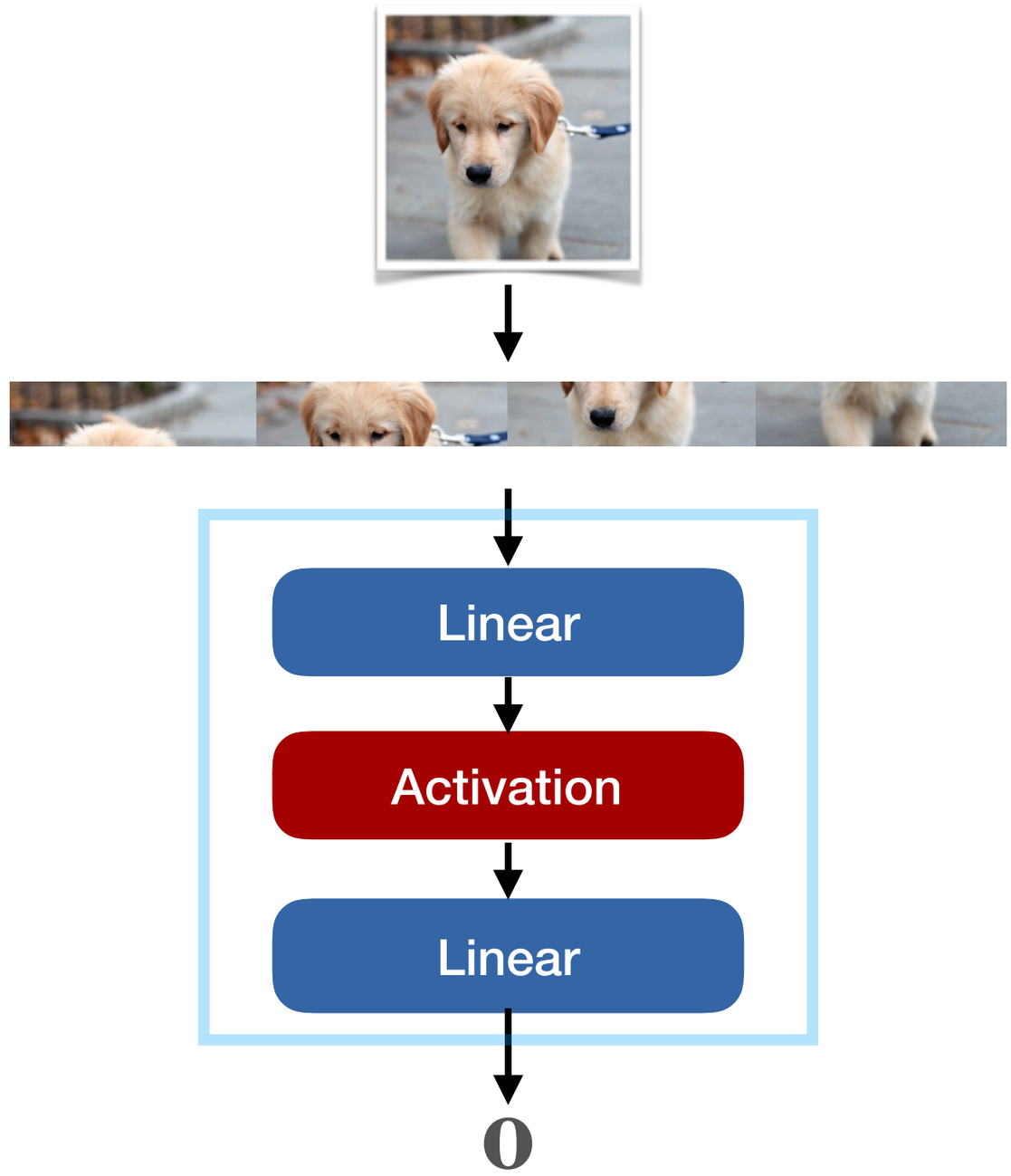
Images and structure

Images and structure

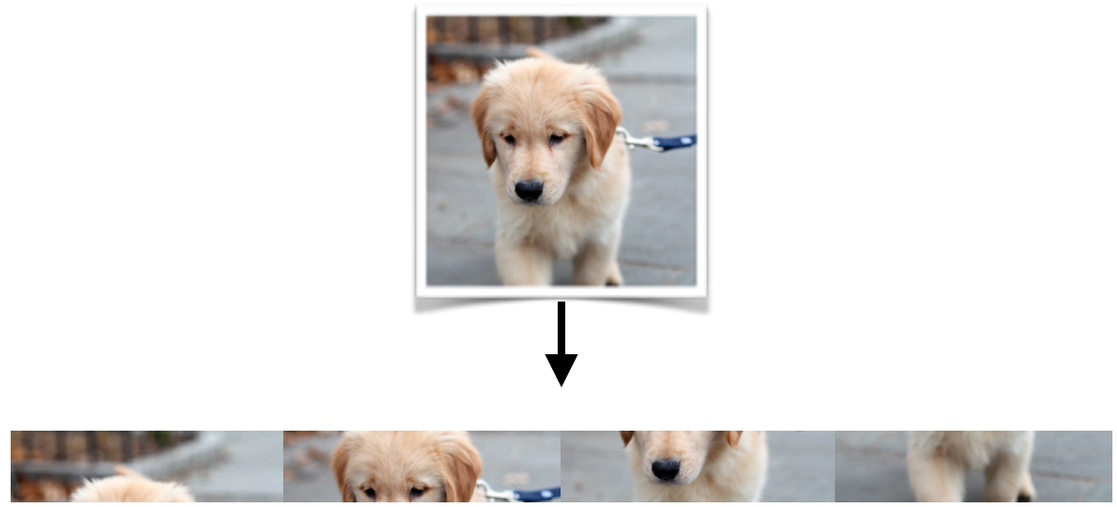
- Fully-connected networks

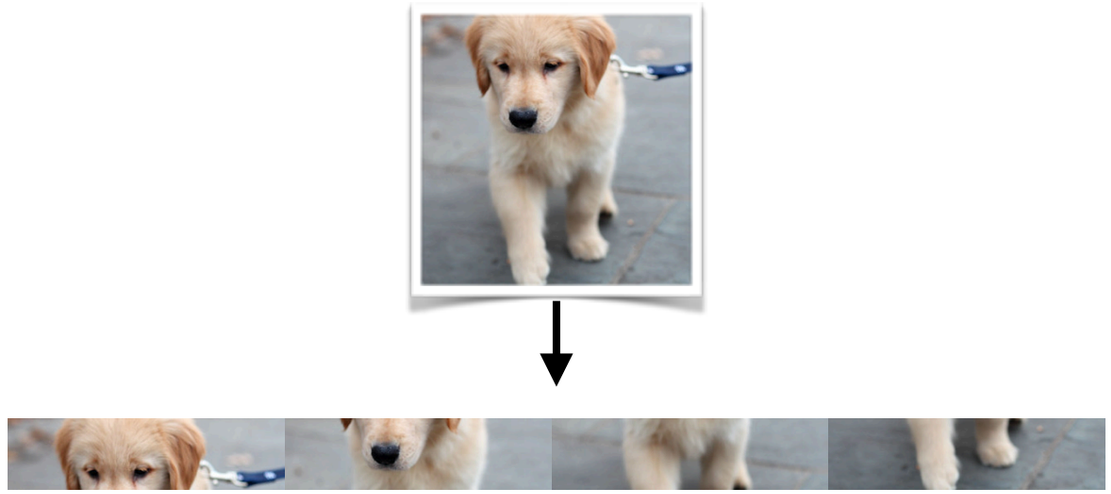


Images and structure

- Visual patterns are shift, rotation and scale invariant

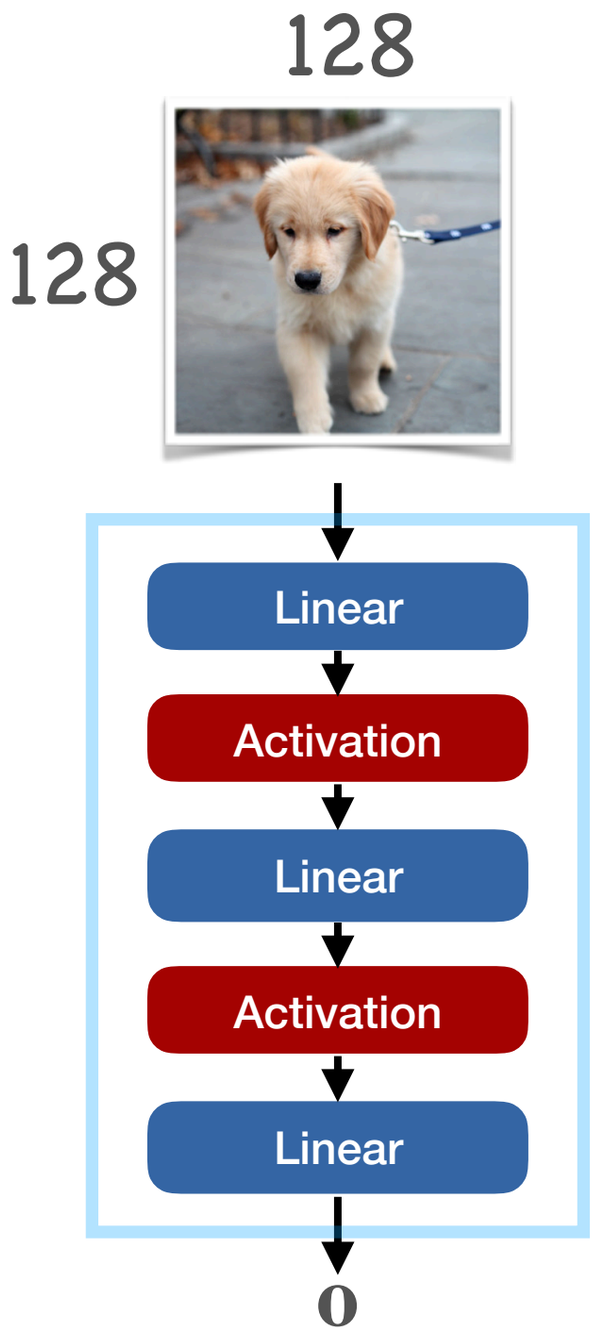
- Fully connected networks are not





High dimensional inputs

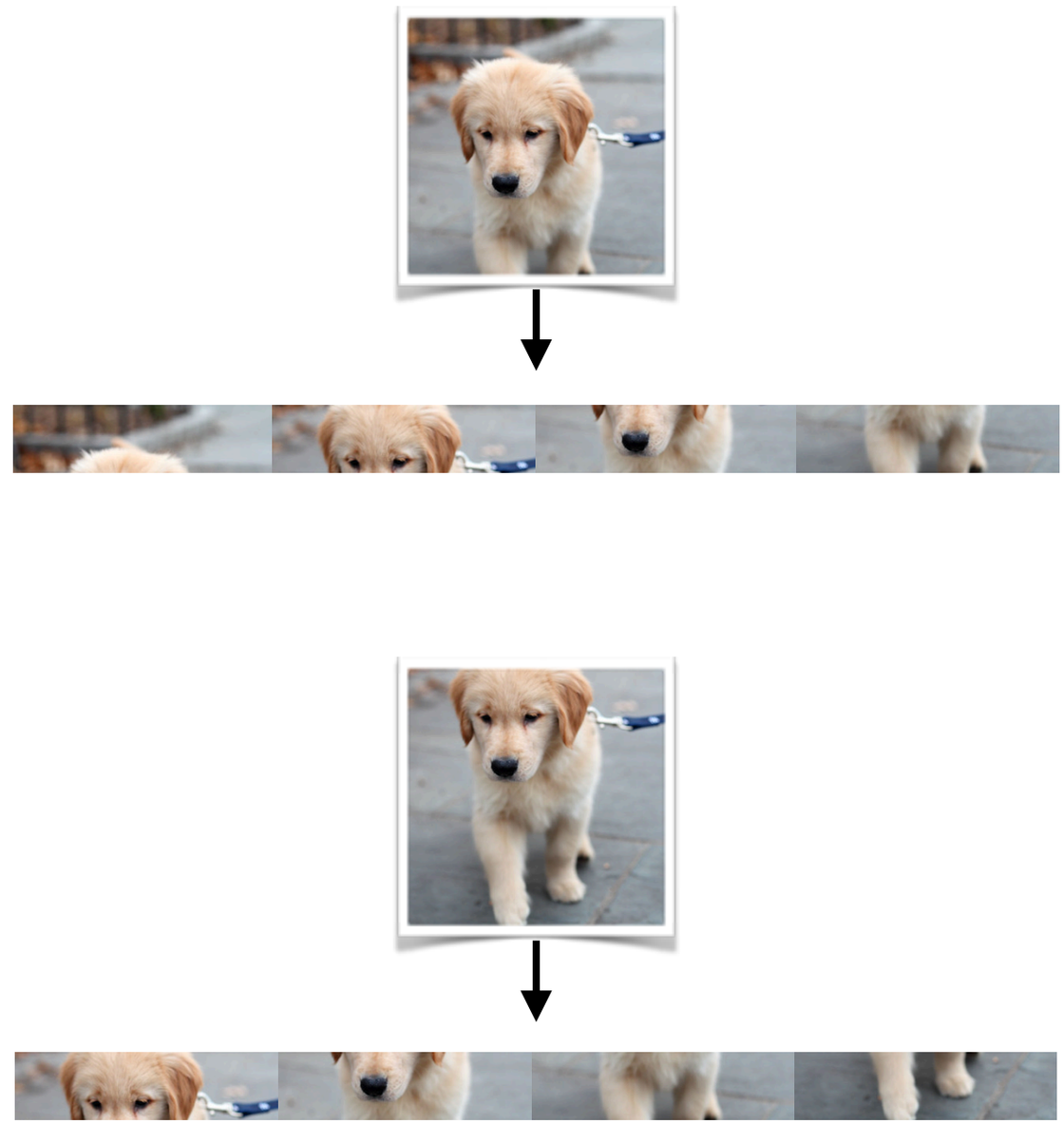
High dimensional inputs



Convolutions

Images and structure

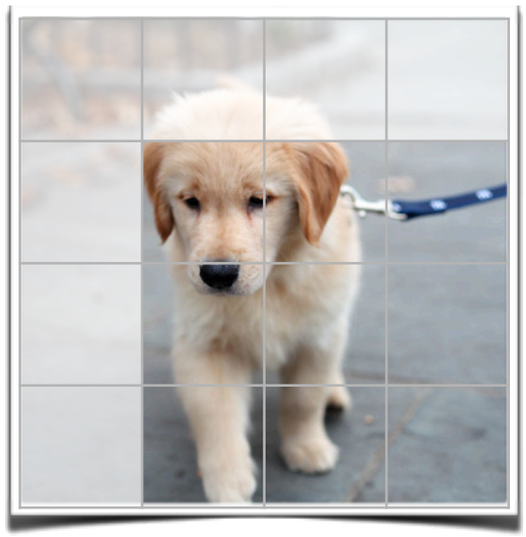
- Fully connected networks are not shift invariant



