Object Images Detection with Deep Learning CNN

Design Specification

Hanan Jalnko

Lawrence Technological University

Computer Science

Spring 2018

1 Introduction

1.1 Overview

Object detection is considered to be the most basic application of computer vision. Rest of the other developments in computer vision are achieved by making small enhancements on top of this. In real life, every time we(humans) open our eyes, we unconsciously detect objects. Deep learning allows computational models of multiple processing layers to learn and represent data with multiple levels of abstraction mimicking how the brain perceives and understands multimodal information. Deep learning is a specific subset of Machine Learning, which is a specific subset of Artificial Intelligence. The power of artificial intelligence is beyond our imagination. Moreover, Autonomous cars have been a topic of increasing interest in recent years as many companies are actively developing related hardware and software technologies toward fully autonomous driving capability with no human intervention. Deep Learning is one way of doing that, using a specific algorithm called a Neural Network it also powers some of the most interesting applications in the world, The core to many of these applications are visual recognition tasks such as image classification, localization and detection. The main difference between machine learning and deep learning is the depth to which the system can autonomously teach itself. When machine learning uses features from input (from training data) and makes predictions based on a single or a few layers of nodes, a deep neural network contains many hidden layers that adds new features and exceeds human coding capacity. This makes deep learning more powerful for complex computing tasks such as object recognition. Noteworthy is the improvement of deep learning using a convolutional neural network, where the input is the whole image and thus embeds feature extraction. Its mapping between features and actions is established during training.

2. Preparing the Input (the training data):

Our dataset contains 398 images for objects classified as (Blue cup, Bottle, Mallet, Hammer, Tape, Scissors Red, Scissors Black, Mug, Pliers and Spray Bottle) containing two subsets: a training set with 300 samples and a validation set with 98 samples witch has been used for training our model and finally a test set with 10 unseen samples for prediction. Some image as shown in Fig 1.

Fig 1. Show some image used

3 Methods

I used pre-trained models in order to help me in the experiment as shown in Diagram 1. The pre-trained models are trained on very large scale image classification problems. The convolutional layers act as feature extractor and the fully connected layers act as Classifiers.with 2D convolutional layers on a pre-trained model where the first layers are blocked from training and then trained the last two layers with the small data. The strategy is to not only replace and retrain the classifier on top of the ConvNet on the new dataset, but to also fine-tune the weights of the pretrained network by continuing the backpropagation. It is possible to fine-tune all the layers of the ConvNet, or it's possible to keep some of the earlier layers fixed (due to overfitting concerns) and only fine-tune some higher-level portion of the network. This is motivated by the observation that the earlier features of a ConvNet contain more generic features (e.g. edge detectors or color blob detectors) that should be useful to many tasks, but later layers of the ConvNet becomes progressively more specific to the details of the classes contained in the original dataset.

