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»

Выполнила:
Студентка группы ББМО-02-22
Бардасова Ирина Александровна

Проверил:
К.т.н. Спирин Андрей Андреевич

Москва, 2023

Содержание

Ход работы

Шаг 1. Выполним импорт библиотек (рисунок 1).

```
In [1]: !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 1.6 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

In [2]: %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))

Tensorflow recognized 1 GPUs
```

Рисунок 1 – Импорт библиотек

Шаг 2. Воспользуемся моделью VGG16, которую вы можете загрузить непосредственно из Keras. VGG16 — это простая и широко используемая архитектура сверточной нейронной сети (CNN), используемая для ImageNet, крупного проекта визуальной базы данных, используемого в исследованиях программного обеспечения для распознавания визуальных объектов (рисунок 2).

```
In [3]: 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 [==============================] - 15s 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_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)
_________________________________________________________________
```

Рисунок 2 – Модель VGG16

Шаг 3. Теперь загрузим и предобработаем исходные изображения. Это необходимо сделать, прежде чем мы сможем отправить их в модель и получить оценки классов. Мои четыре изображения: слон, кошка, кролик и птица (рис. 3-4).

In [5]:

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

# Заголовки наших изображений
image_titles = ['Elephant', 'Cat', 'Rabbit', 'Bird']

# Загружаем изображения и конвертируем их в массив Numpy
img1 = load_img('elephant.jpg', target_size=(224, 224))
img2 = load_img('cat.jpg', target_size=(224, 224))
img3 = load_img('rabbit.jpeg', target_size=(224, 224))
img4 = load_img('bird.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()
```

Рисунок 3 – Предобработка исходных изображений

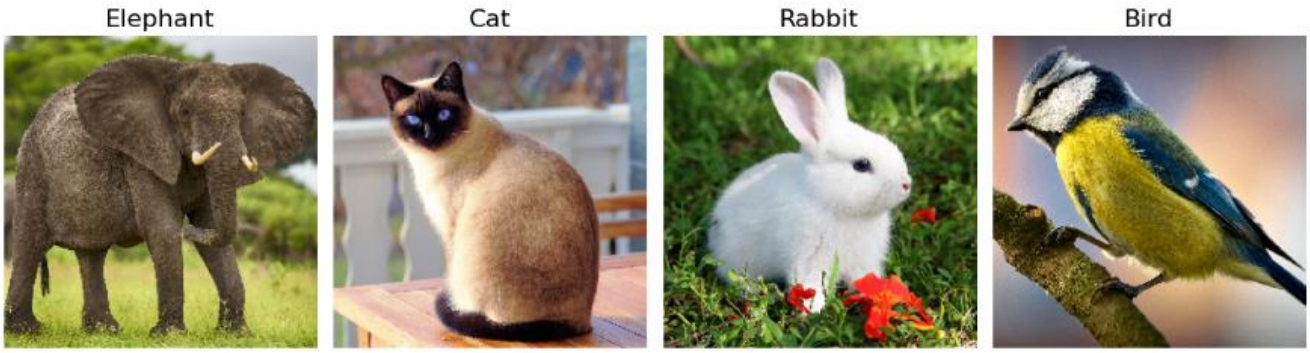


Рисунок 4 – Визуализация наших изображений

Когда функция активации softmax применяется к последнему слою модели, это может препятствовать созданию изображений внимания, поэтому следует заменить эту функцию на функцию линейной активации (рис. 5). Хотя здесь мы создаем и используем экземпляр ReplaceToLinear, мы также можем использовать функцию модификатора модели, определенную нами.

In [6]:
```
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
```

Рисунок 5 – Замена на функцию линейной активации

1 Функция расчета

Шаг 4. В данном шаге мы создаем экземпляр Score или определяем score function, которая возвращает целевые баллы. Здесь они возвращают количество очков, соответствующее слону, кошке, кролику и птице (рисунок 6).

In [7]:
```
from tf_keras_vis.utils.scores import CategoricalScore

score = CategoricalScore([386, 285, 330, 134])
# Где: 386 - слон, 285 - кошка, 330 - кролик, 134 - птица

