Public LB 0.938 & Local LB 0.887 **Model summary:**

Basic info.
Network: se-resnext50
Batch-size: 320
Training set: 25k/class
Valid set: 80/class
Image size: 96*96
LR schedule: SGD (momentum=0.9, weight_decay=0.0001), start from learning rate = 0.01, train until convergence, then change into 0.001, train again until convergence
Tricks

1. Encode image by different colors (in the sequence of stroke), it increases my result from *local 0.877 & public lb 0.926* to *local 0.885 & public lb 0.93*

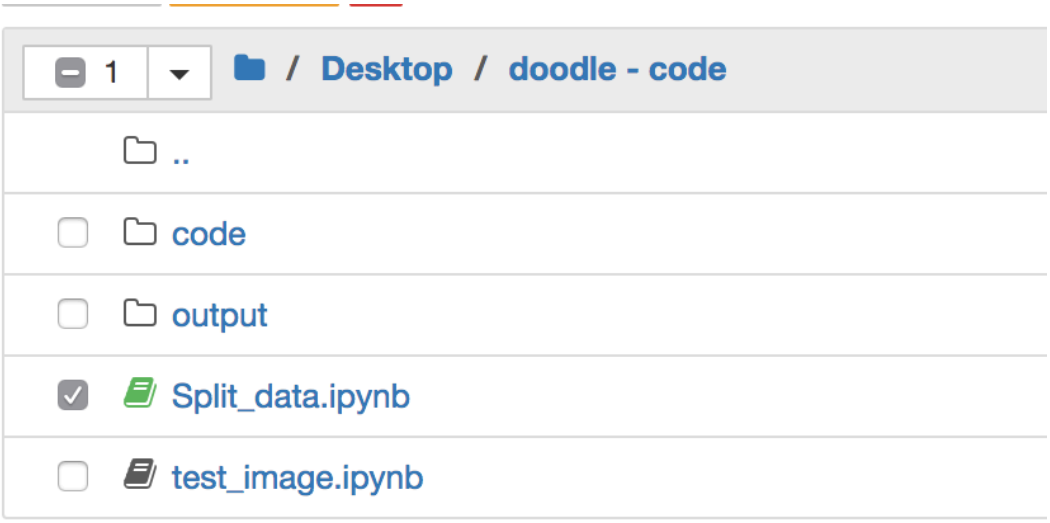
2. Add data augmentation. Specifically, hflip under prob=0.5, and randomly crop 80% of image then resize for training set; center crop 80% of image then resize for valid (and test) set, then the result increases to *local 0.887 & public lb 0.935*

3. TTA. For each test image, do hflip × five different crops (center, up & left, bottom & right and so on) to get 10 images and predict respectively. It achieves *public lb 0.938*

To use the code, please follow the following step:

1. **download** training set (simplified version)

2. **Preparing dataset**. Split dataset, keep one valid set (80 images/class) and one holdout set (80 images/class too)



- Open *Split_data.ipynb*
- Please change the path into the place you put your training set

```
    #output, if you want to use all images for training:
    X_keep.to_csv(path + 'code/split/train_0/' + file, index=False)
#     #if you want to use a part of images for training:
#     X_sample.to_csv(path + 'code/split/train_0/' + file, index=False)
```

- Output all data (*X_keep* in code) or sample data (*X_sample*) for training (all data will definitely increase our results)
- Finally, run code/process/ data_process.ipynb, nothing need to change.

3. **Training**. In code/train/train.ipynb, and set your configurations:

```
def valid_augment(drawing, label, index):
#     image = drawing_to_image_with_color_v2(drawing, 96, 96)
    seq = iaa.Sequential([
    iaa.Crop(percent=(0.05, 0.05, \
                      0.05, 0.05), keep_size=True)
    ])
    image = drawing_to_image_with_color_aug(drawing, 96, 96, seq)
    return image, label, None


def train_augment(drawing, label, index):
    up_rand = np.random.random()
    right_rand = np.random.random()
    percent_crop = 0.1
    seq = iaa.Sequential([
    iaa.Fliplr(0.5),
    iaa.Crop(percent=(up_rand*percent_crop, right_rand*percent_crop, \
                      (1-up_rand)*percent_crop, (1-right_rand)*percent_crop), keep_size=True)
    ])

    image = drawing_to_image_with_color_aug(drawing, 96, 96, seq)
    return image, label, None
```

- You may change the augmentation strategies, here “1” means do center crop for valid set; “2” means do hflip (prob=0.5) and random crop (totally 10% for left&right, and 10% for up and bottom); “3” means input image size is 96*96. Maybe 128 or 256 is better?

```
fold    = 0
out_dir = \
    '../../output'
initial_checkpoint = None
#         '../../output/backup/873_crop.pth'

pretrain_file = None #pre-trained model will be loaded automatically,

batch_size = 256+64
epoch = 6
num_iters   = epoch * 340 * 25000 // batch_size

#     schduler  = NullScheduler(lr=0.01)
schduler = DecayScheduler(base_lr=0.01, decay=0.1, step=num_iters/2)
iter_save_interval = 2000
criterion          = softmax_cross_entropy_criterion
```

- Change your batch_size and epoch in “1”, larger batch_size seems better. As for the epoch, here I use 6 and under my decay scheduler, it will train 3

epoch with lr = 0.01, and 3 epoch with lr = 0.001. Whereas, as I said, you should keep training under lr = 0.01 until your model is convergent. Here, 6 is just my rough estimation.
As shown in “2”, “340 * 25000” should be replaced by the training set size you use.

4. **Predicting**. Move to code/ local_submit-withTTA.ipynb, in step 3, the training log and checkpoints are stored in output/

TTA setting

```
]: mode  = 'test' #'train'
configures =[
     Struct(
        split         = '<NIL>', #'valid_0', #
        out_test_dir = '../split/test',
        checkpoint    = '../../output/backup/887_crop.pth',
     ),
]
#5 elements in aug list: 1st -- flip, 2nd to 5th -- top, right, bottom, left crop
augments = []
for flip_prob in [0,1]:
    for top in [0, 0.1]:
        for right in [0, 0.1]:
            augments.append([flip_prob, top, right, 0.1-top, 0.1-right])
    augments.append([flip_prob, 0.05, 0.05, 0.05, 0.05])
```

- Choose the weights you need for prediction in “1”
- Change the TTA strategies in “2” if you wish. My TTA function is ugly - -, I just input an array to define the corresponding augmentation.