Codes and details for reproducing our study: Classification and mutation prediction based on histopathology H&E images in liver cancer using deep learning

- **Prerequisites**
- **Data**
  - Whole slide images
  - Crop into “Tiles” and convert into jpg
  - Sort “Tiles” into training/test/internal_validation/external_validation sets and put them into appropriate classes
- **Training and testing/validating (choose anyone of following methods)**
  - Method 1: Training model using codes
  - Method 2: Use EASY DL without any codes, which based on the similar Algorithm with Method 1 (For freshman or non-computer specialists and readers)
- **Performance of the model**
  - Performance evaluation

Note: You can find the all Codes and dates from GitHub (https://github.com/drmaxchen-gbc/HCC-deep-learning)

Prerequisites

- Python (3.6)
- Numpy (1.14.3)
- Scipy (1.0.1)
- PyTorch (0.3.1)/CUDA 8.0 The specific binary wheel file is cu80/torch-0.3.1-cp36-cp36m-linux_x86_64.whl. I have not tested on other versions, especially 0.4+, wouldn't recommend using other versions.
- torchvision (0.2.0)
- PIL (5.1.0)
- scikit-image (0.13.1)
- OpenSlide 3.4.1(Please don't use 3.4.0 as some potential issues found on this version)/openslide-python (1.1.0)
- matplotlib (2.2.2)

Most of the dependencies can be installed through pip install with version number, e.g.

```
pip install 'numpy==1.14.3'
```

For PyTorch please consider downloading the specific wheel binary and use

```
pip install torch-0.3.1-cp36-cp36m-linux_x86_64.whl
```

Data

Whole slide images/ WSIs

Liver Cancer images from the GDC database:

We originally downloaded the "Tissue Slides" dataset from the legacy website, "https://portal.gdc.cancer.gov/legacy-archive/search/f" via the gdc-client tool:

- download the client from https://gdc.cancer.gov/access-data/gdc-data-transfer-tool
- Create and download a manifest (gdc_manifest.2019-11-29.txt) and metadata json file (clinical_data.json) from the gdc website
- Download images using the manifest and the API: gdc-client.exe download -m gdc_manifest.txt

Some .svs slides might be corrupted, in which case they could also be downloaded from the new website ("https://portal.gdc.cancer.gov/").

Sort the WSIs (Slides) into a training, test, internal validation cohort

Based on a random split-sample approach, patients from GDC database were then randomly divided into a training cohort (60%), a testing cohort (10%), and an internal validation cohort (30%). Besides, the 67 patients from Sir Run-Run Shaw hospital were identified as external validation. The spreadsheets specifying four cohorts (training, test, internal and external validation cohorts) in three classifiers (normal-tumors.txt, grading.txt and mutation prediction.txt).

Smaller images named “Tiles”

Although the original WSI files contain all the necessary information, they are not directly applicable to train a deep CNN. Therefore, we crop them into much smaller image (Tiles), e.g. 256x256, that a typical deep CNN can handle.

Tile the images using the magnification (20x) and tile size of interest (256x256 px in following example):

```
python 0b_tileLoop4_deepzoom.py    -s 256 -e 0 -j 32 -B 50 -M 20 -o 256px_Tiled "downloaded_data/*/*svs"
```

Note: 0b_tileLoop4_deepzoom.py

Sort the dataset into a test, train and validation cohort for a 2-way classifier (Normal/HCC). You need to create a new directory and run this job from that directory

```
mkdir r1_sorted_2Cla
cd r1_sorted_2Cla
python ../ 0d_SortTiles.py --SourceFolder='../256px_Tiled/' --Magnification=20.0    --MagDiffAllowed=0 --SortingOption=6    --PatientID=12 --nSplit 0 --JsonFile='../clinical_data.json' --PercentTest=10 --PercentValid=35
```

Note: 0d_SortTiles.py

The option "6" might only compatible with this format. If you want to use the new formatted Json files, you will need to modify the code or use corresponded option number and create your own label file from it.

Once the process is complete, it should display how many tiles in each dataset and each class.

For example, a 2-way classifier (Normal/HCC):

The number of tiles in each dataset (first number is total)

```
Normal 24311
Normal_training 12614
Normal_test 2204
Normal_validation 9493
TCGA_HCC 74029
TCGA_HCC_training 41578
TCGA_HCC_test 8157
TCGA_HCC_ivalidation 24294
```

Note: For the slides from the GDC portal, we divided them into the tumor or normal tissues according to the label named “sample_type_id”, where “1” is the tumor and “11” represents normal tissues. Similarly, the histopathological grade based on the label named “neoplasm_histologic_grade” (High-level = well-differentiated (G1); medium = moderate-differentiated (G2); Notably, the low-level consisting of undifferentiation (G4) and low-differentiation (G3) in our study) and gene mutation information is from the gene report.

