Project Report

Group5
Chenxi Sun

Abstract

There are 5 steps for our general process dealing with 'Understanding the Amazon form Space' project.

First, download the data sets, browse images and analyze labels. We find that each image's tag is composed of several single labels. There are 17 single labels in all. And the labels can be divided into different categories.

For the second step, we build two models — weather model and label13 model. Both of these models are built based on convolutional neutral network. we use keras library to built and train these two models. After getting two models, we make a combination model.

Third, after we get the trained model, we evaluate two models and one combination model by calculating F2-score. For label13 model, the F2-score value is 0,8344. For weather, the accuracy is 0.878. For combination model the F2-score value is 0.874. And we also analyze the reason why get these results.

Fourth, we summarize the advantages and disadvantages of two models and try to propose some improved methods.

Finally, we use our models to predict the tags for the images in test-jpg folder.

1. Analyze Data

We download the data from kaggle and get train-jpg folder (contain the images for training set), train_v2.csv (labels for training set's images), test-jpg folder (test data). Then use python on jupyter notebook to do data analysis.

First, show tags of images. We find that each tag is composed of several single labels. For example, train_0.jpg's tag is 'haze primary'. That is, this image's tag contains two single labels — haze and primary.

And we sum up all single labels and there are 17 kinds of single labels in all. see figure 1. The number of images for each single labels, see figure 2.

```
(['haze',
  'primary',
  'agriculture',
  'clear',
  'water',
  'habitation',
  'road',
  'cultivation',
  'slash_burn',
  'cloudy',
  'partly_cloudy',
  'conventional_mine',
  'bare_ground',
  'artisinal_mine',
  'blooming',
  'selective_logging',
  'blow_down'],
 17)
```

figure 1 all single labels

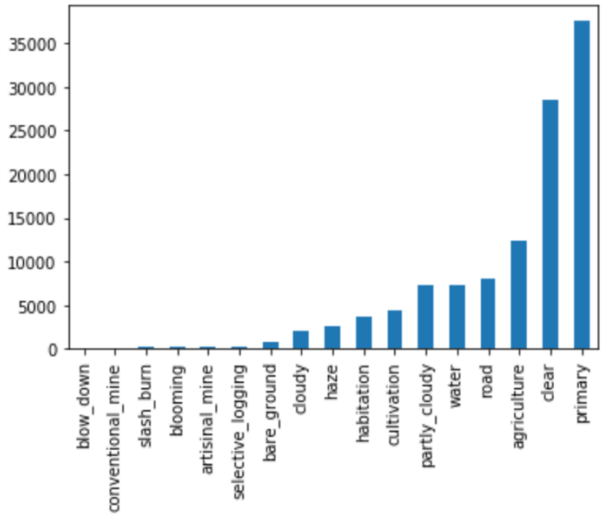


figure 2 number of images for each single labels

We also find these single labels can be divided into several categories. For example, weather labels have ‘haze’, ‘clear’, ‘cloudy’, ‘partly cloudy’. We decide to make two categories — weather class and label13 class.

Weather class have the labels which are used to describe the weather of images. Why we decide to make a weather class? There are two reasons:

- One image could have one weather label at most (analysis shown in figure 3). It is a four-classification problem.
- Every image must have a weather label. (the number of showing times of four labels equal the number of all images)

Label13 class contain 13 kinds of labels, most of them are used to describe the landscape of images. But at first we made two categories for label13 class, one is land class and the other is rare class. Rare labels are that the labels which have rare images. All of their number of images are less than 1000. But we think all of them have the feature of landscape. So we decide to make one label13 class rather than two classes.

Then we do visualization for single labels. We make co-occurrence matrix for each category. Figure 3 is weather’ co-occurrence matrix. It’s clearly that the weather co-occurrence matrix is a diagonal matrix. That is, for each images, it should have exactly one weather label. It makes sense because it’s not possible to have two kinds of weather at the same time.

	clear	partly_cloudy	haze	cloudy
clear	28431	0	0	0
partly_cloudy	0	7261	0	0
haze	0	0	2697	0
cloudy	0	0	0	2089

