Lab 5 Concept-based Explainable AI

Teaching assistant: Eleonora Poeta (eleonora.poeta@polito.it)

Lab 5: Concept-based XAI - CRAFT

CRAFT

CRAFT is a novel post-hoc **concept-based method**.

It does the **automated extraction** of high-level **concepts** that neural networks have learned.

It has the following key attributes:

- **Multi-Layer Concept Extraction**: It allows for the extraction of concepts from *various locations within the model*, thus enabling the identification of the **most pertinent layer** for representing individual concepts.
- **Concept Importance Assessment**: CRAFT computes the **significance** of **individual concepts** concerning the model's predictions through the use of Sobol indices.
- **Concept Attribution Map**: It allows to backpropagate concept scores into the pixel space, leading to "concept attribution heatmaps" generation.

Quick Overview of CRAFT Approach

A CRAFT investigation involves the following steps:

1. **Input images selection**: Select a set of images ***X*** from the dataset where the model's predictions align with the class under investigation.
2. **Layer Selection**: Choose a layer to initiate the investigation, and split the model into 2 parts at this location: the 1st part ***g*** computes the activations of our input images, while the 2nd part ***h*** computes the logits.
   CRAFT will extract concepts from the activation space of this selected layer.
3. **CRAFT Fitting**:
   - **Crops Extraction**: CRAFT automatically **extracts random image crops** from the input dataset. This choice is motivated by the expectation that concepts are present in these crops, and can be subsequently dissected. CRAFT will operate on these crops to factorize the concepts. These crops are also used to visualize the concepts.
   - **Concept Activation Factorization**: In this phase, we use the 1st model ***g*** to compute the random crops activations, and then apply a Non-negative Matrix Factorization

(NMF) to decompose these positive activations into two matrices:

> - W constitutes a dictionary of Concept Activation Vectors (CAVs). It can be understood as a "concept basis" or "concept bank",
> - U represents the concept values, which are coefficients allowing to redefine the data points in our dataset according to the concept basis

4. **Concept Importance Estimation**: **Sensitivity** analysis is employed to estimate the global importance of each concept across the entire dataset.
    - **In general**, the **Sensitivity** measures the degree to which a model's output or predictions are influenced by changes or perturbations in specific features or concepts.
    - CRAFT uses Sobol indices to measure the contribution of each concept on the model's output.
    - Once this step is achieved, it is possible to compute the **contribution of each concept for a specific image**, thus capturing its local importance. CRAFT provides a variety of plotting functions to showcase the concepts and their respective importance.
5. Visualization of **Concept Map Attribution** : CRAFT has the ability to generate concept-wise attribution maps. These maps display the **part of an image that triggered the detection of the concept** by the network.

Exercise 1

To implement CRAFT you have to refer to **Xplique** library available on Github.

This library is composed of different modules:

- The *Concepts* module allows you to extract human concepts from a model and to test their usefulness with respect to a class.

Steps to follow:

0. Select the **GPU Runtime**.
1. Install the **Xplique library** running the given commands.
2. **Download the data**. You will download the class of rabbits. Run the given command
3. . **CRAFT** requires splitting the model in two parts. So, you have to prepare two functions (g, h) such that $f(x) = (g \cdot h)(x)$.

- g is the function that maps our input to the latent space (the penultimate layer of our model).

  > - As g you will use the **ResNet50** from `torchvision.models
  > - g will be `'input_to_latent'` part. We need to take the *first 8 layers* of the ResNet50.

- h is the function that maps from the layer before the classification head to the output.

  > - h will be `'latent_to_logit'` part. You will implement this classifier head (follow the instruction in the comments of the cell)

4. Define **transformation for the image** data from `torchvision.transforms` . You have to:
   - Transform the np array to a **PIL Image**
   - **Resize** the image up to 256
   - **CentreCrop** it to 256
   - Trasform the PIL Image to a **tensor**
   - **Normalize** the image with the given values
5. **Instanciate CRAFT**
6. **Fit** CRAFT
7. Calculate (global) **concept importances**
8. **Plot** some visualizations

Solution:

Imports

```
# Install Xplique
!pip install -q xplique
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 191.4/191.4 kB 1.5 MB/s eta 0:0
```

```
# Get some rabbit images
!curl -O https://share.deel.ai/s/fq78w4E2GTrQ54S/download && mv download rabbit.n
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curre
                                 Dload  Upload   Total   Spent    Left  Speed
100 43.0M  100 43.0M    0     0   645k      0  0:01:08  0:01:08 --:--:--  567
```

```
import torch
import torchvision.models as models
import torch.nn as nn
import numpy as np


# Check the device you are using is cuda
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device
```

```
device(type='cuda')
```

```
# Load a ResNet model from torchvision.models
model = models.resnet50(pretrained=True)
model = model.to(device)
print(model)
```

```
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fals
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_ru
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fals
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fals
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_ru
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=1000, bias=True)
)
```

```
# Cut the model in two part, g and h

# g is our 'input_to_latent'. For ResNet18 you have to take the first 4 layers
g = nn.Sequential(*(list(model.children())[:8]))


print(g)


# h is our 'latent_to_logit'

h = nn.Sequential(
    nn.AdaptiveAvgPool2d((1,1)), # AdaptiveAvgPool2d(1,1)
    nn.Flatten(),
    nn.Dropout(p=0.0, inplace=False), # Dropout with p=0.0
    nn.Linear(in_features=2048, out_features=1000, bias=True), # Linear with in_f
    nn.Identity(), # Identity
  )
h = h.to(device)


print(h)
```

```
Sequential(
  (0): AdaptiveAvgPool2d(output_size=(1, 1))
  (1): Flatten(start_dim=1, end_dim=-1)
  (2): Dropout(p=0.0, inplace=False)
  (3): Linear(in_features=2048, out_features=1000, bias=True)
  (4): Identity()
)
```

```
# Define tranformation for our image data
import torchvision.transforms as transforms

to_pil = transforms.ToPILImage()

transf = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize(256),
    transforms.CenterCrop(256),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )

])


rabbit_class = 330 # imagenet class for rabbit

# loading some images of rabbits !
images = np.load('rabbit.npz')['arr_0'].astype(np.uint8)

images_preprocessed = torch.stack([transf(img) for img in images], 0)

images_preprocessed = images_preprocessed.to(device)

images_preprocessed.shape
```

```
torch.Size([300, 3, 256, 256])
```

Fitting CRAFT: Patches Creation and Concepts Factorization

```
from xplique.concepts import CraftTorch as Craft
from xplique.concepts import DisplayImportancesOrder

# Instanciate CRAFT
craft = Craft(input_to_latent_model = g,
              latent_to_logit_model = h,
              number_of_concepts = 10,
              patch_size = 80,
              batch_size = 64,
              device = device)


# now we can start fit the concept using our rabbit images
# CRAFT will:
# 1. Create the patches
# 2. Find the concepts
# 3. Return the crops (crops), the embedding of the crops (crops_u), and the conc
crops, crops_u, concept_bank_w = craft.fit(images_preprocessed, class_id=rabbit_c

crops.shape, crops_u.shape, concept_bank_w.shape
```

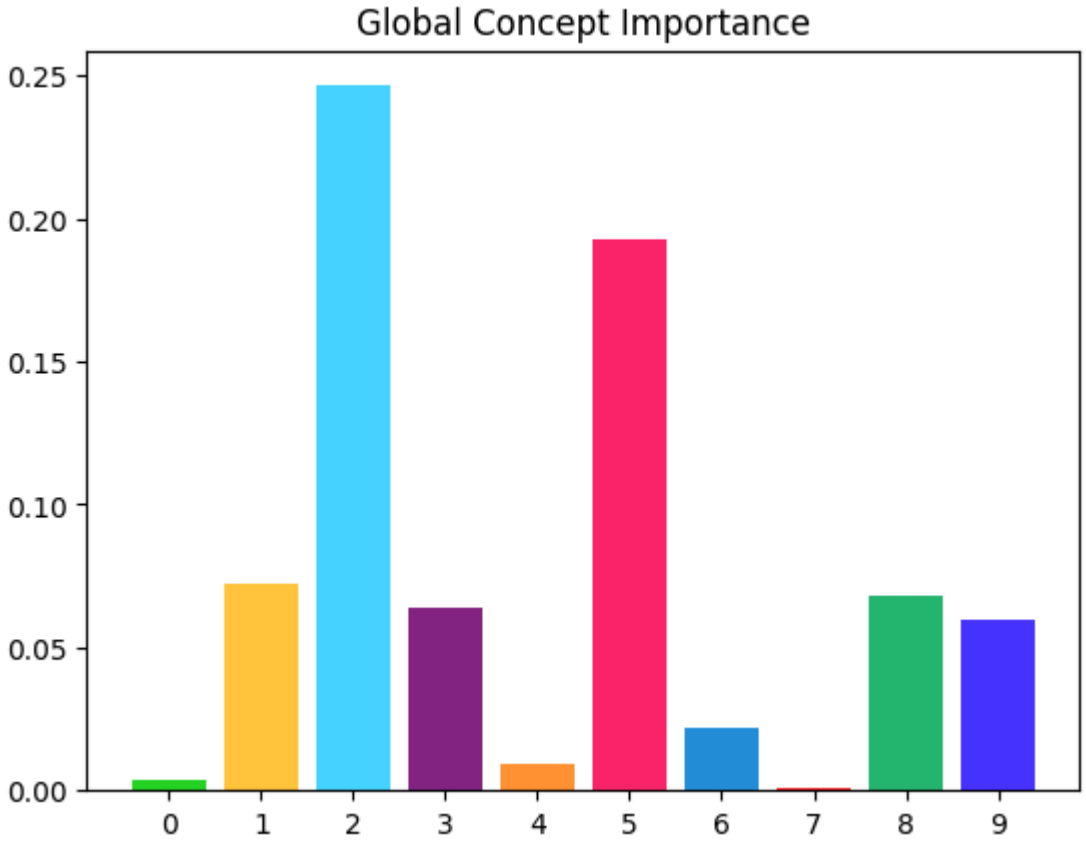
```
/usr/local/lib/python3.10/dist-packages/sklearn/decomposition/_nmf.py:1665: C
  warnings.warn(
((2700, 3, 80, 80), (2700, 10), (10, 2048))
```

```
importances = craft.estimate_importance()
```