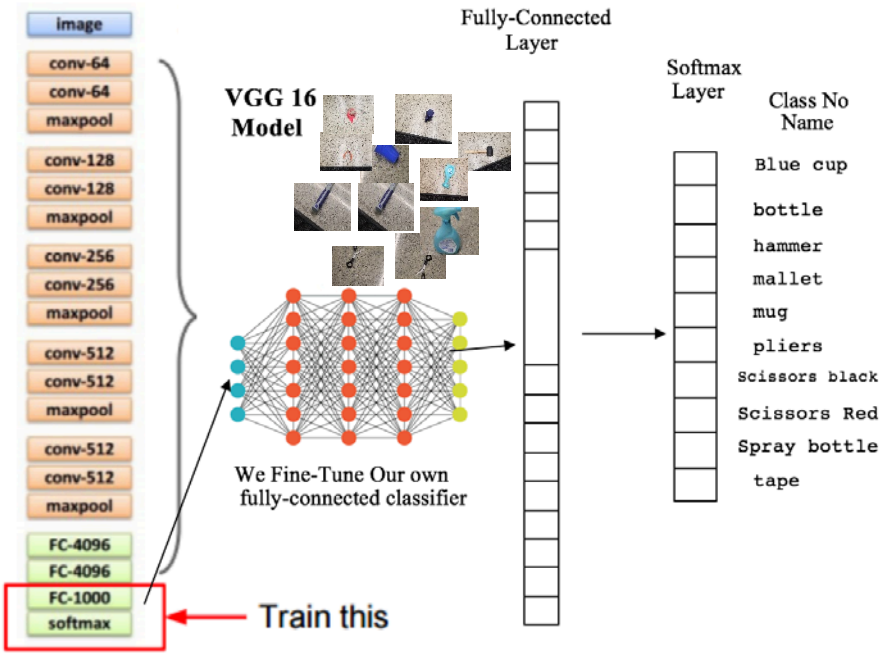


Diagram 1. Show The Processing

3.1 Load the pre-trained model

The VGG16 model was developed by Karen Simonyan and Andrew Zisserman in 2014, a simple and widely used convnet architecture for ImageNet. Although it is a bit of an older model, far from the current state of the art and somewhat heavier than many other recent models, we chose it because its architecture is similar to what you are already familiar with. among others, comes pre-packaged with Keras. We can import it from the keras.applications module. This model is trained on more than a million images and can classify images into 1000 object categories. For example, keyboard, mouse, pencil, and many animals. As a result, the model has learned rich feature representations for a wide range of images. It has 13 convolutional layers followed by rectification and pooling layers, and 3 fully connected layers. All convolutional layers use small 3 × 3 filters and the network performs only 2 × 2 pooling. VGG-16 has a receptive field of size 224 × 224.

```
import keras
keras.__version__

from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))
```

In the above code, we load the VGG16 Model. There is one change – include_top=False. We have not loaded the last two fully connected layers which act as the classifier. We are just loading the convolutional layers.

3.2 Extract Features

Our data is divided into 80:20 ratio and kept in separate train and validation folders. Each folder should contain 10 folders belonging to the respective classes.

```
img_width, img_height = 150, 150

train_dir = '/vccmain1/test'
validation_dir = '/vccmain1/val'
test_dir = '/vcctest1/'
train_tape_dir = '/vccmain1/test/tape'
nb_train_samples = 300
nb_validation_samples = 98
batch_size =16
```

It should be noted that the last layer has a shape of 4 x 4 x 512.To look at how the dimensions of the feature maps change with every successive layer `conv_base.summary()`

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 150, 150, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
```

```
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

The convolutional base of VGG16 has 14,714,688 parameters, which is very large. The classifier we are adding on top has 2 million parameters.

Before we compile and train our model, a very important thing to do is to freeze the convolutional base. "Freezing" a layer or set of layers means preventing their weights from getting updated during training. If we don't do this, then the representations that were previously learned by the convolutional base would get modified during training. Since the Dense layers on top are randomly initialized, very large weight updates would be propagated through the network, effectively destroying the representations previously learned. We will fine-tune the last 3 convolutional layers, which means that all layers up until block4_pool should be frozen, and the
layers block5_conv1, block5_conv2 and block5_conv3 should be trainable.

```
conv_base.trainable = True

set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

For our compilation step, I used the RMSprop optimizer. And we ended our network mod el to predict multiple choices, the loss function to use in this case is categorical_crossentr opy. It measures the distance between two probability distributions: in our case, between the probability distribution output by our network, and the true distribution of the labels. By minimizing the distance between these two distributions, we train our network to outp ut something as close as possible to the true labels

```
from keras import optimizers
model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

3.3 Data preprocessing

The data should be formatted into appropriately pre-processed floating point tensors before being fed into our network. Decode the JPEG content to RBG grids of pixels. Convert these into floating point tensors. Rescale the pixel values (between 0 and 255) to the [0, 1] interval (as you know, neural networks prefer to deal with small input values). Keras has utilities to take care of these steps automatically. Keras has a module with image processing helper tools, located at keras.preprocessing.image. In particular, it contains the class ImageDataGenerator which allows to quickly set up Python generators that can automatically turn image files on disk into batches of pre-processed tensors. This is what I will use here.

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(zoom_range = 0.1,
                  rescale=1./255,
                  height_shift_range = 0.1,
                  width_shift_range = 0.1,
                  rotation_range = 10)
test_datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(
             train_dir,
            target_size=(img_width, img_height),
            batch_size=batch_size,
             class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
            validation_dir,
            target_size=(img_width, img_height),
             batch_size=batch_size,
            class_mode='categorical')
```

3.4 Create the model

I will create a simple feedforward network with a softmax output layer having 4 classes.

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import models
from keras import layers

if K.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)
else:
    input_shape = (img_width, img_height, 3)

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

Now we can start training our model, with the data augmentation configuration

```
from keras import models
history=model.fit_generator(
        train_generator,
        steps_per_epoch=nb_train_samples // batch_size,
        epochs=30,
        validation_data=validation_generator,
        validation_steps=nb_validation_samples // batch_size)
```

```
Epoch 1/20
18/18 [==============================] - 158s - loss: 1.3797 - acc: 0.5243 - val_loss: 0.7154 - val_acc: 0.7750
Epoch 2/20
18/18 [==============================] - 139s - loss: 0.3092 - acc: 0.9201 - val_loss: 0.4942 - val_acc: 0.8378
Epoch 3/20
18/18 [==============================] - 138s - loss: 0.0923 - acc: 0.9826 - val_loss: 0.5334 - val_acc: 0.8514
Epoch 4/20
18/18 [==============================] - 141s - loss: 0.1956 - acc: 0.9340 - val_loss: 0.4498 - val_acc: 0.8514
Epoch 5/20
18/18 [==============================] - 141s - loss: 0.1511 - acc: 0.9506 - val_loss: 0.3177 - val_acc: 0.9054
```