# 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])
```

Рисунок 6 – Присваивание значений

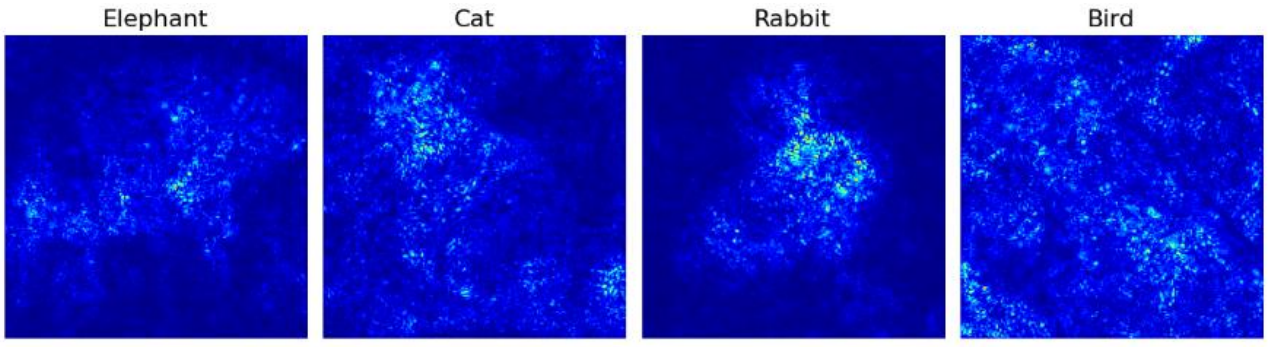
2 Ванильная значимость

Шаг 5. Saliency генерирует карту значимости, на которой отображаются области входного изображения, которые имеют наибольшее влияние на выходное значение (рис. 7-8).

In [8]:

```
%%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()
```

Рисунок 7 – Генерирует карту




```
CPU times: user 5.08 s, sys: 1.61 s, total: 6.69 s
Wall time: 12.2 s
```

Рисунок 8 – Карта значимости

3 SmoothGrad

Шаг 6. Карта значимости Vanilla слишком шумная, поэтому следует удалить шум на карте значимости с помощью SmoothGrad. SmoothGrad — это метод, который уменьшает шум на карте значимости путем добавления шума к входному изображению (рисунок 9).

In [9]:
```
%%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()
```

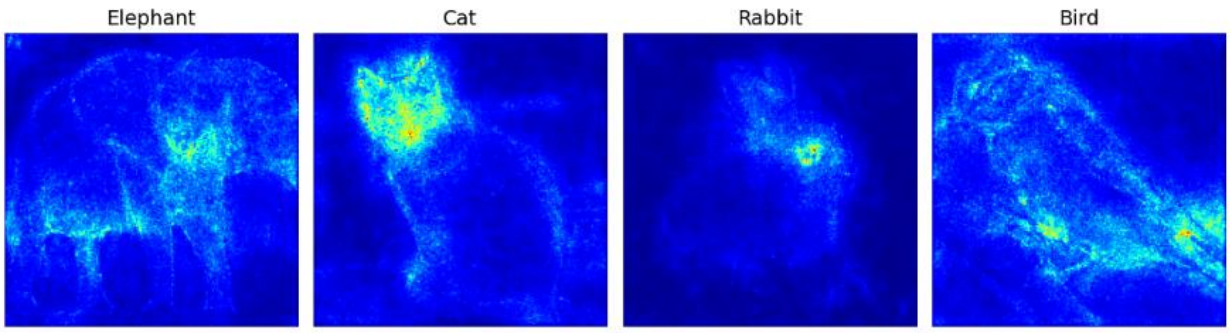


Рисунок 9 – Удаление шума с помощью SmoothGrad

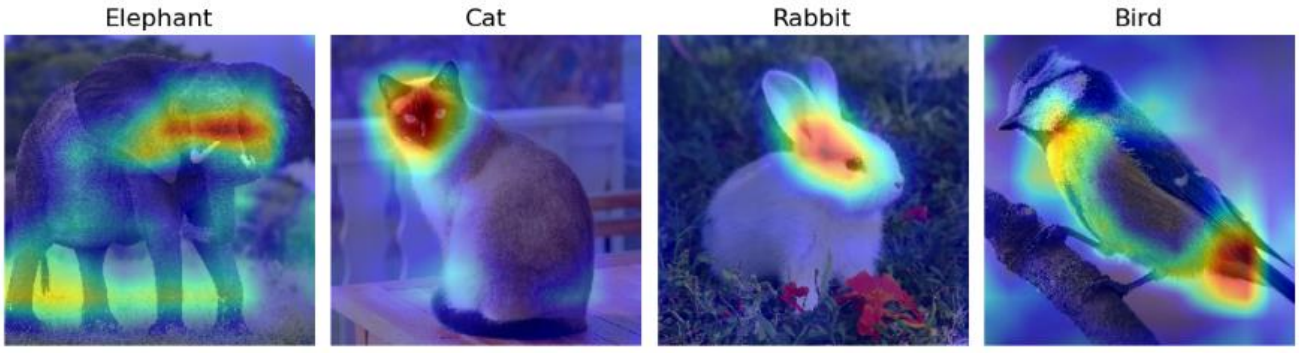
4 Визуализация тепловой карты - GradCAM

Шаг 7. Вместо использования градиентов выходных данных модели он использует выходные данные предпоследнего слоя (то есть сверточного слоя непосредственно перед плотными слоями). Функция визуализации GradCam получает 4 аргумента: model это изученная модель, X это пустой массив предварительно обработанных входных изображений, image_titles это соответствующие имена классов изображений в X, images это пустой массив исходных входных изображений (рис. 10-11).

In [10]:

```
%%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()
```

Рисунок 10 – Функция визуализации GradCam




```
CPU times: user 1.84 s, sys: 836 ms, total: 2.67 s
Wall time: 2.73 s
```

Рисунок 11 – Визуализация

Как видно, тепловые пятна не полностью покрывают цель на изображениях. В следующем шаге-методе, мы решим эту проблему. penultimate_layer. 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.

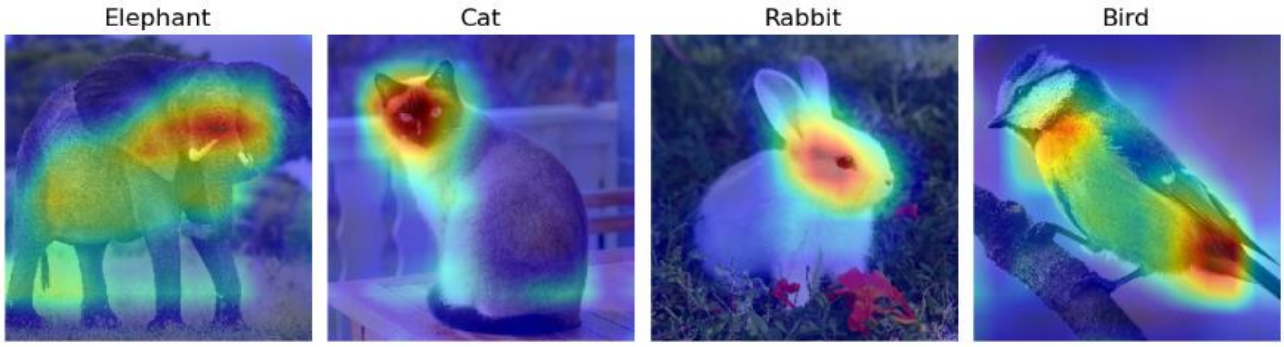
5 GradCAM++

Шаг 8. GradCam++ - улучшенная версия GradCam. Этот метод может обеспечить лучшее визуальное объяснение прогнозов модели CNN (рис. 12-13).

In [11]:

```
%%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()
```

Рисунок 12 – Использование GradCam++




```
CPU times: user 2.35 s, sys: 726 ms, total: 3.08 s
Wall time: 3.61 s
```

Рисунок 13 – Визуализация GradCam++

Заключение

В ходе выполнения данной работы по изучению защиты от атак на модели НС методом защитной дистилляции были выполнены следующие задачи:

- Задать нормалирующие преобразования для набора данных MNIST;
- Подготовить и обучить НС на базе фреймворка torch;
- Создать функции атак FGSM, I-FGSM, MI-FGSM и оценить их успешность;
- Создать два класса НС и переопределить функции обучения и тестирования;
- Создать функцию защиты методом дистилляции;
- Оценить результаты работы защищенных сетей.

Основная идея защитной дистиляции заключается в обучении устойчивой модели, путем передачи знаний от базовой модели, подверженной атакам, к новой модели, которая спроектирована для устойчивости к различным атакам.

Дистилляция дает более плоские локальные минимумы. Следовательно, небольшие изменения во входных данных с меньшей вероятностью изменят прогнозируемые значения.