1. Install ubuntu 16.04 (wsl)
2. Update` sudo apt-get update
3. Install gcc` sudo apt-get install gcc
4. Remove all existing pythons` sudo apt remove python3
5. Install python3.5.2:
   - Dependencies:
     - sudo apt-get install build-essential tk-dev
     - sudo apt-get install libncurses5-dev libncursesw5-dev libreadline6-dev
     - sudo apt-get install libdb5.3-dev libgdbm-dev libsqlite3-dev libssl-dev
     - sudo apt-get install libbz2-dev libexpat1-dev liblzma-dev zlib1g-dev
   - Downloading python
     - wget https://www.python.org/ftp/python/3.5.2/Python-3.5.2.tgz
     - tar zxvf Python-3.5.2.tgz
     - cd Python-3.5.2
     - ./configure --prefix=/usr/local/opt/python-3.5.2 --enable-shared
     - make -j16
     - sudo make install
   - Make the compiled binaries globally available.
     - sudo ln -s /usr/local/opt/python-3.5.2/bin/pydoc3.5 /usr/bin/pydoc3.5
     - sudo ln -s /usr/local/opt/python-3.5.2/bin/python3.5 /usr/bin/python3.5
     - sudo ln -s /usr/local/opt/python-3.5.2/bin/python3.5m /usr/bin/python3.5m
     - sudo ln -s /usr/local/opt/python-3.5.2/bin/pyvenv-3.5 /usr/bin/pyvenv-3.5
     - sudo ln -s /usr/local/opt/python-3.5.2/bin/pip3.5 /usr/bin/pip3.5
     - sudo echo /usr/local/opt/python-3.5.2/lib > /tmp/python3.5.2.conf
     - sudo mv /tmp/python3.5.2.conf /etc/ld.so.conf.d/
     - sudo ldconfig
6. Getting the repositories
     - cd $(HOME)/
     - sudo rm -rf Python3.5.2/
     - sudo rm -rf Python3.5.2.tgz
     - git clone https://github.com/riblidezso/frcnn_cad.git
     - cd frcnn_cad/
     - git clone https://github.com/rbgirshick/py-faster-rcnn ###checkout faster-rcnn
     - cd py-faster-rcnn

        - rmdir caffe-fast-rcnn/
        - git clone https://github.com/rbgirshick/caffe-fast-rcnn
        - cd caffe-fast-rcnn/
        - git checkout faster-rcnn

7. Caffe Part
    - Get dependencies
        - pip3.5 install opencv-python==3.4.0.12 --user
        - sudo apt-get install libatlas-base-dev
        - sudo apt-get install libboost-all-dev
        - sudo pip3.5 install protobuf==3.3.0
        - sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev libopencv-dev libhdf5-serial-dev protobuf-compiler
        - sudo apt-get install the python3.5-dev
        - sudo apt-get install libgflags-dev libgoogle-glog-dev liblmdb-dev
        - sudo apt-get install python3.5-numpy
        - In
        - Change Makefile line 175 or 202 with

```
PYTHON_LIBRARIES ?= boost_python python2.7
```

