test)
y_true=np.argmax(y_test, axis=1)
cm=confusion_matrix(y_target=y_true, y_predicted=y_pred)

df_cm = pd.DataFrame(cm, index = [i for i in ["Building","Barren_land","Tree","Grasslar
                  columns = [i for i in ["Building","Barren_land","Tree","Grassland","F
plt.figure(figsize = (10,8))
sns.heatmap(df_cm, annot=True)
```

Out[14]:

<matplotlib.axes._subplots.AxesSubplot at 0x157bfbdf2e8>

CNN Precision and Recall calculation for multiclass classification

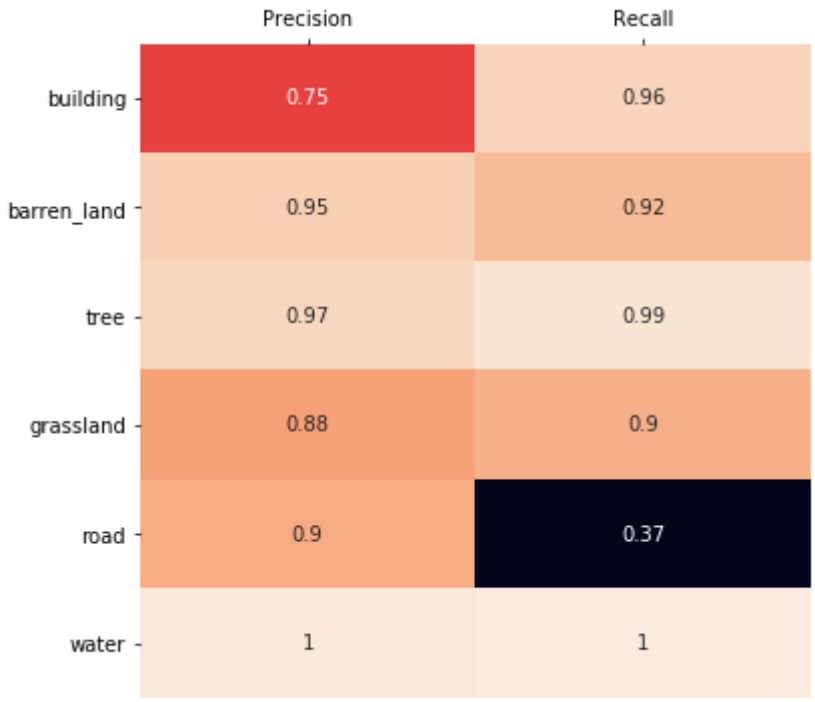
> Precision is the fraction of relevant instances among the retrieved instances, while recall is the fraction of the total amount of relevant instances that were actually retrieved.
> Overall, CNN achieved good results, but road's recall was only 0.37, which means that most road images were not correctly identified. Building precision is 0.75, which means that a small number of images classified as Building are actually wrong.

In [15]:

```
# define function to calculate precision and recall of each class
def precision(label, confusion_matrix):
    col = confusion_matrix[:, label]
    return confusion_matrix[label, label] / col.sum()

def recall(label, confusion_matrix):
    row = confusion_matrix[label, :]
    return confusion_matrix[label, label] / row.sum()

def precision_macro_average(confusion_matrix):
    rows, columns = confusion_matrix.shape
    sum_of_precisions = 0
    for label in range(rows):
        sum_of_precisions += precision(label, confusion_matrix)
    return sum_of_precisions / rows

def recall_macro_average(confusion_matrix):
    rows, columns = confusion_matrix.shape
    sum_of_recalls = 0
    for label in range(columns):
        sum_of_recalls += recall(label, confusion_matrix)
```

In [16]:

```
# loop through six classes to get their precison and recall
dic={}
precision_=[]
recall_=[]
precision_macro_average_=[]
for label in range(6):
    precision_.append(precision(label, cm))
    recall_.append(recall(label, cm))

dic["Precision"]= precision_
dic["Recall"]= recall_

plt.figure(figsize=(6,6))
ax=sns.heatmap(pd.DataFrame(dic, index=["building","barren_land","tree","grassland","ro
plt.yticks(rotation=0)
ax.xaxis.tick_top() # x axis on top

plt.show()
```



RestNet CNN Model Building

In order to use the ResNet, some data cleaning work has to ben done. However, the window size is 28 * 28 * 4, and the minimum picture limit of RestNet50 is 32 * 32 * 3. Therefore, we first use ‘for’ loop to remove the fourth channel data, and then convert the size of the picture through PIL, finally, stretching all the pictures to a size of 64 * 64.

In [84]:

```
# Method to load data and images
def load_data_and_labels(data, labels):
    data_df = pd.read_csv(data, header=None, nrows=10000)
    X = data_df.values
    list_x = []
    for i in range(0,len(X)):
        for j in range(0,len(X[i])):
            if(j%4 != 3):
                list_x.append(X[i][j])
    list_x = np.asarray(list_x)
    list_x = list_x.reshape((-1,28,28,3)).clip(0,255).astype(np.uint8)
    X = []
    for i in range(0,len(list_x)):
        bottle_resized = resize(list_x[i], (64, 64))
        X.append(bottle_resized)
    X = np.asarray(X)

    labels_df = pd.read_csv(labels, header=None, nrows=10000)
    Y = labels_df.values.getfield(dtype=np.int8)
    return X, Y

x_train, y_train = load_data_and_labels(data='./deepsat-sat6/X_train_sat6.csv',
                                        labels='./deepsat-sat6/y_train_sat6.csv')
x_test, y_test = load_data_and_labels(data='./deepsat-sat6/X_test_sat6.csv',
                                      labels='./deepsat-sat6/y_test_sat6.csv')

print(pd.read_csv('./deepsat-sat6/sat6annotations.csv', header=None))

# Print shape of all training, testing data and labels
# Labels are already loaded in one-hot encoded format
print(x_train.shape) # (10000, 64, 64, 3)
print(y_train.shape) # (10000, 6)
print(x_test.shape)  # (10000, 64, 64, 3)
print(y_test.shape)  # (10000, 6)
```

```
             0  1  2  3  4  5  6
0     building  1  0  0  0  0  0
1  barren_land  0  1  0  0  0  0
2        trees  0  0  1  0  0  0
3    grassland  0  0  0  1  0  0
4         road  0  0  0  0  1  0
5        water  0  0  0  0  0  1
(10000, 64, 64, 3)
(10000, 6)
(10000, 64, 64, 3)
(10000, 6)
```

In [18]:

```
# dowload the resnet model
resnet_model = resnet50.ResNet50(weights='imagenet', include_top=False, input_shape=(64
```

In [19]:

```
# Adding classifier on top of Convolution model
# create model
model_resnet = Sequential()

# Add the resnet50 convolutional model
model_resnet.add(resnet_model)

# Adding new Layers
model_resnet.add(Flatten())
model_resnet.add(Dense(1024, activation='relu'))
model_resnet.add(Dropout(0.5))
model_resnet.add(Dense(6, activation='softmax'))
```

In [20]:

```
# summary of model
model_resnet.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Model)             (None, 2, 2, 2048)        23587712
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_3 (Dense)              (None, 1024)              8389632
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 6150
=================================================================
Total params: 31,983,494
Trainable params: 31,930,374
Non-trainable params: 53,120
_________________________________________________________________
```

In [21]:

```
# Compile the model
model_resnet.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

In [22]:

```
#### set the batch size, class number and epoch
batch_size = 256
epochs = 40
```

ResNet CNN Model Training

> We adopted the 40 epochs on ResNet CNN. Finally, the accuracy of the test set is stable at 0.94, and there is no overfitting.
> As shown in training history, the accuracy of ResNet grew slowly, but after 34 epoch, the accuracy of the test set stabilized at 0.98, which was much higher than that of CNN. Preliminary judgment indicates that the classification performance of ResNet is better than CNN.