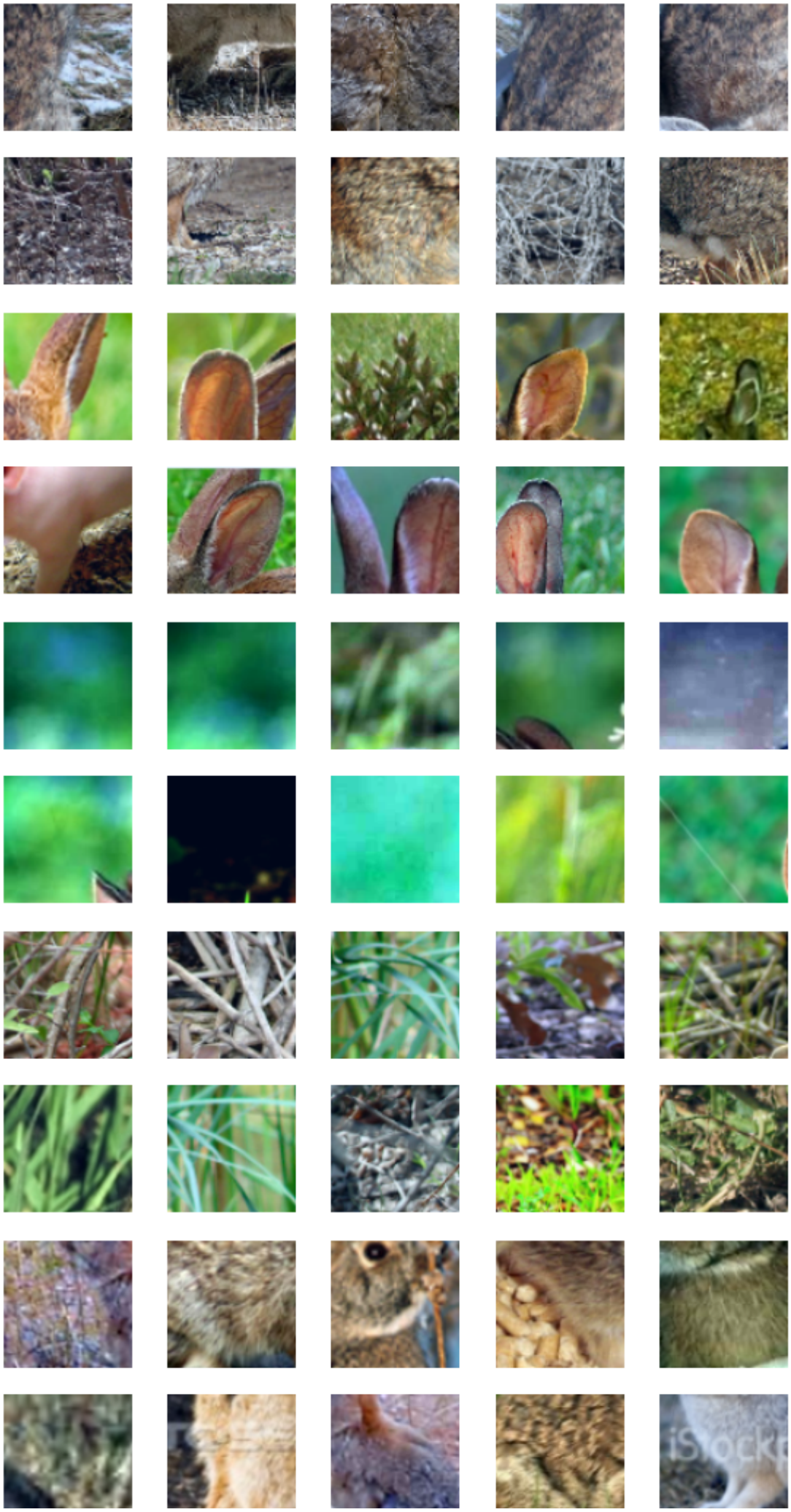
```
# Plot the attribution map for 1 image, and the concepts associated to
# the class around it, ordered by global importance
craft.plot_image_concepts(images_preprocessed[21].cpu())
```


```
# Let's see which concepts matter

# Plot the importance values.
craft.plot_concepts_importances()
```




```
# Let's inspect those concepts by showing the 10 best crops for each concept.
# Limit the display to the 5 most important concepts.
craft.plot_concepts_crops(nb_crops=10, nb_most_important_concepts=5)
```



```
# Display the attribution map for 20 images.
# Limit the display to the 5 most important concepts.
craft.plot_concept_attribution_maps(images_preprocessed[0:20].cpu(),
                                    nb_most_important_concepts=5)
```