Физтех-Школа Прикладной математики и информатики (ФПМИ) МФТИ

Путешествие по Спрингфилду.

Сегодня вам предстоить помочь телекомпании FOX в обработке их контента. Как вы знаете сериал Симсоны идет на телеэкранах более 25 лет и за это время скопилось очень много видео материала. Персоонажи менялись вместе с изменяющимися графическими технологиями и Гомер 2018 не очень похож на Гомера 1989. Нашей задачей будет научиться классифицировать персонажей проживающих в Спрингфилде. Думаю, что нет смысла представлять каждого из них в отдельности.

Установка зависимостей

Ввод [1]:

```
import torch
import numpy as np

train_on_gpu = torch.cuda.is_available()

if not train_on_gpu:
    print('CUDA is not available.  Training on CPU ...')
else:
    print('CUDA is available!  Training on GPU ...')
```

```
CUDA is available!  Training on GPU ...
```

Type *Markdown* and LaTeX: α^2

Ввод [2]:

```
# монтируем диск для датасета
from google.colab import drive
drive.mount('/content/gdrive/')
```

```
Mounted at /content/gdrive/
```

Ввод [3]:

```
# проверяем имя корневого каталога на Gdrive, где будет датасет
!ls /content/gdrive/
```

```
MyDrive
```

Ввод [10]:

```
# проверяем наличие на диске загруженного уже архива с датасетом
!ls -l /content/gdrive/MyDrive/journey-springfield.zip
```

```
-rw------- 1 root root 546224286 Dec 14 18:10 /content/gdrive/MyDrive/journe
y-springfield.zip
```

Ввод [11]:

```
# разархивируем датасет
!unzip -q /content/gdrive/MyDrive/journey-springfield.zip
```

Ввод [12]:

```
# проверяем, что есть в разархивированной папке:
!ls -1
```

```
characters_illustration.png
gdrive
sample_data
sample_submission.csv
testset
train
```

Ввод [13]:

```
# определяем переменные - пути к трэйн и тест датасетам:
from pathlib import Path
TRAIN_DIR = Path('train/simpsons_dataset')
TEST_DIR = Path('testset/testset')
```

Ввод [14]:

```
# проверка, что данные из датасетов грузятся:
train_val_files = list(TRAIN_DIR.rglob('*.jpg'))
test_files = list(TEST_DIR.rglob('*.jpg'))
print(len(train_val_files)) #=> 20933
print(len(test_files)) #=> 991
```

```
20933
991
```

Ввод [15]:

```
!nvidia-smi
import torch
torch.cuda.is_available()
```

```
Sun Dec 19 17:38:08 2021
+-----------------------------------------------------------------------------
--+
| NVIDIA-SMI 495.44       Driver Version: 460.32.03    CUDA Version: 11.2
|
|-------------------------------+----------------------+----------------------
--+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. EC
C |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute
M. |
|                               |                      |               MIG
M. |
|===============================+======================+======================
==|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |
0 |
| N/A   70C    P8    36W / 149W |      3MiB / 11441MiB |      0%      Defaul
t |
|                               |                      |                  N/
A |
+-------------------------------+----------------------+----------------------
--+

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

Out[15]:

```
True
```

В нашем тесте будет 990 картнок, для которых вам будет необходимо предсказать класс.

Ввод [16]:

```
import pickle
import numpy as np
from skimage import io

from tqdm import tqdm, tqdm_notebook
from PIL import Image
from pathlib import Path

from torchvision import transforms
from multiprocessing.pool import ThreadPool
from sklearn.preprocessing import LabelEncoder
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn

from matplotlib import colors, pyplot as plt
%matplotlib inline

# в sklearn не все гладко, чтобы в colab удобно выводить картинки
# мы будем игнорировать warnings
import warnings
warnings.filterwarnings(action='ignore', category=DeprecationWarning)
```

Ввод [17]:

```
# разные режимы датасета
DATA_MODES = ['train', 'val', 'test']
# все изображения будут масштабированы к размеру 224x224 px
RESCALE_SIZE = 224
# работаем на видеокарте
DEVICE = torch.device("cuda")
```

https://jhui.github.io/2018/02/09/PyTorch-Data-loading-preprocess_torchvision/ (https://jhui.github.io/2018/02/09/PyTorch-Data-loading-preprocess_torchvision/)

Ниже мы исспользуем враппер над датасетом для удобной работы. Вам стоит понимать, что происходит с LabelEncoder и с torch.Transformation.

ToTensor конвертирует PIL Image с параметрами в диапазоне [0, 255] (как все пиксели) в FloatTensor размера (C x H x W) [0,1] , затем производится масштабирование: $input = \frac{input - \mu}{\text{standard deviation}}$, константы - средние и дисперсии по каналам на основе ImageNet

Стоит также отметить, что мы переопределяем метод **getitem** для удобства работы с данной структурой данных. Также используется LabelEncoder для преобразования строковых меток классов в id и обратно. В описании датасета указано, что картинки разного размера, так как брались напрямую с видео, поэтому следуем привести их к одному размер (это делает метод _prepare_sample)

Ввод [18]:

```
class SimpsonsDataset(Dataset):
    """
    Датасет с картинками, который паралельно подгружает их из папок
    производит скалирование и превращение в торчевые тензоры
    """
    def __init__(self, files, mode):
        super().__init__()
        # список файлов для загрузки
        self.files = sorted(files)
        # режим работы
        self.mode = mode

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

        self.len_ = len(self.files)

        self.label_encoder = LabelEncoder()

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

            with open('label_encoder.pkl', 'wb') as le_dump_file:
                  pickle.dump(self.label_encoder, le_dump_file)

    def __len__(self):
        return self.len_

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

    def __getitem__(self, index):
        # для преобразования изображений в тензоры PyTorch и нормализации входа
        transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ])
        x = self.load_sample(self.files[index])
        x = self._prepare_sample(x)
        x = np.array(x / 255, dtype='float32')
        x = transform(x)
        if self.mode == 'test':
            return x
        else:
            label = self.labels[index]
            label_id = self.label_encoder.transform([label])
            y = label_id.item()
            return x, y

    def _prepare_sample(self, image):
        image = image.resize((RESCALE_SIZE, RESCALE_SIZE))
        return np.array(image)
```

Ввод [19]:

```
def imshow(inp, title=None, plt_ax=plt, default=False):
    """Imshow для тензоров"""
    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_ax.set_title(title)
    plt_ax.grid(False)
```

Ввод [20]:

```
TRAIN_DIR = Path('train/simpsons_dataset')
TEST_DIR = Path('testset/testset')

train_val_files = sorted(list(TRAIN_DIR.rglob('*.jpg')))
test_files = sorted(list(TEST_DIR.rglob('*.jpg')))
```

Ввод [21]:

```
from sklearn.model_selection import train_test_split

train_val_labels = [path.parent.name for path in train_val_files]
train_files, val_files = train_test_split(train_val_files, test_size=0.25, \
                                          stratify=train_val_labels)
```

Ввод [22]:

```
val_dataset = SimpsonsDataset(val_files, mode='val')
```

Давайте посмотрим на наших героев внутри датасета.

Ввод [23]:

```
fig, ax = plt.subplots(nrows=3, ncols=3,figsize=(8, 8), \
                       sharey=True, sharex=True)
for fig_x in ax.flatten():
    random_characters = int(np.random.uniform(0,1000))
    im_val, label = val_dataset[random_characters]
    img_label = " ".join(map(lambda x: x.capitalize(),\
                val_dataset.label_encoder.inverse_transform([label])[0].split('_')))
    imshow(im_val.data.cpu(), \
          title=img_label,plt_ax=fig_x)
```

Можете добавить ваши любимые сцены и классифицировать их. (веселые результаты можно кидать в чат)

Построение нейросети

Запустить данную сеть будет вашим мини-заданием на первую неделю, чтобы было проще участвовать в соревновании.

Данная архитектура будет очень простой и нужна для того, чтобы установить базовое понимание и получить простенький сабмит на Kaggle

Описание слоев:

1. размерность входа: 3x224x224
2. размерности после слоя: 8x111x111
3. 16x54x54
4. 32x26x26
5. 64x12x12
6. выход: 96x5x5

Ввод [ ]:

```
# Очень простая сеть
class SimpleCNN(nn.Module):

    def __init__(self, n_classes):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=8, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )
        self.conv4 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=128, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )
        self.conv5 = nn.Sequential(
            nn.Conv2d(in_channels=128, out_channels=96, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )

        self.out = nn.Linear(96 * 5 * 5, 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)

        x = x.view(x.size(0), -1)
        logits = self.out(x)
        return logits
```

Ввод [ ]:

```
# Пробуем BatchNorm:
class ModelBatchNorm(nn.Module):

    def __init__(self, n_classes):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=8, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(8)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(16)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(32)
        )
        self.conv4 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=128, kernel_size=3),

            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(128)

        )
        self.conv5 = nn.Sequential(
            nn.Conv2d(in_channels=128, out_channels=96, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)

