```
import torch
import cv2
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import torchvision
import os
from torch import nn
from torch.optim import lr_scheduler
from tqdm import tqdm
import torch.optim as optim
import torchvision.transforms as transforms
import PIL
from albumentations import Compose, HorizontalFlip, RandomContrast,
Crop, RandomBrightnessContrast, RandomCrop, Flip, RandomSizedCrop,
OneOf, PadIfNeeded, Normalize, Resize, RandomCrop
```

Определение вычислительного устройства

```
device = torch.device('cuda') if torch.cuda.is_available() else
torch.device('cpu')
```

Подключения диска с данными

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!pip install torchmetrics
```

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Collecting torchmetrics
  Downloading torchmetrics-0.11.0-py3-none-any.whl (512 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 512.4/512.4 KB 37.9 MB/s eta
0:00:00
ent already satisfied: typing-extensions in
/usr/local/lib/python3.8/dist-packages (from torchmetrics) (4.4.0)
Requirement already satisfied: torch>=1.8.1 in
/usr/local/lib/python3.8/dist-packages (from torchmetrics)
(1.13.0+cu116)
Requirement already satisfied: packaging in
/usr/local/lib/python3.8/dist-packages (from torchmetrics) (21.3)
Requirement already satisfied: numpy>=1.17.2 in
/usr/local/lib/python3.8/dist-packages (from torchmetrics) (1.21.6)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/usr/local/lib/python3.8/dist-packages (from packaging->torchmetrics)
(3.0.9)
Installing collected packages: torchmetrics
Successfully installed torchmetrics-0.11.0
```

```
!nvidia-smi
```

```
Sat Jan  7 19:03:40 2023
+-----------------------------------------------------------------------
--------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version:
11.2     |
|-------------------------------+----------------------
+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util
Compute M. |
|                               |                      |
MIG M. |
|
===============================+======================+===============
=======|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |
0 |
| N/A   53C    P0    29W /  70W |      3MiB / 15109MiB |      0%
Default |
|                               |                      |
N/A |
+-------------------------------+----------------------
+----------------------+


+-----------------------------------------------------------------------
--------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name                  GPU
Memory |
|        ID   ID
Usage      |
|
=======================================================================
=======|
|  No running processes found
|
+-----------------------------------------------------------------------
--------+
```

Распаковка данных

```
# !unzip /content/drive/MyDrive/nubers/CCPD2019-dl1.zip
```

Создание модели

```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.in_fea = 0
```

```
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 7, kernel_size=3, stride=1, padding=1,
bias=False),
            nn.BatchNorm2d(7),
            nn.ReLU(inplace=True))
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)
        self.conv2 = nn.Sequential(
            nn.Conv2d(7, 7, kernel_size=3, stride=1, padding=1,
bias=False),
            nn.BatchNorm2d(7),
            nn.ReLU(inplace=True))
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)
        self.fc1 = nn.Sequential(
            nn.Linear(512, 32),
            nn.ReLU(inplace=True))
        self.gru1 = nn.GRU(32, 512)
        self.gru1_b = nn.GRU(32, 512)
        self.gru2 = nn.GRU(512, 512)
        self.gru2_b = nn.GRU(512, 512)
        self.fc2 = nn.Sequential(
            nn.Linear(1024, 66),
            nn.ReLU(inplace=True))
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode="fan_out",
nonlinearity="relu")
            elif isinstance(m, (nn.BatchNorm2d, nn.GroupNorm)):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)

    def forward(self, x):
        x = self.conv1(x)
        x = self.maxpool1(x)
        x = self.conv2(x)
        x = self.maxpool2(x)
        self.in_fea = x.size(2) * x.size(3)
        x = torch.reshape(x, (x.size(0), 7, 512))
        x = self.fc1(x)
        x_1, h_1 = self.gru1(x)
        x_2, h_2 = self.gru1_b(x)
        x = x_1 + x_2
        x_1, h_1 = self.gru2(x)
        x_2, h_2 = self.gru2_b(x)
        x = torch.cat((x_1, x_2), 2)
        x = self.fc2(x)
        return x
```