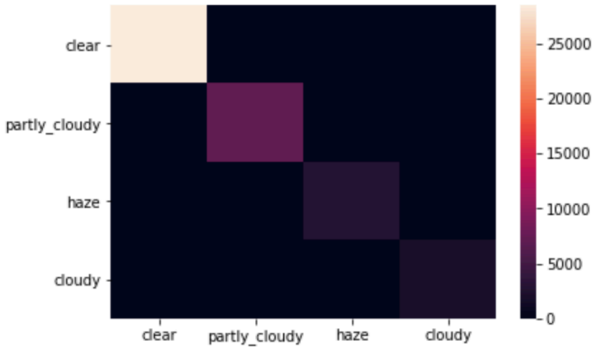


figure 3 weather's co-occurrence matrix

Figure 4 is land's co-occurrence matrix and figure 5 is rare labels' co-occurrence matrix. We can find that both of them are not diagonal matrix. One image can have several land labels and rare labels at the same time. And that is also why we decide to make these two kinds of categories as one class.

	primary	agriculture	water	cultivation	habitation	road
primary	37513	11972	7001	4455	3469	7728
agriculture	11972	12315	2712	3377	2737	6034
water	7001	2712	7411	868	915	2125
cultivation	4455	3377	868	4477	895	1294
habitation	3469	2737	915	895	3660	2786
road	7728	6034	2125	1294	2786	8071

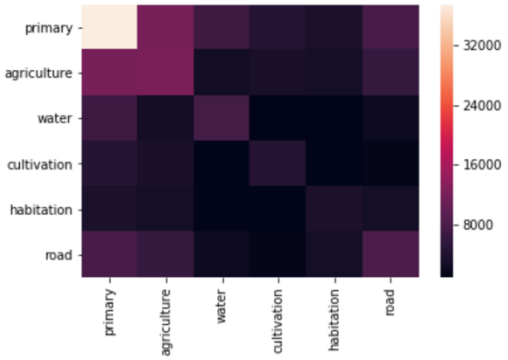


figure 4 land's co-occurrence matrix

	slash_burn	conventional_mine	bare_ground	artisinal_mine	blooming	selective_logging	blow_down
slash_burn	209	0	10	0	2	2	2
conventional_mine	0	100	10	4	0	0	0
bare_ground	10	10	862	40	3	13	4
artisinal_mine	0	4	40	339	0	6	0
blooming	2	0	3	0	332	7	1
selective_logging	2	0	13	6	7	340	1
blow_down	2	0	4	0	1	1	98

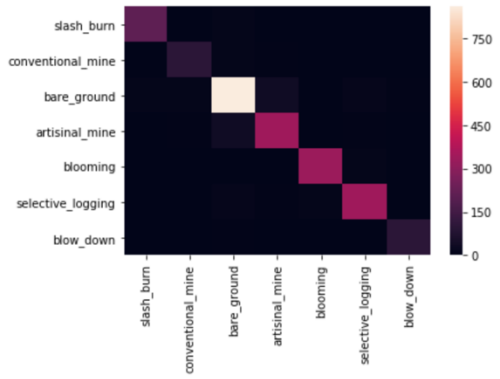


figure 5 rare co-occurrence matrix

Then make a big co-occurrence matrix for all of the labels, see figure 6. In this matrix, in 'cloudy row', only 'cloudy column' has the value, other column's values are 0. Thus, we reckon that if one image has the 'cloudy' label, it will not have other labels.

	haze	primary	agriculture	clear	water	habitation	road	cultivation	slash_burn	cloudy	partly_cloudy	conventional_mine	bare_ground	ar
haze	2697	2670	672	0	613	129	394	202	3	0	0	2	41	
primary	2670	37513	11972	27668	7001	3469	7728	4455	209	0	7175	94	683	
agriculture	672	11972	12315	9150	2712	2737	6034	3377	119	0	2493	24	225	
clear	0	27668	9150	28431	5502	3090	6295	3527	173	0	0	70	747	
water	613	7001	2712	5502	7411	915	2125	868	24	0	1295	26	206	
habitation	129	3469	2737	3090	915	3660	2786	895	41	0	441	36	163	
road	394	7728	6034	6295	2125	2786	8071	1294	36	0	1382	59	323	
cultivation	202	4455	3377	3527	868	895	1294	4477	126	0	748	4	89	
slash_burn	3	209	119	173	24	41	36	126	209	0	33	0	10	
cloudy	0	0	0	0	0	0	0	0	0	2089	0	0	0	
partly_cloudy	0	7175	2493	0	1295	441	1382	748	33	0	7261	28	74	
conventional_mine	2	94	24	70	26	36	59	4	0	0	28	100	10	
bare_ground	41	683	225	747	206	163	323	89	10	0	74	10	862	
artisinal_mine	5	324	38	307	299	29	110	18	0	0	27	4	40	
blooming	4	332	32	311	16	4	10	35	2	0	17	0	3	
selective_logging	5	340	65	308	49	13	151	58	2	0	27	0	13	
blow_down	0	98	22	85	3	3	2	8	2	0	13	0	4	