Training and testing deep-learning model

Method 1

- **Convert data into TFRecord files for each dataset**

```
mkdir r1_TFRecord_test
mkdir r1_TFRecord_valid
mkdir r1_TFRecord_train
python TFRecord_2or3_Classes/build_TF_test.py --directory='r1_sorted_2Cla/'  --output_directory='r1_TFRecord_test' --num_threads=1 --one_FT_per_Tile=False --ImageSet_basename='test'

python TFRecord_2or3_Classes/build_TF_test.py --directory='r1_sorted_2Cla/'  --output_directory='r1_TFRecord_valid' --num_threads=1 --one_FT_per_Tile=False --ImageSet_basename='valid'

python TFRecord_2or3_Classes/build_image_data.py --directory='r1_sorted_2Cla/' --output_directory='r1_TFRecord_train' --train_shards=1024  --validation_shards=128 --num_threads=16
```

Note: TFRecord_2or3_Classes/build_TF_test.py

TFRecord_2or3_Classes/build_image_data.py

- **Train the 2-way classifier**

```
mkdir r1_results
bazel build inception/imagenet_train
bazel-bin/inception/imagenet_train --num_gpus=4 --batch_size=400 --train_dir='r1_results' --data_dir='r1_TFRecord_train' --ClassNumber=2 --mode='0_softmax' --NbrOfImages=54192 --save_step_for_chekcpoint=1000 --max_steps=10000
```

As the first checkpoint appear, you can start running the validation set on it. Create a "labelref_r1.txt" text file with the list of possible classes. To run it in on loop on all existing checkpoints, the following code can be adapted:

```
mkdir r1_valid
export CHECKPOINT_PATH='/fullpath_to/r1_results'
export OUTPUT_DIR='/fullpath_to/r1_valid'
export DATA_DIR='r1_TFRecord_valid'
export LABEL_FILE='labelref_r1.txt'
# check if next checkpoint available
declare -i count=1000
declare -i step=1000
declare -i NbClasses=2
# create temporary directory for checkpoints
mkdir    -p $OUTPUT_DIR/tmp_checkpoints
export CUR_CHECKPOINT=$OUTPUT_DIR/tmp_checkpoints


while true; do
        echo $count
        if [ -f $CHECKPOINT_PATH/model.ckpt-$count.meta ]; then
                echo $CHECKPOINT_PATH/model.ckpt-$count.meta " exists"
                # check if there's already a computation for this checkpoinmt
                export TEST_OUTPUT=$OUTPUT_DIR/test_$count'k'
                if [ ! -d $TEST_OUTPUT ]; then
                        mkdir -p $TEST_OUTPUT


                        ln -s $CHECKPOINT_PATH/*-$count.*
$CUR_CHECKPOINT/.
                        touch $CUR_CHECKPOINT/checkpoint
                        echo 'model_checkpoint_path:
"'$CUR_CHECKPOINT'/model.ckpt-'$count'"' > $CUR_CHECKPOINT/checkpoint
                        echo 'all_model_checkpoint_paths:
"'$CUR_CHECKPOINT'/model.ckpt-'$count'"' >> $CUR_CHECKPOINT/checkpoint
```

```
                        # Test
                        python xClasses/nc_imagenet_eval.py --
checkpoint_dir=$CUR_CHECKPOINT --eval_dir=$OUTPUT_DIR --data_dir=$DATA_DIR
--batch_size 300    --run_once --ImageSet_basename='valid_' --ClassNumber $NbClasses --
mode='0_softmax'    --TVmode='test'
                        # wait

                        mv $OUTPUT_DIR/out* $TEST_OUTPUT/.

                        # ROC
                        export
OUTFILENAME=$TEST_OUTPUT/out_filename_Stats.txt
                        python
03_postprocessing/0h_ROC_MultiOutput_BootStrap.py --file_stats=$OUTFILENAME    --
output_dir=$TEST_OUTPUT --labels_names=$LABEL_FILE

                else
                        echo 'checkpoint '$TEST_OUTPUT' skipped'
                fi

        else
                echo $CHECKPOINT_PATH/model.ckpt-$count.meta " does not exist"
                break
        fi

        # next checkpoint
        count=`expr "$count" + "$step"`
done
```

```
# summarize all AUC per slide (average probability) for class 1:
ls -tr $OUTPUT_DIR/test_*/out2_roc_data_AvPb_c1a*    | sed -e
's/k\/out2_roc_data_AvPb_c1a/ /' | sed -e 's/test_/ /' | sed -e 's
/_/ /g' | sed -e 's/.txt//'      > $OUTPUT_DIR/valid_out2_AvPb_AUCs_1.txt

ls -tr $OUTPUT_DIR/test_*/out2_roc_data_AvPb_c2*    | sed -e
's/k\/out2_roc_data_AvPb_c2/ /' | sed -e 's/test_/ /' | sed -e 's/_/ /g' | sed -e 's/.txt//'      >
$OUTPUT_DIR/valid_out2_AvPb_AUCs_2.txt
```

Note:

xClasses/nc_imagenet_eval.py

0h_ROC_MultiOutput_BootStrap.py

The same code can be adapted to run the checkpoints on the internal and external validation set. Notably, for the tumor grading (G1, G2, and G3), and mutation prediction (Yes/No), we should delete the dataset of normal liver tissues. Besides, a similar code can be used for the validation check by modifying the corresponding options and inputs.

