```python
import pandas as pd
import numpy as np
import torch
import torchvision
import random
from PIL import Image
import cv2
import os
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
!pip install evaluate jiwer
```

```
Collecting evaluate
  Downloading evaluate-0.4.0-py3-none-any.whl (81 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.4/81.4 kB 2.9 MB/s eta
0:00:00
ent already satisfied: huggingface-hub>=0.7.0 in
/opt/conda/lib/python3.7/site-packages (from evaluate) (0.10.1)
Requirement already satisfied: requests>=2.19.0 in
/opt/conda/lib/python3.7/site-packages (from evaluate) (2.28.1)
Requirement already satisfied: xxhash in
/opt/conda/lib/python3.7/site-packages (from evaluate) (3.0.0)
Requirement already satisfied: packaging in
/opt/conda/lib/python3.7/site-packages (from evaluate) (21.3)
Requirement already satisfied: numpy>=1.17 in
/opt/conda/lib/python3.7/site-packages (from evaluate) (1.21.6)
Requirement already satisfied: dill in /opt/conda/lib/python3.7/site-
packages (from evaluate) (0.3.5.1)
Requirement already satisfied: importlib-metadata in
/opt/conda/lib/python3.7/site-packages (from evaluate) (4.13.0)
Requirement already satisfied: responses<0.19 in
/opt/conda/lib/python3.7/site-packages (from evaluate) (0.18.0)
Requirement already satisfied: fsspec[http]>=2021.05.0 in
/opt/conda/lib/python3.7/site-packages (from evaluate) (2022.8.2)
Requirement already satisfied: datasets>=2.0.0 in
/opt/conda/lib/python3.7/site-packages (from evaluate) (2.1.0)
Requirement already satisfied: multiprocess in
/opt/conda/lib/python3.7/site-packages (from evaluate) (0.70.13)
Requirement already satisfied: tqdm>=4.62.1 in
/opt/conda/lib/python3.7/site-packages (from evaluate) (4.64.0)
Requirement already satisfied: pandas in
/opt/conda/lib/python3.7/site-packages (from evaluate) (1.3.5)
Collecting levenshtein==0.20.2
  Downloading Levenshtein-0.20.2-cp37-cp37m-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.4 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 24.1 MB/s eta
0:00:00a 0:00:01
ent already satisfied: rapidfuzz<3.0.0,>=2.3.0 in
/opt/conda/lib/python3.7/site-packages (from levenshtein==0.20.2-
>jiwer) (2.11.1)
Requirement already satisfied: pyarrow>=5.0.0 in
```

```
/opt/conda/lib/python3.7/site-packages (from datasets>=2.0.0-
>evaluate) (5.0.0)
Requirement already satisfied: aiohttp in
/opt/conda/lib/python3.7/site-packages (from datasets>=2.0.0-
>evaluate) (3.8.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/opt/conda/lib/python3.7/site-packages (from huggingface-hub>=0.7.0-
>evaluate) (4.1.1)
Requirement already satisfied: pyyaml>=5.1 in
/opt/conda/lib/python3.7/site-packages (from huggingface-hub>=0.7.0-
>evaluate) (6.0)
Requirement already satisfied: filelock in
/opt/conda/lib/python3.7/site-packages (from huggingface-hub>=0.7.0-
>evaluate) (3.7.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/opt/conda/lib/python3.7/site-packages (from packaging->evaluate)
(3.0.9)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/conda/lib/python3.7/site-packages (from requests>=2.19.0-
>evaluate) (1.26.12)
Requirement already satisfied: charset-normalizer<3,>=2 in
/opt/conda/lib/python3.7/site-packages (from requests>=2.19.0-
>evaluate) (2.1.0)
Requirement already satisfied: idna<4,>=2.5 in
/opt/conda/lib/python3.7/site-packages (from requests>=2.19.0-
>evaluate) (3.3)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.7/site-packages (from requests>=2.19.0-
>evaluate) (2022.9.24)
Requirement already satisfied: zipp>=0.5 in
/opt/conda/lib/python3.7/site-packages (from importlib-metadata-
>evaluate) (3.8.0)
Requirement already satisfied: python-dateutil>=2.7.3 in
/opt/conda/lib/python3.7/site-packages (from pandas->evaluate) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in
/opt/conda/lib/python3.7/site-packages (from pandas->evaluate)
(2022.1)
Requirement already satisfied: multidict<7.0,>=4.5 in
/opt/conda/lib/python3.7/site-packages (from aiohttp->datasets>=2.0.0-
>evaluate) (6.0.2)
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in
/opt/conda/lib/python3.7/site-packages (from aiohttp->datasets>=2.0.0-
>evaluate) (4.0.2)
Requirement already satisfied: frozenlist>=1.1.1 in
/opt/conda/lib/python3.7/site-packages (from aiohttp->datasets>=2.0.0-
>evaluate) (1.3.0)
Requirement already satisfied: yarl<2.0,>=1.0 in
/opt/conda/lib/python3.7/site-packages (from aiohttp->datasets>=2.0.0-
>evaluate) (1.7.2)
Requirement already satisfied: asynctest==0.13.0 in
```

```
/opt/conda/lib/python3.7/site-packages (from aiohttp->datasets>=2.0.0-
>evaluate) (0.13.0)
Requirement already satisfied: aiosignal>=1.1.2 in
/opt/conda/lib/python3.7/site-packages (from aiohttp->datasets>=2.0.0-
>evaluate) (1.2.0)
Requirement already satisfied: attrs>=17.3.0 in
/opt/conda/lib/python3.7/site-packages (from aiohttp->datasets>=2.0.0-
>evaluate) (21.4.0)
Requirement already satisfied: six>=1.5 in
/opt/conda/lib/python3.7/site-packages (from python-dateutil>=2.7.3-
>pandas->evaluate) (1.15.0)
Installing collected packages: levenshtein, jiwer, evaluate
  Attempting uninstall: levenshtein
    Found existing installation: Levenshtein 0.20.7
    Uninstalling Levenshtein-0.20.7:
      Successfully uninstalled Levenshtein-0.20.7
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
python-levenshtein 0.20.7 requires Levenshtein==0.20.7, but you have
levenshtein 0.20.2 which is incompatible.