figure 6 co-

We show the sample image for each single label to see how them look like. Here, we just put 4 images for 4 labels (partly-cloudy, road, primary, agriculture) as figure 7.

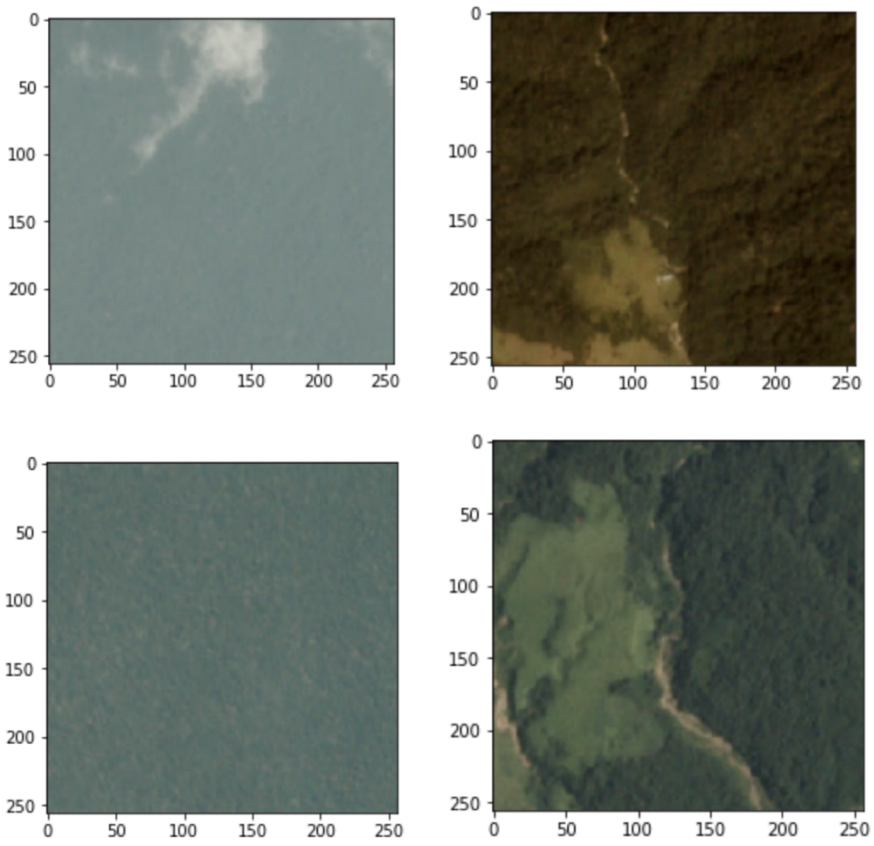


figure 7 partly-cloudy, road, primary, agriculture images

2. Build Models

2.1 Label13 Model

As one image can have several labels in label13 class. So we can build model for every separated labels. One original idea is that we create 13 models of every labels. And each model can solve the binary classification problem. For example, a road model's mission is that telling whether the image has road feature or not. 13 models will produce 13 results, assembling these results will get the final results.

In practice, we just build one model to solve this problem. First we choose to use convolutional neural network to make the model. As all of 13 models are to solve the binary classification problem, they will have the same structure of cnn. Thus, making one general model and letting 13 models to share it's layers is a better choice. Our label13 model using keras is shown in figure 8.

```
optimizer = RMSprop(lr=1e-5)
objective = 'binary_crossentropy'
def convnet():

    model = Sequential()

    model.add(Convolution2D(32, kernel_size=(3, 3), padding='same', input_shape=(64, 64, 3), activation='relu'))
    model.add(Convolution2D(32, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Convolution2D(64, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(Convolution2D(64, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Convolution2D(128, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(Convolution2D(128, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Convolution2D(256, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(Convolution2D(256, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
    model.add(Dropout(0.5))

    model.add(Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
    model.add(Dropout(0.5))

    model.add(Dense(13))
    model.add(Activation('sigmoid'))

    model.compile(loss=objective, optimizer=optimizer, metrics=['accuracy'])
    return model
```