TO

```
PYTHON_LIBRARIES := boost_python-py35 python3.5m
```

        - Change Makefile.config in the following way (If the config file does not exist create one)## Refer to http://caffe.berkeleyvision.org/installation.html

```
# Contributions simplifying and improving our build system are welcome!

# cuDNN acceleration switch (uncomment to build with cuDNN).

# USE_CUDNN := 1

# CPU-only switch (uncomment to build without GPU support).

CPU_ONLY := 1

# uncomment to disable IO dependencies and corresponding data layers

# USE_OPENCV := 0
# USE_LEVELDB := 0
# USE_LMDB := 0
# uncomment to allow MDB_NOLOCK when reading LMDB files (only if necessary)
#   You should not set this flag if you will be reading LMDBs with any
#   possibility of simultaneous read and write
# ALLOW_LMDB_NOLOCK := 1

# Uncomment if you're using OpenCV 3 or 4
OPENCV_VERSION := 3.4.0
USE_PKG_CONFIG := 1

# To customize your choice of compiler, uncomment and set the following.
# N.B. the default for Linux is g++ and the default for OSX is clang++
# CUSTOM_CXX := g++

# CUDA directory contains bin/ and lib/ directories that we need.
# CUDA_DIR := /usr/local/cuda
# On Ubuntu 14.04, if cuda tools are installed via
# "sudo apt-get install nvidia-cuda-toolkit" then use this instead:
# CUDA_DIR := /usr

# CUDA architecture setting: going with all of them.
# For CUDA < 6.0, comment the lines after *_35 for compatibility.
# CUDA_ARCH := -gencode arch=compute_30,code=sm_30 \
#              -gencode arch=compute_35,code=sm_35 \
#              -gencode arch=compute_50,code=sm_50 \
#              -gencode arch=compute_52,code=sm_52 \
#              -gencode arch=compute_60,code=sm_60 \
#              -gencode arch=compute_61,code=sm_61 \
#              -gencode arch=compute_70,code=sm_70 \
#              -gencode arch=compute_80,code=sm_80

# BLAS choice:
# atlas for ATLAS (default)
# mkl for MKL
# open for OpenBlas
 BLAS := atlas
#BLAS := open
# Custom (MKL/ATLAS/OpenBLAS) include and lib directories.
# Leave commented to accept the defaults for your choice of BLAS
# (which should work)!
# BLAS_INCLUDE := /path/to/your/blas
# BLAS_LIB := /path/to/your/blas


# Homebrew puts openblas in a directory that is not on the standard search path
# BLAS_INCLUDE := $(shell brew --prefix openblas)/include
# BLAS_LIB := $(shell brew --prefix openblas)/lib

# This is required only if you will compile the matlab interface.
# MATLAB directory should contain the mex binary in /bin.
# MATLAB_DIR := /usr/local
# MATLAB_DIR := /Applications/MATLAB_R2012b.app

# NOTE: this is required only if you will compile the python interface.
# We need to be able to find Python.h and numpy/arrayobject.h.
# PYTHON_INCLUDE := /usr/include/python2.7 \
#       /usr/lib/python2.7/dist-packages/numpy/core/include
# Anaconda Python distribution is quite popular. Include path:
# Verify anaconda location, sometimes it's in root.
# ANACONDA_HOME := $(HOME)/anaconda2
# PYTHON_INCLUDE := $(ANACONDA_HOME)/include \
#       $(ANACONDA_HOME)/include/python2.7 \
#       $(ANACONDA_HOME)/lib/python2.7/site-packages/numpy/core/include \

# Uncomment to use Python 3 (default is Python 2)
PYTHON_LIBRARIES := boost_python-py35 python3.5m
PYTHON_INCLUDE := /usr/include/python3.5m \

/home/yourUsername/.local/lib/python3.5/dist-packages/numpy/core/include
```

```
# We need to be able to find libpythonX.X.so or .dylib.
PYTHON_LIB := /usr/lib
# PYTHON_LIB := $(ANACONDA_HOME)/lib

# Homebrew installs numpy in a non standard path (keg only)
# PYTHON_INCLUDE += $(dir $(shell python -c 'import numpy.core; print(numpy.core.__file__)'))/include
# PYTHON_LIB += $(shell brew --prefix numpy)/lib

# Uncomment to support layers written in Python (will link against Python libs)
WITH_PYTHON_LAYER := 1

# Whatever else you find you need goes here.
INCLUDE_DIRS := $(PYTHON_INCLUDE) /usr/local/include /usr/include/hdf5/serial /usr/include/opencv2
LIBRARY_DIRS := $(PYTHON_LIB) /usr/local/lib /usr/lib /usr/lib/x86_64-linux-gnu/hdf5/serial /usr/lib/x86_64-linux-gnu




# If Homebrew is installed at a non standard location (for example your home directory) and you use it for general dependencies
# INCLUDE_DIRS += $(shell brew --prefix)/include

# LIBRARY_DIRS += $(shell brew --prefix)/lib



# N.B. both build and distribute dirs are cleared on `make clean`

BUILD_DIR := build

DISTRIBUTE_DIR := distribute



# Uncomment for debugging. Does not work on OSX due to https://github.com/BVLC/caffe/issues/171

# DEBUG := 1



# The ID of the GPU that 'make runtest' will use to run unit tests.

TEST_GPUID := 0



# enable pretty build (comment to see full commands)

Q ?= @
```