        )

        self.out = nn.Linear(96 * 5 * 5, 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)

        x = x.view(x.size(0), -1)
        logits = self.out(x)
        return logits
```

Ввод [ ]:

```
# Пробуем Dropout:
class ModelDropout(nn.Module):

    def __init__(self, n_classes):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=8, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(8)

        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(16)

        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(32)

        )
        self.conv4 = nn.Sequential(

            nn.Conv2d(in_channels=32, out_channels=128, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(128)

        )
        self.conv5 = nn.Sequential(
            nn.Dropout(p=0.3),
            nn.Conv2d(in_channels=128, out_channels=96, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)


        )

        self.out = nn.Linear(96 * 5 * 5, 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)

        x = x.view(x.size(0), -1)
        logits = self.out(x)
        return logits
```

Ввод [24]:

```
# Продолжаем экспериментировать со слоями - MYCNN:
class MYCNN(nn.Module):

    def __init__(self, n_classes):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=8, kernel_size=(3, 3), stride=(1, 1), pad
            nn.BatchNorm2d(8),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),


        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(in_channels=8, out_channels=16, kernel_size=(3, 3), stride=(1, 1), pa
            nn.BatchNorm2d(16),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),


        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=(3, 3), stride=(1, 1), p
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),


        )
        self.conv4 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=(3, 3), stride=(1, 1), p
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),


        )
        self.conv5 = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=256, kernel_size=(3, 3), stride=(1, 1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),


        )

        self.conv14 = nn.Sequential(
            nn.AdaptiveAvgPool2d(output_size=(5, 5)),
            nn.Conv2d(in_channels=256, out_channels=128, kernel_size=(3, 3), stride=(1, 1),

            nn.ReLU(inplace=True),
            nn.Dropout(p=0.3, inplace=False)
        )
        self.conv15 = nn.Sequential(
            nn.Conv2d(in_channels=128, out_channels=96, kernel_size=(3, 3), stride=(1, 1),

            nn.ReLU(inplace=True),
            nn.Dropout(p=0.3, inplace=False)
        )
```

```
        self.out = nn.Linear(96 * 5 * 5, 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)

        x = self.conv14(x)
        x = self.conv15(x)

        x = x.view(x.size(0), -1)
        logits = self.out(x)
        return logits
```

Добавляем lr_scheduler:

Ввод [25]:

```
def fit_epoch(model, train_loader, criterion, optimizer, scheduler):
    running_loss = 0.0
    running_corrects = 0
    processed_data = 0

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

        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=2, gamma=0.1)
        scheduler.step()
        preds = torch.argmax(outputs, 1)
        running_loss += loss.item() * inputs.size(0)
        running_corrects += torch.sum(preds == labels.data)
        processed_data += inputs.size(0)

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

Ввод [26]:

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

    for inputs, labels in val_loader:
        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.data)
        processed_size += inputs.size(0)
    val_loss = running_loss / processed_size
    val_acc = running_corrects.double() / processed_size
    return val_loss, val_acc
```

Ввод [27]:

```
def train(train_files, val_files, model, epochs, batch_size):
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

    history = []
    log_template = "\nEpoch {ep:03d} train_loss: {t_loss:0.4f} \
    val_loss {v_loss:0.4f} train_acc {t_acc:0.4f} val_acc {v_acc:0.4f}"

    with tqdm(desc="epoch", total=epochs) as pbar_outer:
        opt = torch.optim.Adam(model.parameters())
        scheduler = torch.optim.lr_scheduler.StepLR(opt, step_size=2, gamma=0.1)
        #opt = torch.optim.Adam(model.parameters(), lr=0.0001)
        criterion = nn.CrossEntropyLoss()

        for epoch in range(epochs):
            train_loss, train_acc = fit_epoch(model, train_loader, criterion, opt, schedule
            print("loss", train_loss)

            val_loss, val_acc = eval_epoch(model, val_loader, criterion)
            history.append((train_loss, train_acc, val_loss, val_acc))

            pbar_outer.update(1)
            tqdm.write(log_template.format(ep=epoch+1, t_loss=train_loss,\
                                           v_loss=val_loss, t_acc=train_acc, v_acc=val_acc)

    return history
```

Ввод [28]:

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

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

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

Простая сеть:

Ввод [ ]:

```
n_classes = len(np.unique(train_val_labels))
simple_cnn = SimpleCNN(n_classes).to(DEVICE)
print("we will classify :{}".format(n_classes))
print(simple_cnn)
```