In [23]:

```
# fit ResNet CNN with 40 epochs
model_resnet.fit(x_train.astype('float32'), y_train,
                 batch_size=batch_size,
                 epochs=epochs,
                 verbose=1,
                 validation_data=(x_test.astype('float32'), y_test))
```

```
Train on 10000 samples, validate on 10000 samples
Epoch 1/40
10000/10000 [==============================] - 28s 3ms/step - loss: 0.2756 -
acc: 0.9276 - val_loss: 2.0676 - val_acc: 0.1537
Epoch 2/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0391 -
acc: 0.9879 - val_loss: 9.9120 - val_acc: 0.2260
Epoch 3/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0150 -
acc: 0.9941 - val_loss: 6.5680 - val_acc: 0.0664
Epoch 4/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0180 -
acc: 0.9963 - val_loss: 13.3296 - val_acc: 0.3709
Epoch 5/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0059 -
acc: 0.9980 - val_loss: 21.4434 - val_acc: 0.1535
Epoch 6/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0023 -
acc: 0.9991 - val_loss: 29.8675 - val_acc: 0.1535
Epoch 7/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0063 -
acc: 0.9980 - val_loss: 69.4319 - val_acc: 0.1535
Epoch 8/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0146 -
acc: 0.9968 - val_loss: 74.0145 - val_acc: 0.1535
Epoch 9/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0102 -
acc: 0.9989 - val_loss: 36.2867 - val_acc: 0.1535
Epoch 10/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0018 -
acc: 0.9994 - val_loss: 39.2587 - val_acc: 0.1535
Epoch 11/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0047 -
acc: 0.9989 - val_loss: 65.3538 - val_acc: 0.1535
Epoch 12/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0030 -
acc: 0.9991 - val_loss: 48.2184 - val_acc: 0.1535
Epoch 13/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0383 -
acc: 0.9986 - val_loss: 195.1852 - val_acc: 0.1787
Epoch 14/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0034 -
acc: 0.9990 - val_loss: 65.4905 - val_acc: 0.1530
Epoch 15/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0057 -
acc: 0.9989 - val_loss: 46.4396 - val_acc: 0.0836
Epoch 16/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0092 -
acc: 0.9983 - val_loss: 77.6292 - val_acc: 0.1541
Epoch 17/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0051 -
acc: 0.9985 - val_loss: 125.0396 - val_acc: 0.1427
```

```
Epoch 18/40
10000/10000 [==============================] - 22s 2ms/step - loss: 3.7157e-
04 - acc: 0.9999 - val_loss: 93.2130 - val_acc: 0.1764
Epoch 19/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0060 -
acc: 0.9996 - val_loss: 84.7926 - val_acc: 0.1957
Epoch 20/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0218 -
acc: 0.9986 - val_loss: 6.1501 - val_acc: 0.3990
Epoch 21/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0014 -
acc: 0.9994 - val_loss: 4.3582 - val_acc: 0.5043
Epoch 22/40
10000/10000 [==============================] - 22s 2ms/step - loss: 2.4620e-
04 - acc: 0.9999 - val_loss: 3.1582 - val_acc: 0.5944
Epoch 23/40
10000/10000 [==============================] - 22s 2ms/step - loss: 6.3334e-
04 - acc: 0.9998 - val_loss: 1.5877 - val_acc: 0.7007
Epoch 24/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0307 -
acc: 0.9989 - val_loss: 1.2608 - val_acc: 0.8362
Epoch 25/40
10000/10000 [==============================] - 22s 2ms/step - loss: 9.1302e-
04 - acc: 0.9997 - val_loss: 0.8559 - val_acc: 0.8788
Epoch 26/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0010 -
acc: 0.9998 - val_loss: 0.8279 - val_acc: 0.8855
Epoch 27/40
10000/10000 [==============================] - 22s 2ms/step - loss: 2.7240e-
05 - acc: 1.0000 - val_loss: 0.7316 - val_acc: 0.9168
Epoch 28/40
10000/10000 [==============================] - 22s 2ms/step - loss: 3.6702e-
05 - acc: 1.0000 - val_loss: 0.6052 - val_acc: 0.9357
Epoch 29/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0087 -
acc: 0.9989 - val_loss: 0.7119 - val_acc: 0.9344
Epoch 30/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0012 -
acc: 0.9998 - val_loss: 0.6898 - val_acc: 0.8965
Epoch 31/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0044 -
acc: 0.9989 - val_loss: 2.8380 - val_acc: 0.7930
Epoch 32/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0024 -
acc: 0.9994 - val_loss: 0.6651 - val_acc: 0.9312
Epoch 33/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0459 -
acc: 0.9986 - val_loss: 0.4625 - val_acc: 0.9561
Epoch 34/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0052 -
acc: 0.9986 - val_loss: 0.1211 - val_acc: 0.9828
Epoch 35/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0016 -
acc: 0.9998 - val_loss: 0.0933 - val_acc: 0.9858
Epoch 36/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0013 -
acc: 0.9998 - val_loss: 0.0873 - val_acc: 0.9872
Epoch 37/40
10000/10000 [==============================] - 22s 2ms/step - loss: 8.9813e-
04 - acc: 0.9996 - val_loss: 0.0907 - val_acc: 0.9880
Epoch 38/40
```

```
10000/10000 [==============================] - 22s 2ms/step - loss: 4.8363e-
05 - acc: 1.0000 - val_loss: 0.0850 - val_acc: 0.9873
Epoch 39/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0038 -
acc: 0.9994 - val_loss: 0.3018 - val_acc: 0.9611
Epoch 40/40
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0012 -
acc: 0.9996 - val_loss: 0.1130 - val_acc: 0.9836
```