Finding shift-invariant patterns

Convolutions

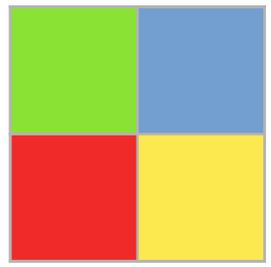
- "Sliding" linear transformation



*

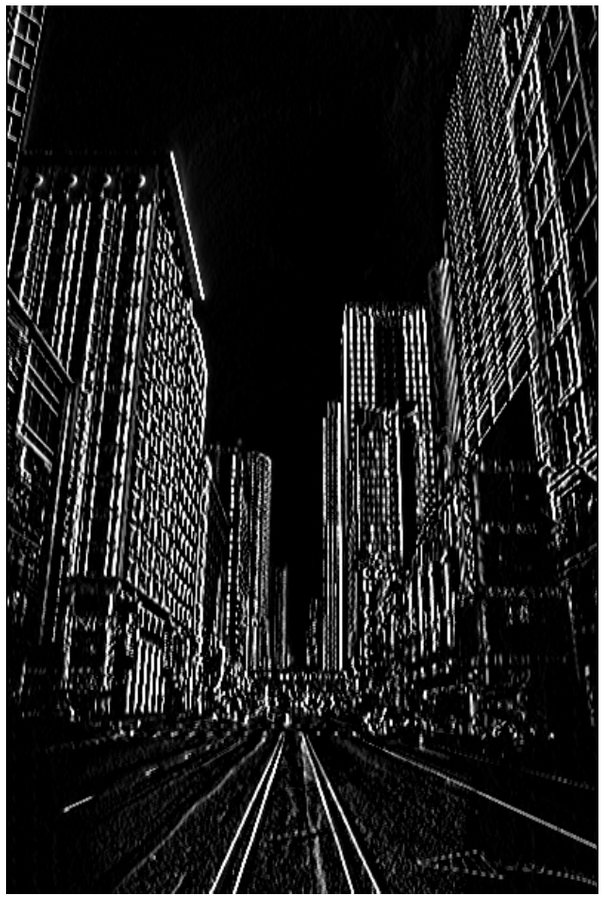
a	b	c
d	e	f
g	h	i

=



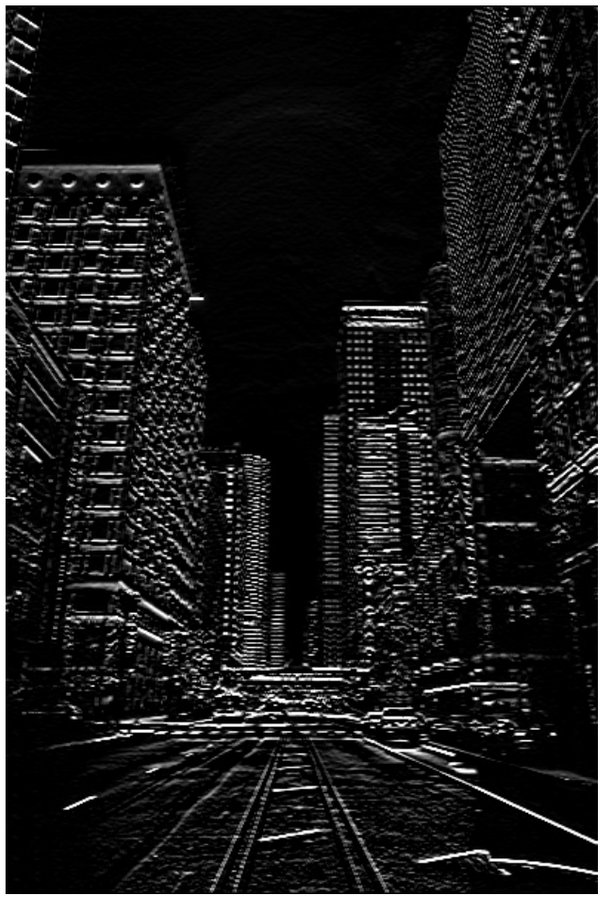
Examples of convolutions

Original



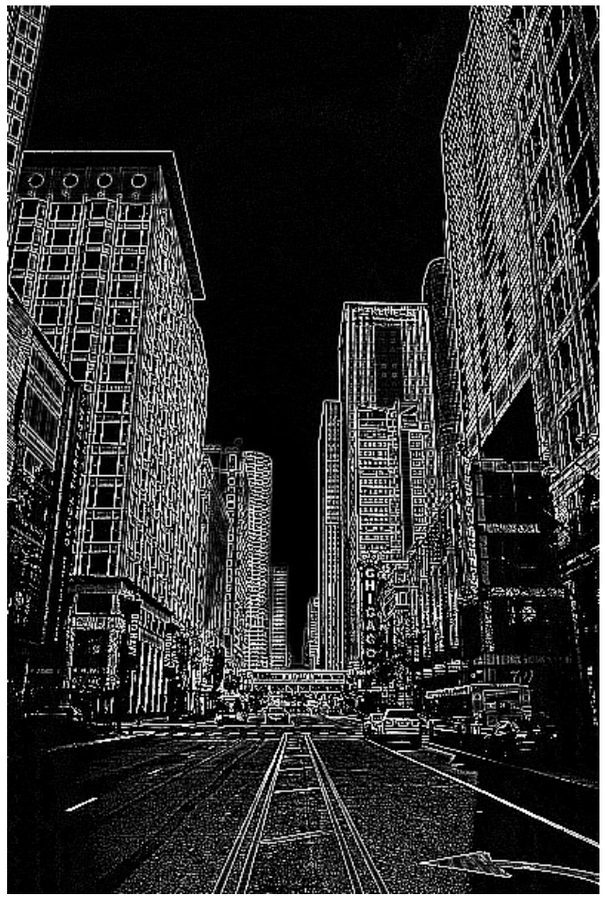
Vertical edges

-1	0	1
-1	0	1
-1	0	1



Horizontal edges

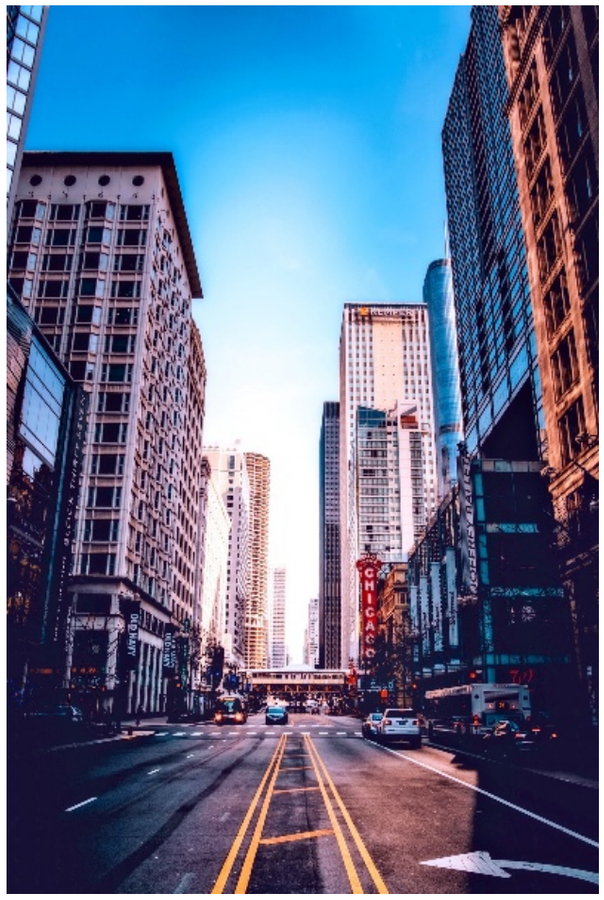
-1	-1	-1
0	0	0
1	1	1



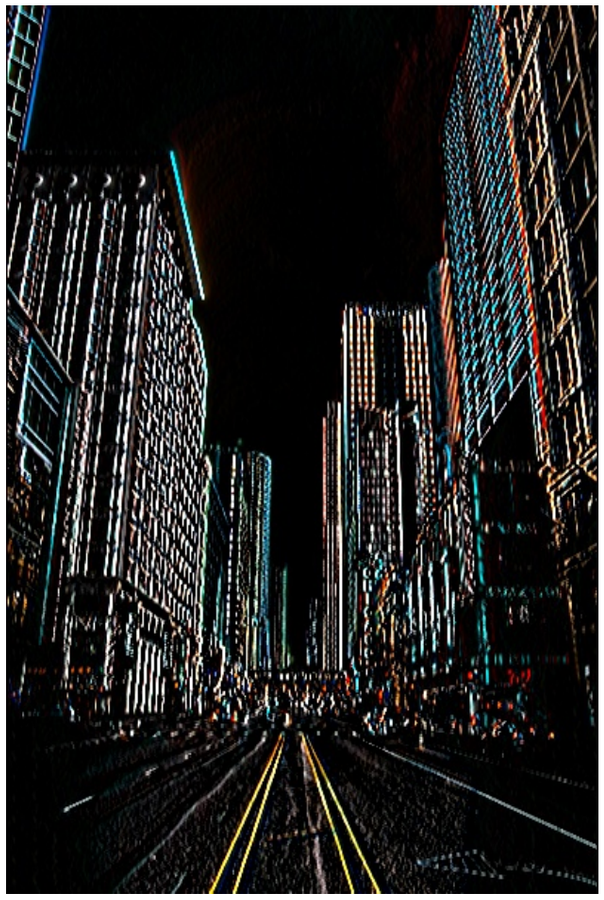
Laplace filter

-1	-1	-1
-1	8	-1
-1	-1	-1

Convolutions on multiple channels



Original



Vertical edges

-1	0	1
-1	0	1
-1	0	1

Horizontal edges

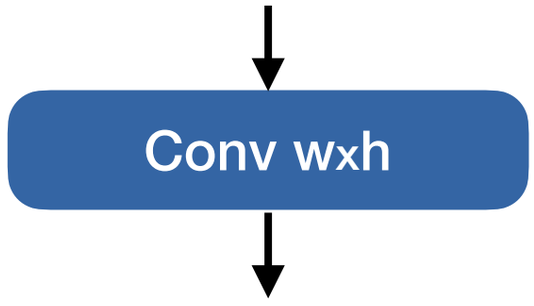
-1	-1	-1
0	0	0
1	1	1

Laplace filter

-1	-1	-1
-1	8	-1
-1	-1	-1

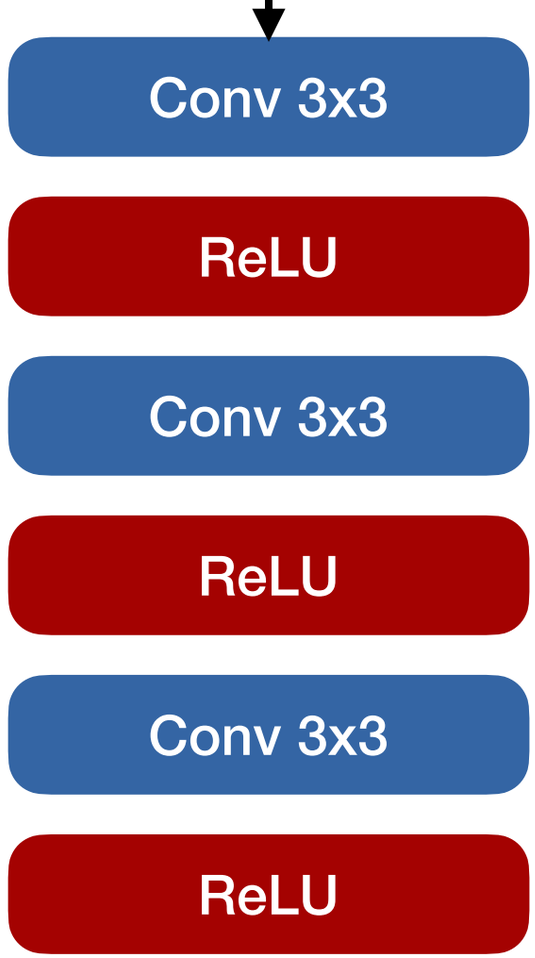
Formal definition

- Input: $\mathbf{X} \in \mathbb{R}^{H \times W \times C_1}$
- Kernel: $\mathbf{w} \in \mathbb{R}^{h \times w \times C_1 \times C_2}$
- Bias: $\mathbf{b} \in \mathbb{R}^{C_2}$
- Output: $\mathbf{Z} \in \mathbb{R}^{(H-h+1) \times (W-w+1) \times C_2}$



$$\mathbf{Z}_{a,b,c} = \mathbf{b}_c + \sum_{i=0}^{h} \sum_{j=0}^{w} \sum_{k=0}^{C_1} \mathbf{X}_{a+i,b+j,k} W_{i,j,k,c}$$

Stacking multiple layers



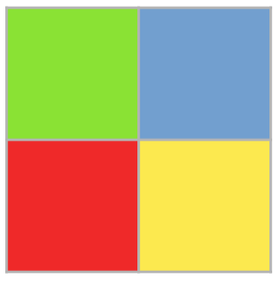
Convolution as a linear layer

input: 3 x 3 x 1

kernel: 2 x 2 x 1 x 1

a	b
c	d

output: 2 x 2 x 1



input: 9

weight: 4 x 9

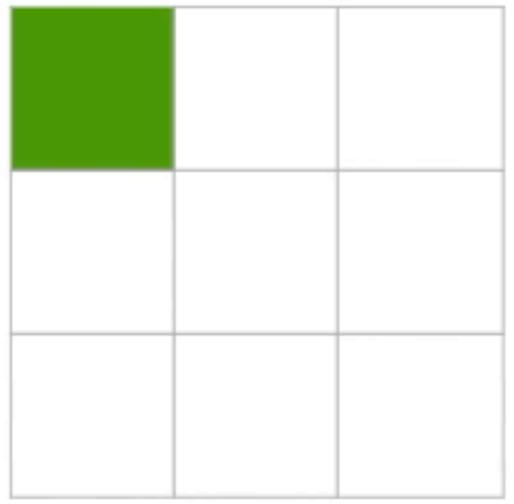
a	b		c	d				
	a	b		c	d			
			a	b		c	d	
				a	b		c	d

output: 4

Special case: 1x1 convolution

- Pixel-wise linear transformation
- Kernel: $1 \times 1 \times C_1 \times C_2$

$*$ $\mathbf{W}$ $=$



03

January 23, 2024

[1]:
```
%pylab inline
import torch
import sys
sys.path.append('..')
sys.path.append('../..')
from data import load
train_data, train_label = load.get_dogs_and_cats_data(resize=(32,32),␣
 ↪n_images=10)
device = torch.device('cuda') if torch.cuda.is_available() else torch.
 ↪device('cpu')
print('device = ', device)
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
device =  cuda
```

[2]:
```
class ConvNet(torch.nn.Module):
    def __init__(self, layers=[], n_input_channels=3, kernel_size=3):
        super().__init__()
        L = []
        c = n_input_channels
        for l in layers:
            L.append(torch.nn.Conv2d(c, l, kernel_size))
            L.append(torch.nn.ReLU())
            c = l
        L.append(torch.nn.Conv2d(c, 1, kernel_size=1))
        self.layers = torch.nn.Sequential(*L)

    def forward(self, x):
        return self.layers(x).mean(dim=[1,2,3])

net = ConvNet([32,64])
```

[3]:
```
print( train_data[:1].shape )
print( net(train_data[:1]).shape )
```

```
torch.Size([1, 3, 32, 32])
torch.Size([1])
```

[4]:
```
net2 = ConvNet([32, 64, 128])
print( net2(train_data[:1]).shape )
```

```
torch.Size([1])
```

[5]:
```
%load_ext tensorboard
import tempfile
log_dir = tempfile.mkdtemp()
%tensorboard --logdir {log_dir} --reload_interval 1
```

```
<IPython.core.display.HTML object>
```

[6]:
```
from util import train
train.train(net2, batch_size=128, resize=(32,32), log_dir=log_dir,␣
 ↪device=device, n_epochs=100)
```

```
WARNING:root:loading dataset
WARNING:root:loading done
```

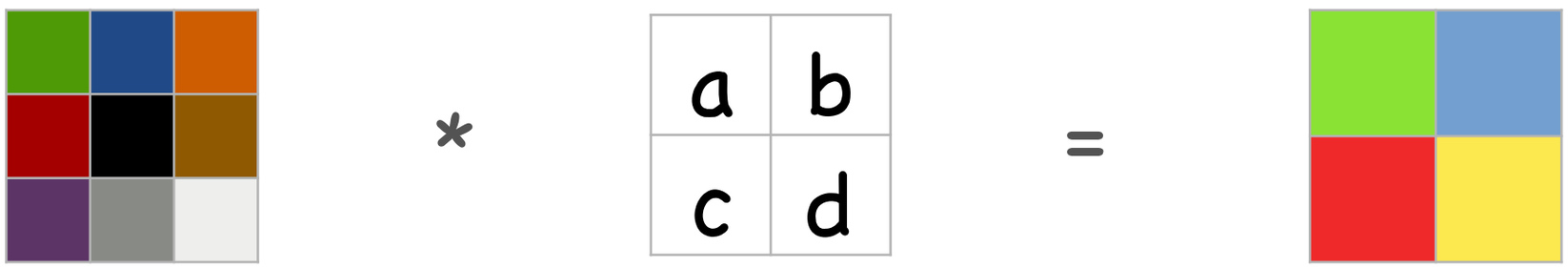
```
  0%|          | 0/100 [00:00<?, ?it/s]
```

[ ]:
```

```

Convolutional operators and their structure

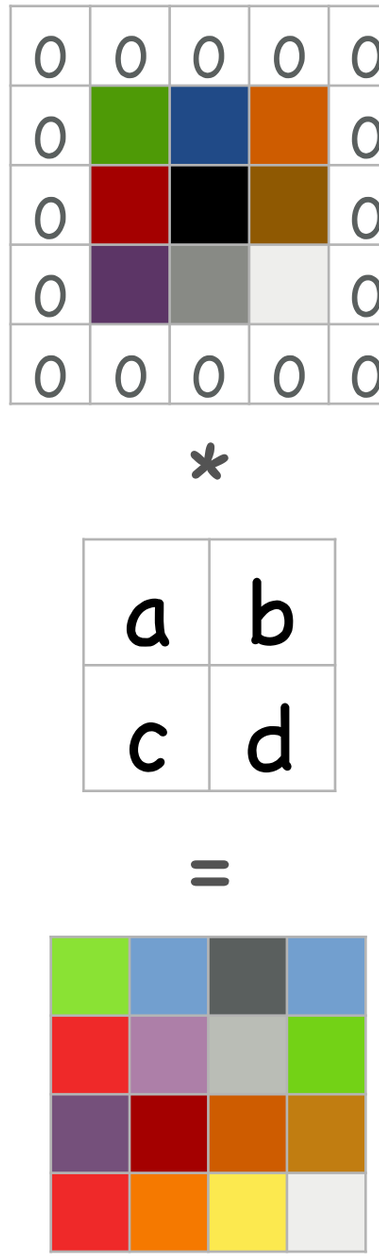
Output size

- Input: $\mathbf{X} \in \mathbb{R}^{H \times W \times C_1}$
- Kernel: $\mathbf{w} \in \mathbb{R}^{h \times w \times C_1 \times C_2}$
- Output: $\mathbf{Z} \in \mathbb{R}^{(H-h+1) \times (W-w+1) \times C_2}$



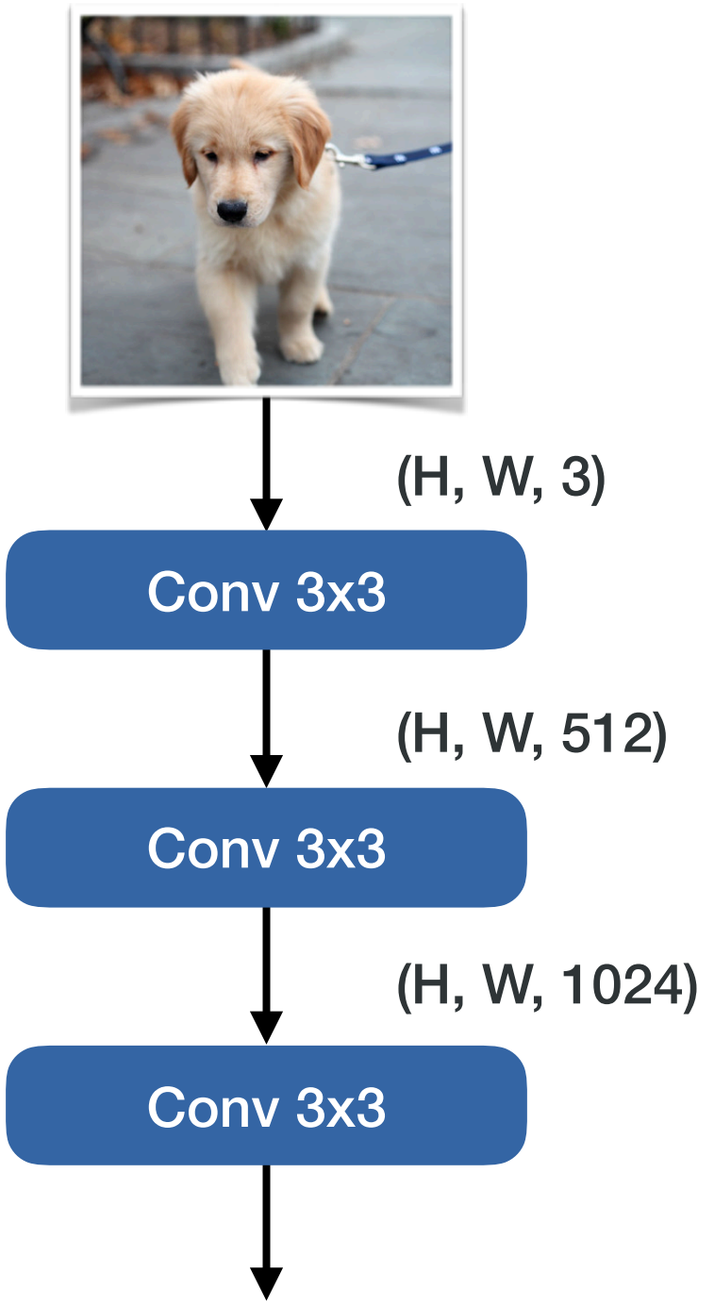
Padding

- Add p_w, p_h zeros in each dimension
- Input: $\mathbf{X} \in \mathbb{R}^{H \times W \times C_1}$
- Kernel: $\mathbf{w} \in \mathbb{R}^{h \times w \times C_1 \times C_2}$
- Output: $\mathbf{Z} \in \mathbb{R}^{(H-h+2p_h+1) \times (W-w+2p_w+1) \times C_2}$



Output resolution

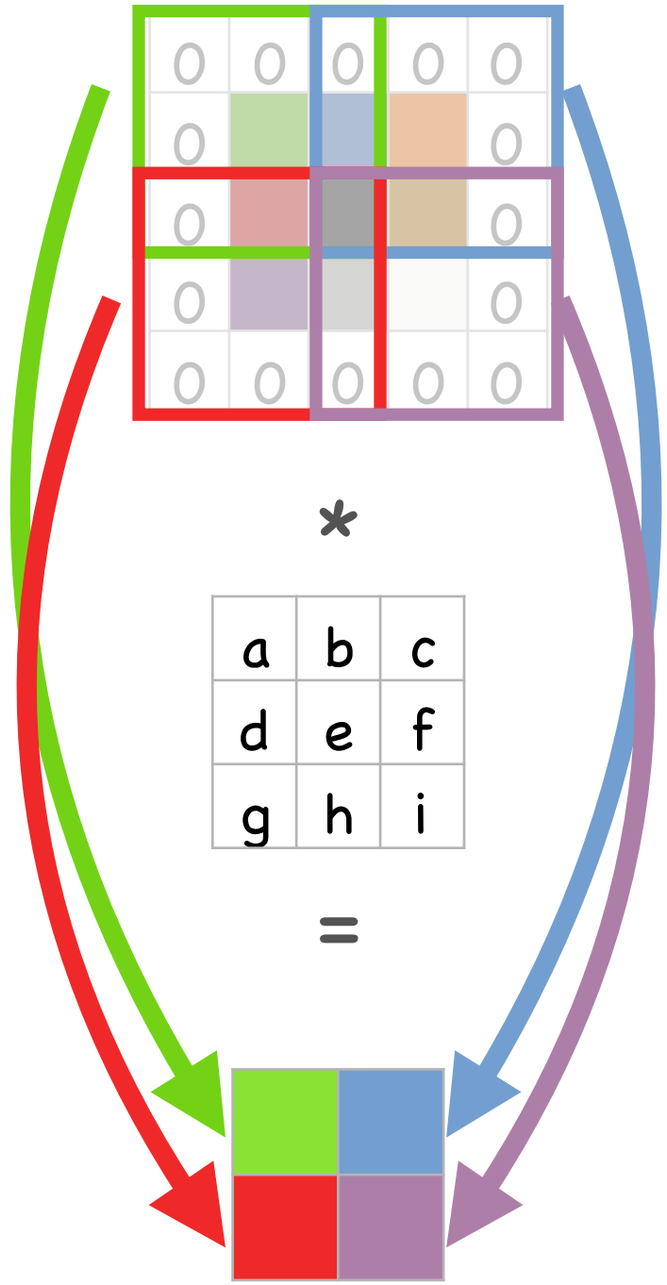
- High output resolution
  - Slow computation



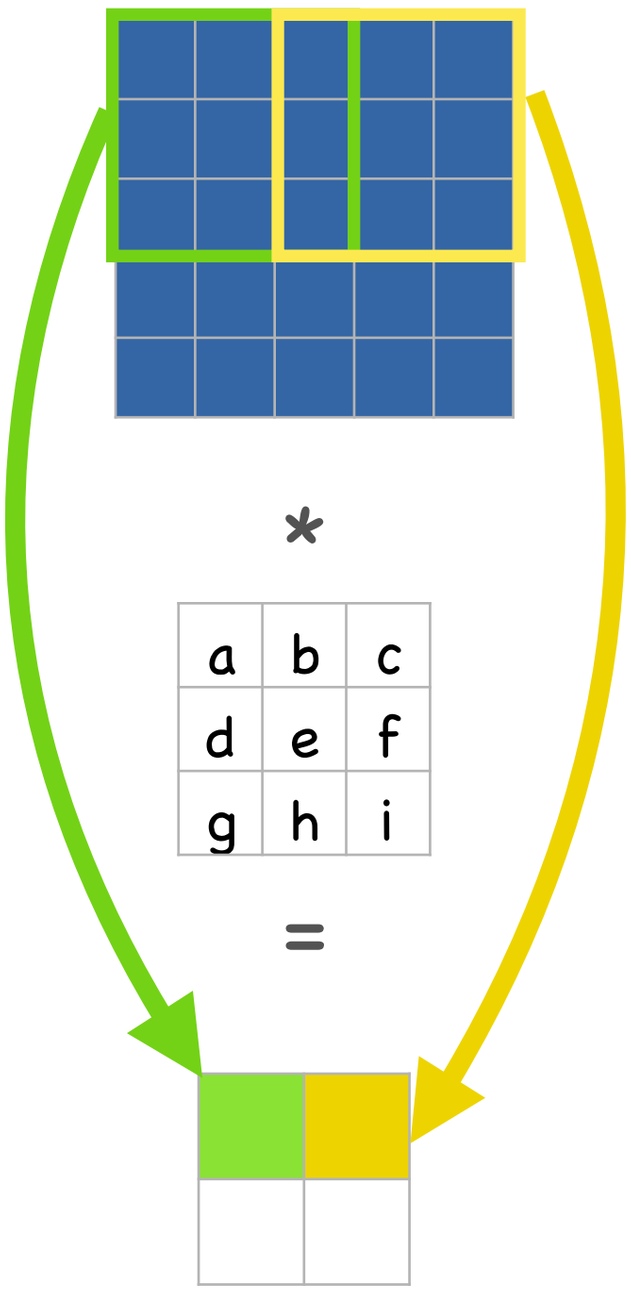
Striding

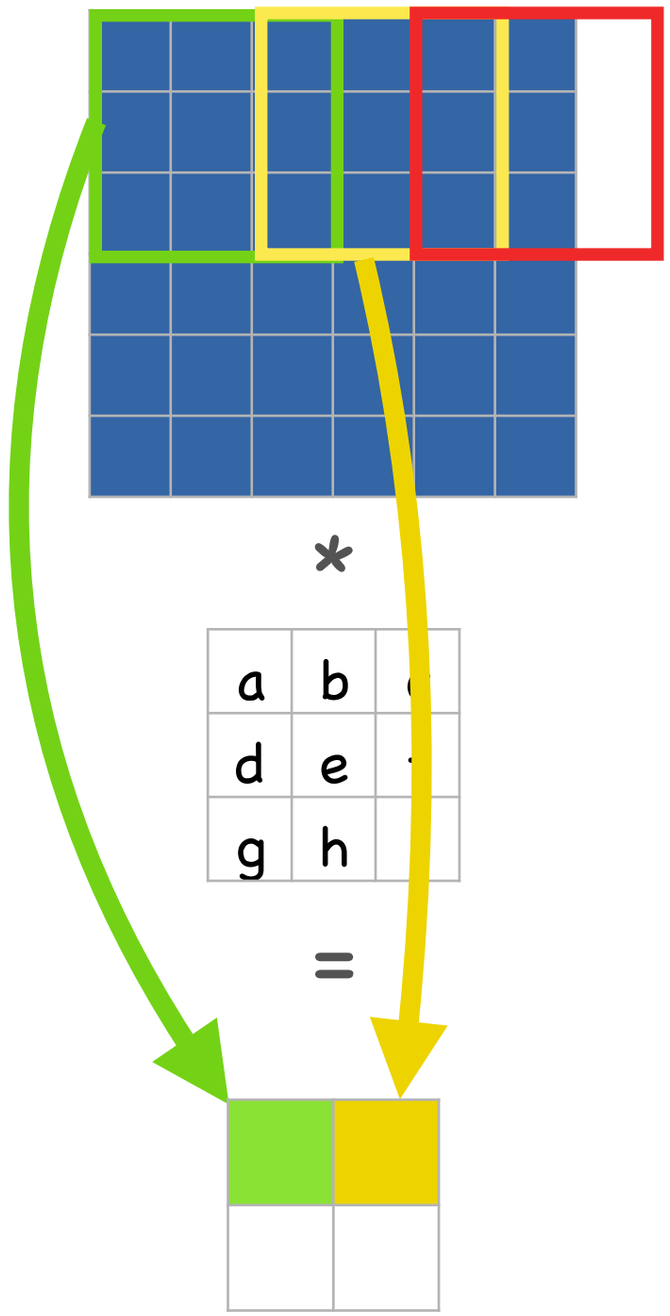
- Only compute every n-th output: s_w, s_h
- Input: $\mathbf{X} \in \mathbb{R}^{H \times W \times C_1}$
- Kernel: $\mathbf{w} \in \mathbb{R}^{h \times w \times C_1 \times C_2}$
- Output:

$$\mathbf{Z} \in \mathbb{R}^{\left(\frac{H-h+2p_h}{s_h}+1\right) \times \left(\frac{W-w+2p_w}{s_w}+1\right) \times C_2}$$



