```python
import torch
from tqdm import tqdm
from torchvision import transforms

from pathlib import Path
import os
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

import numpy as np

import pickle
from sklearn.preprocessing import LabelEncoder
from torch.utils.data import Dataset, DataLoader
from random import randint

from sklearn.model_selection import train_test_split

from torch import nn

import gc
```

▶ Launch TensorBoard Session

```python
from torch.utils.tensorboard import SummaryWriter
```

Python

#Инициализируем видеокарту, записываем пути, куда мы в будущем положим нашу обучающую выборку

```python
DEVICE = torch.device ("cuda")
DATASET_PATH = "/content/drive/MyDrive/dataset/"
IMAGE_SIZE = (260, 325)
```

Python

```python
!nvidia-smi
```

Python

#Путь до директории с изначальной десяткой картинок

```python
SOURCE_PATH = "/content/drive/MyDrive/start/"
```

Python

#Проверяем, что cuda доступна

```python
torch.cuda.is_available ()
```

Python

```
True
```

#Инициализируем сиды, чтобы при следующих запусках обучения, данные не менялись

```python
random_seed = 42 # or any of your favorite number
torch.manual_seed(random_seed)
torch.cuda.manual_seed(random_seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
np.random.seed(random_seed)
```

Python

#Функция transform, одна из основных. Она производит аугментацию картинок и сохраняет их в память

```python
def transform (file_, count, index):
    x, size = load_image (file_)
    x = x.convert('RGB')

    transforms_train1 = transforms.Compose([
            transforms.RandomRotation(degrees=(-5, 5), expand=True),
            transforms.RandomPerspective(p=0.4),
            transforms.RandomAffine (degrees=(-15, 15), translate=(0.15, 0.15)),
            transforms.RandomInvert(),

            transforms.Resize (IMAGE_SIZE)
        ])

    transforms_train2 = transforms.Compose([
            transforms.RandomRotation(degrees=(-5, 5), expand=True),

            transforms.RandomPerspective(p=0.4),
            transforms.RandomAffine (degrees=(-15, 15), translate=(0.15, 0.15)),
            transforms.CenterCrop ((IMAGE_SIZE[0], IMAGE_SIZE[1] * 2)),
            transforms.RandomInvert(),

            transforms.Resize (IMAGE_SIZE)
        ])

    for i in tqdm (range (count)):
        if (i < count / 2):
            new_x = transforms_train1(x)

        else:
            new_x = transforms_train2(x)

        new_x.save(DATASET_PATH + f"{index}/{index}{i}.png", "PNG")
```

Python

#Генерация обучающей выборки

```python
for i in range(10):
    transform (SOURCE_PATH + f"{i}.png", 2000, i)
```

Python

```
100%|██████████| 2000/2000 [00:25<00:00, 77.73it/s]
100%|██████████| 2000/2000 [00:24<00:00, 82.09it/s]
100%|██████████| 2000/2000 [00:25<00:00, 76.93it/s]
100%|██████████| 2000/2000 [00:28<00:00, 70.80it/s]
100%|██████████| 2000/2000 [00:27<00:00, 73.94it/s]
100%|██████████| 2000/2000 [00:26<00:00, 75.09it/s]
100%|██████████| 2000/2000 [00:26<00:00, 75.63it/s]
100%|██████████| 2000/2000 [00:25<00:00, 78.12it/s]
100%|██████████| 2000/2000 [00:26<00:00, 75.14it/s]
100%|██████████| 2000/2000 [00:27<00:00, 71.74it/s]
```

