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

```
%%bash
# If you are on Google Colab, this sets up everything needed.
!(stat -t /usr/local/lib/*/dist-packages/google/colab > /dev/null 2>&1) && exit
wget -O requirements.txt https://cs7150.baulab.info/2022-Fall/setup/hw1_requirements.txt
pip install -r requirements.txt
# If you are not on Google Colab, you can run these pip requirements on your own command-line.
```

```
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (from importlib-metadata->diffusers>=0.2.4->-r requirements.txt (line 2)) (3.17.0)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from jedi<1,>=0.18->pudb>=2019.2->-r requirements.txt (line 4)) (0.8.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.20.3->test-tube>=0.7.5->-r requirements.txt (line 10)) (2023.3.po
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil<3,>=2.7.3->streamlit>=0.73.1->-r requirements.txt (line 11)) (1
Requirement already satisfied: scikit-learn>=0.19.1 in /usr/local/lib/python3.10/dist-packages (from qudida>=0.0.4->albumentations>=0.4.3->-r requirements.txt (line 1))
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->diffusers>=0.2.4->-r requirements.txt (line 2)) (3.3.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->diffusers>=0.2.4->-r requirements.txt (line 2)) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->diffusers>=0.2.4->-r requirements.txt (line 2)) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->diffusers>=0.2.4->-r requirements.txt (line 2)) (2023.7.22)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich<14,>=10.14.0->streamlit>=0.73.1->-r requirements.txt (line 11
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.16.1->albumentations>=0.4.3->-r requirements.txt (line 1))
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.16.1->albumentations>=0.4.3->-r requirements.txt (li
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.10/dist-packages (from tensorboard>=1.15.0->test-tube>=0.7.5->-r requirements.txt (line 10)) (1.4.
Requirement already satisfied: grpcio>=1.48.2 in /usr/local/lib/python3.10/dist-packages (from tensorboard>=1.15.0->test-tube>=0.7.5->-r requirements.txt (line 10)) (1.
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard>=1.15.0->test-tube>=0.7.5->-r requirements.txt (line 1
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard>=1.15.0->test-tube>=0.7.5->-r requirements.tx
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard>=1.15.0->test-tube>=0.7.5->-r requirements.txt (line 10)) (3
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard>=1.15.0->test-tube>=0.7.5->-r requirem
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard>=1.15.0->test-tube>=0.7.5->-r requirements.txt (line 10)) (3
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.10/dist-packages (from tensorboard>=1.15.0->test-tube>=0.7.5->-r requirements.txt (line 10)) (0.41.
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->invisible-watermark>=0.1.5->-r requirements.txt (line 5)) (1.12)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch->invisible-watermark>=0.1.5->-r requirements.txt (line 5)) (2.1.0)
Requirement already satisfied: wcwidth>=0.2.5 in /usr/local/lib/python3.10/dist-packages (from ftfy->clip->-r requirements.txt (line 18)) (0.2.8)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning>=1.4
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightnin
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning>=1.
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning>
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning>=
Requirement already satisfied: smmap<6,>=3.0.1 in /usr/local/lib/python3.10/dist-packages (from gitdb<5,>=4.0.1->gitpython!=3.1.19,<4,>=3.0.7->streamlit>=0.73.1->-r req
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard>=1.15.0->test-tube>=0.7.5->-r r
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard>=1.15.0->test-tube>=0.7.5->-r requireme
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<1.1,>=0.5->tensorboard>=1.15.0->test-tube>
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->altair<6,>=4.0->streamlit>=0.73.1->-r requirements.txt (line 11)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit>=0.73.1-
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit>=0.73.1->-r requirements.
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit>=0.73.1->-r requirements.txt (
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich<14,>=10.14.0->streamlit>=0.73.1->-r requirements.
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.19.1->qudida>=0.0.4->albumentations>=0.4.3->-r requirement
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.19.1->qudida>=0.0.4->albumentations>=0.4.3->-r requ
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch->invisible-watermark>=0.1.5->-r requirements.txt (line 5)) (1.
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard>=1.15.0->
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<1.1,>=0.5->tensorboard>=1
--2023-10-20 01:08:08--  https://cs7150.baulab.info/2022-Fall/setup/hw1_requirements.txt
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: 578 [application/octet-stream]
Saving to: 'requirements.txt'

     0K                                                       100%  150M=0s

2023-10-20 01:08:08 (150 MB/s) - 'requirements.txt' saved [578/578]

  Running command git clone --filter=blob:none --quiet https://github.com/davidbau/baukit /tmp/pip-install-4zb1edki/baukit_5ab0e10fd8864b0cb9e8b9c2f8ced7b4
  Running command git clone --filter=blob:none --quiet https://github.com/openai/CLIP.git /tmp/pip-install-4zb1edki/clip_abf3b6a37d29466a8366938c3aa98249
  Running command git clone --filter=blob:none --quiet https://github.com/CompVis/taming-transformers.git /tmp/pip-install-4zb1edki/taming-transformers_7aa180ba10264187
  Running command git clone --filter=blob:none --quiet https://github.com/CompVis/stable-diffusion.git /tmp/pip-install-4zb1edki/latent-diffusion_f7057985a09141adbe2c3c
```

```
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
```

```
--2023-10-20 01:09:03--  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... 304 Not Modified
File 'dog-and-cat-example.jpg' not modified on server. Omitting download.

--2023-10-20 01:09:03--  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... 304 Not Modified
File 'hungry-cat.jpg' not modified on server. Omitting download.
```

