```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 607.4 kB/s eta
0:00:00a 0:00:01
ent already satisfied: tqdm>=4.62.1 in /opt/conda/lib/python3.7/site-
packages (from evaluate) (4.64.0)
Requirement already satisfied: xxhash in
/opt/conda/lib/python3.7/site-packages (from evaluate) (3.0.0)
Requirement already satisfied: fsspec[http]>=2021.05.0 in
/opt/conda/lib/python3.7/site-packages (from evaluate) (2022.8.2)
Requirement already satisfied: pandas in
/opt/conda/lib/python3.7/site-packages (from evaluate) (1.3.5)
Requirement already satisfied: multiprocess in
/opt/conda/lib/python3.7/site-packages (from evaluate) (0.70.13)
Requirement already satisfied: importlib-metadata in
/opt/conda/lib/python3.7/site-packages (from evaluate) (4.13.0)
Requirement already satisfied: packaging in
/opt/conda/lib/python3.7/site-packages (from evaluate) (21.3)
Requirement already satisfied: datasets>=2.0.0 in
/opt/conda/lib/python3.7/site-packages (from evaluate) (2.1.0)
Requirement already satisfied: dill in /opt/conda/lib/python3.7/site-
packages (from evaluate) (0.3.5.1)
Requirement already satisfied: responses<0.19 in
/opt/conda/lib/python3.7/site-packages (from evaluate) (0.18.0)
Requirement already satisfied: huggingface-hub>=0.7.0 in
/opt/conda/lib/python3.7/site-packages (from evaluate) (0.10.1)
Requirement already satisfied: numpy>=1.17 in
/opt/conda/lib/python3.7/site-packages (from evaluate) (1.21.6)
Requirement already satisfied: requests>=2.19.0 in
/opt/conda/lib/python3.7/site-packages (from evaluate) (2.28.1)
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 4.4 MB/s eta
0:00:0000:0100:01
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: aiohttp in
```

```
/opt/conda/lib/python3.7/site-packages (from datasets>=2.0.0-
>evaluate) (3.8.1)
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: 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: idna<4,>=2.5 in
/opt/conda/lib/python3.7/site-packages (from requests>=2.19.0-
>evaluate) (3.3)
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: certifi>=2017.4.17 in
/opt/conda/lib/python3.7/site-packages (from requests>=2.19.0-
>evaluate) (2022.9.24)
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: 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: 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: 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: 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: 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: 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: 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: 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
device
```

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
# some augmentetions for model regularization
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 = 64
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)) # take a look to random image

