Aerial Cactus Identification

Author: Xiangxin She, ECE Department, Purdue University Northwest

Abstract

The project can be used to conserve the environment. We found this Aerial Cactus Identification competition on Kaggle with relevant data sets. In this work, we use a Deep Learning approach to detect cactus. The project uses CNN for the detection of the cactus through image processing. In this project we use keras and vgg16 respectively for the detection of cactus. On both models, we achieved almost 98% accuracy, and both models can be used to make predictions about other organisms or species.

Keywords: Convolutional neural network, Deep learning, Keras, Vgg16, Cactus

Introduction

Tehuacán-Cuicatlán Valley is a semi-arid zone in the south of Mexico. It was inscribed in the World Heritage List by UNESCO in 2018. This unique area has wide biodiversity including several endemic plants. Unfortunately, human activity is constantly affecting the area. A way to preserve a protected area is to carry out autonomous surveillance of the area. A step to reach this autonomy is to automatically detect and recognize elements in the area.

Dataset

Since this project is a competition started by Kaggle, so the dataset we chose is from Kaggle too and the link of the dataset is https://www.kaggle.com/c/aerial-cactus-identification/data. This dataset contains a large number of 32 x 32 thumbnail images containing aerial photos of a columnar cactus (Neobuxbaumia tetetzo). Kaggle has resized the images from the original dataset to make them uniform in size. The file name of an image corresponds to its `id`. Figure 1 shows the dataset on Kaggle website.

Figure 1. dataset on Kaggle

The train.zip contains 17500 32 x 32 thumbnail images. Figure 2 shows one of these images.

Figure 2. sample of images

The train.csv matches each image with the ground truth. “1” stands for that there is cactus in that image and “0” stands for none. Figure 3 shows part of the train.csv file.

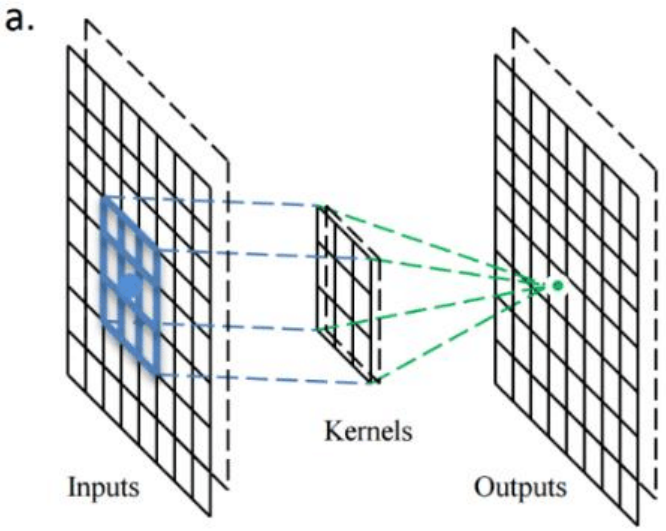
	A	B
1	id	has_cactus
2	0004be2cfeaba1c0361d39e2b000257b.jpg	1
3	000c8a36845c0208e833c79c1bffedd1.jpg	1
4	000d1e9a533f62e55c289303b072733d.jpg	1
5	0011485b40695e9138e92d0b3fb55128.jpg	1
6	0014d7a11e90b62848904c1418fc8cf2.jpg	1
7	0017c3c18ddd57a2ea6f9848c79d83d2.jpg	1
8	002134abf28af54575c18741b89dd2a4.jpg	0
9	0024320f43bdd490562246435af4f90b.jpg	0
10	002930423b9840e67e5a54afd4768a1e.jpg	1

Figure 3. part of the train.csv file

Methods

Since convolutional neural networks (CNNs) have achieved state-of-the-art results on real-world applications such as image classification [He et al., 2016] and object detection [Ren et al., 2015], we decided to use CNN as our machine learning algorithm for this project.

Convolution neural network algorithm is a multilayer perceptron including input layer, convolution layer, sample layer and output layer. The CNN architecture is built layer by layer. Each convolutional layer contains a series of filters known as convolutional kernels. These filters introduce translation invariance and parameter sharing and get the output feature map as figure 4 shows.The Max Pooling will do the descending sampling process job and halve the length and width of the feature map.