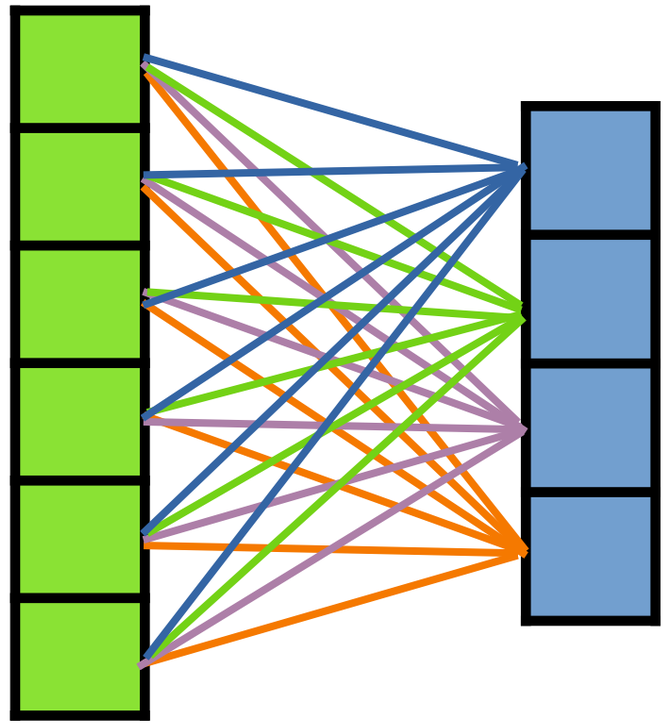
Output size with striding





Parameters

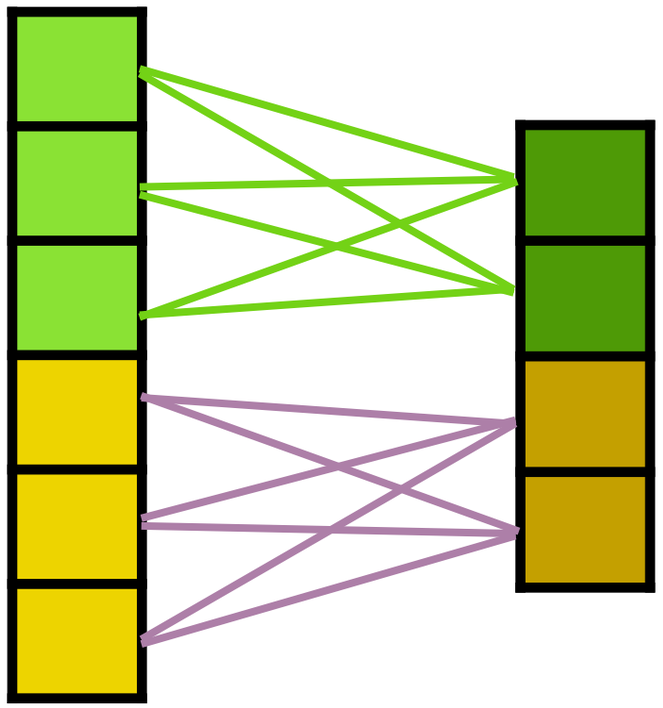
- Every input channel C_1 is connected to every output channel C_2



$C_1 = 6$ $C_2 = 4$

Grouping

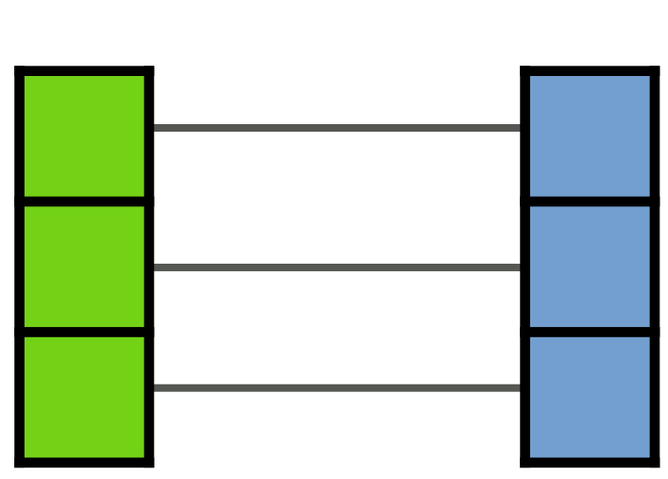
- Split channels into g groups
- Reduce parameters and computation by factor g



$C_1 = 6$ $C_2 = 4$

Depthwise convolution

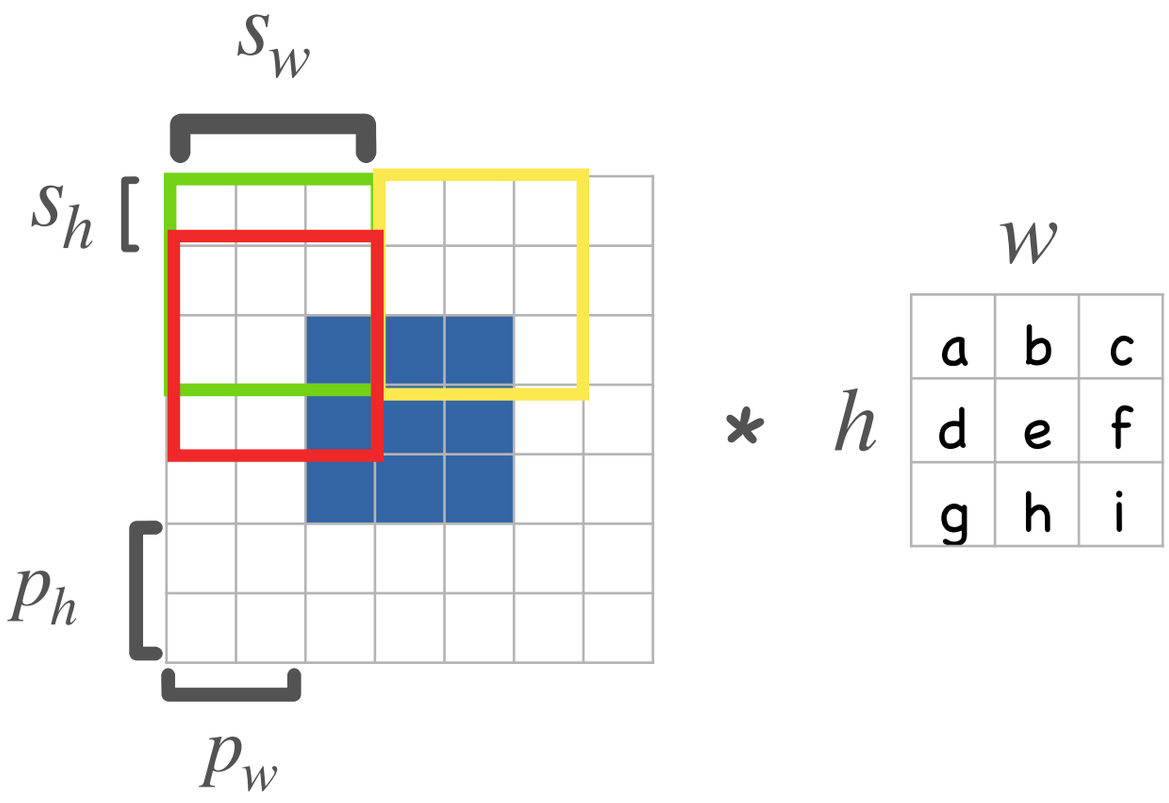
- Special grouping
  - $C_1 = g$
  - $C_2 = g$



$C_1 = 3$ $C_2 = 3$

Hyper-parameters of convolutions

- Kernel size: $w \times h$
- Padding: p_w , p_h
- Stride: s_w , s_h



Convolutional operators

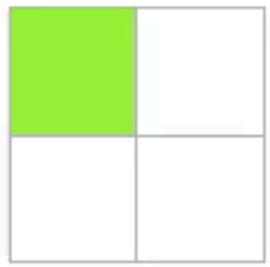
- Run arbitrary operation $f(\mathbf{x})$ "over" image

$f(\mathbf{x})$

$*$

a	b	c
d	e	f
g	h	i

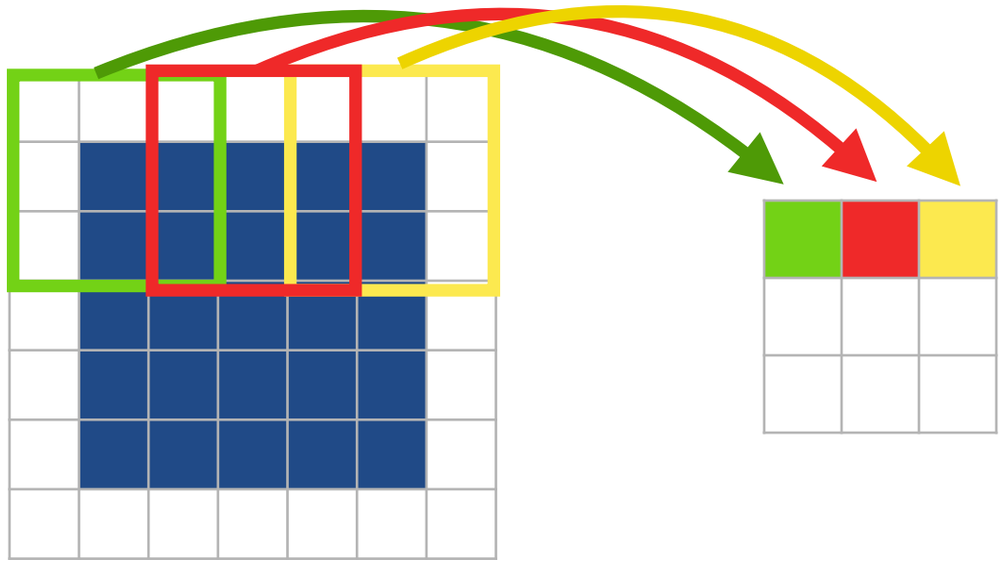
$=$



Average pooling

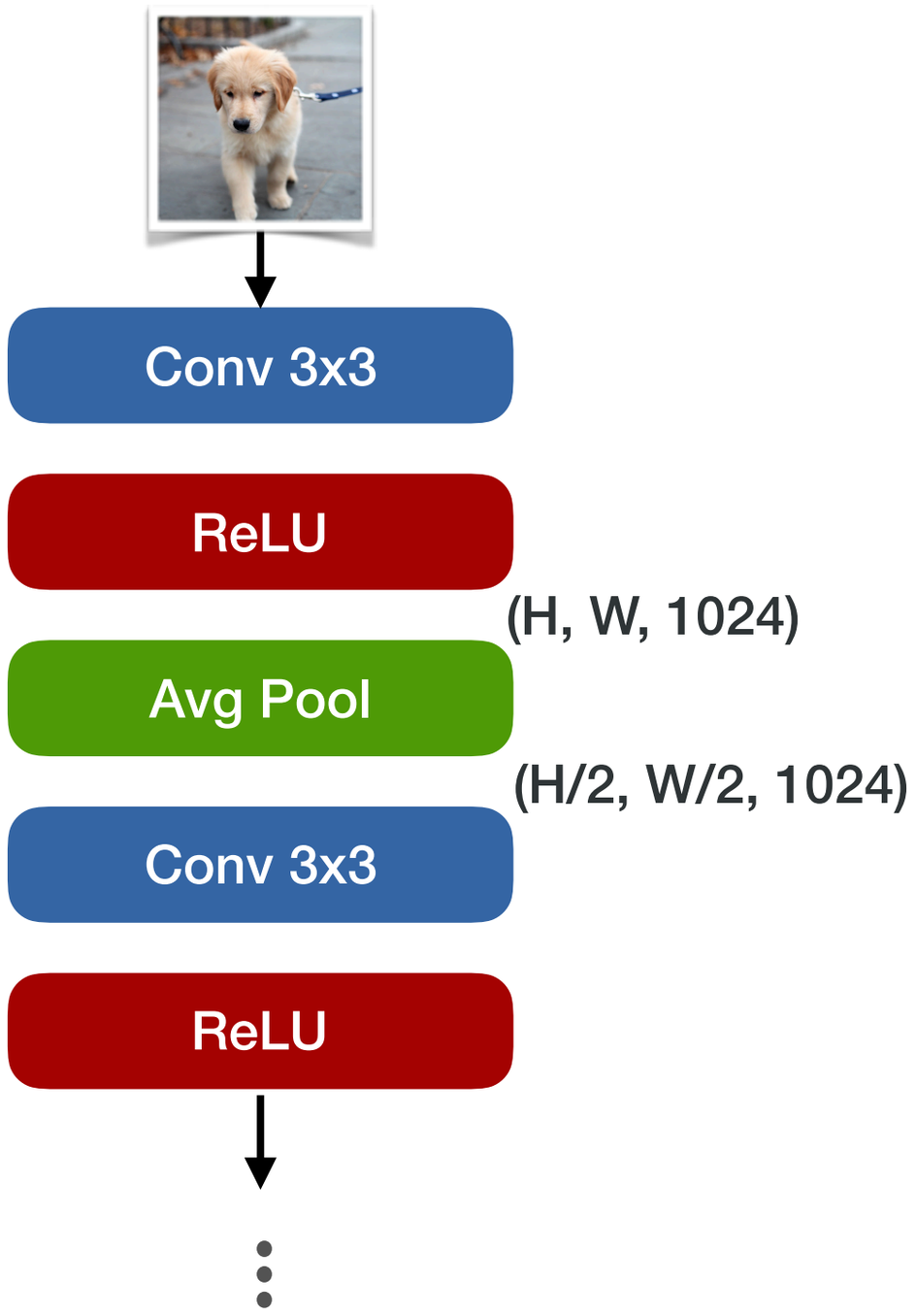
Average pooling

- Convolutional operator
  - $f_c(\mathbf{x}) = \text{mean}_{i,j}(\mathbf{x}_{i,j,c})$



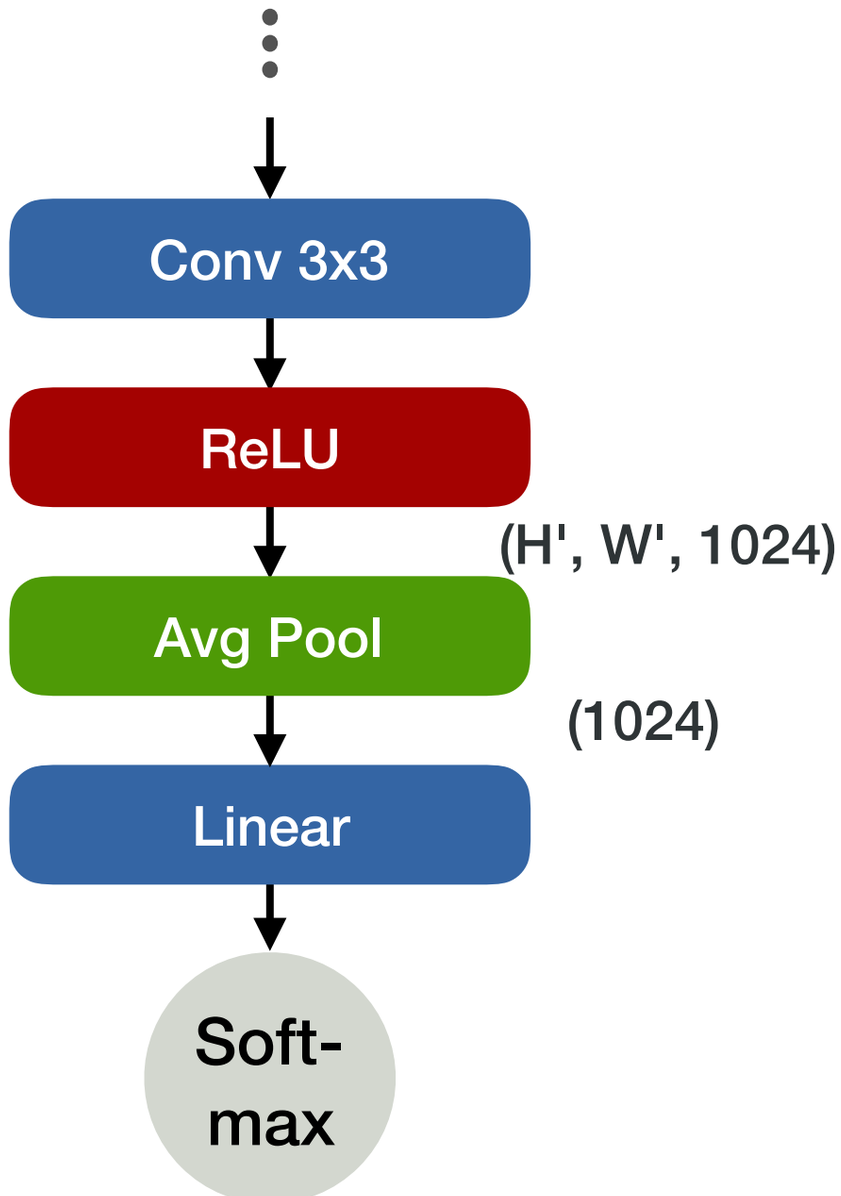
Where to use average pooling?

- Older networks:
  - Inside a network



Where to use average pooling?

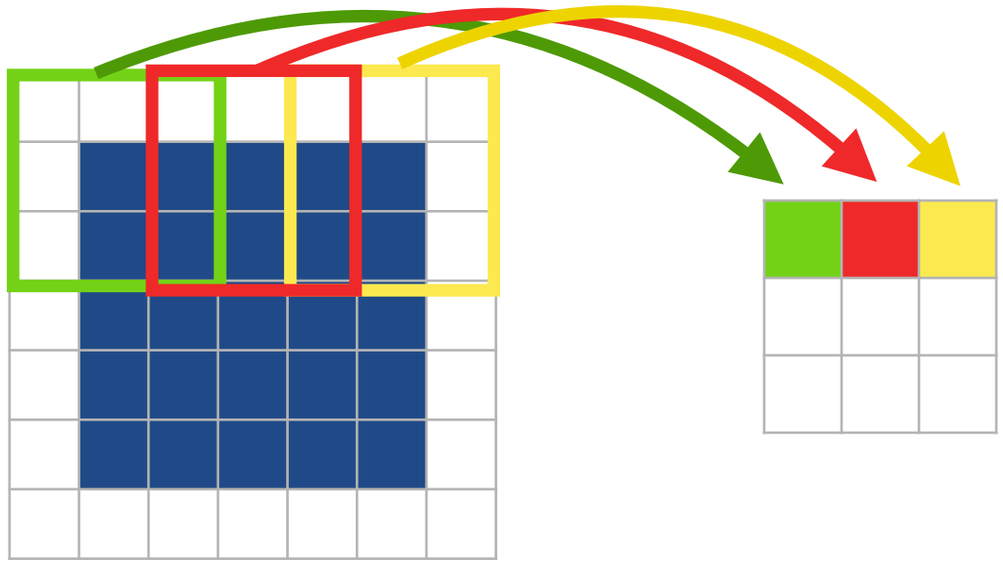
- Modern networks
  - Global average pooling



Max pooling

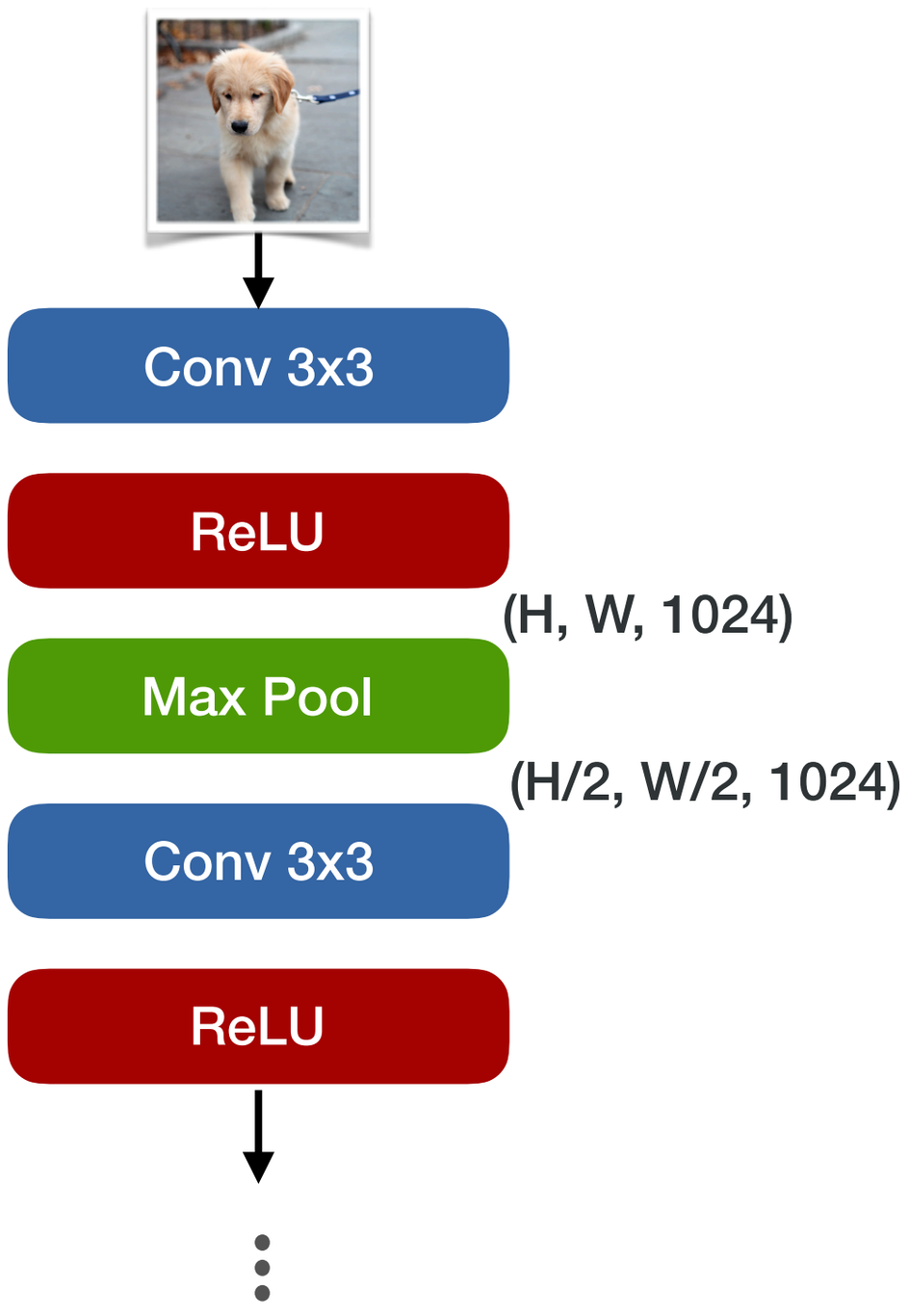
Max pooling

- Convolutional operator
  - $f_c(\mathbf{x}) = \max_{i,j}(\mathbf{x}_{i,j,c})$



Where to use max pooling?

