Домашнее задание. Классификация изображений

Сегодня вам предстоить помочь телекомпании FOX в обработке их контента. Как вы знаете, сериал "Симпсоны" идет на телеэкранах более 25 лет, и за это время скопилось очень много видеоматериала. Персоонажи менялись вместе с изменяющимися графическими технологиями, и Гомер Симпсон-2018 не очень похож на Гомера Симпсона-1989. В этом задании вам необходимо классифицировать персонажей, проживающих в Спрингфилде. Думаю, нет смысла представлять каждого из них в отдельности.

Общее описание решения

При анализе данных, первое что бросается в глаза это дисбаланс классов. Первое желание было сгенерировать с помощью аугментации доподнительные данные используя популярные библиотеки (например albumentations). Идея была в увеличении самого датасета за счет генерации аугментированных изображений таким образом, чтобы в тренировочном датасете для каждого класса было не менее например по 1000 примеров. Однако при дальнейшем изучении вопроса вяснилось, что подход в корне другой:

1. Подсчитываем кол-во данных для каждого класса через Counter
2. Определяем для каждого класса его вес (1/кол-во сэмплов класса). Используя WeightedRandomSampler формируем сэмплер, который будет выдавать уравновешенное распределение данных по классам.
3. Чтобы разнообразить данные по "редким" классам (где мало данных) испоьлзуем аугментацию. По сути аугментация и трансформация делается на лету, т.е. при использовании даталоудера запрашиваем данные согласно нашему сэмплеру (грубо говоря по правилам которые опредлены сэмплере).

Плюс такого подхода в простоте реализации. Минус в том что аугментация на лету делается для всех классов, в том числе для которых достаточный объем обучающих данных, по хорошему для таких классов надо использовать оригинальные данные без аугментации.

Дополнительно относительно базовой модели добавлена батчнормализация в каждом слое

Импортируем необходимые библиотеки

In [28]:
```
from multiprocessing.pool import ThreadPool
from pathlib import Path

import numpy as np
import pandas as pd
from collections import Counter

import PIL
from PIL import Image
import pickle

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, TensorDataset
from torch.utils.data.sampler import WeightedRandomSampler
import torchvision
from torchvision import transforms


from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from skimage import io

from tqdm import tqdm, tqdm_notebook

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)
```

Определяем наличие поддержки GPU

In [2]:
```
# !nvidia-smi
torch.cuda.is_available()
train_on_gpu = torch.cuda.is_available()

if not train_on_gpu:
    # работаем на процессоре
    print('CUDA не доступна.  Работаем на CPU ...')
    DEVICE = torch.device("cpu")
else:
    # работаем на видеокарте
    print('CUDA доступна! Работаем на GPU ...')
    DEVICE = torch.device("cuda")
```

```
CUDA доступна! Работаем на GPU ...
```

In [3]:
```
# Для колаба монтируем диск, для локального запуска это необходмио закомментировать
```

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

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

Объявляем константы и пути

In [12]:
```
# разные режимы датасета
DATA_MODES = ['train', 'val', 'test']
# все изображения будут масштабированы к размеру 224x224 px
RESCALE_SIZE = 224
# PATH = "simpsons_datasets/"
PATH = "gdrive/MyDrive/DLS/simpsons/"
TRAIN_DIR = Path(PATH+'train_dataset')
TEST_DIR = Path(PATH+'test_dataset')
```

Загружаем данные

In [ ]:
```
# загрузка данных
train_val_files = sorted(list(TRAIN_DIR.rglob('*.jpg')))
test_files = sorted(list(TEST_DIR.rglob('*.jpg')))
```

Определяем класс SimpsonsDataset (наследуем от Dataset). В классе SimpsonsDataset определяем необходимые процедуры по загрузке и предобработке данных. Аугментация производится именно здесь для каждого запрошенного объекта.

In [14]:
```
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
        if self.mode == "train":
            transform = transforms.Compose([
                transforms.ToTensor(),
                transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
                transforms.Compose([
                    transforms.Resize(size=300, max_size=401),
                    transforms.CenterCrop(size=300),
                    transforms.RandomCrop(250),
                ]),
                transforms.RandomRotation(degrees=(-30, 30)),
                transforms.RandomHorizontalFlip(p=0.5),
            ])
        else:
            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)
```

Определяем методы по визуализации данных. Обычный метод визуализации загруженных изображений и визуализации изображений в виде тензоров

In [ ]:
```
def visualize(image):
    """ Метод визуализации изображения"""
    plt.figure()
    plt.imshow(image)
    return plt

def visualize_tensor(inp):
    """
    Метод визуализации для тензоров
    При переводе в тензор каналы цветов перемещены на первую позицию,
    а при обычной загрузке каналы цветов расположены в конце.
    Поэтому перед визуализацией делается доп. преобразование
    """
    inp = inp.numpy().transpose((1, 2, 0))
    plt.figure()
    plt.imshow(inp)

def imshow(inp, title=None, plt_ax=plt, default=False):
    """Imshow метод визуализации для тензоров с нормализацией. Метод взят из бейзлайна"""
    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.figure()
    plt_ax.imshow(inp)
    plt.rc('font', size=10)
    if title is not None:
        plt_ax.set_title(title)
    plt_ax.grid(False)


image = Image.open(train_val_files[2000])
image.load()

visualize(image)
visualize_tensor(transforms.ToTensor()(image))
imshow(transforms.ToTensor()(image))
```

