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

In [1]:

```
# ВНИМАНИЕ; версия Pillow установввленна по умолчанию - '7.0.0'
# Эта версия ломает torchvision
import PIL
PIL.__version__
```

Out[1]:

```
'5.3.0'
```

In [ ]:

```
!pip uninstall -y Pillow
!pip install -Iv Pillow==5.3.0
```

ВНИМАНИЕ: На этом этапе обязательно перезагрузить рантайм (Runtime -> Restart Runtime). Иначе версия 5.3.0 не будет загружена в рантайм.

После перезапуска рантайма следует продолжить исполнять ячейки далее.

In [ ]:

```
!pip install -U torch torchvision
```

In [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 ...
```

In [2]:

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

```
Fri May  8 02:25:33 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.59       Driver Version: 440.59       CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce MX150       Off  | 00000000:01:00.0 Off |                  N/A |
| N/A   43C    P0    N/A /  N/A |    464MiB /  2002MiB |     12%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      1383      G   /usr/lib/xorg/Xorg                            28MiB |
|    0      2082      G   /usr/lib/xorg/Xorg                           155MiB |
|    0      2343      G   /usr/bin/gnome-shell                          95MiB |
|    0      4196      G   ...AAAAAAAAAAACAAAAAAAAAA= --shared-files   126MiB |
+-----------------------------------------------------------------------------+
```

Out[2]:

```
True
```

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

In [3]:

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

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

import torchvision
from torchvision import transforms
from multiprocessing.pool import ThreadPool
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import f1_score
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler
import torch.nn as nn
import copy

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)
```

```
/home/philipp/anaconda3/lib/python3.7/site-packages/sklearn/utils/__init__.py:4: DeprecationWarning: Using
or importing the ABCs from 'collections' instead of from 'collections.abc' is deprecated, and in 3.8 it wil
l stop working
  from collections import Sequence
```

In [4]:

```
# 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)

In [5]:

```
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])
        ])
        # аугментация трейна для увеличения трейн выборки

        augmentation = transforms.Compose([
            transforms.RandomHorizontalFlip(),
            transforms.RandomResizedCrop(224, scale=(0.8, 1), ratio=(0.75, 1.3333333333333333), interpolation=2),
            transforms.RandomPerspective(distortion_scale=0.3, p=0.9, interpolation=3, fill=0),
            transforms.RandomAffine(degrees=20, shear=20, resample=False),
            transforms.Resize((224, 224))
        ])

        x = self.load_sample(self.files[index])
        if self.mode == 'train':
            x = augmentation(x)
        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)
```

In [6]:

```
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)
```

Ниже необходимо поменять пусть к train и test данным.

In [52]:

```
TRAIN_DIR = Path('/home/philipp/Projects/simpson classification/journey-springfield/train')
TEST_DIR = Path('/home/philipp/Projects/simpson classification/journey-springfield/testset')

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

In [8]:

```
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)
```

In [9]:

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

Давайте посмотрим на наших героев внутри датасета.

In [10]:

```
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)
```

Видно, что аугментация работает, когда мы берем изображение из трейна оно проходит ряд рандомных трансформаций, ха счет этого в каждой эпохе мы получаем разные изображения, трейновая выборка увеличивается.

Построение нейросети

Сначала обьявим необходимые функции для обучения модели. За основу взяты функции из ноутбука с бейслайном с моими преобразованиями.

Так как выборка несбалансированная, реализуем WeightedRandomSampler. Чем больше обьектов в классе, тем меньше вес класса.

Описание эпохи обучения.

In [11]:

```
def fit_epoch(model, train_loader, criterion, optimizer):
    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()
        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
```

Описание эпохи валидации.

In [12]:

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

    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)

        val_preds += preds.tolist()
        val_labels += labels.tolist()
        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
    val_f1 = f1_score(val_labels, val_preds, average='macro')
    return val_loss, val_acc, val_f1
```

In [13]:

```
def train(train_dataset, val_dataset, model, epochs, batch_size, opt, criterion, scheduler):
#     увеличиваю num_workers для того, чтобы данные обрабатывались многопоточно и обучение происходило быстрее
    train_loader = DataLoader(train_dataset, batch_size=batch_size, num_workers=4, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=batch_size, num_workers=4, shuffle=False)

    start_time = time.time()
    history = []
    history_f1 = []
    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} val_f1 {v_f1:0.4f}"


    with tqdm(desc="epoch", total=epochs) as pbar_outer:

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

            val_loss, val_acc, val_f1 = eval_epoch(model, val_loader, criterion)
            history.append((train_loss, train_acc, val_loss, val_acc))
            history_f1.append(val_f1)
#             как и описано ниже, в sheduler.step передается метрика, и вызывается он после валидации
            scheduler.step(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, v_f1=val_f1))

    end_time = time.time()

    print('total time:',end_time-start_time)
    print('average time per epoch:',(end_time-start_time)/epochs)
    return history, history_f1, best_f1
```

In [14]:

```
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
```

Будем использрвать предобученную сеть ResNet на ImageNet, потому что размер выборки невелик. С помощью feature extractor не удалось получить высоких результатов, поэтому будем применять метод fine tuning, с разморокой последнего сверточного слоя.

In [15]:

```
from torchvision import models
```

Команды, которые я нашел в какой-то статье, которые увеличивают скорость обучения, но теряется воспроизводимость:

In [16]:

```
torch.backends.cudnn.enabled = True
torch.backends.cudnn.benchmark =  True
torch.backends.cudnn.deterministic = False
```

In [17]:

```
model_resnet_ft = models.resnet18(pretrained=True).cuda()
```

Посмотрим на архитектуру сети:

In [18]:

```
print(model_resnet_ft)
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
```

```
    (layer3): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (layer4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
```

```
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

Заморозим 3 сверточных слоя и обучим на остальном, заменим полносвязный слой на свой, с количеством выходов, равным количеству классов.

In [19]:

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

```
we will classify :42
```

In [20]:

```
for param in model_resnet_ft.parameters():
    param.requires_grad = False
```

In [21]:

```
model_resnet_ft.fc = nn.Linear(in_features=512, out_features=n_classes, bias=True).cuda()
```

In [22]:

```
for param in model_resnet_ft.layer4.parameters():
    param.requires_grad = True
for param in model_resnet_ft.fc.parameters():
    param.requires_grad = True
```

Обьявим необходимые параметры для обучения. Обращу внимание на шедулер, который уменьшает скорость обучение, когда выбранная метрика перестает улучшаться, поэтому в sheduler.step передается параметр metrics, в данном случае accuracy на валидации.

In [23]:

```
train_dataset = SimpsonsDataset(train_files, "train")
val_dataset = SimpsonsDataset(val_files, "val")
opt = torch.optim.AdamW(model_resnet_ft.parameters())
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(opt, mode='max', factor=0.1, patience=3, verbose=False)
criterion = nn.CrossEntropyLoss()
```

Почистим мусор в памяти, в том числе и в GPU.

In [26]:

```
import gc
gc.collect()
```

Out[26]:

47

Обучим на 30 эпохах:

In [27]:

```
history, history_f1, best_f1 = train(train_dataset, val_dataset, model=model_resnet_ft, best_model=best_model, epochs=30, \
                                     batch_size=64, opt=opt, criterion=criterion, scheduler=scheduler)
```

```
epoch:   0%|          | 0/30 [00:00<?, ?it/s]

loss 0.762524320860905

/home/philipp/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetric
Warning: F-score is ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
epoch:   3%|▎         | 1/30 [02:09<1:02:42, 129.74s/it]

Epoch 001 train_loss: 0.7625     val_loss 0.4536 train_acc 0.8036 val_acc 0.8731 val_f1 0.5783
loss 0.5788387969801613

epoch:   7%|▋         | 2/30 [04:12<59:37, 127.77s/it]

Epoch 002 train_loss: 0.5788     val_loss 0.4580 train_acc 0.8478 val_acc 0.8750 val_f1 0.6431
loss 0.3657207142563335

epoch:  10%|█         | 3/30 [06:15<56:44, 126.08s/it]

Epoch 003 train_loss: 0.3657     val_loss 0.3502 train_acc 0.9013 val_acc 0.9085 val_f1 0.7329
loss 0.3400301751349943

epoch:  13%|█▎        | 4/30 [08:17<54:07, 124.91s/it]

Epoch 004 train_loss: 0.3400     val_loss 0.3261 train_acc 0.9081 val_acc 0.9091 val_f1 0.7221
loss 0.27463041913779696

epoch:  17%|█▋        | 5/30 [10:19<51:42, 124.12s/it]

Epoch 005 train_loss: 0.2746     val_loss 0.3579 train_acc 0.9256 val_acc 0.9102 val_f1 0.7471
loss 0.24667037579540813

epoch:  20%|██        | 6/30 [12:21<49:25, 123.56s/it]

Epoch 006 train_loss: 0.2467     val_loss 0.3007 train_acc 0.9320 val_acc 0.9240 val_f1 0.8255
loss 0.23024864496698927

epoch:  23%|██▎       | 7/30 [14:24<47:13, 123.18s/it]

Epoch 007 train_loss: 0.2302     val_loss 0.2699 train_acc 0.9378 val_acc 0.9278 val_f1 0.7930
loss 0.20781358748036544

epoch:  27%|██▋       | 8/30 [16:26<45:04, 122.91s/it]
```

```
Epoch 008 train_loss: 0.2078     val_loss 0.3014 train_acc 0.9447 val_acc 0.9228 val_f1 0.7810
loss 0.19693381196139811

epoch:  30%|███       | 9/30 [18:28<42:57, 122.73s/it]

Epoch 009 train_loss: 0.1969     val_loss 0.2621 train_acc 0.9470 val_acc 0.9345 val_f1 0.8170
loss 0.17269890459033818

epoch:  33%|███       | 10/30 [20:30<40:52, 122.60s/it]

Epoch 010 train_loss: 0.1727     val_loss 0.2581 train_acc 0.9505 val_acc 0.9392 val_f1 0.8397
loss 0.16600289379106944

epoch:  37%|███       | 11/30 [22:33<38:49, 122.60s/it]

Epoch 011 train_loss: 0.1660     val_loss 0.2644 train_acc 0.9539 val_acc 0.9352 val_f1 0.8480
loss 0.16761836027097107

epoch:  40%|████      | 12/30 [24:35<36:45, 122.53s/it]

Epoch 012 train_loss: 0.1676     val_loss 0.3108 train_acc 0.9535 val_acc 0.9272 val_f1 0.7939
loss 0.13986763184368933

epoch:  43%|████      | 13/30 [26:38<34:41, 122.45s/it]


Epoch 013 train_loss: 0.1399     val_loss 0.2686 train_acc 0.9604 val_acc 0.9379 val_f1 0.8456
loss 0.13115520904121736

epoch:  47%|████      | 14/30 [28:40<32:38, 122.40s/it]

Epoch 014 train_loss: 0.1312     val_loss 0.2725 train_acc 0.9621 val_acc 0.9455 val_f1 0.8664