```
we will classify :42
SimpleCnn(
  (conv1): Sequential(
    (0): Conv2d(3, 8, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
  )
  (conv2): Sequential(
    (0): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
  )
  (conv3): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
  )
  (conv4): Sequential(
    (0): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
  )
  (conv5): Sequential(
    (0): Conv2d(128, 96, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
  )
  (out): Linear(in_features=2400, out_features=42, bias=True)
)
```

Батч нормы:

Ввод [ ]:

```
n_classes = len(np.unique(train_val_labels))
batch_norm = ModelBatchNorm(n_classes).to(DEVICE)
print("we will classify :{}".format(n_classes))
print(batch_norm)
```

```
we will classify :42
ModelBatchNorm(
  (conv1): Sequential(
    (0): Conv2d(3, 8, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
    (3): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
  )
  (conv2): Sequential(
    (0): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
    (3): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
  )
  (conv3): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
    (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
  )
  (conv4): Sequential(
    (0): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
  )
  (conv5): Sequential(
    (0): Conv2d(128, 96, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
  )
  (out): Linear(in_features=2400, out_features=42, bias=True)
)
```

Итоговая сеть:

Ввод [29]:

```
n_classes = len(np.unique(train_val_labels))
my_CNN = MYCNN(n_classes).to(DEVICE)
print("we will classify :{}".format(n_classes))
print(my_CNN)
```

```
we will classify :42
MYCNN(
  (conv1): Sequential(
    (0): Conv2d(3, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_runni
ng_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
  )
  (conv3): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
  )
  (conv4): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
  )
  (conv5): Sequential(
    (0): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
  )
  (conv14): Sequential(
    (0): AdaptiveAvgPool2d(output_size=(5, 5))
    (1): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.3, inplace=False)
  )
  (conv15): Sequential(
```

```
    (0): Conv2d(128, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.3, inplace=False)
  )
  (out): Linear(in_features=2400, out_features=42, bias=True)
)
```

Запустим обучение сети.

Ввод [30]:

```
if val_dataset is None:
    val_dataset = SimpsonsDataset(val_files, mode='val')

train_dataset = SimpsonsDataset(train_files, mode='train')
```

Результаты простой сети:

Ввод [ ]:

```
history = train(train_dataset, val_dataset, model=simple_cnn, epochs=2, batch_size=64)
```

```
epoch:   0%|          | 0/2 [00:00<?, ?it/s]

loss 2.547144606915452

epoch:  50%|█████     | 1/2 [03:01<03:01, 181.34s/it]


Epoch 001 train_loss: 2.5471     val_loss 1.9382 train_acc 0.2887 val_acc 0.
4671
loss 1.5861061661438893

epoch: 100%|██████████| 2/2 [05:59<00:00, 179.89s/it]


Epoch 002 train_loss: 1.5861     val_loss 1.3237 train_acc 0.5630 val_acc 0.
6395
```

Результаты с батч-норм:

Ввод [ ]:

```
history = train(train_dataset, val_dataset, model=batch_norm, epochs=4, batch_size=128)
```

```
epoch:   0%|          | 0/4 [00:00<?, ?it/s]

loss 1.8771517480040092

epoch:  25%|██        | 1/4 [03:03<09:10, 183.47s/it]


Epoch 001 train_loss: 1.8772     val_loss 1.3765 train_acc 0.5021 val_acc 0.
6229
loss 0.9935658911594396

epoch:  50%|█████     | 2/4 [06:04<06:04, 182.08s/it]


Epoch 002 train_loss: 0.9936     val_loss 0.9526 train_acc 0.7253 val_acc 0.
7442
loss 0.6067790430976719

epoch:  75%|███████   | 3/4 [08:59<02:58, 178.98s/it]


Epoch 003 train_loss: 0.6068     val_loss 0.7936 train_acc 0.8294 val_acc 0.
7893
loss 0.39144481629791955

epoch: 100%|██████████| 4/4 [11:53<00:00, 178.33s/it]


Epoch 004 train_loss: 0.3914     val_loss 0.7376 train_acc 0.8866 val_acc 0.
8091
```

Результаты с дропаут:

Ввод [ ]:

```
history = train(train_dataset, val_dataset, model=drop_out, epochs=4, batch_size=64)
```

```
epoch:   0%|          | 0/4 [00:00<?, ?it/s]

loss 1.8327388546559316

epoch:  25%|██        | 1/4 [03:00<09:00, 180.04s/it]


Epoch 001 train_loss: 1.8327     val_loss 1.2729 train_acc 0.5140 val_acc 0.
6666
loss 0.9209098321281806

epoch:  50%|█████     | 2/4 [06:00<06:00, 180.04s/it]


