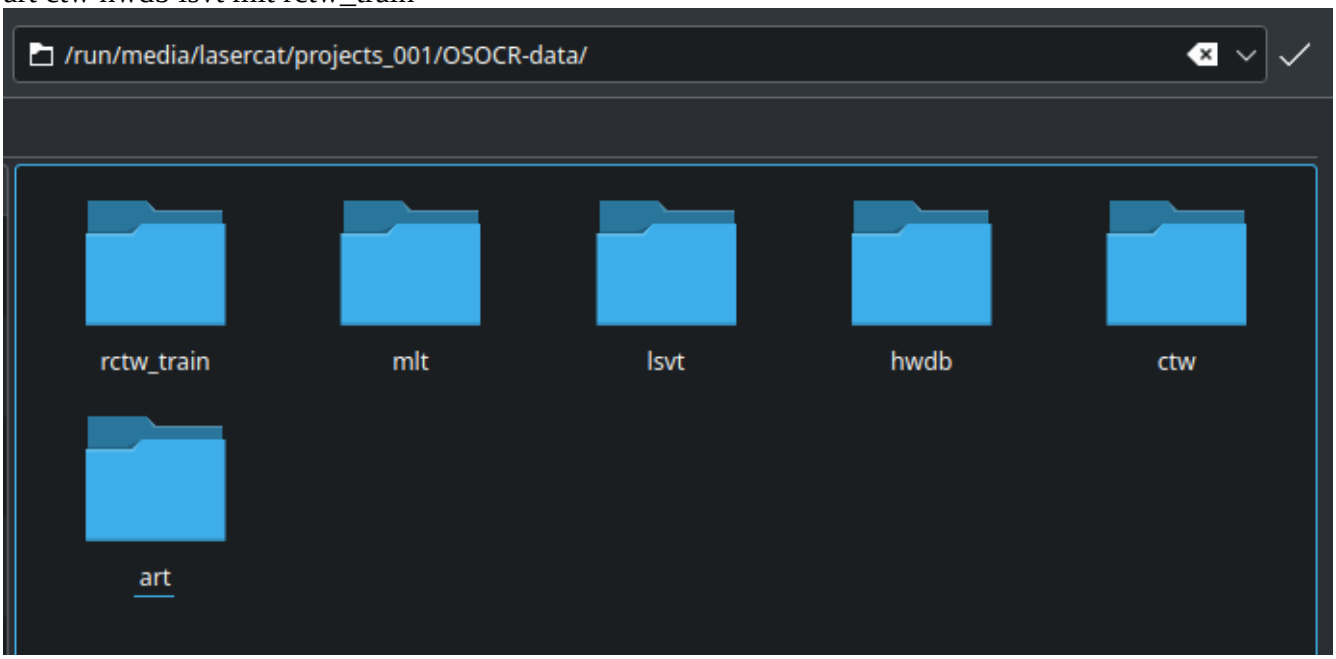
Data Builder for OSOCR-Family

1. Building datasets used in the paper.

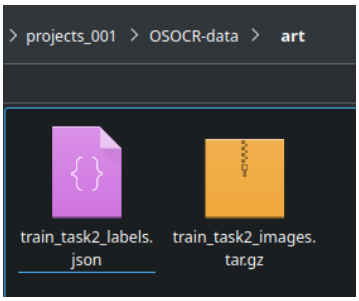
1. Make a dataset source dir ${SRC}, which has the following subfolders:

art ctw hwdb lsvt mlt rctw_train



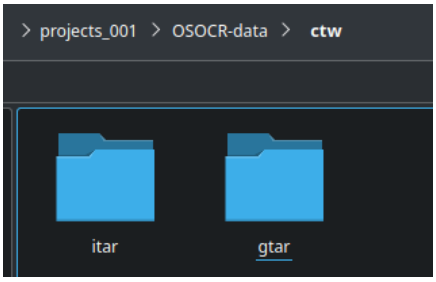
2. Download datasets listed in dataset_sources.txt into corresponding dirs. Since I am on a paid-by-data network, let me skip the re-downloading process… If anything goes south, please open an issue.