Создание класса для формирования кастомного датасета, объявление функции для создания даталоадера

```
class MyDataset(torch.utils.data.Dataset):
    def __init__(self, work_dir:str, state:bool, transform=None):
        self.work_dir = work_dir
        # self.tta = tta
        self.state = state
        self.transform = transform
        self.policies = transforms.AutoAugmentPolicy.CIFAR10
        self.conv = 
transforms.Compose([transforms.AutoAugment(self.policies)])
        self.data = self.parsing_data()

    def parsing_data(self):
        list_data = os.listdir(self.work_dir)
        result_dataframe = list()
        for i, image in enumerate(list_data):
            image = image.split('-')[1].split('.')

result_dataframe.append([f'{self.work_dir}/{list_data[i]}', image[0]])
        return pd.DataFrame(result_dataframe, columns=[0, 1])

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

    def __getitem__(self, idx:int):
        if self.state:
            image = cv2.imread(self.data.iloc[idx, 0])
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        else:
            image = PIL.Image.open(self.data.iloc[idx, 
0]).convert('RGB')
        label = self.data.iloc[idx, 1]
        if self.transform:
            if not self.state:
                image = self.conv(image)
                image = np.array(image)
            transforming = self.transform(image=image)
            image = transforming["image"]
        image = torchvision.transforms.functional.to_tensor(image)
        return image, label

def CreateDataloader(data_type:str, transforms, shuffle:bool, 
state:bool):
    work_dir = f'./CCPD2019-dl1/{data_type}'
    dataset = MyDataset(work_dir, state, transforms)
    dataloader = torch.utils.data.DataLoader(dataset, batch_size=256, 
shuffle=shuffle, num_workers=2, pin_memory=True)
```

```
    data_size = len(dataset)
    return dataloader, data_size
```

Немного некрасивого кода для получения словараей со всеми символами из лэйблов и соответствующими им порядковыми номерами

```
work_dir = '/content/CCPD2019-dl1/train'
d = {}
inv_d = {}
list_data = os.listdir(work_dir)
for i, image in enumerate(list_data):
    image = image.split('-')[1].split('.')
    if len(image[0]) != 7:
        print(image[0])
    for sumbul in image[0]:
        if sumbul not in d:
            inv_d[len(d)] = sumbul
            d[sumbul] = len(d)
```

Функция конвертирования символьного лэйбла в числовой

```
def ConverLabelToNum(d:dict, lbl:torch.Tensor):
    res = list()
    for iteration in lbl:
        tmp = list()
        for sumbul in iteration:
            tmp.append(d[sumbul])
        res.append(tmp)
    return torch.tensor(res)
```

Объявление базовых аугментаций

```
tr = Compose([Resize(64, 128, p=1.0),
        Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224,
0.225), max_pixel_value=255.0, always_apply=False, p=1.0)])
```

Фиксирование рандомов для воспроизводимости результатов экспериментов

```
import random

def set_seed(seed:int=1):
    np.random.seed(seed)
    random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
set_seed()
```

Объявление функций для подсчета метрик при обучении и валидации

```
from torchmetrics import CharErrorRate


def accuracy(preds:torch.Tensor, label:torch.Tensor):
    result = 0
    for i in range(len(preds)):
        count = 0
        for j in range(len(preds[i])):
            if preds[i][j] == label[i][j]:
                count += 1
            if count == 7:
                result += 1
    return result

def CER(inv_d:dict, lbl:torch.Tensor, preds:torch.Tensor, metric):
    preds = preds.cpu().detach().numpy()
    result_preds, result_lbl = list(), list()
    for i in range(len(preds)):
        tmp_preds, tmp_lbl = '', ''
        for j in range(len(preds[i])):
            tmp_preds += inv_d[preds[i][j]]
            tmp_lbl += lbl[i][j]
        result_preds.append(tmp)
        result_lbl.append(t)
    return metric(result_preds, result_lbl)
```

