Лабораторная работа №1

Выполнил студент группы ББМО-01-23 Буланов Андрей Алексеевич

Импорт библиотек

```
!pip install tf-keras-vis
```

```
Collecting tf-keras-vis
  Downloading tf_keras_vis-0.8.7-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: scipy in
/usr/local/lib/python3.11/dist-packages (from tf-keras-vis) (1.13.1)
Requirement already satisfied: pillow in
/usr/local/lib/python3.11/dist-packages (from tf-keras-vis) (11.1.0)
Requirement already satisfied: deprecated in
/usr/local/lib/python3.11/dist-packages (from tf-keras-vis) (1.2.18)
Requirement already satisfied: imageio in
/usr/local/lib/python3.11/dist-packages (from tf-keras-vis) (2.36.1)
Requirement already satisfied: packaging in
/usr/local/lib/python3.11/dist-packages (from tf-keras-vis) (24.2)
Requirement already satisfied: wrapt<2,>=1.10 in
/usr/local/lib/python3.11/dist-packages (from deprecated->tf-keras-
vis) (1.17.2)
Requirement already satisfied: numpy in
/usr/local/lib/python3.11/dist-packages (from imageio->tf-keras-vis)
(1.26.4)
Downloading tf_keras_vis-0.8.7-py3-none-any.whl (52 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 52.5/52.5 kB 2.1 MB/s eta
0:00:00
```

```
%reload_ext autoreload
%autoreload 2

import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
import tensorflow as tf
from tf_keras_vis.utils import num_of_gpus
_, gpus = num_of_gpus()
print('Tensorflow recognized {} GPUs'.format(gpus))
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.applications.vgg16 import preprocess_input
```

```
Tensorflow recognized 1 GPUs
```

```
from tensorflow.keras.applications.vgg16 import VGG16 as Model
```

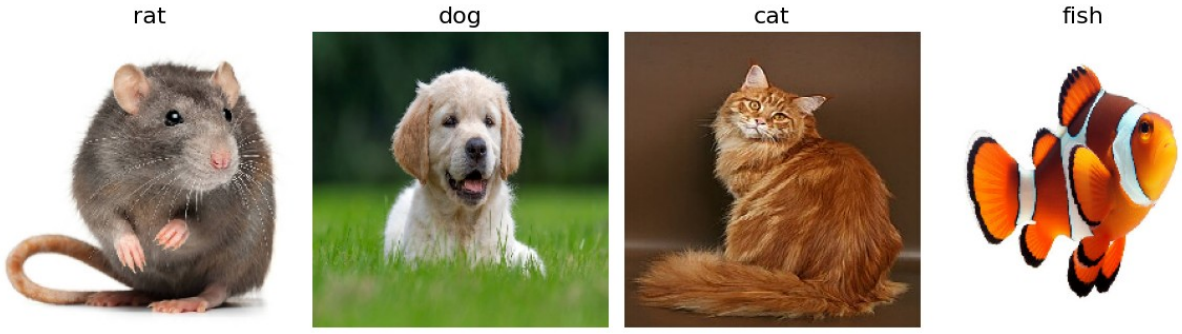
```
model = Model(weights='imagenet', include_top=True)
model.summary()

Model: "vgg16"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━━┩
│ input_layer_1 (InputLayer)           │ (None, 224, 224, 3)         │
0 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ block1_conv1 (Conv2D)                │ (None, 224, 224, 64)        │
1,792 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ block1_conv2 (Conv2D)                │ (None, 224, 224, 64)        │
36,928 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ block1_pool (MaxPooling2D)           │ (None, 112, 112, 64)        │
0 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ block2_conv1 (Conv2D)                │ (None, 112, 112, 128)       │
73,856 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ block2_conv2 (Conv2D)                │ (None, 112, 112, 128)       │
147,584 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ block2_pool (MaxPooling2D)           │ (None, 56, 56, 128)         │
0 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ block3_conv1 (Conv2D)                │ (None, 56, 56, 256)         │
295,168 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ block3_conv2 (Conv2D)                │ (None, 56, 56, 256)         │
590,080 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ block3_conv3 (Conv2D)                │ (None, 56, 56, 256)         │
590,080 │
├──────────────────────────────────────┼─────────────────────────────┼
```

```
──────────────────┤
│ block3_pool (MaxPooling2D)            │ (None, 28, 28, 256)          │
0 │
├──────────────────────────────────────┼──────────────────────────────┼
──────────────────┤
│ block4_conv1 (Conv2D)                 │ (None, 28, 28, 512)          │
1,180,160 │
├──────────────────────────────────────┼──────────────────────────────┼
──────────────────┤
│ block4_conv2 (Conv2D)                 │ (None, 28, 28, 512)          │
2,359,808 │
├──────────────────────────────────────┼──────────────────────────────┼
──────────────────┤
│ block4_conv3 (Conv2D)                 │ (None, 28, 28, 512)          │
2,359,808 │
├──────────────────────────────────────┼──────────────────────────────┼
──────────────────┤
│ block4_pool (MaxPooling2D)            │ (None, 14, 14, 512)          │
0 │
├──────────────────────────────────────┼──────────────────────────────┼
──────────────────┤
│ block5_conv1 (Conv2D)                 │ (None, 14, 14, 512)          │
2,359,808 │
├──────────────────────────────────────┼──────────────────────────────┼
──────────────────┤
│ block5_conv2 (Conv2D)                 │ (None, 14, 14, 512)          │
2,359,808 │
├──────────────────────────────────────┼──────────────────────────────┼
──────────────────┤
│ block5_conv3 (Conv2D)                 │ (None, 14, 14, 512)          │
2,359,808 │
├──────────────────────────────────────┼──────────────────────────────┼
──────────────────┤
│ block5_pool (MaxPooling2D)            │ (None, 7, 7, 512)            │
0 │
├──────────────────────────────────────┼──────────────────────────────┼
──────────────────┤
│ flatten (Flatten)                     │ (None, 25088)                │
0 │
├──────────────────────────────────────┼──────────────────────────────┼
──────────────────┤
│ fc1 (Dense)                           │ (None, 4096)                 │
102,764,544 │
├──────────────────────────────────────┼──────────────────────────────┼
──────────────────┤
│ fc2 (Dense)                           │ (None, 4096)                 │
16,781,312 │
├──────────────────────────────────────┼──────────────────────────────┼
──────────────────┤
```

```
| predictions (Dense)                  | (None, 1000)                  |
4,097,000 |
└──────────────────────────────────────┴──────────────────────────────┴
──────────────────┘

 Total params: 138,357,544 (527.79 MB)

 Trainable params: 138,357,544 (527.79 MB)

 Non-trainable params: 0 (0.00 B)
```