img
```

```
dataset_test[122] # take a look to element from dataset
```

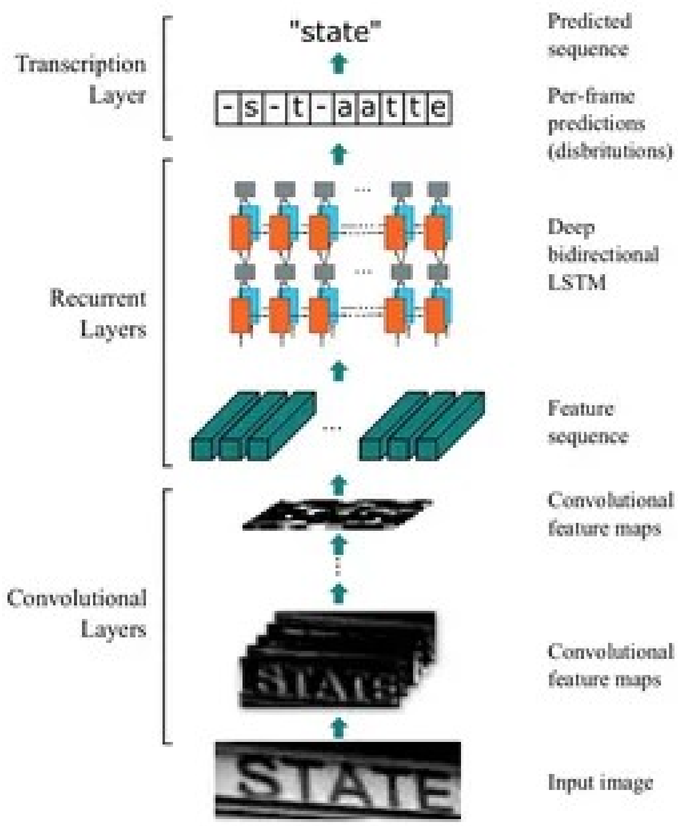
```
(tensor([[[0.5451, 0.5529, 0.5608,  ..., 0.4157, 0.4118, 0.4706],
          [0.4431, 0.4039, 0.4039,  ..., 0.4157, 0.5020, 0.5647],
          [0.3333, 0.2980, 0.3059,  ..., 0.2353, 0.3137, 0.4353],
          ...,
          [0.3333, 0.3137, 0.2667,  ..., 0.1608, 0.1490, 0.1843],
          [0.3216, 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')
```

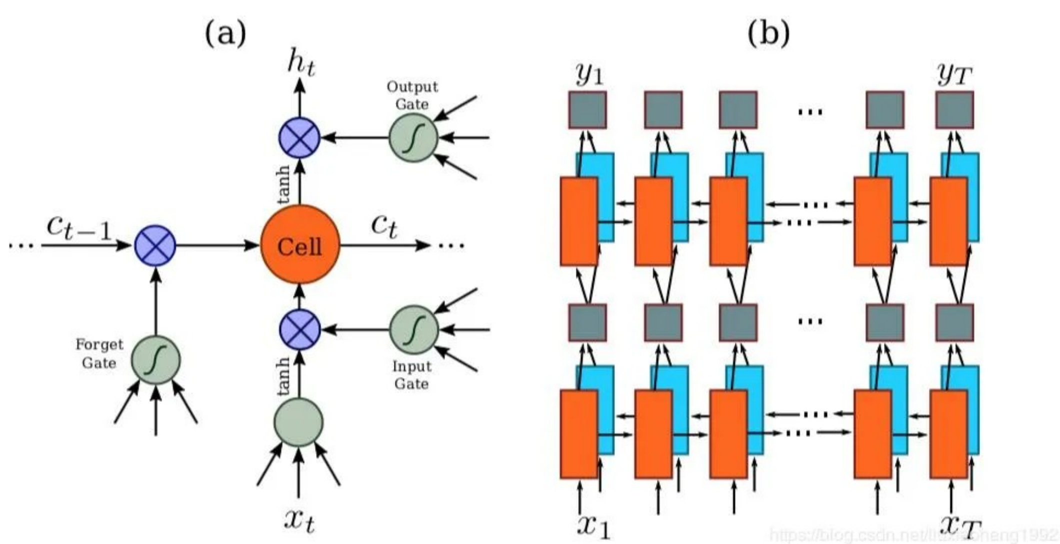
Define model

```
from torch import nn
```

To solve the problem i use CRNN structure

Approximate model structure and LSTM structure






```python
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

"""
5 ResNet blocks, 3 BiLSTM, Linear Classifier
in order for the model to be able to identify more complex
dependencies in the data, since the dataset allows you to enter
a deep neural network
"""
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) # the
model predicts the probability of characters from the alphabet
        )

    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

```
# functions for saving models
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 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]) #
for CTC-loss
        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

accs = {}
def train(dataloader, epochs, trained_model=None,
trained_model_epochs=0):
    train_loader, val_loader = dataloader['train'],
dataloader['valid']
    if trained_model == None:
        model = CRNN(number_class_symbols=tokenizer.get_num_chars())
        model.apply(weights_init)
```

```
        model.to(device)
    else:
        model = trained_model

    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)
        accs[acc_avg] = epoch
        scheduler.step(acc_avg)
        if acc_avg > best_acc:
            best_acc = acc_avg
        if trained_model_epochs == 0:
            safe(model, f'model_{epoch}')
        else:
            safe(model, f'model_{epoch + trained_model_epochs}')

train(dataloader, 8)
```

```
Validation, acc: 0.0000

Epoch 0, Loss: 1.68883, LR: 0.0010000
Validation, acc: 0.3340

Epoch 1, Loss: 0.23575, LR: 0.0010000
Validation, acc: 0.6538

Epoch 2, Loss: 0.10533, LR: 0.0010000
Validation, acc: 0.9317

Epoch 3, Loss: 0.05218, LR: 0.0010000
Validation, acc: 0.9395

Epoch 4, Loss: 0.04142, LR: 0.0010000
Validation, acc: 0.9573

Epoch 5, Loss: 0.03328, LR: 0.0010000
Validation, acc: 0.9699

Epoch 6, Loss: 0.02488, LR: 0.0010000
Validation, acc: 0.9650
```

```
Epoch 7, Loss: 0.02500, LR: 0.0010000
Validation, acc: 0.9668
```

```
model = read(f'/kaggle/working/model_{accs[max(accs.keys())]}') # load
model with best acc on validation
```

```
train(dataloader, 10, model, 8) # more epochs
```

```
Validation, acc: 0.9684

Epoch 0, Loss: 0.02766, LR: 0.0010000
Validation, acc: 0.9715

Epoch 1, Loss: 0.02236, LR: 0.0010000
Validation, acc: 0.9669

Epoch 2, Loss: 0.01953, LR: 0.0010000
Validation, acc: 0.9771

Epoch 3, Loss: 0.01721, LR: 0.0010000
Validation, acc: 0.9784

Epoch 4, Loss: 0.01611, LR: 0.0010000
Validation, acc: 0.9821

Epoch 5, Loss: 0.01570, LR: 0.0010000
Validation, acc: 0.9808

Epoch 6, Loss: 0.01318, LR: 0.0010000
Validation, acc: 0.9733

Epoch 7, Loss: 0.01456, LR: 0.0010000
Validation, acc: 0.9831

Epoch 8, Loss: 0.01173, LR: 0.0010000
Validation, acc: 0.9848

Epoch 9, Loss: 0.01146, LR: 0.0010000
Validation, acc: 0.9827
```

