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

ИКБ направление «Кибербразведка 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-02-22
Выполнил:
Давыдов И.Д.

Проверил:
Спирин А.А.

Москва 2024

Установка инструмента для визуализации для TensorFlow Keras.

```
[3] # установим инструмент визуализации для tensorflow keras
    !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 968.1 kB/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-k
    Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from imageio->tf-keras-vis) (1
    Installing collected packages: deprecated, tf-keras-vis
    Successfully installed deprecated-1.2.14 tf-keras-vis-0.8.6
```

Далее подключим необходимые библиотеки.

```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import cm
import tensorflow as tf
from tf_keras_vis.gradcam import Gradcam
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.applications.vgg16 import VGG16 as Model
from tf_keras_vis.utils.model_modifiers import ReplaceToLinear
from tf_keras_vis.utils.scores import CategoricalScore
from tf_keras_vis.saliency import Saliency
from tf_keras_vis.gradcam_plus_plus import GradcamPlusPlus
```

Загрузим предварительно обученную модель VGG16, на ImageNet датасете. После чего отобразим сводку по модели.

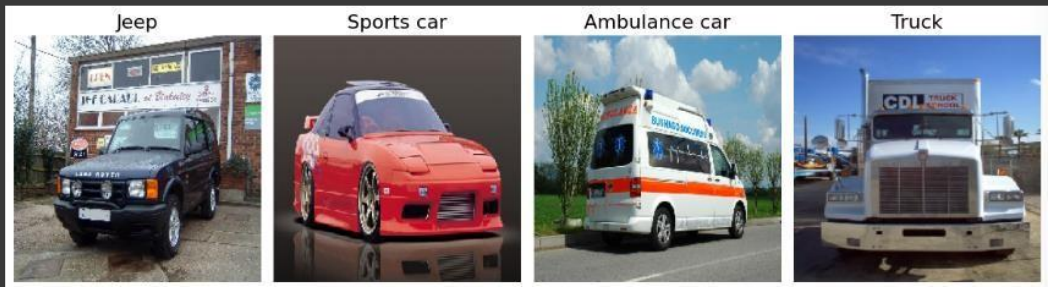
```
[5] # загружаем предварительно обученную модель VGG16 с весами, обученными
    # на ImageNet
    from tensorflow.keras.applications.vgg16 import VGG16 as Model
    model = Model(weights='imagenet', include_top=True)
    # и отобразим описание модели
    model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
553467096/553467096 [==============================] - 7s 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)
```

Загрузим несколько изображений датасета ImageNet и выполним их предварительную обработку перед использованием.

\
```
# загрузим несколько изображений и выполним их предварительную обработку
# перед использованием их в нашей модели
img = load_img('car.jpeg', target_size=(224, 224))
img1 = load_img('car1.jpeg', target_size=(224, 224))
img2 = load_img('car2.jpeg', target_size=(224, 224))
img3 = load_img('truck.jpeg', target_size=(224, 224))
images = np.asarray([np.array(img), np.array(img1), np.array(img2), np.array(img3)])
X = preprocess_input(images)
```

```
[21] # отобразим загруженные изображения
     img_titles = ['Jeep', 'Sports car', 'Ambulance car', 'Truck']
     f, ax = plt.subplots(nrows=1, ncols=4, figsize=(12, 4))
     for i, title in enumerate(img_titles):
         ax[i].set_title(title, fontsize=16)
         ax[i].imshow(images[i])
         ax[i].axis('off')
     plt.tight_layout()
     plt.savefig('original.png')
     plt.show()
```