Функция тренировки модели

```
def model_train(label_smoothing:float=1e-5, weight_decay:float=1e-5,
num_epochs:int=25):
    dataloader_test, data_test_size =
CreateDataloader(data_type='test', transforms=tr, shuffle=False,
state=True)
    dataloader_train, data_train_size =
CreateDataloader(data_type='train', transforms=tr, shuffle=True,
state=False) # train
    model = Model()

    optimizer = optim.AdamW(model.parameters(), lr=0.001,
weight_decay=weight_decay)
    scheduler = lr_scheduler.ReduceLROnPlateau(optimizer, 'min',
factor=0.5)
    model.to(device)
    metric = CharErrorRate()
    criterion = nn.CrossEntropyLoss(label_smoothing=label_smoothing)

    for ep in range(num_epochs):
        running_loss = 0.0
        running_corrects = 0
```

```
        running = 0
        loss = 0.0
        err = 0
        model.train()
        for img, lbl in tqdm(dataloader_train):
            img = img.to(device, non_blocking=True)
            optimizer.zero_grad()
            out = model(img)
            lbl_convert = ConverLabelToNum(d, lbl).to(device,
non_blocking=True)
            _, preds = torch.max(out, 2)
            for out_iter, lbl_iter in zip(out, lbl_convert):
                loss += criterion(out_iter, lbl_iter)
            loss.backward()
            optimizer.step()
            running_loss += loss * out.size(1)
            loss = 0.0
            running_corrects += accuracy(preds, lbl_convert.data)
            running += torch.sum(preds == lbl_convert.data)
            err += CER(inv_d, lbl, preds, metric)
        epoch_loss = running_loss / (data_train_size * out.size(1))
        epoch_acc = running_corrects / (data_train_size * out.size(1))
        print(f'Epoch {ep + 1} train: loss: {epoch_loss},  char_acc:
{running / (data_train_size * out.size(1))}, accuracy: {epoch_acc},
CER: {err / data_train_size}')
        running_loss = 0.0
        running_corrects = 0
        running = 0
        loss = 0.0
        err = 0
        model.eval()
        with torch.no_grad():
            for img, lbl in tqdm(dataloader_test):
                img = img.to(device, non_blocking=True)
                optimizer.zero_grad()
                out = model(img)
                lbl_convert = ConverLabelToNum(d, lbl).to(device,
non_blocking=True)
                _, preds = torch.max(out, 2)
                for out_iter, lbl_iter in zip(out, lbl_convert):
                    loss += criterion(out_iter, lbl_iter)
                running_loss += loss * out.size(1)
                loss = 0.0
                running_corrects += accuracy(preds, lbl_convert.data)
                running += torch.sum(preds == lbl_convert.data)
                err += CER(inv_d, lbl, preds, metric)
            epoch_loss = running_loss / (data_test_size * out.size(1))
            epoch_acc = running_corrects / (data_test_size *