A stride one 3x3 convolutional kernel acting on a 8x8 input image, outputting an 8x8 filter/channel. Source: https://www.researchgate.net/figure/a-Illustration-of-the-operation-principle-of-the-convolution-kernel-convolutional-layer_fig2_309487032

Figure 4. overview of a CNN architecture
(source:https://link.springer.com/content/pdf/10.1007/s13244-018-0639-9.pdf)

A model's performance under particular kernels and weights is calculated with a loss function through forward propagation on a training dataset, and learnable parameters, i.e., kernels and weights, are updated according to the loss value through backpropagation with gradient descent optimization algorithm. Figure 5 shows an overview of a convolutional neural network (CNN) architecture and the training process.

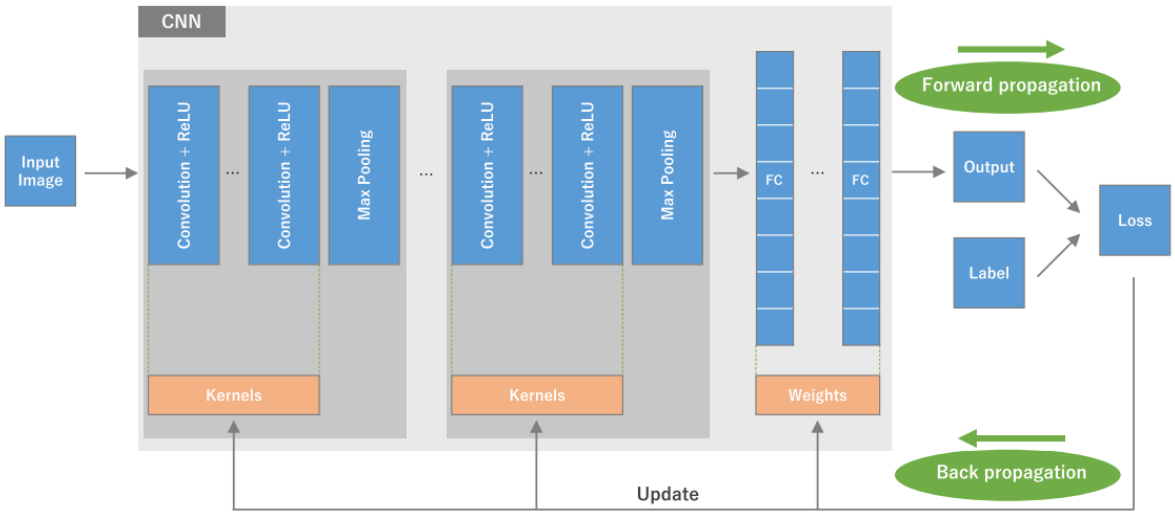


Figure 5. creating a feature map from a convolutional kernel

For the implementation, we used two method, one is Keras, and the other is VGG16. Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling

fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. We directly use keras library with tensorflow running backend to build our convolutional neural network.

-VGG16

VGG is a convolutional neural network model，the microarchitecture of VGG16 can be seen in Figure 6.