Подготовливаем данные

Из тренировочных данных выделяем валидационные данные

In [16]:
```
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 [17]:
```
val_dataset = SimpsonsDataset(val_files, mode='val')
train_dataset = SimpsonsDataset(train_files, mode='train')
```

Проверка загруженных данных

In [18]:
```
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)
```


```
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
```

Построение модели

Сама модель:

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

На каждом конв-слое добавлена батчнормализация. В качестве активации используется LeakyReLU

In [19]:
```python
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.BatchNorm2d(8),
            nn.LeakyReLU(),
            nn.MaxPool2d(kernel_size=2)
        )

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

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

        self.conv4 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(),
            nn.MaxPool2d(kernel_size=2)
        )

        self.conv5 = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=96, kernel_size=3),
            nn.BatchNorm2d(96),
            nn.LeakyReLU(),
```

```
            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
```

Обучение модели на тренировочных данных

In [20]:
```
def fit_epoch(model, train_loader, criterion, optimizer):
    running_loss = 0.0
    running_corrects = 0
    processed_data = 0

    for inputs, labels in tqdm(train_loader, position = 0, leave=True):
        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
```

Прогон модели на валидационных данных

In [21]:
```
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
```

С целью борьбы с дисбалансом классов изменяется сэмплер DataLoader.

Метод для формивраония правила выборки данных (сэмплеров) на основе WeightedRandomSampler

In [22]:
```
# Формируем распределение данных для выгрузки из DataLoader сбалансированных данных
def create_sampler(ds):
    print(len(ds))
    class_count = Counter(ds.labels)
    class_weights = {name: 1/cnt for name, cnt in class_count.items()}

    sample_weights = [0] * len(ds)
    for i, label in enumerate(ds.labels):
        class_weight = class_weights[label]
        sample_weights[i] = class_weight
    # Максимальное кол-во объкетов = максимальное кол-во записей среди всех классов в трейне * на кол-во классов
    N_count = max(class_count.values()) * len(class_count)

    sampler = WeightedRandomSampler(sample_weights, num_samples=N_count, replacement=True)
    return sampler
```

Главный цикл обучения по эпохам

In [23]:
```
def train(train_dataset, val_dataset, 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)
    train_loader = DataLoader(train_dataset, batch_size=batch_size, sampler=create_sampler(train_dataset))
    val_loader = DataLoader(val_dataset, batch_size=batch_size, sampler=create_sampler(val_dataset))

    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())
        criterion = nn.CrossEntropyLoss()

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

Предиктовый метод для предсказаний по тестовым данным

In [24]:
```
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
```

Итоговая структура модели

In [25]:
```
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): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): 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): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): 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): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv4): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv5): Sequential(
    (0): Conv2d(64, 96, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (out): Linear(in_features=2400, out_features=42, bias=True)
)
```

Обучение модели

In [ ]:
```
history = train(train_dataset, val_dataset, model=simple_cnn, epochs=5, batch_size=64)
```

```
15699
5234
```

```
27%|██▋       | 298/1106 [48:39<1:15:13,  5.59s/it]
```

In [ ]:
```
loss, acc, val_loss, val_acc = zip(*history)
```

Отображаем графики ошибки и валиадции

In [ ]:
```
plt.figure(figsize=(5, 5))
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()


plt.figure(figsize=(5, 5))
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()
```

Формируем прогноз по тестовым данным

In [ ]:
```
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
```

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

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

probs_ims = predict(simple_cnn, imgs)
```

In [ ]:
```
label_encoder = pickle.load(open("label_encoder.pkl", 'rb'))
```

In [ ]:
```
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]
```

In [ ]:
```
len(y_pred)
```

Out[ ]:
```
20
```

Проверяем получившийся скор

In [ ]:
```
from sklearn.metrics import f1_score

# f1_score(actual_labels, preds_class)
f1_score(actual_labels, list(y_pred), average='micro')
```

Выводим примеры тестовых данных и получившихся прогнозов

In [ ]:
```
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(simple_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[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')
```

In [ ]:
```
test_dataset = SimpsonsDataset(test_files, mode="test")
test_loader = DataLoader(test_dataset, shuffle=False, batch_size=64)
probs = predict(simple_cnn, test_loader)


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

Формируем итоговый файл с прогнозом

In [ ]:
```
import pandas as pd
submission_path = PATH + 'sample_submission.csv'
my_submit = pd.read_csv(submission_path)
my_submit.head()
```

Out[ ]:

	Id	Expected
0	img0.jpg	bart_simpson
1	img1.jpg	bart_simpson
2	img2.jpg	bart_simpson
3	img3.jpg	bart_simpson
4	img4.jpg	bart_simpson

In [ ]:
```
my_submit = pd.DataFrame({'Id': test_filenames, 'Expected': preds})
```

In [ ]:
```
my_submit.to_csv(PATH + 'my_submission.csv', index=False)
my_submit.head()
```

Out[ ]:

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

In [ ]:
```

```