figure 8 label13 model using keras

2.1.1 model details

Input

Choose 10000 images in train-jpg folder. There are rare images which have labels of slash_burn, conventional_mine, bare_ground, artisinal_main, blooming, selective_logging and blow_down. We dare the network cannot learn the pattern of rare labels. so we write code to extract images which have the specific labels to pick out more images which have these rare labels and add them into training set. The number of training images is 10798. the size of training image is 64*64*3. But at first they are 256*256*4. We find the 4th channel (alpha channel) is always 255. it's no use for the feature learning and the training speed will be faster without this channel. Input x is an array which are got from the resized input images.

One image has one corresponding tag. We define each tag is a vector contain 13 values. '1' means that this image has this single label, '0' means that the image does not have this single label. That is one-hot code. Of course one vector can have several '1' which means one image can have several single labels.

Convolutional Neural Network

Use keras to build this model. First, two convolutional layer which have 32 kinds of 3*3 kernels, of cause the active function are relu. And then use the 2*2 max pooling layer. Then 2 convolutional layers which both have 64 kinds of 3*3 kennels. And then one max pooling layer. Then similar layers. Then fully connected layer. Finally, output layer. In this model we put 13 output neurons in output layer. And each of them present one kind of label. For every neuron we use sigmoid to active it. Because each neuron's job is to tell whether the input owns the corresponding label or not (binary classification problem). If the output is more than 0.5, it means the input image should have the corresponding single label.

Use RMSprop as the optimizer, the learning rate is 0.00001. Loss function use binary crossentropy. We add dropout to fix the over fitting problem.

Train the Model

Put the training set into this model, compile and fit it. we use early stopping to protect it to be over fitting. The patient is 16. 75% inputs are training set, 25% are validation set. Run it and get

the result as figure 9. After early stopping, finally the accuracy on training set is 0.9434 and the validation accuracy is 0.9020.

```
Epoch 173/1024
8098/8098 [==============================] - 437s - loss: 0.1571 - acc: 0.9430 - val_loss: 0.2707 - val_acc: 0.9000
Epoch 174/1024
8098/8098 [==============================] - 434s - loss: 0.1574 - acc: 0.9424 - val_loss: 0.2896 - val_acc: 0.9006
Epoch 175/1024
8098/8098 [==============================] - 426s - loss: 0.1574 - acc: 0.9431 - val_loss: 0.2817 - val_acc: 0.9030
Epoch 176/1024
8098/8098 [==============================] - 456s - loss: 0.1566 - acc: 0.9426 - val_loss: 0.2876 - val_acc: 0.9034
Epoch 177/1024
8098/8098 [==============================] - 452s - loss: 0.1569 - acc: 0.9427 - val_loss: 0.2801 - val_acc: 0.9044
Epoch 178/1024
8098/8098 [==============================] - 407s - loss: 0.1561 - acc: 0.9424 - val_loss: 0.2782 - val_acc: 0.9012
Epoch 179/1024
8098/8098 [==============================] - 435s - loss: 0.1554 - acc: 0.9428 - val_loss: 0.2963 - val_acc: 0.9020
Epoch 180/1024
8098/8098 [==============================] - 441s - loss: 0.1553 - acc: 0.9426 - val_loss: 0.2892 - val_acc: 0.9027
Epoch 181/1024
8098/8098 [==============================] - 436s - loss: 0.1547 - acc: 0.9434 - val_loss: 0.3022 - val_acc: 0.9020
Epoch 00180: early stopping
```

figure 9 training model

Draw the loss graph as figure 10. We find both the training loss and the validation loss become lower and lower until There's a tendency that the validation loss starts to increase.

