Лабораторная работа №3

Выполнил: Васильекв Григорий.

Группа: ББМО-02-23

1. Установка и проверка доступных GPU:

В этом блоке производится установка библиотеки tf-keras-vis, затем проверяется, сколько GPU доступно для работы с TensorFlow.

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

```
Collecting tf-keras-vis
  Downloading tf_keras_vis-0.8.7-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from tf-keras-vis) (1.13.1)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from tf-keras-vis) (11.0.0)
Requirement already satisfied: deprecated in /usr/local/lib/python3.10/dist-packages (from tf-keras-vis) (1.2.15)
Requirement already satisfied: imageio in /usr/local/lib/python3.10/dist-packages (from tf-keras-vis) (2.36.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tf-keras-vis) (24.2)
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.10/dist-packages (from deprecated->tf-keras-vis) (1.17.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from imageio->tf-keras-vis) (1.26.4)
Downloading tf_keras_vis-0.8.7-py3-none-any.whl (52 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 52.5/52.5 kB 2.5 MB/s eta 0:00:00
Installing collected packages: tf-keras-vis
Successfully installed tf-keras-vis-0.8.7
Tensorflow recognized 0 GPUs
```

2. Загрузка и подготовка модели VGG16:

В этом блоке загружается предобученная модель VGG16 с ImageNet и выводится ее сводка.

VGG16 — это архитектура сверточной нейронной сети (CNN), разработанная группой исследователей из Оксфордского университета в рамках работы над проектом Visual Geometry Group (VGG).

```
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"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102,764,544
fc2 (Dense)	(None, 4096)	16,781,312
predictions (Dense)	(None, 1000)	4,097,000

Total params: 138,357,544 (527.79 MB)
Trainable params: 138,357,544 (527.79 MB)
Non-trainable params: 0 (0.00 B)

3. Загрузка изображений:

Здесь загружаются 4 изображения (кот, лиса, кролик и собака) и они конвертируются в массивы numpy. Эти изображения будут использоваться для дальнейшего анализа.

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

# Заголовки наших изображений
image_titles = ['cat', 'fox', 'rabbit', 'rat']

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

4. Модификация модели для использования линейной активации:

В этом блоке создается функция, которая модифицирует последнюю активацию модели VGG16, чтобы она стала линейной. Это необходимо для работы с техникой визуализации, которая требует линейных выходов модели.

```
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
```

5. Определение функции для вычисления баллов для разных классов:

В этом блоке создается функция score_function, которая будет использоваться для выбора интересующих классов (например, кошка, лиса, кролик, мышь) при визуализации карт внимания.

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

```
score = CategoricalScore([285, 277, 330, 675])
# Где: 277 - лиса, 285 - кошка, 330 - кролик, 675 - мыш

# 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][285], output[1][277], output[2][330],
output[3][675])
```

6. Визуализация карты внимания с использованием Saliency:

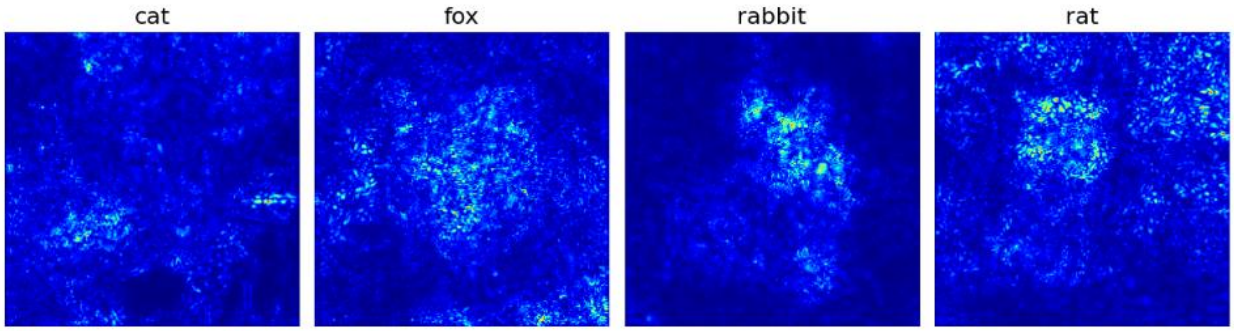
В этом блоке создается объект Saliency, который используется для генерации карты внимания для каждого из изображений. Карта отображает, какие части изображения наиболее важны для классификации.