In this part, thanks to EASL DL team for helping us reword the codes; Thanks to Nicolas Coudray *e.t.* sharing the primary codes (The DeepPATH framework, https://github.com/ncoudray/DeepPATH)

Method 2

Summary about using EASY DL

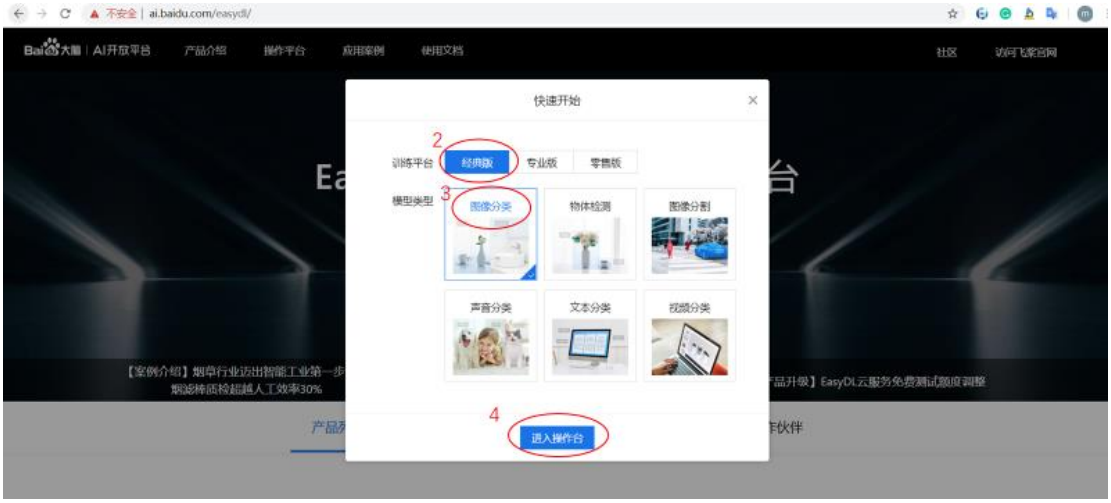
- **Prepare and upload the training set**

If you never use the EASY DL, you need register an account at first.

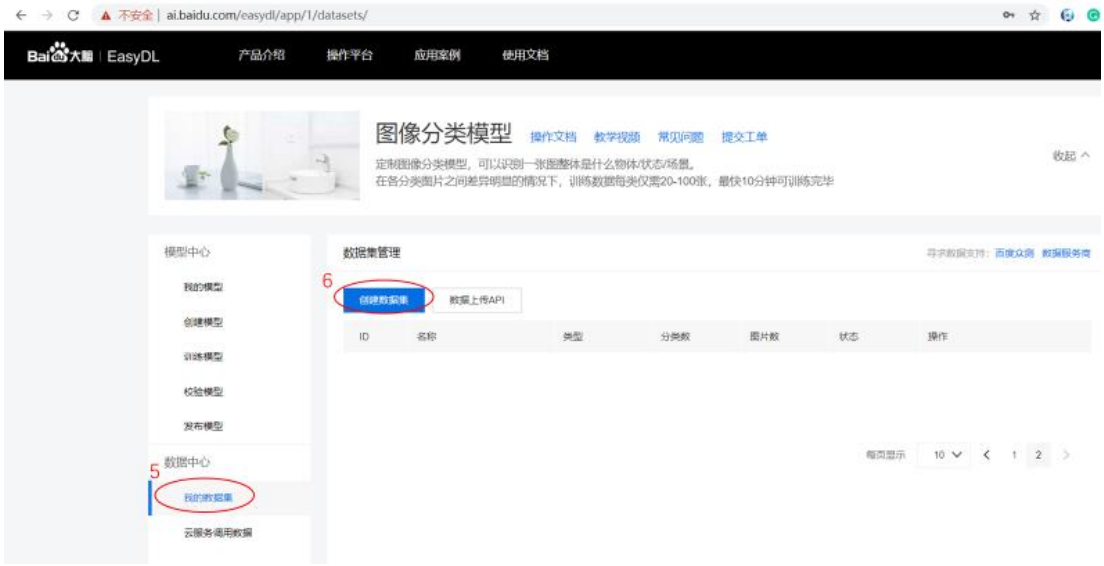
EASY DL(https://ai.baidu.com/easydl/)

1. Click to the button “begin”

2. Training platform: Choose “classic version”
3. Model type: choose “image classification”
4. Confirm and enter the platform