- Inside a network
  - With strides, as down-sampling



Max pooling as a non-linearity

- Similar to maxout

07

January 23, 2024

[1]:
```
%pylab inline
import torch
import sys
sys.path.append('..')
sys.path.append('../..')
from data import load
train_data, train_label = load.get_dogs_and_cats_data(resize=(32,32),␣
 ↪n_images=10)
device = torch.device('cuda') if torch.cuda.is_available() else torch.
 ↪device('cpu')
print('device = ', device)
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
device =  cuda
```

[2]:
```
class ConvNet(torch.nn.Module):
    def __init__(self, layers=[], n_input_channels=3, kernel_size=3, stride=2):
        super().__init__()
        L = []
        c = n_input_channels
        for l in layers:
            L.append(torch.nn.Conv2d(c, l, kernel_size, padding=(kernel_size-1)/
 ↪/2, stride=stride))
            L.append(torch.nn.ReLU())
            c = l
        L.append(torch.nn.Conv2d(c, 1, kernel_size=1))
        self.layers = torch.nn.Sequential(*L)

    def forward(self, x):
        return self.layers(x).mean([1,2,3])

net = ConvNet([32,64])
```

[3]:
```
print( train_data[:1].shape )
print( net(train_data[:1]).shape )
```

```
torch.Size([1, 3, 32, 32])
```

```
torch.Size([1])
```

[4]:
```
net2 = ConvNet([32,64,128])
print( net2(train_data[:1]).shape )
```

```
torch.Size([1])
```

[5]:
```
class ConvNet2(torch.nn.Module):
    def __init__(self, layers=[], n_input_channels=3, kernel_size=3, stride=2):
        super().__init__()
        L = []
        c = n_input_channels
        for l in layers:
            L.append(torch.nn.Conv2d(c, l, kernel_size, padding=(kernel_size-1)/
 ↪/2))
            L.append(torch.nn.ReLU())
            L.append(torch.nn.MaxPool2d(3, padding=1, stride=stride))
            c = l
        L.append(torch.nn.Conv2d(c, 1, kernel_size=1))
        self.layers = torch.nn.Sequential(*L)

    def forward(self, x):
        return self.layers(x).mean([1,2,3])

net3 = ConvNet2([32,64,128])
```

[6]:
```
print( net3(train_data[:1]).shape )
```

```
torch.Size([1])
```

[7]:
```
%load_ext tensorboard
import tempfile
log_dir = tempfile.mkdtemp()
%tensorboard --logdir {log_dir} --reload_interval 1
```

```
<IPython.core.display.HTML object>
```

[8]:
```
from util import train
train.train(net2, batch_size=128, resize=(32,32), log_dir=log_dir+'/net2',␣
 ↪device=device, n_epochs=100)
```

```
WARNING:root:loading dataset
WARNING:root:loading done
  0%|          | 0/100 [00:00<?, ?it/s]
```

[9]:
```
from util import train
```

```
train.train(net3, batch_size=128, resize=(32,32), log_dir=log_dir+'/net3',␣
↪device=device, n_epochs=100)
```

```
WARNING:root:loading dataset
WARNING:root:loading done
  0%|          | 0/100 [00:00<?, ?it/s]
```

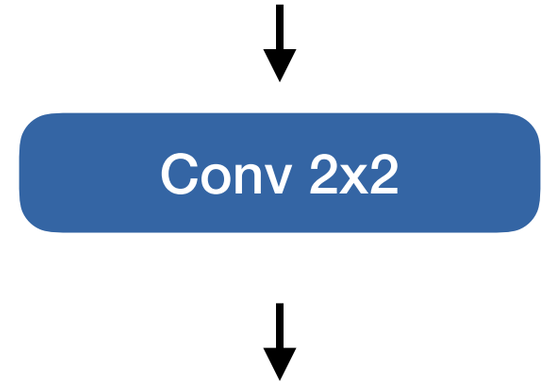
[ ]:

Receptive fields

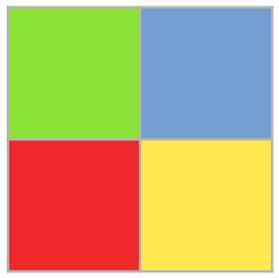
Receptive fields

- Can input $\mathbf{x}_{abc}$ affect output $\mathbf{z}_{ijk}$?

input: 3 x 3 x 1



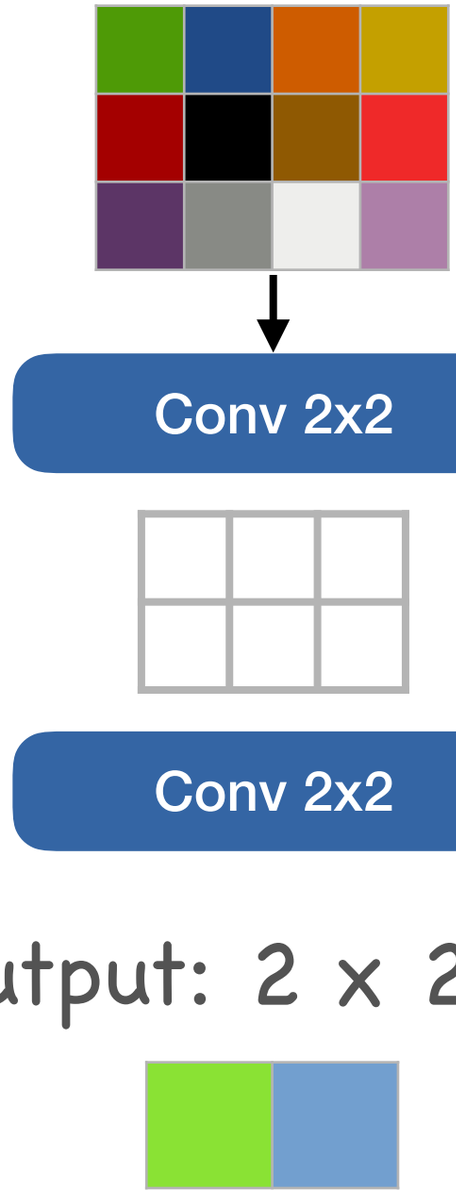
output: 2 x 2 x 1



Receptive fields

- Can input $\mathbf{x}_{abc}$ affect output $\mathbf{z}_{ijk}$?

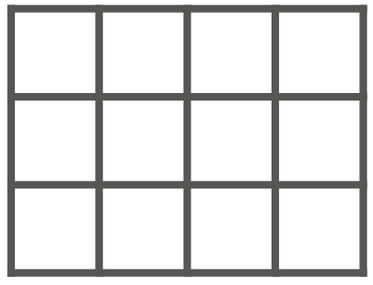
input: 3 x 4 x 1

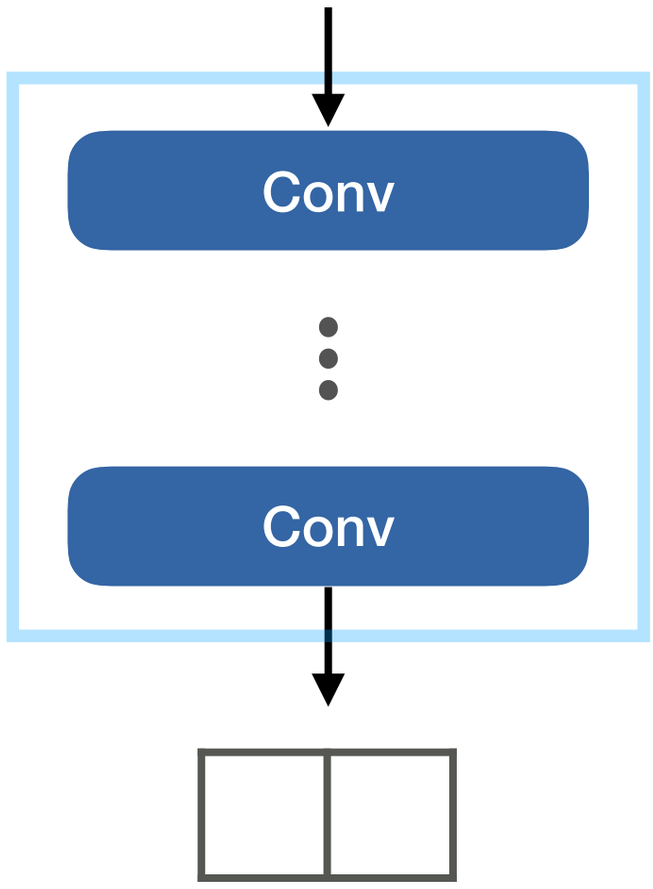


output: 2 x 2 x 1

How do we compute the receptive field

- Option 1: lots of math
- Option 2: Computationally
  - Feed a image of 0s to the network
  - Change a single element to NaN
    - See output changes



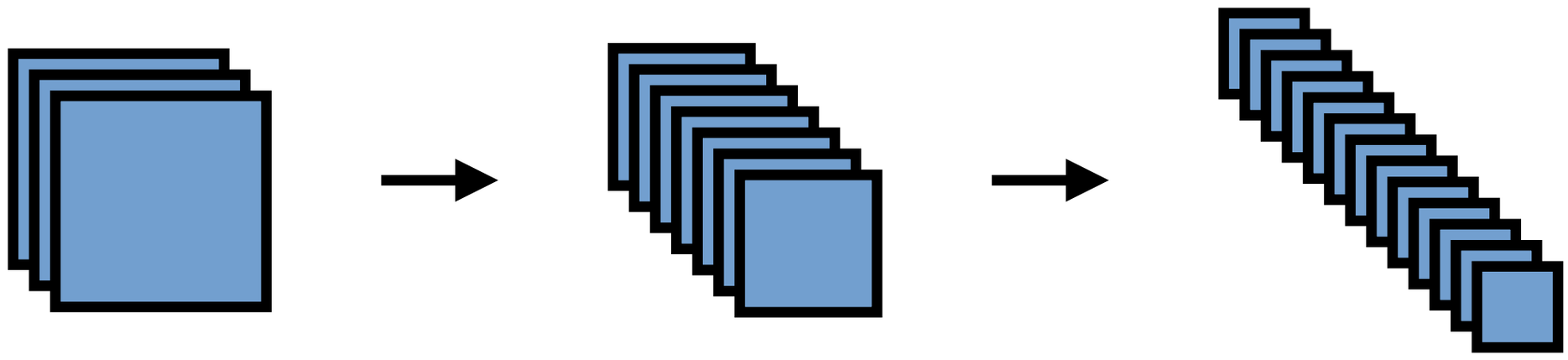


Structure of receptive field?

Design principles of convolutional networks

Use striding, increase channels

- Trade spatial resolution for channels
- Balance computation

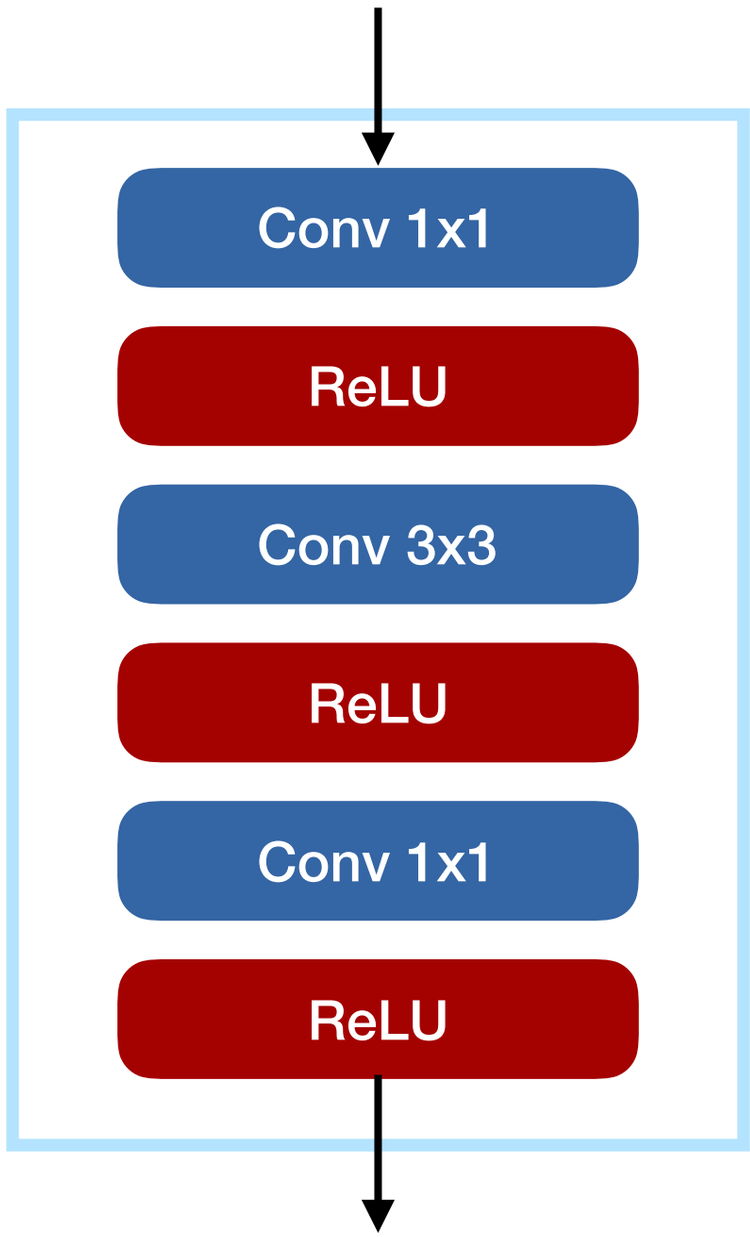


Keep kernels small

- 3x3 kernels almost everywhere
- exception:
  - first layer up to 7x7

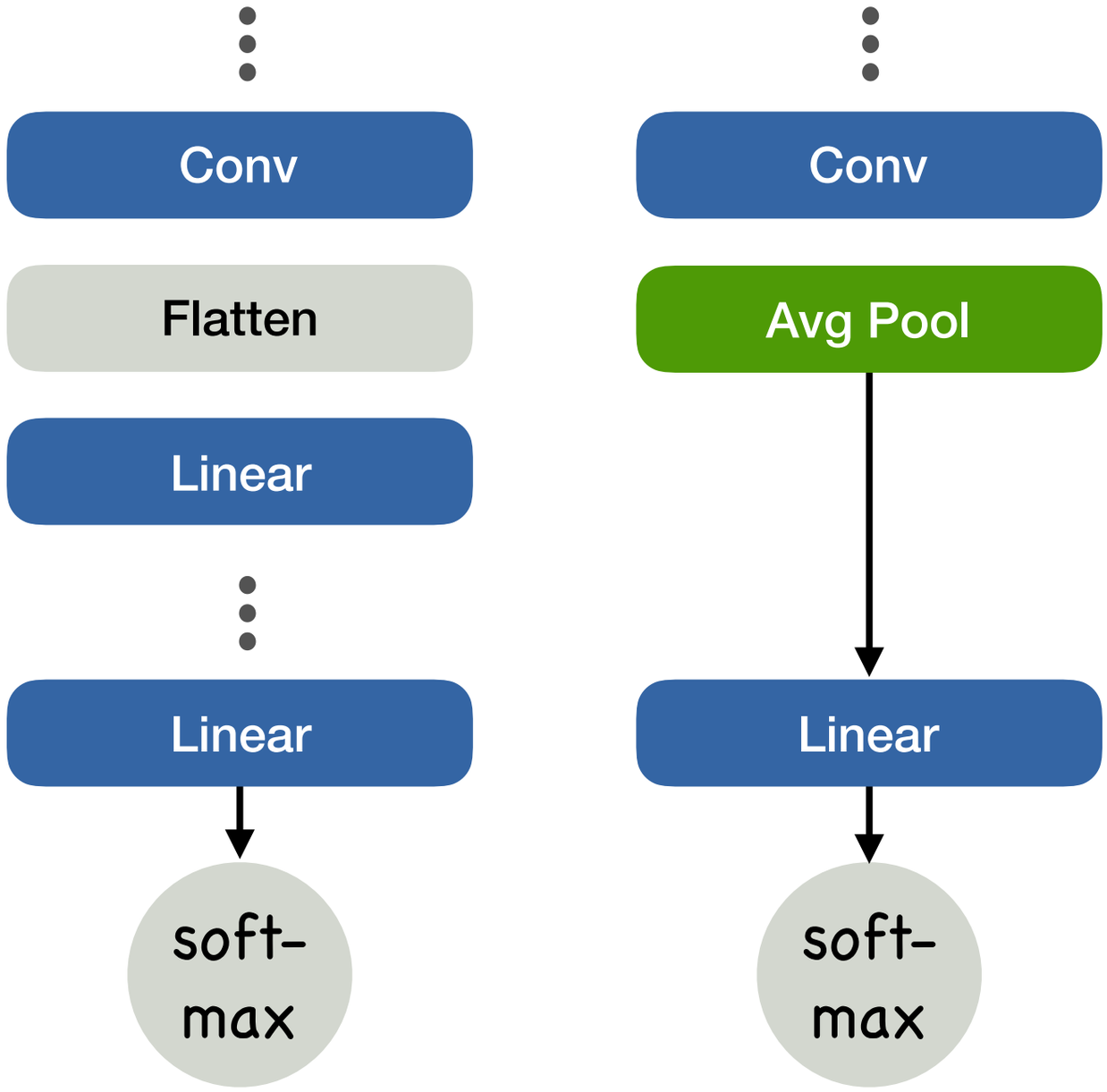
Repeat patterns

- First layer or two are special and not repeated
- All others usually follow a fixed pattern



All-convolutional

- Average in the end
- Fewer parameters
- Better training signal
- "Ensemble"/voting effect for testing



10

January 23, 2024

[1]:
```
%pylab inline
import torch
import sys
sys.path.append('..')
sys.path.append('../..')
from data import load
train_data, train_label = load.get_dogs_and_cats_data(resize=(128,128),␣
 ↪n_images=10)
device = torch.device('cuda') if torch.cuda.is_available() else torch.
 ↪device('cpu')
print('device = ', device)
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
device =  cuda
```

[2]:
```
class ConvNet(torch.nn.Module):
    class Block(torch.nn.Module):
        def __init__(self, n_input, n_output, stride=1):
            super().__init__()
            self.net = torch.nn.Sequential(
              torch.nn.Conv2d(n_input, n_output, kernel_size=3, padding=1,␣
 ↪stride=stride),
              torch.nn.ReLU(),
              torch.nn.Conv2d(n_output, n_output, kernel_size=3, padding=1),
              torch.nn.ReLU()
            )

        def forward(self, x):
            return self.net(x)

    def __init__(self, layers=[32,64,128], n_input_channels=3):
        super().__init__()
        L = [torch.nn.Conv2d(n_input_channels, 32, kernel_size=7, padding=3,␣
 ↪stride=2),
             torch.nn.ReLU(),
             torch.nn.MaxPool2d(kernel_size=3, stride=2, padding=1)]
```

```
        c = 32
        for l in layers:
            L.append(self.Block(c, l, stride=2))
            c = l
        self.network = torch.nn.Sequential(*L)
        self.classifier = torch.nn.Linear(c, 1)

    def forward(self, x):
        # Compute the features
        z = self.network(x)
        # Global average pooling
        z = z.mean(dim=[2,3])
        # Classify
        return self.classifier(z)[:,0]

net = ConvNet()
```

[3]:
```
z = net(train_data[:1])
print(z)
```

```
tensor([0.0218], grad_fn=<SelectBackward0>)
```

[4]:
```
%load_ext tensorboard
import tempfile
log_dir = tempfile.mkdtemp()
%tensorboard --logdir {log_dir} --reload_interval 1
```

```
<IPython.core.display.HTML object>
```

[5]:
```
from util import train
train.train(net, batch_size=128, resize=(128,128), log_dir=log_dir,␣
 ↪device=device, n_epochs=100)
```

```
WARNING:root:loading dataset
WARNING:root:loading done
```

```
  0%|          | 0/100 [00:00<?, ?it/s]
```