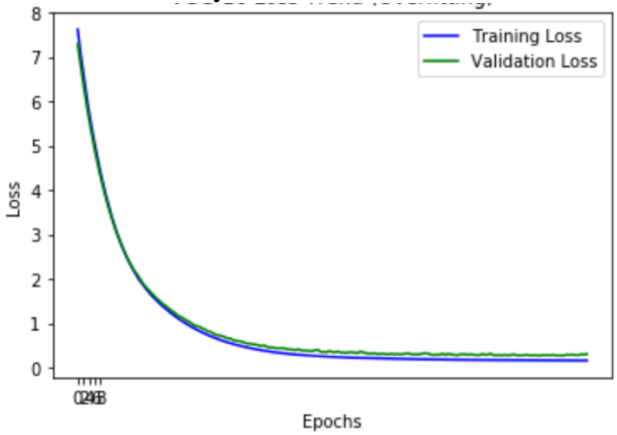


figure 10 loss graph

Save Model

Finally save this model and the weight in h5 file.

Prediction

When one image is put into this model, a vector will be given back. This vector has 13 values. Each value represents the possibility of the image having corresponding label. If a value is higher than 0.5, we will accept the corresponding label.

2.2 Weather Model

As one image can have at most one weather label, this is a four-classification problem. we also use cnn based on keras to build this model.

Four kinds of labels' images and each single label have 1000 images. All of them is 4000 images. Input labels are vectors which have 4 values. There is only one '1' in 4 values as one images can only have one weather label. The image size is same as label13 model input.

convolutional neural network is similar with label13 model but the output layer has 4 neurons and use softmax to figure out which one is the the best predicting label.

The model is shown in figure 12.

```
optimizer = RMSprop(lr=1e-5)
objective = 'binary_crossentropy'
def convnet():

    model = Sequential()

    model.add(Convolution2D(32, kernel_size=(3, 3), padding='same', input_shape=(64, 64, 3), activation='relu'))
    model.add(Convolution2D(32, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Convolution2D(64, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(Convolution2D(64, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Convolution2D(128, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(Convolution2D(128, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Convolution2D(256, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(Convolution2D(256, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
    model.add(Dropout(0.5))

    model.add(Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
    model.add(Dropout(0.5))

    model.add(Dense(4))
    model.add(Activation('softmax'))

    model.compile(loss=objective, optimizer=optimizer, metrics=['accuracy'])
    return model
```

figure 12 weather model using keras

When training the weather model, use early stopping with validation loss and patient 100 to avoid over fitting. Shown as figure 13.

```
Epoch 223/1024
3000/3000 [==============================] - 144s - loss: 0.5968 - acc: 0.9490 - val_loss: 4.5334 - val_acc: 0.5125
Epoch 224/1024
3000/3000 [==============================] - 143s - loss: 0.5929 - acc: 0.9481 - val_loss: 4.2494 - val_acc: 0.5045
Epoch 225/1024
3000/3000 [==============================] - 143s - loss: 0.5856 - acc: 0.9518 - val_loss: 4.9710 - val_acc: 0.5105
Epoch 226/1024
3000/3000 [==============================] - 143s - loss: 0.5912 - acc: 0.9497 - val_loss: 5.2269 - val_acc: 0.5065
Epoch 227/1024
3000/3000 [==============================] - 144s - loss: 0.5840 - acc: 0.9506 - val_loss: 5.0768 - val_acc: 0.5050
Epoch 228/1024
3000/3000 [==============================] - 143s - loss: 0.5837 - acc: 0.9483 - val_loss: 4.3084 - val_acc: 0.5070
Epoch 229/1024
3000/3000 [==============================] - 143s - loss: 0.5689 - acc: 0.9534 - val_loss: 4.8560 - val_acc: 0.5035
Epoch 230/1024
3000/3000 [==============================] - 143s - loss: 0.5707 - acc: 0.9531 - val_loss: 5.1557 - val_acc: 0.5062
Epoch 231/1024
3000/3000 [==============================] - 143s - loss: 0.5723 - acc: 0.9498 - val_loss: 3.9560 - val_acc: 0.5168
Epoch 00230: early stopping
```

figure 13 result

The loss graph is shown in figure 14.

