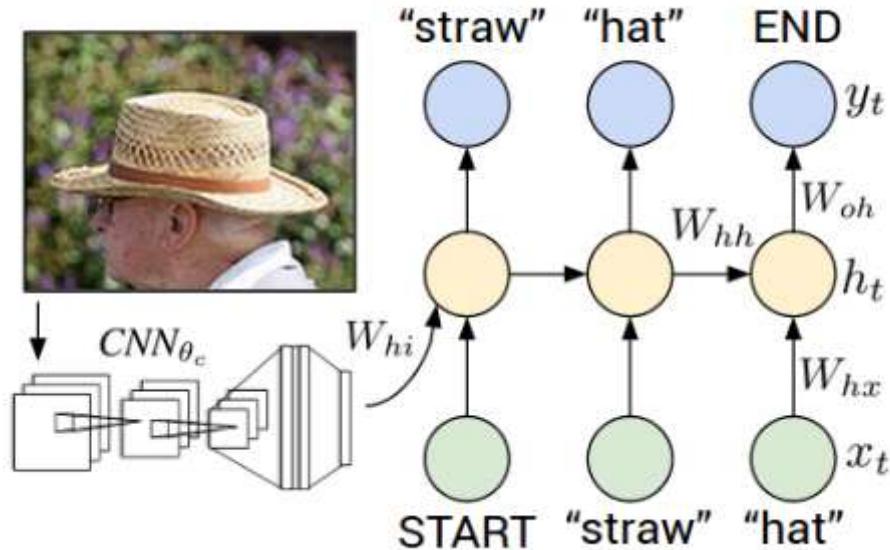


Image Captioning

Итак, мы сейчас займемся image captioning =)



Описание

Image captioning -- это когда мы подаем модели картинку, а она возвращает нам текст с описанием того, что на ней изображено.

Как мы знаем из нашего замечательного курса, с картинками лучше всегоработают модели CNN, а с текстом -- RNN. Поэтому логично, что для image captioning нужно совместить и то, и другое =)

Для удобства (и, в какой-то степени, экономии времени), мы будем строить не одну большую модель CNN+RNN, которая будет кушать картинку и выдавать текст, а разобьем ее на две. Первая модель будет кушать картинку и выдавать вектор картинки, а вторая модель будет кушать этот вектор и генерировать текст. Вектор, по сути, будет числовым "описанием" картинки, в котором будет содержаться вся необходимая информация для второй сети, чтобы та смогла нагенерить текста с описанием. Короче, как в автоэнкодерах)

▼ План

Итак, как мы будем действовать:

Датасет: MSCOCO: [описание, ссылка для скачивания](#)

Базовая часть:

1. Скачаем датасет (векторы картинок и соответствующие описания) и предобработаем описания так, как мы любим. Ну, токенизация там (да, в 100500-ый раз, только теперь сами)
2. В качестве первой сети возьмем Inception-v3 и скачаем к ней предобученные веса (тренировать и генерировать веса -- это оч долго, поверьте мне).
3. Напишем вторую сетку, которая будет брать векторы из Inception-v3 и генерить описания.
4. Обучим вторую сеть на MSCOCO

Вариативная часть:

Что еще можно сделать:

1. Нагуглить другой датасет (в MSCOCO видны паттерны -- все тексты выглядят как "что-то с чем-то что-то делает")
2. Взять не Inception-v3, а другую предобученную сеть
3. Запилить аттеншен во второй сети (не, ну а вдруг)
4. Написать бота))0))
5. Whatever comes to your head

▼ Базовая часть:

▼ 1. Предобработка текстов из датасета

```
!pip install opencv-python
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.7/dist-packages  
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (
```

```
DATA_PATH = ''  
%matplotlib inline  
  
# For Google Colab only:  
# import sys  
# sys.path.append('/content/gdrive/My Drive/Colab Notebooks')  
#from reco_utils.recommender.sar.sar_singlenode import SARSingleNode  
from google.colab import drive  
drive.mount('/content/gdrive/', force_remount=True)
```

Mounted at /content/gdrive/

```
!cp '/content/gdrive/MyDrive/Image Captioning/beheaded_inception3.py' ./
!cp '/content/gdrive/MyDrive/Image Captioning/handout.tar' ./
!tar -xf /content/handout.tar

import numpy as np
import json

# загружаем датасет
img_codes = np.load("./data/image_codes.npy")
captions = json.load(open('./data/captions_tokenized.json'))

# посмотрим на датасет
print("Each image code is a 2048-unit vector [ shape: %s ]" % str(img_codes.shape))
print(img_codes[0,:10], end='\n\n')
print("For each image there are 5 reference captions, e.g.: \n")
print('\n'.join(captions[0]))
```