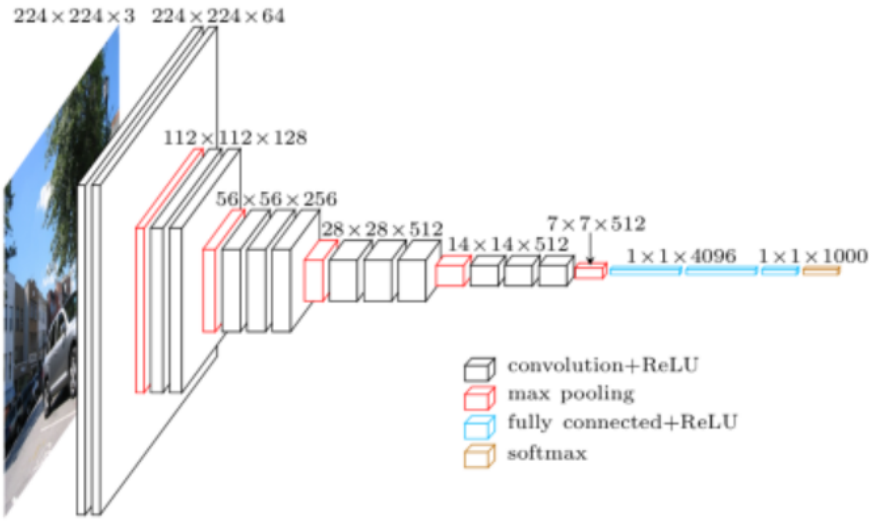


Figure 6. microarchitecture of VGG16

From left to right, a picture is inputted to the network. The white box is the convolution layer, the red is the pooling layer, the blue is the fully connected layer, and the brown box is the prediction layer. The function of the prediction layer is to convert the output information of the whole connection layer into the corresponding category probability, which plays the role of classification. VGG16 is composed of 13 convolutional layers and 3 fully connected layers.

-Transfer learning.

When the data set is not large enough to train the entire CNN network, it is usually possible to adapt the pre-trained imageNet network (such as VGG16) to the new task.

In general, there are two types of transfer learning:feature extraction and fine-tuning.In this project, we use the second one to train the model.We change the structure of pre-training model, removing the fully connection layer and adding our own defined fully connection layer.

Experiment/Result/Discussion

To begin with, we first associate image data and its ground truth with two numpy a rray from train.csv and “train” folder by creating a function.

```
def load_data(df):
```

```
train_x, train_y = load_data(training_df)
```

Figure 7.get training set

Then we did normalization to the training dataset.

```
train_x = train_x.reshape(17500,32*32*3)
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit_transform(train_x)
```

Figure 8. n normalization

Since this project is a competition, so we did not have the test set with the grou nd truth. Thus, we seperate a validation set from the training set to evaluate our model.

```
train_x = train_x.reshape(17500,32,32,3)
```

```
x_train, x_valid, y_train, y_valid = train_test_split(
...     train_x, train_y, test_size=0.1, random_state=42)
```

Figure 9. create vaidation set

After the dataset preparation, we can build our network to train the model. First we directly applied the cnn from the HW7 assignment and the performance is bad.

```
score = CNN.evaluate(x_valid, y_valid, verbose=0)
print("loss:",score[0])
print("accu:",score[1])
```

```
loss: 3.7905249900817872
accu: 0.75142854
```

Figure 10. hw7 cnn performance

Then we changed the network structure and modified some parameters such as learni ng rate and dropout layer value. (source:https://www.kaggle.com/frlemarchand/simple-cnn-using-keras/notebook)

```
model.compile(loss=keras.losses.binary_crossentropy,
              optimizer=keras.optimizers.Adam(lr=3e-02),
              metrics=['accuracy'])
```

```
score = model.evaluate(x_valid, y_valid, verbose=0)
print("loss:",score[0])
print("accu:",score[1])
```

```
loss: 0.13834960545812333
accu: 0.9474286
```

Figure 11. first time learning rate:3e-02, and result accuracy

```
score = model.evaluate(x_valid, y_valid, verbose=0)
print("loss:",score[0])
print("accu:",score[1])
```

```
loss: 0.06041610588559083
accu: 0.97485715
```

Figure 12. second time learning rate:0.01, and result accuracy

```
score = model.evaluate(x_valid, y_valid, verbose=0)
print("loss:",score[0])
print("accu:",score[1])
```

```
loss: 0.029557557436
accu: 0.993143
```

Figure 13. third time learning rate: 0.001, and result accuracy

Since VGG16 is a pre-trained model, we do not need to consider the parameters for setting the model. Here are the steps for training and predicting.

First, we need to load the pre-trained modelthat does not contain the fully connection layer.

```
vgg = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
vgg.trainable = False
```

Figure 14. load VGG16

Second, create the CNN for training data and add fully connection layer.

```
model = Sequential()
model.add(vgg) # to load the pre trained model
model.add(Flatten())# to convert two dimensional array into a vector
model.add(Dense(256))# a fully connected layer
model.add(Activation('relu'))
model.add(Dropout(0.5)) # helps in removing the overfitting from the layer to make it mor
model.add(Dense(1)) #fully connected layer which gives a single output
model.add(Activation('sigmoid')) # activation functio for making a decision
```

Figure 15. create CNN

Third, compile the model architecture.

```
model.compile(loss='binary_crossentropy',optimizer=Adam(lr=0.01),metrics=['accuracy'])
```

Figure 16. compile the model

Forth, train the model, set the batch size and epoch.