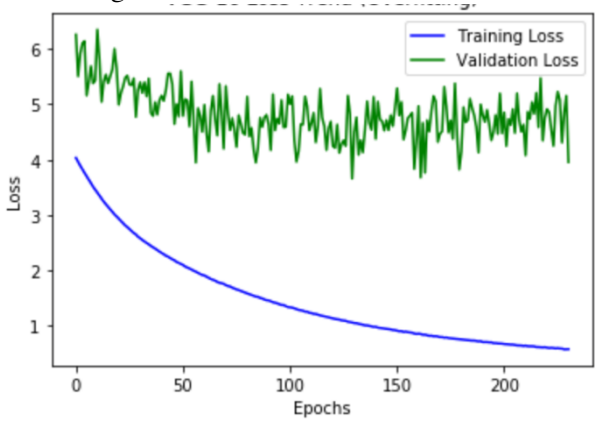


figure 14 loss graph

As we can see, the last validation accuracy is not good, only 0.5168. And the validation loss does not have the decrease tendency. But the training accuracy is high to 0.95. In this cease, there is no over fitting as we use the early stopping. So it dose get the best weight for this model. If the accuracy is bed, there is something wrong with the model itself, but not the epoch or over fitting. But when we make prediction of some sample using this model, the accuracy is not bad. Detailed testing and analysis are presented in part 3.

Prediction

When one image is put into this model and predict it labels. The out out is a vector which have 4 values. We will take the label which have the highest value as the image's weather label.

2.3 Combination Model

The combination model is a serialization of weather model and label13 model. The prediction label result link two parts of labels which are got from two models.

3. Model Evaluation

3.1 Label13 Model Evaluation

Select 1000 images in train-jpg folder and use this model to predict labels. Using F2-score to analyze. F2-score is:

$$(1+\beta^2)\frac{pr}{\beta^2 p+r} \quad \text{where } p=\frac{tp}{tp+fp},\ r=\frac{tp}{tp+fn},\ \beta=2.$$

Write a function to calculate F2-score, the function is shown as figure 15.

```
In [99]: def f2_score1():
             acc=[]
             for i in range(1000):
                 acc.append([])
             accuracy1=0.0
             accuracy2=0.0
             accuracy3=0.0

             for i in range(len(trainset_y)-1):
                 right=0.0
                 pre1real0=0.0
                 pre0real1=0.0
                 for j in range(0,len(trainset_y[i])-1):
                     if trainset_y[i][j]==real_y[i][j]==1:
                             right+=1
                     if trainset_y[i][j]==1 and real_y[i][j]==0:
                             pre1real0+=1
                     if trainset_y[i][j]==0 and real_y[i][j]==1:
                             pre0real1+=1
                 acc[i].append(right)
                 acc[i].append(pre1real0)
                 acc[i].append(pre0real1)
                 if right+pre1real0==1 or right+pre0real1==0:
                     accuracy3=accuracy3+1
                 else:
                     accuracy1=right/(right+pre1real0)
                     accuracy2=right/(right+pre0real1)
                     if(accuracy2+accuracy1==0):
                         accuracy3+=2
                     else:
                         accuracy3=accuracy3+5*accuracy1*accuracy2/(4*accuracy2+accuracy1)
             print (accuracy3)/1000
```

figure 15 F2-score function

Put the result of prediction into F2-score function, we can get the F2-score is 0.83442

```
In [100]: f2_score1()
          0.834419308894
```

The average F2-score for the predict is 0.83442. It's not good.

We also make F2-score for each single label, and the result is as figure 16

```
F2_score for each single lable
primary:           0.984226
agriculture:       0.878044
water:             0.716883
habitation:        0.715616
road:              0.822789
cultivatiob:       0.691184
slash_burn:        0.000000
conventional_mine:0.000000
bare_ground        0.219298
artisinal_mine     0.000000
blooming           0.000000
selective_logging  0.000000
blow_down:         0.000000
```

figure 16 F2-score

In the F2-score for each single label, we can find that many label's score is zero. Print the true-positive, false-positive, false-negative values for last 7 labels. The value is small, even less than 10. That is, there are a few images have these 7 labels in this prediction set. At the same time, we find that both these 7 labels are rare labels. as the training set have a few images of these labels, so the feature might be not learned well.