2.1 Art:
After downloading, make sure the following two files lies in the art folder:
train_task2_images.tar.gz and train_task2_labels.json

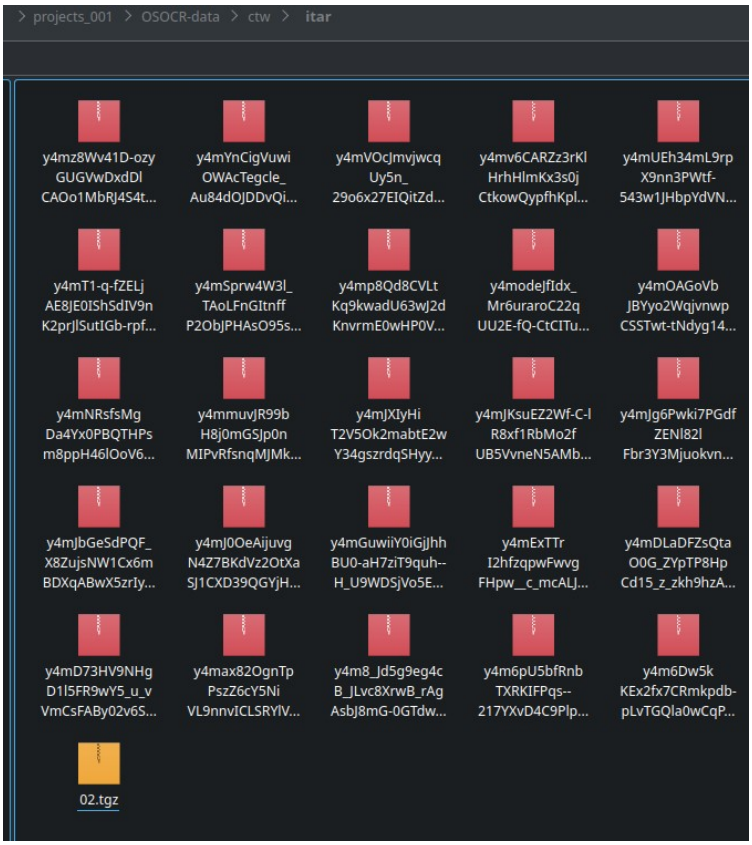


2.2 CTW:

2.2.1 Make two folders: itar for image, and gtar for gts.

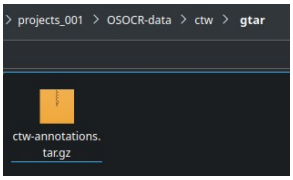


2.2.2 Download all parts into itar

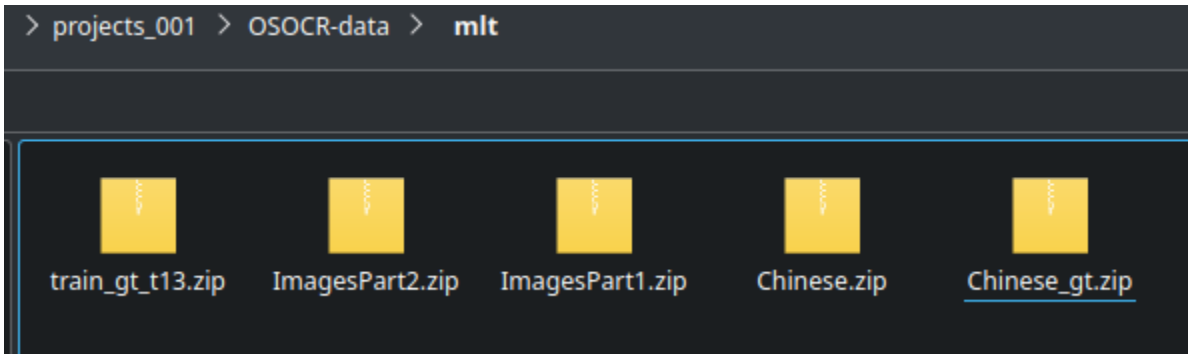


2.2.3. Make a tarlist for the CTW dataset:
ls |grep -v tarlist>tarlist

2.2.4. Download annotation to the gtar folder:

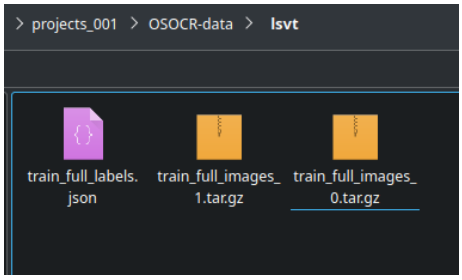


2.3. Download MLT-19 real and Chinese synthetic data to the mlt folder.

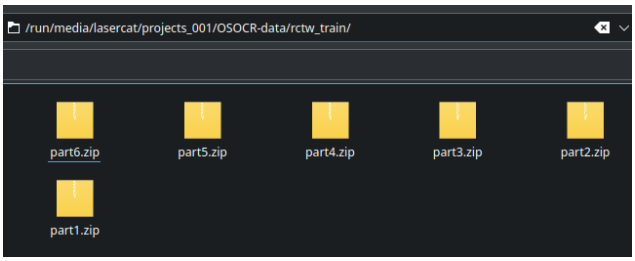


The synthetic data is not actually used…. But… Please make sure it's there...