Out[23]:

```
<keras.callbacks.callbacks.History at 0x15932c7cd30>
```

ResNet Confusion Matrix

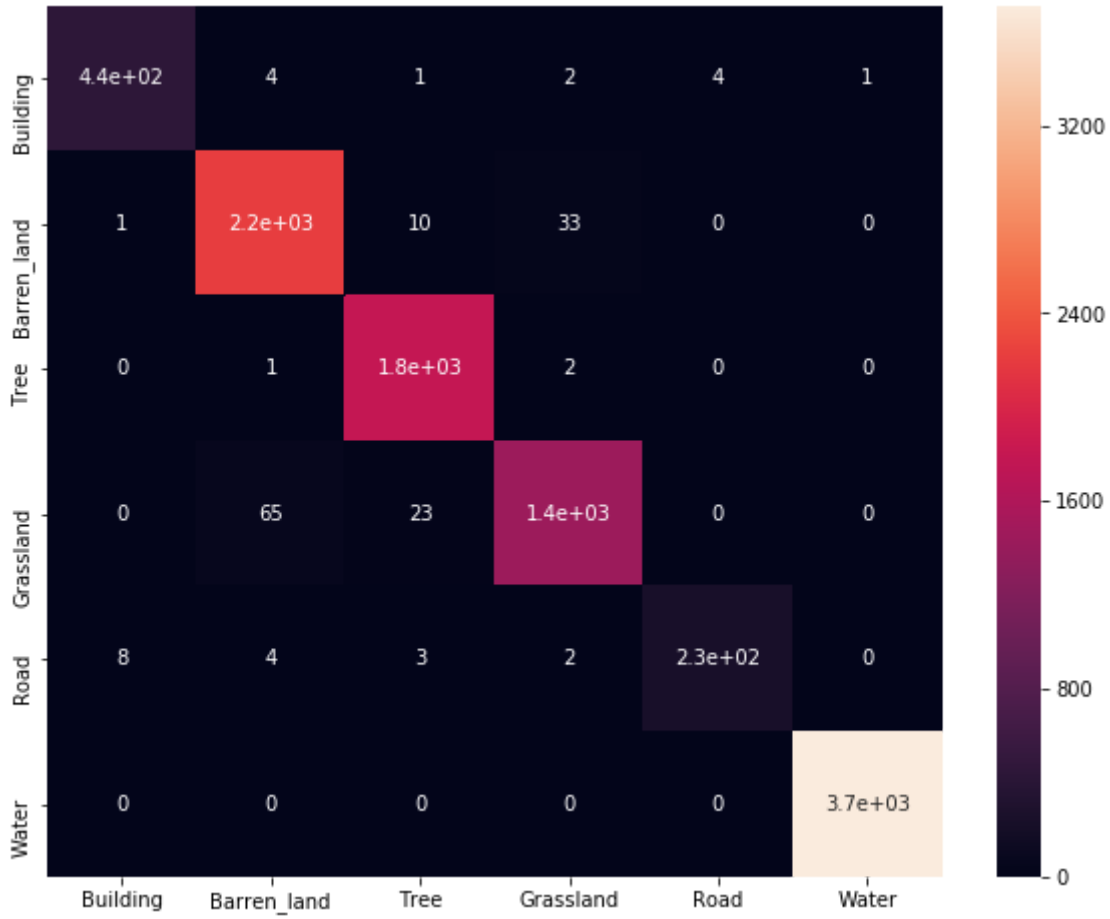
> ResNet's confusion matrix supports its better view. All six categories are barely mixed with the others, proving our classification to be distinguished.