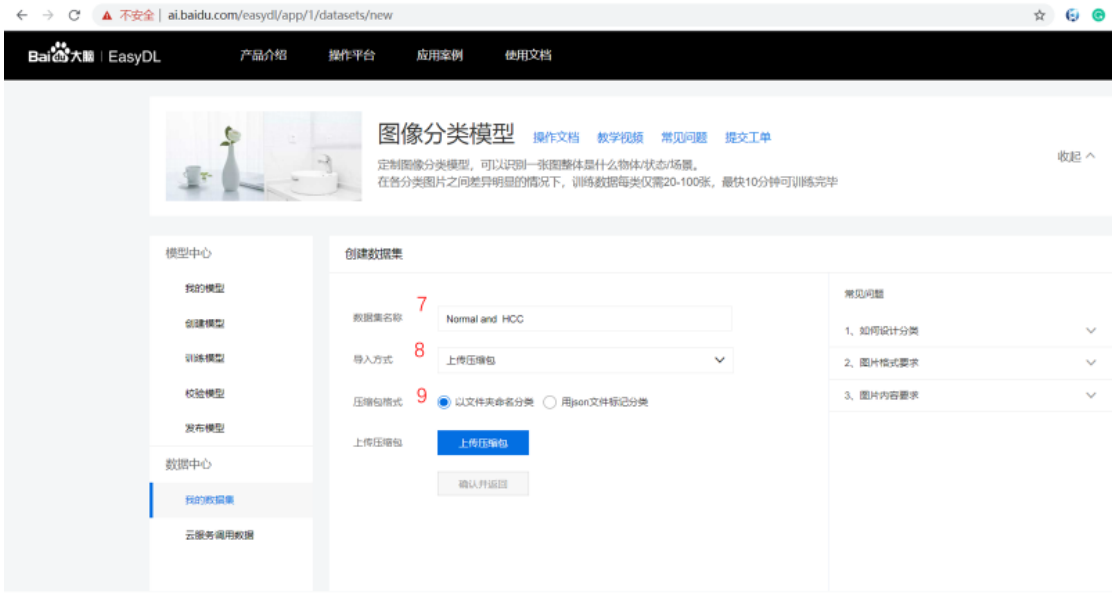
5. Database: choose “my database”
6. Management of database: choose “building new database”



7. Database name: for example, “normal and HCC”
8. Upload method: upload all images using “zip” or “one-by-one image”.

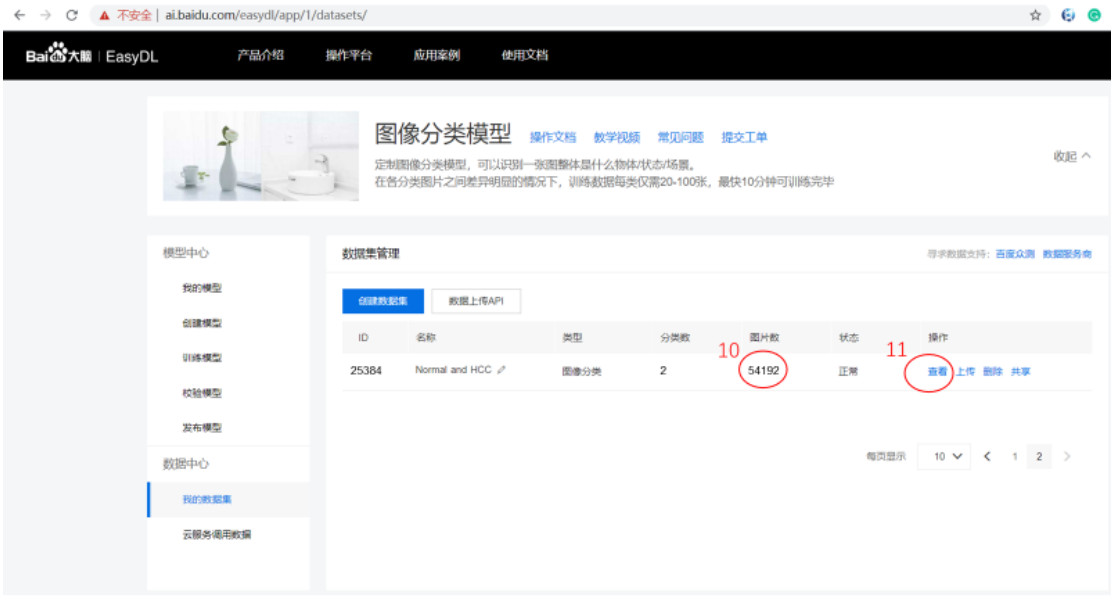
Note: The former one is recommended for more than 1000 images.

9. Label for subgroup: “using file name directly” or “json”.



10. Check the number of images (total number)

If you want to check each subgroup number and each class, click the “details”.