#Класс датасет, который используется для удобства доступа к данным

```
class NumbersDataset(Dataset):
    def __init__(self, files, mode):
        super().__init__()
        self.data_modes = ['train', 'val', 'test']
        self.files = sorted(files)
        self.mode = mode

        if self.mode not in self.data_modes:
            print(f"{self.mode} is not correct; correct modes: {self.data_modes}")
            raise NameError

        self.len_ = len(self.files)

        self.label_encoder = LabelEncoder()

        if self.mode != 'test':
            self.labels = [int(path.parent.name) for path in self.files]

    def __getitem__(self, index):

        x, size = self.load_sample(self.files[index])

        transforms_test = transforms.Compose([
            transforms.Resize (IMAGE_SIZE),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ])

        transforms_train = transforms.Compose([
            transforms.Resize (IMAGE_SIZE),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ])

        transforms_val = transforms.Compose([
            transforms.Resize (IMAGE_SIZE),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ])
```

```python
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])

    transforms_val = transforms.Compose([
        transforms.Resize (IMAGE_SIZE),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])

    x = x.convert('RGB')
    y = int (self.files[index].parent.name)

    if self.mode == 'test':
        x = transforms_test(x)
        return x, y

    elif self.mode == 'train':
        x = transforms_train(x)
        return x, y

    elif self.mode == 'val':
        x = transforms_val(x)
        return x, y

def __len__(self):
    return self.len_

def load_sample(self, file):
    image = Image.open(file)
    image.load()

    return image, image.size
```

Python

#Загрузка файлов, разбиение на test и val выборки

```python
TRAIN_DIR = Path (DATASET_PATH)

train_val_files = sorted (list (TRAIN_DIR.rglob ("*.png")))
```

```python
train_val_labels = [path.parent.name for path in train_val_files]
train_files, val_files = train_test_split (train_val_files, test_size=0.10, stratify=train_val_labels)
```

```python
train_dataset = NumbersDataset (train_files, 'train')
val_dataset   = NumbersDataset (val_files, 'val')
```

#Функция для вывода картинок из датасета с использованием matplotlib

```python
def imshow (inp, title=None, plt_ax=plt):

    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array ([0.485, 0.456, 0.406])
    std = np.array ([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip (inp, 0, 1)

    plt_ax.imshow (inp)

    if title is not None:
        plt.title (title)

    plt_ax.grid(False)
```

```python
class SimpleCnn_exp(nn.Module):

    def __init__(self, n_classes):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Dropout (0.4),
            # nn.BatchNorm2d (3),
            nn.Conv2d(in_channels=3, out_channels=8, kernel_size=3),
            # nn.BatchNorm2d (8),
            nn.LeakyReLU(),
            nn.AvgPool2d (kernel_size=2)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3),
            # nn.BatchNorm2d (16),
            nn.LeakyReLU(),
            nn.AvgPool2d (kernel_size=2)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3),
            # nn.BatchNorm2d (32),
            nn.LeakyReLU(),
            nn.AvgPool2d (kernel_size=2)
        )
        self.conv4 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3),
            # nn.BatchNorm2d (64),
            nn.LeakyReLU(),
            nn.AvgPool2d (kernel_size=2)
        )
        self.conv5 = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3),
            # nn.BatchNorm2d (128),
            nn.LeakyReLU(),
            nn.AvgPool2d (kernel_size=2),
            # nn.BatchNorm2d (128)
        )

        self.fc = nn.Sequential(
            nn.Linear(128 * 6 * 8, 64),
            nn.ReLU(),
            nn.Linear(64, n_classes)
        )
```

```
    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)

        # print (x.shape)

        x = x.view(x.size(0), -1)

        # print (x.shape)

        logits = self.fc (x)

        return logits
```

#Инициализация модели, перенос ее на видеокарту

```python
model = SimpleCnn (10)
model.to(DEVICE)
```

#Функция обучения нейросети (одна эпоха)

```python
def fit_epoch(model, train_loader, criterion, optimizer, writer, epoch_num):
    model.train()
    running_loss = 0.0
    running_corrects = 0
    processed_data = 0

    for iter, data in enumerate(train_loader):
        inputs, labels = data
        inputs = inputs.to(DEVICE)
        labels = labels.to(DEVICE)
        optimizer.zero_grad()

        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        preds = torch.argmax(outputs, 1)
        running_loss += loss.item() * inputs.size(0)
        running_corrects += torch.sum(preds == labels)
        processed_data += inputs.size(0)

        writer.add_scalar ("train loss", loss.item(), iter + epoch_num * len(train_loader))

    train_loss = running_loss / processed_data
    train_acc = running_corrects.cpu().numpy() / processed_data

    return train_loss, train_acc
```

