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

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

Отчёт по лабораторной работе № 3

По дисциплине

«Анализ защищенности систем искусственного интеллекта»

Тема: «Использование механизмов внимания в нейронных сетях. Внимание в CHC VGG (карта значимости признаков и grad-CAM)»

Студент Невретдинов Руслан

Группа ББМО-01-22

Работу проверил

Спирин А.А.

Москва, 2023

Установка tf-keras-vis.

```
[ ] !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 1.4 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
```

Выполним импорт необходимых библиотек с общей настройкой GPU и Colab.

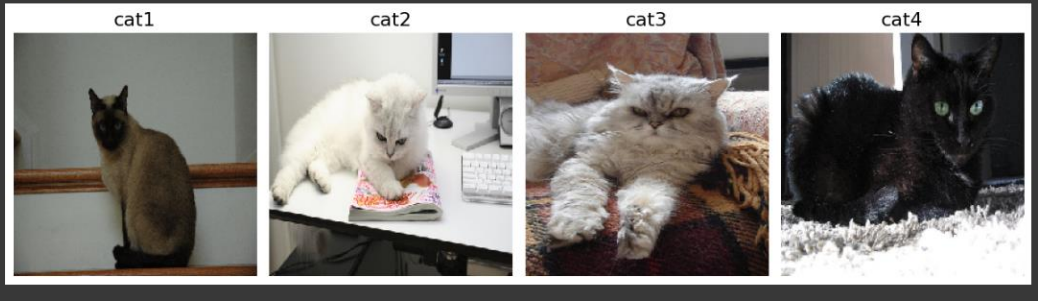
```
[ ] %reload_ext autoreload
    %autoreload 2
    import numpy as np
    from matplotlib import pyplot as plt
    %matplotlib inline
    import tensorflow as tf
    from tf_keras_vis.utils import num_of_gpus
    _, gpus = num_of_gpus()
    print('Tensorflow recognized {} GPUs'.format(gpus))
    from tensorflow.keras.preprocessing.image import load_img
    from tensorflow.keras.applications.vgg16 import preprocess_input
```

Далее происходит загрузка модели.

```
[ ] # Загрузка модели
    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 [==============================] - 3s 0us/step
    Model: "vgg16"
```

После чего, выполним загрузку изображений из ImageNet и выведем их.

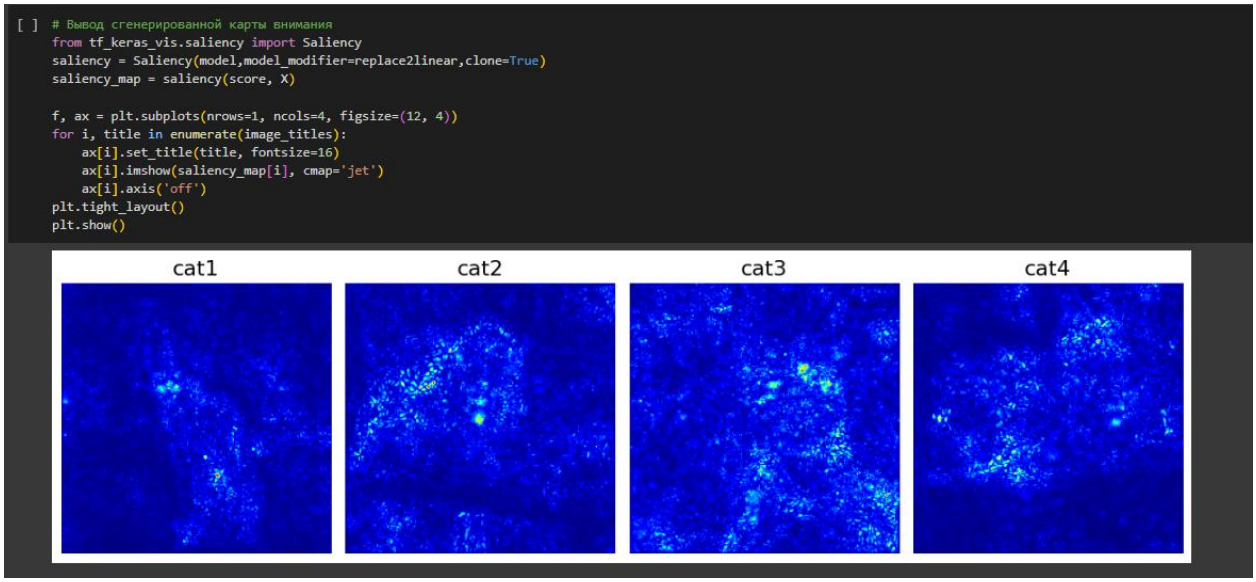
```
[ ] # Загрузка изображений из ImageNet
    image_titles = ['cat1', 'cat2', 'cat3', 'cat4']

    img0 = load_img('cat1.jpg', target_size=(224, 224))
    img1 = load_img('cat2.jpg', target_size=(224, 224))
    img2 = load_img('cat3.jpg', target_size=(224, 224))
    img3 = load_img('cat4.jpg', target_size=(224, 224))
    images = np.asarray([np.array(img0), np.array(img1), np.array(img2), np.array(img3)])

    X = preprocess_input(images)

    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()
```



Далее произведем замену на линейную функцию и создадим функции модификатора модели

```
[ ]  # Замена 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
     # Создание функции модификатора модели
     from tf_keras_vis.utils.scores import CategoricalScore
     score = CategoricalScore([41, 42, 62, 63])
     def score_function(output):
         return (output[0][41], output[1][42], output[2][62], output[3][63])
```

Выполним вывод сгенерированной карты внимания.