out.size(1))
            print(f'Epoch {ep + 1} val: loss: {epoch_loss}, char_acc:
```

```
{running / (data_test_size * out.size(1))}, accuracy: {epoch_acc},
CER: {err / data_test_size}')
            scheduler.step(epoch_loss)
    return model

model_train(label_smoothing=1e-2, weight_decay=1e-3, num_epochs=30)
```

```
100%|██████████| 782/782 [07:36<00:00,  1.71it/s]

Epoch 1 train: loss: 1.8567851781845093,  char_acc: 0.518879771232605,
accuracy: 0.006472075779006472, CER: 0.0018750018207356334

100%|██████████| 40/40 [00:13<00:00,  2.91it/s]

Epoch 1 val: loss: 0.7625036239624023, char_acc: 0.8160673379898071,
accuracy: 0.03701798751303702, CER: 0.0007356948917731643

100%|██████████| 782/782 [07:26<00:00,  1.75it/s]

Epoch 2 train: loss: 0.8289841413497925,  char_acc:
0.7943872809410095, accuracy: 0.05313031303130313, CER:
0.0008015538332983851

100%|██████████| 40/40 [00:13<00:00,  2.88it/s]

Epoch 2 val: loss: 0.4584707021713257, char_acc: 0.9005900621414185,
accuracy: 0.07856499935707857, CER: 0.0003970635880250484

100%|██████████| 782/782 [07:29<00:00,  1.74it/s]

Epoch 3 train: loss: 0.635337233543396,  char_acc: 0.8425799608230591,
accuracy: 0.07535967882502535, CER: 0.0006137409363873303

100%|██████████| 40/40 [00:13<00:00,  2.88it/s]

Epoch 3 val: loss: 0.37493717670440674, char_acc: 0.9209206700325012,
accuracy: 0.09156629948709157, CER: 0.0003149012627545744

100%|██████████| 782/782 [07:34<00:00,  1.72it/s]

Epoch 4 train: loss: 0.5687251091003418,  char_acc: 0.860406756401062,
accuracy: 0.0825046790393325, CER: 0.0005444237031042576

100%|██████████| 40/40 [00:14<00:00,  2.85it/s]

Epoch 4 val: loss: 0.34645402431488037, char_acc: 0.9281642436981201,
accuracy: 0.09618104667609619, CER: 0.00028492434648796916

100%|██████████| 782/782 [07:33<00:00,  1.72it/s]

Epoch 5 train: loss: 0.5318421125411987,  char_acc:
0.8709792494773865, accuracy: 0.08667009558098666, CER:
0.0005032974295318127

100%|██████████| 40/40 [00:13<00:00,  2.88it/s]
```

```
Epoch 5 val: loss: 0.33238011598587036, char_acc: 0.9330790042877197,
accuracy: 0.09858128670009858, CER: 0.0002666226355358958

100%|██████████| 782/782 [07:44<00:00,  1.68it/s]

Epoch 6 train: loss: 0.5030744075775146,  char_acc:
0.8791272044181824, accuracy: 0.08928964325003928, CER:
0.0004714971873909235

100%|██████████| 40/40 [00:14<00:00,  2.83it/s]

Epoch 6 val: loss: 0.3111875057220459, char_acc: 0.9384509921073914,
accuracy: 0.10206734959210206, CER: 0.0002493367064744234

100%|██████████| 782/782 [07:39<00:00,  1.70it/s]

Epoch 7 train: loss: 0.4838995635509491,  char_acc:
0.8845641613006592, accuracy: 0.09135770719929136, CER:
0.00045025479630567133

100%|██████████| 40/40 [00:14<00:00,  2.82it/s]

Epoch 7 val: loss: 0.3074006140232086, char_acc: 0.9395082592964172,
