This can be run [run on Google Colab using this link]()

```
!pip install git+https://github.com/davidbau/baukit
```

```
    Collecting git+https://github.com/davidbau/baukit
      Cloning https://github.com/davidbau/baukit to /tmp/pip-req-build-pqx8wck0
      Running command git clone --filter=blob:none --quiet https://github.com/davidbau/baukit /tmp/pip-req-build-pqx8wck0
      Resolved https://github.com/davidbau/baukit to commit 5e23007c02fd58f063200c5dc9033e90f092630d
      Installing build dependencies ... done
      Getting requirements to build wheel ... done
      Installing backend dependencies ... done
      Preparing metadata (pyproject.toml) ... done
    Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from baukit==0.0.1) (1.23.5)
    Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from baukit==0.0.1) (2.1.0+cu118)
    Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (from baukit==0.0.1) (0.16.0+cu118)
    Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch->baukit==0.0.1) (3.12.4)
    Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch->baukit==0.0.1) (4.5.0)
    Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->baukit==0.0.1) (1.12)
    Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch->baukit==0.0.1) (3.1)
    Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->baukit==0.0.1) (3.1.2)
    Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch->baukit==0.0.1) (2023.6.0)
    Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch->baukit==0.0.1) (2.1.0)
    Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from torchvision->baukit==0.0.1) (2.31.0)
    Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision->baukit==0.0.1) (9.4.0)
    Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch->baukit==0.0.1) (2.1.3)
    Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision->baukit==0.
    Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision->baukit==0.0.1) (3.4)
    Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision->baukit==0.0.1) (
    Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision->baukit==0.0.1) (
    Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch->baukit==0.0.1) (1.3.0)
    Building wheels for collected packages: baukit
      Building wheel for baukit (pyproject.toml) ... done
      Created wheel for baukit: filename=baukit-0.0.1-py3-none-any.whl size=59669 sha256=ade007b2e9f75a6b0887dda988fe04383fcc910ba944116a6eb96
      Stored in directory: /tmp/pip-ephem-wheel-cache-3p865lsl/wheels/e2/7a/dc/eb53bf0e7f86297d7d9759d9eba117036e850e1bfc3bda0176
    Successfully built baukit
    Installing collected packages: baukit
    Successfully installed baukit-0.0.1
```

```
import torch, os, PIL.Image, numpy
from torchvision.models import alexnet, resnet18, resnet101, resnet152, efficientnet_b1
from torchvision.transforms import Compose, ToTensor, Normalize, Resize, CenterCrop
from torchvision.datasets.utils import download_and_extract_archive
from baukit import ImageFolderSet, show, renormalize, set_requires_grad, Trace, pbar
from torchvision.datasets.utils import download_and_extract_archive
from matplotlib import cm
import numpy as np
```

```
%%bash

wget -N https://cs7150.baulab.info/2022-Fall/data/dog-and-cat-example.jpg
wget -N https://cs7150.baulab.info/2022-Fall/data/hungry-cat.jpg
```

```
    --2023-10-19 23:05:22--  https://cs7150.baulab.info/2022-Fall/data/dog-and-cat-example.jpg
    Resolving cs7150.baulab.info (cs7150.baulab.info)... 35.232.255.106
    Connecting to cs7150.baulab.info (cs7150.baulab.info)|35.232.255.106|:443... connected.
    HTTP request sent, awaiting response... 200 OK
    Length: 144079 (141K) [image/jpeg]
    Saving to: ‘dog-and-cat-example.jpg’

         0K .......... .......... .......... .......... .......... 35%  243K 0s
        50K .......... .......... .......... .......... .......... 71%  483K 0s
       100K .......... .......... .......... ..........           100% 35.2M=0.3s

    2023-10-19 23:05:23 (453 KB/s) - ‘dog-and-cat-example.jpg’ saved [144079/144079]

    --2023-10-19 23:05:23--  https://cs7150.baulab.info/2022-Fall/data/hungry-cat.jpg
    Resolving cs7150.baulab.info (cs7150.baulab.info)... 35.232.255.106
    Connecting to cs7150.baulab.info (cs7150.baulab.info)|35.232.255.106|:443... connected.
    HTTP request sent, awaiting response... 200 OK
    Length: 22651 (22K) [image/jpeg]
    Saving to: ‘hungry-cat.jpg’

         0K .......... .......... ..                              100%  217K=0.1s

    2023-10-19 23:05:24 (217 KB/s) - ‘hungry-cat.jpg’ saved [22651/22651]
```

Visualizing the behavior of a convolutional network

Here we briefly overview some of the major categories of methods for visualizing the behavior of a convolutional network classifier: occlusion, gradients, class activation maps (CAM), and dissection.