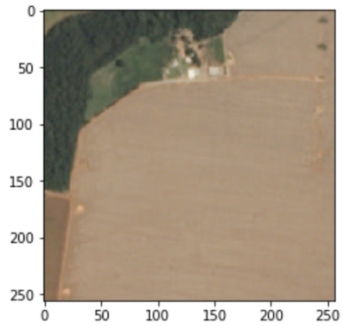
And here are other accuracy -- accuracy of all labels right, accuracy of allowing one label wrong, accuracy of allowing two labels wrong. The values of them are as follow.

```
In [101]: accuracy2/1000,accuracy3/1000,accuracy4/1000
Out[101]: (0.707, 0.847, 0.922)
```

Have quick look at predicting labels and it's corresponding images. See figure 16

```
In [500]: #prediction
          test_y[40642]
Out[500]: ['primary', 'agriculture', 'road']
```

```
In [499]: #real image
          numofimg=40642
          test_img=mpimg.imread(PLANET_KAGGLE_ROOT+"/test-jpg/test_"+str(numofimg)+".jpg")
          plt.imshow(test_img)
Out[499]: <matplotlib.image.AxesImage at 0x1cc7dc8ad0>
```



```
In [102]: #the predict label of one image
          test_y[0]

Out[102]: ['primary']

In [103]: #image
          numofimg=0
          test_img=mpimg.imread(PLANET_KAGGLE_ROOT+"/test-jpg/test_"+str(numofimg)+".jpg")
          plt.imshow(test_img)

Out[103]: <matplotlib.image.AxesImage at 0x1c4139a310>
```

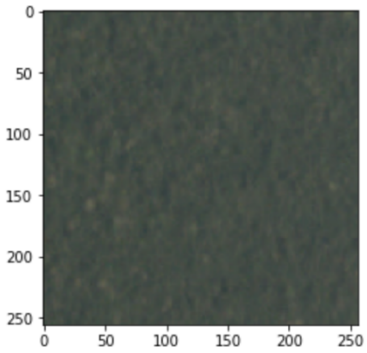


figure 16

3.2 weather model evaluation

As one image can have one weather label at most, we choose simple accuracy but not F2-score to evaluate the model. The accuracy is defined as follow:

$$accuracy = \frac{number\ of\ right\ prediction}{number\ of\ test\ images}$$

We write the function to calculate this accuracy. Extract 1000 images from training set as test set. Put it into model and predict the labels. Use the result and the accuracy function to calculate the accuracy. Then we can get the accuracy which is 0.878. See figure 17.

```
In [95]: right_num=0.0
         for i in range(0,1000):
             if (weather_real_y[i]==np.array(weather_predict_y[i])).all():
                 right_num+=1

In [96]: right_num/1000

Out[96]: 0.878
```

figure 17 weather model test accuracy

We have analyzed in part 2, when training the model, the validation accuracy is always 0.5. But here, the accuracy is more than 0.8. We have made sure that the test images are different from the training images. And we also make many tests, all tests' results are more than 0.8. But why does this happen? We are not sure yet, but we have some speculation:

The validation accuracy 0.5 is not bad for four-classification problem.

When training the model, using 0.25 splitting images as training set and validation set has chance. Maybe the validation set have many images which are difficult to classify by weather model.

When training weather model, the training accuracy is more than 0.9. It is also a principle to define the presentation of model. Because the model is not over fitting.

But we have no idea about the most reasonable explanation. We have to continue learning.

3.3 Combination model evaluation

Link the label13 model and weather model, we get the combination model. Put 1000 images as samples into the combination model and do prediction. Then we can get the prediction labels which can be seen as the linking of model13 model prediction labels and weather model prediction labels. The calculate the F2-score. The function is created as figure 18.

```
def f2_score():
    f2_score=0.0
    precision=0.0
    recall=0.0
    for i in range(len(combination_y)):
        pre1real1=0.0
        pre1real0=0.0
        pre0real1=0.0
        for j in range(len(combination_y[i])):
            if combination_y[i][j]==combination_real_y[i][j]==1:
                    pre1real1+=1
            if combination_y[i][j]==1 and combination_real_y[i][j]==0:
                    pre1real0+=1
            if combination_y[i][j]==0 and combination_real_y[i][j]==1:
                    pre0real1+=1
        if pre1real1+pre1real0==0 or pre1real1+pre0real1==0:
            f2_score=f2_score+1
        else:
            precision=pre1real1/(pre1real1+pre1real0)
            recall=pre1real1/(pre1real1+pre0real1)
            if(precision+recall==0):
                f2_score+=1
            else:
                f2_score=f2_score+5*precision*recall/(4*precision+recall)
    print (f2_score)/1000
```

figure 18 F2-score function for combination model

Run this function, we get the value of F2-score, the value is 0.866

```
f2_score()
```

```
0.866439420212
```

Then we change the test set, using other 5000 images to get the F2-score, the value is 0.874. It is a not bad accuracy.