Python

#Функция обработки валидационной выборки (в одной эпохе)

```python
def eval_epoch(model, val_loader, criterion, writer, epoch_num):
    model.eval()
    running_loss = 0.0
    running_corrects = 0
    processed_size = 0

    for iter, data in enumerate(val_loader):
        inputs, labels = data
        inputs = inputs.to(DEVICE)
        labels = labels.to(DEVICE)

        with torch.set_grad_enabled(False):
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            preds = torch.argmax(outputs, 1)

        running_loss += loss.item() * inputs.size(0)
        running_corrects += torch.sum(preds == labels)
        processed_size += inputs.size(0)

        writer.add_scalar ("val loss", loss.item(), iter + epoch_num * len(val_loader))

    val_loss = running_loss / processed_size
    val_acc = running_corrects.double() / processed_size
    return val_loss, val_acc
```

Python

```python
from torch.optim.lr_scheduler import ReduceLROnPlateau

def train_exp (train_files, val_files, model, epochs, batch_size):
    writer = SummaryWriter(log_dir="/content/runs/")
    best_loss = float("inf")

    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

    with tqdm(desc="epoch", total=epochs) as pbar_outer:
        opt = torch.optim.AdamW(model.parameters())

        scheduler = ReduceLROnPlateau(opt, 'min', min_lr=1e-8)

        criterion = nn.CrossEntropyLoss()

        for epoch in range(epochs):
            train_loss, train_acc = fit_epoch(model, train_loader, criterion, opt, writer, epoch)

            val_loss, val_acc = eval_epoch(model, val_loader, criterion, writer, epoch)

            scheduler.step (val_loss)

            if val_loss < best_loss:
                best_loss = val_loss
                torch.save (model.state_dict(), f"/content/drive/MyDrive/model/model_exp_6_best.pt")

            pbar_outer.update(1)

    writer.close()

    torch.save (model.state_dict(), "/content/drive/MyDrive/model/model_exp_6_final.pt")
```

Python

#Функция которая возвращает предсказанные значения для конкретного объекта

```python
def predict(model, test_loader):
    model.eval()
    with torch.no_grad():
        logits = []

        for inputs in test_loader:
            inputs = inputs.to(DEVICE)
            outputs = model(inputs).cpu()
            logits.append(outputs)

    probs = nn.functional.softmax(torch.cat(logits), dim=-1).numpy()

    return probs
```

Python

```python
def predict_one_sample(model, inputs, device=DEVICE):
    model.eval()

    with torch.no_grad():
        inputs = inputs.to(device)
        logit = model(inputs).cpu()
        probs = torch.nn.functional.softmax(logit, dim=-1).numpy()
    return probs
```

Python

#Высчитываем процент "угадываний" и f1 метрику на валидационной выборке

```python
idxs = list(map(int, np.random.uniform(0, len(val_dataset), 1000)))

imgs = [val_dataset[id][0].unsqueeze(0) for id in idxs]
actual_labels = [val_dataset[id][1] for id in idxs]
```

Python

```python
probs_ims = predict(test_model, imgs)
```

Python

```python
y_pred = np.argmax(probs_ims,-1)
```

Python

```python
sum (y_pred == actual_labels) / y_pred.shape[0]
```

Python

```
0.978
```

```python
val_results = y_pred == actual_labels
val_results.size
```

Python

```
1000
```

```python
from sklearn.metrics import f1_score

f1_score(actual_labels, y_pred, average='macro')
```

Python

```
0.9785104549525597
```

#Разумно сгенерировать тестовую выборку немного с иными параметрами, чтобы проверить эффективность модели при отклонениях