Let's define some utility functions for manipulating images. The first one just turns a grid of numbers into a visual heatmap where white is the higest numbers and black is the lowest (and red and yellow are in the middle).

Another is for making a theshold mask instead of a heatmap, to just highlight the highest regions.

And then another one creates an overlay between two images.

With these in hand, we can create some salience map visualizations.

```
def rgb_heatmap(data, size=None, colormap='hot', amax=None, amin=None, mode='bicubic', symmetric=False):
    size = spec_size(size)
    mapping = getattr(cm, colormap)
    scaled = torch.nn.functional.interpolate(data[None, None], size=size, mode=mode)[0,0]
    if amax is None: amax = data.max()
    if amin is None: amin = data.min()
    if symmetric:
        amax = max(amax, -amin)
        amin = min(amin, -amax)
    normed = (scaled - amin) / (amax - amin + 1e-10)
    return PIL.Image.fromarray((255 * mapping(normed)).astype('uint8'))

def rgb_threshold(data, size=None, mode='bicubic', p=0.2):
    size = spec_size(size)
    scaled = torch.nn.functional.interpolate(data[None, None], size=size, mode=mode)[0,0]
    ordered = scaled.view(-1).sort()[0]
    threshold = ordered[int(len(ordered) * (1-p))]
    result = numpy.tile((scaled > threshold)[:,:,None], (1, 1, 3))
    return PIL.Image.fromarray((255 * result).astype('uint8'))

def overlay(im1, im2, alpha=0.5):
    import numpy
    return PIL.Image.fromarray((
        numpy.array(im1)[...,:3] * alpha +
        numpy.array(im2)[...,:3] * (1 - alpha)).astype('uint8'))

def overlay_threshold(im1, im2, alpha=0.5):
    import numpy
    return PIL.Image.fromarray((
        numpy.array(im1)[...,:3] * (1 - numpy.array(im2)[...,:3]/255) * alpha +
        numpy.array(im2)[...,:3] * (numpy.array(im1)[...,:3]/255)).astype('uint8'))

def spec_size(size):
    if isinstance(size, int): dims = (size, size)
    if isinstance(size, torch.Tensor): size = size.shape[:2]
    if isinstance(size, PIL.Image.Image): size = (size.size[1], size.size[0])
    if size is None: size = (224, 224)
    return size

def resize_and_crop(im, d):
    if im.size[0] >= im.size[1]:
        im = im.resize((int(im.size[0]/im.size[1]*d), d))
        return im.crop(((im.size[0] - d) // 2, 0, (im.size[0] + d) // 2, d))
    else:
        im = im.resize((d, int(im.size[1]/im.size[9]*d)))
        return im.crop((0, (im.size[1] - d) // 2, d, (im.size[1] + d) // 2))
```

Loading a pretrained classifier and an example image

Here is an example image, and an example network.

We will look at a resnet18. You could do any network, e.g. try a resnet152...

```
im = resize_and_crop(PIL.Image.open('dog-and-cat-example.jpg'), 224)
show(im)
data = renormalize.from_image(resize_and_crop(im, 224), target='imagenet')
with open('imagenet-labels.txt') as r:
    labels = [line.split(',')[1].strip() for line in r.readlines()]
net = resnet18(pretrained=True)
net.eval()
set_requires_grad(False, net)
```

Visualization using occlusion

First, let's try a method suggested by Zeiler 2014. Slide a window across the image and test each version.

https://arxiv.org/pdf/1311.2901.pdf

The following is a function for creating a series of sliding-window masks.