Each image code is a 2048-unit vector [shape: (118287, 2048)]
[0.3659946 0.2016555 0.9245725 0.57063824 0.547268 0.8275868
 0.3687277 0.12085301 0.0561931 0.49758485]

For each image there are 5 reference captions, e.g.:

people shopping in an open market for vegetables .
an open market full of people and piles of vegetables .
people are shopping at an open air produce market .
large piles of carrots and potatoes at a crowded outdoor market .
people shop for vegetables like carrots and potatoes at an open air market .

Как можно видеть, в датасете все captions (тексты-описания) уже токенизированы и приведены в нижний регистр. Нам осталось сделать следующее:

1. Добавить ко всем описаниям символы начала и конца предложения
2. Посчитать частоту встречания каждого слова из словаря и оставить только те, которые встречаются больше X раз (например, X=5)
3. Создать словарь из оставшихся слов + символов начала, конца предложения и PAD символа
4. Написать функцию, которая будет возвращать батч из описаний. Мы такое уже делали на прошлых занятиях. Батч должен выглядеть примерно так: ВАЖНО! Почему я советую писать отдельную функцию, которая генерирует батч: дело в том, что в датасете для каждой картинки есть несколько (5-7) различных описаний. Когда создаете батч, лучше, чтобы в нем были разные картинки, и к каждой картинке при

создании батча выбирать одно из ее описаний рандомно. Это проще реализовать в отдельной функции (но вы, конечно, можете писать код как хотите)

5. Поделить выборку на train/test

```
[[ 1, 525, 8955, 5392, 9640, 4713, 7470, 525, 7341, 2296, 7696, 2, 3, 3, 3, 3, 3, 3, 3, 3],  
[ 1, 525, 8955, 6784, 3557, 525, 7341, 2296, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],  
[ 1, 525, 8955, 9209, 3557, 5486, 8335, 3071, 2296, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],  
[ 1, 6292, 1508, 8955, 9209, 6784, 3557, 3071, 6971, 5520, 7696, 2, 3, 3, 3, 3, 3, 3, 3, 3],  
[ 1, 525, 8955, 6784, 3557, 525, 7341, 6919, 2919, 6292, 250, 393, 525, 4618, 8335, 6292, 7882, 7696,  
2]]
```

То есть, короткие предложения дополняются PAD символами, слишком длинные обрезаются, в начале и конце по коду символа начал и конца предложения.

Уверена, эта часть вам покажется очень знакомой и легкой =)

```
import copy
START = "<START>"
END = "<END>"
PAD = "<PAD>"
UNK = "<UNK>"
captions_processed = copy.deepcopy(captions)
for imgCaptions in range(len(captions)):
    for caption in range(len(captions[imgCaptions])):
        sentence = captions[imgCaptions][caption]
        captions_processed[imgCaptions][caption] = [START]+sentence.split(' ')+[END]

from collections import Counter
wordCounts = Counter()

for imgCaptions in range(len(captions_processed)):
    for caption in range(len(captions_processed[imgCaptions])):
        for word in captions_processed[imgCaptions][caption][1:-1]:
            wordCounts[word] += 1

vocab = []
vocab = [k for k, v in wordCounts.items() if v >= 5 if k not in vocab]
vocab += [UNK, START, END, PAD]
tokensLength = len(vocab)

wordToIndex = {w: i for i, w in enumerate(vocab)}

len(vocab)
```