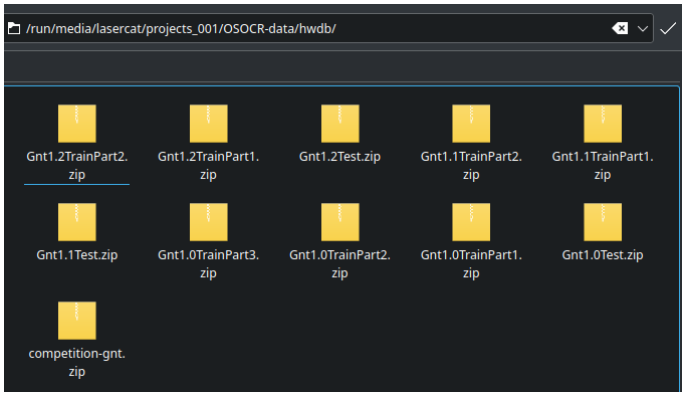
2.4. Download the LSVT data to the lsvt folder.
Only the fully supervised part needed.



2.5 Download the 6 parts of rctw training set into the rctw_train folder.



2.6 Download OLHWDB 1.0-1.2 and the competition set to the hwdb folder:



2.7 Almost there! Let’s check the recipe:

ls -R >all.my;
Compare all.my to the all.txt in the repo, make sure you do not miss a file or two

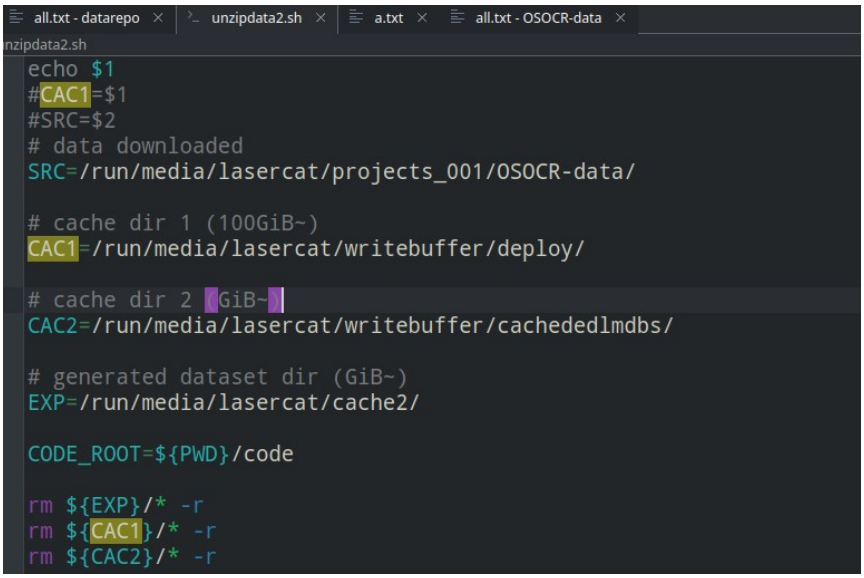
3. Set up enviornment

The scripts contain some rmdir, rm, mv commands, so please isolate it from important data.
!!!!YOU ARE WARNED!!!!!

3.1 Follow this link to setup
https://github.com/lancercat/make_env/

3.2 Buy a used 240Gib SSD from ebay or elsevier. Two 120 Gib drives shall do as well. The gist is not to write a lot temporary files to your expensive main SSD.

3.3 Setup paths:
Open up unzipdata2.sh and set SRC, CAC1, CAC2, and EXP.



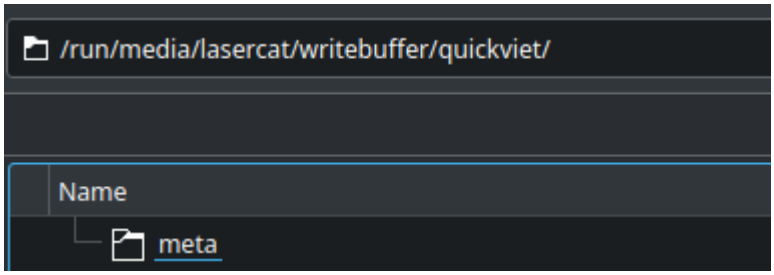
SRC is your dataset source folder.
CAC1 is one cache folder, and CAC2 is another, redirect them to some cheap disks. EXP is where built datasets are going to be stored. These folders should be **EMPTY**
The scripts contain some rmdir, rm, mv commands, so please CORRECTLY set the paths.
One step wrong, your data GONE.
!!!!YOU ARE WARNED!!!!!
4. Grab some snacks and push the **RED BUTTON**, wait for a few hours and your lmdbs will be there:
sh unzipdata2.sh