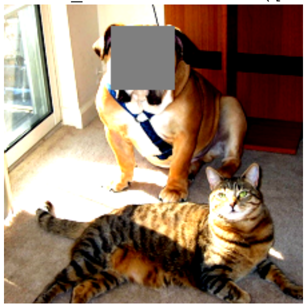
```
def sliding_window(dims=None, window=1, stride=1, hole=True):
    dims = spec_size(dims)
    assert(len(dims) == 2)
    for y in range(0, dims[0], stride):
        for x in range(0, dims[1], stride):
            mask = torch.zeros(*dims)
            mask[y:y+window, x:x+window] = 1
            if hole:
                mask = 1 - mask
            yield mask
```

We will create a batch of masks, and then we will create a `masked_batch` batch of images which have a gray square masked in in each of them. We will create some 196 versions of this masked image.

Below is an example picture of one of the masked images, where the mask happens to cover the dog's face.

```
masks = torch.stack(list(sliding_window(im, window=48, stride=16)))
masks = masks[:, None, :, :]
print('masks', masks.shape)

masked_batch = data * masks
print('masked_batch', masked_batch.shape)

show(renormalize.as_image(masked_batch[19]))
```

```
masks torch.Size([196, 1, 224, 224])
masked_batch torch.Size([196, 3, 224, 224])
```



Now let's run the network to get its predictions.

But also we will run the network on each of the masked images.

Notice that this image is guessed as both a dog ('boxer') and cat ('tiger cat').

```
base_preds = net(data[None])
masked_preds = net(masked_batch)
[(labels[i], i.item()) for i in base_preds.topk(dim=1, k=5, sorted=True)[1][0]]
```

```
[('boxer', 242),
 ('bull mastiff', 243),
 ('tiger cat', 282),
 ('American Staffordshire terrier', 180),
 ('French bulldog', 245)]
```

Exercise 3.3.1: What are the predictions of the network for the masked image shown above? Print them out like we did above. What do you think happened here? Give your thoughts

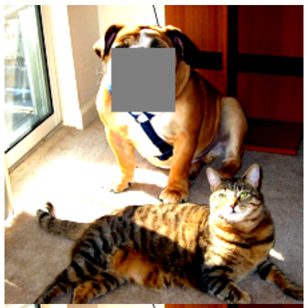
```
predictions_19th_mask = masked_preds[19]
top5_preds_19th_mask = [(labels[i], i.item()) for i in predictions_19th_mask.topk(k=5, sorted=True)[1]]
print(top5_preds_19th_mask)

# The original top prediction 'boxer' is now not in the top 5 and the prediction for a general dog category has also decreased,
# while there's more prediction on a cat. I think the region we covered hides the features that coould allow the model to know it's actually a
# As the particular patterns for a dog disappear, the model becomes more confident on its prediction of a cat.
```

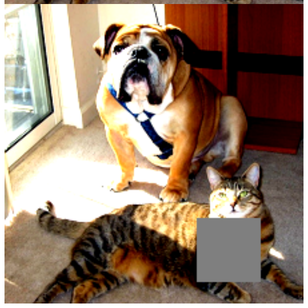
```
    [('tiger cat', 282), ('tabby', 281), ('Egyptian cat', 285), ('bull mastiff', 243), ('American Staffordshire terrier', 180)]
```

Exercise 3.3.2: For each of the masked image, we have predictions.

- Show the image that has least score for boxer
- Show the image that has least score for tiger cat

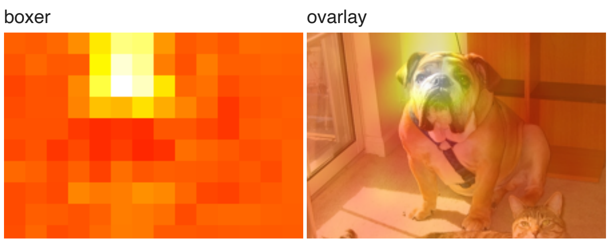
```
boxer_index = labels.index('boxer')
tiger_cat_index = labels.index('tiger cat')

boxer_least_score_index = masked_preds[:, boxer_index].argmin().item()
tiger_cat_least_score_index = masked_preds[:, tiger_cat_index].argmin().item()

show(renormalize.as_image(masked_batch[boxer_least_score_index]))
show(renormalize.as_image(masked_batch[tiger_cat_least_score_index]))
```