```
%%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 19.3 s, sys: 4.45 s, total: 23.7 s
Wall time: 16.3 s
```

7. Визуализация карты внимания с сглаживанием:

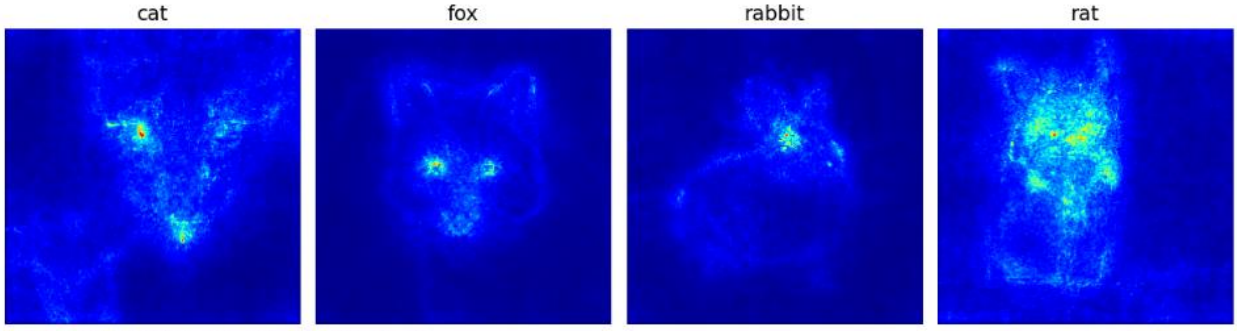
Этот блок использует функцию Saliency с параметрами сглаживания, чтобы уменьшить шум на карте внимания и улучшить визуализацию.

Функция Saliency — это метод визуализации, который позволяет анализировать, какие части изображения важны для принятия решения моделью.

```
%%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 5min 40s, sys: 13.2 s, total: 5min 54s
Wall time: 3min 45s
```

8. Визуализация с использованием GradCAM:

В этом блоке используется метод GradCAM для генерации тепловой карты, которая показывает, какие области изображения наиболее важны для классификации модели.

Метод GradCAM (Gradient-weighted Class Activation Mapping) — это метод визуализации, который используется для понимания того, какие части изображения имеют наибольшее влияние на решение модели. GradCAM позволяет выделить области, которые наиболее важны для принятия решения моделью, особенно в контексте сверточных нейронных сетей (CNN).

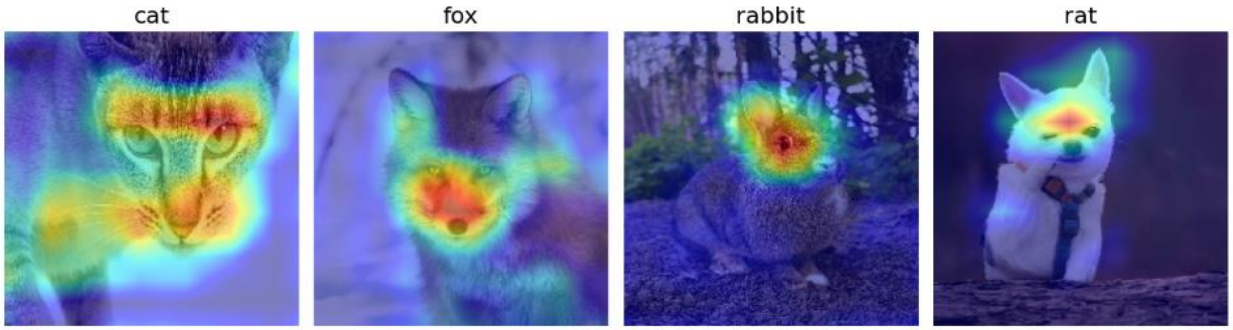
```
%%time

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 19.7 s, sys: 1.84 s, total: 21.6 s
Wall time: 13.4 s
```

9. Визуализация с использованием GradCAM++:

Здесь используется улучшенная версия GradCAM, называемая GradCAM++, для создания тепловых карт, которые могут давать более точные визуализации внимания модели.

GradCAM++ (Gradient-weighted Class Activation Mapping Plus Plus) — это усовершенствованная версия метода GradCAM, разработанная для улучшения качества визуализации и точности при анализе, какие части изображения влияют на предсказание модели. Основная цель GradCAM++ заключается в улучшении способности метода выделять более точные и высококонтрастные регионы изображения, которые наиболее значимы для классификации.

```
%%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 20 s, sys: 3.29 s, total: 23.3 s
Wall time: 15.1 s
```

Вывод:

Код демонстрирует использование различных методов визуализации внимания модели VGG16 с помощью библиотеки tf-keras-vis. Включены такие методы, как Saliency, GradCAM и GradCAM++, которые позволяют исследовать, какие части изображений оказывают наибольшее влияние на предсказания модели. Эти методы могут быть полезны для объяснения работы модели и повышения прозрачности нейронных сетей.