Загрузка и предобработка исходных изображений

```
# Заголовки наших изображений
image_titles = ['rat', 'dog', 'cat', 'fish']

# Загружаем изображения и конвертируем их в массив Numpy
img1 = load_img('rat.jpg', target_size=(224, 224))
img2 = load_img('dog.jpg', target_size=(224, 224))
img3 = load_img('cat.jpg', target_size=(224, 224))
img4 = load_img('fish.jpg', target_size=(224, 224))
images = np.asarray([np.array(img1), np.array(img2), np.array(img3),
np.array(img4)])

# Подготавливаем входы для VGG16
X = preprocess_input(images)

# Выводим
f, ax = plt.subplots(nrows=1, ncols=4, figsize=(12, 4))
for i, title in enumerate(image_titles):
    ax[i].set_title(title, fontsize=16)
    ax[i].imshow(images[i])
    ax[i].axis('off')
plt.tight_layout()
plt.show()
```



Когда функция активации softmax применяется к последнему слою модели, это может препятствовать созданию изображений внимания, поэтому следует заменить эту функцию на функцию линейной активации. Хотя здесь мы создаем и используем экземпляр

ReplaceToLinear, мы также можем использовать функцию модификатора модели, определенную нами.

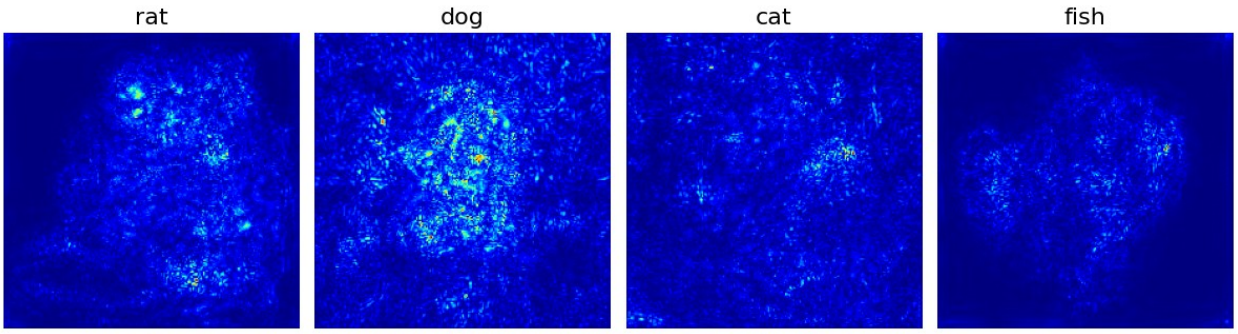
```
from tf_keras_vis.utils.model_modifiers import ReplaceToLinear

replace2linear = ReplaceToLinear()

def model_modifier_function(cloned_model):
    cloned_model.layers[-1].activation = tf.keras.activations.linear

from tf_keras_vis.utils.scores import CategoricalScore

score = CategoricalScore([285, 277, 330, 675])
# Где: 277 - собака, 285 - крыса, 330 - кошка, 675 - рыба

# Вместо использования объекта CategoricalScore
# определим функцию с нуля следующим образом:
def score_function(output):
    # Переменная `output` ссылается на выходы модели,
    # таким образом, что размерность `output` равна `(3, 1000)` где,
(номер примера, номер класса)
    return (output[0][285], output[1][277], output[2][330], output[3]
[675])
```