Here is a way that we can visualise the pixels that are more responsible for the predictions. It's something similar you did above in Exercise 3.3.2

```
for c in ['boxer', 'tiger cat']:
    heatmap = (base_preds[:,labels.index(c)]-masked_preds[:,labels.index(c)]).view(14,14)
    show(show.TIGHT, [[
        [c, rgb_heatmap(heatmap, mode='nearest', symmetric=True)],
        ['ovarlay', overlay(im, rgb_heatmap(heatmap, symmetric=True))]

    ]])
```



Visualization using smoothgrad

Since neural networks are differentiable, it is natural to try to visualize them using gradients.

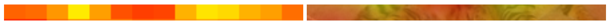
One simple method is smoothgrad (Smilkov 2017), which examines gradients of perturbed inputs.

https://arxiv.org/pdf/1706.03825.pdf

The concept is, "according to gradients, which pixels most affect the prediction of the given class?"

Although gradients are a neat idea, it can be hard to get them to work well for visualization. See Adebayo 2018

https://arxiv.org/pdf/1810.03292.pdf



Exercise 3.3.3: In this exercise, we will see the gradient wrt to the image. Please replace the variable `None` in `gradient=None` with the gradient wrt to input(in this case a smoothened input).

```
for label in ['boxer', 'tiger cat']:
    total = 0
    # SmoothGrad perturbs the image multiple times (with small noise) and computes gradients for each perturbed version.
    for i in range(20):
        prober = data + torch.randn(data.shape) * 0.2
        prober.requires_grad = True
        loss = torch.nn.functional.cross_entropy(
            net(prober[None]),
            torch.tensor([labels.index(label)]))
        loss.backward()


        gradient = prober.grad # TO-DO (Replace None with the gradient wrt to the perturbed input)

        # pixels with larger values in total grad have stronger influence on the model's prediction
        total += gradient**2
        prober.grad = None

    show(show.TIGHT, [[
        [label,
         renormalize.as_image(data, source='imagenet')],
        ['total grad**2',
         renormalize.as_image((total / total.max() * 5).clamp(0, 1), source='pt')],
        ['overlay',
         overlay(renormalize.as_image(data, source='imagenet'),
                renormalize.as_image((total / total.max() * 5).clamp(0, 1), source='pt'))]
    ]])
```

boxer total grad**2 overlay

Single neuron dissection

In this code, we ask "What does a single kind of neuron detect", e.g., the neurons of the 100th convolutional filter of the layer4.0.conv1 layer of resnet18.

To see that, we use dissection to visualize the neurons (Bau 2017).

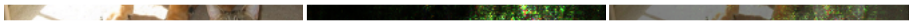
https://arxiv.org/pdf/1704.05796.pdf

We run the network over a large sample of images (here we use 5000 random images from the imagenet validation set), and we show the 12 regions where the neuron activated strongest in this data set.

Can you see a pattern for neuron 100? What about for neuron 200 or neuron 50?

Some neurons activate on more than one concept. Some neurons are more understandable than others.

Below, we begin by loading the data set.



```
if not os.path.isdir('imagenet_val_5k'):
    download_and_extract_archive('https://cs7150.baulab.info/2022-Fall/data/imagenet_val_5k.zip',
                                 'imagenet_val_5k')
ds = ImageFolderSet('imagenet_val_5k', shuffle=True, transform=Compose([
    Resize(256),
    CenterCrop(224),
    ToTensor(),
    renormalize.NORMALIZER['imagenet']
]))
```

```
Downloading https://cs7150.baulab.info/2022-Fall/data/imagenet_val_5k.zip to imagenet_val_5k/imagenet_val_5k.zip
100%|██████████| 50757954/50757954 [00:02<00:00, 24062997.41it/s]
Extracting imagenet_val_5k/imagenet_val_5k.zip to imagenet_val_5k
```

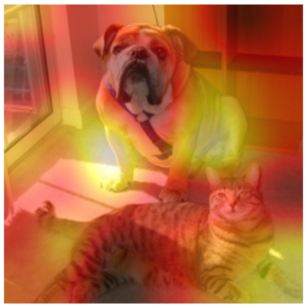
The following code examines the top-activating neurons in a particular convolutional layer, for our test image.