In [24]:

```
#Label converter
# [1,0,0,0,0,0]=building
# [0,1,0,0,0]=barren_land
# [0,0,1,0,0,0]=tree
# [0,0,0,1,0,0]=grassland
# [0,0,0,0,1,0]=road
# [0,0,0,0,0,1]=water


##Building confusion matrix

y_pred=model_resnet.predict_classes(x_test)
y_true=np.argmax(y_test, axis=1)
cm=confusion_matrix(y_target=y_true, y_predicted=y_pred)

df_cm = pd.DataFrame(cm, index = [i for i in ["Building","Barren_land","Tree","Grasslar
                   columns = [i for i in ["Building","Barren_land","Tree","Grassland","F
plt.figure(figsize = (10,8))
sns.heatmap(df_cm, annot=True)
```

Out[24]:

<matplotlib.axes._subplots.AxesSubplot at 0x159348009e8>

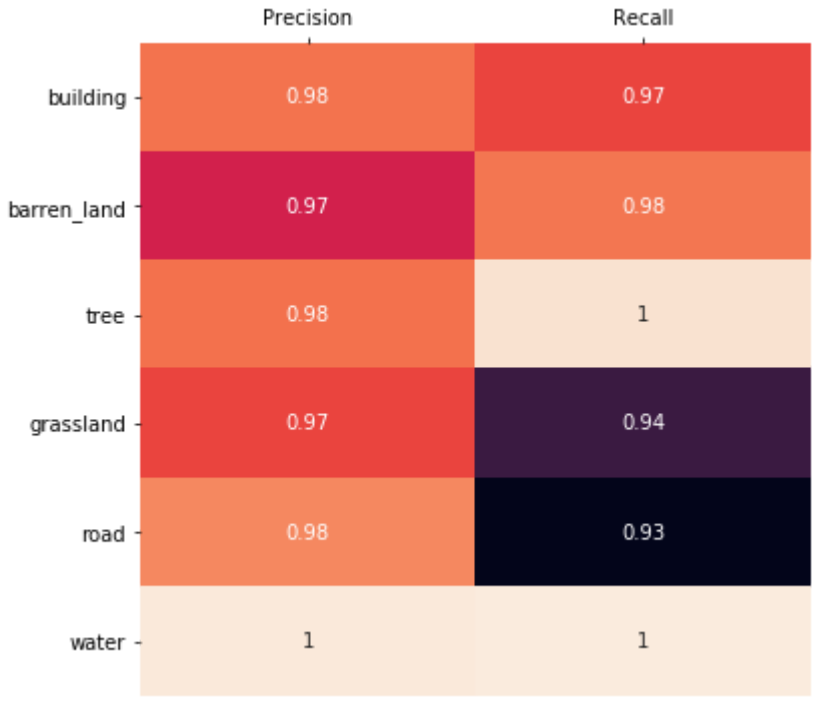


ResNet Precision and Recall calculation for multiclass classification

> ResNet CNN achieved very good results. The precision of all classes reach 0.97, and the recall of all classes reach 0.93. Although road is also the worst one, its precision as 0.98 and recall as 0.93 are very excellent result.

In [25]:

```
# loop through six classes to get their precison and recall
dic={}
precision_=[]
recall_=[]
precision_macro_average_=[]
for label in range(6):
    precision_.append(precision(label, cm))
    recall_.append(recall(label, cm))

dic["Precision"]= precision_
dic["Recall"]= recall_

plt.figure(figsize=(6,6))
ax=sns.heatmap(pd.DataFrame(dic, index=["building","barren_land","tree","grassland","ro
plt.yticks(rotation=0)
ax.xaxis.tick_top() # x axis on top

plt.show()
```



> In conclusion, in the first part, both CNN and ResNet CNN have excellent performance for our sat-4 data set. The precision of CNN is 0.94, and that of ResNet CNN is 0.98. Combining the results of confusion matrix, precision and recall, we can draw the conclusion that ResNet CNN is better than CNN, so we choose to apply ResNet CNN to the classification of satellite images of northwest China in the second part.

4.3 ResNet CNN Application on Northwest China Images

> In the second application part of the project, we need to crop the sentianl2 image of Northwest China manually. Similarly, 'for' loop were implemented to divide the image into 28 * 28 areas to keep the size consistent.

