Project 2

Step 1: Read Data

Read data from train.txt and val.txt.

```
train = pd.read_csv('train.txt', sep=' ')
train.columns = ['path', 'label']
val = pd.read_csv('val.txt', sep=' ')
val.columns = ['path', 'label']
```

We get the path and label by reading train.txt and val.txt

	path	label
0	./flower_photos/daisy/8446495985_f72d851482.jpg	0
1	./flower_photos/tulips/6931748252_68f06086b3.jpg	4
2	./flower_photos/dandelion/4571923094_b9cefa943...	1

Step 2: Transform image to matrix

Transform image to matrix by using imread model from scipy.misc

```
img_matrix = imread(train.iloc[0, :].path)
img_label = train.iloc[0, 1]
```

I print the img_matrix of the first image and get the height, width and depth are (333,500,3) respectively.

```
img_matrix.shape
```

```
(333, 500, 3)
```

But it is not the regular size that we need, the traditional detection algorithm is that fixing input image, use different sizes of sliding window to support different scales of objects. For CNN, the size of the sliding window is the size of the input image during training and it can't be changed. The approach CNN supports multi-scale is to fix the sliding window size, change the size of the input image. Specifically, for a given image to be processed resize the image to the corresponding scale.

Step 3: Define height, width and depth

Define resize function， resize the image to fixed size， which can make the train and test data have equal size.

```
height, width, depth = imresize(img_matrix, (224, 224)).shape
```

Define height width and depth： height:224 width:224 depth:3 respectively.

```
height,width,depth
```

```
(224, 224, 3)
```

Step 4: Number of train, validation and classes.

Calculate the number of train, validation and classes.

```
train_number = len(train)
val_number = len(val)
classes_number = len(np.unique(train.label.values))
```

They are 2569, 550 ,5 respectively

Step 5: Assign values

Assign variables to y_train and y_val

```
y_train = train.label.values.reshape((-1, 1))
y_val = val.label.values.reshape((-1, 1))
```

```
y_train
```

```
array([[0],
       [4],
       [1],
       ...,
       [1],
       [4],
       [4]], dtype=int64)
```

Read data by line and transform the format of matrix to the model need

```
for i in range(train_number):
    img_matrix = imread(train.iloc[i, :].path)
    img = imresize(img_matrix, (224, 224))
    x_train = (i,img)
```

```
x_train.shape
```

```
(2569, 224, 224, 3)
```

Step 6: Create the InceptionV3 model and pre-train the model

I chose InceptionV3 from kears

```
# create the base pre-trained model
input_tensor = Input(shape=(224, 224, 3))
base_model = InceptionV3(input_tensor=input_tensor,weights='imagenet', include_top=False,)

# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(64, activation='relu')(x)
# and a logistic layer -- let's say we have 200 classes
predictions = Dense(5, activation='Adagrad')(x)

# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)
for layer in base_model.layers:
    layer.trainable = False
model.compile(optimizer='Adagrad',
```

```
                loss='categorical_crossentropy',
                metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=64, nb_epoch=20, validation_split=0.2
    )
```

Step 7: Predict accuracy of training set and validation set.

```
Epoch 19/20
2055/2055 [==============================] - 725s 353ms/step - loss: 0.1488 - acc: 0.9489 - val_loss: 0.3947 - val_acc: 0.8891
Epoch 20/20
2055/2055 [==============================] - 709s 345ms/step - loss: 0.1364 - acc: 0.9567 - val_loss: 0.3327 - val_acc: 0.8852
```

The accuracy is 88.52%.

Step 8: Save InceptionV3 model.

```
from keras.models import load_model

model.save('my_model.h5')
# creates a HDF5 file 'my_model.h5'

# returns a compiled model
# identical to the previous one
model = load_model('my_model.h5')
```

Step 9: Call the model and make predictions

1.Process test.txt

```
test = pd.read_table('test.txt',header=None, sep=' ',encoding='gb2312')
test.columns = ['path']
```

2.Transform test set as training set to regular format we need
3.Call the model and make predictions
4.In the end ,the shape of prediction is(550,5), that’s means that we got a prediction probability, and the maximum is the result we want to get.

```
    result = np.argmax(prediction,axis = 1)
```

And the we print the result out.

```
np.savetxt('result.txt', result, fmt="%d",delimiter=",")
```