Epoch 002 train_loss: 0.9209     val_loss 0.9266 train_acc 0.7421 val_acc 0.
7447
loss 0.5833515053369018

epoch:  75%|███████   | 3/4 [08:58<02:59, 179.41s/it]


Epoch 003 train_loss: 0.5834     val_loss 0.7568 train_acc 0.8357 val_acc 0.
8059
loss 0.364153602197828

epoch: 100%|██████████| 4/4 [11:57<00:00, 179.46s/it]


Epoch 004 train_loss: 0.3642     val_loss 0.8172 train_acc 0.8943 val_acc 0.
7958
```

Результаты окончательной сетки:

Ввод [31]:

```
history = train(train_dataset, val_dataset, model=my_CNN, epochs=10, batch_size=128)
```

```
epoch:   0%|          | 0/10 [00:00<?, ?it/s]/usr/local/lib/python3.7/dist-p
ackages/torch/optim/lr_scheduler.py:134: UserWarning: Detected call of `lr_s
cheduler.step()` before `optimizer.step()`. In PyTorch 1.1.0 and later, you
should call them in the opposite order: `optimizer.step()` before `lr_schedu
ler.step()`.  Failure to do this will result in PyTorch skipping the first v
alue of the learning rate schedule. See more details at https://pytorch.org/
docs/stable/optim.html#how-to-adjust-learning-rate (https://pytorch.org/doc
s/stable/optim.html#how-to-adjust-learning-rate)
  "https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate",
 UserWarning)

loss 2.4993903396524315

epoch:  10%|▉         | 1/10 [03:03<27:27, 183.07s/it]


Epoch 001 train_loss: 2.4994     val_loss 1.8302 train_acc 0.3049 val_acc 0.
4989
loss 1.5045287799695486

epoch:  20%|█▊        | 2/10 [06:07<24:29, 183.63s/it]


Epoch 002 train_loss: 1.5045     val_loss 1.2793 train_acc 0.5822 val_acc 0.
6437
loss 1.036425022518827

epoch:  30%|██▋       | 3/10 [09:04<21:05, 180.78s/it]


Epoch 003 train_loss: 1.0364     val_loss 1.0376 train_acc 0.7085 val_acc 0.
7092
loss 0.7625440242290649

epoch:  40%|███▌      | 4/10 [12:02<17:59, 179.84s/it]


Epoch 004 train_loss: 0.7625     val_loss 0.8675 train_acc 0.7788 val_acc 0.
7635
loss 0.5899535724483286

epoch:  50%|████▌     | 5/10 [15:01<14:56, 179.31s/it]


Epoch 005 train_loss: 0.5900     val_loss 0.7830 train_acc 0.8229 val_acc 0.
7831
loss 0.43225151957452007

epoch:  60%|█████▍    | 6/10 [17:59<11:55, 178.98s/it]


Epoch 006 train_loss: 0.4323     val_loss 0.7399 train_acc 0.8680 val_acc 0.
8099
loss 0.3475781446528652

epoch:  70%|██████▎   | 7/10 [20:57<08:55, 178.60s/it]


Epoch 007 train_loss: 0.3476     val_loss 0.7701 train_acc 0.8931 val_acc
```

```
0.8057
loss 0.25741823823355253

epoch:  80%|████████  | 8/10 [23:53<05:55, 177.95s/it]


Epoch 008 train_loss: 0.2574     val_loss 0.7527 train_acc 0.9206 val_acc 0.
8225
loss 0.20519595342773156

epoch:  90%|█████████ | 9/10 [26:55<02:59, 179.03s/it]


Epoch 009 train_loss: 0.2052     val_loss 0.7772 train_acc 0.9345 val_acc 0.
8242
loss 0.1578348716020326

epoch: 100%|██████████| 10/10 [29:59<00:00, 179.91s/it]