In [27]:

```
# read the pictures in the EuroSAT
path = "./pic/"

files = []
for r, d, f in os.walk(path):
    for file in f:
        if '.jpg' in file:
            files.append(os.path.join(r, file))
```

In [28]:

```
df = pd.DataFrame({'path':files})
```

In [30]:

```
# create a function to create a RGB pic
def creatRGB(array):
    array = imageio.imread(array)
    pixels = np.asarray(array)
    imge = pixels.astype('float32')
    return imge
```

In [31]:

```
# create column to store RGB
df['RGB'] = df.path.apply(creatRGB)
```

In [32]:

```
df.head()
```

Out[32]:

	path	RGB
0	./pic/1.jpg	[[[0.0, 1.0, 0.0], [0.0, 1.0, 0.0], [0.0, 1.0,...
1	./pic/10.jpg	[[[0.0, 1.0, 4.0], [0.0, 1.0, 4.0], [0.0, 1.0,...
2	./pic/100.jpg	[[[9.0, 0.0, 0.0], [12.0, 0.0, 0.0], [16.0, 0....
3	./pic/1000.jpg	[[[197.0, 171.0, 154.0], [214.0, 188.0, 171.0]...
4	./pic/10000.jpg	[[[190.0, 211.0, 194.0], [212.0, 212.0, 204.0]...

In [33]:

```
# Normalize RGB
pic = np.array(df.RGB.to_list())/255
```

In [34]:

```
# resize the images to 64*64
X = []
for i in range(0,len(pic)):
    bottle_resized = resize(pic[i], (64, 64))
    X.append(bottle_resized)
X = np.asarray(X)
```

Use our ResNet Model to Predict

In [35]:

```
# get the list of predict
result_label = model_resnet.predict_classes(X)
df['Predict'] = result_label
```

Prediciton Visualization

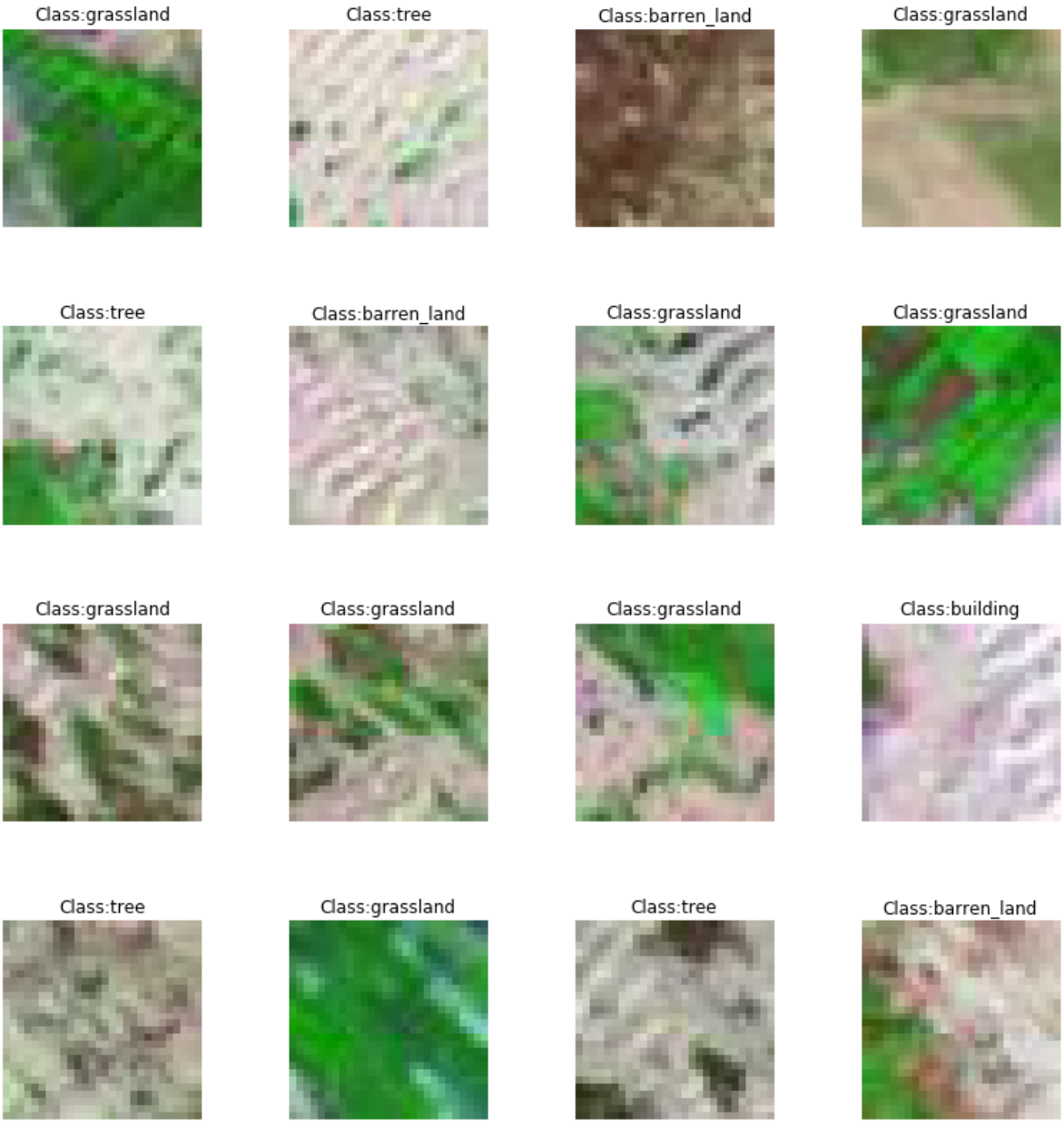
> 16 images were randomly selected for visualization, with predicted classes. It can be found that the effect of our model classification is not so good, and there are some errors. For example, the image of (3, 4) should be barren land, but it is classified as building. (4, 1) and (4, 3) have no green pixels but are recognized as trees.