10403

```

eos_ix = word_to_index[END]
unk_ix = word_to_index[UNK]
pad_ix = word_to_index[PAD]

def as_matrix(sequences, max_len=None):
    """ Convert a list of tokens into a matrix with padding """
    max_len = max_len or max(map(len, sequences))

    matrix = np.zeros((len(sequences), max_len), dtype='int32') + pad_ix
    for i, seq in enumerate(sequences):
        row_ix = [word_to_index.get(word, unk_ix) for word in seq[:max_len]]
        matrix[i, :len(row_ix)] = row_ix

    return matrix

as_matrix(captions_processed[2545])

array([[10400,     20,    117,    118,     54,     20,    439,   3305,   1658,
       10401, 10402, 10402, 10402, 10402],
       [10400,     20,    127,     43,    117,     54,     20,   4806,    326,
       118,      8, 10401, 10402, 10402],
       [10400,  3698,     11,    769,    118,    137,     10,     20,    117,
       10401, 10402, 10402, 10402, 10402],
       [10400,     20,     43,    118,    136,      2,     20,    117,    370,
       20,  3601,     98,      8, 10401],
       [10400,     41,    117,     37,    769,     11,    620,    113,    512,
       1853,      8, 10401, 10402, 10402]], dtype=int32)

from random import choice
import torch, torch.nn as nn
import torch.nn.functional as F

def generate_batch(img_codes, captions, batch_size, max_caption_len=None):

    random_image_ix = np.random.randint(0, len(img_codes), size=batch_size)
    batch_images = img_codes[random_image_ix]
    captions_for_batch_images = captions[random_image_ix]

    batch_captions = list(map(choice, captions_for_batch_images))
    batch_captions_ix = as_matrix(batch_captions, max_len=max_caption_len)

    return torch.tensor(batch_images, dtype=torch.float32), torch.tensor(batch_captions_ix, d

from sklearn.model_selection import train_test_split

captions = np.array(captions)
train_img_codes, val_img_codes, train_captions, val_captions = train_test_split(img_codes, ca

```

▼ 2. Напишем свою сеть из RNN для вывода описаний

Сейчас мы напишем сеть, которая будет получать выходы CNN-сетки (эмбеддинги картинок) и преобразовывать их в текст.

```
class CaptionNet(nn.Module):
    def __init__(self, cnn_feature_size=2048, lstm_feature_size=256, emb_size=128, tokens_size=1000):
        super(self.__class__, self).__init__()

        # стандартная архитектура такой сети такая:
        # 1. линейные слои для преобразования эмбеддинга картинки в начальные состояния h0 и c0
        self.cnn_h0 = nn.Linear(cnn_feature_size, lstm_feature_size)
        self.cnn_c0 = nn.Linear(cnn_feature_size, lstm_feature_size)

        # 2. слой эмбедднга
        self.emb = nn.Embedding(tokens_size, emb_size, padding_idx = pad_idx)

        # 3. несколько LSTM слоев (для начала не берите больше двух, чтобы долго не ждать)
        self.lstm = nn.LSTM(input_size=emb_size, hidden_size=lstm_feature_size, batch_first=True)

        # 4. линейный слой для получения логитов
        self.logits = nn.Linear(lstm_feature_size, tokens_size)

    def forward(self, image_vectors, captions_ix):
        """
        Apply the network in training mode.

        :param image_vectors: torch tensor, содержащий выходы inception. Те, из которых будем
        :param captions_ix:
        :return: логиты для сгенерированного текста описания, shape: [batch, word_i, n_tokens]
        """

        # 1. инициализируем LSTM state
        initial_cell = self.cnn_c0(image_vectors)
        initial_hid = self.cnn_h0(image_vectors)
        # 2. применим слой эмбеддингов к image_vectors
        captions_emb = self.emb(captions_ix)
        # 3. скормим LSTM captions_emb
        lstm_out, _ = self.lstm(captions_emb, (initial_cell.unsqueeze(0), initial_hid.unsqueeze(0)))
        # 4. посчитаем логиты из выхода LSTM
        logit = self.logits(lstm_out)

        return logit
```

```

def compute_loss(network, image_vectors, captions_ix):
    """
    :param image_vectors: torch tensor с выходами inception. shape: [batch, cnn_feature_size]
    :param captions_ix: torch tensor с описаниями (в виде матрицы). shape: [batch, word_i].
    :returns: scalar crossentropy loss (neg log likelihood) for next captions_ix given previous
    """

    image_vectors = image_vectors.to(device)
    captions_ix = captions_ix.to(device)

    # реализуйте стандартный cross entropy loss: итоговый лосс есть сумма лоссов для каждого
    captions_ix_inp = captions_ix[:, :-1].contiguous()
    captions_ix_next = captions_ix[:, 1:].contiguous()
    mask = captions_ix_next != pad_ix
    # 1. Получаем логиты, прогоняя image_vectors через сеть
    logits_for_next = network(image_vectors.to(device), captions_ix_inp)
    # 2. Вычисляем лосс-функцию между полученными логитами и captions_ix. Будьте внимательны:
    # вычисляйте лосс между логитами, полученными из сети, и соответствующими им значениями и
    next_reshaped = captions_ix_next.view(-1)
    logits_reshaped = logits_for_next.view(-1, logits_for_next.size()[-1])

    # ВАЖНО: не забудьте, что PADDING не должен влиять на лосс -- лосс должен складываться
    # только из тех мест, где должно быть предсказано слово, а не PAD
    # это можно сделать либо заведя маску из нулей и единиц (captions_ix_next != pad_ix) и умножив
    # либо просто используя ignore_index, который в торче есть как аргумент у некоторых лоссо
    cross_entropy = nn.CrossEntropyLoss(reduction = 'none').to(device)
    loss = cross_entropy(logits_reshaped, next_reshaped).view(captions_ix_next.size()[0], caption_length)

    return loss.mean(dim = -1).mean(dim = -1).view(-1)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
network = CaptionNet(tokens_size=tokens_length).to(device)

optimizer = torch.optim.Adam(network.parameters(), lr=1e-3) # favourite one