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)
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the futu
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.
  warnings.warn(msg)
```

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

```
# Type your solution here
print("Shape of masked batch: ", masked_batch[19].shape)
customData = masked_batch[19][None,:,:,:]
print("Shape of custom data: ", customData.shape)

masked_preds_val = net(customData)
print("Predicitons:", masked_preds_val.shape)

[(labels[i], i.item()) for i in masked_preds_val.topk(dim=1, k=5, sorted=True)[1][0]]
```

```
Shape of masked batch:  torch.Size([3, 224, 224])
Shape of custom data:  torch.Size([1, 3, 224, 224])
Predicitons: torch.Size([1, 1000])
[('tiger cat', 282),
 ('tabby', 281),
 ('Egyptian cat', 285),
 ('bull mastiff', 243),
 ('American Staffordshire terrier', 180)]
```

Observation:

When the face was obscured, the model's top 5 predictions shifted. This is understandable as many dogs have similar body structures, with distinct facial features setting them apart. Without access to the face, the model struggled to correctly classify the boxer.

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

Double-click (or enter) to edit

```
# Type your solution here

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

# Finding the image index where we have the least score for 'boxer dog'
least_boxer_score_index = masked_preds[:, boxer_index].argmin().item()

# Finding the image index where we have the least score for 'tiger cat'
least_tiger_cat_score_index = masked_preds[:, tiger_cat_index].argmin().item()
```

```
# Showing the image where we have the least score for 'boxer dog'
show(renormalize.as_image(masked_batch[least_boxer_score_index]))

# Showing the image where we have the least score for 'tiger cat'
show(renormalize.as_image(masked_batch[least_tiger_cat_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?"

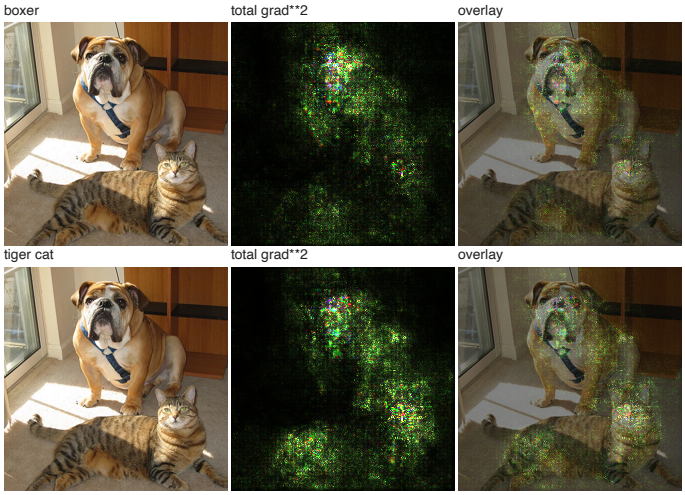
Although gradients are a neat idea, it can be hard to get them to work well for visualization. See Adebayo 2018

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
    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)
```

```
        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'))]
    ]])
```



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']
]))
```