2. Building custom datasets.

Okay, so you decide to train the model with other data in different languages, for example, Vietnamese. Here is a quick guide:

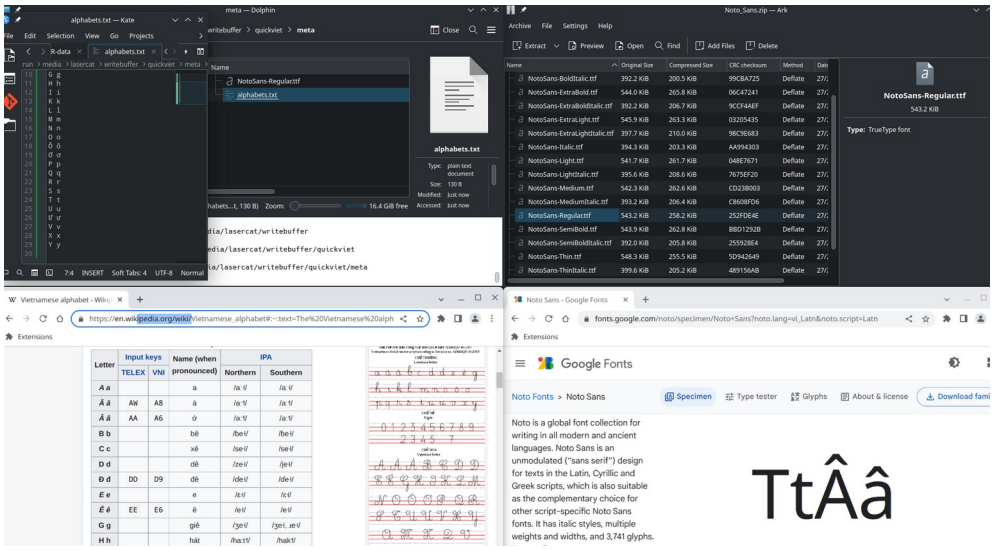
1. Collect data

1.1 Make a working directory, say quickviet, and add a meta folder inside.

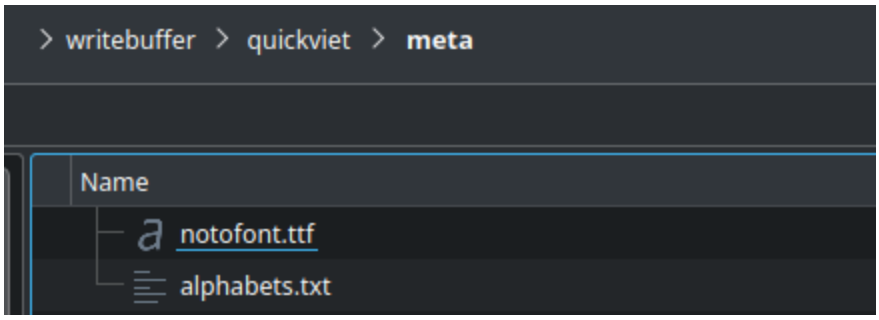


1.2 Collect alphabetas and fonts. The alphabet file has to be named “alphabet.txt”.

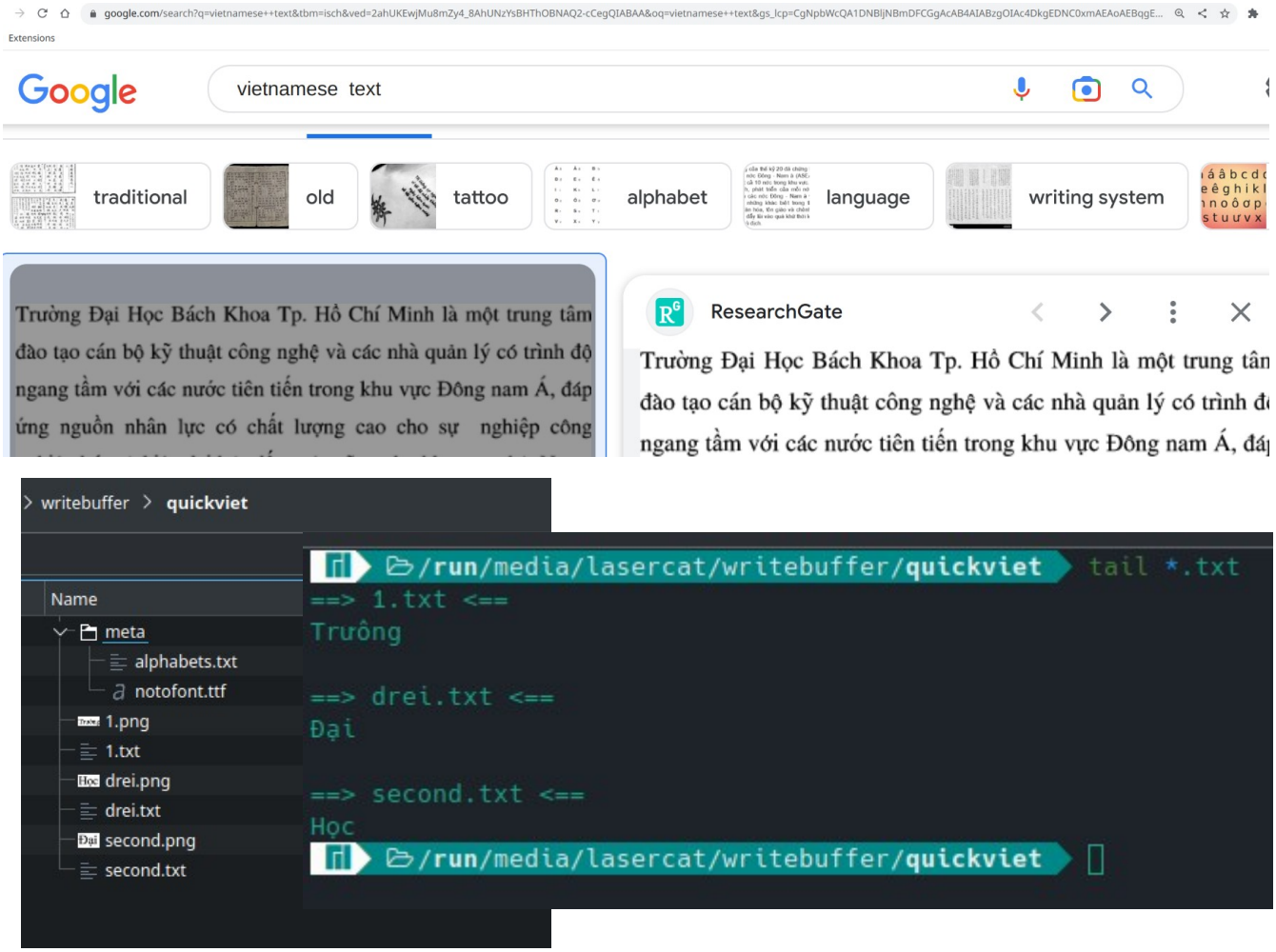