loss 0.13826043727944798

epoch:  50%|█████     | 15/30 [30:42<30:35, 122.37s/it]

Epoch 015 train_loss: 0.1383     val_loss 0.2586 train_acc 0.9624 val_acc 0.9329 val_f1 0.8498
loss 0.11404709977453337

epoch:  53%|█████     | 16/30 [32:45<28:34, 122.48s/it]

Epoch 016 train_loss: 0.1140     val_loss 0.2456 train_acc 0.9687 val_acc 0.9442 val_f1 0.8704
loss 0.11106675847746915

epoch:  57%|██████    | 17/30 [34:47<26:31, 122.42s/it]
```

```
Epoch 017 train_loss: 0.1111     val_loss 0.2698 train_acc 0.9684 val_acc 0.9377 val_f1 0.8300
loss 0.10336261829742771

epoch:  60%|██████    | 18/30 [36:50<24:28, 122.37s/it]

Epoch 018 train_loss: 0.1034     val_loss 0.2727 train_acc 0.9691 val_acc 0.9377 val_f1 0.8515
loss 0.054609456543130694

epoch:  63%|██████    | 19/30 [38:52<22:25, 122.34s/it]

Epoch 019 train_loss: 0.0546     val_loss 0.2023 train_acc 0.9845 val_acc 0.9578 val_f1 0.8909
loss 0.03935330710295215

epoch:  67%|██████    | 20/30 [40:54<20:23, 122.38s/it]

Epoch 020 train_loss: 0.0394     val_loss 0.2036 train_acc 0.9890 val_acc 0.9587 val_f1 0.9058
loss 0.03115538232295678

epoch:  70%|███████   | 21/30 [42:57<18:21, 122.35s/it]

Epoch 021 train_loss: 0.0312     val_loss 0.2151 train_acc 0.9906 val_acc 0.9574 val_f1 0.8957
loss 0.027616971092928287

epoch:  73%|███████   | 22/30 [44:59<16:18, 122.33s/it]

Epoch 022 train_loss: 0.0276     val_loss 0.2147 train_acc 0.9908 val_acc 0.9583 val_f1 0.9089
loss 0.02763536022810528

epoch:  77%|███████   | 23/30 [47:01<14:16, 122.36s/it]

Epoch 023 train_loss: 0.0276     val_loss 0.2086 train_acc 0.9915 val_acc 0.9599 val_f1 0.9059
loss 0.022488450592345655

epoch:  80%|████████  | 24/30 [49:04<12:14, 122.35s/it]

Epoch 024 train_loss: 0.0225     val_loss 0.2202 train_acc 0.9932 val_acc 0.9591 val_f1 0.9008
loss 0.021140766558400217

epoch:  83%|████████  | 25/30 [51:06<10:11, 122.39s/it]

Epoch 025 train_loss: 0.0211     val_loss 0.2260 train_acc 0.9927 val_acc 0.9601 val_f1 0.9099
loss 0.022584688649040313

epoch:  87%|████████  | 26/30 [53:08<08:09, 122.36s/it]
```

```
Epoch 026 train_loss: 0.0226     val_loss 0.2346 train_acc 0.9924 val_acc 0.9553 val_f1 0.8602
loss 0.020074938883395413

epoch:  90%|████████▉ | 27/30 [55:11<06:07, 122.36s/it]

Epoch 027 train_loss: 0.0201     val_loss 0.2263 train_acc 0.9939 val_acc 0.9585 val_f1 0.9036
loss 0.017605773764047556

epoch:  93%|█████████▎| 28/30 [57:13<04:04, 122.37s/it]

Epoch 028 train_loss: 0.0176     val_loss 0.2236 train_acc 0.9942 val_acc 0.9616 val_f1 0.9058
loss 0.020706436624021853

epoch:  97%|█████████▋| 29/30 [59:16<02:02, 122.51s/it]

Epoch 029 train_loss: 0.0207     val_loss 0.2384 train_acc 0.9939 val_acc 0.9593 val_f1 0.8943
loss 0.017673553186712602

epoch: 100%|██████████| 30/30 [1:01:19<00:00, 122.66s/it]

Epoch 030 train_loss: 0.0177     val_loss 0.2362 train_acc 0.9946 val_acc 0.9605 val_f1 0.9057
total time: 3679.926388025284
average time per epoch: 122.66421293417612
```