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

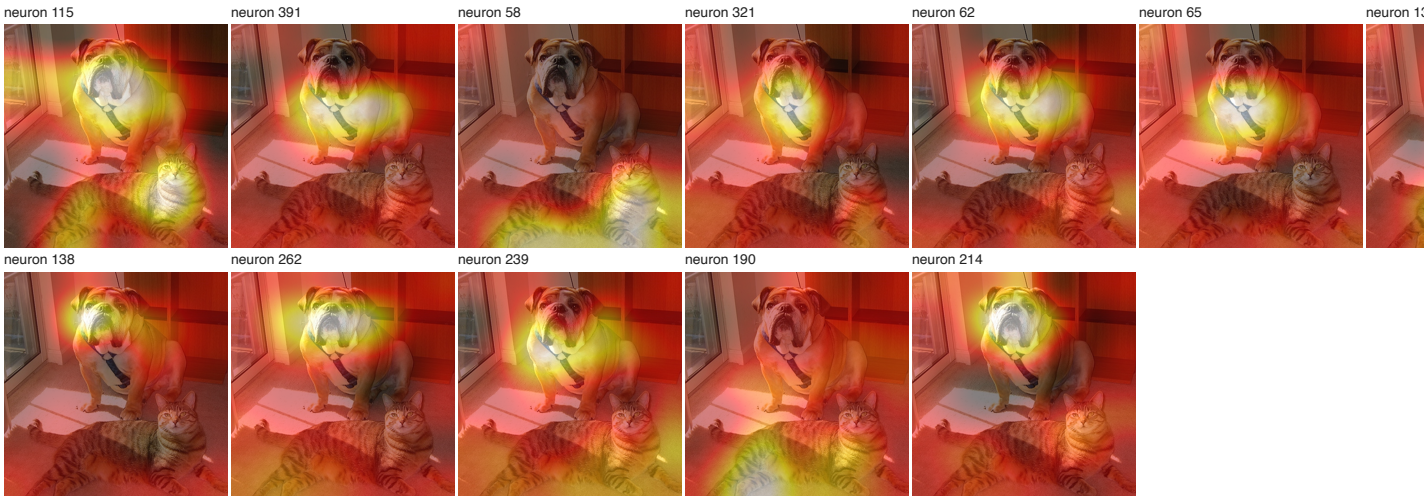
Which is the first neuron that activates for the cat but not the dog?

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]))]
    ])
```

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]

```
# Type your solution

# 1. Computing max activation for each filter
max_values = tr.output[0].max(dim=-1).values.max(dim=-1).values

# 2. Ranking the filters based on max activations and get the top 12
_, top_indices = max_values.topk(12,largest = True)

filter_visualizations = []

# 3. Visualize the top 12 filters
for idx in top_indices:
    overlay_img = overlay(im, rgb_heatmap(tr.output[0, idx]))
    filter_visualizations.append([f'neuron {idx}', overlay_img])

show(show.WRAP, filter_visualizations)
```




```
top_indices
```

```
tensor([115, 391,  58, 321,  62,  65,  13, 138, 262, 239, 190, 214])
```

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

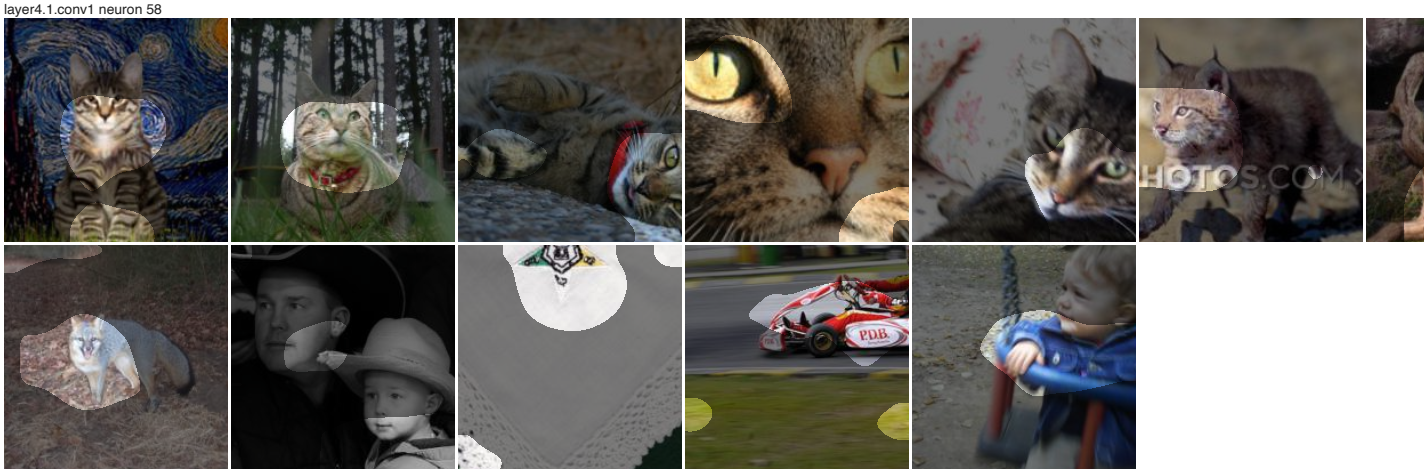
Choose one and run the network on all the data and sort to find the maximum-activating data. Let's see how the neuron you found to be top activating generalizes. We will trace the neuron activations of the entire dataset and visualise the top 12 images and display the regions where the chosen neurons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

Observation:

Not exactly. The neuron also activates for images with babies. While it seems to respond strongly to features of cats like eyes, fur, and faces, it doesn't entirely capture all aspects of a cat, indicating its generalization might be limited.

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.

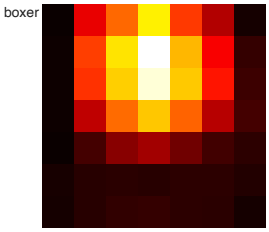
```
boxer_weights = net.fc.weight[labels.index('boxer')]
```

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.

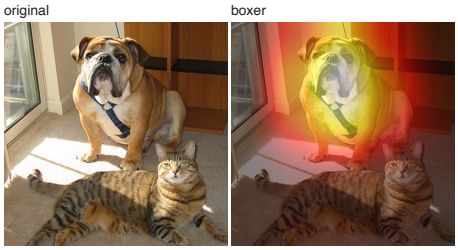
The result will be a 7x7 grid of dot product strengths, which we can render as a heatmap.

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

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



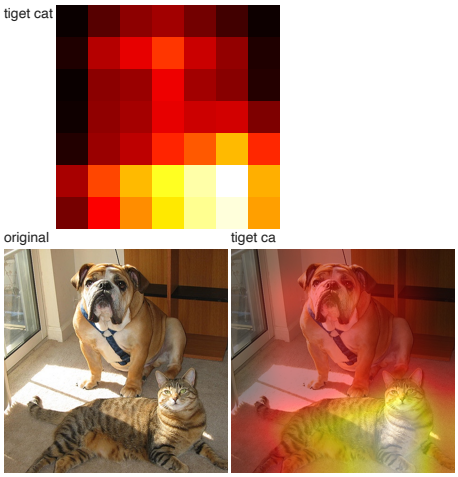
In the following code we smooth the heatmaps and overlay them on top of the original image.