```

▼ Train it

Как обычно, пишем цикл тренировки, запоминаем лоссы для графиков и раз в X такстов тренировки считаем val_loss.

```

batch_size = 128
n_epochs = 200
n_batches_per_epoch = 300
n_validation_batches = 10

```


100% | ██████████ | 300/300 [00:30<00:00, 9.73it/s]

Epoch: 188, train loss: 0.3624765756726265, val loss: 0.35672990083694456
100% | ██████████ | 300/300 [00:30<00:00, 9.73it/s]

Epoch: 189, train loss: 0.3617238087952137, val loss: 0.35402107536792754
100% | ██████████ | 300/300 [00:30<00:00, 9.89it/s]

Epoch: 190, train loss: 0.36473326245943705, val loss: 0.36850764602422714
100% | ██████████ | 300/300 [00:30<00:00, 9.94it/s]

Epoch: 191, train loss: 0.36429423665006955, val loss: 0.39506266117095945
100% | ██████████ | 300/300 [00:29<00:00, 10.02it/s]

Epoch: 192, train loss: 0.3672594804565112, val loss: 0.35805927217006683
100% | ██████████ | 300/300 [00:29<00:00, 10.17it/s]

Epoch: 193, train loss: 0.37374286423126857, val loss: 0.3862571716308594
100% | ██████████ | 300/300 [00:29<00:00, 10.08it/s]

Epoch: 194, train loss: 0.36819708908597626, val loss: 0.37332547307014463
100% | ██████████ | 300/300 [00:29<00:00, 10.17it/s]

Epoch: 195, train loss: 0.3715069902439912, val loss: 0.34868636131286623
100% | ██████████ | 300/300 [00:30<00:00, 9.88it/s]

Epoch: 196, train loss: 0.36682710578044253, val loss: 0.3628829330205917
100% | ██████████ | 300/300 [00:30<00:00, 9.99it/s]

Epoch: 197, train loss: 0.3671202095846335, val loss: 0.41836997866630554
100% | ██████████ | 300/300 [00:29<00:00, 10.12it/s]

Epoch: 198, train loss: 0.37274823024868964, val loss: 0.39757572412490844
100% | ██████████ | 300/300 [00:29<00:00, 10.13it/s]

```
torch.save(network.state_dict(), './network')
```

```
!nvidia-smi
```

```
Sun Feb 13 09:12:48 2022
```

NVIDIA-SMI 460.32.03		Driver Version: 460.32.03		CUDA Version: 11.2	
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util Compute M.
0	Tesla T4	Off	00000000:00:04.0	Off	0
N/A	46C	P0	26W / 70W	7142MiB / 15109MiB	0% Default
					N/A

```
+-----+  
| Processes:  
+-----+
```

	GPU	GI	CI	PID	Type	Process name	GPU Usage	Memory
+	-	-	-	-	-	-	-	-

▼ Inception и получение результатов

```
# загружаем inception, чтобы можно было прогонять через него новые картинки,
# получать их эмбеддинги и генерировать описания с помощью нашей сети
from beheaded_inception3 import beheaded_inception_v3
inception = beheaded_inception_v3().train(False)

/usr/local/lib/python3.7/dist-packages/torchvision/models/inception.py:83: FutureWarning:
  ' due to scipy/scipy#11299), please set init_weights=True.', FutureWarning)
Downloading: "https://download.pytorch.org/models/inception_v3_google-1a9a5a14.pth" to ,
100%                                         104M/104M [00:03<00:00, 14.4MB/s]
```

