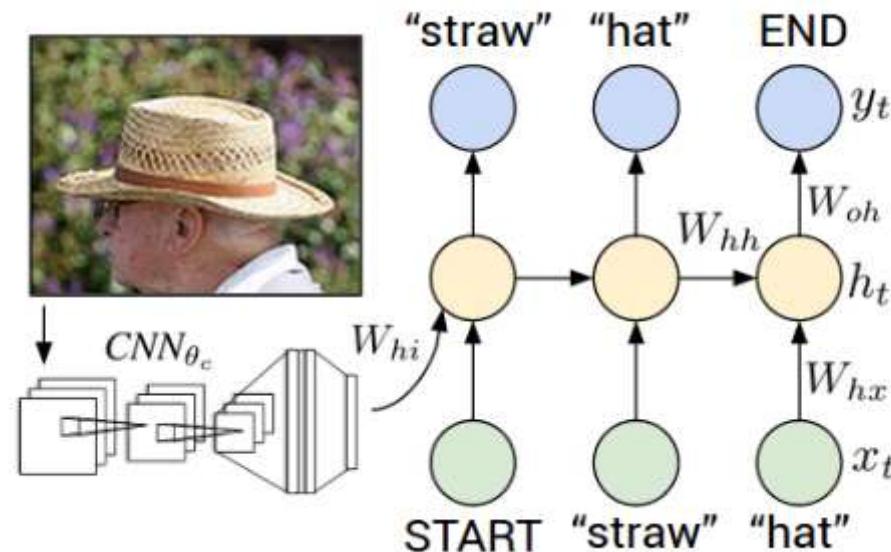


# Image Captioning



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 генерировать 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 сетьку, 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 (4.1.2.30)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-python) (1.19.5)

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]))\n\nEach image code is a 2048-unit vector [ shape: (118287, 2048) ]\n[0.3659946  0.2016555  0.9245725  0.57063824 0.547268   0.8275868\n 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],\n [ 1, 525, 8955, 6784, 3557, 525, 7341, 2296, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3],\n [ 1, 525, 8955, 9209, 3557, 5486, 8335, 3071, 2296, 2, 3, 3, 3, 3, 3, 3, 3, 3],\n [ 1, 6292, 1508, 8955, 9209, 6784, 3557, 3071, 6971, 5520, 7696, 2, 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 img_captions in range(len(captions)):
    for caption in range(len(captions[img_captions])):
        sentence = captions[img_captions][caption]
        captions_processed[img_captions][caption] = [START]+sentence.split(' ')+[END]

from collections import Counter
word_counts = Counter()

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

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

word_to_index = {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
```

```
captions_processed = np.array(captions_processed)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (array of arrays) is deprecated and will be removed in a future version. You're likely trying to convert a list of lists into an array of arrays, for example by calling np.array(list_of_lists).  
    """Entry point for launching an IPython kernel.
```

```
as_matrix(captions_processed[54])
```

```
array([[10400, 108, 58, 553, 20, 624, 54, 625, 31,
       41, 195, 10401, 10402, 10402, 10402, 10402, 10402,
       10402, 10402, 10402, 10402, 10402],
      [10400, 350, 626, 627, 499, 179, 553, 20, 10399,
       628, 8, 10401, 10402, 10402, 10402, 10402, 10402,
       10402, 10402, 10402, 10402, 10402],
      [10400, 134, 463, 629, 58, 620, 6, 41, 630,
       8, 10401, 10402, 10402, 10402, 10402, 10402, 10402,
       10402, 10402, 10402, 10402, 10402],
      [10400, 41, 10399, 13, 631, 2, 119, 632, 84,
       436, 633, 376, 634, 455, 41, 10399, 2, 41,
       10399, 484, 635, 8, 10401],
      [10400, 84, 156, 11, 43, 58, 90, 189, 113,
       104, 105, 20, 636, 8, 10401, 10402, 10402, 10402,
       10402, 10402, 10402, 10402]], dtype=int32)
```

```
vocab[553]
```

```
'behind'
```

```
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, dtype=torch.int64)

generate_batch(img_codes, captions_processed, 1)

(tensor([[0.2873, 0.3952, 0.5296, ..., 0.0550, 0.4615, 0.1198]]),
 tensor([[10400, 20, 321, 144, 31, 41, 327, 10, 20, 93,
         2, 155, 10, 20, 264, 8, 10401]]))
```