Saliency генерирует карту значимости, на которой отображаются области входного изображения, которые имеют наибольшее влияние на выходное значение.

```
%%time
from tensorflow.keras import backend as K
from tf_keras_vis.saliency import Saliency

# Создаем объект внимания
saliency = Saliency(model,
                    model_modifier=replace2linear,
                    clone=True)

# Генерируем карту внимания
saliency_map = saliency(score, X)

# Выводим
f, ax = plt.subplots(nrows=1, ncols=4, figsize=(12, 4))
for i, title in enumerate(image_titles):
    ax[i].set_title(title, fontsize=16)
    ax[i].imshow(saliency_map[i], cmap='jet')
    ax[i].axis('off')
plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/models/
functional.py:237: UserWarning: The structure of `inputs` doesn't
```

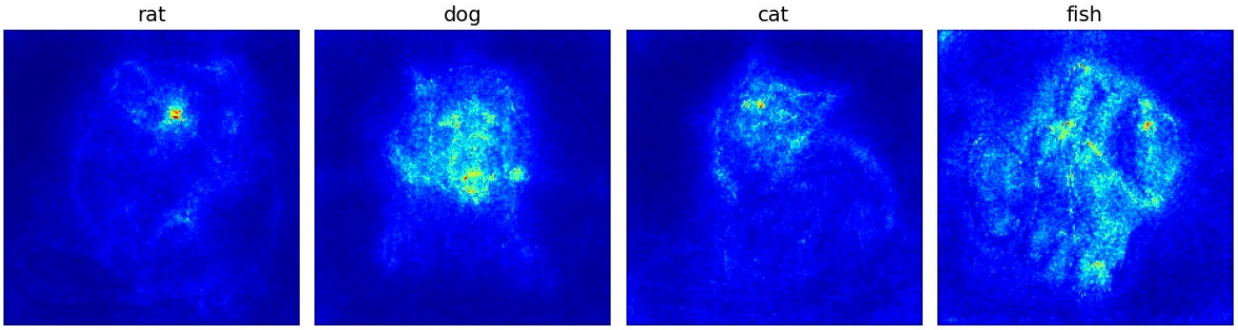
```
match the expected structure.
Expected: keras_tensor_23
Received: inputs=['Tensor(shape=(4, 224, 224, 3))']
  warnings.warn(msg)
```




```
CPU times: user 3.89 s, sys: 1.01 s, total: 4.9 s
Wall time: 6.53 s
```

Карта значимости Vanilla слишком шумная, поэтому следует удалить шум на карте значимости с помощью SmoothGrad, который уменьшает шум на карте значимости путем добавления шума к входному изображению.

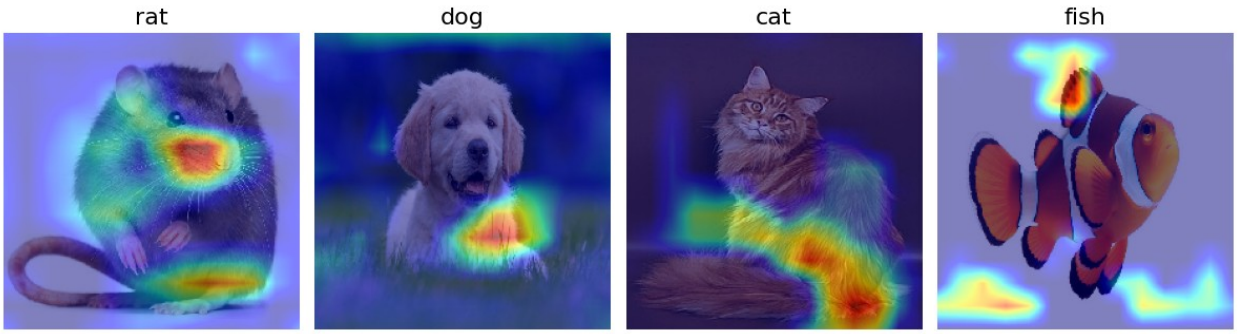
```
%%time

# Генерируем карту внимания со сглаживанием, которое уменьшает шум за счет добавления шума
saliency_map = saliency(score,
                        X,
                        smooth_samples=20, # Количество итераций расчета градиентов
                        smooth_noise=0.20) # уровень распространения шума

# Выводим
f, ax = plt.subplots(nrows=1, ncols=4, figsize=(12, 4))
for i, title in enumerate(image_titles):
    ax[i].set_title(title, fontsize=14)
    ax[i].imshow(saliency_map[i], cmap='jet')
    ax[i].axis('off')
plt.tight_layout()
plt.savefig('smoothgrad.png')
plt.show()
```




```
CPU times: user 2.37 s, sys: 281 ms, total: 2.65 s
Wall time: 3.68 s
```