Successfully installed evaluate-0.4.0 jiwer-2.5.1 levenshtein-0.20.2
WARNING: Running pip as the 'root' user can result in broken
permissions and conflicting behaviour with the system package manager.
It is recommended to use a virtual environment instead:
https://pip.pypa.io/warnings/venv
```

```python
"""Seed everything!"""
random.seed(42)
os.environ['PYTHONHASHSEED'] = str(42)
np.random.seed(42)
torch.manual_seed(42)
torch.cuda.manual_seed(42)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = True
```

Load dataset

Tokenize car numbers

```python
# Get the list of car numbers
def exec_text(path):
    return path[path.find('-') + 1:path.find('.')]

input_dir_train = '/kaggle/input/labtinkoff/CCPD2019-dl1/train'

car_numbers = [exec_text(path) for  path in
os.listdir(input_dir_train)]
```

```python
# Get the alphabet of symbols from all car numbers
seq = ''
for car_number in car_numbers:
    seq += car_number
alphabet = ''
for symbol in sorted(set(seq)):
    alphabet += symbol
alphabet
```

```
'0123456789ABCDEFGHJKLMNOPQRSTUVWXYZ云京冀吉宁川新晋桂沪津浙渝湘琼甘皖粤苏蒙藏豫贵赣辽鄂闽陕青鲁黑'
```

```python
OOV_TOKEN = '<OOV>' # out of vocabulary token
CTC_BLANK = '<BLANK>' # token for ctc matrix
PAD_TOKEN = '<PAD>' # padding token


def get_char_map(alphabet):
    """Make from string alphabet character2int dict.
    Add BLANK char for CTC loss and OOV char for out of vocabulary symbols."""
    char_map = {value: idx + 3 for (idx, value) in enumerate(alphabet)}
    char_map[CTC_BLANK] = 0
    char_map[OOV_TOKEN] = 1
    char_map[PAD_TOKEN] = 2
    return char_map


class Tokenizer:
    """Class for encoding and decoding string word to sequence of int
    (and vice versa) using alphabet."""

    def __init__(self, alphabet):
        self.char_map = get_char_map(alphabet)
        self.rev_char_map = {val: key for key, val in self.char_map.items()}

    def encode(self, word_list):
        enc_words = []
        for word in word_list:
            enc_words.append(
                [self.char_map[char] if char in self.char_map
                 else self.char_map[OOV_TOKEN]
                 for char in word]
            )
        return enc_words

    def get_num_chars(self):
        return len(self.char_map)
    def decode(self, enc_word_list):
        dec_words = []
        for word in enc_word_list:
            word_chars = ''
            for idx, char_enc in enumerate(word):
                if (
                    char_enc != self.char_map[OOV_TOKEN]
                    and char_enc != self.char_map[CTC_BLANK]
                    and not (idx > 0 and char_enc == word[idx - 1])
                ):
                    word_chars += self.rev_char_map[char_enc]
            dec_words.append(word_chars)
        return dec_words

tokenizer = Tokenizer(alphabet)

class Laba_dataset(torch.utils.data.Dataset):
    def __init__(self, root, tokenizer, transform=None):
        self.root = root
        self.transform = transform
        self.tokenizer = tokenizer
        self.img_paths = [os.path.join(self.root, img_path) for
img_path in os.listdir(self.root)]
        self.text = [exec_text(path) for path in
os.listdir(self.root)]
        self.enc_text = self.tokenizer.encode(self.text)

    def __getitem__(self, ind):
        img = Image.open(self.img_paths[ind]) # resize
        if self.transform is not None:
            img = self.transform(img) # make some augmentations
        # return image, encoded_text, source_text
        return (img, torch.LongTensor(self.enc_text[ind]),
self.text[ind])

    def __len__(self):
        return len(self.img_paths)

def collate_fn(batch):
    images, enc_texts, texts = zip(*batch)
    images = torch.stack(images, 0)
    enc_pad_texts = torch.nn.utils.rnn.pad_sequence(enc_texts,
batch_first=True, padding_value=tokenizer.char_map[PAD_TOKEN])
    return images, enc_pad_texts, texts

from sklearn.model_selection import train_test_split
batch_size = 128