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

```
model = read(f'/kaggle/working/model_{16}') # take model with the best
acc on validation
```

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

```
['皖KLJ029']
```

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

Compute metrics

CER = (S + D + I) / N = (S + D + I) / (S + D + C)

where

S is the number of substitutions, D is the number of deletions, I is the number of insertions, C is the number of correct characters, N is the number of characters in the reference (N=S+D+C).

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

```python
references = dataset_test.text
```

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

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

```
0.0058005800580058
```

CER's output is not always a number between 0 and 1, in particular when there is a high number of insertions. This value is often associated to the percentage of characters that were incorrectly predicted. The lower the value, the better the performance of the ASR system with a CER of 0 being a perfect score.

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

```
True
```

```python
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
```

```python
errors
```

```
{'皖HL108D': '皖NL108D',
 '皖AE0082': '皖AF888S',
 '皖AN5R29': '冀AN5R29',
 '皖AE8F76': '皖AF8F76',
 '冀JS2363': '川JS2363',
 '皖A06Z16': '皖AD6Z16',
```

```
'皖AL1D76': '皖AL1D70',
'豫ADL439': '甘ADL439',
'皖AZZ2D8': '皖AZZ208',
'鄂E6V671': '赣E6V671',
'鲁CKB654': '粤CKB654',
'赣E269JY': '湘E269JY',
'鄂AN08Q3': '豫AN08Q3',
'皖SC716': '皖SCC716',
'皖AB9017': '皖AP901T',
'皖AH1179': '皖AH1178',
'皖AHA18C': '皖AHA180',
'鄂A607MG': '黑A607MG',
'皖A9C714': '皖ADX714',
'苏N71031': '沪N71031',
'闽BB1188': '鄂BB1188',
'皖A75156': '皖A751S6',
'皖A280D5': '皖A28G05',
'皖AL13DD': '皖AL130D',
'皖AZW563': '皖AZW063',
'皖AC38DM': '皖AC380M',
'浙GLQ029': '津GLQ029',
'皖E659B8': '赣E659B8',
'冀FMK521': '鄂FMK521',
'皖AG3327': '皖A03327',
'皖AEDN66': '皖AE0N66',
'皖APX498': '皖ADX498',
'浙CN060V': '闽CN060V',
'皖AN866G': '皖AN866B',
'京Q6099K': '鲁Q6099K',
'苏E160C0': '苏E16GL0',
'苏A68R93': '苏A68RX3',
'皖A050C0': '皖A05000',
'皖MQE698': '皖NQE698',
'京NRB208': '鲁NRB208',
'皖EXX665': '陕EXX665',
'晋BQA986': '鲁BQA986',
'皖NBB293': '粤NBB293',
'皖DR22P': '皖RL222P',
'豫SC6Q25': '粤SC6Q25',