Epoch 010 train_loss: 0.1578     val_loss 0.9348 train_acc 0.9474 val_acc 0.
8124
```

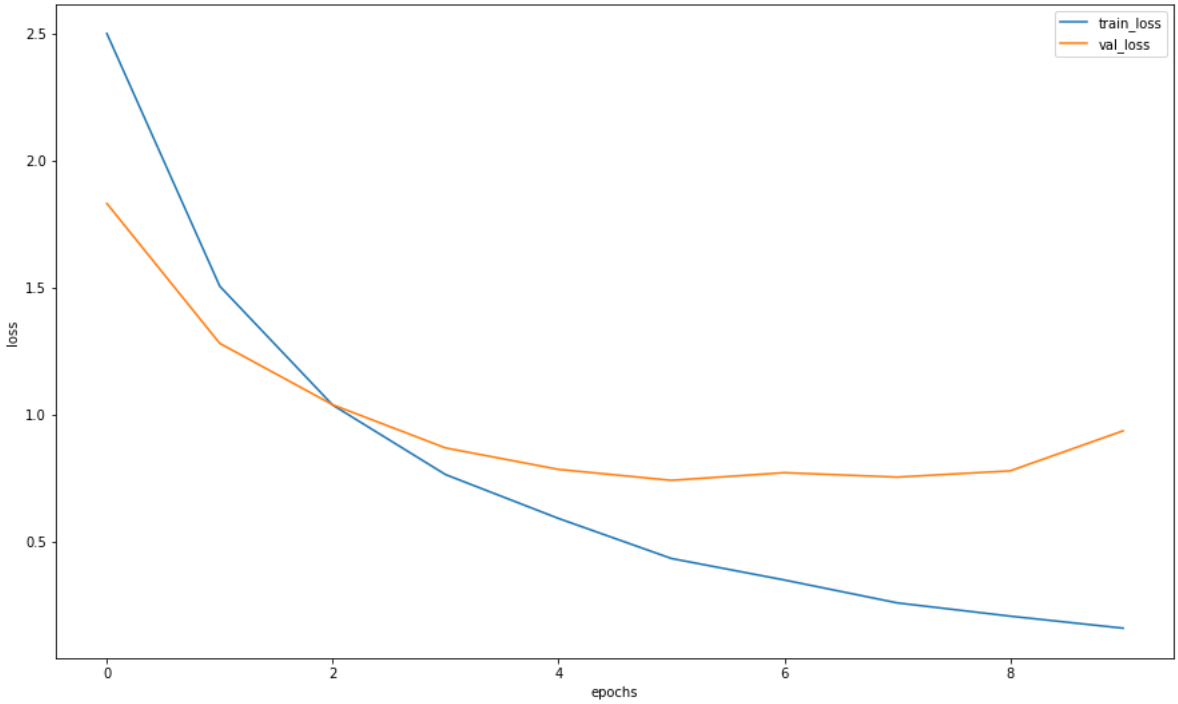
Построим кривые обучения

Ввод [32]:

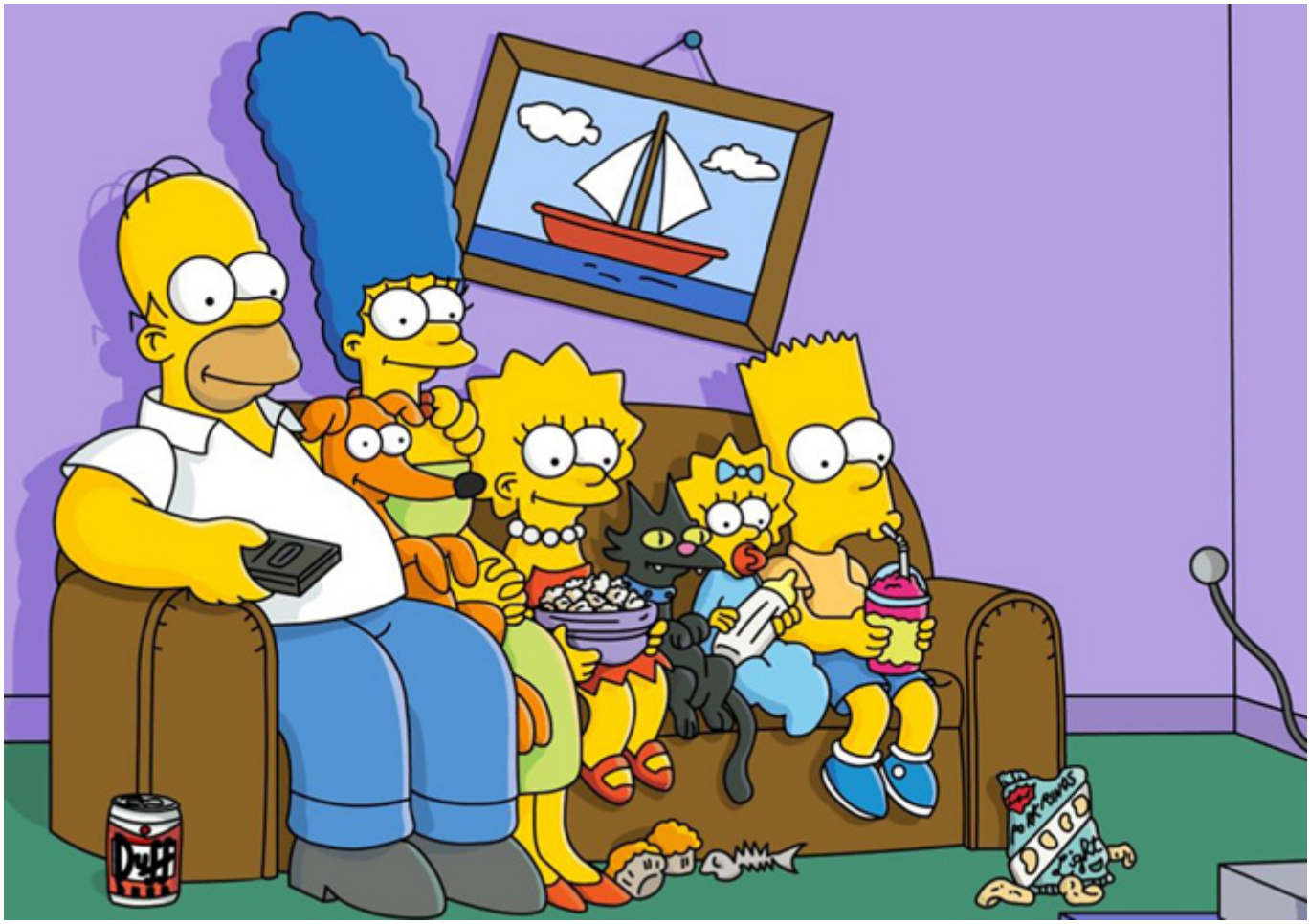
```
loss, acc, val_loss, val_acc = zip(*history)
```

Ввод [33]:

```
plt.figure(figsize=(15, 9))
plt.plot(loss, label="train_loss")
plt.plot(val_loss, label="val_loss")
plt.legend(loc='best')
plt.xlabel("epochs")
plt.ylabel("loss")
plt.show()
```



Ну и что теперь со всем этим делать?



Хорошо бы понять, как сделать сабмит. У нас есть сеть и методы eval у нее, которые позволяют перевести сеть в режим предсказания. Стоит понимать, что у нашей модели на последнем слое стоит softmax, которые позволяет получить вектор вероятностей того, что объект относится к тому или иному классу. Давайте воспользуемся этим.

Ввод [34]:

```
def predict_one_sample(model, inputs, device=DEVICE):
    """Предсказание, для одной картинки"""
    with torch.no_grad():
        inputs = inputs.to(device)
        model.eval()
        logit = model(inputs).cpu()
        probs = torch.nn.functional.softmax(logit, dim=-1).numpy()
    return probs
```

Ввод [35]:

```
random_characters = int(np.random.uniform(0,1000))
ex_img, true_label = val_dataset[random_characters]
probs_im = predict_one_sample(my_CNN, ex_img.unsqueeze(0))
```

Ввод [36]:

```
idxs = list(map(int, np.random.uniform(0,1000, 20)))
imgs = [val_dataset[id][0].unsqueeze(0) for id in idxs]

probs_ims = predict(my_CNN, imgs)
```

Ввод [37]:

```
label_encoder = pickle.load(open("label_encoder.pkl", 'rb'))
```

Ввод [38]:

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

actual_labels = [val_dataset[id][1] for id in idxs]

preds_class = [label_encoder.classes_[i] for i in y_pred]
```

Ввод [39]:

```
actual_labels
```

Out[39]:

```
[0, 6, 0, 6, 0, 2, 4, 2, 4, 6, 6, 2, 4, 0, 0, 4, 0, 2, 1, 4]
```

Обратите внимание, что метрика, которую необходимо оптимизировать в конкурсе --- f1-score. Вычислим целевую метрику на валидационной выборке.

Ввод [40]:

```
from sklearn.metrics import f1_score

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

Out[40]:

```
0.8000000000000002
```

Сделаем классную визуализацию, чтобы посмотреть насколько сеть уверена в своих ответах. Можете исспользовать это, чтобы отлаживать правильность вывода.

Ввод [41]:

```
import matplotlib.patches as patches
from matplotlib.font_manager import FontProperties

fig, ax = plt.subplots(nrows=3, ncols=3,figsize=(12, 12), \
                       sharey=True, sharex=True)
for fig_x in ax.flatten():
    random_characters = int(np.random.uniform(0,1000))
    im_val, label = val_dataset[random_characters]
    img_label = " ".join(map(lambda x: x.capitalize(),\
                val_dataset.label_encoder.inverse_transform([label])[0].split('_')))



    imshow(im_val.data.cpu(), \
          title=img_label,plt_ax=fig_x)

    actual_text = "Actual : {}".format(img_label)