Each line corresponds to a label, where different cases are separated by spaces.



1.3 Rename the font file to notofont.ttf



1.4 Collect data and annotation, let’s take 3 images from google to make this short.



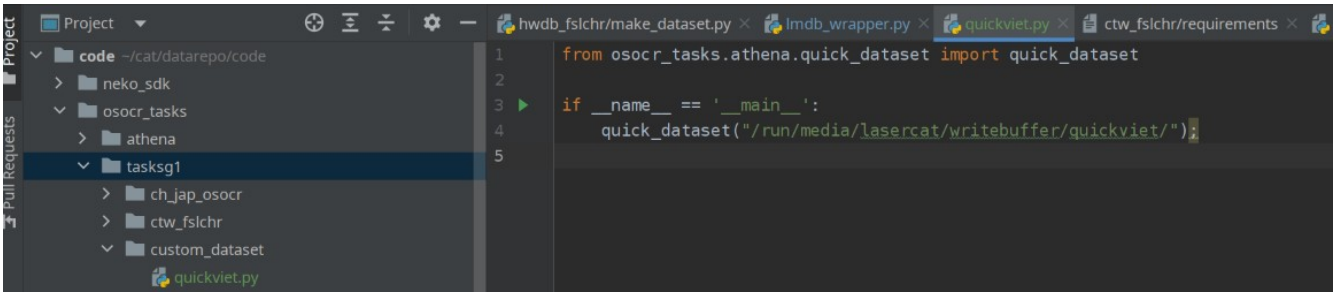
Hope I got that straight… Well I think I missed belowdoted(?) characters in meta, which will cause the model to constantly treat them as unknown. Nevermind, let’s carry on.