Визуализация тепловой карты - GradCam

```
%%time

from matplotlib import cm
from tf_keras_vis.gradcam import Gradcam

# Создаём объект визуализации Gradcam
gradcam = Gradcam(model,
                  model_modifier=replace2linear,
                  clone=True)

# Генерируем тепловую карту с помощью GradCAM
cam = gradcam(score,
              X,
              penultimate_layer=-1)

# Выводим
f, ax = plt.subplots(nrows=1, ncols=4, figsize=(12, 4))
for i, title in enumerate(image_titles):
    heatmap = np.uint8(cm.jet(cam[i])[..., :4] * 255)
    ax[i].set_title(title, fontsize=16)
    ax[i].imshow(images[i])
    ax[i].imshow(heatmap, cmap='jet', alpha=0.5) # overlay
    ax[i].axis('off')
plt.tight_layout()
plt.show()
```

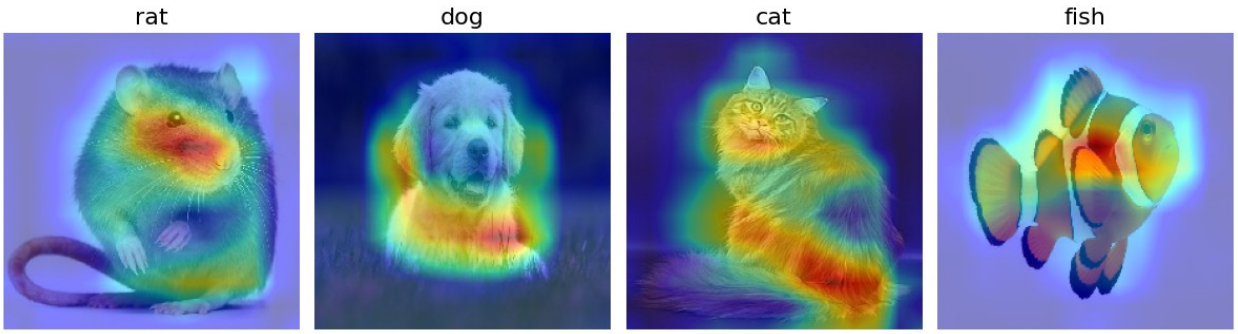



```
CPU times: user 1.47 s, sys: 877 ms, total: 2.35 s
Wall time: 2.22 s
```

Как видно, тепловые пятна не полностью покрывают цель на изображениях. В следующем шаге мы решим эту проблему. Как видно, на данном шаге создания карты появился новый аргумент. Здесь предпоследний слой — это сверточный слой, ближайший к плотным слоям. Выходные данные этого слоя — это то, откуда GradCam получает градиенты.

GradCam++ — улучшенная версия GradCam, позволяющий обеспечить лучшее визуальное объяснение прогнозов модели CNN.

```
%%time

from tf_keras_vis.gradcam_plus_plus import GradcamPlusPlus

# Создаем объект GradCAM++
gradcam = GradcamPlusPlus(model,
                          model_modifier=replace2linear,
                          clone=True)
# Генерируем тепловую карту с помощью GradCAM++
cam = gradcam(score,
              X,
              penultimate_layer=-1)
# Визуализируем
f, ax = plt.subplots(nrows=1, ncols=4, figsize=(12, 4))
for i, title in enumerate(image_titles):
    heatmap = np.uint8(cm.jet(cam[i])[..., :4] * 255)
    ax[i].set_title(title, fontsize=16)
    ax[i].imshow(images[i])
    ax[i].imshow(heatmap, cmap='jet', alpha=0.5)
    ax[i].axis('off')
plt.tight_layout()
plt.savefig('gradcam_plus_plus.png')
plt.show()
```




```
CPU times: user 1.71 s, sys: 979 ms, total: 2.69 s
Wall time: 2.69 s
```