'皖L35P91': '粤L35P91',
'鄂FLK476': '贵FLK476',
'皖AJ2928': '皖AV2928',
'浙DK8733': '沪DK8733',
'粤BL1507': '琼BL1507',
'皖AA79D6': '皖AA7S06',
'皖AP1P17': '皖AP1P13',
'皖A812A5': '皖A81245',
'粤MS8230': '黑MS8230',
'皖HYA266': '皖EYA266',
```

```
'浙EXQ3C3': '浙EXQ303',
'京P76N06': '京P76M06',
'皖AJ6408': '皖AKL408',
'皖ABT220': '皖ABT820',
'皖AZ33DT': '皖AZ330T',
'皖A2M2K8': '皖A252K8',
'皖A6S33D': '皖A6S330',
'冀FKK507': '鲁FKK507',
'鄂A52NH4': '鄂A52VH4',
'皖AB1Q30': '皖AB1930',
'浙B58K96': '浙B58KX6',
'皖A522B0': '皖A522R0',
'皖ANY863': '皖ANY833',
'闽CL7029': '鄂CL7029',
'皖AH3099': '皖AH309S',
'皖ADK195': '皖A0K195',
'沪A0X007': '蒙A0X007',
'皖A3151H': '云A3151H',
'鄂A71XY8': '皖A771X8',
'皖A235H0': '皖A235U0',
'豫LLD156': '鄂LLD155',
'皖AD0926': '皖A00926',
'皖BZ996J': '皖RZ996J',
'皖AD0T99': '皖AD0T89',
'皖G99C66': '皖Q99066',
'皖AB011M': '皖AD011M',
'皖AFQ958': '云AFQ958',
'皖A9Z5D5': '皖A9Z505',
'渝DZ8121': '晋DZ8121',
'皖AD76DD': '皖AD760D',
'皖A18RQ0': '皖A18R00',
'粤SG6U10': '豫SG6U10',
'皖A6X91': '皖AX6X91',
'粤L3N780': '鄂L3N780',
'浙F7H069': '湘F7H069',
'赣M35612': '沪M35612',
'皖AD1P09': '皖AP1P09',
'皖A75W26': '皖A75W76',
'皖AF0A61': '皖AP0A61',
'皖AK088N': '皖AK098N',
'皖FXH016': '皖EXH016',
'皖A368SP': '浙A368SP',
'粤X937L9': '鲁Y037L9',
'皖AK801V': '皖AK801W',
'赣G2L260': '豫G2L260',
'皖A65019': '皖A650T9',
'皖AUT949': '皖A0T949',
'皖AY6A32': '皖A96A39',
'皖AU3Y97': '冀AU3Y97',
```

```
'皖AN6255': '新AN6255',
'皖HPS138': '皖HPS178',
'皖AG961C': '皖AG961L',
'浙BSX779': '沪BSX779',
'鲁HFP512': '浙HFP512',
'皖A8Y564': '皖A0Y064',
'皖AWU097': '皖AWE097',
'浙A7304H': '湘A7304H',
'豫A883T8': '蒙A883T8',
'皖AHBV95': '皖AH8V95',
'皖A25605': '皖A2W003',
'皖A829X9': '皖A829Y9',
'浙EHJ267': '粤EHJ267',
'皖AF0K08': '皖AF0V08',
'鄂A02SC9': '鄂A02SU9',
'皖A12376': '皖A123X5',
'皖ASF6C4': '皖ASF604',
'晋C1G999': '浙C1G999',
'皖A918B1': '皖A918Q1',
'皖ACS380': '皖AES380',
'鄂A7829P': '豫A7829P',
'湘A5K75S': '豫A5K75S',
'皖AMS995': '新AMS995',
'皖AMP669': '皖AMP600',
'皖A6915C': '皖AJ915C',
'皖C23D77': '皖C23077',
'皖AD289X': '皖AD289W',
'皖AC3J66': '皖AC3J88',
'皖AVE081': '皖AVL084',
'皖AR316B': '皖AR316D',
'京HA8A15': '苏FA8A15',
'鄂GK3511': '赣GK3511',
'皖AWQ94F': '皖AW094F',
'苏E67N97': '苏E67NX7',
'苏B20QQB': '苏B200QB',
'皖AH81Q2': '辽AH81Q2',
'皖AHE220': '皖AHF220',
'皖AWB8D4': '皖AWB804',
'皖AGS39P': '皖AGS308',
'京PN8230': '辽PN8230',
'粤HA149Y': '粤YA149Y',
'皖A981SU': '皖A981S0',
'浙BB906Y': '苏BB906Y',
'川A9XT19': '鄂A9XT19',