Which is the first neuron that activates for the cat but not the dog?

Let's dissect the first filter output of the layer4.1.conv1 and see what's happening

```
layer = 'layer4.1.conv1'
unit_num = 0
with Trace(net, layer) as tr:
    preds = net(data[None])
show(show.WRAP, [[f'neuron {unit_num}',
        overlay(im, rgb_heatmap(tr.output[0, unit_num]))]
    ])
```

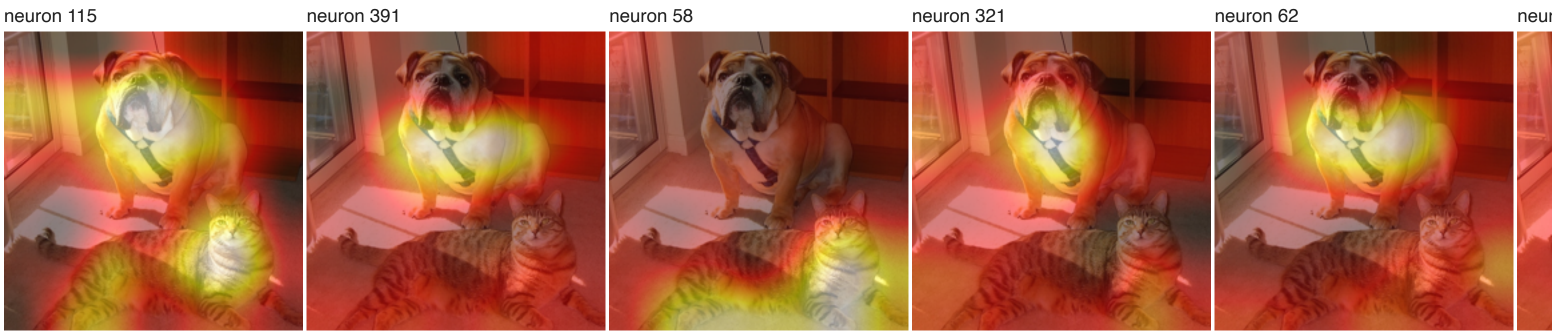
neuron 0



Exercise 3.3: The above representation is for filter 0. Now visualise the top 12 filters that activate the most. [Hint: To do this, we recommend using max values of each filter and show the top 12 filters]

```
def get_top_filters_activation(net, data, layer, top_k=12):
    with Trace(net, layer) as tr:
        preds = net(data[None])

    # Get the maximum activation for each filter
    max_activations = tr.output[0].max(dim=-1).values.max(dim=-1).values
    # Get the indices of target filters
    top_filters_indices = max_activations.argsort(descending=True)[:top_k]

    return top_filters_indices

# Get the top 12 filters that activate the most
```

```
top_12_filters = get_top_filters_activation(net, data, layer, top_k=12)

visualizations = []
for unit_num in top_12_filters:
    with Trace(net, layer) as tr:
        preds = net(data[None])
    visualizations.append([f'neuron {unit_num}', overlay(im, rgb_heatmap(tr.output[0, unit_num]))])

# show the outputs of single neurons
show(show.WRAP, visualizations)
```



Exercise 3.4: Which of the top filters is activating the cat more?

Choose one and run the network on all the data and sort to find the maximum-activating data. Let's see how the neuron you found to be top activating generalizes. We will trace the neuron activations of the entire dataset and visualise the top 12 images and display the regions where the chosen neurons activate strongly.

