



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

**«МИРЭА – Российский технологический университет»  
РТУ МИРЭА**

ИКБ направление «Киберразведка и противодействие угрозам с применением технологий  
искусственного интеллекта» 10.04.01

Кафедра КБ-4 «Интеллектуальные системы информационной безопасности»

### **Лабораторная работа №3**

по дисциплине: «Анализ защищенности систем искусственного  
интеллекта»

Группа:

ББМО-01-22

Выполнил:

Гребенник Г.С

Проверил:

к.т.н. Спирин А.А.

Москва, 2023

## Содержание

Ход работы.....	3
Вывод: .....	8

## Ход работы

Перед началом работы, стал известен факт обнаруженный одним из одноклассников, что keras-vis указанный в работе не работает с tensorflow второй версии, поэтому будет применено найденное им решение в виде обновленного keras-vis: *tf-keras-vis*.

### 1. Устанавливаем tf-keras-vis:

```
[3] !pip install tf-keras-vis
```

Requirement already satisfied: tf-keras-vis in /usr/local/lib/python3.10/dist-packages (0.8.6)  
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from tf-keras-vis) (1.11.4)  
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from tf-keras-vis) (9.4.0)  
Requirement already satisfied: deprecated in /usr/local/lib/python3.10/dist-packages (from tf-keras-vis) (1.2.14)  
Requirement already satisfied: imageio in /usr/local/lib/python3.10/dist-packages (from tf-keras-vis) (2.31.6)  
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tf-keras-vis) (23.2)  
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.10/dist-packages (from deprecated->tf-keras-vis) (1.14.1)  
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from imageio->tf-keras-vis) (1.23.5)

### 2. Добавим необходимые библиотеки:

```
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
```

### 3. Загружаем модель VGG16:

```
[5] from tensorflow.keras.applications.vgg16 import VGG16 as Model
model = Model(weights='imagenet', include_top=True)
model.summary()
```

Downloading data from [https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16\\_weights\\_tf\\_dim\\_ordering\\_tf\\_kernels.h5](https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5)  
553467096/553467096 [=====] - 8s 0us/step  
Model: "vgg16"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

=====  
Total params: 138357544 (527.79 MB)  
Trainable params: 138357544 (527.79 MB)  
Non-trainable params: 0 (0.00 Byte)

4. Загружаем изображения из датасета ImageNet, после чего  
подготавливаем входы для VGG16



## 5. Реализуем необходимые функции:

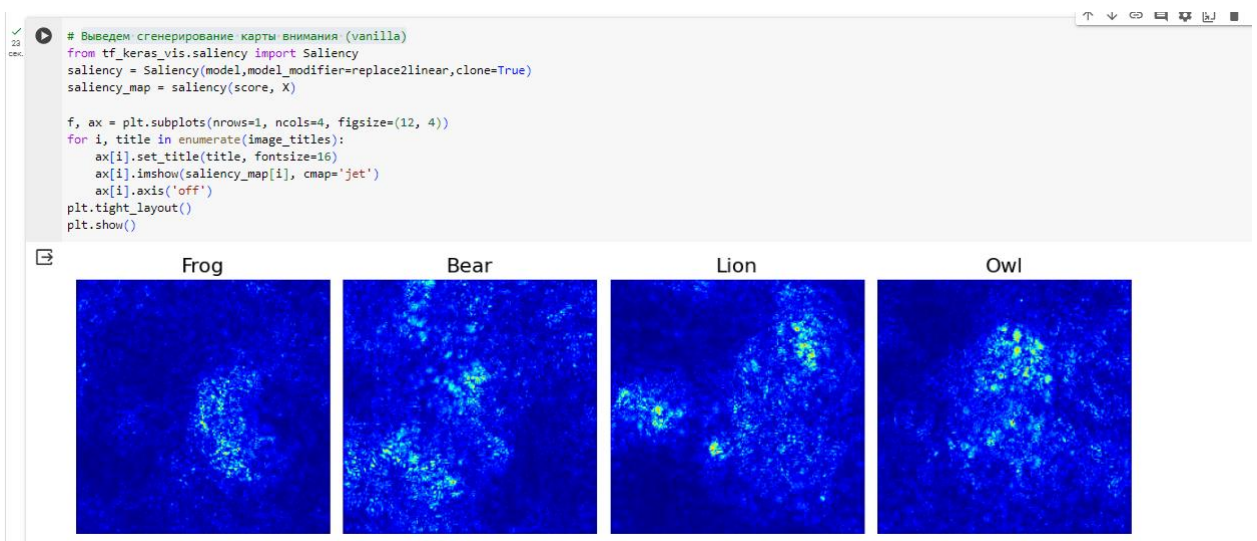
- Функция линейной активации и расчета score

```

0
# Заменяю на линейную функцию
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([56, 57, 60, 87])
def score_function(output):
    return (output[0][56], output[1][57], output[2][60], output[3][87])

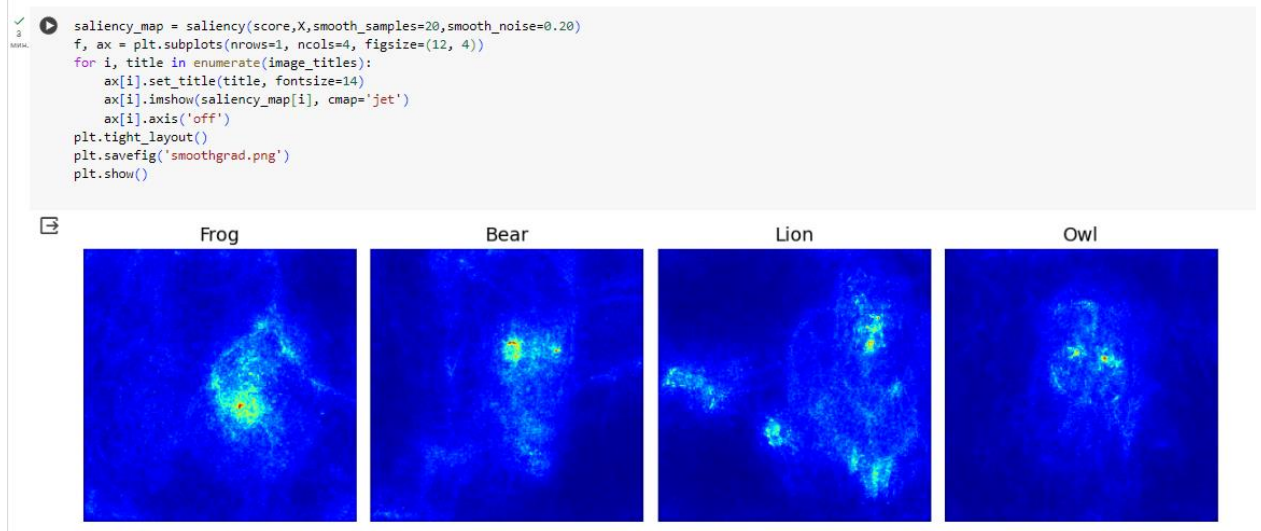
```