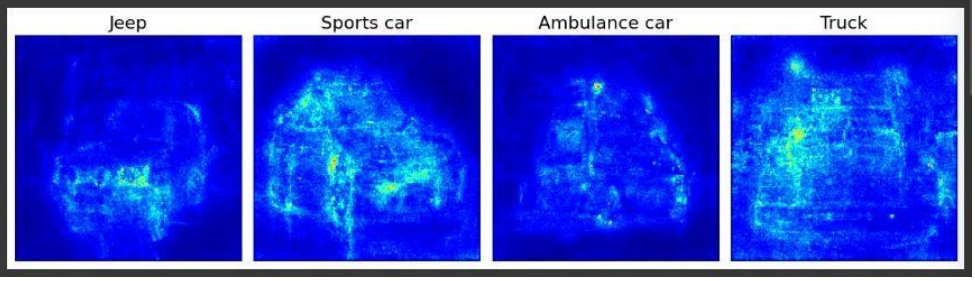


Далее создаем модификатор и функцию для использования с визуализацией из tf_keras_vis. Заменим активацию последнего слоя модели на линейную функцию, которая указывает классы для визуализации.

```
[8] # создаем модификатор модели и функцию оценки для исползьвания с
    # визуализацией tf_keras_vis
    # заменяем активацию последнего слоя модели на линейную функцию, которая
    # указывает классы для визуализации
    replace2linear = ReplaceToLinear()
    def model_modifier_function(cloned_model):
        cloned_model.layers[-1].activation = tf.keras.activations.linear
    score = CategoricalScore([609, 817, 407, 867])
    def score_function(output):
        return (output[0][609], output[1][817], output[2][407], output[3][867])
```

```
[9] # загрузим сайленси карты
    saliency = Saliency(model,model_modifier=replace2linear,clone=True)
    saliency_map = saliency(score, X)
```

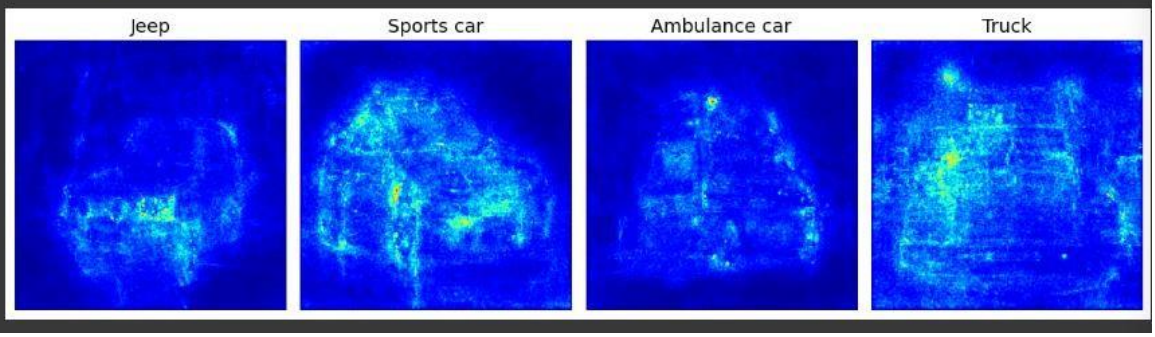
```
[20] # отобразим изображения с картой сайленси, чтобы увидеть, какие части
     # изображений вносят наибольший вклад в принятие решения
     f, ax = plt.subplots(nrows=1, ncols=4, figsize=(12, 4))
     for i, title in enumerate(img_titles):
         ax[i].set_title(title, fontsize=16)
         ax[i].imshow(saliency_map[i], cmap='jet')
         ax[i].axis('off')
     plt.tight_layout()
     plt.savefig('saliency.png')
     plt.show()
```



Применим метод SmoothGrad Для уменьшения шума на сайленси картах и отобразим получившиеся изображения, сохраним эту визуализацию.