accuracy: 0.10332461817610332, CER: 0.00024240516358986497

100%|██████████| 782/782 [07:39<00:00,  1.70it/s]

Epoch 8 train: loss: 0.4709721803665161,  char_acc:
0.8883402943611145, accuracy: 0.0927942794279428, CER:
0.0004356029094196856

100%|██████████| 40/40 [00:13<00:00,  2.87it/s]

Epoch 8 val: loss: 0.29945072531700134, char_acc: 0.9421370625495911,
accuracy: 0.10436757961510437, CER: 0.000229334706091322

100%|██████████| 782/782 [07:27<00:00,  1.75it/s]

Epoch 9 train: loss: 0.45876970887184143,  char_acc:
0.8919827938079834, accuracy: 0.09381366708099381, CER:
0.00042141301673837006

100%|██████████| 40/40 [00:13<00:00,  2.92it/s]

Epoch 9 val: loss: 0.30085158348083496, char_acc: 0.941165566444397,
accuracy: 0.10396753961110397, CER: 0.0002376688498770818

100%|██████████| 782/782 [07:30<00:00,  1.73it/s]

Epoch 10 train: loss: 0.4473724663257599,  char_acc:
0.8952316641807556, accuracy: 0.09501735887874502, CER:
0.00040875450940802693

100%|██████████| 40/40 [00:13<00:00,  2.94it/s]
```

```
Epoch 10 val: loss: 0.2925727367401123, char_acc: 0.9435943961143494,
accuracy: 0.10502478819310503, CER: 0.00022470623662229627

100%|██████████| 782/782 [07:27<00:00,  1.75it/s]

Epoch 11 train: loss: 0.46112462878227234,  char_acc:
0.8921034932136536, accuracy: 0.09319003328904318, CER:
0.00042118324199691415

100%|██████████| 40/40 [00:13<00:00,  2.90it/s]

Epoch 11 val: loss: 0.2790398895740509, char_acc: 0.948051929473877,
accuracy: 0.10819653393910819, CER: 0.00020807885448448836

100%|██████████| 782/782 [07:33<00:00,  1.72it/s]

Epoch 12 train: loss: 0.43275952339172363,  char_acc:
0.8995749950408936, accuracy: 0.09675038932464675, CER:
0.00039180615567602217

100%|██████████| 40/40 [00:13<00:00,  2.87it/s]

Epoch 12 val: loss: 0.2794395983219147, char_acc: 0.9480233788490295,
accuracy: 0.10848227679910848, CER: 0.00021093628311064094

100%|██████████| 782/782 [07:27<00:00,  1.75it/s]

Epoch 13 train: loss: 0.4286563992500305,  char_acc:
0.9008393883705139, accuracy: 0.09716471647164716, CER:
0.00038687893538735807

100%|██████████| 40/40 [00:16<00:00,  2.49it/s]

Epoch 13 val: loss: 0.27603912353515625, char_acc: 0.949223518371582,
accuracy: 0.10921092109210921, CER: 0.00020350255363155156

100%|██████████| 782/782 [07:30<00:00,  1.74it/s]

Epoch 14 train: loss: 0.4223909378051758,  char_acc:
0.9024723768234253, accuracy: 0.09763904961924764, CER:
0.00038049707654863596

100%|██████████| 40/40 [00:15<00:00,  2.57it/s]

Epoch 14 val: loss: 0.2787240147590637, char_acc: 0.9483948349952698,
accuracy: 0.10866800965810867, CER: 0.00020679523004218936

100%|██████████| 782/782 [07:25<00:00,  1.76it/s]

Epoch 15 train: loss: 0.42099496722221375,  char_acc:
0.9029895663261414, accuracy: 0.0978033517637478, CER:
0.0003783924912568182

100%|██████████| 40/40 [00:13<00:00,  2.91it/s]
```