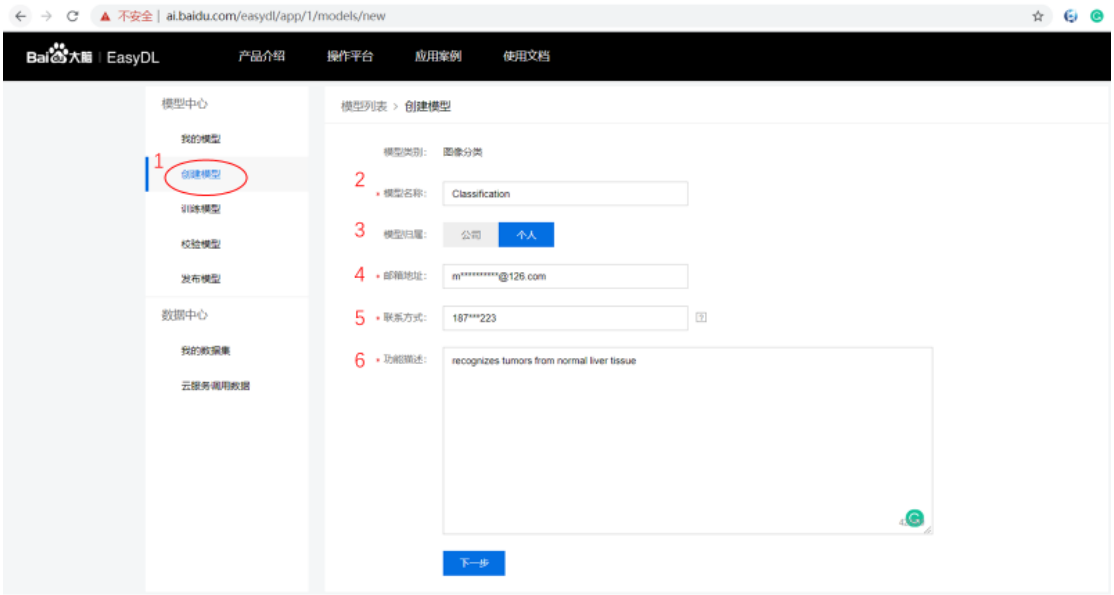


- **Training deep-learning model**

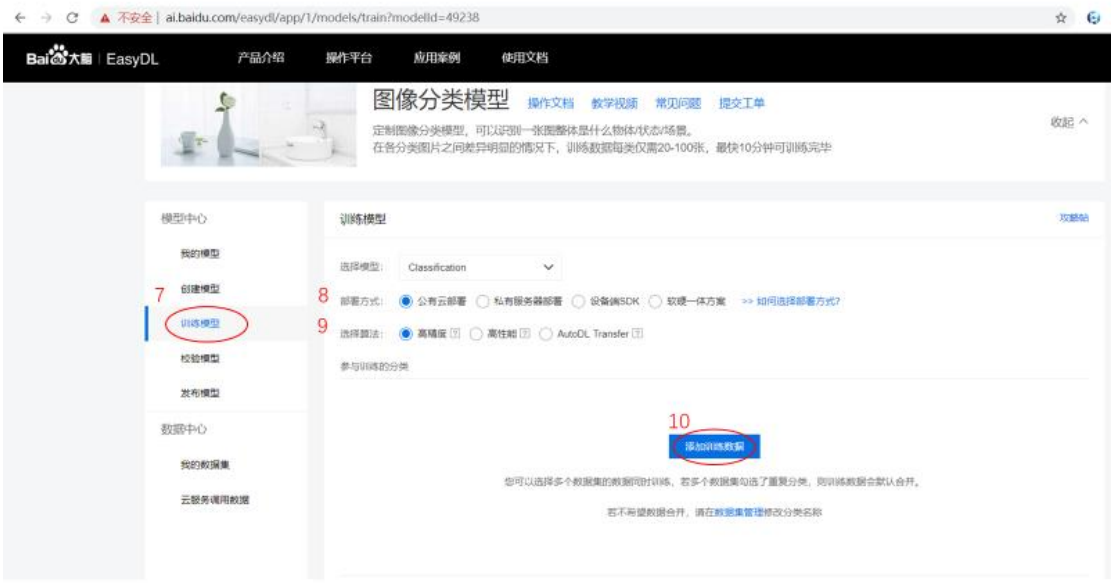
1. Building new model

Basic information

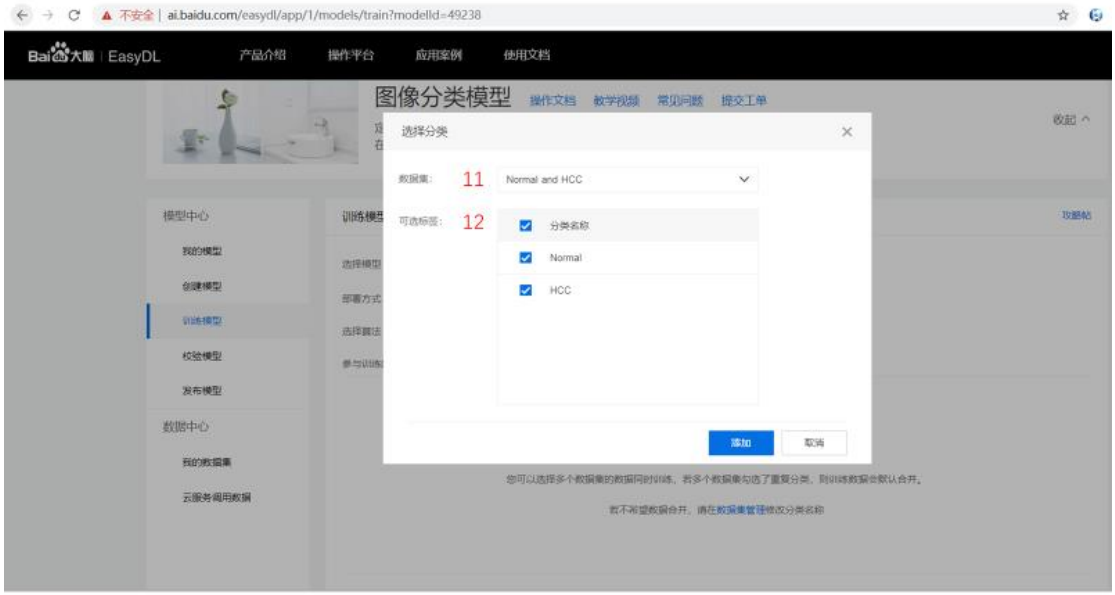
2. Model name: for example, “Classification”
3. Model belongs to “company” or “private”
4. Email
5. Telephone
6. Function description



