**EdgeNet - Shoe Image Classifier Challenge Document**

**(Download Instruction Videos: https://www.dropbox.com/s/2yv7kjz6zlho9ra/videos.zip)**

1. **Installation (V0\_Installation.mov)**
   1. **Prepare overall environment**

There are two options to deploy the solution: Docker (recommend) and Conda.

* + - **Docker (recommend)**

**Pre-requirements:** docker, nvidia-docker

**Deploy:**

1. Build docker image:

|  |
| --- |
| cd submission  docker build -t edgenet . |

1. Run docker with GPU:

|  |
| --- |
| docker run --gpus all --ipc=host --shm-size 8G \  -p 8081:8081 -p 8099:8099 -p 8989:8989 -it \  --name edgenet\_ins edgenet |

* + - **Conda (not recommend)**

**Pre-requirements:** CUDA(>=10.1), Cudnn(>=7)

**Deploy:**

1. Install miniconda and python requirements

|  |
| --- |
| cd submission  source installation\_conda.sh |

**Note**: This may run into unexpected bugs due to different systems.

* 1. **Prepare for deployment**

1. Download model checkpoints

|  |
| --- |
| ./download\_deploy.sh |

1. Test classify\_images function

|  |
| --- |
| python -m http.server 8099 > http.log 2>&1 &  python test\_function.py |

**Note: If only want to use classify\_images() function, stop here.**

* 1. **Prepare for training**

If want to training and testing using jupyter notebook, please run this part.

1. Download data and ImageNet pretrain weights

|  |
| --- |
| ./download\_train.sh |

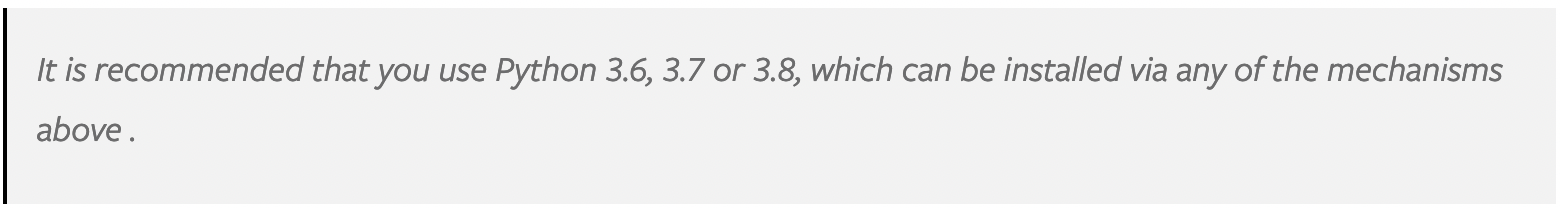
1. Setup jupyter notebook

|  |
| --- |
| jupyter notebook --generate-config  jupyter notebook password  nohup jupyter notebook --ip=0.0.0.0 --port=8081 \  --no-browser --allow-root --notebook-dir=./ & |

Note: Then you can open http://<host\_address>:8081/ to run **train.ipybn** and **test.ipynb**

* 1. **Explanation**

As declared in challenge discussion, **we use Python 3.8.5 instead of 3.9.5**, since there is no official PyTorch support for Python 3.9 at this point (18 June 2021):

****

Please see <https://pytorch.org/get-started/locally/#linux-python> for detail information.

Other Python dependency libraries used mainly include:

* Catalyst ( <https://github.com/catalyst-team/catalyst/> )
* Albumentations ( <https://github.com/albumentations-team/albumentations/> )

Both of them following MIT license, allowed for commercial use.

The training and validation data are download from Internet (Google image or Baidu image). Since only the deep-learning model is deployed in the final product, this will not be a problem.

1. **Training and Testing (V1\_Train.mov, V2\_Test.mov)**

We use jupyter notebook for training and testing. Just open the corresponding notebook and follow the instruction.

**None: The validation images “./data/val” is used for test in default, and the output of test is saved to “./output/test.csv”. Please see test.ipynb for details.**

1. **Function classify\_images() (V3\_Function\_and\_API.mov)**

Function classify\_images() is defined in “**function.py**”.

* 1. **Input**

Input is a list of path or URL.

For example: [“./demo\_images/val\_00150.jpg”, “http://localhost:8099/demo\_images/val\_00150.jpg”]

* 1. **Output**

Output is a list of result labels.

For example: [“boots”, “boots”]

* 1. **Exception**

If one of the path or URL in the list is wrong, the function will capture the Exception and return the Exception string as the results at corresponding position.

For example, if some inputs are wrong path and URL: […, “error\_path”, “http://error”, …]. Then the function will output: […, “ValueError(…)”, “URLError(…)”, …].

* 1. **Test classify\_images**

We can use “**test\_function.py**” to test this function. The usage of test\_function.py is as following:

|  |
| --- |
| python test\_function.py --paths <path\_or\_url\_0> <path\_or\_url\_1> ... |

The prediction results list will be print out directly. And the inputs and outputs are reorganized and print out as following:

|  |
| --- |
| <path\_or\_url\_0> -> output\_label\_0  <path\_or\_url\_1> -> output\_label\_1  <wrong\_path\_or\_url> -> exception\_message  … |

1. **API (V3\_Function\_and\_API.mov)**

API is defined in “start\_api.py” which is a simple wrapper of classify\_images function.

* 1. **Run API service**

Run following command to start API service:

|  |
| --- |
| python -u start\_api.py <GPU\_ID> <APP\_PORT> |

<GPU\_ID> is the id of GPU (start from 0) if you have multiple GPUs. Use 0 if have single GPU.

<APP\_PORT> is the Port to run the API.

For example:

|  |
| --- |
| python -u start\_api.py 0 8989 > api\_log.txt 2>&1 & |

* 1. **Request API**

Send a POST Request to use the API:

|  |
| --- |
| request\_json\_data = {  images: [<path1>, <url2>, ..., <err\_url\_5>, ...]  } |

The response should be:

|  |
| --- |
| response\_info = {  result: [<label1>, <label2>, ..., <err\_message\_5>, ...]  } |

* 1. **Test API**

We provide a script “**test\_api.py**” to test API.

|  |
| --- |
| python test\_api.py <api\_host> <api\_port> \  --path <path\_or\_url\_0> <path\_or\_url\_1> ... |

For example:

|  |
| --- |
| python test\_api.py localhost 8989 --paths \  ./demo\_images/val\_00150.jpg \  http://localhost:8099/demo\_images/val\_00150.jpg \  ./error\_path \  http://error |

The output of “test\_api.py” to stdout is as same as “test\_function.py”.