[ ]:
```

```

Deep representations and exploiting the structure of the data

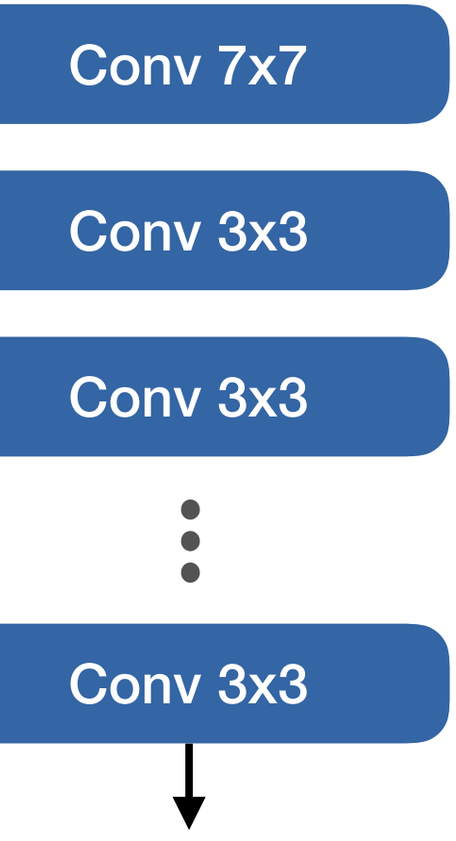
Structure of input data

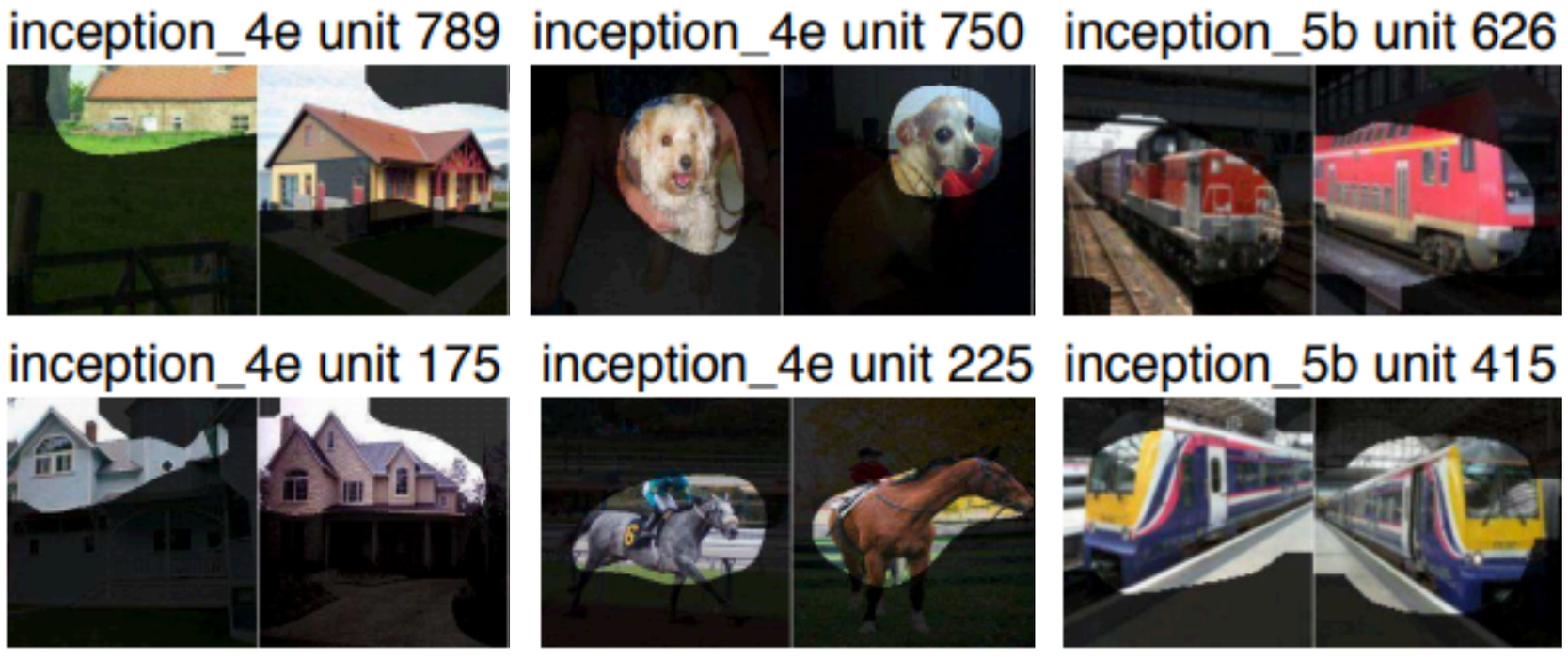
- Images
- Repeating patterns
- at various scales

7x7 patches

Structure of convolutional networks

- Exploit repeating structure of images



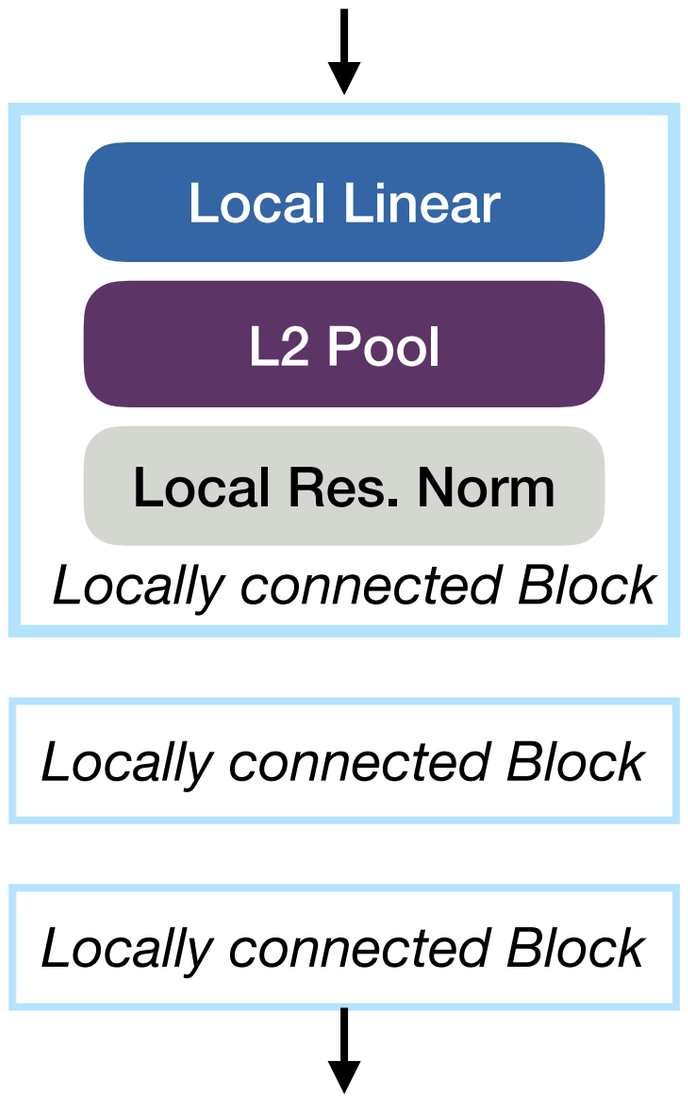


Network Dissection: Quantifying Interpretability of Deep Visual Representations, D. Bau etal, CVPR 2017

What do networks learn?

Linear layered networks do not work well on images

- Largest linear network for computer vision
  - Locally connected



Building high-level features using large-scale unsupervised learning, Q. Le etal, ICML 2012
Large Scale Distributed Deep Networks, J. Dean etal. NeurIPS 2012

12

January 23, 2024

[1]:
```
%pylab inline

import json
import numpy as np
from PIL import Image

import torch
import torchvision.models as models
import torchvision.transforms as transforms
device = torch.device('cuda') if torch.cuda.is_available() else torch.
 ↪device('cpu')
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
```

[ ]:
```
model = models.resnet18(pretrained=True)
model.eval()
class_idx = json.load(open("network_vis/imagenet_class_index.json"))
```

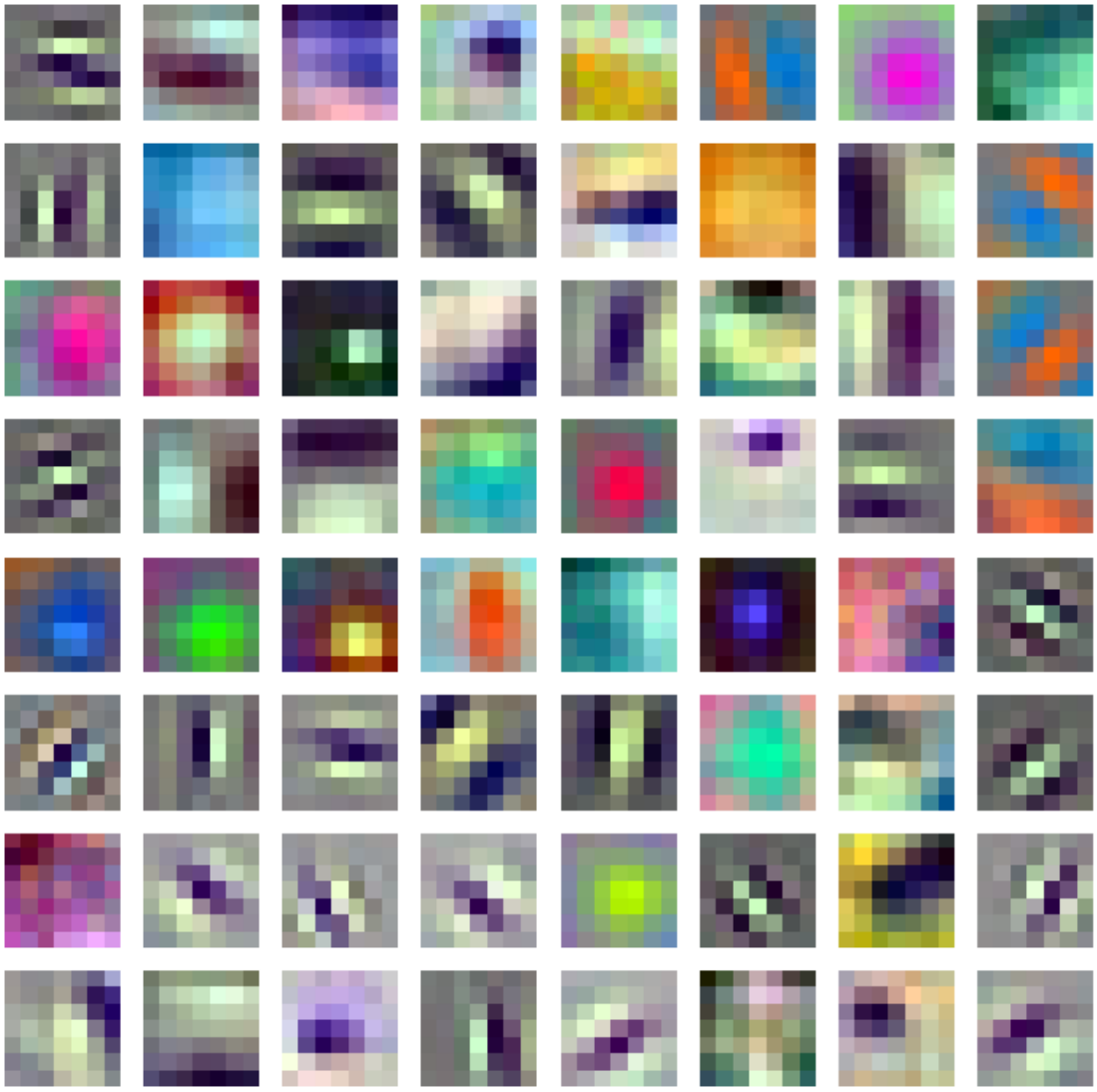
[4]:
```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])])
```

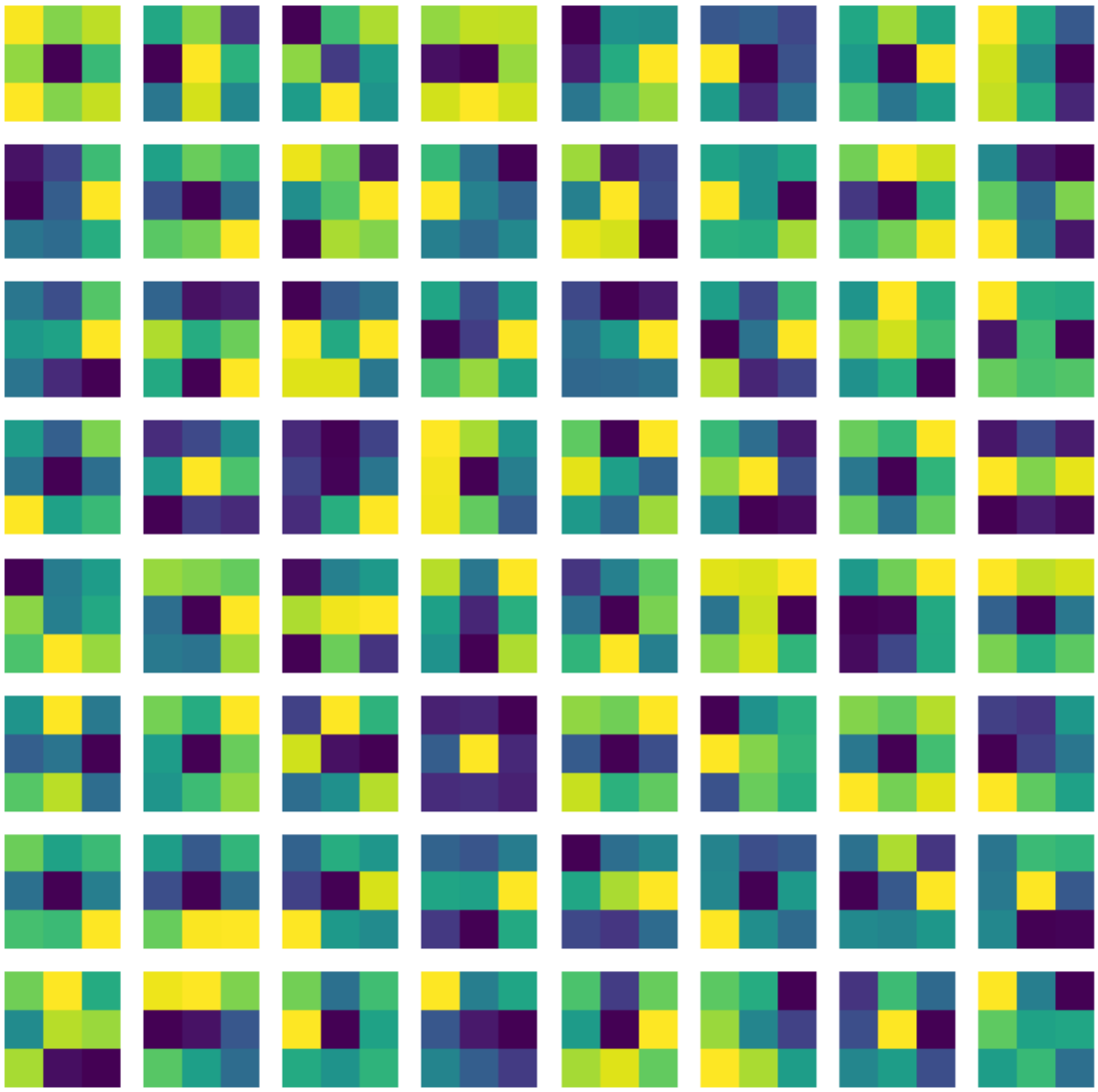
[5]:
```
image = transform(Image.open('network_vis/dog.jpg'))
plt.imshow(plt.imread('network_vis/dog.jpg'))

predictions = model(image.unsqueeze(0)).detach().numpy()[0]

print('Ranking: class (score)')
for i, idx in enumerate(np.argsort(predictions)[-10:][::-1]):
    print('%d: %s (%.04f)' % (
            i,
            class_idx[str(idx)][1],
            predictions[idx]))
```

```
Ranking: class (score)
0: wire-haired_fox_terrier (10.5043)
1: toy_terrier (9.4139)
2: English_foxhound (9.1540)
3: Ibizan_hound (8.7189)
4: Chihuahua (8.0401)
5: Lakeland_terrier (7.9056)
6: Border_terrier (7.8533)
7: Brittany_spaniel (7.7991)
8: Walker_hound (7.7619)
9: basenji (7.7018)
```

[6]:
```python
def visualize_layer(weight):
    fig=plt.figure(figsize=(8, 8))

    for i in range(64):
        x = weight[i, ...].transpose([1, 2, 0])
        x = (x - np.min(x))/np.ptp(x)

        fig.add_subplot(8, 8, i + 1)
        if x.shape[2] == 3:
            imshow(x)
```

```
        else:
            imshow(x[:,:,0])
        axis('off')

    show()

visualize_layer(model.conv1.weight.data.cpu().numpy())
visualize_layer(model.layer1[0].conv1.weight.data.cpu().numpy())
```





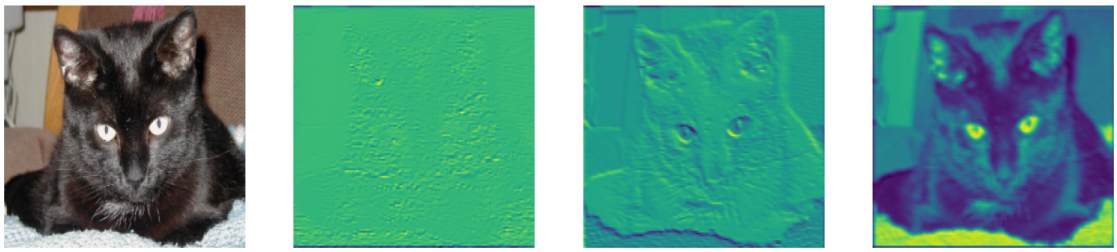
[7]:
```
import sys
sys.path.append('..')
sys.path.append('../..')
from data import load
valid_data = load.get_dogs_and_cats('valid', resize=None, batch_size=128,␣
 ↪is_resnet=True)
```

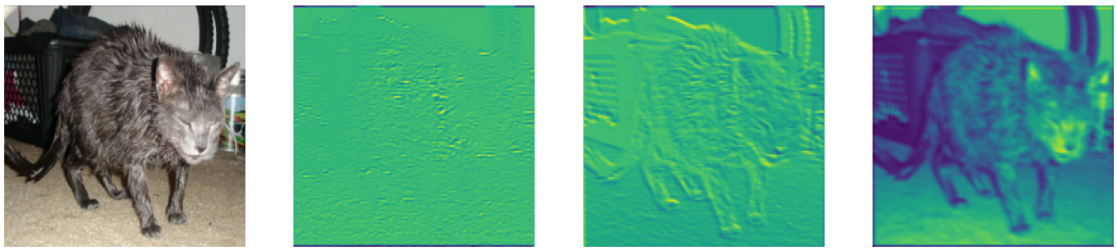
[8]:
```
def undo_transform(x):
    mean = np.array([0.485, 0.456, 0.406])[np.newaxis, :, np.newaxis, np.
 ↪newaxis]
    std = np.array([0.229, 0.224, 0.225])[np.newaxis, :, np.newaxis, np.newaxis]
    return (x * std) + mean
```

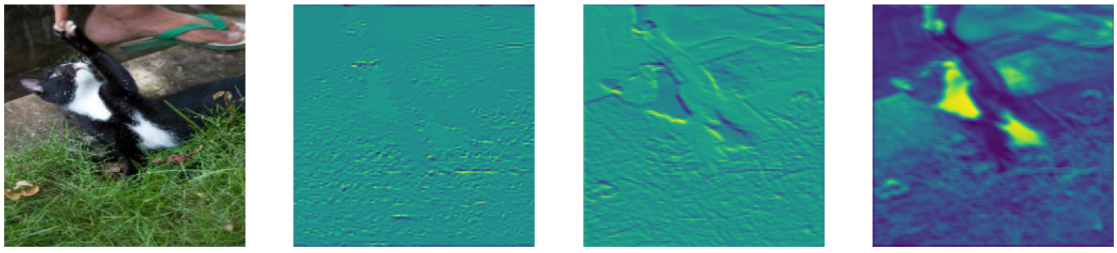
[9]:
```
all_data = []
C1 = []
C2 = []
C3 = []

def store_activation(L, m, grad_in, grad_out):
    L.append(grad_out.detach().cpu().numpy())

model.to(device)

try:
    h1 = model.conv1.register_forward_hook(lambda *args: store_activation(C1,␣
 ↪*args))
    h2 = model.layer1[0].conv1.register_forward_hook(lambda *args:␣
 ↪store_activation(C2, *args))
    h3 = model.layer2[0].conv1.register_forward_hook(lambda *args:␣
 ↪store_activation(C3, *args))

    for it, (data, label) in enumerate(valid_data):
        all_data.append(undo_transform(data.numpy()))
        if device is not None:
            data, label = data.to(device), label.to(device)
        result = model(data)
finally:
    h1.remove()
    h2.remove()
    h3.remove()

all_data = np.vstack(all_data)
C1 = np.vstack(C1)
C2 = np.vstack(C2)
C3 = np.vstack(C3)
```

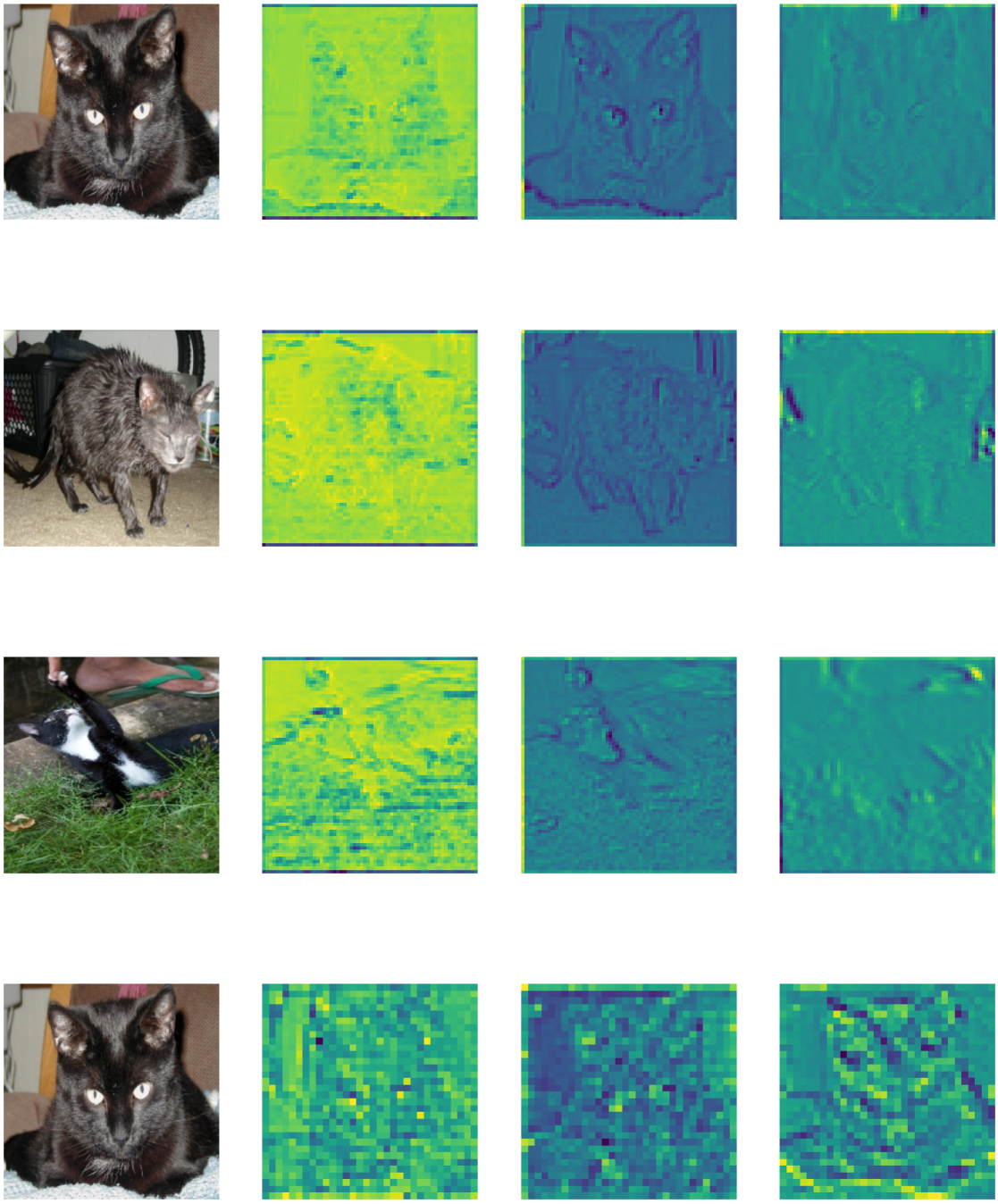
[10]:
```
def to_pil(x):
    x = x.transpose((1, 2, 0))
    x -= x.min(0).min(0)[None,None]
    x /= x.max(0).max(0)[None,None]
    return Image.fromarray((x * 255).astype('uint8'))

def viz_act(img, act):

    num_acts = len(act)
    img = to_pil(img)

    fig=plt.figure(figsize=(20, 20))

    fig.add_subplot(1, 1+num_acts, 1)
```