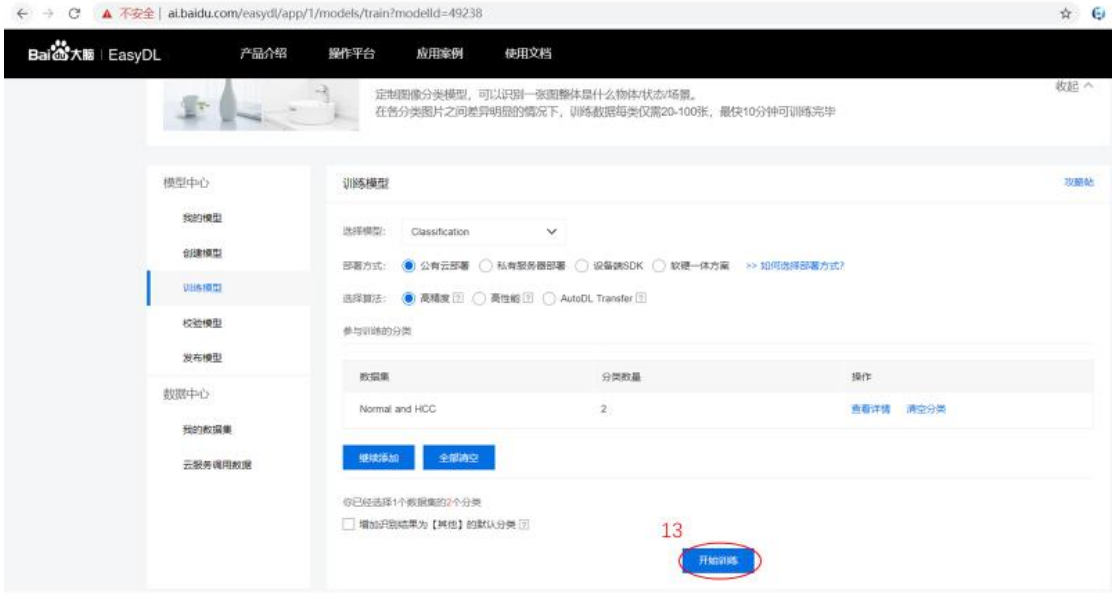
7. Training model
8. Publish way: Public iClouds, Private iClouds, PC/iPhone, Other
9. Algorithm: there are three types, please choose “Auto DL Transfer”
10. Add database



11. Choose database for training: For example, Normal and HCC (details summarized in 4.1)
12. Choose the group in the related database.

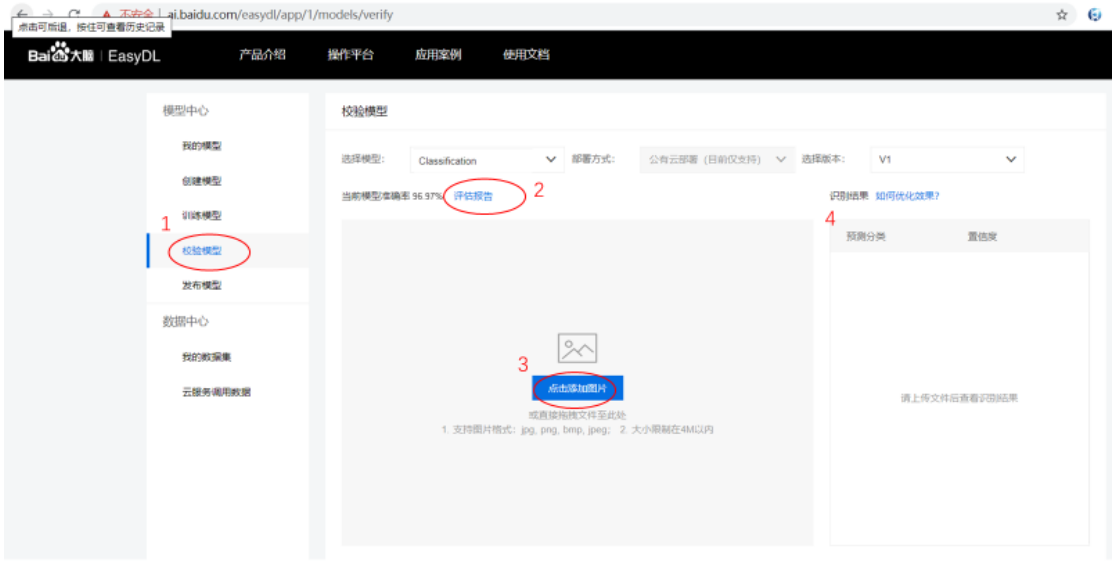


13. Confirm and begin to training. Different models need different time for build. You didn't need to wait the page. When training model finished, the system would note you by message. Therefore, you should make sure that the telephone number is available in the basic information.