▼ Сгенерируем описание

```
def generate_caption(image, caption_prefix=(START,), t=4, sample=True, max_len=100):
    assert isinstance(image, np.ndarray) and np.max(image) <= 1 \
        and np.min(image) >= 0 and image.shape[-1] == 3

    with torch.no_grad():
        image = torch.tensor(image.transpose([2, 0, 1]), dtype=torch.float32)

        vectors_8x8, vectors_neck, logits = inception(image[None])
        caption_prefix = list(caption_prefix)

        # слово за словом генерируем описание картинки
        for _ in range(max_len):
            # 1. представляем caption_prefix в виде матрицы
            caption_prefix_matrix = as_matrix([caption_prefix])
            caption_prefix_matrix = torch.tensor(caption_prefix_matrix, dtype=torch.int64)
            # 2. Получить из RNN-ки логиты, передав ей vectors_neck и матрицу из п.1
            logits = network(vectors_neck.to(device), caption_prefix_matrix.to(device))[0, -1]
            # 3. Перевести логиты RNN-ки в вероятности (например, с помощью F.softmax)
            probs = F.softmax(logits, dim=-1).cpu().numpy()
            # 4. сэмплировать следующее слово в описании, используя полученные вероятности. М
            # (тупо слово с самой большой вероятностью), можно сэмплировать из распределения
            probs = probs**t/np.sum(probs**t)

            if sample:
                word = np.random.choice(vocab, p=probs)
```

```

else:
    word = vocab[np.argmax(probs)]

    # 5. Добавляем новое слово в caption_prefix
    caption_prefix.append(word)
    # 6. Если RNN-ка сгенерила символ конца предложения, останавливаемся
    if caption_prefix == END:
        break

return caption_prefix

```

▼ Скачаем пару картинок, чтобы проверить качество:

```

from matplotlib import pyplot as plt
# from scipy.misc import imresize
import cv2
%matplotlib inline

#sample image
!wget https://pixel.nymag.com/imgs/daily/selectall/2018/02/12/12-tony-hawk.w710.h473.jpg -O img
img = plt.imread('img.jpg')
img = cv2.resize(img, (299, 299)).astype('float32') / 255.

--2022-02-13 09:13:06-- https://pixel.nymag.com/imgs/daily/selectall/2018/02/12/12-tony-hawk.w710.h473.jpg
Resolving pixel.nymag.com (pixel.nymag.com)... 199.232.192.70, 199.232.196.70
Connecting to pixel.nymag.com (pixel.nymag.com)|199.232.192.70|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://pyxis.nymag.com/v1/imgs/6ac/2a6/b48d3a180f333298f5bee60ff80f4ba886-12-151.101.2.217
--2022-02-13 09:13:06-- https://pyxis.nymag.com/v1/imgs/6ac/2a6/b48d3a180f333298f5bee60ff80f4ba886-12-151.101.2.217
Resolving pyxis.nymag.com (pyxis.nymag.com)... 151.101.2.217, 151.101.66.217, 151.101.151.101.2.217
Connecting to pyxis.nymag.com (pyxis.nymag.com)|151.101.2.217|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 54731 (53K) [image/jpeg]
Saving to: 'img.jpg'

img.jpg          100%[=====] 53.45K  ---KB/s   in 0.002s

2022-02-13 09:13:06 (29.7 MB/s) - 'img.jpg' saved [54731/54731]