```
from sklearn.model_selection import train_test_split

train_img_codes, val_img_codes, train_captions, val_captions = train_test_split(img_codes, captions_processed, test_size=0.1)
```

## ▼ 2. Напишем свою сетку 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=0):
        super(self.__class__, self).__init__()

        # стандартная архитектура такой сети такая:
        # 1. линейные слои для преобразования эмбеддинга картинки в начальные состояния h0 и c0 LSTM-ки
        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_ix)

        # 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, num_layers=1)

        # 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 генерить текст
                             shape: [batch, cnn_feature_size]
        :param captions_ix:
                             таргет описания картинок в виде матрицы
        :returns: 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
logits = self.logits(lstm_out)

return logits

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 ones
    """

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

    # реализуйте стандартный cross entropy loss: итоговый лосс есть сумма лоссов для каждого слова.
    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 из captions_ix!
    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 склыдваться
    # 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 умножить на нее лосс,
    # либо просто используя ignore_index, который в торче есть как аргумент у некоторых лоссов.
    cross_entropy = nn.CrossEntropyLoss(reduction = 'none').to(device)
    loss = cross_entropy(logits_reshaped, next_reshaped).view(captions_ix_next.size()[0], captions_ix_next.size()[1]) * mask

    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

network.load_state_dict(torch.load('./network'))

<All keys matched successfully>

from tqdm import tqdm

for epoch in range(n_epochs):
    network.train()
    train_loss = 0
    for _ in tqdm(range(n_batches_per_epoch)):

        img_batch, captions_batch = generate_batch(train_img_codes, train_captions, batch_size)
        loss_t = compute_loss(network, img_batch, captions_batch)

        loss_t.backward()
        optimizer.step()
        optimizer.zero_grad()

        train_loss += loss_t.cpu().data.numpy()[0]
```

```
train_loss /= n_batches_per_epoch

val_loss = 0
network.eval()
with torch.no_grad():
    for _ in range(n_validation_batches):
        loss_t = compute_loss(network, *generate_batch(val_img_codes, val_captions, batch_size))
        val_loss += loss_t.cpu().data.numpy()[0]
    val_loss /= n_validation_batches
print('\nEpoch: {}, train loss: {}, val loss: {}'.format(epoch + 1, train_loss, val_loss))
```

Epoch: 118, train loss: 1.095436034798622, val loss: 1.3435924410820008  
100%|██████████| 300/300 [00:17<00:00, 17.52it/s]

Epoch: 119, train loss: 1.073200099269549, val loss: 1.283762514591217  
100%|██████████| 300/300 [00:17<00:00, 17.54it/s]

Epoch: 120, train loss: 1.0731476294994353, val loss: 1.2139656007289887  
100%|██████████| 300/300 [00:17<00:00, 17.54it/s]

Epoch: 121, train loss: 1.0719336090485254, val loss: 1.2698242306709289  
100%|██████████| 300/300 [00:17<00:00, 17.37it/s]

Epoch: 122, train loss: 1.0640631046891214, val loss: 1.0770138859748841  
100%|██████████| 300/300 [00:16<00:00, 17.65it/s]

Epoch: 123, train loss: 1.0742420812447866, val loss: 1.2855647802352905  
100%|██████████| 300/300 [00:17<00:00, 17.53it/s]

Epoch: 124, train loss: 1.0728501349687576, val loss: 1.294817864894867  
100%|██████████| 300/300 [00:16<00:00, 17.83it/s]

Epoch: 125, train loss: 1.085890524983406, val loss: 1.2873378038406371  
100%|██████████| 300/300 [00:16<00:00, 17.72it/s]

Epoch: 126, train loss: 1.079671054283778, val loss: 1.2042407095432281  
100%|██████████| 300/300 [00:17<00:00, 17.34it/s]

Epoch: 127, train loss: 1.0666995946566264, val loss: 1.2171140253543853

100%|██████████| 300/300 [00:16<00:00, 17.84it/s]

Epoch: 128, train loss: 1.0892577996850015, val loss: 1.1394583582878113

100%|██████████| 300/300 [00:17<00:00, 17.47it/s]

Epoch: 129, train loss: 1.069142993092537, val loss: 1.1116535663604736

100%|██████████| 300/300 [00:17<00:00, 17.60it/s]

Epoch: 130, train loss: 1.0709451194604238, val loss: 1.2586759507656098

100%|██████████| 300/300 [00:17<00:00, 17.53it/s]

Epoch: 131, train loss: 1.070426097313563, val loss: 1.3287044882774353

100%|██████████| 300/300 [00:17<00:00, 17.49it/s]

Epoch: 132, train loss: 1.072276900013288, val loss: 1.16154488325119

100%|██████████| 300/300 [00:17<00:00, 17.32it/s]

Epoch: 133, train loss: 1.0501350896557171, val loss: 1.2755453824996947

100%|██████████| 300/300 [00:17<00:00, 17.63it/s]

Epoch: 134, train loss: 1.0678497389952342, val loss: 1.1847934603691102

100%|██████████| 300/300 [00:17<00:00, 17.46it/s]

Epoch: 135, train loss: 1.0678985770543417, val loss: 1.2311964333057404

100%|██████████| 300/300 [00:17<00:00, 17.55it/s]

Epoch: 136, train loss: 1.065927947362264, val loss: 1.2175229847431184

100%|██████████| 300/300 [00:16<00:00, 17.92it/s]

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

```
!nvidia-smi
```

Sun Feb 13 13:41:39 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.	MIG M.