```
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

```
# Type your solution here

# neuron weights
tiger_weights = net.fc.weight[labels.index('tiger cat')]

# dot product
tiger_heatmap = torch.einsum('bcyx, c -> yx', tr.output, tiger_weights)

# 7x7 grid of dot product strengths
show(show.TIGHT,
     [
       ['tiget cat',
        rgb_heatmap(tiger_heatmap, mode='nearest')]])

# smoothing the heatmaps and overlaying them on top of the original image
show(show.TIGHT,
    [['original', im],
     ['tiget ca', overlay(im, rgb_heatmap(tiger_heatmap, im))]
    ]]
)
```



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

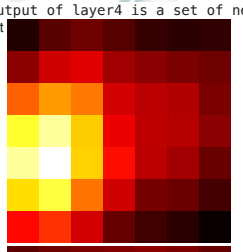
Load the image `hungry-cat.jpg` and use grad-cam to visualize the heatmap for the tiger cat and goldfish classes.

```
# Type your solution

# Loading the image
im = resize_and_crop(PIL.Image.open('hungry-cat.jpg'), 224)
```

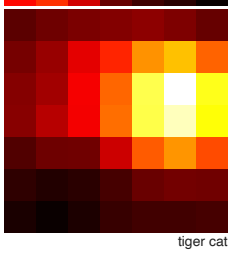
```
show(im)
data = renormalize.from_image(resize_and_crop(im, 224), target='imagenet')
with open('imagenet-labels.txt') as r:
    labels = [line.split(',')[1].strip() for line in r.readlines()]
net = resnet18(pretrained=True)
net.eval()
set_requires_grad(False, net)


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)


# neuron weights
tiger_cat_weights = net.fc.weight[labels.index('tiger cat')]
goldfish_weights = net.fc.weight[labels.index('goldfish')]

# dot product
tiger_cat_heatmap = torch.einsum('bcyx, c -> yx', tr.output, tiger_cat_weights)
goldfish_heatmap = torch.einsum('bcyx, c -> yx', tr.output, goldfish_weights)



show(show.TIGHT,
     [
       ['tiger cat', rgb_heatmap(tiger_cat_heatmap, mode='nearest')],
       ['goldfish', rgb_heatmap(goldfish_heatmap, mode='nearest')]
     ])

# Displaying the overlay of the heatmaps on the original image

show(show.TIGHT,
    [[['original', im],
     ['tiger cat', overlay(im, rgb_heatmap(tiger_cat_heatmap, im))],
      ['goldfish', overlay(im, rgb_heatmap(goldfish_heatmap, im))]
    ]]
)
```

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

tiger cat



goldfish



original tiger cat goldfish