```

```
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7faf47b94b90>
```



```
for i in range(10):
    print(' '.join(generate_caption(img, t=5., sample=True)[1:-1]))
```

```
a r g e <UNK> w o m a n <UNK> i s <UNK> s t a n d i n g <UNK> o n <UNK> a <UNK> s k a t
a r g e <UNK> w o m a n <UNK> w i t h <UNK> a <UNK> b l u e <UNK> s k a t e b o a r d <l
a r g e <UNK> b o y <UNK> i s <UNK> s k a t e b o a r d i n g <UNK> d o w n <UNK> a <UNK>
a r g e <UNK> w o m a n <UNK> i s <UNK> s t a n d i n g <UNK> o n <UNK> a <UNK> s k a t
a r g e <UNK> w o m a n <UNK> i s <UNK> s t a n d i n g <UNK> o n <UNK> a <UNK> s k a t
a r g e <UNK> w o m a n <UNK> i s <UNK> s t a n d i n g <UNK> o n <UNK> a <UNK> s k a t
a r g e <UNK> w o m a n <UNK> i n <UNK> a <UNK> s u i t <UNK> a n d <UNK> t i e <UNK> s
a r g e <UNK> w o m a n <UNK> i s <UNK> s t a n d i n g <UNK> o n <UNK> a <UNK> s k a t
a r g e <UNK> w o m a n <UNK> i s <UNK> s t a n d i n g <UNK> o n <UNK> a <UNK> s k a t
a r g e <UNK> w o m a n <UNK> i s <UNK> s t a n d i n g <UNK> o n <UNK> a <UNK> s k a t
a r g e <UNK> w o m a n <UNK> i s <UNK> s t a n d i n g <UNK> o n <UNK> a <UNK> s k a t
```

```
!wget http://ccanimalclinic.com/wp-content/uploads/2017/07/Cat-and-dog-1.jpg -O img1.jpg
img = plt.imread('img1.jpg')
img = cv2.resize(img, (299, 299)).astype('float32') / 255.

plt.imshow(img)
plt.show()

for i in range(10):
    print(' '.join(generate_caption(img, t=5.))[1:-1]))
```

```
--2022-02-13 09:13:27-- http://ccanimalclinic.com/wp-content/uploads/2017/07/Cat-and-dog-1.jpg
Resolving ccanimalclinic.com (ccanimalclinic.com)... 162.159.135.42
Connecting to ccanimalclinic.com (ccanimalclinic.com)|162.159.135.42|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://ccanimalclinic.com/wp-content/uploads/2017/07/Cat-and-dog-1.jpg [follow]
--2022-02-13 09:13:27-- https://ccanimalclinic.com/wp-content/uploads/2017/07/Cat-and-dog-1.jpg
Connecting to ccanimalclinic.com (ccanimalclinic.com)|162.159.135.42|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 106870 (104K) [image/jpeg]
Saving to: 'img1.jpg'
```

img1.jpg 100%[=====] 104.37K 525KB/s in 0.2s

2022-02-13 09:13:28 (525 KB/s) - 'img1.jpg' saved [106870/106870]



▼ Demo

ВОТ ЩАС БУИТ СМИШНО

Теперь ищите свои картинки, применяйте к ним сетку, смотрите че получится, реализовывайте вариативную часть =)

```
о м а п <UNK> в т л п <UNK> а <UNK> о а с п г о о м <UNK> в т л п <UNK> а <UNK> о е а <UNK>
```

```
# apply your network on images you've found
!wget https://www.tensorflow.org/tutorials/text/image\_captioning\_files/output\_9Psd1quzaAwg\_2.jpg

img = plt.imread('img2.jpg')
img = cv2.resize(img, (299, 299)).astype('float32') / 255.

plt.imshow(img)
plt.show()

for i in range(10):
    print(' '.join(generate_caption(img, t=5.))[1:-1]))
```



```
--2022-02-13 09:13:39-- https://www.tensorflow.org/tutorials/text/image_captioning_file
Resolving www.tensorflow.org (www.tensorflow.org)... 142.250.31.100, 142.250.31.102, 142.250.31.103
Connecting to www.tensorflow.org (www.tensorflow.org)|142.250.31.100|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 462356 (452K) [image/png]
Saving to: 'img2.jpg'
```

img2.jpg 100%[=====] 451.52K ---KB/s in 0.1s

2022-02-13 09:13:39 (3.55 MB/s) - 'img2.jpg' saved [462356/462356]



<UNK> p e r s o n <UNK> s t a n d i n g <UNK> n e x t <UNK> t o <UNK> a <UNK> b l u e <UNK>

▼ Бонус: Achtung

<LINK> m a n <LINK> i n <LINK> a <LINK> w h i t e <LINK> c h i r + <LINK> a n d <LINK> t i e <LINK>

В качестве бонусного задания предлагается реализовать механизм attention в rnn-сети, которую мы писали в базовой части.

https://colab.research.google.com/drive/1GDzceh8TsiW_XIqStYS-1Cir-2LRb2Qp#scrollTo=xBzfq5GnjOCG&printMode=true

✓ 8s completed at 15:13