transform = torchvision.transforms.Compose([
    torchvision.transforms.Resize((32, 128)),
    torchvision.transforms.RandomRotation(5),
    torchvision.transforms.ColorJitter(),
    torchvision.transforms.GaussianBlur(3),
    torchvision.transforms.ToTensor()
        ])
dataset_full = Laba_dataset(input_dir_train, tokenizer,
transform=transform)
# split full dataset
train_idx, valid_idx =
train_test_split(list(range(len(dataset_full))), train_size=0.9)
dataset = {
    'train': torch.utils.data.Subset(dataset_full, train_idx),
    'valid': torch.utils.data.Subset(dataset_full, valid_idx)
}

dataset_size = {ds: len(dataset[ds]) for ds in ['train', 'valid']}

dataloader = {
    'train': torch.utils.data.DataLoader(
        dataset=dataset['train'], batch_size=batch_size, shuffle=True,
collate_fn=collate_fn
    ),
    'valid': torch.utils.data.DataLoader(
        dataset=dataset['valid'], batch_size=batch_size,
shuffle=False, collate_fn=collate_fn
    ),
}

input_dir_test = '/kaggle/input/labtinkoff/CCPD2019-dl1/test'
batch_size = 128
transform_test = torchvision.transforms.Compose([
    torchvision.transforms.Resize((32, 128)),
    torchvision.transforms.ToTensor()
        ])
dataset_test = Laba_dataset(input_dir_test, tokenizer,
transform=transform_test)
dataloader_test = torch.utils.data.DataLoader(
        dataset=dataset_test, batch_size=batch_size, shuffle=False,
collate_fn=collate_fn
    )
```

```
next(iter(dataloader['train']))[0].shape
```

```
torch.Size([128, 3, 32, 128])
```

```
img = torchvision.transforms.ToPILImage()(dataset_full[173]
[0].squeeze(0))
```

```
img
```

```
dataset_test[122]
```

```
(tensor([[[0.5451, 0.5529, 0.5608,  ..., 0.4157, 0.4118, 0.4706],
          [0.4431, 0.4039, 0.4039,  ..., 0.4157, 0.5020, 0.5647],
          [0.3333, 0.2980, 0.3059,  ..., 0.2353, 0.3137, 0.4353],
          ...,
          [0.3333, 0.3137, 0.2667,  ..., 0.1608, 0.1490, 0.1843],
          [0.3216, 0.3176, 0.3098,  ..., 0.4471, 0.4706, 0.5020],
          [0.3255, 0.3333, 0.3529,  ..., 0.3922, 0.4314, 0.4471]],

         [[0.5059, 0.5059, 0.5059,  ..., 0.4745, 0.4706, 0.5373],
          [0.4275, 0.3686, 0.3608,  ..., 0.5216, 0.6000, 0.6627],
          [0.3569, 0.3020, 0.2980,  ..., 0.3804, 0.4431, 0.5490],
          ...,
          [0.4431, 0.4235, 0.3725,  ..., 0.2353, 0.2392, 0.2824],
          [0.4157, 0.4157, 0.4078,  ..., 0.5255, 0.5647, 0.6000],
          [0.4157, 0.4235, 0.4431,  ..., 0.4784, 0.5255, 0.5412]],

         [[0.6275, 0.6353, 0.6431,  ..., 0.5451, 0.5176, 0.5333],
          [0.5961, 0.5765, 0.6000,  ..., 0.6000, 0.6353, 0.6471],
          [0.5490, 0.5725, 0.6471,  ..., 0.5059, 0.4902, 0.5294],
          ...,
          [0.4314, 0.4471, 0.4510,  ..., 0.2980, 0.2039, 0.2000],
          [0.3882, 0.3922, 0.3922,  ..., 0.5529, 0.5216, 0.5294],
          [0.3922, 0.3961, 0.4118,  ..., 0.4784, 0.4863, 0.4941]]]),
 tensor([54, 13,  5,  8, 33,  8, 12]),
 '皖A25V59')
```

Define model

```
from torch import nn

# To solve the problem, I used the standard CRNN structure

class ResNetBlock(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size=3,
stride=1, padding=0, dropout=0.15):
        super().__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size,
stride, padding, bias=False)
        self.bn = nn.BatchNorm2d(out_channels)
        self.relu = nn.LeakyReLU()
        self.dropout = nn.Dropout(dropout)
        self.downsample = None
        if in_channels != out_channels:
            self.downsample = nn.Conv2d(in_channels, out_channels, 1,
stride=2)

    def forward(self, x, identity=True):
        out = self.dropout(self.bn(self.conv(x)))
        if identity:
            if self.downsample is not None:
                x = self.downsample(x)
            return self.relu(out + x)
        else:
            return self.relu(out)

class CNN(nn.Module):
    def __init__(self, in_channels=1, num_layers=2, dropout=0.1):
        super().__init__()