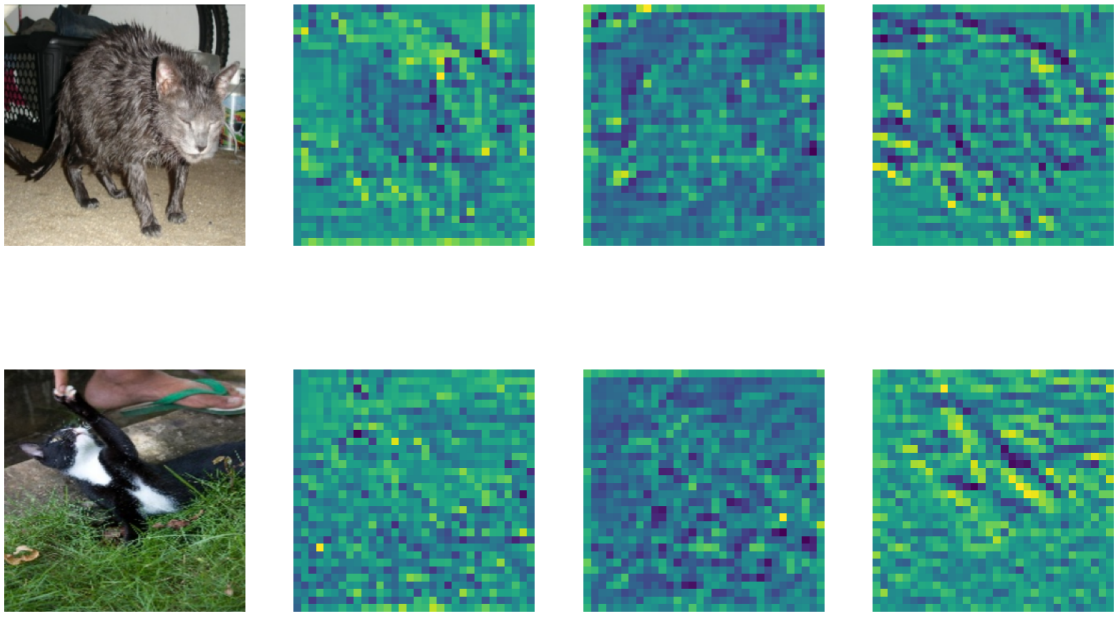
```
    imshow(img)
    axis('off')

    for i, c in enumerate(act):
        fig.add_subplot(1, 1+num_acts, i + 2)
        imshow(c)
        axis('off')

    show()

for i in range(3):
    viz_act(all_data[i], C1[i, :3])


for i in range(3):
    viz_act(all_data[i], C2[i, :3])

for i in range(3):
    viz_act(all_data[i], C3[i, :3])
```



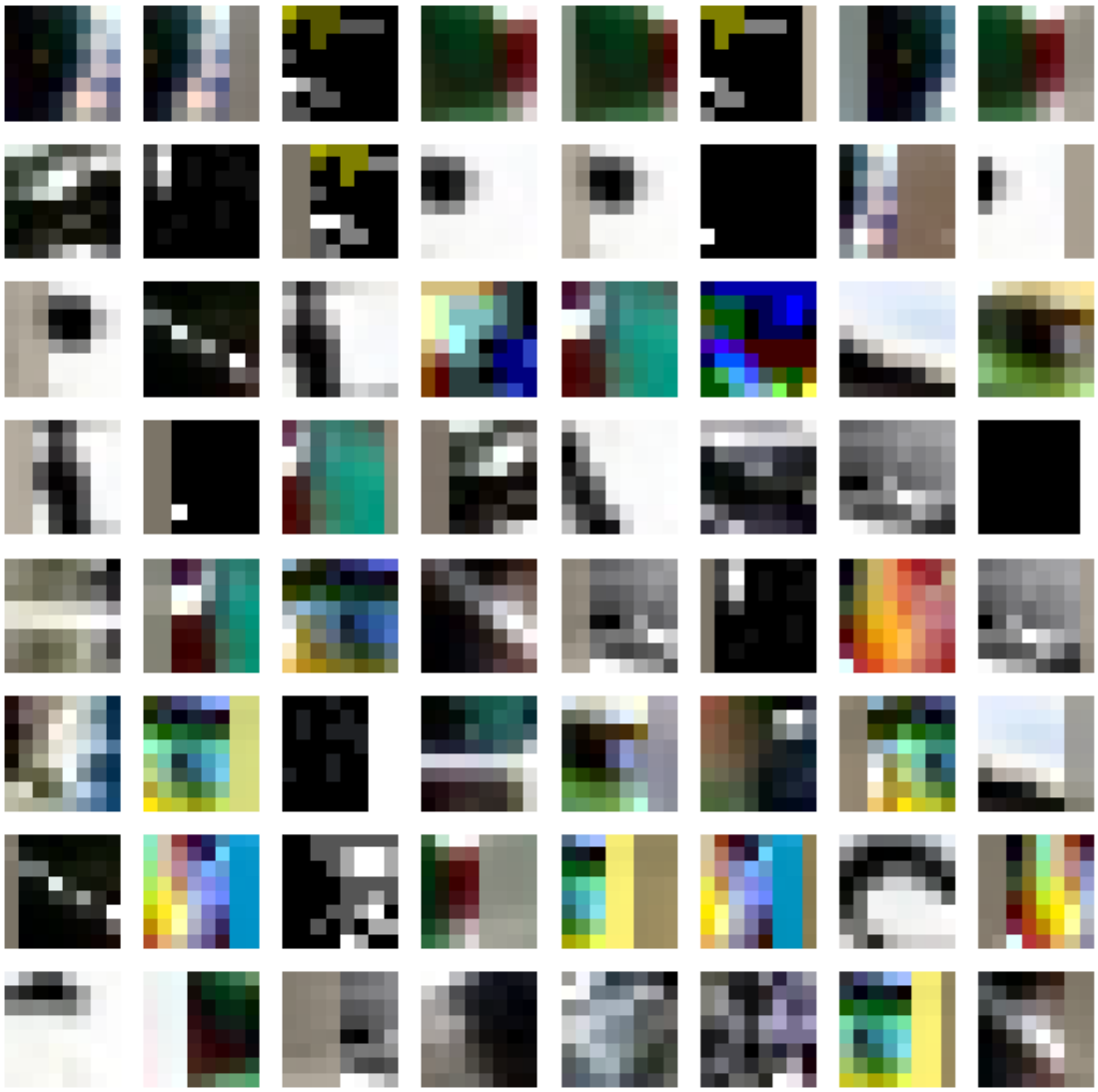


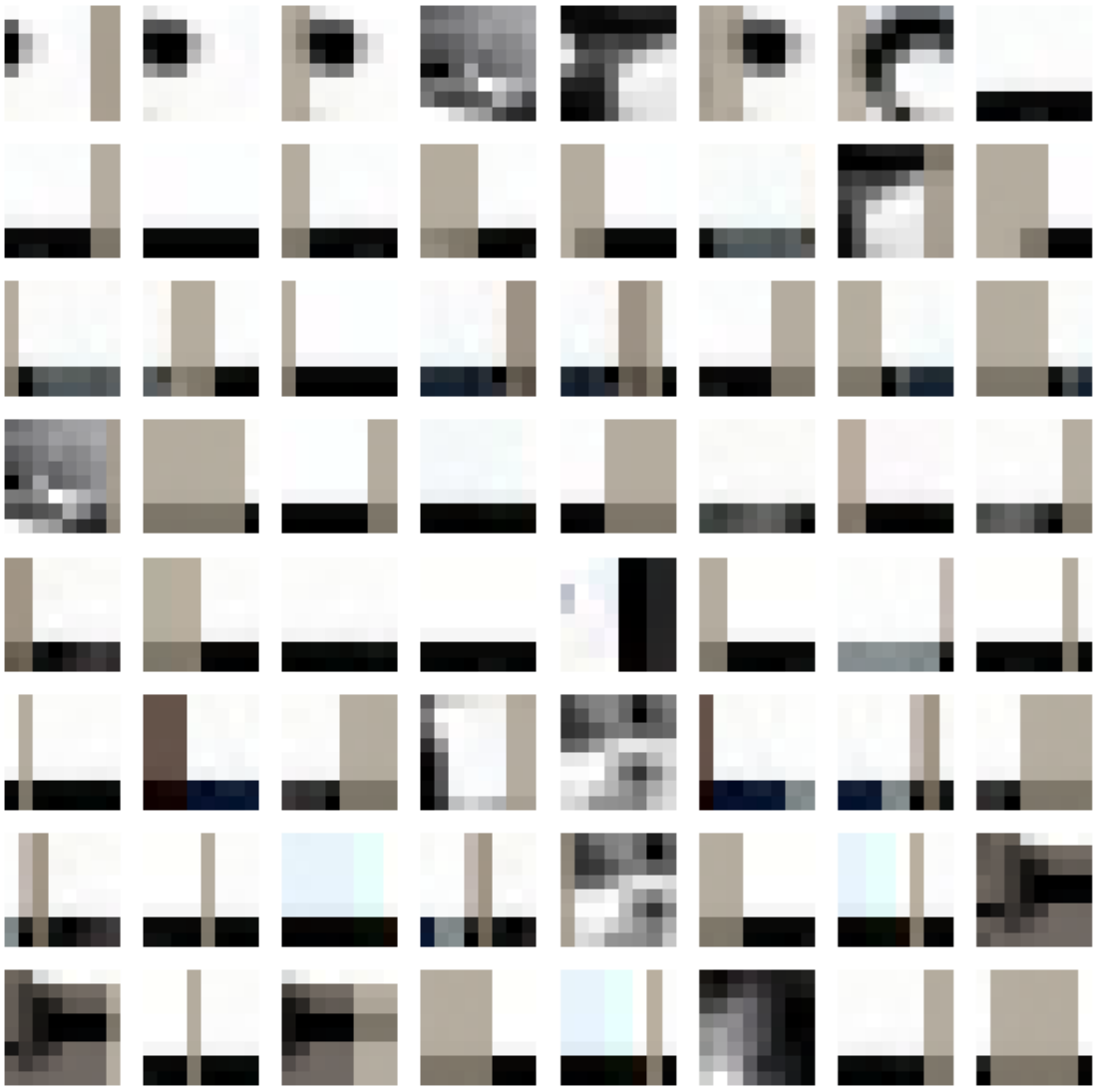




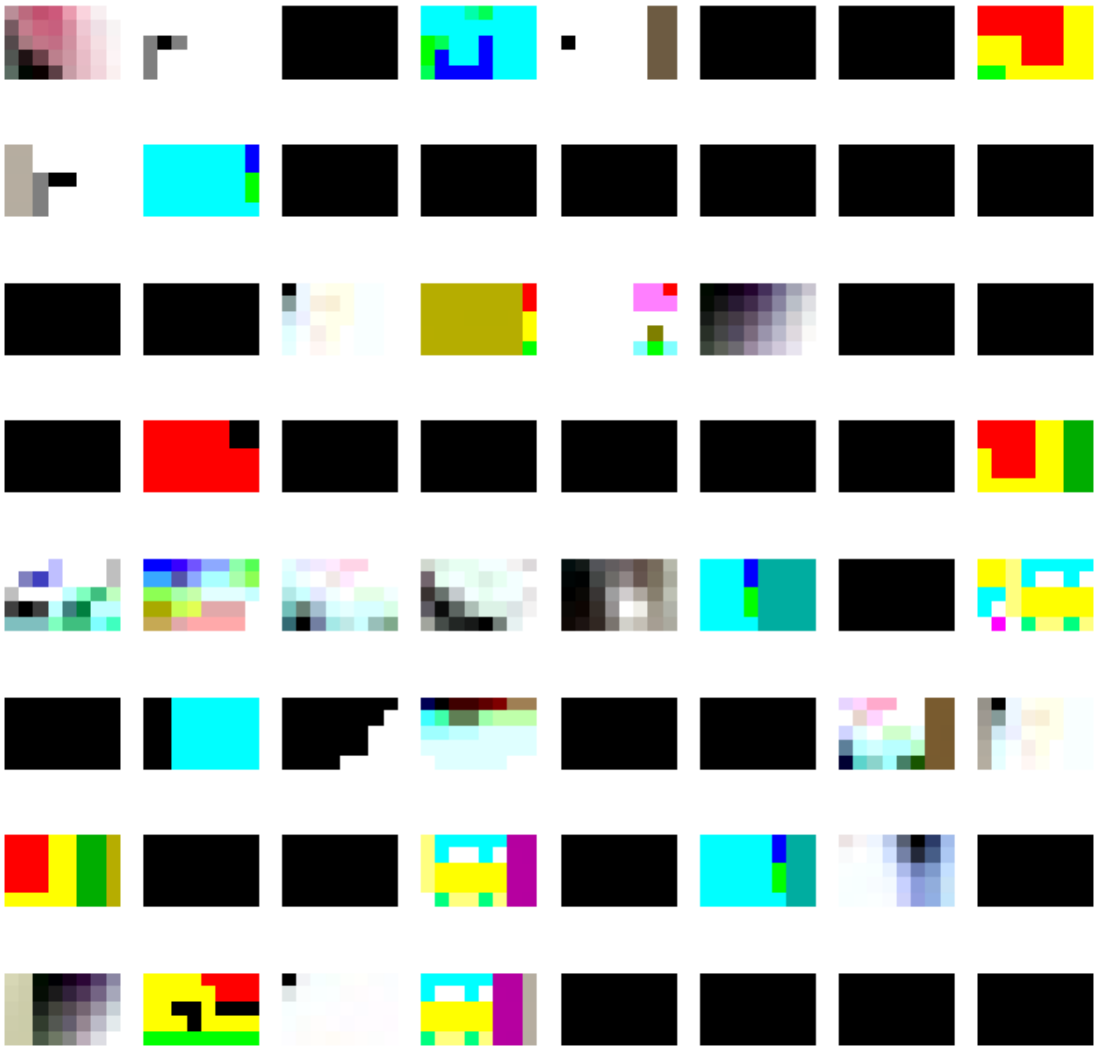


[11]:
```
def find_max_act(C, filter, W=10, img_size=[224, 224]):
    fig=plt.figure(figsize=(8, 8))

    filter_act = C[:, filter]
    for i, idx in enumerate(np.argsort(-filter_act, axis=None)[:64]):
        max_pos = list(np.unravel_index(idx, filter_act.shape))

        max_pos[1] = int(max_pos[1] * float(img_size[0]) / filter_act.shape[1])
        max_pos[2] = int(max_pos[2] * float(img_size[1]) / filter_act.shape[2])

        img = to_pil(all_data[max_pos[0],
                              :,
                              max(max_pos[1] - W, 0) : max_pos[1] + W,
                              max(max_pos[2] - W, 0) : max_pos[2] + W])

        fig.add_subplot(8, 8, i + 1)
        imshow(img)
        axis('off')

    show()

find_max_act(C1, filter=0, W=4)#, img_size=img.size)
find_max_act(C1, filter=1, W=4)#, img_size=img.size)
find_max_act(C1, filter=2, W=4)#, img_size=img.size)
```



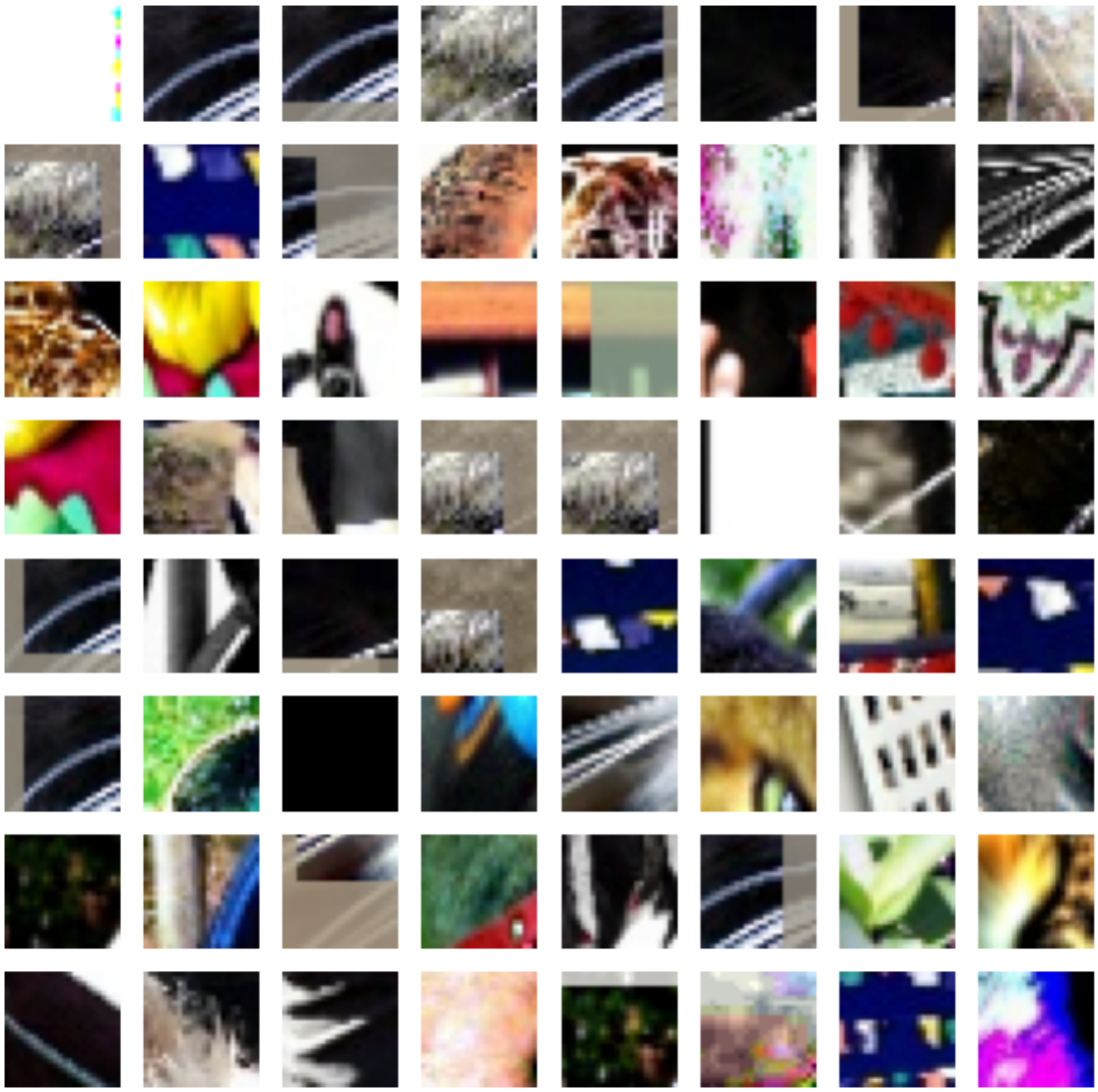


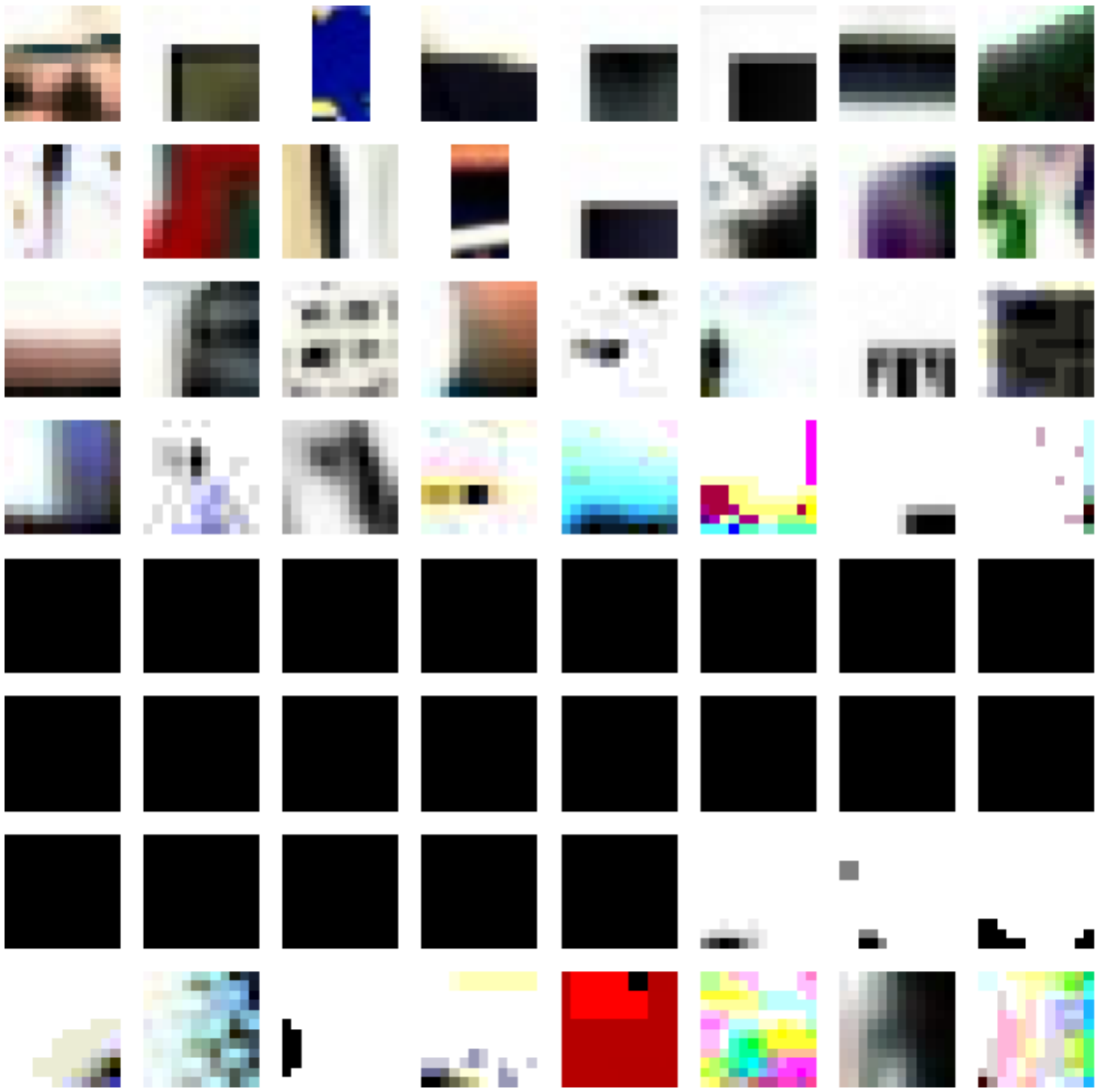
```
/tmp/ipykernel_2833/1084252010.py:4: RuntimeWarning: invalid value encountered
in divide
  x /= x.max(0).max(0)[None,None]
/tmp/ipykernel_2833/1084252010.py:5: RuntimeWarning: invalid value encountered
in cast
  return Image.fromarray((x * 255).astype('uint8'))
```