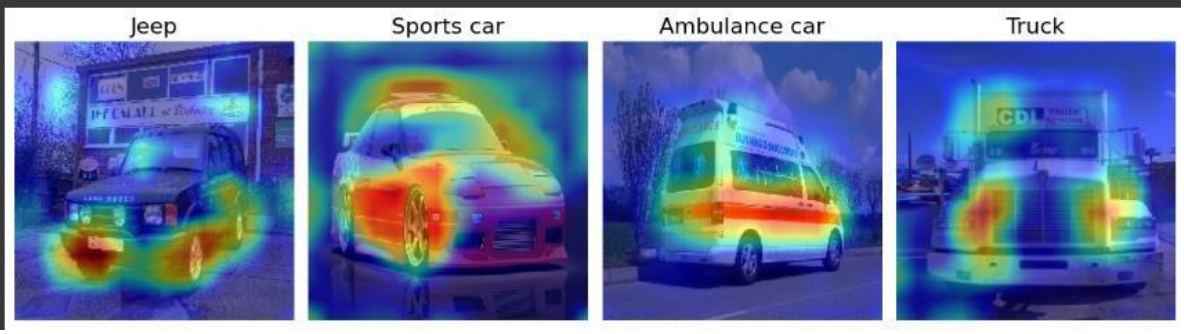
```
# теперь используем метод SmoothGrad Для уменьшения шума на кмайленси картах
saliency_map = saliency(score,X,smooth_samples=20,smooth_noise=0.20)
```

```
[14] # отобразим изображения и сохраним графическое представление в отдельное
     # изображение
     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(img_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('sgrad.png')
     plt.show()
```



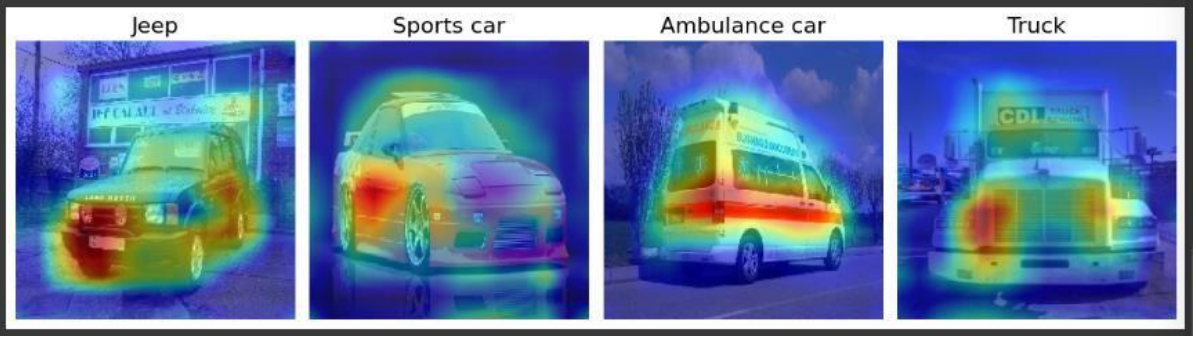
Далее применим метод Grad-CAM для визуализации активаций нейросети относительно заданных классов так мы опять же сможем увидеть области изображения, которые модель считает важными для определения классов.

```
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(img_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('gcam.png')
plt.show()
```



После этого используем улучшенную версию Grad-CAM - CradCAM++, который имеет улучшенный алгоритм для визуализации активаций. Этот метод позволяет более точно выделить области изображения, которые модель считает важными для определения классов. Также сохраним эту визуализацию.

```
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(img_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('gcampp.png')
plt.show()
```



Вывод

Применение методов визуализации, таких как Grad-CAM, Grad-CAM++, Saliency, SmoothGrad и аналогичные, представляет собой ценный инструмент для анализа, какие части изображений играют ключевую роль в принятии решений моделью машинного обучения. Методы Grad-CAM и Grad-CAM++ позволяют наглядно представить активации в различных областях изображений, что помогает определить, на какие участки фокусируется модель при определении классов. Карты активаций облегчают изучение воздействия объектов или участков изображения на решение модели. Применение метода SmoothGrad способствует снижению шума и придает картам сайленси большую интерпретируемость.