'皖AFB872': '皖AFR872',
'渝DZ9968': '晋DZ9968',
'皖A2N389': '皖AQN359',
'皖E27765': '皖E97765',
'鲁QA46D1': '粤QA4601',
```

```
'皖AEZ152': '皖AFZ152',
'皖A88Z68': '皖AS8Z68',
'闽DEV572': '鄂DEV572',
'皖MR407': '皖AMR407',
'鄂A7172J': '贵A7172J',
'皖AHY612': '皖ANY612',
'皖AR8229': '皖AK927W',
'皖N2Q059': '皖AMQ059',
'皖AL666M': '皖A1666M',
'皖AZ15D2': '皖AZ1502',
'苏NT1199': '沪NT1199',
'浙F9K908': '赣F9K908',
'皖BZR331': '皖RZR331',
'皖AJ22J5': '皖AH2W15',
'皖ABH663': '皖A8H467',
'鲁CJ0001': '闽CJ0001',
'皖R0D963': '皖R00963',
'粤CVT910': '豫CVT910',
'晋BQ6M90': '浙BQ6M90',
'皖AD5R59': '皖A05R59',
'皖AX7M72': '皖AX7M75',
'苏E33P00': '苏E33VC0',
'京QXK818': '皖QXK818',
'皖AN7211': '皖AN721L',
'浙DK3785': '沪DK3785',
'皖AH7677': '皖AH767W',
'皖A166E4': '皖A166F4',
'浙ALC568': '青ALC568',
'皖AP1723': '皖APH723',
'鲁MEU969': '粤HEU969',
'豫ME4815': '赣ME4815',
'皖AFN286': '皖APY862',
'皖AZ6111': '皖A76F11',
'鄂A031JU': '豫A031JU',
'鄂AH448P': '粤AH448P',
'浙BK6857': '湘BK6857',
'皖A896P7': '皖A896P1',
'皖HR3241': '皖NR3241',
'浙LV0652': '津LV0652',
'浙F77X15': '渝F77X15',
'浙B886B8': '苏B886B8',
'皖AD5J86': '皖AH5J86',
'皖AJ0T64': '皖AJ0T61',
'皖AK0W20': '皖AM0W20',
'皖AXD162': '皖AXD167',
'皖ASL190': '皖A9L190',
'浙F368K1': '渝F368K1',
'皖AN9V96': '皖AN9V16',
'皖A97166': '皖A97160',
```

```
'皖A2R383': '皖A8R383',
'皖A75D70': '皖A75H70',
'皖H312V1': '鲁M712V1',
'川A0791X': '陕A0791X',
'苏K75F95': '蒙K75F95',
'皖A23466': '皖A23464',
'苏E23F07': '苏E23JP7',
'皖AWP446': '皖AWF446',
'皖AX8146': '宁AX8146',
'皖HFG886': '皖N1G886',
'皖AN7G5': '皖A787G5',
'皖AD6383': '皖A06383',
'苏E51QQ7': '苏E51QG7',
'粤A86QJ8': '豫A86QJ8',
'皖AT560P': '皖AT560F',
'皖AC6S66': '皖AC6S68',
'浙A619V8': '皖A619V9',
'皖ANZ536': '皖ANZ538',
'皖A72JT9': '皖A72J15',
'鲁RNX611': '冀RNX611',
'鄂AT155V': '豫AT155V',
'皖A83T7': '皖A83T07',
'川AN5R29': '冀AN5R29',
'皖A36031': '皖AEA031',
'闽DHZ909': '粤DHZ909',
'皖AB397G': '皖AB3978',
'皖ACF6C1': '皖ACF601',
'皖ASW324': '皖ASW824',
'皖ATY735': '皖ATV735',
'皖AX788V': '皖AX788W',
'浙AR309R': '鲁AR309R',
'皖A06D08': '皖A06DD8',
'粤A892BU': '陕A892BU',
'皖A42E80': '皖A42F80',
'皖H5H606': '皖N5H606',
'鄂AU70P5': '陕AU70P5',
'皖AP0E86': '皖AP0L86',
'皖AJ3684': '皖ABJ356',
'皖ADU527': '皖A0U527',
'苏B2711K': '苏B271UK',
'皖AEE325': '皖AEL325',
'皖AQN6C8': '皖AQN608',
'粤BTW936': '吉BTW976',
'皖AHX094': '皖AHXQ94',
'皖ABSF79': '皖AD9F79',
'鲁CXL543': '粤CXL543',
'浙AZE739': '浙JZE739',
'皖AWA6Q5': '皖AWA605',
'皖A231Q7': '皖A23107',
```