In [37]:

```
# name each class in prediction
name_list = ["building","barren_land","tree","grassland","road","water"]
def changeName(num):
    return name_list[num]

df["Name"] = df.Predict.apply(changeName)
```

In [83]:

```
# random select 16 images to have a look
def data_visualization_predict(data, n):
    ##data: training or test data
    ##lable: training or test labels
    ## n: number of data point, it should be less than or equal to no. of data points
    fig = plt.figure(figsize=(14, 14))
    ax = []  # ax enables access to manipulate each of subplots
    rows, columns=4,4
    for i in range(columns*rows):
        index=np.random.randint(1,n)
        img= df.RGB[index].astype(np.uint8)
        ax.append( fig.add_subplot(rows, columns, i+1) ) # create subplot and append to
        ax[-1].set_title("Class:"+str(df.Name[index]))  # set class
        plt.axis("off")
        plt.imshow(img)

    plt.subplots_adjust(wspace=0.1,hspace=0.5)
    plt.show()  # finally, render the plot
    ""
data_visualization_predict(df, n = 11770 )
```



Finally, we made statistics on the predicted classes (Figure 4). The pie chart shows that in the study area, barren land has the highest proportion as 45.5%. Next is grassland, accounting for 22.9%. Building and tree account for 15.5% and 13.9%, respectively. Finally, roads and waters account for 0.7% and 1.5%.

