Hack A Drone
Hands On Lab Guide

November 2, 2017

Contributers: Remco Runge, Ronald van Broeckhuijsen, Ömer Ergul, Robert Horvers, Evert Poots

Contents

1 Introduction

Welcome to the Ordina JTech Hack A Drone Hands On Lab! During this Hands On Lab you will learn to control your very own drone with the help of Java. At the end of this session your drone will be able to use his autopilot to fly predefined routes and your drone will be able to recognize objects in the world around him with his camera and a Deep Learning network.

2 Requirements

The requirements for this workshop are as follows:

- A drone
- Laptop with the following software installed:
  - Latest Java
  - Latest Maven
  - Your favorite IDE

2.1 Drone

A short overview of the system specifications of the drone. Fly time and charge time might very a bit per drone. Note that the propellers can be replaced if damaged.
The hardware of the drone is as followed:

- 2.4 GHz WiFi Remote Control
- 3.7 Volt 150 mAh battery
- 0.3 Mega Pixel Camera (640x480)
- Weight 18 gram
- Flying time 4-5 minutes
- Charging time 20-30 minutes

3 Set Up

The Java code can be acquired from the GitHub: `https://github.com/Ordina-JTech/hack-a-drone.git` or from the usb distributed during the 2017 JFall conference.

3.1 WiFi connection Drone

The drone can only be connected via WiFi. No more than one user can be connected to the drone at the same time.

- Turn the drone on
- Connect to the WiFi of the drone
  - The drone can be recognized this pattern for its SSID: "CX-10WD-******". The drones handed out during JFall are labelled with their SSID (inside the box).
  - Set the drone close the laptop for easy detection
- Start the Java-code by running the main class **GUI.java** in *nl.ordina.jtech.hackadrone.gui*

4 Challenges

4.1 Challenge 1

In this first challenge, you are gone take manual control over the drone with your keyboard. Use the controls as defined in Keyboard.java (nl.ordina.jtech.hackadrone.io) and try to fly the drone.

- Use the left arrow to take off
- Use the right arrow to land
- Use the up arrow to increase the height during flight
- Use the down arrow to decrease the height during the flight
- Use the W key to go forward
- Use the S key to go backwards
- Use the A key to go left
- Use the D key to go right
- Use the Q key to turn left
- Use the E key to turn right

4.2 Challenge 2

Manual control is great off course, but wouldn't it be better to have a drone which is able to fly a predefined route all by itself? In this challenge you need to write a functional Auto Pilot.

Edit the *start()* method in the class **AutoPilot.java** (package: *nl.ordina.jtech.hackadrone.core.io*) to let the drone fly your very own route. In order to get you started we have already defined an example route for taking off, flying in a straight line, turning 180 degrees, flying back and landing.
Tips:

- Use the *chill(int ms)* method to make sure that the drone gets time to level out after moving in a direction.
- Make sure to always include a *takeOff()* and *land()* command. We do not want the drones to get out of control.
- Start by using low millisecond (ms) numbers as parameters to prevent the drone from overshooting its target.

4.3 Challenge 3

Now that your autopilot is complete, wouldn't it be nice if the drone was able to recognise the world around itself? Within this challenge you are going to use a Deep Learning network, to give the drone the ability to recognize objects. If you are new to Deep Learning, you can read chapter 6.

Normally a drone would have to train itself, to recognise the world around it and create a data model accordingly. However, for this challenge we will use a pre-trained data model. For Part 1 we will first load the existing data model into the Deep Learning network. The model we are using is the VGG16 network which was trained created by the Oxford University in 2016. This network has been trained the ImageNet dataset has millions of labelled images.

Within this challenge, we are gone use Deeplearning4j as our Java Deep Learning framework. Which is an open-source, distributed Deep Learning Libray for the JVM.

4.3.1 Challenge 3 - Loading the model

Download the data model (VGG16.zip) from the following link (if you have not done this already): `https://drive.google.com/open?id=0B1kt4UHsfEYAWU44RlZZUllLT0E`

The pre-trained data model can be loaded in the **DeepLearning.java** class in the method *void loadDataModel()* by using the following syntax, surrounded by a *try/catch* block for the *IOException* it might throw.

```
computationGraphVGG16 = ModelSerializer.restoreComputationGraph("deeplearning/VGG16.zip");
```

The *restoreComputationGraph(String src)* method restores the weights of the trained VGG16 network.

4.3.2 Challenge 3 - Pre-processing the image

To be able to classify the images of the drone, they need to be pre-processed. The pixel data of the image has to be converted to a matrix which can be used as input for the network. For part 2 of this challenge we are going to implement the image processing. The first processing step is to save the image as a matrix which can be used to activate the input nodes of the network. Add the following code to the
INDArray processImage(BufferedImage bufferedImage) method.

```
INDArray imageMatrix = nativeImageLoader.asMatrix(bufferedImage);
```

Again, we have to surround it by a try-catch block, because it might throw an IOException. Each element of the created matrix is used as the input for a single input node in the network.

4.3.3 Challenge 3 - Classify

In part 3 of the Deep Learning challenge, we can start classifying the objects that the drone sees. In order to classify the image we can use the method

```
INDArray[] output(boolean train, INDArray... input)
```

of the *ComputationalGraph* from deeplearning4j. This fist parameter of this method determines whether the network needs to be trained. Since we have already trained the network, we can set the boolean to *false*. The second parameter is an *INDArray* containing the pre-processed image. Add the following code to the *classify(BufferedImage bufferedImage)* method:

```
INDArray[] output = computationGraphVGG16.output(false, processedImage);
```

This method uses the image to activate the network and generate a prediction for each possible image class. The *INDArray[]* that is returned contains an array of arrays of the output nodes of the neural network and a score based on the activation of the node. The nodes and their activation represent the chance the input image is of a certain class according to the network. Since we called the method with just one image, the size of the *INDArray[]* is 1.