    fig_x.add_patch(patches.Rectangle((0, 53),86,35,color='white'))
    font0 = FontProperties()
    font = font0.copy()
    font.set_family("fantasy")
    prob_pred = predict_one_sample(my_CNN, im_val.unsqueeze(0))
    predicted_proba = np.max(prob_pred)*100
    y_pred = np.argmax(prob_pred)

    predicted_label = label_encoder.classes_[y_pred]
    predicted_label = predicted_label[:len(predicted_label)//2] + '\n' + predicted_label[le
    predicted_text = "{} : {:.0f}%".format(predicted_label,predicted_proba)

    fig_x.text(1, 59, predicted_text , horizontalalignment='left', fontproperties=font,
                    verticalalignment='top',fontsize=8, color='black',fontweight='bold')
```

Ввод [ ]:

Попробуйте найти те классы, которые сеть не смогла расспознать. Изучите данную проблему, это понадобится в дальнейшем.

Submit на Kaggle

Ввод [ ]:

```
test_dataset = SimpsonsDataset(test_files, mode="test")
test_loader = DataLoader(test_dataset, shuffle=False, batch_size=64)
probs = predict(batch_norm, test_loader)


preds = label_encoder.inverse_transform(np.argmax(probs, axis=1))
test_filenames = [path.name for path in test_dataset.files]
```

Ввод [42]:

```
test_dataset = SimpsonsDataset(test_files, mode="test")
test_loader = DataLoader(test_dataset, shuffle=False, batch_size=64)
probs = predict(my_CNN, test_loader)
```

Ввод [43]:

```
preds = label_encoder.inverse_transform(np.argmax(probs, axis=1))
test_filenames = [path.name for path in test_dataset.files]
```

Ввод [44]:

```
! ls
```

```
characters_illustration.png  label_encoder.pkl  sample_submission.csv  train
gdrive                       sample_data        testset
```

Ввод [45]:

```
!ls -l /content/
```

```
total 636
-rw-r--r-- 1 root root 598494 Apr 23  2020 characters_illustration.png
drwx------ 5 root root   4096 Dec 19 17:27 gdrive
-rw-r--r-- 1 root root   4288 Dec 19 17:39 label_encoder.pkl
drwxr-xr-x 1 root root   4096 Dec  3 14:33 sample_data
-rw-r--r-- 1 root root  23686 Apr 23  2020 sample_submission.csv
drwxr-xr-x 3 root root   4096 Dec 19 17:37 testset
drwxr-xr-x 3 root root   4096 Dec 19 17:37 train
```

Ввод [46]:

```
import pandas as pd
my_submit = pd.read_csv("/content/sample_submission.csv")
my_submit = pd.DataFrame({'Id': test_filenames, 'Expected': preds})
my_submit.head()
```

Out[46]:

	Id	Expected
0	img0.jpg	nelson_muntz
1	img1.jpg	bart_simpson
2	img10.jpg	ned_flanders
3	img100.jpg	chief_wiggum
4	img101.jpg	apu_nahasapeemapetilon

Ввод [ ]:

```
# TODO : сделайте сабмит (это важно, если Вы не справляетесь, но дошли до этой ячейки, то с
```

Ввод [47]:

```
my_submit.to_csv('/content/gdrive/MyDrive/my_CNN_baseline.csv', index=False)
```

Приключение?

А теперь самое интересное, мы сделали простенькую сверточную сеть и смогли отправить сабмит, но получившийся скор нас явно не устраивает. Надо с этим что-то сделать.

Несколько срочныйх улучшейни для нашей сети, которые наверняка пришли Вам в голову:

- Учим дольше и изменяем гиперпараметры сети
- learning rate, batch size, нормализация картинки и вот это всё
- Кто же так строит нейронные сети? А где пулинги и батч нормы? Надо добавлять
- Ну разве Адам наше все? adamW (https://www.fast.ai/2018/07/02/adam-weight-decay/) для практика, статейка для любителей (https://openreview.net/pdf?id=ryQu7f-RZ) (очень хороший анализ), наши (https://github.com/MichaelKonobeev/adashift/) эксперименты для заинтересованных.

- Ну разве это deep learning? Вот ResNet и Inception, которые можно зафайнтьюнить под наши данные, вот это я понимаю (можно и обучить в колабе, а можно и [готовые (https://github.com/Cadene/pretrained-models.pytorch)](https://github.com/Cadene/pretrained-models.pytorch) скачать).
- Данных не очень много, можно их аугументировать и доучититься на новом датасете (который уже будет состоять из, как пример аугументации, перевернутых изображений)
- Стоит подумать об ансамблях

Надеюсь, что у Вас получится!