- make all -j16
- make test -j16
- make runtest -j16
- make pycaffe -j16
- In .bashrc add

```
export PYTHONPATH="/home/yourUsername/frcnn_cad/py-faster-rcnn/caffe-fast-rcnn/python":$PYTHONPATH
export PYTHONPATH="/home/yourUsername/frcnn_cad/py-faster-rcnn/lib":$PYTHONPATH
export PYTHONPATH="/home/yourUsername/frcnn_cad/py-faster-rcnn/lib/rpn":$PYTHONPATH
```

- pip3.5 install grpcio --user
- Change in proposal_layer.py line 26 self.param_str to self.param_str_

- pip3.5 install Cython
- In py-faster-rcnn/lib/Makefile change `python setup.py build_ext --inplace` to `python3.5 setup.py build_ext --inplace`
- Change py-faster-rcnn/lib/setup.py to this (All gpu parts are commentesd)

```
# --------------------------------------------------------
# Fast R-CNN
# Copyright (c) 2015 Microsoft
# Licensed under The MIT License [see LICENSE for details]
# Written by Ross Girshick
# --------------------------------------------------------

import os
from os.path import join as pjoin
from setuptools import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext
import subprocess
import numpy as np

def find_in_path(name, path):
    "Find a file in a search path"
    # Adapted from
    # http://code.activestate.com/recipes/52224-find-a-file-given-a-search-path/
    for dir in path.split(os.pathsep):
        binpath = pjoin(dir, name)
        if os.path.exists(binpath):
            return os.path.abspath(binpath)
    return None


# def locate_cuda():
#     """Locate the CUDA environment on the system

#     Returns a dict with keys 'home', 'nvcc', 'include', and 'lib64'
#     and values giving the absolute path to each directory.

#     Starts by looking for the CUDAHOME env variable. If not found, everything
#     is based on finding 'nvcc' in the PATH.
#     """

#     # first check if the CUDAHOME env variable is in use
#     if 'CUDAHOME' in os.environ:
#         home = os.environ['CUDAHOME']
#         nvcc = pjoin(home, 'bin', 'nvcc')
#     else:
#         # otherwise, search the PATH for NVCC
#         default_path = pjoin(os.sep, 'usr', 'local', 'cuda', 'bin')
#         nvcc = find_in_path('nvcc', os.environ['PATH'] + os.pathsep + default_path)
#         if nvcc is None:
#             raise EnvironmentError('The nvcc binary could not be '
#                 'located in your $PATH. Either add it to your path, or set $CUDAHOME')
#         home = os.path.dirname(os.path.dirname(nvcc))

#     cudaconfig = {'home':home, 'nvcc':nvcc,
#                   'include': pjoin(home, 'include'),
#                   'lib64': pjoin(home, 'lib64')}
#     for k, v in cudaconfig.iteritems():
#         if not os.path.exists(v):
#             raise EnvironmentError('The CUDA %s path could not be located in %s' % (k, v))

#     return cudaconfig
# CUDA = locate_cuda()


# Obtain the numpy include directory.  This logic works across numpy versions.
try:
    numpy_include = np.get_include()
except AttributeError:
    numpy_include = np.get_numpy_include()

def customize_compiler_for_nvcc(self):
    """inject deep into distutils to customize how the dispatch
    to gcc/nvcc works.

    If you subclass UnixCCompiler, it's not trivial to get your subclass
    injected in, and still have the right customizations (i.e.
    distutils.sysconfig.customize_compiler) run on it. So instead of going
    the OO route, I have this. Note, it's kindof like a wierd functional
    subclassing going on."""

    # tell the compiler it can processes .cu