Note all images have to be in png format, where png has to be in lower case.

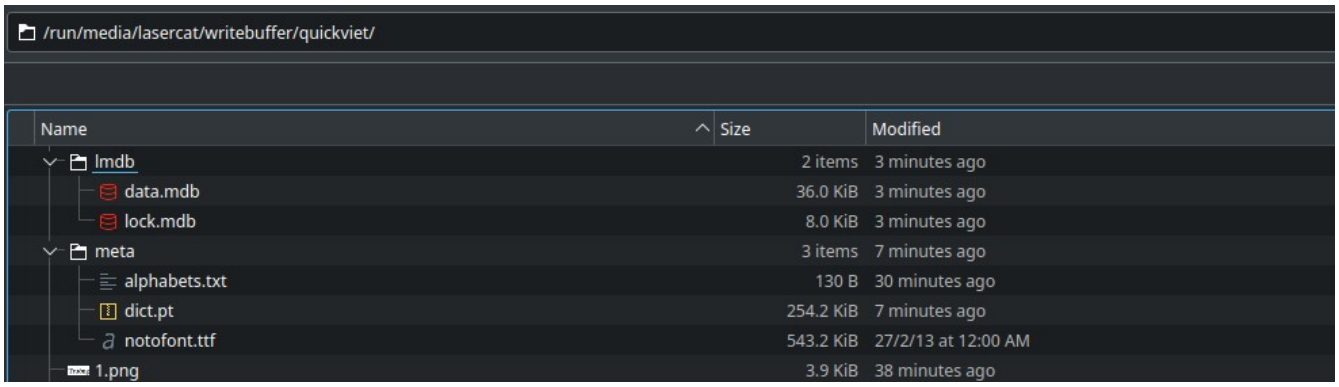
All annotation files have to be in txt format, with name identical to corresponding png files.

2. Build the dataset

2.1 Create a custom dataset script in taskg1/customdataset. We call it quickviet here. Other languages can be proceeded in the same manner.



2.2 Point the path to your working dir and launch the script, it will build the lmdb and text for you.

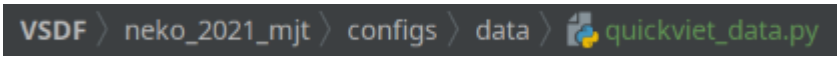


Simple, eh?

3. Setup training & testing protocol in OpenCCD

Now time to take that for a spin, we will show you how using OpenCCD as example. Well we have 3 training samples, so we are training and testing with the same set.

3.1 Define a new training dataset, a new testing dataset and a new testing dataset collection in neko_2021_mjt/configs/data:



3.1.1 Training set

```python
from neko_2020nocr.dan.dataloaders.dataset_scene import colored_lmdbDataset,colored_lmdbDatasetT
from neko_2020nocr.dan.configs.datasets.ds_paths import *
from torchvision import transforms
from torch.utils.data import DataLoader


def get_quickviet_training_cfg(root,maxT,bs=48,hw=[32,128],random_aug=True):
    rdic={
                "type": colored_lmdbDataset,
                'ds_args': {
                    'roots': ["/run/media/lasercat/writebuffer/quickviet/lmdb/"],
                    'img_height': hw[0],
                    'img_width': hw[1],
                    'transform': transforms.Compose([transforms.ToTensor()]),
                    'global_state': 'Train',
                    "maxT": maxT,
                    'qhb_aug': random_aug
                },
                "dl_args":
                {
                    'batch_size': bs,
                    'shuffle': False,
                    'num_workers': 8,
                }
            }
    return rdic;
```

Note we use absloute path to the lmdb here, you can put it in the dataroot and use relative path.

This helps you to deploy across devices.