        """
        As feature extractor i use resnet, passing through the cut the 
images are
        transformed from the dimension tensor (C: 1, W: 128, H: 32) to 
the
        dimension tensor (C: 1, W: 4, H: 1)
        """
        self.start = ResNetBlock(3, 64, 7, 1, 0, 0.0)
        self.maxpool = nn.MaxPool2d(3, 2, 1)
        self.blocks1 = nn.ModuleList([ResNetBlock(64, 64, padding=1) 
for _ in range(num_layers)])
        self.blocks2 = nn.ModuleList([ResNetBlock(64, 128, padding=1, 
stride=2)] + [ResNetBlock(128, 128, padding=1) for _ in 
range(num_layers)])
        self.blocks3 = nn.ModuleList([ResNetBlock(128, 256, padding=1, 
stride=2)] + [ResNetBlock(256, 256, padding=1) for _ in 
range(num_layers)])
        self.blocks4 = nn.ModuleList([ResNetBlock(256, 512, padding=1, 
stride=2)] + [ResNetBlock(512, 512, padding=1) for _ in 
range(num_layers)])
        self.blocks5 = nn.ModuleList([ResNetBlock(512, 1024, 
padding=1, stride=2)] + [ResNetBlock(1024, 1024, padding=1) for _ in 
range(num_layers)])
        self.blocks = [self.blocks1, self.blocks2, self.blocks3, 
self.blocks4, self.blocks5]

    def forward(self, x):
        out = self.maxpool(self.start(x, identity=False))
        for blocks in self.blocks:
            for block in blocks:
                out = block(out)

        return out

class BiLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, 
dropout=0.1):
        super().__init__()
        self.lstm = nn.LSTM(
            input_size, hidden_size, num_layers,
            dropout=dropout, batch_first=True, bidirectional=True)
    def forward(self, x):
        out, _ = self.lstm(x)
        return out


class CRNN(nn.Module):
    def __init__(
        self, number_class_symbols, time_feature_count=256,
lstm_hidden=256,
        lstm_len=3,
    ):
        super().__init__()
        self.feature_extractor = CNN(dropout=0.15)
        self.avg_pool = nn.AdaptiveAvgPool2d(
            (time_feature_count, time_feature_count))
        self.bilstm = BiLSTM(time_feature_count, lstm_hidden,
lstm_len, dropout=0.15)
        self.classifier = nn.Sequential(
            nn.Linear(lstm_hidden * 2, time_feature_count),
            nn.GELU(),
            nn.Dropout(0.15),
            nn.Linear(time_feature_count, number_class_symbols)
        )

    def forward(self, x):
        x = self.feature_extractor(x)
        b, c, h, w = x.size()
        x = x.view(b, c * h, w)
        x = self.avg_pool(x)
        x = x.transpose(1, 2)
        x = self.bilstm(x)
        x = self.classifier(x)
        x = nn.functional.log_softmax(x, dim=2).permute(1, 0, 2)
        return x
```

Define accuracy metric for evaluate validation dataset

```
class AverageMeter:
    def __init__(self):
        self.reset()

    def reset(self):
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count
```

```
def get_accuracy(y_true, y_pred):
    scores = []
    for true, pred in zip(y_true, y_pred):
        scores.append(true == pred)
    avg_score = np.mean(scores)
    return avg_score
```

Training loop

```
import pickle as pkl

def safe(obj, filename):
    with open(filename, 'wb') as outp:
        pkl.dump(obj, outp)

def read(filename):
    with open(filename, 'rb') as inp:
        return pkl.load(inp)

def weights_init(m):
    classname = m.__class__.__name__
    if type(m) in [nn.Linear, nn.Conv2d, nn.Conv1d]:
        torch.nn.init.xavier_uniform_(m.weight)
        if m.bias is not None:
            m.bias.data.fill_(0.01)
    elif classname.find('BatchNorm') != -1:
        m.weight.data.normal_(1.0, 0.02)
        m.bias.data.fill_(0)

def val_loop(data_loader, model, tokenizer, device):
    acc_avg = AverageMeter()
    for images, enc_texts, texts in data_loader:
        batch_size = len(texts)
        text_preds = predict(images, model, tokenizer, device)
        acc_avg.update(get_accuracy(texts, text_preds), batch_size)
    print(f'Validation, acc: {acc_avg.avg:.4f}')
