aiGuard

Monitors security cameras images, uses AI to tag them and sends notifications.

Dependencies

```
# imageai dependencies
pip3 install tensorflow numpy scipy opencv-python pillow matplotlib h5py keras
# if you have an nvidia gpu, libcoda and nvidia kernel module you can install
tensorflow-gpu instead of tensorflow
pip3 install imageai --upgrade
# custom dependencies
pip3 install watchdog piexif python-pushover
```

The simple setup/motivation

I have several ip cameras which generates a lot of events due to:

- shadows, wind, leaves
- animals (cows, horses, dogs)

The snapshots generated on these events are stored in a ftp server. I want to install this program on the same machine to monitor several folders for IO events (new files), process them via imageai, find features and send notifications to my phone in case a person is detected.

Classification ideea

Here is the current way I see images can be classified based on features (objects) detected:

- no feature is detected. Move the file to a "nothing" dir. In the future run a slower detection (retinaNet).
- any of the "hot" features are detected. For example a person is detected. Move the file to a "hot" dir and send a notification.
- all of the "dull" features are detected. For example only cows, horses and benches are detected. Move the file to a "dull" dir.
- other features are detected. For example an elephant is detected. Move the file to a "check" dir for later inspection.

Notifications

I use pushover for notifications to my phone since it allows 7500 notifications/month for free. Some other mechanism can be implemented (ie mail).

Testing

I have a few months worth of photos to test and I simulate the copy with:

```
#!/bin/bash

for f in *.jpg; do mv "$f" $FTP_PATH; sleep 1; done
```

As a funny note: I had 90k files and mv gave: `Argument list too long`, see https://stackoverflow.com/questions/11289551/argument-list-too-long-error-for-rm-cp-mv-commands

Running on a kvm virtual machine

As I developed this on my local machine I had no speed issues. However after installing on the final machine (a kvm virtual machine) I had a number of issues. First, when running the .py I had a `Illegal instruction (core dumped)`. This is similar to the problem described here. The first approach was to downgrade to tensorflow 1.5:

```
pip3 uninstall tensorflow
pip3 install tensorflow==1.5
```

this worked however the speed was terrible. Checking for CPU flags only gave:

```
grep flags -m1 /proc/cpuinfo | cut -d ":" -f 2 | tr '[:upper:]' '[:lower:]' | {
read FLAGS; OPT="-march=native"; for flag in $FLAGS; do case "$flag" in "sse4_1" |
"sse4_2" | "ssse3" | "fma" | "cx16" | "popcnt" | "avx" | "avx2") OPT+=" -m$flag";;
esac; done; MODOPT=${OPT//_/\.}; echo "$MODOPT"; }
-march=native -mcx16 -mpopcnt
```

The solution was to stop the virtual machine and edit its configuration from:

```
<vcpu>1</vcpu>
```

to:

```
<cpu mode='host-passthrough' check='none'>
    <cache mode='passthrough'/>
</cpu>
<vcpu>1</vcpu>
```

after restarting the guest I had the following flags: `-march=native -mssse3 -mfma -mcx16 -msse4.1 -msse4.2 -mpopcnt -mavx -mavx2`, I was able to install the latest tensorflow version (1.14) and the speed greatly improved. (`virsh nodeinfo` shows the available number of cores you can use)