Epoch 15 val: loss: 0.28025251626968384, char_acc: 0.9475519061088562, accuracy: 0.10839655394110839, CER: 0.00020835042232647538

100%|██████████| 782/782 [07:24<00:00, 1.76it/s]

Epoch 16 train: loss: 0.4165715277194977, char_acc: 0.9041961431503296, accuracy: 0.09827911362564828, CER: 0.00037372848601080477

100%|██████████| 40/40 [00:13<00:00, 2.90it/s]

Epoch 16 val: loss: 0.2838301956653595, char_acc: 0.9462375044822693, accuracy: 0.10762504821910762, CER: 0.00021331745665520430

100%|██████████| 782/782 [07:25<00:00, 1.76it/s]

Epoch 17 train: loss: 0.4124135673046112, char_acc: 0.9050226807594299, accuracy: 0.09847556184189847, CER: 0.0003707066352944821

100%|██████████| 40/40 [00:14<00:00, 2.77it/s]

Epoch 17 val: loss: 0.28496626019477844, char_acc: 0.9456660151481628, accuracy: 0.10653922535110653, CER: 0.00021661390201188624

100%|██████████| 782/782 [07:25<00:00, 1.76it/s]

Epoch 18 train: loss: 0.4094652831554413, char_acc: 0.9061884880065918, accuracy: 0.09883345477404884, CER: 0.00036614114651456475

100%|██████████| 40/40 [00:15<00:00, 2.54it/s]

Epoch 18 val: loss: 0.27151376008987427, char_acc: 0.9497092366218567, accuracy: 0.10948237680910948, CER: 0.00020255746494513005

100%|██████████| 782/782 [07:29<00:00, 1.74it/s]

Epoch 19 train: loss: 0.4056735932826996, char_acc: 0.907252848148346, accuracy: 0.09921277842069921, CER: 0.0003618175978772342

100%|██████████| 40/40 [00:13<00:00, 2.89it/s]

Epoch 19 val: loss: 0.2796367108821869, char_acc: 0.9483519792556763, accuracy: 0.10819653393910819, CER: 0.00020696267893072218

100%|██████████| 782/782 [07:25<00:00, 1.76it/s]

Epoch 20 train: loss: 0.40058350563049316, char_acc: 0.9086322784423828, accuracy: 0.09987855928449987, CER: 0.0003566835366655141

100%|██████████| 40/40 [00:13<00:00, 2.90it/s]

Epoch 20 val: loss: 0.2658836543560028, char_acc: 0.9522380828857422, accuracy: 0.11046818967611047, CER: 0.00019099752535112202

100%|██████████| 782/782 [07:31<00:00, 1.73it/s]

Epoch 21 train: loss: 0.4019123911857605, char_acc: 0.9082808494567871, accuracy: 0.09982998299829983, CER: 0.00035791919799521565

100%|██████████| 40/40 [00:13<00:00, 2.89it/s]

Epoch 21 val: loss: 0.267156720161438, char_acc: 0.9517523050308228, accuracy: 0.11048247681911048, CER: 0.00019289502233732492

100%|██████████| 782/782 [07:26<00:00, 1.75it/s]

Epoch 22 train: loss: 0.3971936106681824, char_acc: 0.9094659686088562, accuracy: 0.10013429914420013, CER: 0.00035321267205290496

100%|██████████| 40/40 [00:13<00:00, 2.90it/s]

Epoch 22 val: loss: 0.2635924816131592, char_acc: 0.952580988407135, accuracy: 0.1112111211121112, CER: 0.00019139560754410923

100%|██████████| 782/782 [07:28<00:00, 1.74it/s]

Epoch 23 train: loss: 0.3940209746360779, char_acc: 0.910578191280365, accuracy: 0.10062363379195062, CER: 0.00034887457150034606

100%|██████████| 40/40 [00:13<00:00, 2.93it/s]

Epoch 23 val: loss: 0.264801949262619, char_acc: 0.9522380828857422, accuracy: 0.11118254682611119, CER: 0.00019273506768513 47

100%|██████████| 782/782 [07:28<00:00, 1.74it/s]

Epoch 24 train: loss: 0.3927193582057953, char_acc: 0.9108639359474182, accuracy: 0.10073793093595074, CER: 0.00034787729964591563

100%|██████████| 40/40 [00:13<00:00, 2.90it/s]

Epoch 24 val: loss: 0.26273638010025024, char_acc: 0.9527952671051025, accuracy: 0.11158258683011159, CER: 0.00018971762619912624

100%|██████████| 782/782 [07:31<00:00, 1.73it/s]

Epoch 25 train: loss: 0.39044690132141113, char_acc: 0.9115097522735596, accuracy: 0.10091223408055092, CER: 0.00034537925967015326