    return acc_avg.avg

def predict(images, model, tokenizer, device):
    model.eval()
    images = images.to(device)
    with torch.no_grad():
        output = model(images)
    pred = torch.argmax(output.detach().cpu(), -1).permute(1,
0).numpy()
    text_preds = tokenizer.decode(pred)
    return text_preds

def val_loop_ensemble(data_loader, models, tokenizer, device):
    acc_avg = AverageMeter()
    for images, enc_texts, texts in data_loader:
        batch_size = len(texts)
        text_preds = predict_ensemble(images, models, tokenizer,
device)
        acc_avg.update(get_accuracy(texts, text_preds), batch_size)
    print(f'Validation, acc: {acc_avg.avg:.4f}')
    return acc_avg.avg

def predict_ensemble(images, models, tokenizer, device):
    [model.eval() for model in models]
    images = images.to(device)
    with torch.no_grad():
        output = sum([model(images) for model in models]) /
len(models)
    pred = torch.argmax(output.detach().cpu(), -1).permute(1,
0).numpy()
    text_preds = tokenizer.decode(pred)
    return text_preds

def train_loop(data_loader, model, criterion, optimizer, epoch):
    loss_avg = AverageMeter()
    model.train()
    for images, enc_texts, texts in data_loader:
        model.zero_grad()
        images = images.to(device)
        batch_size = len(texts)
        output = model(images)
        output_lenghts = torch.full(
            size=(output.size(1),),
            fill_value=output.size(0),
            dtype=torch.long
        )
        text_lens = torch.LongTensor([len(text) for text in texts])
        loss = criterion(output, enc_texts, output_lenghts, text_lens)
        loss_avg.update(loss.item(), batch_size)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 2)
        optimizer.step()
    for param_group in optimizer.param_groups:
        lr = param_group['lr']
    print(f'\nEpoch {epoch}, Loss: {loss_avg.avg:.5f}, LR: {lr:.7f}')
    return loss_avg.avg

def train(dataloader, epochs):
    train_loader, val_loader = dataloader['train'],
dataloader['valid']
    model = CRNN(number_class_symbols=tokenizer.get_num_chars())
    model.apply(weights_init)
    model.to(device)
    criterion = torch.nn.CTCLoss(blank=0, reduction='mean',
zero_infinity=True)
    optimizer = torch.optim.AdamW(model.parameters(), lr=0.001,
                                  weight_decay=0.01)
    scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
        optimizer=optimizer, mode='max', factor=0.5, patience=5)
    best_acc = -np.inf
    acc_avg = val_loop(val_loader, model, tokenizer, device)
    for epoch in range(epochs):
        loss_avg = train_loop(train_loader, model, criterion,
optimizer, epoch)
        acc_avg = val_loop(val_loader, model, tokenizer, device)
        scheduler.step(acc_avg)
        if acc_avg > best_acc:
            best_acc = acc_avg
        safe(model, f'model_{epoch}')
```

```
train(dataloader, 8)
```

```
Validation, acc: 0.0000

Epoch 0, Loss: 1.52695, LR: 0.0010000
Validation, acc: 0.8252

Epoch 1, Loss: 0.08662, LR: 0.0010000
Validation, acc: 0.9335

Epoch 2, Loss: 0.05117, LR: 0.0010000
Validation, acc: 0.9433

Epoch 3, Loss: 0.03628, LR: 0.0010000
Validation, acc: 0.9570

Epoch 4, Loss: 0.03567, LR: 0.0010000
Validation, acc: 0.9253

Epoch 5, Loss: 0.03618, LR: 0.0010000
Validation, acc: 0.9701

Epoch 6, Loss: 0.02495, LR: 0.0010000
Validation, acc: 0.9591

Epoch 7, Loss: 0.02684, LR: 0.0010000
Validation, acc: 0.9676
```

```
model = read('/kaggle/working/model_5') # load model with best acc on
validation
```

```
img, enc_label, label = dataset_full[2001]
```

```
pred = predict(img.unsqueeze(0).to(device), model, tokenizer, device)
# sample pred
pred
```

```
['皖KLJ029']
```

```
real_img = torchvision.transforms.ToPILImage()(img)
real_img
```

Compute metrics

```
from evaluate import load
cer = load("cer")
```

```
{"version_major":2,"version_minor":0,"model_id":"15cd030f4c6147509acee