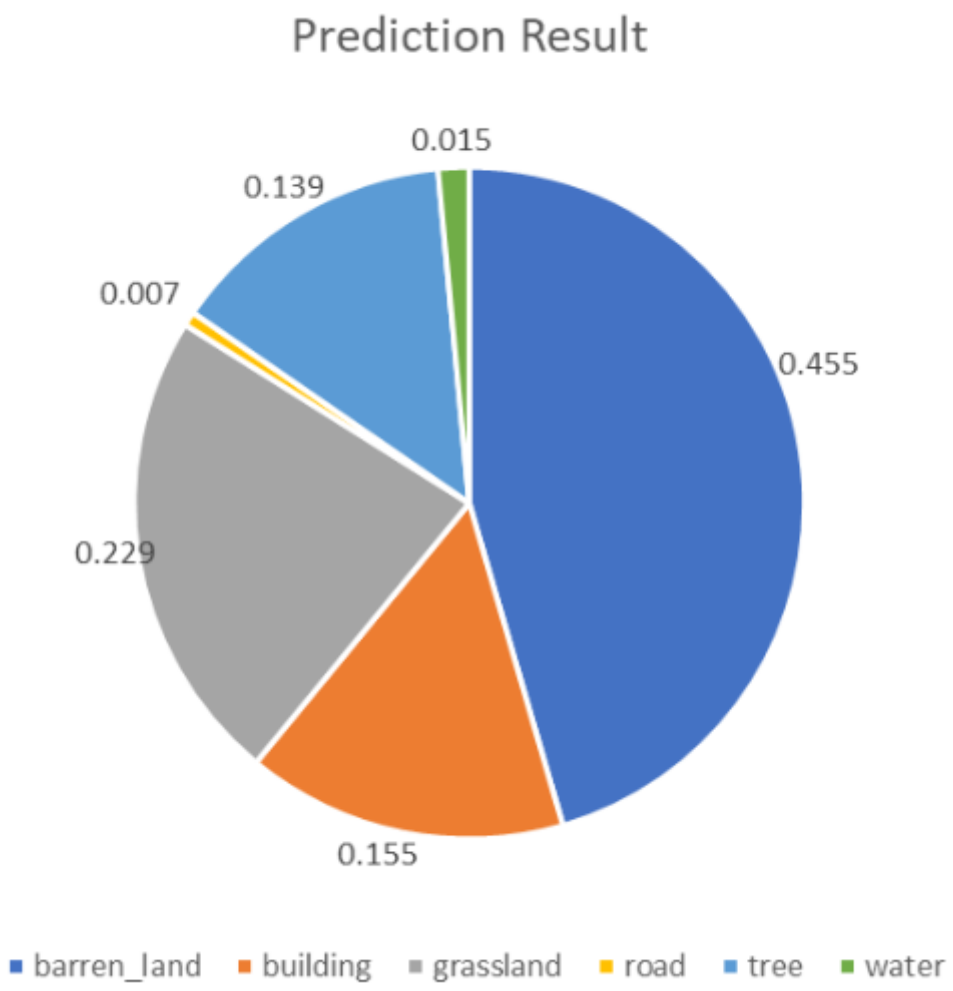


Figure 4.ResNet Prediciton on Sentinal 2 Image

5. Discussion

In summary, both of the deep learning algorithms used in this study performed well on the SAT-6 data set, and their accuracies are higher than 0.9. This is in line with our expectations because processing images is a typical function of deep learning. Specifically, ResNet CNN is better than CNN, especially on the road and barren land classes. Also, the precision and recall of ResNet CNN are all higher than 0.93, so the deeper neural network does have a certain degree of advantage when processing large amounts of satellite data.

The application result of ResNet in sentinal2 image in northwest China shows that the proportion of green (grassland and trees) is about 36.8%, which is very close to the current reported vegetation coverage rate as 33%, which proves the reliability of the model. However, the percentage of barren land and building is worth discussing. The northwestern part of China is mostly plateau and sparsely populated. The model application result shows that the building percentage is as high as 0.16, which is not in line with common sense. Through the sample review of result image, we can find that there are many barren land images showing high-gloss white, very similar to the building images in the SAT-6 dataset. This may cause the model to wrongly classify barren land as building, reducing the proportion of barren land.

Given this error, there are two possible factors: First, the two datasets are from the United States and China, respectively, and the geographical environments of the two places are different. Northwest China has a high altitude, and direct sunlight leads to high ground brightness, while California in the United States has a low altitude and a mild climate, which causes the land color to be yellowish.

In this study, due to data limitations, we cannot obtain labeled data for the same satellite in a similar area, so the results show excellent performance on the original dataset but poor performance on other datasets. Therefore, the future improvement direction can be collecting data from the same satellite and similar areas for model training and application, so the powerful capabilities of ResNet CNN in the land use classification can be exerted.

6. References

Ai, F. F., Bin, J., Zhang, Z. M., Huang, J. H., Wang, J. B., Liang, Y. Z., ... & Yang, Z. Y. (2014). Application of random forests to select premium quality vegetable oils by their fatty acid composition. Food chemistry, 143, 472-478.

Basu, S., Ganguly, S., Mukhopadhyay, S., DiBiano, R., Karki, M., & Nemani, R. (2015). Deepsat: a learning framework for satellite imagery. In Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems (pp. 1-10).

Chris Crawford (2018). DeepSat (SAT-6) Dataset.Retrieve from https://www.kaggle.com/crawford/deepsat-sat6 (https://www.kaggle.com/crawford/deepsat-sat6)

Mnih, V., & Hinton, G. E. (2010). Learning to detect roads in high-resolution aerial images. In European Conference on Computer Vision (pp. 210-223). Springer, Berlin, Heidelberg.

Nasa (2019). China and India Lead the Way in Greening. Retrieve from https://earthobservatory.nasa.gov/images/144540/china-and-india-lead-the-way-in-greening (https://earthobservatory.nasa.gov/images/144540/china-and-india-lead-the-way-in-greening)

Priya Dwivedi (2019). Understanding and Coding a ResNet in Keras. Retrieve from https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33 (https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33)

Wikipedia (2020) Convolutional neural network. Retrieve from https://en.wikipedia.org/wiki/Convolutional_neural_network (https://en.wikipedia.org/wiki/Convolutional_neural_network)
Zhihu (2020). There is no one before! China is about to wipe out the Mu Us Desert! Retrieve from https://zhuanlan.zhihu.com/p/51319209 (https://zhuanlan.zhihu.com/p/51319209) on May 8, 2020