100%|██████████| 40/40 [00:13<00:00, 2.90it/s]

```
Epoch 25 val: loss: 0.2574233114719391, char_acc: 0.9540525674819946,
accuracy: 0.11215407255011216, CER: 0.00018480642756912857

100%|██████████| 782/782 [07:26<00:00,  1.75it/s]

Epoch 26 train: loss: 0.3892573118209839,  char_acc:
0.912127673625946, accuracy: 0.10116083036875116, CER:
0.0003429107600823045

100%|██████████| 40/40 [00:13<00:00,  2.90it/s]

Epoch 26 val: loss: 0.2669270634651184, char_acc: 0.9518951773643494,
accuracy: 0.11035389253211035, CER: 0.00019413027621340007

100%|██████████| 782/782 [07:28<00:00,  1.74it/s]

Epoch 27 train: loss: 0.3875270187854767,  char_acc:
0.9125733971595764, accuracy: 0.10139513951395139, CER:
0.00034144692472182214

100%|██████████| 40/40 [00:13<00:00,  2.89it/s]

Epoch 27 val: loss: 0.26594510674476624, char_acc: 0.9522380828857422,
accuracy: 0.11092537825211092, CER: 0.00019189418526366353

100%|██████████| 782/782 [07:29<00:00,  1.74it/s]

Epoch 28 train: loss: 0.3859250247478485,  char_acc:
0.9129284620285034, accuracy: 0.1012965582272513, CER:
0.0003396820102352649

100%|██████████| 40/40 [00:13<00:00,  2.93it/s]

Epoch 28 val: loss: 0.26369455456733704, char_acc: 0.9524524211883545,
accuracy: 0.11062534824911062, CER: 0.00019469954713713378

100%|██████████| 782/782 [07:29<00:00,  1.74it/s]

Epoch 29 train: loss: 0.38409167528152466,  char_acc:
0.9135385155677795, accuracy: 0.10182661123255182, CER:
0.000337468198267743

100%|██████████| 40/40 [00:15<00:00,  2.60it/s]

Epoch 29 val: loss: 0.2619416117668152, char_acc: 0.9530239105224609,
accuracy: 0.11132541825611132, CER: 0.00018798380915541202

100%|██████████| 782/782 [07:28<00:00,  1.74it/s]

Epoch 30 train: loss: 0.3957415819168091,  char_acc:
0.9103460311889648, accuracy: 0.10028431414570028, CER:
0.00034999812487512827

100%|██████████| 40/40 [00:13<00:00,  2.90it/s]
```

```
Epoch 30 val: loss: 0.26697391271591187, char_acc: 0.9521952271461487,
accuracy: 0.11078250682211079, CER: 0.00019200582755729556

Model(
  (conv1): Sequential(
    (0): Conv2d(3, 7, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (1): BatchNorm2d(7, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (maxpool1): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
  (conv2): Sequential(
    (0): Conv2d(7, 7, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (1): BatchNorm2d(7, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (maxpool2): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
  (fc1): Sequential(
    (0): Linear(in_features=512, out_features=32, bias=True)
    (1): ReLU(inplace=True)
  )
  (gru1): GRU(32, 512)
  (gru1_b): GRU(32, 512)
  (gru2): GRU(512, 512)
  (gru2_b): GRU(512, 512)
  (fc2): Sequential(
    (0): Linear(in_features=1024, out_features=66, bias=True)
    (1): ReLU(inplace=True)
  )
)
```

Подведение итогов

Были выполнены все условия предложенной работы, за исключением анализа ошибок.

Максильные результаты по метрикам не были достигнуты, поскольку была проведена только часть экспериментов. Улчшение качества можно получить потем более точного подбора аугментаций (в работе был применен Autoaugment -- он преобразует изображение только в цветовой палитре).

Также можно провести эксперименты с масштабированием применяемой архитектуры, посмотреть на результаты: с разными шедулерами, TTA, ema, оптимизаторами (к ним можно запустить поиск гиперпарамтров), L2, докинуть на полносвязные слои Dropout.