f3bae3b6bf2"}
```

```
references = dataset_test.text
```

```
predictions = []
for imgs, enc_text, text in dataloader_test:
    predictions += predict(imgs, model, tokenizer, device)
```

```
cer.compute(predictions=predictions, references=references)
```

```
0.008486562942008486
```

```
len(references) == len(predictions)
```

```
True
```

```
errors = {} # dict of errors {predictions: references}
for pred, refer in zip(predictions, references):
    if cer.compute(predictions=[pred], references=[refer]) != 0.0:
        errors[pred] = refer
```

```
errors
```

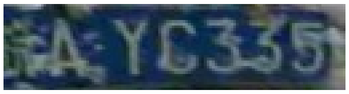
```
{'苏AYC335': '皖AYC335',
 '皖AL8D': '皖NL108D',
 '皖AF9098': '皖AF888S',
 '苏D726BX': '苏D726PX',
 '苏A475C': '苏A4759C',
 '皖AN5R29': '冀AN5R29',
 '浙JS2363': '川JS2363',
 '鄂AQE526': '沪AQE526',
 '皖A06Z16': '皖AD6Z16',
 '粤R8A001': '新R8A001',
 '皖ADL439': '甘ADL439',
 '皖AZZ2G8': '皖AZZ208',
 '皖CKB654': '粤CKB654',
 '赣E269JY': '湘E269JY',
 '浙AN08Q3': '豫AN08Q3',
 '粤B32Y07': '鲁R32Y07',
 '皖AP9011': '皖AP901T',
 '皖AU1178': '皖AH1178',
 '豫AYH944': '粤XYH944',
 '鄂A607MG': '黑A607MG',
 '皖AG7C17': '皖AG7C07',
 '皖AC14': '皖ADX714',
 '皖AG51F': '皖AG511F',
 '京N71031': '沪N71031',
 '粤BB1188': '鄂BB1188',
 '皖A2F5Q6': '皖A2F506',
 '皖AC444A': '皖AC444J',
 '皖AL13DD': '皖AL130D',
 '皖AH773S': '皖AH776S',
 '皖AC38DM': '皖AC380M',
 '沪C262KA': '苏C262KA',
 '浙GLQ029': '津GLQ029',
 '皖A390B3': '皖A390B0',
 '浙E659B8': '赣E659B8',
 '皖A59H56': '皖A59U56',
 '皖AVD433': '皖AZD433',
 '浙FMK521': '鄂FMK521',
 '皖AQ3327': '皖A03327',
 '皖AEDN66': '皖AE0N66',
 '皖A806G1': '皖A806P1',
 '皖AL5714': '皖AL571A',
 '浙CA1115': '沪CA1115',
 '皖AN8666': '皖AN866B',
 '皖C0H288': '皖CQH288',
 '京Q6099K': '鲁Q6099K',
 '苏E16G50': '苏E16GL0',
 '苏A68BX3': '苏A68RX3',
 '皖AQ9998': '皖KQ9998',
 '皖AY8H16': '皖AY8L16',
 '皖A03P16': '皖AD3P16',
 '皖A050D0': '皖A05000',
 '皖AQE698': '皖NQE698',
 '皖AF398J': '皖AF398U',
 '苏A130FP': '浙A130FP',
 '皖AZ21DZ': '皖AZ210Z',
 '京NRB208': '鲁NRB208',
 '皖A04830': '皖AD4830',
 '皖AA5Q62': '鄂AA5Q62',
 '赣EXX665': '陕EXX665',
 '粤BQA986': '鲁BQA986',
 '皖NBB293': '粤NBB293',
 '皖S5222P': '皖RL222P',
 '豫SC6Q25': '粤SC6Q25',
 '皖A35P91': '粤L35P91',
 '浙FLK476': '贵FLK476',
 '浙AGJ792': '皖AGJ792',
 '皖AVQ566': '皖AVQ568',
 '粤BUS137': '浙BUS137',
 '皖AV8H880': '皖AV8H80',
 '皖AG5A91': '皖AG5A21',
 '皖AY2928': '皖AV2928',
 '渝DK8733': '沪DK8733',
 '粤BL1507': '琼BL1507',
 '皖AA75D6': '皖AA7S06',
 '浙BCF297': '苏BCF297',
 '皖MS8230': '黑MS8230',
 '苏C188NH': '苏C189KH',
 '皖AYY66': '皖EYA266',
 '皖AZU733': '皖AZU737',
 '苏E90RC0': '苏E90R0C',
 '皖AY522G': '皖AY522E',
 '皖A079VM': '陕A079VM',
 '浙EXQ3D3': '浙EXQ303',
 '苏E02G58': '苏E02GR8',
 '皖AX5290': '皖AX529C',
 '皖AMS53S': '皖AM553S',
 '皖AX195B': '皖AX195P',
 '皖AN9123': '皖AN912J',
 '皖NGG234': '皖NDG234',
 '皖AG03S6': '皖AG0396',
 '皖AJ8528': '皖AJ8K28',
 '皖A3408': '皖AKL408',
 '皖ABT920': '皖ABT820',
 '皖A252K5': '皖A252K8',
 '皖A6S33D': '皖A6S330',
 '赣FKK507': '鲁FKK507',
 '粤RL0550': '豫RL0550',
 '粤A52YH4': '鄂A52VH4',
 '皖AG1030': '皖AB1930',
 '粤B58KX6': '浙B58KX6',
 '皖A522RQ': '皖A522R0',
 '皖AN1863': '皖ANY833',
 '浙CL7029': '鄂CL7029',
 '皖AH3D9S': '皖AH309S',
 '浙AC25T6': '苏AC25T6',
 '浙A0X007': '蒙A0X007',
 '浙A70P03': '鄂A70P03',
 '皖BL222P': '皖RL222P',
 '赣GJ0579': '浙GJ0579',
 '皖A3151H': '云A3151H',
 '粤A5X8': '皖A771X8',
 '苏LLD155': '鄂LLD155',
 '粤A00T98': '皖AD0T89',
 '皖QQ9066': '皖Q99066',
 '皖ADS329': '皖AUS329',
 '皖AG011M': '皖AD011M',
 '浙AF0958': '云AFQ958',
 '皖A9Z5D5': '皖A9Z505',
 '京N6H339': '鄂N6H339',
 '苏E777GD': '苏E77G7D',
 '浙CP7826': '沪CP7826',
 '皖AZP121': '晋DZ8121',
 '粤BBS767': '豫RBS767',
 '皖AL7862': '皖AL786Z',
 '苏E777G7D': '苏E77G7D',
 '皖AK6X91': '皖AX6X91',
 '浙DZ8121': '晋DZ8121',
 '沪L3N780': '鄂L3N780',
 '赣F7H069': '湘F7H069',
 '赣M35612': '沪M35612',
 '皖AB1B09': '皖AP1P09',
 '皖AK009N': '皖AK098N',
 '皖AW58S80': '皖AW5S80',
 '皖BM8089': '皖RM8089',
 '皖AY5007': '皖AY5L07',
 '粤BU967M': '浙BU967M',
 '苏NK7956': '沪NK7956',
 '沪C7918F': '浙C7918F',
 '鄂A368SP': '浙A368SP',
 '浙CZQ117': '沪CZQ117',
 '浙C037L9': '鲁Y037L9',
 '皖AM3H93': '皖AM3N93',
 '皖AK801V': '皖AK801W',