Because the data sets are large, we choose the smaller epoch, which I set to 10.

```
history = model.fit(x_train, y_train,batch_size=32, epochs=10, validation_split=0.1, shuffle=True, verbose=2)
```

Figure 17. train the model

```
score = model.evaluate(x_valid, y_valid, verbose=0)
print("loss:", score[0])
print("accu:", score[1])
```

```
loss: 0.08911195290940148
accu: 0.9702857136726379
```

Figure 18. Loss and accuracy of the model

For same learning rate (lr =0.1), we set the batch size to 32 and 64 respectively and obtain different loss and accuracy. For batch size =32, loss ≈0.69 and accuracy ≈0.76 (Figure 19 and Figure 20). For batch size = 64, loss ≈ 0.36 and accuracy ≈ 0.80. The two accuracy rates are not very high, so I change the learning rate to 0.01. Finally, the accuracy ≈ 0.97 and loss ≈ 0.09.

```
model.compile(loss='binary_crossentropy',optimizer=Adam(lr=0.1),metrics=['accuracy'])
#optimizer:deciding the weights for the network
```

```
history = model.fit(x_train, y_train,batch_size=32, epochs=10, validation_split=0.1, shuffle=True, verbose=2)
```

Figure 19. learning rate=0.1, batch size=32

```
score = model.evaluate(x_valid, y_valid, verbose=0)
print("loss:", score[0])
print("accu:", score[1])
```

```
loss: 0.6946629463604518
accu: 0.7571428418159485
```

Figure 20. cooresponding loss and accuracy

We also set the batch size =64 and learning rate=0.01, we got loss ≈ 1.45 and accuracy ≈ 0.75(Figure 19 and Figure 20). So, the best choice is to set batch size = 32, learning rate = 0.01.

```
model.compile(loss='binary_crossentropy',optimizer=Adam(lr=0.01),metrics=['accuracy'])
#optimizer:deciding the weights for the network
```

```
history = model.fit(x_train, y_train,batch_size=64, epochs=10, validation_split=0.1, shuffle=True, verbose=2)
```

Figure 21. learning rate=0.01,batch size=64

```
score = model.evaluate(x_valid, y_valid, verbose=0)
print("loss:", score[0])
print("accu:", score[1])
```

```
loss: 1.4499272661209106
accu: 0.7548571228981018
```

Figure 22. corresponding loss and accuracy

Fifth, predict the test data. And store the predictions in a csv file.

```
final_df = pd.DataFrame(predictions, columns=['has_cactus'])
#Assigning 1 to values greater than 0.75 or else 0 best on the prediction made
final_df['has_cactus'] = final_df['has_cactus'].apply(lambda x: 1 if x > 0.75 else 0)
final_df.head()
```

	has_cactus
0	1
1	1
2	0
3	1
4	1

Figure 23. predict the test data.

Conclusion

In this work, we accomplish Aerial Cactus Identification by using deep learning algorithm. We mainly apply the algorithm of convolution neural network to excavate the deep information of multi-layer network in the process of object identification. We implement two methods to compare the difference between them. Experimental results show that we have achieved good results. For keras method we get 99% accuracy and for VGG16 transfer learning we get 97% accuracy. Actually the same network can yield different results due to the shuffle mode, so we should train the model several time and get the average accuracy. In addition, there are still many parameters that can be fine-grained in the algorithm.Those two factors will be the focus for us in the future to continue to improve the work.

References

[1]. Lin, Xiaofan, Cong Zhao, and Wei Pan. "Towards accurate binary convolutional neural network." In *Advances in Neural Information Processing Systems*, pp. 345-353. 2017.

[2]. Yamashita, Rikiya, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. "Convolutional neural networks: an overview and application in radiology." *Insights into imaging* 9, no. 4 (2018): 611-629.

[3]. Liu, Tianyi, Shuangsang Fang, Yuehui Zhao, Peng Wang, and Jun Zhang. "Implementation of training convolutional neural networks." *arXiv preprint arXiv:1506.01195* (2015).