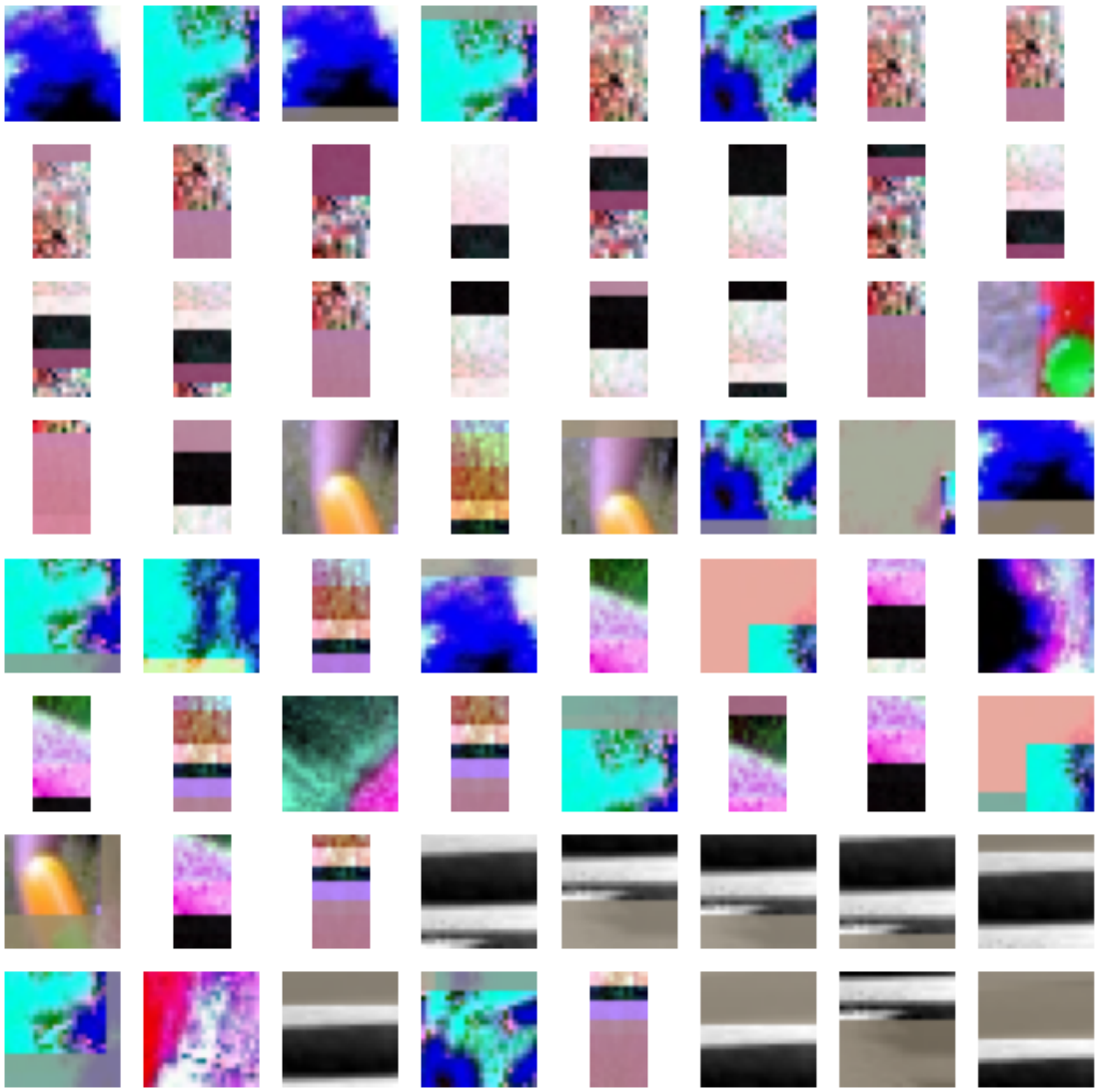


[12]:
```
find_max_act(C2, filter=0, W=12)#, img_size=img.size)
find_max_act(C2, filter=1, W=12)#, img_size=img.size)
find_max_act(C2, filter=2, W=12)#, img_size=img.size)
```

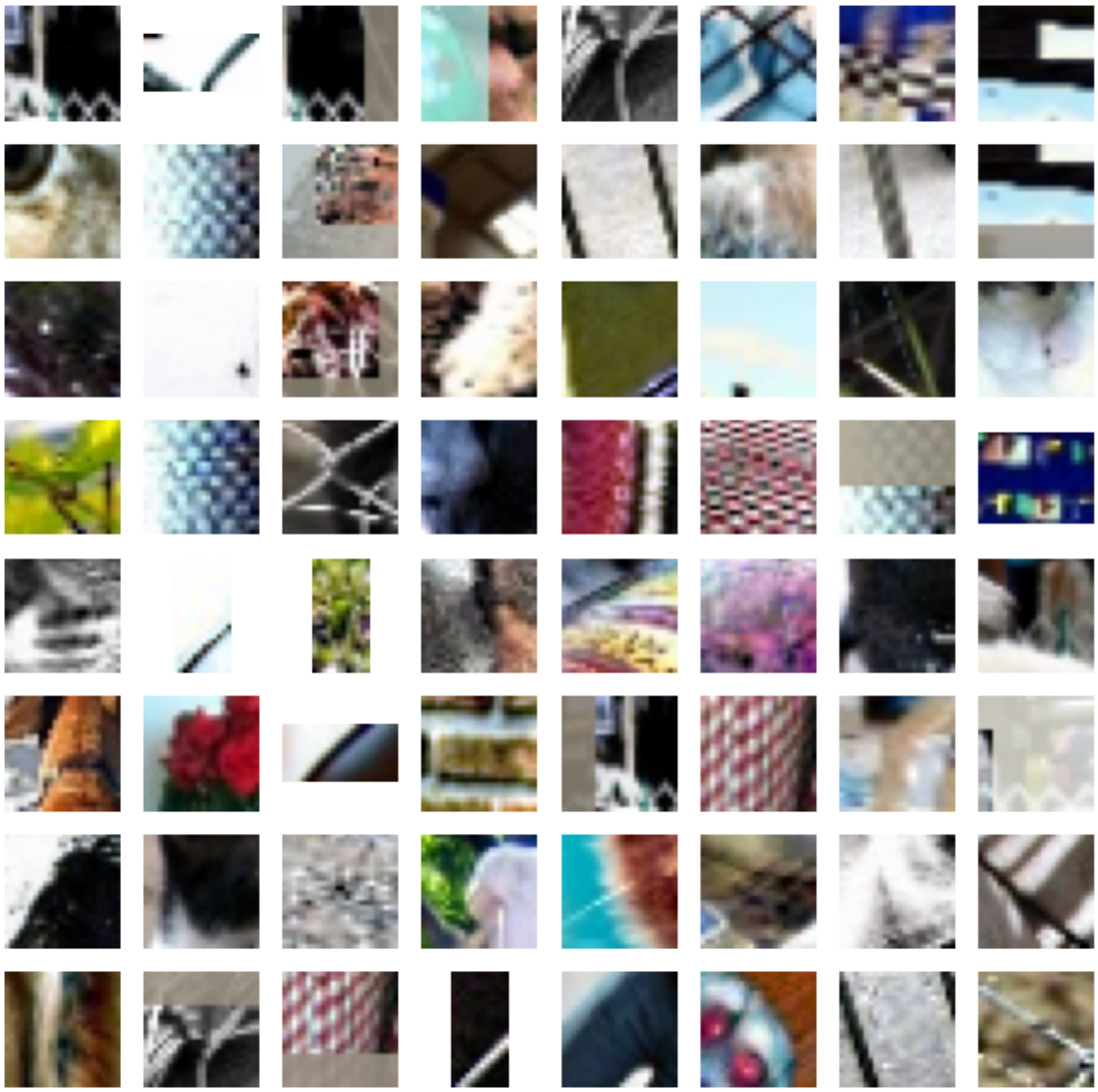
```
/tmp/ipykernel_2833/1084252010.py:4: RuntimeWarning: invalid value encountered
in divide
  x /= x.max(0).max(0)[None,None]
/tmp/ipykernel_2833/1084252010.py:5: RuntimeWarning: invalid value encountered
in cast
  return Image.fromarray((x * 255).astype('uint8'))
```







[13]:
```
find_max_act(C3, filter=0, W=12)#, img_size=img.size)
find_max_act(C3, filter=1, W=12)#, img_size=img.size)
find_max_act(C3, filter=2, W=12)#, img_size=img.size)
```



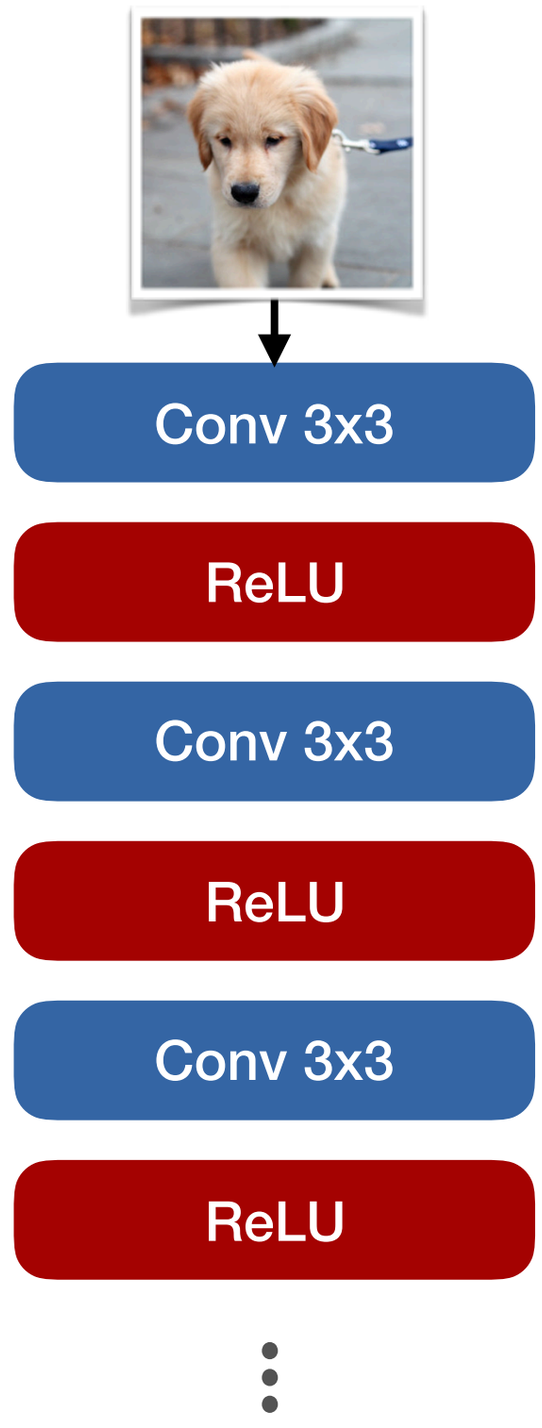
[ ]:

Dilation

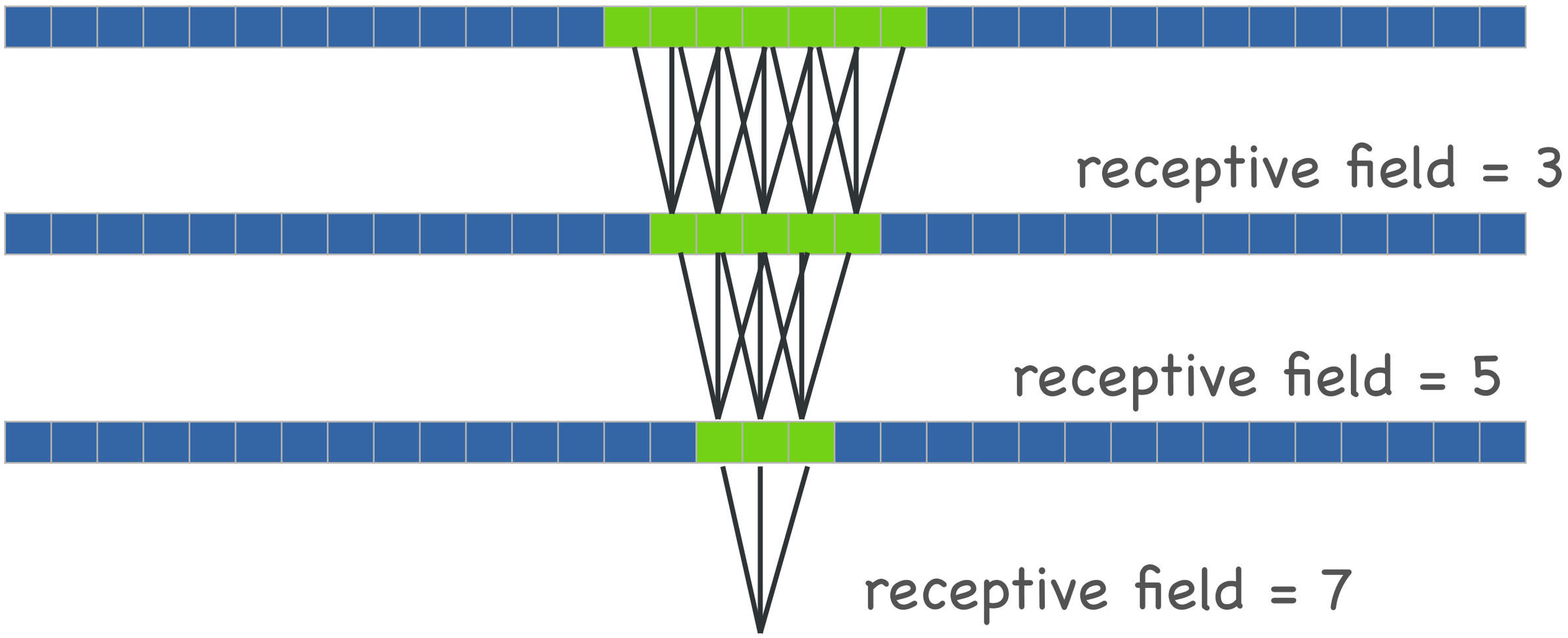
Segmentation

Receptive field

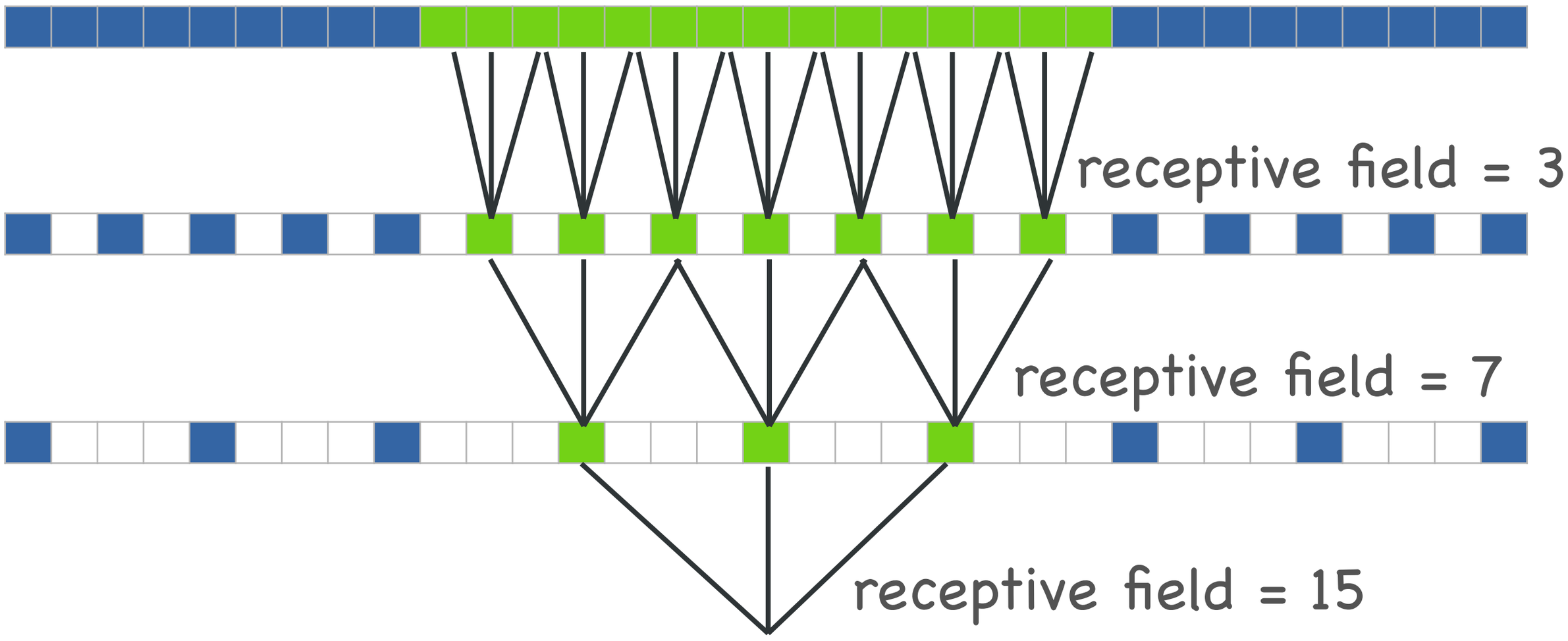
- How to increase receptive field?
- Large kernel size
- Striding