 '赣G2L260': '豫G2L260',
 '皖A650J9': '皖A650T9',
 '皖K0T949': '皖A0T949',
 '粤B8HG01': '浙B8HG01',
 '皖AU3Y97': '冀AU3Y97',
 '皖AN6255': '新AN6255',
 '赣HPS178': '皖HPS178',
 '浙BSX779': '沪BSX779',
 '浙CNF878': '沪CNF878',
 '皖A6D829': '皖A6U829',
 '皖AB7851': '皖AP7851',
 '鄂AST758': '浙AST758',
 '苏LGD815': '苏LG0815',
 '沪HFP512': '浙HFP512',
 '皖A80064': '皖A0Y064',
 '皖A8Q1M5': '皖A801M5',
 '苏ABA268': '沪ABA268',
 '苏AHT260': '皖AHT260',
 '浙A7304H': '湘A7304H',
 '苏E03MM6': '苏E03MR6',
 '皖A97Y7': '皖A971Y7',
 '粤A883T8': '蒙A883T8',
 '京Q858T8': '鲁Q85818',
 '皖A2W35': '皖A2W003',
 '苏AA855F': '浙AA855F',
 '苏AVX317': '沪AVX317',
 '粤BA6X90': '浙BA6X90',
 '粤B8WK61': '浙B8WK61',
 '浙EHJ267': '粤EHJ267',
 '皖AY0288': '皖AYQ288',
 '浙G180H0': '赣G180H0',
 '皖AV2417': '皖AV2414',
 '皖SCC564': '皖ACC564',
 '皖AF0W08': '皖AF0V08',
 '浙A025U9': '鄂A02SU9',
 '浙Y0UM77': '沪C0UM77',
 '皖A1230': '皖A123X5',
 '皖A835B8': '浙A835E8',
 '皖AY028D': '皖AY0280',
 '皖AT01C6': '鄂AT01C6',
 '皖AN805H': '皖AN804N',
 '粤AA721D': '浙AA721D',
 '皖AEPS380': '皖AES380',
 '鄂A7829P': '豫A7829P',
 '皖A6Z1D8': '皖A6Z108',
 '苏A5K75S': '豫A5K75S',
 '皖AMS995': '新AMS995',
 '皖AMP665': '皖AMP600',
 '皖A229X2': '皖A229K2',
 '皖A591BG': '皖AJ915C',
 '皖AL222W': '皖AL222V',
 '皖AH5V2Q': '皖AH5V20',
 '皖AD289X': '皖AD289W',
 '皖AC3J886': '皖AC3J88',
 '皖AVEQ84': '皖AVL084',
 '皖AN9D55': '皖AN9D99',
 '冀FA8A15': '苏FA8A15',
 '皖AT2W80': '鄂AT2W80',
 '皖A188W0': '皖A188V0',
 '皖AWQ94F': '皖AW094F',
 '苏E67N57': '苏E67NX7',
 '皖A8S623': '皖ABW623',
 '浙AM1C88': '鄂AM1C88',
 '皖AH81Q2': '辽AH81Q2',
 '皖AWB8D4': '皖AWB804',
 '鄂ASB777': '皖ASB777',
 '京PN8230': '辽PN8230',
 '粤PA149Y': '粤YA149Y',
 '皖A981SU': '皖A981S0',
 '浙BB906Y': '苏BB906Y',
 '浙DZ9968': '晋DZ9968',
 '皖AK359': '皖AQN359',
 '皖FL778D': '皖FL7780',
 '皖EQ7765': '皖E97765',
 '京QA46D1': '粤QA4601',
 '皖AEZ152': '皖AFZ152',
 '皖A88Z68': '皖AS8Z68',
 '浙DEV572': '鄂DEV572',
 '鄂A7172J': '贵A7172J',
 '皖AHX112': '皖ANY612',
 '皖A001J8': '皖A001T8',
 '皖ARS85': '皖AK927W',
 '皖B006S0': '皖R00690',
 '沪C1MA36': '浙C1MA36',
 '皖A9336': '皖A93376',
 '皖AT918G': '皖AT918P',
 '皖AY5Y39': '皖AT5Y39',
 '皖P30059': '皖AMQ059',
 '皖A55CCC6': '皖A55CC6',
 '皖ADW787': '皖ADV787',
 '皖AL666M': '皖A1666M',
 '皖AZ15D2': '皖AZ1502',
 '苏E00HL5': '苏E00HC5',
 '皖AK369M': '皖AK369H',
 '京NT1199': '沪NT1199',
 '浙F9K908': '赣F9K908',
 '皖A1C8U5': '皖AH2W15',
 '鄂A1W5A1': '苏A1W5A1',
 '皖ABHA63': '皖A8H467',
 '浙ANZ246': '鄂ANZ246',
 '皖AJL299': '皖A1L299',
 '浙CJ0001': '闽CJ0001',
 '皖B00963': '皖R00963',
 '浙CVT910': '豫CVT910',
 '皖AX7M85': '皖AX7M75',
 '皖AN7211': '皖AN721L',
 '浙DK3785': '沪DK3785',
 '皖AH7632': '皖AH767W',
 '浙ALC568': '青ALC568',
 '鄂A8XQ86': '浙A8XQ86',
 '皖A53723': '皖APH723',
 '皖A43243': '皖A43247',
 '皖AEU969': '粤HEU969',
 '皖AT6506': '皖AT6M06',
 '豫ME4815': '赣ME4815',
 '皖AP8': '皖APY862',
 '苏FK332T': '浙FK332T',
 '皖A7055': '皖A76F11',
 '鄂A031JU': '豫A031JU',
 '皖GF8206': '皖GE8206',
 '浙AH448P': '粤AH448P',
 '皖AAN88S': '皖AAN885',
 '粤BK6857': '湘BK6857',
 '皖A3K4D7': '皖A3K407',
 '沪LV0652': '津LV0652',
 '冀F77X15': '渝F77X15',
 '皖AM5J86': '皖AH5J86',
 '皖AJ0T64': '皖AJ0T61',
 '浙B945J5': '粤B945J5',
 '皖AN0W20': '皖AM0W20',
 '皖BG5737': '皖MG5737',
 '皖A8A7T0': '皖A8A710',
 '皖AXD166': '皖AXD167',
 '皖A9U190': '皖A9L190',
 '冀F368K1': '渝F368K1',
 '皖AR6230': '皖AR623U',
 '皖AD8G28': '皖AD8C28',
 '浙CJP533': '沪CJP533',
 '皖AY8011': '皖AYR011',
 '皖AVD683': '皖AV0683',
 '苏JL173T': '浙JL173T',
 '皖A75M70': '皖A75H70',
 '皖A312V1': '鲁M712V1',
 '鄂A0791X': '陕A0791X',
 '苏K75F95': '蒙K75F95',
 '苏AF1595': '苏AF159L',
 '皖J2L882': '皖A2L882',
 '皖AY2ZZ99': '皖AY2Z99',
 '皖A23463': '皖A23464',
 '苏E23JG7': '苏E23JP7',
 '皖AX8146': '宁AX8146',
 '京NFG886': '皖N1G886',
 '皖ACG05': '皖A787G5',
 '皖A6B928': '皖A6P928',
 '皖ABS597': '皖APS597',
 '苏E51GG7': '苏E51QG7',
 '粤A86QJ8': '豫A86QJ8',
 '皖AT560P': '皖AT560F',
 '皖A59V9': '皖A619V9',
 '皖A72JJ5': '皖A72J15',
 '粤BNX611': '冀RNX611',
 '浙CEF489': '沪CEF489',
 '皖BB6556': '皖RB6556',
 '皖AT155V': '豫AT155V',
 '皖A531': '皖AEA031',
 '渝DHZ909': '粤DHZ909',
 '皖AS85DU': '皖AS850U',
 '皖AM021T': '皖AH021T',
 '皖ASW484': '皖ASW824',
 '皖AGM926': '皖AGH926',
 '皖ALZ735': '皖ATV735',