    # self.src_extensions.append('.cu')

    # save references to the default compiler_so and _comple methods
    default_compiler_so = self.compiler_so
    super = self._compile

    # now redefine the _compile method. This gets executed for each
    # object but distutils doesn't have the ability to change compilers
    # based on source extension: we add it.
    def _compile(obj, src, ext, cc_args, extra_postargs, pp_opts):
        if os.path.splitext(src)[1] == '.cu':
            # use the cuda for .cu files
            # self.set_executable('compiler_so', CUDA['nvcc'])
            # use only a subset of the extra_postargs, which are 1-1 translated
            # from the extra_compile_args in the Extension class
            # postargs = extra_postargs['nvcc']
            print("CU")
        else:
            postargs = extra_postargs['gcc']

        super(obj, src, ext, cc_args, postargs, pp_opts)
        # reset the default compiler_so, which we might have changed for cuda
        self.compiler_so = default_compiler_so

    # inject our redefined _compile method into the class
    self._compile = _compile


# run the customize_compiler
class custom_build_ext(build_ext):
    def build_extensions(self):
        customize_compiler_for_nvcc(self.compiler)
        build_ext.build_extensions(self)
```

```
ext_modules = [
    Extension(
        "utils.cython_bbox",
        ["utils/bbox.pyx"],
        extra_compile_args={'gcc': ["-Wno-cpp", "-Wno-unused-function"]},
        include_dirs = [numpy_include]
    ),
    Extension(
        "nms.cpu_nms",
        ["nms/cpu_nms.pyx"],
        extra_compile_args={'gcc': ["-Wno-cpp", "-Wno-unused-function"]},
        include_dirs = [numpy_include]
    ),
    # Extension('nms.gpu_nms',
    #     ['nms/nms_kernel.cu', 'nms/gpu_nms.pyx'],
    #     library_dirs=[CUDA['lib64']],
    #     libraries=['cudart'],
    #     language='c++',
    #     runtime_library_dirs=[CUDA['lib64']],
    #     # this syntax is specific to this build system
    #     # we're only going to use certain compiler args with nvcc and not with
    #     # gcc the implementation of this trick is in customize_compiler() below
    #     extra_compile_args={'gcc': ["-Wno-unused-function"],
    #                         'nvcc': ['-arch=sm_35',
    #                                  '--ptxas-options=-v',
    #                                  '-c',
    #                                  '--compiler-options',
    #                                  "'-fPIC'"]},
    #     include_dirs = [numpy_include, CUDA['include']]
    # ),
    Extension(
        'pycocotools._mask',
        sources=['pycocotools/maskApi.c', 'pycocotools/_mask.pyx'],
        include_dirs = [numpy_include, 'pycocotools'],
        extra_compile_args={
            'gcc': ['-Wno-cpp', '-Wno-unused-function', '-std=c99']},
    ),
]

setup(
    name='fast_rcnn',
    ext_modules=ext_modules,
    # inject our custom trigger
    cmdclass={'build_ext': custom_build_ext},
)
```

- cd /home/yourUsername/frcnn_cad/py-faster-rcnn/lib
- In nms_wrapper.py change all the prints according to the syntax of python 3.5, also find xrange in a for loop and refactor it to range
- In generate_anchors.py change print functions too (add parentheses)
- make
- sudo apt-get install python3-tk
- There are more small changes needed in files. Fix other syntax errors in files such as prints, xranges, iteritem() to item() …