МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт кибербезопасности и цифровых технологий
КБ-4 «Интеллектуальные системы информационной безопасности»

Отчет по лабораторной работе №3

по дисциплине: «Анализ защищенности систем искусственного интеллекта»

Выполнил:
Тимофеев И.О.

Проверил:
К.т.н. Спирин А.А

Москва, 2023

Выполним импорт библиотек

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

```
Collecting tf-keras-vis
  Downloading tf_keras_vis-0.8.6-py3-none-any.whl (52 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 52.1/52.1 kB 1.5 MB/s eta 0:00:00
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)
Collecting deprecated (from tf-keras-vis)
  Downloading Deprecated-1.2.14-py2.py3-none-any.whl (9.6 kB)
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)
Installing collected packages: deprecated, tf-keras-vis
Successfully installed deprecated-1.2.14 tf-keras-vis-0.8.6
```

```
%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))
```

Импортируем модель VGG16

```
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
553467096/553467096 [==============================] - 23s 0us/step
Model: "vgg16"
_________________________________________________________________
```

Далее нужно загрузить исходные изображения. В данном случае у нас изображения кошки, лисы, волка и собаки.

```python
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.applications.vgg16 import preprocess_input

# Заголовки наших изображений
image_titles = ['Cat', 'Fox', 'Wolf', 'Dog']

# Загружаем изображения и конвертируем их в массив Numpy
img1 = load_img('cat.jpeg', target_size=(224, 224))
img2 = load_img('fox.jpeg', target_size=(224, 224))
img3 = load_img('wolf.jpeg', target_size=(224, 224))
img4 = load_img('dog.jpeg', 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()
```

Заменяем функцию активации на линейную.

```python
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
```

Создаем экземпляр Score или определяем score function

```python
from tf_keras_vis.utils.scores import CategoricalScore

score = CategoricalScore([386, 285, 330, 134])
# Где: 386 - кот, 285 - лис, 330 - волк, 134 - собака

# 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][386], output[1][285], output[2][330], output[3][134])
```

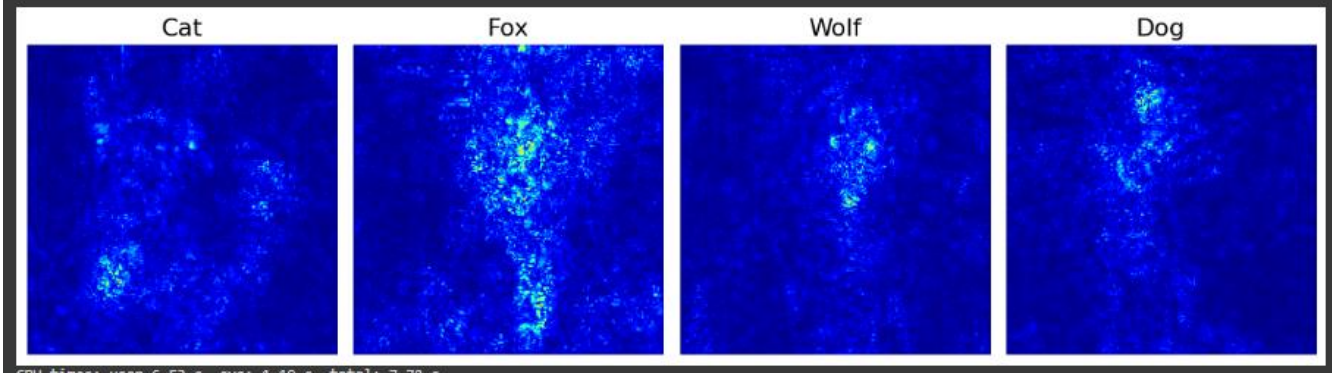
Импотируем Saliency - генерирует карту значимости, на которой отображаются области входного изображения, которые имеют наибольшее влияние на выходное значение