Receptive field

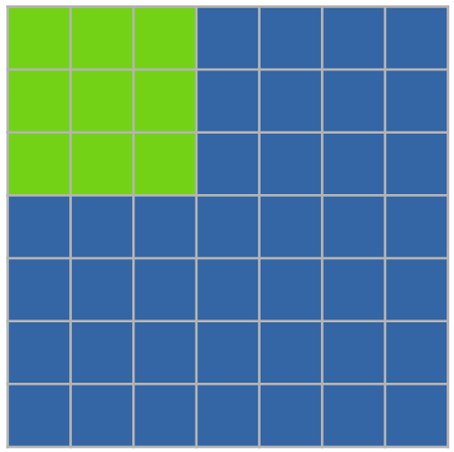


Receptive field and striding



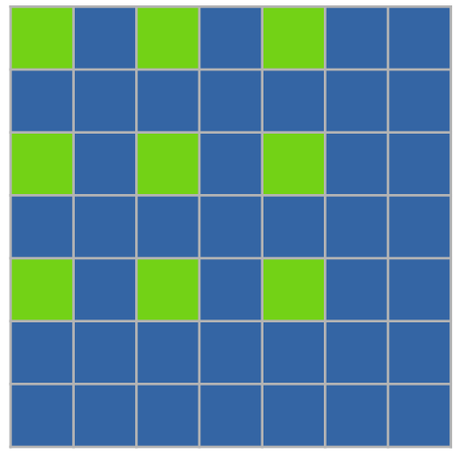
Dilation

- Add 0-padding between values in convolutional kernel



*

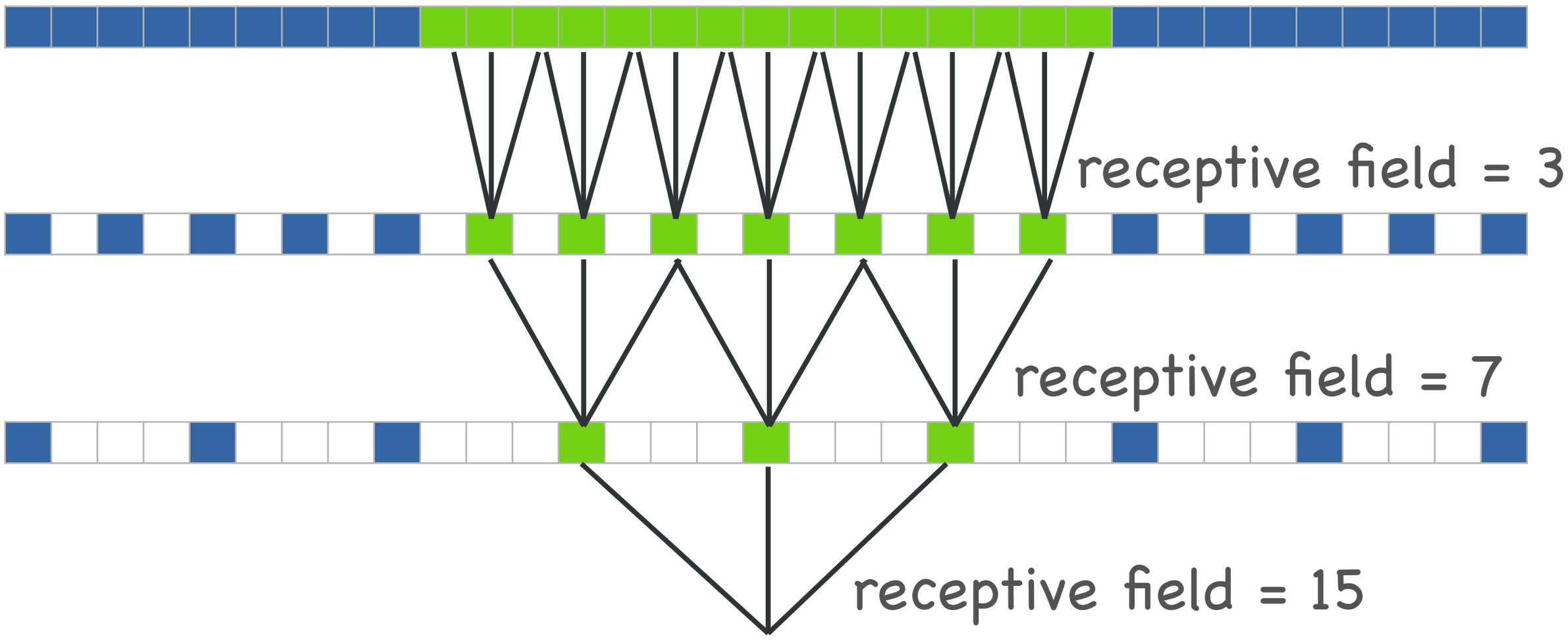
a	b	c
d	e	f
g	h	i



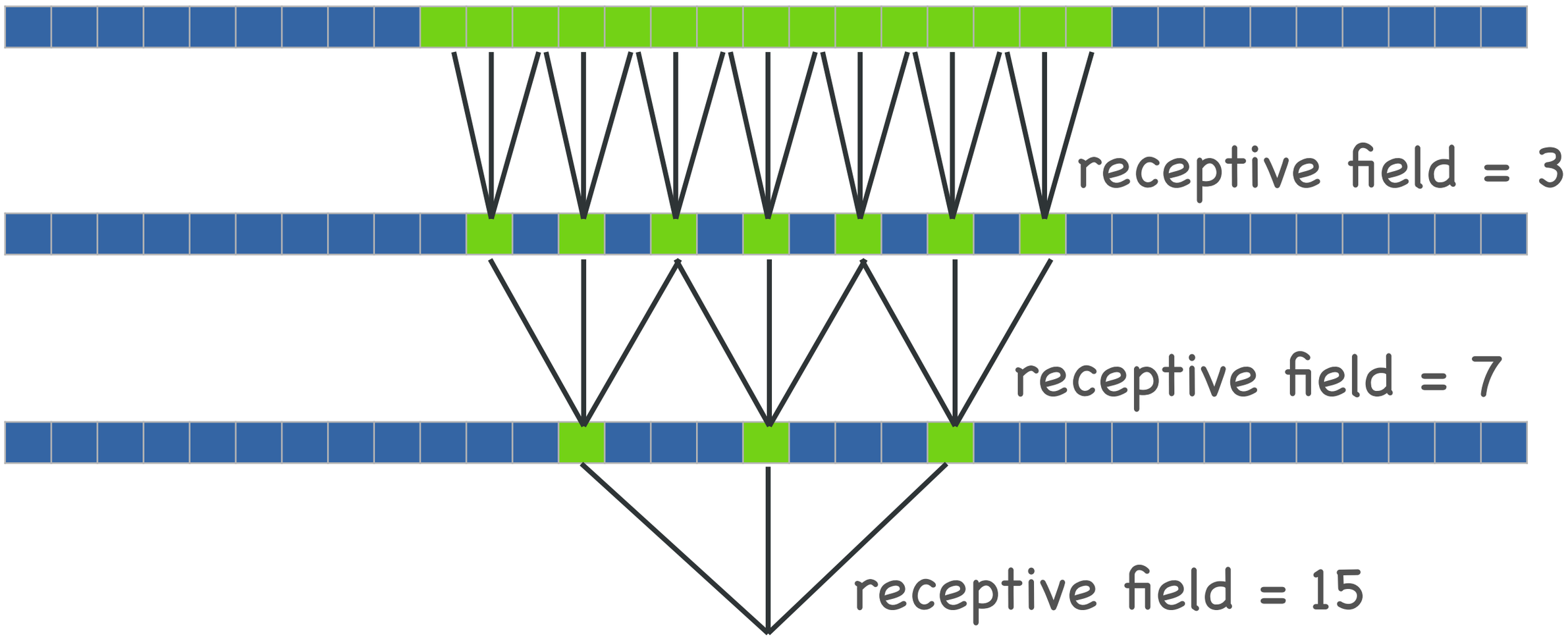
*

a		b		c
d		e		f
g		h		i

Dilation vs striding



Dilation vs striding

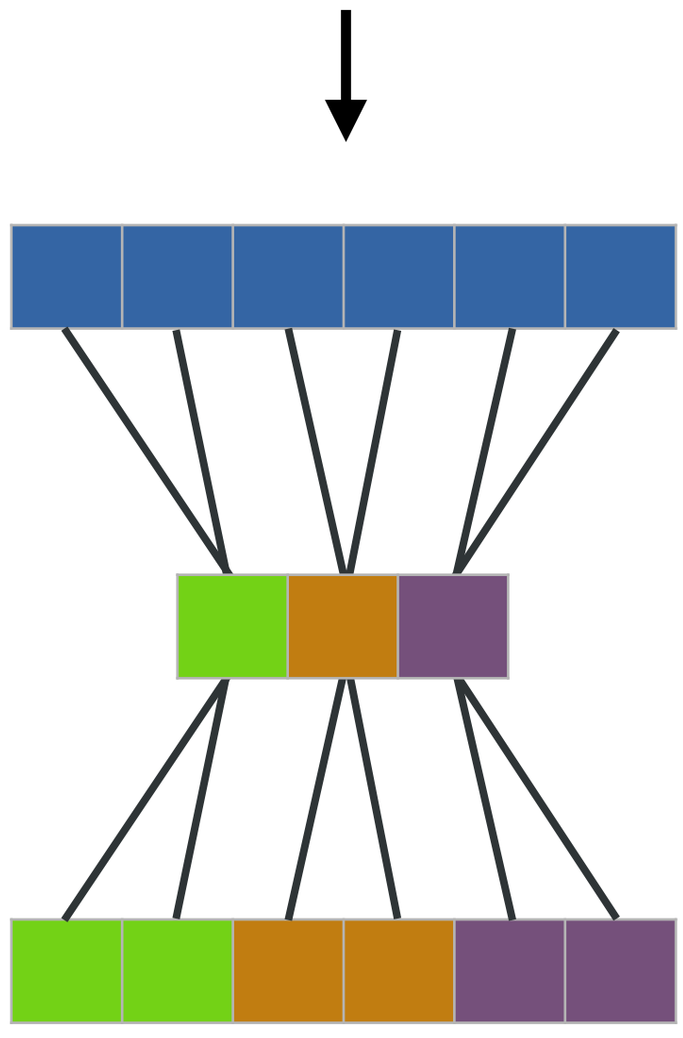


The many names of dilation

- hole
- a trous

Up-convolution

Inverse of strided convolution

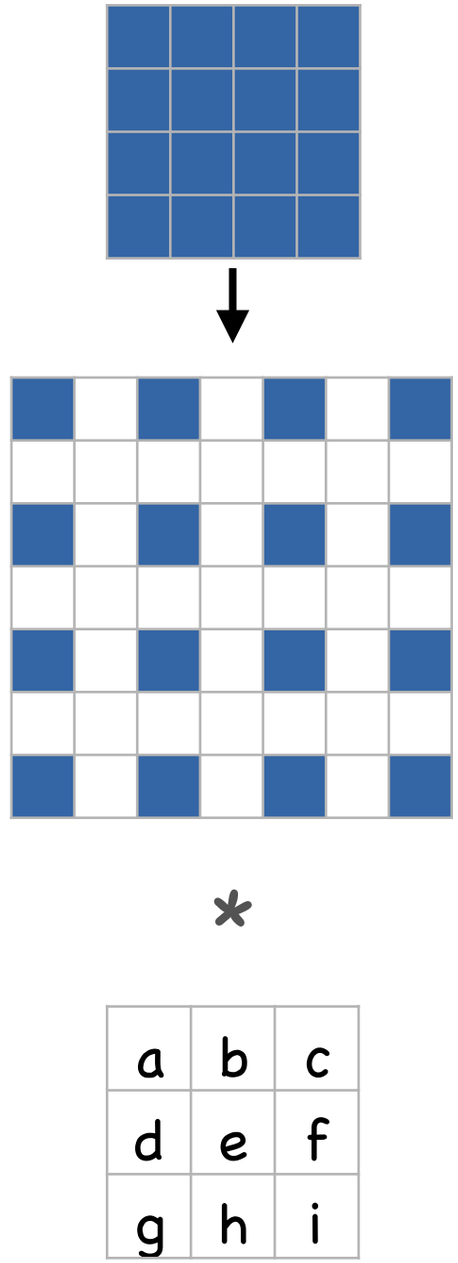


Strided conv

Upsampling

Up-convolution

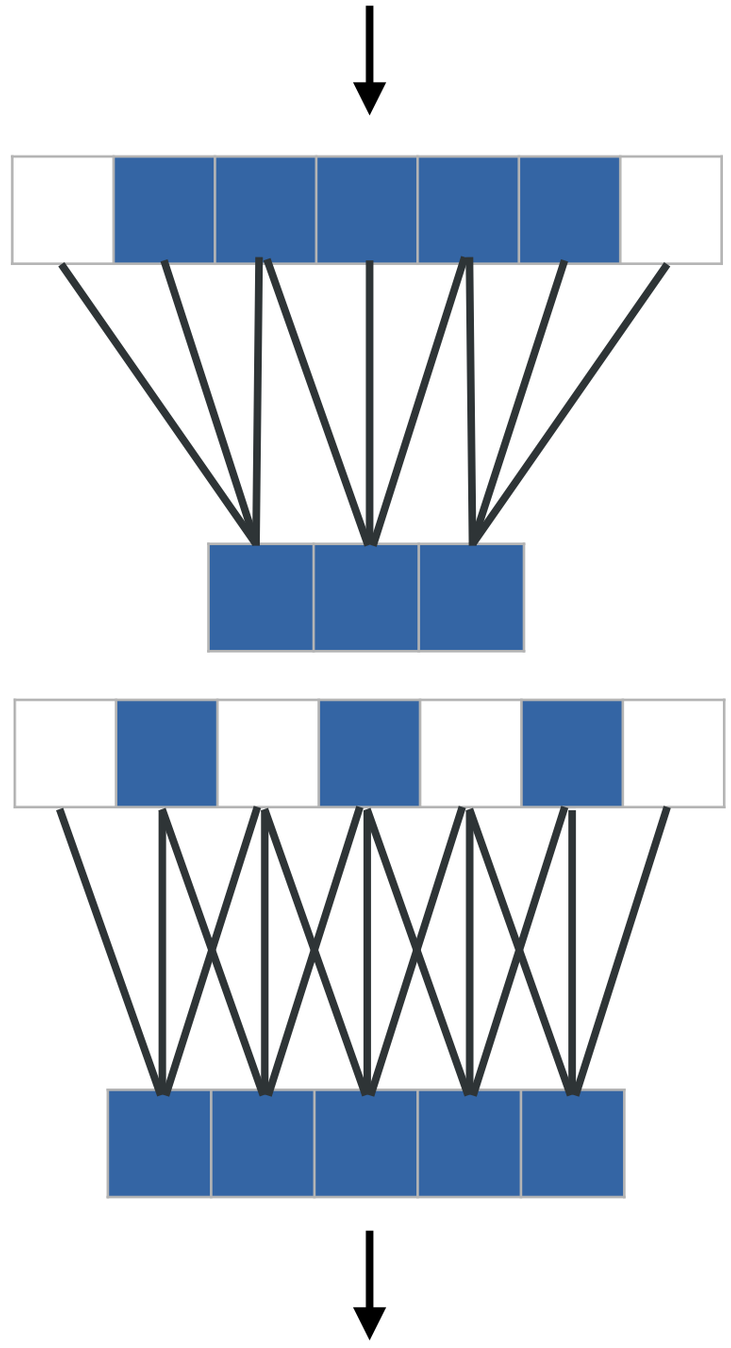
- Dilation of the input



Rounding

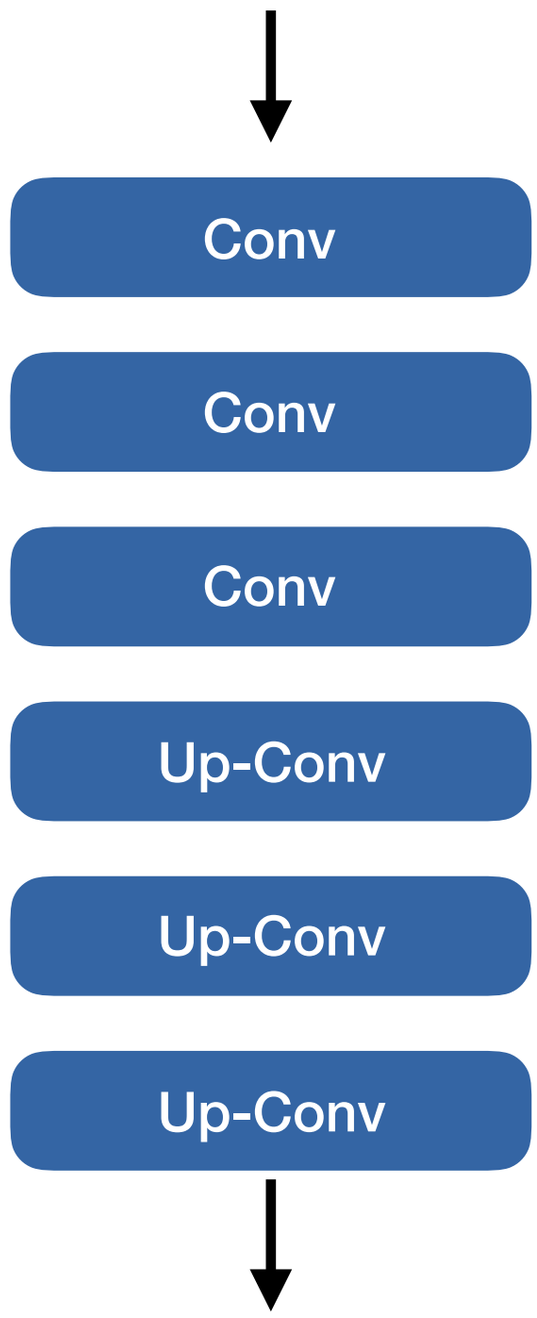
- Strided convolution rounds down

- How to correct for this?



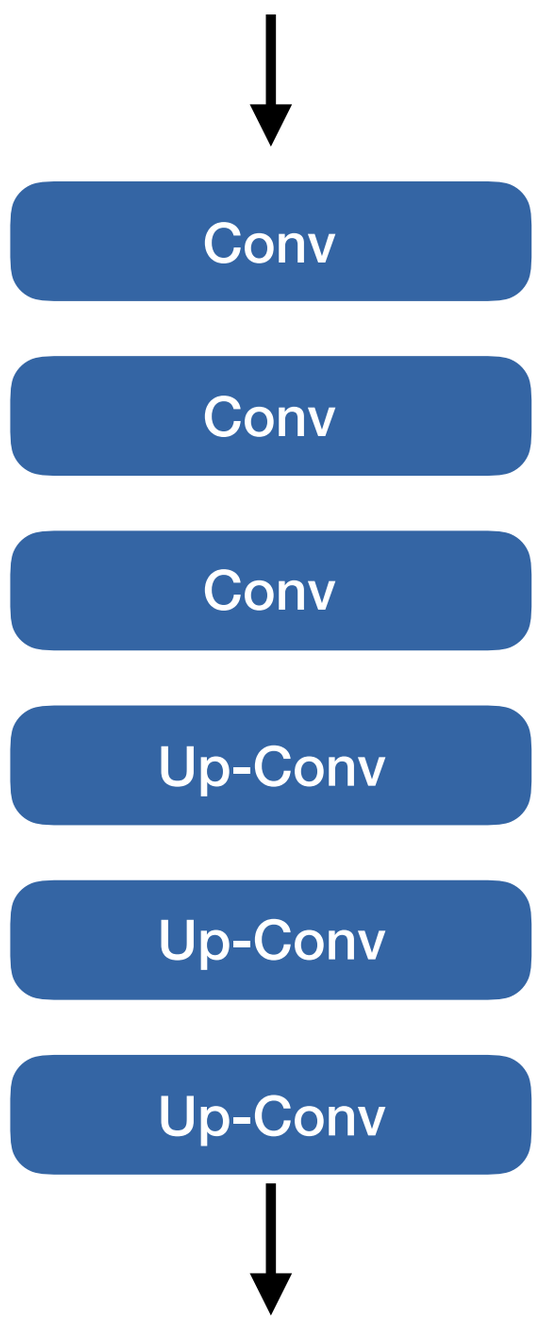
Up-convolution in action

- Used closer to output layers



Up-convolutions and skip connections

- Provides lower-level high-resolution features to output



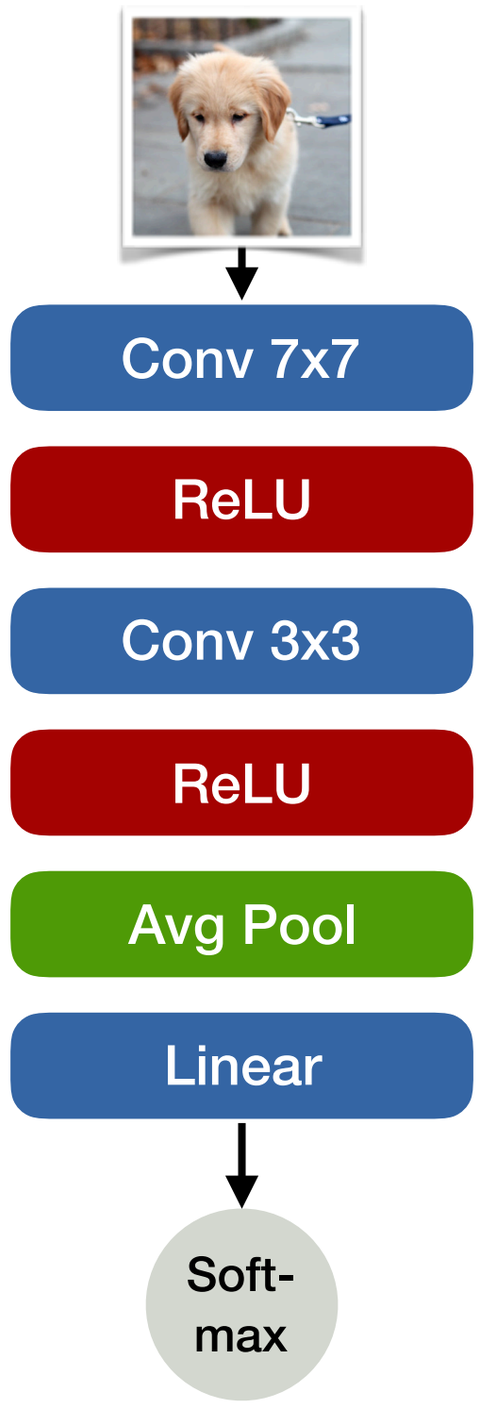
The many names of up-convolution

- Transpose convolution
- "Deconvolution"
- fractionally strided convolutions

Summary

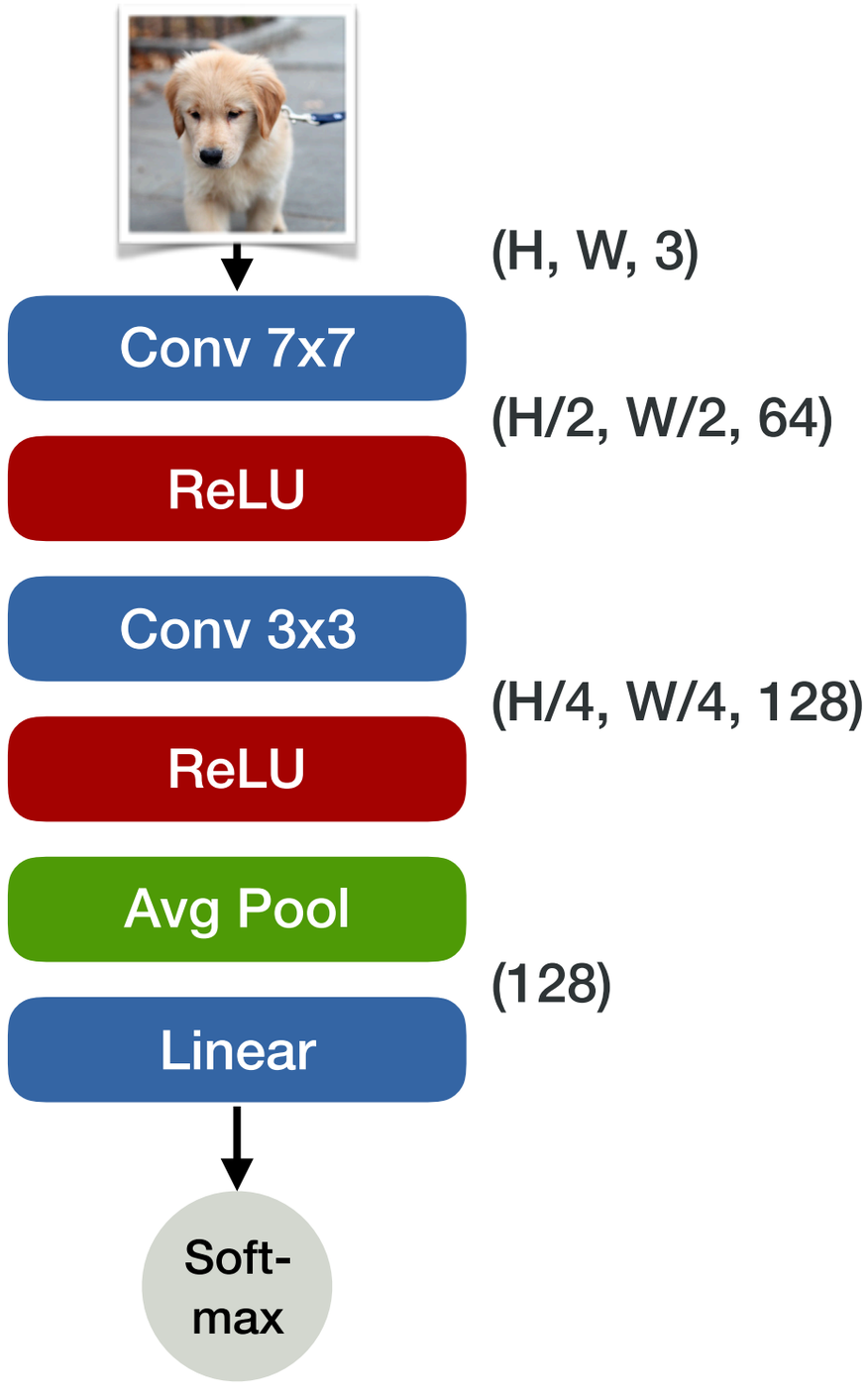
Convolutional networks

- Linear transformations
  - Convolutions
- Convolutional non-linearities
  - Pooling



ConvNet Design

- Stride and increase channels
- Small kernel size
- All convolutional
- Up-convolution close to output (optional)



Applications of ConvNets

- Autonomous vehicles
- Analyze medical images
- Geoscience: Analyze scans of rock formations to find oil