- Выведем сгенерирование карты внимания (vanilla)



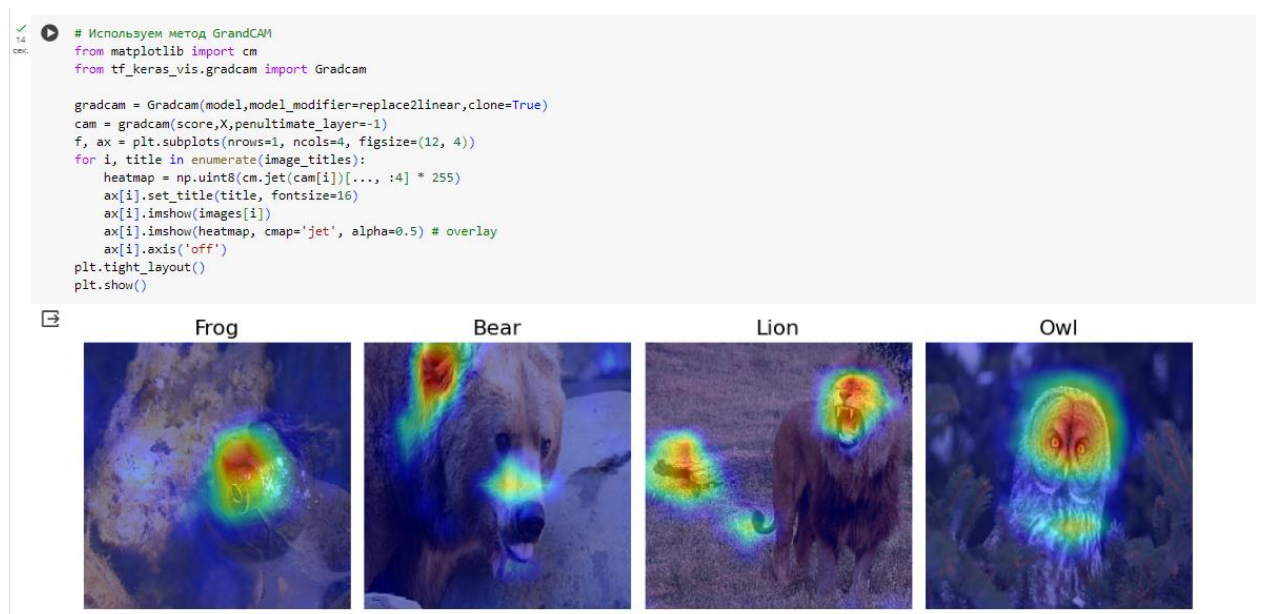
Saliency создает карты внимания, выделяющие из входного изображения важные области. Они влияют на результат, карта помогает определить ключевые части изображения для получения того или иного результата.

- Карты Smoothgrad:



Smoothgrad выделяет на входном изображении области с наибольшим воздействием, что бы в последствии выделить на выходном наиболее важные объекты. В сравнение с Saliency мы видим более четкий контур, хоть и все равно не понятный для определения объекта на изображении.

- Способ GradCaM:



Данный способ хорошо справляется с маленькими объектами на изображении, такими как лягушка, но крупные объекты плохо



обрабатываются, например медведь, площадь покрытия которого меньше 30%.

- Способ GrandCAM++



Данный способ лучше справляется с задачей, чем GrandCAM, мы можем видеть, что он предоставляет нам более точное представление объекта и даже крупного в виде медведя. Данный способ можно выделить как самый предпочтительный из рассмотренных.

### **Вывод:**

В ходе выполнения лабораторной работы были рассмотрены способы построения карт вынимания для анализа изображений из набора данных ImageNet. Были рассмотрены и построены карты значимости для выбранных нами изображений методами: Silency, SmoothGrad, GrandCAM, GrandCam++. Наибольшую эффективность в результате анализа изображений показал GrandCAM++.