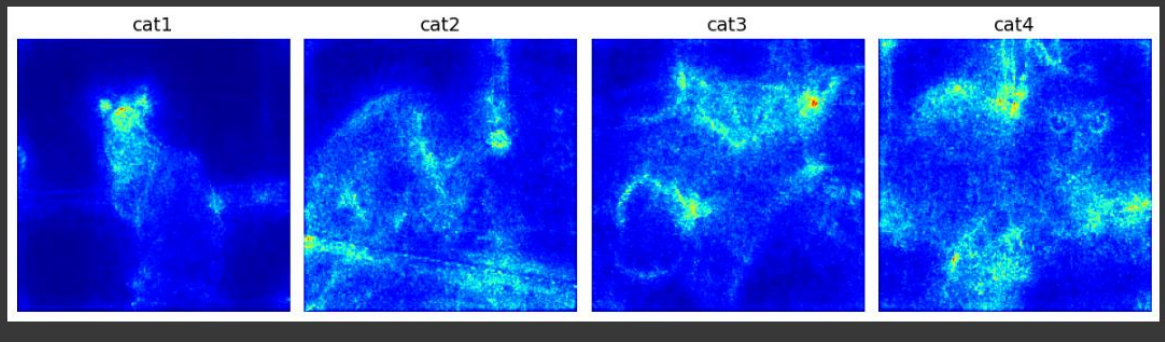
Использование данных карт позволяет увидеть исправления или устранения некоторых недостатков базовых методов saliency, таких как нежелательные пиксели или шум.

Принцип работы заключается в применении некоторой операции фильтрации или регуляризации к базовой карте saliency для уменьшения влияния шумовых компонентов и сглаживания результатов. Это может включать в себя использование различных методов фильтрации или функций активации.

Данная визуализация карт может помочь в интерпретации того, какие части изображений были ключевыми для принятия решений модели.

Далее выведем карту с уменьшением шума с помощью Smoothgrad

```
[ ] # Уменьшение шума с помощью SmoothGrad
    saliency_map = saliency(score,X,smooth_samples=20,smooth_noise=0.20)
    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()
```



SmoothGrad — это техника, предназначенная для сглаживания карт выделенности с целью снижения шума и повышения интерпретируемости. Подобные методы, как и предыдущий, широко используются в анализе вывода нейронных сетей, когда необходимо понять, какие части входных данных больше всего влияют на выход модели. Принцип работы метода SmoothGrad заключается в многократном добавлении небольших случайных шумов к входным данным и агрегации результатов для создания сглаженной карты выделенности. Это позволяет уменьшить влияние случайных шумов и сделать карту более устойчивой и интерпретируемой.

Следующим шагом будет построение карт значимости с использованием метода GradCam.

```
[ ] # Использование метода GrandCAM
    from matplotlib import cm
    from tf_keras_vis.gradcam import Gradcam

    gradcam = Gradcam(model,model_modifier=replace2linear,clone=True)
    cam = gradcam(score,X,penultimate_layer=-1)
    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()
```

Данный решение представляет собой метод визуализации активации нейронов в сверточных нейронных сетях, который позволяет понять, какие участки входного изображения были наиболее значимыми для принятия окончательного решения моделью. GradCam (Gradient-weighted Class Activation Mapping) помогает интерпретировать результаты классификации, выделяя области входного изображения, которые больше всего влияли на принятое решение.

Основная идея GradCam заключается в том, чтобы использовать градиенты, вычисленные по отношению к активациям последнего сверточного слоя модели, для создания взвешенной карты значимости. Эта карта подсвечивает области изображения, которые сильнее всего влияют на принадлежность к определенному классу.

Далее выполним переход к обновленной версии. Построение карт значимости с использованием метода GradCam++.

```
[ ] # Использование метода GrandCAM++
    from tf_keras_vis.gradcam_plus_plus import GradcamPlusPlus

    gradcam = GradcamPlusPlus(model,model_modifier=replace2linear,clone=True)
    cam = gradcam(score,X,penultimate_layer=-1)
    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()
```

Построение карт значимости с использованием метода GradCam++ представляет собой расширение метода GradCam (Gradient-weighted Class Activation Mapping) с добавлением учета вторых производных для улучшения интерпретации активаций нейронов в сверточных нейронных сетях. GradCam+++ учитывает не только градиенты первого порядка, но и градиенты второго порядка, что может привести к более точному выделению важных областей на изображении.

Таким образом, в результате сравнения данных методов удалось выделить несколько основных ключевых различий между ними:

GradCam хоть и является предыдущей версией, часто даёт хорошие результаты и легко справляется для большинства случаев, но может иметь тенденцию к размытию, особенно в случае сложных зависимостей между пикселями. Его «старшая» версия лишена этого недостатка и при правильной настройке может предоставлять более четкие и высокоуровневые карты значимости, но из-за этого является более вычислительно сложной.

Выводы по проделанной работе:

Работа с методами SmoothGrad, GradCAM, GradCAM++ и Saliency предоставляет комплексный инструментарий для визуализации и интерпретации действия сверточных нейронных сетей. Использование всех четырех методов в комбинации может предоставить более полное понимание работы модели. SmoothGrad помогает сгладить результаты и улучшить их интерпретируемость. GradCAM и GRADCAM++ позволяют визуализировать и углубиться в активации, учитывая различные аспекты зависимостей. Saliency больше подходит для общего обзора.

Выбор методов зависит от конкретных целей анализа и требований к интерпретации модели, а их комбинация может предоставить наиболее полную картину о том, как модель принимает решения.