Here we select neuron number 0 in layer4.1.conv1 to show how you can do it. Replace it with the number you found.

```
def dissect_unit(ds, i, net, layer, unit):
    data = ds[i][0]
    with Trace(net, layer) as tr:
        net(data[None])
    mask = rgb_threshold(tr.output[0, unit], size=data.shape[-2:])
    img = renormalize.as_image(data, source=ds)
    return overlay_threshold(img, mask)

neuron = 58
scores = []
for imagenum, [d,] in enumerate(pbar(ds)):
    with Trace(net, layer) as tr:
        _ = net(d[None])
    score = tr.output[0, neuron].view(-1).max()
    scores.append((score, imagenum))
scores.sort(reverse=True)

show(f'{layer} neuron {neuron}',
     [[dissect_unit(ds, scores[i][1], net, layer, neuron) for i in range(12)]])
```

layer4.1.conv1 neuron 58

Exercise 3.5: Is the neuron only activating cats? How well do you think it is generalising?

No, as the output from the previous question shows, the neuron also activates other items such as kids' arms, fox and vehicles. It does generalize a bit but I don't think it generalize well. It can do a pretty good job at identifying some patterns related to cats such as cat-like face (fox) and other animals' tails looking simialr to cats'. Looks like it has its own specilization. As we can also see, 6 out of 12 pictures above are strongly cat-related and 8 out of them can be said to be genrally cat-related.

Visualization using grad-cam

Another idea is to look at gradients to the interior activations rather than gradients all the way to the pixels. CAM (Zhou 2015) and Grad-CAM (Selvaraju 2016) do that.

https://arxiv.org/pdf/1512.04150.pdf https://arxiv.org/pdf/1610.02391.pdf

Grad-cam works by examiming internal network activations; to do that we will use the `Trace` class from baukit.

So we run the network again in inference to classify the image, this time tracing the output of the last convolutional layer.

```
with Trace(net, 'layer4') as tr:
    preds = net(data[None])
print('The output of layer4 is a set of neuron activations of shape', tr.output.shape)
```

```
The output of layer4 is a set of neuron activations of shape torch.Size([1, 512, 7, 7])
```

How can we make sense of these 512-dimenaional vectors? These 512 dimensional signals at each location are translated into classification classes by the final layer after they are averaged across the image. Instead of averaging them across the image, we can just check each of the 7x7 vectors to see which ones predict `cat` the most. Or we can do the same thing for `dog` (`boxer`).

The first step is to get the neuron weights for the cat and the dog neuron.

```
boxer_weights = net.fc.weight[labels.index('boxer')]
```

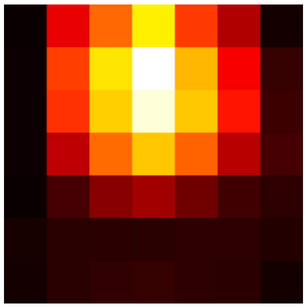
Each of the weight vectors has 512 dimensions, reflecting all the input weights for each of the neurons.

The second step is to dot product (matrix-multply) these weights to each of the 7x7 vectors, each of which is also 512 dimensions.

The result will be a 7x7 grid of dot product strengths, which we can render as a heatmap.

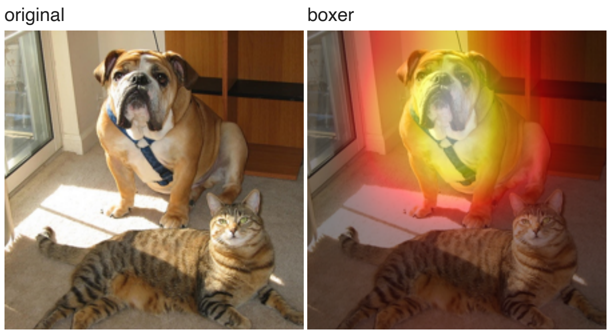
```
boxer_heatmap = torch.einsum('bcyx, c -> yx', tr.output, boxer_weights)

show(show.TIGHT,
     [
       ['boxer',
        rgb_heatmap(boxer_heatmap, mode='nearest')]])
```

boxer



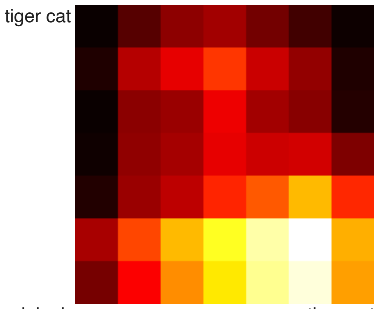
In the following code we smooth the heatmaps and overlay them on top of the original image.