- **Test/validate the model**

1. Test the deep-learning model
2. Evaluation reports: include F1-score, Precision, Accuracy, and Recall rate, False Positive Rate, True Positive Rate.
3. Upload image
4. The Prediction result: probability, 95%IC

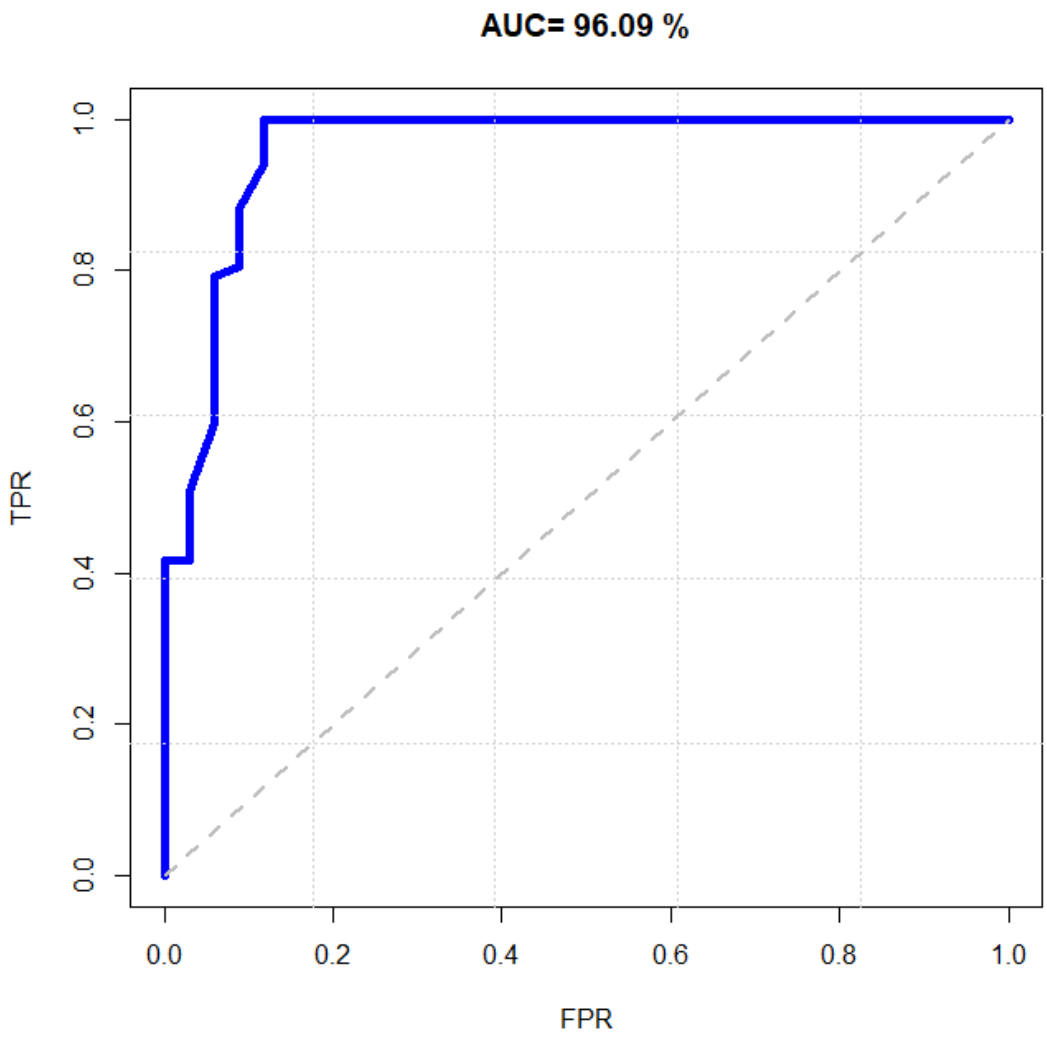


Performance of the model

Data: N_H_e_validation.txt; **Code:** ROC/AUC/PR-curves.R (Performed by R 3.6.0)

- **ROC, AUC**

```
a <- read.table("N_H_e_Validation.txt")
a <- as.matrix(a)
label <- a[,2]
decision <- a[,1]
ngrids <- 100
TPR <- rep(0, ngrids)
FPR <- rep(0, ngrids)
p0 <- rep(0, ngrids)
for(i in 1:ngrids)
{
   p0[i] <- i/ngrids
   pred_label <- 1*(decision > p0[i])
   TPR[i] <- sum(pred_label * label) / sum(label)
   FPR[i] <- sum(pred_label * (1-label)) / sum(1-label)
}
## calculate AUC
pos.decision <- decision[which(label == 1)]
neg.decision <- decision[which(label == 0)]
aucs <- replicate(2000,mean(sample(pos.decision,1000,replace=T) >
sample(neg.decision,1000,replace=T)))
auc2 <- round(mean(aucs),4)
## or
#auc <- mean(sample(pos.decision,1000,replace=T) > sample(neg.decision,1000,replace=T))
plot(FPR, TPR, col=4,lwd=5, type="l", main=paste("AUC=",auc2*100,"%"))
grid(5, 5, lwd = 1)
points(c(0,1), c(0,1), type="l", lty=2, lwd=2, col="grey")
```



- **Accuracy, Precision, Recall, F1, MCC**

```
cut.op <- p0[which(TPR-FPR == max(TPR-FPR))]

cut.op

##calculate Accuracy, Precision, Recall, F1, MCC

TP<-sum(pos.decision>cut.op)

FN<-sum(pos.decision<cut.op)

FP<-sum(neg.decision>cut.op)

TN<-sum(neg.decision<cut.op)

Accuracy<-(TP+TN)/(TP+TN+FP+FN)

Precision<-TP/(TP+FP)

Recall<-TP/(TP+FN)

F1<-2*(Precision*Recall)/(Precision+Recall)
```