4.3.4 Challenge 3 - Ranking the predictions

The method:

```
List<Prediction> decodePredictions(INDArray predictions)
```

is used to rank and identify all of the predictions returned by our classification method. In this method the list of activated output nodes is mapped to a jsonmap of objects, in which each node represents a learned object.

For example, if output node 5 of the neural network has the highest activation, and node 5 represents a water bottle in the jsonmap. Then the image most likely contains a water bottle.

Add the following code to the *classify(BufferedImage bufferedImage)* method:

```
List<Prediction> predictions = decodePredictions(output[0]);
```

Now you are done! Run the application and press the start Deep Learning button to see how well your drone can see the world!

5 Thanks

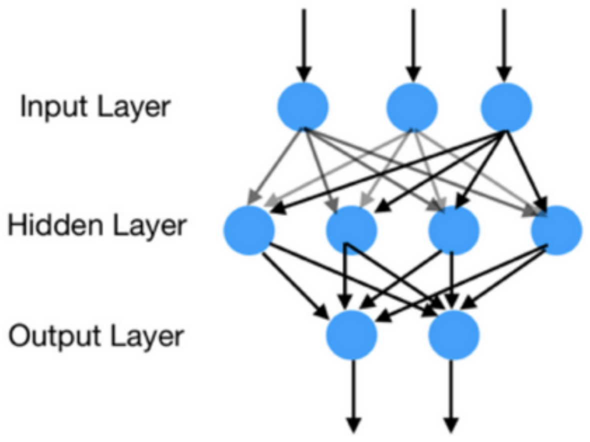
Thanks for participating in this Ordina JTech Hands On Lab! If you liked this Hands On Lab please rate us in the JFall App!

6 Deep Learning - A short introduction

Our brain is a very powerful machine which is able to perform a lot off tasks very efficiently while computers struggle at these tasks. Example of these tasks are recognising objects in the world around us, understanding speech and speaking itself. Deep Learning was developed to give computers these kinds of abilities.

The base of deep learning is the Artificial Neural Network (ANN). The workings of a ANN are loosely based on the neural networks in our head. Just like our brain acts on input from our senses, an ANN will also act on bases of a certain input.

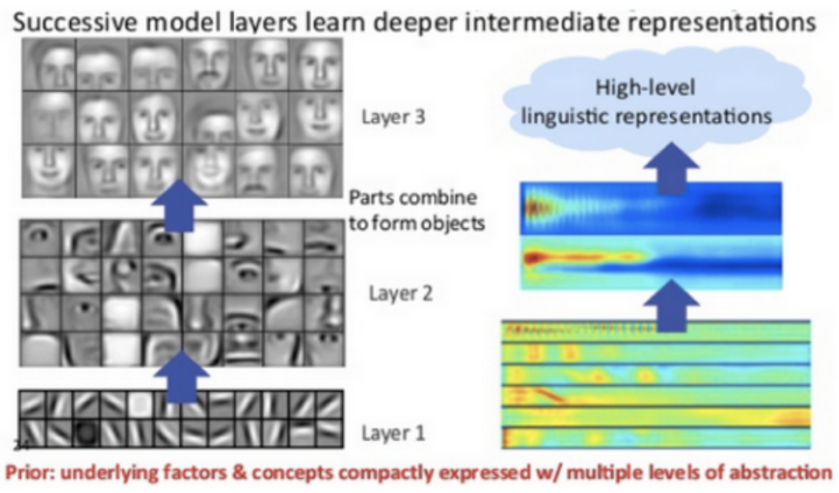
An ANN is composed of nodes. Within these nodes, the computation takes place. As can be seen in the image the first layer of a ANN consists of the input nodes. These nodes are activated by the input of the network. The output of these nodes are fed to the next layer called the hidden layers. The output of the last hidden layer is then used as input for the final layer of the network, called the output layer.



For each node, all the inputs are combined based on a weight for each connection. An activation function then determines the output of that neuron. By adjusting the weights of the connections, the network is able to learn. The learning process of the network is called the training phase. During this phase, the network is presented with a dataset filled with trainings data. This this trainings data consists of the input, as well as the desired output. For example, when presented with image data of cats and dogs, the input neurons are activated by the pixel data, and the output neurons represent the classes (one output neuron for cats and one for dogs).

When the trainings data is presented, the network will determine its output. By comparing the output of the network to the desired output, the network can adjust its weight to decrease the difference between its outcome and the desired outcome. After the trainings phase, the weights are fixed and a new set of data is presented (the test set). By measuring how well the network performs on this new data, the performance of the network can be determined.

Deep learning is the process in which multiple large ANNs (large = lots of hidden layers) are combined in layers. Each of these networks specialise in certain features. For example, in the first most basic layer, the networks might specialise in recognising vertical lines. In the second layer the ANNs uses the output of the first layer to recognise, for example noses and eye, while the third layer again combines the output from the second layer to recognise faces. As you can see in this image:


Successive model layers learn deeper intermediate representations
Layer 3
High-level linguistic representations
Parts combine to form objects
Layer 2
Layer 1
Prior: underlying factors & concepts compactly expressed w/ multiple levels of abstraction

Thanks to this method of combining several layers, Deep Learning networks are able to deliver excellent results.

7 References

Based on https://github.com/Otacon/wifi_china_drone_controller