0	Tesla T4	Off		00000000:00:04.0 Off			0						
N/A	38C	P0	27W / 70W	1378MiB / 15109MiB		0%	Default		N/A				
<hr/>													
<hr/>													
Processes:													
GPU	GI	CI	PID	Type	Process name	GPU Memory							
ID	ID					Usage							
<hr/>													
<hr/>													

## ▼ 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: The default weight initialization
  ' due to scipy/scipy#11299), please set init_weights=True.', FutureWarning)
Downloading: "https://download.pytorch.org/models/inception\_v3\_google-1a9a5a14.pth" to /root/.cache/torch/hub/checkpoint
100%                                         104M/104M [00:01<00:00, 91.5MB/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. Можно сэмплировать жадно
    # (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 word == 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.jpg
img = plt.imread('img.jpg')
img = cv2.resize(img, (299, 299)).astype('float32') / 255.

--2022-02-13 13:42:48-- 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-tony-hawk.h473.w710.jpg [follow
--2022-02-13 13:42:48-- https://pyxis.nymag.com/v1/imgs/6ac/2a6/b48d3a180f333298f5bee60ff80f4ba886-12-tony-hawk.h473.w710.jpg
Resolving pyxis.nymag.com (pyxis.nymag.com)... 151.101.2.217, 151.101.66.217, 151.101.130.217, ...
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 0.001s

2022-02-13 13:42:48 (51.4 MB/s) - 'img.jpg' saved [54731/54731]
```

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

```
<matplotlib.image.AxesImage at 0x7f6b85b3cdd0>

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

    a man riding a skateboard up the side of a ramp .
    a man riding a skateboard up the side of a ramp .
    a man riding a skateboard up the side of a ramp .
    a man on a skateboard is performing a trick .
    a man on a skateboard is performing a trick .
    a man riding a skateboard up the side of a ramp .
    a man riding a skateboard on a ramp in a skate park .
    a man riding a skateboard up a ramp on a skateboard .
    a man riding a skateboard on a ramp .
    a man on a skateboard is performing a trick .

!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.

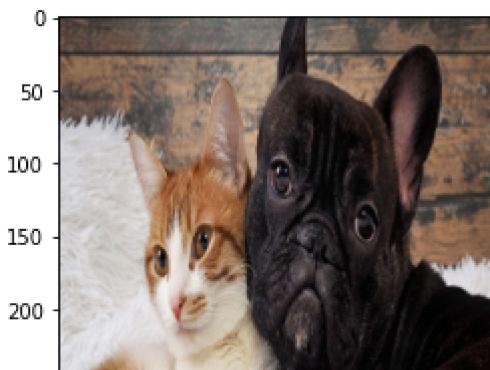
plt.imshow(img)
plt.show()

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

```
--2022-02-13 13:45:03-- 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 [following]
--2022-02-13 13:45:04-- 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   546KB/s   in 0.2s

2022-02-13 13:45:04 (546 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 =)

a cat is standing on the floor with a suitcase .

```
# apply your network on images you've found
!wget https://www.tensorflow.org/tutorials/text/image\_captioning\_files/output\_9Psd1quzaAwg\_2.png -O img2.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 13:45:11-- https://www.tensorflow.org/tutorials/text/image\_captioning\_files/output\_9Psd1quzaAWg\_2.png
Resolving www.tensorflow.org (www.tensorflow.org)... 172.217.15.110, 2607:f8b0:4004:837::200e
Connecting to www.tensorflow.org (www.tensorflow.org)|172.217.15.110|: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 13:45:11 (3.73 MB/s) - 'img2.jpg' saved [462356/462356]
```



a man in a suit and tie with a bow tie .  
a man wearing a suit and bow tie .  
a man in a suit and tie looking at his cell phone .  
a man wearing a suit and tie with a bow tie .  
a man wearing a black jacket and a bow tie .  
a man wearing a suit and tie with a bow tie .  
a man wearing a suit and tie with a bow tie .  
a man wearing a suit and tie holding a camera .  
a man in a suit and tie with a bow tie .  
a man in a suit and tie is smiling .

## ▼ Бонус: Achtung

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

---

✓ 5s completed at 19:45