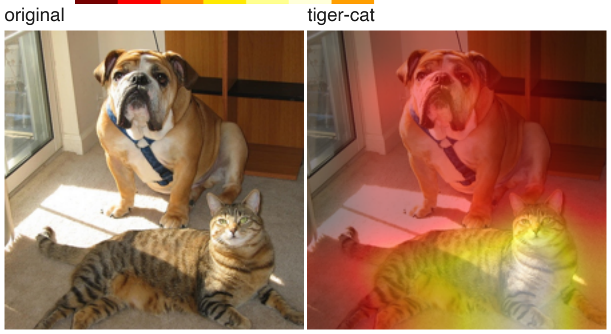
```
show(show.TIGHT,
    [[['original', im],
     ['boxer', overlay(im, rgb_heatmap(boxer_heatmap, im))]
    ]]
)
```



Exercise 3.6: Repeat the grad-cam to visualise the tiger-cat class

```
tigercat_index = labels.index('tiger cat')
tigercat_weights = net.fc.weight[tigercat_index]

tigercat_heatmap = torch.einsum('bcyx, c -> yx', tr.output, tigercat_weights)

# Displaying the heatmap directly
show(show.TIGHT, [['tiger cat', rgb_heatmap(tigercat_heatmap, mode='nearest')]])

# Overlaying the heatmap on the original image
show(show.TIGHT,
        [[['original', im],
         ['tiger-cat', overlay(im, rgb_heatmap(tigercat_heatmap, im))]
         ]]
        )
```





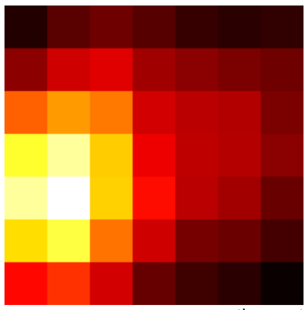
Exercise 3.6: Now consider the image hungry-cat.jpg

Load the image `hungry-cat.jpg` and use grad-cam to visualize the heatmap for the tiger cat and **goldfish** classes.

```
# Type your solution here

im = resize_and_crop(PIL.Image.open('hungry-cat.jpg'), 224)
show(im)

data = renormalize.from_image(resize_and_crop(im, 224), target='imagenet')

with Trace(net, 'layer4') as tr:
    preds = net(data[None])
```

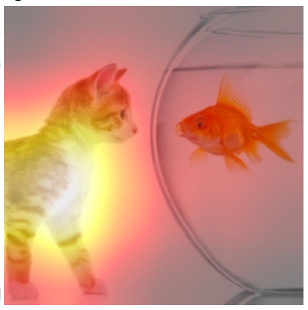
```
tigercat_index = labels.index('tiger cat')
tigercat_weights = net.fc.weight[tigercat_index]

tigercat_heatmap = torch.einsum('bcyx, c -> yx', tr.output, tigercat_weights)

# Displaying the heatmap directly
show(show.TIGHT, [['tiger cat', rgb_heatmap(tigercat_heatmap, mode='nearest')]])

# Overlaying the heatmap on the original image
show(show.TIGHT,
        [[['original', im],
         ['tiger-cat', overlay(im, rgb_heatmap(tigercat_heatmap, im))]
         ]]
        )
```

tiger cat



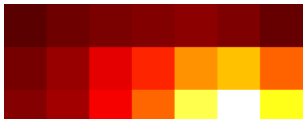
original

tiger-cat



```
goldfish_index = labels.index('goldfish')
goldfish_weights = net.fc.weight[goldfish_index]

goldfish_heatmap = torch.einsum('bcyx, c -> yx', tr.output, goldfish_weights)

# Displaying the heatmap directly
show(show.TIGHT, [['goldfish', rgb_heatmap(goldfish_heatmap, mode='nearest')]])

# Overlaying the heatmap on the original image
show(show.TIGHT,
        [[['original', im],
         ['goldfish', overlay(im, rgb_heatmap(goldfish_heatmap, im))]
         ]]
        )
```

goldfish



Start coding or generate with AI.

original

goldfish