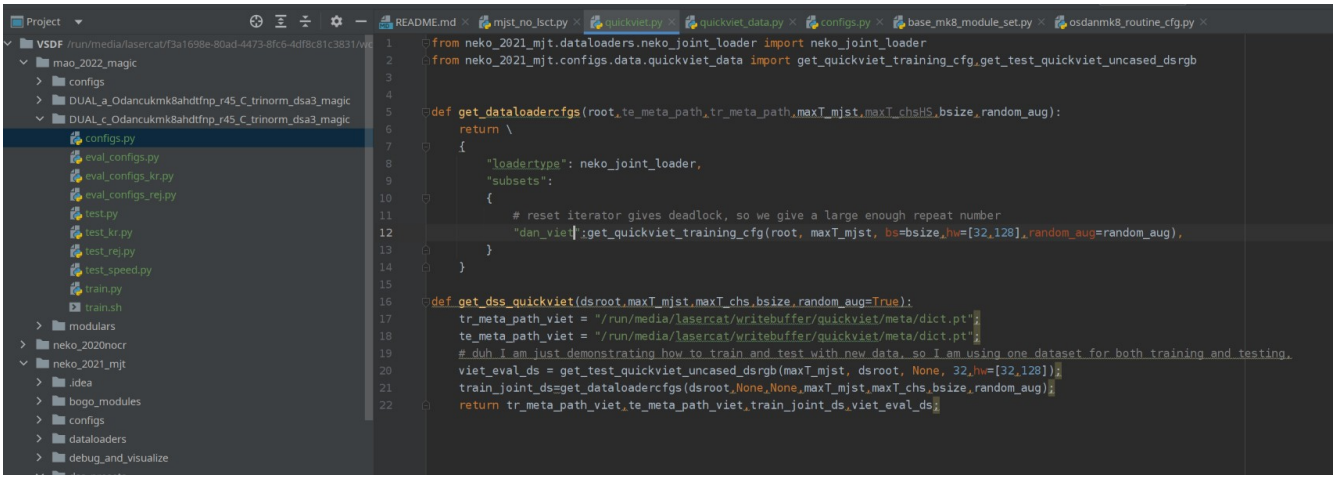
3.1.2 Testing set and the collection

```python
def get_quickviet_testC(maxT,root,dict_dir,batch_size=128,hw=[32,128]):
    return {
        'type': colored_lmdbDatasetT,
        'ds_args': {
            'roots': ["/run/media/lasercat/writebuffer/quickviet/lmdb/"],
            'img_height': hw[0],
            'img_width': hw[1],
            'transform': transforms.Compose([transforms.ToTensor()]),
            'global_state': 'Test',
            "maxT": maxT,
        },
        'dl_args': {
            'batch_size': batch_size,
            'shuffle': False,
            'num_workers': 8,
        },
    }

def get_test_all_uncased_dsrgb(maxT=25,root="/home/lasercat/ssddata/",dict_dir=None,batchsize=128,hw=[32,128]):
    return {
        "dict_dir":dict_dir,
        "case_sensitive": False,
        "te_case_sensitive": False,
            "datasets":{
            "quickviet": get_quickviet_testC(maxT, get_cute(root), dict_dir,batchsize,hw),
            }
        }
```

3.2 Make a preset off the dataset (neko_2021_mjt/dss_presets/quickviet.py):

Pair datasets with dicts:

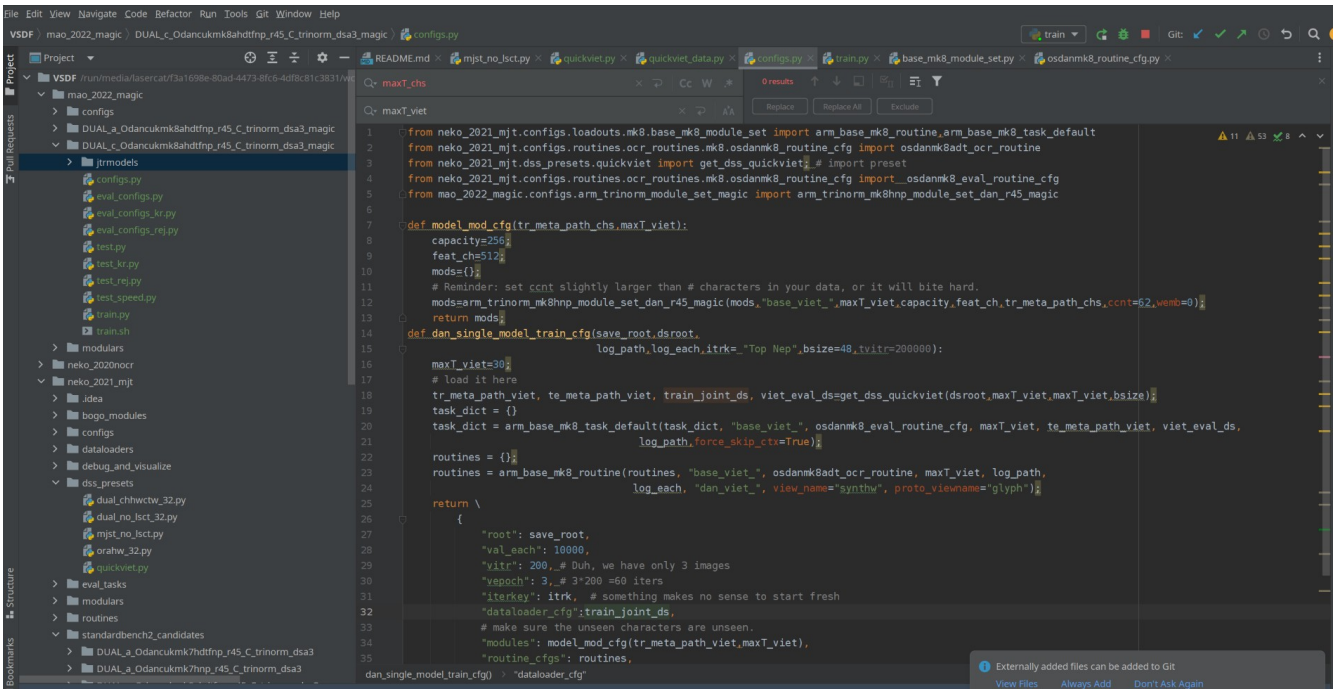


3.3 Build a model and train:

mao_2022_magic/DUAL_c_Odancukmk8ahdtfnp_r45_C_trinorm_dsa3_magic

3.3.1 Config training

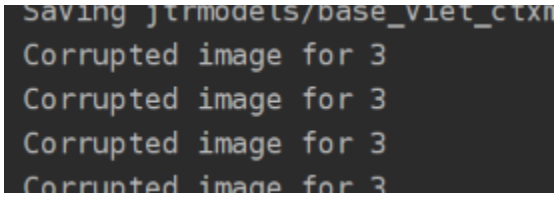
mao_2022_magic/DUAL_c_Odancukmk8ahdtfnp_r45_C_trinorm_dsa3_magic/config:



```
from neko_2021_mjt.configs.loadouts.mk8.base_mk8_module_set import arm_base_mk8_routine,arm_base_mk8_task_default
from neko_2021_mjt.configs.routines.ocr_routines.mk8.osdanmk8_routine_cfg import osdanmk8adt_ocr_routine
from neko_2021_mjt.dss_presets.quickviet import get_dss_quickviet; # import preset
from neko_2021_mjt.configs.routines.ocr_routines.mk8.osdanmk8_routine_cfg import _osdanmk8_eval_routine_cfg
from mao_2022_magic.configs.arm_trinorm_module_set_magic import arm_trinorm_mk8hnp_module_set_dan_r45_magic

def model_mod_cfg(tr_meta_path_chs,maxT_viet):
    capacity=256;
    feat_ch=512;
    mods={};
    # Reminder: set ccnt slightly larger than # characters in your data, or it will bite hard.
    mods=arm_trinorm_mk8hnp_module_set_dan_r45_magic(mods,"base_viet_",maxT_viet,capacity,feat_ch,tr_meta_path_chs,ccnt=62,womb=0);
    return mods;
def dan_single_model_train_cfg(save_root,dsroot,
                               log_path,log_each,itrk="Top Nep",bsize=48,tvitr=200000):
    maxT_viet=30;
    # load it here
    tr_meta_path_viet, te_meta_path_viet, train_joint_ds, viet_eval_ds=get_dss_quickviet(dsroot,maxT_viet,maxT_viet,bsize);
    task_dict = {}
    task_dict = arm_base_mk8_task_default(task_dict, "base_viet_", osdanmk8_eval_routine_cfg, maxT_viet, te_meta_path_viet, viet_eval_ds,
                                          log_path,force_skip_ctx=True);
    routines = {};
    routines = arm_base_mk8_routine(routines, "base_viet_", osdanmk8adt_ocr_routine, maxT_viet, log_path,
                                    log_each, "dan_viet_", view_name="synthw", proto_viewname="glyph");
    return \
        {
            "root": save_root,
            "val_each": 10000,
            "vitr": 200,  # Duh, we have only 3 images
            "vepoch": 3, # 3*200 =60 iters
            "iterkey": itrk,  # something makes no sense to start fresh
            "dataloader_cfg":train_joint_ds,
            # make sure the unseen characters are unseen.
            "modules": model_mod_cfg(tr_meta_path_viet,maxT_viet),
            "routine_cfgs": routines,
```

3.3.2 Train the model.



```
Saving jtrmodels/base_viet_ctxn
Corrupted image for 3
Corrupted image for 3
Corrupted image for 3
Corrupted image for 3
```

We have 3 images and the third is corrupted..Pfffft.

The bug is caused by zero-starting, since it only affects one image, we won’t fix it. If you have limited # of images, just repeat the data twice while building the LMDB.

Anyway, that’s about how you add a new training language.

If anything goes south, don’t hesitate mailing me.