 '皖AX788N': '皖AX788W',
 '皖AR309R': '鲁AR309R',
 '鄂A0A9S5': '苏A0A9S5',
 '皖A06D8': '皖A06DD8',
 '皖A892BU': '陕A892BU',
 '皖AET507': '皖AE1307',
 '皖AU70P5': '陕AU70P5',
 '皖A36': '皖ABJ356',
 '皖RR0356': '皖KR0356',
 '苏B271UW': '苏B271UK',
 '粤BTW976': '吉BTW976',
 '皖AHX094': '皖AHXQ94',
 '皖AA6V68': '皖AA6W68',
 '皖AG9F79': '皖AD9F79',
 '皖AA475S': '皖AA4755',
 '浙AZP383': '沪AZP383',
 '浙CXL543': '粤CXL543',
 '皖A65AA6': '皖A654A6',
 '粤BL509J': '浙BL509J',
 '苏AJH176': '沪AJH176',
 '浙A1F739': '浙JZE739',
 '浙J6775A': '苏J6775A',
 '皖AWA6Q5': '皖AWA605',
 '皖AE1S5': '皖AE1S95',
 '皖AJH63Q': '皖AJH630',
 '皖A85R83': '赣A85R83',
 '皖A22995': '皖A22T99',
 '苏AN097W': '浙AN097W',
 '皖A5ZL7': '皖A606Z7',
 '皖AYL867': '皖AY1867',
 '鄂A338T0': '沪A338T0',
 '皖A67Y6': '皖A671Y6',
 '鄂A0K088': '津A0K088',
 '苏A0E050': '苏A0E0Y0',
 '皖D48635': '皖D48637',
 '浙A6KC23': '湘A6KC23',
 '粤BM7D20': '浙BM7D20',
 '浙AK556M': '黑AK556M',
 '皖AQE5D7': '皖AQE507',
 '皖AT770Z': '川AT770Z',
 '皖ACS693': '皖ALS693',
 '皖A061': '辽D26261',
 '皖A80A35': '闽A80A35',
 '沪CB80P8': '浙CP80P8',
 '皖AV0C07': '皖AV0C02',
 '浙AD70H9': '豫AD70H9',
 '皖A7930T': '皖AZ930T',
 '京PRE029': '鲁VRE029',
 '皖AAD872': '皖AA0872',
 '皖ADK11U': '皖ADK111',
 '粤BYS812': '沪BYS812',
 '皖AZ3P05': '苏AZ3P05',
 '皖A2Q9D0': '皖A2Q900',
 '皖AR8H886': '皖AR8H86',
 '皖AW3118': '皖AWJ118',
 '苏JD0076B': '苏JD876B',
 '皖NY687': '皖HY6887',
 '苏H5S820': '浙H5S820',
 '皖ALDL10': '皖AL0L10',
 '皖AY1U32': '皖AYU642',
 '皖B08953': '皖R08953',
 '皖ASY26D': '皖ASY260',
 '皖A831SK': '冀A831SK',
 '浙BKA662': '沪BKA662',
 '皖A17SS9': '皖A179S9',
 '皖A51M12': '皖A51M17',
 '粤BP9J37': '浙BP9J37',
 '皖AL3BD0': '皖AL3B00',
 '苏BE33G6': '苏EE33G6',
 '皖AF098W': '皖AF099W',
 '皖AF8660': '皖AF866C',
 '皖A4Y55S': '皖A4Y559',
 '皖AVR768': '皖AWR768',
 '皖QS601X': '皖SS601X',
 '皖A7Q567': '皖A1Q567',
 '苏QV366C': '苏DV366C',
 '皖AAQ320': '皖AAQ720',
 '粤B8Q82K': '鲁B8Q82K',
 '鄂A6E11K': '川A6E11K',
 '皖A8813': '皖AM8812',
 '京Q19BK0': '京Q19BN0',
 '皖A97Z79': '皖AP7Z79',
 '苏LC822D': '皖AC822D',
 '京N0N779': '皖N0N779',
 '皖A98Z98': '皖A98Z89',
 '皖A8378A': '皖A83784',
 '皖A1F251': '皖A1E251',
 '鄂A9E03A': '川A9E03A',
 '皖A27DD3': '皖A27CD3',
 '粤BK18Q3': '鲁BK18Q3',
 '粤A8HK21': '浙A8HK21',
 '皖A893': '皖ACH993',
 '粤R8865A': '豫R8865A',
 '苏J92B33': '苏J92E33',
 '沪C0P0T2': '沪C9P0T2',
 '皖A26X58': '皖A26Y58',
 '赣E45042': '鄂E45042',
 '皖AK262K': '皖AA262K',
 '豫MU2785': '川MU2785',
 '浙AH7279': '苏JH7279',
 '皖MH9E18': '皖HH9E18',
 '沪LBT526': '蒙LBT526',
 '皖AJF666': '皖AJL666',
 '皖ABS007': '皖ABS001',
 '粤L67A77': '辽L87A77',
 '粤B0311C': '闽B0311C',
 '皖A8R561': '皖A5R551',
 '皖BM9628': '皖RM9628',
 '皖KJQ929': '蒙KJQ929',
 '皖MRY333': '津MRY333',
 '皖AMX258': '鄂AMX258',
 '皖AY5596': '皖AY5598',
 '浙A8ZH33': '苏A8ZH33',
 '沪L83C23': '粤L8T283',
 '粤B90MG9': '浙B90KG9',
 '粤B655RR': '浙B655RR',
 '京PCX385': '豫PCX385',
 '皖AG8592': '京AG8592',
 '皖AHD77G': '皖AH077G',
 '粤RE3635': '苏BE767W',
 '沪C3KM55': '浙C3KM55',
 '鄂ASF521': '浙ASF521',
 '沪C7VV26': '浙C7VV26',
 '皖AF619': '皖ACF619',
 '粤S8Y585': '豫S8Y585',
 '浙EH4B78': '鄂EH4P78',
 '浙AM062J': '晋AM062J',
 '苏A3B5G5': '苏A3B8G5',
 '皖NA2S56': '皖NA2556',
 '皖AG838U': '皖AG838D',
 '皖A83927': '皖A83922',
 '赣GVY921': '豫GVY921',
 '皖AX442X': '皖AX447X',
 '皖A26': '皖ADD226',
 '沪C9HH82': '浙C9HH82',
 '鄂A0450D6': '苏D750D6',
 '皖AX83DR': '皖AX830R',
 '苏DB809J': '苏DR809J',
 '皖AAG490': '皖AAR490',
 '京N89N13': '豫N89N13',
 '苏E48F88': '赣E48F88',
 '皖A952D2': '皖A952U2',
 '苏DB7208': '苏DR7208',
 '浙DQ7222': '渝DQ7222',
 '皖CJ0001': '闽CJ0001',
 '浙G06B57': '浙C06BS7'}
```

As we can see, the model is most wrong on Chinese characters, I tried to fix it with augmentations, but still there are numbers on which the model is wrong

```
key_err_0 = list(errors.values())[0]
```

```
ind_err_0 = dataset_test.text.index(key_err_0)
```

```
torchvision.transforms.ToPILImage()(dataset_test[ind_err_0][0])
```



```
list(errors.keys())[0]
```

```
'苏AYC335'
```

```
key_err_0
```

```
'皖AYC335'
```

```
acc_avg = val_loop(dataloader_test, model, tokenizer, device)
```

```
Validation, acc: 0.9533
```

```
acc_avg
```

```
0.9532953295329533
```