'皖AJH63Q': '皖AJH630',
'鄂A85R83': '赣A85R83',
'皖A22T96': '皖A22T99',
'皖A6G6Z7': '皖A606Z7',
'皖AYL867': '皖AY1867',
'皖ADA5D5': '皖ADA505',
'川A0K088': '津A0K088',
'皖AQD975': '皖AQD915',
'皖D4863J': '皖D48637',
'鄂A6KC23': '湘A6KC23',
'皖AK556M': '黑AK556M',
'浙GD2500': '苏GD2500',
'鄂AT770Z': '川AT770Z',
'赣M26261': '辽D26261',
'皖A80A35': '闽A80A35',
'皖AHF024': '皖AHFQ24',
'粤URE029': '鲁VRE029',
'皖AT06QT': '皖AT060T',
'皖AAU872': '皖AA0872',
'苏AD876B': '苏JD876B',
'皖AV1662': '皖AYU642',
'豫NQ0551': '京NQ0551',
'鄂A831SK': '冀A831SK',
'闽BB12S9': '粤BB12S9',
'粤B8Q82K': '鲁B8Q82K',
'鄂A6E11K': '川A6E11K',
'浙B3ZD36': '苏B3ZD36',
'皖ANG812': '皖AM8812',
'鲁Q19GN0': '京Q19BN0',
'皖A97Z79': '皖AP7Z79',
'苏AC822D': '皖AC822D',
'皖A1F251': '皖A1E251',
'鄂A9E03A': '川A9E03A',
'皖A27CB3': '皖A27CD3',
'皖AUE198': '皖AJE198',
'皖AY918C': '皖AY915C',
'粤BK18Q3': '鲁BK18Q3',
'皖A9993': '皖ACH993',
'新R8865A': '豫R8865A',
'皖B547C0': '皖B54700',
'皖AA621N': '皖AA631N',
'赣E45042': '鄂E45042',
'浙G6606N': '浙G6606K',
'皖AJ262K': '皖AA262K',
'豫MU2785': '川MU2785',
'皖AH7279': '苏JH7279',
'皖MH9E18': '皖HH9E18',
'赣LBT526': '蒙LBT526',
'皖ABS007': '皖ABS001',

```
'赣L87A77': '辽L87A77',
 '粤B0311C': '闽B0311C',
 '皖A5R561': '皖A5R551',
 '浙KJQ929': '蒙KJQ929',
 '京MRY333': '津MRY333',
 '皖AMX258': '鄂AMX258',
 '赣L87283': '粤L8T283',
 '粤B90KG9': '浙B90KG9',
 '皖AD5Q8C': '皖AD506C',
 '川AG8592': '京AG8592',
 '皖AHD77G': '皖AH077G',
 '苏AT58X8': '苏AF58X8',
 '鲁MGE679': '苏BE767W',
 '皖AF812Q': '皖AFS120',
 '皖A3F619': '皖ACF619',
 '皖A0Y0D0': '皖A0Y000',
 '皖AUE118': '皖AUF118',
 '粤S8Y585': '豫S8Y585',
 '川EH4B78': '鄂EH4P78',
 '皖A79565': '皖A7S565',
 '皖AM062J': '晋AM062J',
 '皖AG838U': '皖AG838D',
 '皖AC7783': '皖FC7783',
 '赣GVY921': '豫GVY921',
 '皖AJ226': '皖ADD226',
 '京N89N13': '豫N89N13',
 '皖AE6719': '皖AF6719',
 '浙DQ7222': '渝DQ7222'}
```

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. These errors most likely arise due to poor image quality, this can also be corrected by expanding the sample, or training the model specifically for recognizing Chinese characters and then merging it with the main model. Also, the model sometimes makes mistakes in the length of the number and in the middle characters. This can be fixed with the help of augmentations: for example, painting over some part of a certain symbol.

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

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

Errors can also be due to damage to numbers, which we see in this example

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


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

```
'皖HL108D'
```

```
key_err_0
```

```
'皖NL108D'
```

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

```
Validation, acc: 0.9667
```

```
acc_avg
```

```
0.9666966696669667
```

Totals by metrics:

Accuracy on test: 0.9666966696669667

CER on test: 0.0058005800580058