Сохраним веса, чтобы не потерять полученный результат.

In [29]:

```
torch.save(model_resnet_ft.state_dict(), "/home/philipp/Projects/simpson classification/resnet__weights.pth")
```

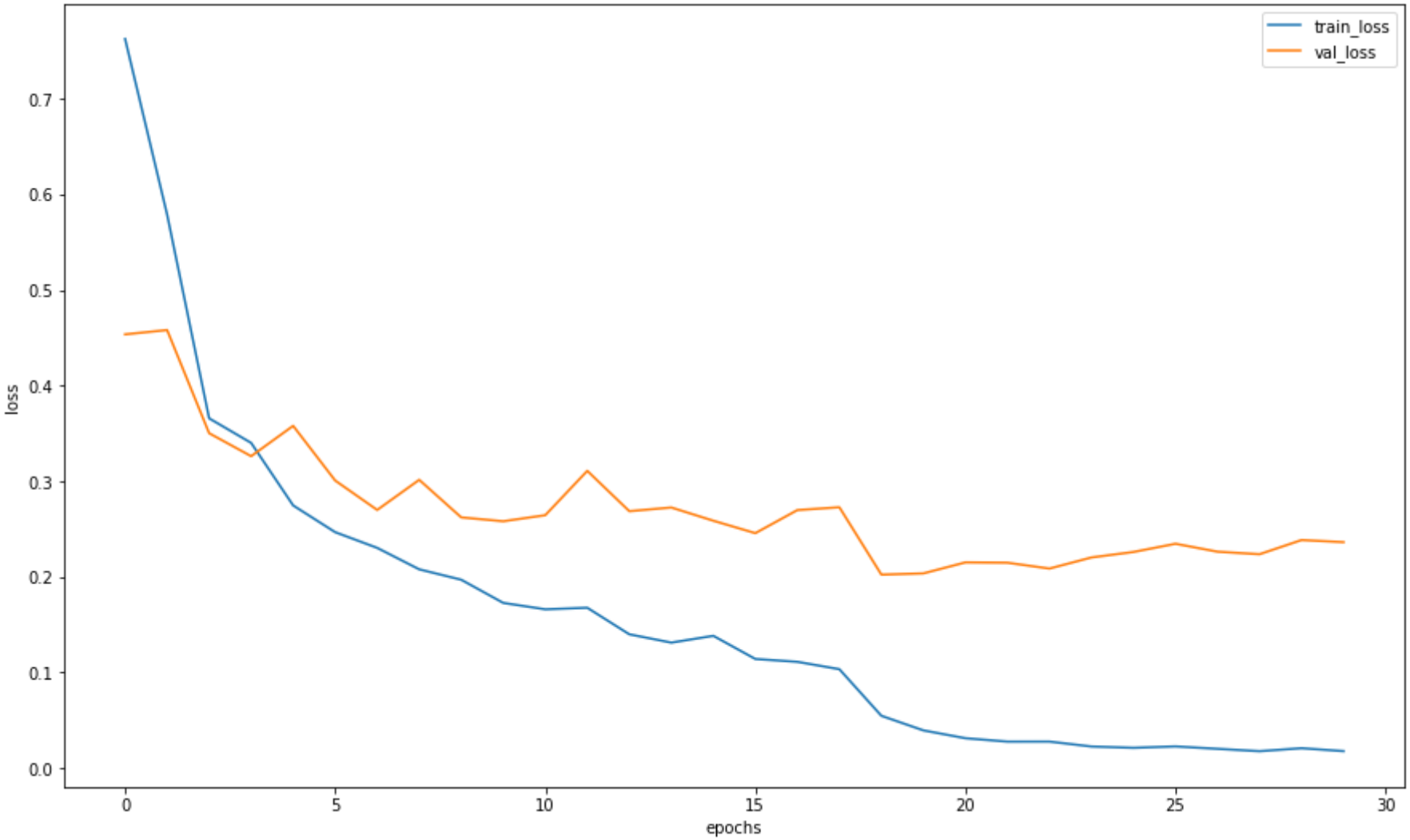
Изобразим результат.

In [30]:

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

In [31]:

```
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()
```


train_loss
val_loss
loss
epochs
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
5
10
15
20
25
30

In [ ]:

```
plt.figure(figsize=(15, 9))
plt.plot(acc, label="train_acc")
plt.plot(val_acc, label="val_acc")
plt.legend(loc='best')
plt.xlabel("epochs")
plt.ylabel("acc")
plt.show()
```

Сабмит на kaggle:

In [55]:

```
test_dataset = SimpsonsDataset(test_files, mode="test")
test_loader = DataLoader(test_dataset, shuffle=False, batch_size=8)
probs = predict(model_resnet_ft, test_loader)

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

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

In [57]:

```
import pandas as pd
df = pd.DataFrame()
df['Id'] = test_filenames
df['Expected'] = preds
df.to_csv('/home/philipp/Projects/simpson classification/model_resnet18.csv', index=False)
```

In [35]:

```
def preIdict_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
```

In [36]:

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

In [37]:

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

probs_ims = predict(model_resnet_ft, imgs)
```

In [38]:

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

In [39]:

```
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]
```

Сделаем классную визуализацию, чтобы посмотреть насколько сеть уверена в своих ответах.

In [40]:

```
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(model_resnet_ft, 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[len(predicted_label)//2:]
    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')
```

```
findfont: Font family ['fantasy'] not found. Falling back to DejaVu Sans.
```


Charles Montgomery Burns
charles_mont
gomery_burns : 100%
Charles Montgomery Burns
charles_mont
gomery_burns : 100%
Bart Simpson
bart_s
impson : 100%
Bart Simpson
bart_s
impson : 100%
Bart Simpson
bart_s
impson : 100%
Abraham Grampa Simpson
abraham_gra
mpa_simpson : 100%
Producteur Délégué
DAVID M. STERN
Bart Simpson
bart_s
impson : 100%
Charles Montgomery Burns
charles_mont
gomery_burns : 100%
Bart Simpson
bart_s
impson : 100%