Korea Advanced Institute of Science and Technology
CS492 Spring 2020
Project 1

Seunghyo Kang
shkang@casys.kaist.ac.kr

Due: Apr. 21

Rules

- Do not copy from others. You will get huge panelty according to the University policy. Sharing of code between students is viewed as cheating.
- This project is a team project.
- Every code was written and will be tested in the provided docker environment. You need to implement your code in the given environment.
- Read carefully not only this document but the comment in the skeleton code.
- This is the version 1.0 document which means not perfect. Any specifications or errors can be changed during the project period with announcement. Stay tuned to Piazza.

Introduction

This project is made to study the basic understanding of ML inference using high-level framework called tensorflow with graph construction. Your job is to implement an inference code dectecting objects with YOLO v2 tiny algorithm using the framework. A sample video is given, you can check your implementation using the video as input. Or you can take your video to test your code.

Project Sturcture

This project contains the following files:

- `Dockerfile`: File for docker environment setting.
- `__init__.py`: Main source file. Following command will produce a labeled video:

  ```
  $ python3 __init__.py [path_to_input] [path_to_output]
  ```

- `yolov2tiny.py`: Source file of `YOLO_V2_TINY` class.
- `y2t_weights.pickle`: File that contains weight parameters.
- `sample.mp4`: Sample input video for the inference job.

Run Docker

Docker is a virtualization technique that is mainly used to distribute the same environment to users. For instance, a program may not be executable under the different version from development environment. Or an experiment result may vary between the versions of Linux or libraries. However, installing every libraries with version checking has high chance of causing mistakes. Using docker, we can sync the environment easily. Install docker-ce refer to this link. After that, install nvidia-docker from this link to enable your docker to use gpus.
There are plenty of useful and pre-built docker images. We can use one of them not to make docker images from the beginning. `Dockerfile` is a configuration file of your image that contains several commands executed at the initial set up. Let's look at some lines from our `Dockerfile`. You can easily see the role of `Dockerfile`.

```
# Start from the existing docker image
FROM nvidia/cuda:10.0-cudnn7-devel-ubuntu16.04
...

# Install packages
RUN apt-get update
RUN apt-get install -y build-essential wget python3 python3-pip ...
...

# Install python packages
RUN python3 -m pip install opencv-python tensorflow-gpu==1.15.2 ...
...
```

After you write `Dockerfile`, you can build your image based on the file. Following command will find `Dockefile` in the current directory, run the command for the new environment, and tag it `cs492:tensorflow-gpu`.

```
$ docker build -t cs492:tensorflow-gpu .
```

Note that there is a period mark at the end which indicates the current directory. Run the environment built in previous step and name it `cs492-gpu` with following command.

```
$ docker run --gpus all -it -v $HOME/projects:/home/cs492/projects
    --name cs492-gpu cs492:tensorflow-gpu bash
```

Press `Ctrl+p, Ctrl+q` to detach from your docker and go back to host machine. If you want to re-attach to the docker, use this command.

```
$ docker attach cs492-gpu
```

Refer to this document for more docker commands.

Implementation

Read and Write Videos (5pt)

The function `open_video_with_opencv` in `__init__.py` takes two pathes to videos from command-line arguments and opens them. Your first job is to write a code to create objects for each videos with an image library, `opencv`. You will be able to read and write image frames through these objects. Open the given input and output video file using `cv2.VideoCapture` and `cv2.VideoWriter` respectively and return them. You may need the size of the input video or something else; return whatever you want for future computation.

Tensorflow Graph Construction (20pt)

Look carefully at the `YOLO_V2_TINY` class in `yolov2tiny.py`. It has a constuctor, a function that builds a tensor graph, and a function that initiates the inference. Your job is to implement `build_graph` with tensorflow library which builds a **tensor graph**. The graph will be consist of nodes, called **tensor**, which represent internal computations in the inference job. You need to add the tensors according to the YOLO v2 tiny algorithm model. Specifically, return the starting tensor of graph and the list of all tensors to compute intermediate values. By the way, they will be evaluated not on this construction but on the running a session at the `inference`. Check how the

return values are used.
Tensorflow API provides basic operations for machine learning computation such as convolution or adding a bias. Refer to this API website. `tf.maximum, tf.nn` will come in handy. Furthermore, use 10^{-5} for ϵ in batch normalization layers.

Inference (10pt)

Now you are ready to process the video in `video_object_detection`. Before starting the inferecing, make a `YOLO_V2_TINY` object with proper argument to construct a graph for inference. Take frames from opened video and do inference job with the graph you've implemented. Your code must include following jobs for each frame.

- Do the `inference`,
- Run `postprocessing` in `yolov2tiny.py` using the inference result, accumulate them through the video writer object. Note that coordinates from `postprocessing` are calculated according to resized input; you must adjust them to fit into the original video.
- Measure the end-to-end time and time spent only for inferencing. `time` library will be helpful.

Note that the raw input must be adjusted to fit into the algorithm, including resizing the frame and changing the dimension.
Plus, save the intermediate values of the *first frame* in the `intermediate` directory. `np.save` will help you. The name of the files must be *layer_i.npy* where i is the layer number. It will be graded by comparing them to the output from refernce code.

Report (5pt)

Write a report of one or two pages long. Your report must include 1. how you implemented, 2. execution time and how many FPS processed (end-to-end, only for inference), 3. comparison the execution time from CPU and GPU and analyze it. The purpose of the report is to show your understanding. Please write the answer short and clear.

Handin

Your final outcome should be a single tar file that contains two python files and a report in pdf format. The name of the file should be `[team_number].tar`. Upload your tar file to KLMS.