4. Advantages and Disadvantages

4.1 Advantages

- We separate the labels into two reasonable categories. Let several single labels share the layers of one model.
- We make some changes when we decide whether a label belongs to an image or not. In label13 model, one original idea is that as we use sigmoid active function, one label belongs to an image if the predict accuracy is higher than 0.5. But it performs badly when make a decision about the rare label. Then we change the threshold to 0.1 for rare labels. The result becomes better.
- Our model's F2-score is not bad. Higher than 0.85 and average is 0.87.

4.2 Disadvantages

- Although we do some changes when we make a decision about the rare labels, the result is not good yet.
- There is still a big space for improvement in accuracy
- Use cnn as the model and use library's function, do not have much innovation.

4.3 Improvement method

- For rare labels' images, we can make horizontal/vertical flip and rotation to get more images to rich the data set. It's one reason that the rare labels prediction is bad.
- To do some images' data enhancement. As for weather, the feature of weather is global but not detailed. We can do something to enhance images' weather feature.
- Choice other neural network to solve this problem. Because our knowledge is limited, we still have to learn more knowledge of machine learning.

5. Test-jpg prediction

We use our model to predict the images' tags in test-jpg folder. Then save as csv file. We try to put it on kaggle and get the score, but it failed. Because kaggle's submission form must have more data (mapping work) which we don't do. So we just get the prediction tags in csv file but we have no way to get its accuracy. The result is shown in figure 19.

image_name	tags
test_0	['primary', 'clear']
test_1	['primary', 'clear']
test_2	['primary', 'partly_cloudy']
test_3	['primary', 'partly_cloudy']
test_4	['primary', 'partly_cloudy']
test_5	['primary', 'clear']
test_6	['primary', 'agriculture', 'partly_cloudy']
test_7	['primary', 'agriculture', 'habitation', 'road', 'clear']
test_8	['primary', 'clear']
test_9	['primary', 'haze']
test_10	['primary', 'partly_cloudy']
test_11	['primary', 'clear']
test_12	['partly_cloudy']
test_13	['primary', 'clear']
test_14	['primary', 'clear']
test_15	['primary', 'clear']
test_16	['primary', 'agriculture', 'habitation', 'road', 'clear']
test_17	['primary', 'partly_cloudy']
test_18	['primary', 'haze']
test_19	['primary', 'clear']
test_20	['primary', 'water', 'artisinal_mine', 'clear']
test_21	['primary', 'agriculture', 'partly_cloudy']
test_22	['primary', 'agriculture', 'road', 'clear']
test_23	['cloudy']
test_24	['primary', 'agriculture', 'road', 'partly_cloudy']
test_25	['primary', 'clear']
test_26	['primary', 'agriculture', 'road', 'partly_cloudy']
test_27	['primary', 'clear']
test_28	['primary', 'agriculture', 'road', 'haze']
test_29	['primary', 'clear']
test_30	['primary', 'partly_cloudy']
test_31	['cloudy']
test_32	['primary', 'clear']
test_33	['primary', 'haze']
test_34	['cloudy']
test_35	['primary', 'clear']
test_36	['primary', 'clear']
test_37	['primary', 'clear']
test_38	['primary', 'clear']

	image_name	tags
0	test_0	[primary, clear]
1	test_1	[primary, clear]
2	test_2	[primary, partly_cloudy]
3	test_3	[primary, partly_cloudy]
4	test_4	[primary, partly_cloudy]
5	test_5	[primary, clear]
6	test_6	[primary, agriculture, partly_cloudy]
7	test_7	[primary, agriculture, habitation, road, clear]
8	test_8	[primary, clear]
9	test_9	[primary, haze]
10	test_10	[primary, partly_cloudy]
11	test_11	[primary, clear]
12	test_12	[partly_cloudy]
13	test_13	[primary, clear]
14	test_14	[primary, clear]
15	test_15	[primary, clear]
16	test_16	[primary, agriculture, habitation, road, clear]
17	test_17	[primary, partly_cloudy]
18	test_18	[primary, haze]
19	test_19	[primary, clear]
20	test_20	[primary, water, artisinal_mine, clear]
21	test_21	[primary, agriculture, partly_cloudy]

figure 19 two csv open forms