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

# 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()
```



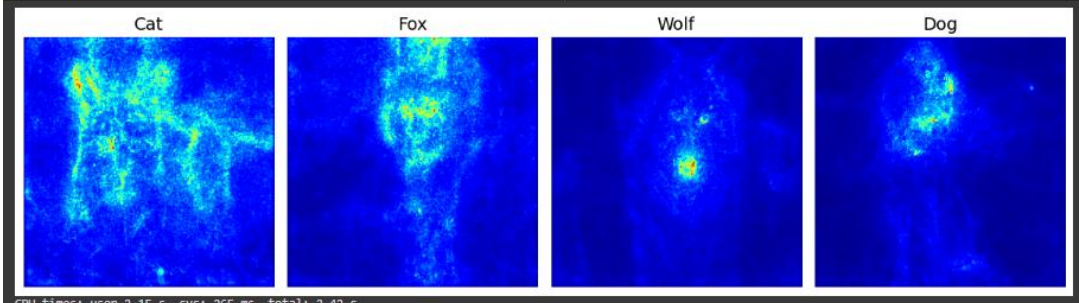

```
CPU times: user 6.53 s, sys: 1.19 s, total: 7.72 s
Wall time: 9.25 s
```

SmoothGrad

```
%%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 2.15 s, sys: 265 ms, total: 2.42 s
Wall time: 3.55 s
```

Grad-CAM

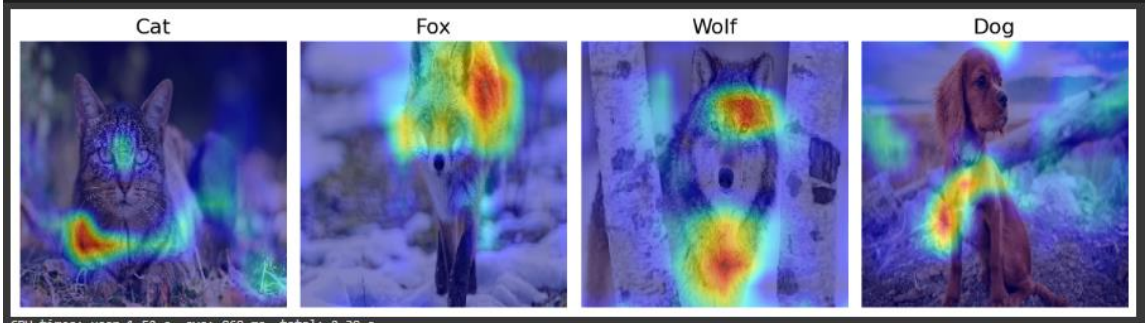
```
%%time

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()
```



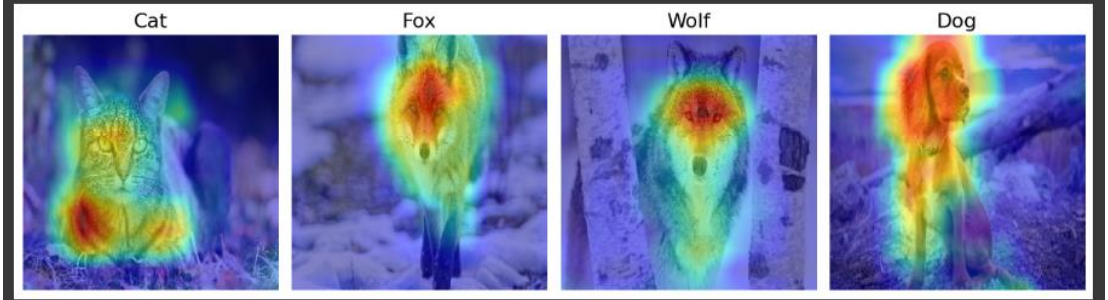

```
CPU times: user 1.52 s, sys: 868 ms, total: 2.38 s
Wall time: 2.29 s
```

GradCam++

```
%%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 2.45 s, sys: 1.04 s, total: 3.49 s
Wall time: 4.15 s
```

Заключение

В данной работе использовались методы SmoothGrad, GrandCam, GrandCam. SmoothGrad – техника, предназначенная для снижения шума. GrandCam использует градиенты, чтобы создать взвешенной карты значимости. Метод GrandCam++ - расширенный метод GrandCam, котрый учитывает также градиенты второго порядка.

Отличия между этими методами заключаются в подходе к вычислению важности пикселей и методах сглаживания или учета дополнительных данных, таких как градиенты. Каждый метод имеет свои преимущества и может быть применен в зависимости от конкретных задач и требований.