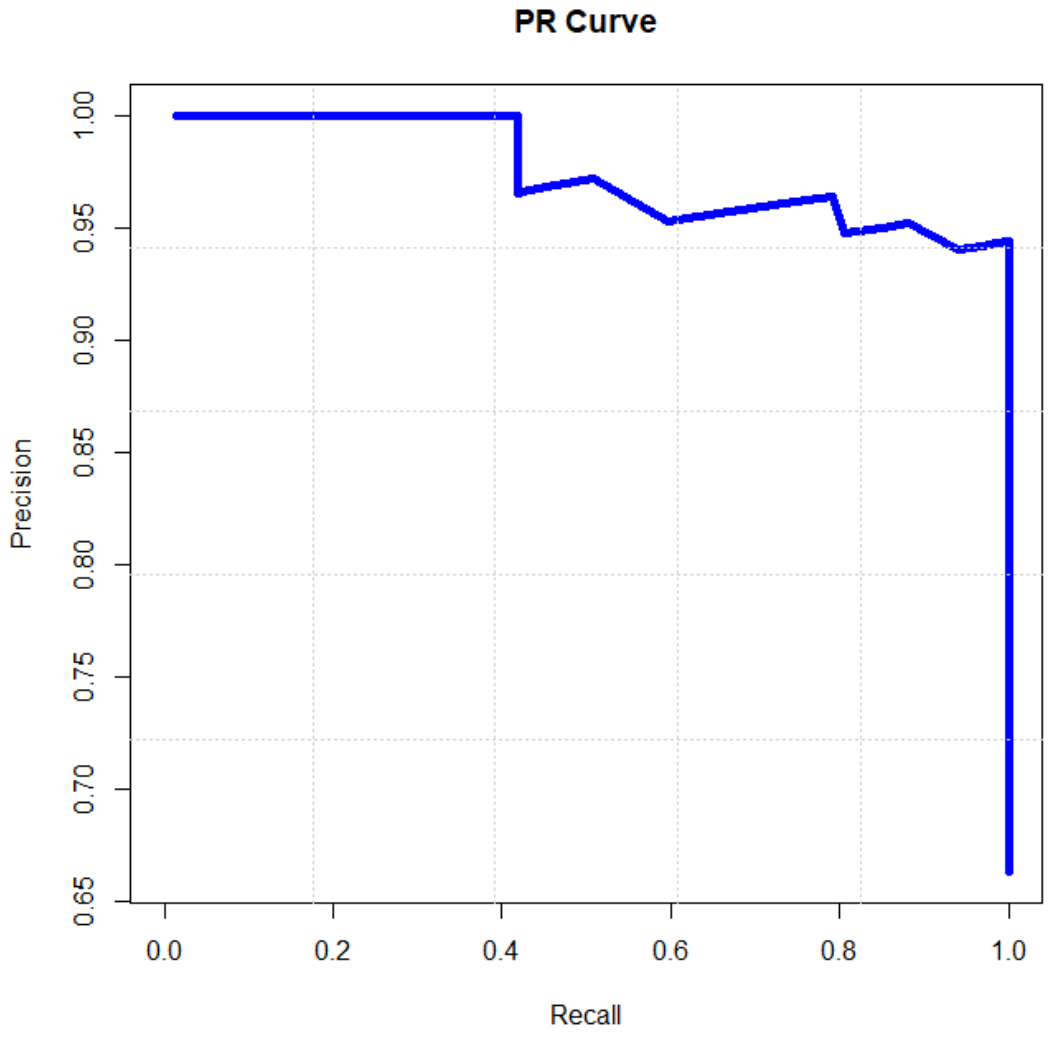
```
MCC<- (TP*TN-FP*FN)/(sqrt((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN)))

Accuracy

Precision

Recall

F1
```

```
> Accuracy
[1] 0.960396
> Precision
[1] 0.943662
> Recall
[1] 1
> F1
[1] 0.971014
> MCC
[1] 0.912493
```

- **Precision-recall curves (PR-curves) (Performed by R 3.6.0)**

```
</pre>

a <- read.table("N_H_e_Validation.txt")

a <- as.matrix(a)

label <- a[,2]

decision <- a[,1]

ngrids <- 100

c<-ngrids-1

P <- rep(0, ngrids)

R <- rep(0, ngrids)

p0 <- rep(0, ngrids)

A <- rep(0, ngrids)

for(i in 0:ngrids)

{

   p0[i] <- i/ngrids

   pred_label <- 1*(decision > p0[i])

   R[i] <- sum(pred_label * label) / sum(label)
```

```
   P[i] <- sum(pred_label * label) / sum(pred_label)
   A[i] <- sum((pred_label == label)*1)/nrow(a)
}
plot(R, P, col=4,lwd=5, type="l",xlab="Recall",ylab="Precision", main="PR Curve")
grid(5, 5, lwd = 1)
accuracy <- max(A)
accuracy
<pre>
```



Note: If you use the "method 2", the F1-scores, precision, accuracy, recall rates were automatically calculated. However, you need summarized the predicted result of each slide into a spreadsheet for calculating AUC, and drawing ROC and PR-curves.
You used Python scikit-learn, you can obtain the codes form github (https://github.com/9468305/python-script/tree/master/auc_pr_roc/)