```
Epoch 18/20
18/18 [==============================] - 138s - loss: 0.0895 - acc: 0.9826 - val_loss: 1.0316 - val_acc: 0.8649
Epoch 19/20
18/18 [==============================] - 143s - loss: 0.0350 - acc: 0.9896 - val_loss: 0.2065 - val_acc: 0.9625
Epoch 20/20
18/18 [==============================] - 140s - loss: 0.0150 - acc: 0.9965 - val_loss: 1.1670 - val_acc: 0.8514
```

In order to save the model for future classification and prediction

```
model.save('history.h5')
```

4 Results

We quickly reach an accuracy of (85.0%) on the training data. Now let's check that our model performs well on the test set too:

```
test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(150, 150),
        batch_size=2,
        class_mode='categorical')

test_loss, test_acc = model.evaluate_generator(test_generator, steps
=50)
print('test acc:', test_acc)
```

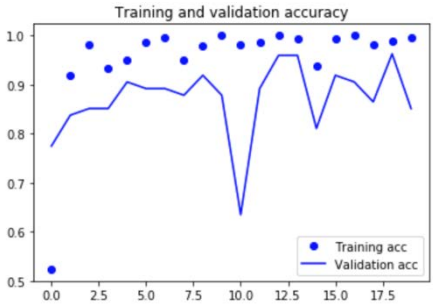
```
test acc: 0.85
```

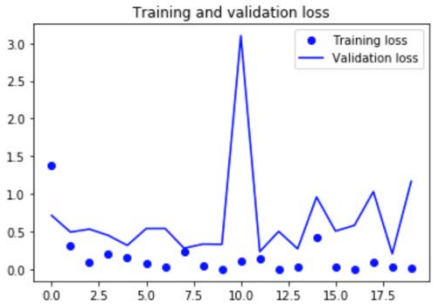
```
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





4.1 Check Performance(Prediction)

We will use unseen images and would like to see if the model can get the right classification for them I upload new 6 images and here is the result.

```
1/1 [==============================] - 0s
1/1 [==============================] - 0s
[[ 0.  0.  0.  0.  0.  0.  0.  0.  1.  0.]]
[8]
1.spray_bottle
```

```
1/1 [==============================] - 0s
1/1 [==============================] - 0s
[[  6.33273075e-27   1.72481985e-18   1.98248066e-23   1.00000000e+00
    2.04014205e-17   3.42009068e-24   2.37824812e-14   7.26928685e-22
    2.10985057e-21   1.51609072e-20]]
[3]
1.mallet
```

```
1/1 [==============================] - 0s
1/1 [==============================] - 0s
[[ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
[0]
1.blue_cup
```

```
[==============================] - 0s
[==============================] - 0s
2.36774333e-25   8.40487816e-20   1.05336717e-20   4.06102303e-17
7.43576502e-16   1.09484153e-17   1.00000000e+00   2.10965493e-16
2.51397293e-21   6.77969781e-23]]

issors_black
```

```
1/1 [==============================] - 0s
1/1 [==============================] - 0s
[[  6.03944550e-17   5.72853982e-30   1.24902681e-26   3.61144551e-21
    1.00000000e+00   9.43729339e-26   2.89014563e-21   2.27327526e-28
    6.27205357e-18   2.47774249e-24]]
[4]
1.Mug
```

5 Conclusions

The results of this project show that Fine tune and deep learning are a viable approach to image analysis there are many exciting research directions that transfer learning offers and in particular many applications that are in need of models that can transfer knowledge to new tasks and adapt to new domains. we choose to only fine-tune the last convolutional block rather than the entire network in order to prevent overfitting, since the entire network would have a very large entropic capacity and thus a strong tendency to over fit. The features learned by low-level convolutional blocks are more general, less abstract than those found higher-up, so it is sensible to keep the first few blocks fixed (more general features) and only fine-tune the last one (more specialized features). Overall, the system gates the right prediction for the amount of epochs we used and only minimal data was proved.

6 Acknowledgments

This work and the images was supported by Professor Dr. CJ Chung. Special thanks for his help and support.

References

1. Chollet, Francois. Deep Learning with Python. ser. 1617294438, 9781617294433, Manning Publications Co., 2018.
2. VGG16 ModelKaren, Simonyan, and Zisserman Andrew. "Very Deep Convolutional Networks for Large-Scale Image Recognition." [1409.1556] Very Deep Convolutional Networks for Large-Scale Image Recognition, 10 Apr. 2015, arxiv.org/abs/1409.1556.
3. Ciresan, D., Meier, U. Masci, J. & Schmidhuber, J. Multi-column deep neural network for traffic sign classification. Neural Networks 32, 333–338 (2012).