В половине файлов тестовой выборки используется размытие Гаусса

```python
from torchvision.transforms.transforms import GaussianBlur
def test_transform (file_, count, index):
    x, size = load_image (file_)
    x = x.convert('RGB')

    transforms_train1 = transforms.Compose([
            transforms.RandomAffine (degrees=(-15, 15), translate=(0.15, 0.15)),
            transforms.RandomInvert(),
            transforms.Resize (IMAGE_SIZE)
        ])

    transforms_train2 = transforms.Compose([
            transforms.RandomAffine (degrees=(-15, 15), translate=(0.15, 0.15)),
            transforms.RandomInvert(),
            transforms.GaussianBlur (kernel_size=(int(IMAGE_SIZE[0] * 0.3 + 1), int(IMAGE_SIZE[1] * 0.3)), sigma=(1.0, 2.0)),
            transforms.Resize (IMAGE_SIZE)
        ])

    for i in tqdm (range (count)):
        new_x = transforms_train1(x)
        new_x.save(DATASET_TEST_PATH + f"{index}/{index}{i}.png", "PNG")

    for i in tqdm (range (count)):
        new_x = transforms_train2(x)
        new_x.save(DATASET_TEST_PATH + f"{index}/{index}{i + count}.png", "PNG")
```

Python

Проверка на тестовой выборке

```python
TEST_DIR = Path (DATASET_TEST_PATH)

test_files = sorted (list (TEST_DIR.rglob ("*.png")))
```

Python

```python
test_dataset = NumbersDataset (test_files, 'test')
```

Python

```python
test_imgs   = [test_dataset[id][0].unsqueeze(0) for id in range(len(test_dataset))]
test_labels = [test_dataset[id][1] for id in range(len(test_dataset))]
```

Python

```python
test_probs = predict (test_model, test_imgs)
```

Python

```python
test_pred = np.argmax (test_probs, -1)
```

Python

```python
test_results = test_labels == test_pred
test_results
```

Python

```
array([False, False,  True, ..., False, False, False])
```

Процент "угадываний"

```python
sum (test_results) / len (test_results)
```

Python

```
0.3585
```

f1 метрика на тестовой выборке

```python
from sklearn.metrics import f1_score

f1_score (test_labels, test_pred, average="macro")
```

Python

```
0.31978938602173634
```

f1 метрика отдельно для каждого класса картинок (для класса нулей, единиц и т.д.)

```python
for i in range(10):
    print(f1_score (torch.ones(200), test_results[200 * i:200 * (i + 1)]))
```

Python

f1 метрика отдельно для каждого класса картинок (для класса нулей, единиц и т.д.)

```python
for i in range(10):
    print(f1_score (torch.ones(200), test_results[200 * i:200 * (i + 1)]))
```

Python

```
0.47908745247148293
0.7499999999999999
0.6710963455149502
0.9159891598915989
0.5611510791366906
0.09523809523809523
0.1042654028436019
0.5074626865671642
0.6440677966101694
0.019801980198019802
```

При высчитывании f1 метрики отдельного для каждого класса в тестовой выборке прослеживается тенденция маленького процента "отгадывания" цифры 9

1. Без нормализации батчей и дропаутов, с шедулером - 0.97 метрика на валидационной, 0.39 на тестовой

2. Без нормализации, дропаутов, шедулера - 0.97 на валидационной, 0.15 на тестовой

3. С нормализацией, шедулером, без дропаутов - 0.997 на валидационной, 0.18 на тестовой

4. С нормализацией только перед первым слоем, с шедулером, без дропаутов - 0.979 на валидационной, 0.299 на тестовой

5. С нормализацией перед первым слоем и перед линейными слоями, с шедулером, без дропаутов - 0.97 на валидационной, 0.17 на тестовой

6. Без нормализации, с дропаутом и шедулером - 0.98 на валидационной, 0.38 на тестовой

7. С дропаутом перед первым слоем, с шедулером, с нормализацией перед первым слоем, с заменой ReLU на LeakyReLU - 0.996 на валидационной, 0.315 на тестовой