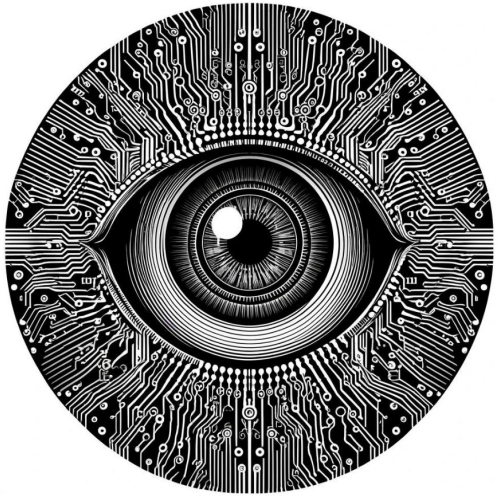
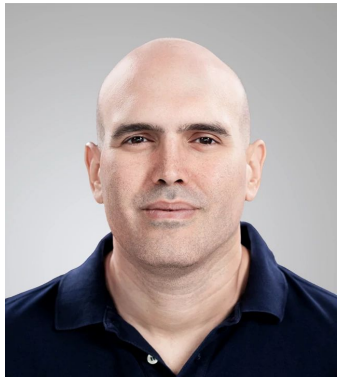
Workshop 8 - Image Upsampling and Semantic Segmentation with U-net



Antonio Rueda-Toicen

About me

- AI Researcher at Hasso Plattner Institute, AI Engineer & DevRel for Voxel51
- Organizer of the Berlin Computer Vision Group
- Instructor at Nvidia's Deep Learning Institute and Berlin's Data Science Retreat
- Preparing a MOOC for OpenHPI (now open for registration)



LinkedIn

How to use our Discord channel during the workshop

- Our channel is **#practical-computer-vision-workshops.** Please ask all questions there instead of the Zoom chat. Through Discord we can have better and more detailed discussions.

- **Step 1 - Use the Discord invite on the Voxel51 website** https://discord.com/invite/fiftyone-community
- **Step 2 - Access channel** (use the direct link below or search) http://bit.ly/3YmvPXG

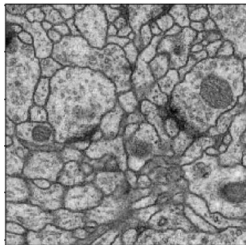
Agenda

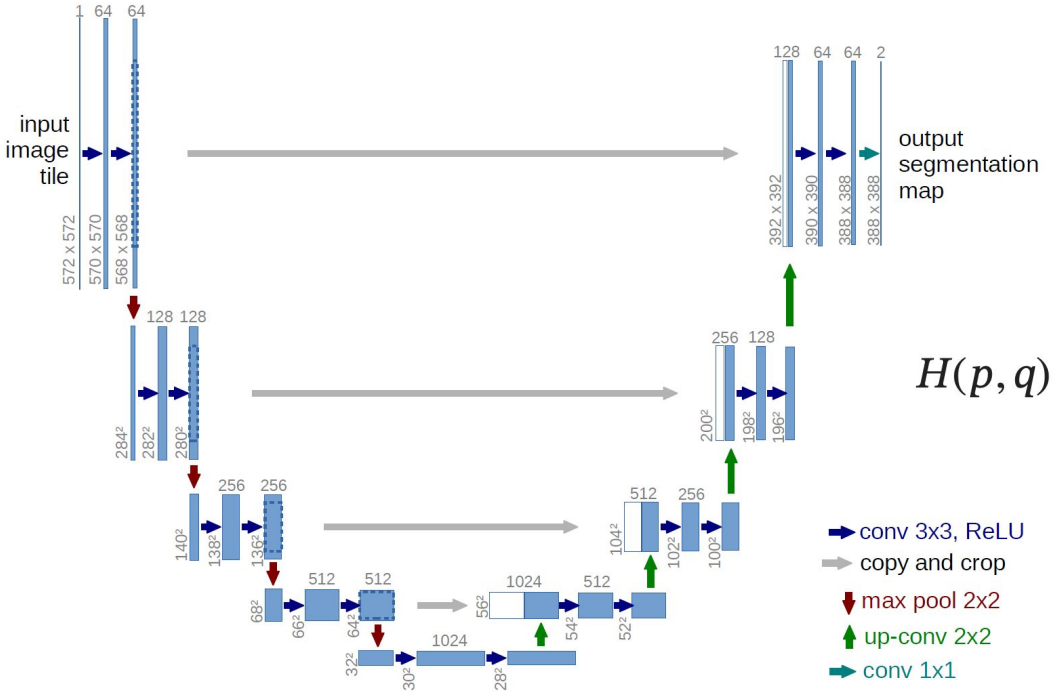
- Fundamentals of convolutions (review)
- Skip connections (review)
- Upsampling and channel mixing with convolutions

Notebooks

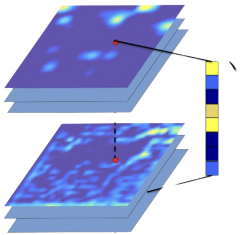
- Semantic segmentation with U-net
  - Google Colab Notebook

Semantic segmentation with U-Net





$$H(p,q) = -\frac{1}{n}\sum_{i=1}^{n} p(i)\log q(i)$$



[U-Net: Convolutional Networks for Biomedical Image Segmentation]

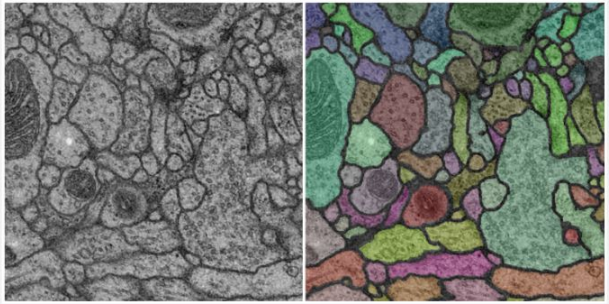
Segmentation of neuronal structures in EM stacks challenge - ISBI 2012

The ISBI 2012 challenge server has been decommissioned, but you can still download the training and test data!

The IEEE International Symposium on Biomedical Imaging (ISBI) is the premier forum for the presentation of technological advances in theoretical and applied biomedical imaging and image computing. In the context of the ISBI 2012 conference which was held in Barcelona, Spain, from 2 to 5 May 2012, we organized a challenge workshop on **segmentation of neuronal structures in EM stacks**.

In this **challenge**, a full stack of EM slices will be used to train machine learning algorithms for the purpose of **automatic segmentation of neural structures**.

The images are representative of actual images in the real-world, containing some noise and small image alignment errors. None of these problems led to any difficulties in the manual labeling of each element in the image stack by an expert human neuroanatomist. The aim of the challenge is to compare and rank the different competing methods based on their pixel and object classification accuracy.



Example of ssTEM image and its corresponding segmentation

Relevant dates

- **Deadline for submitting results**: ~~February 1st~~ March 1st, 2012
- **Notification of the evaluation**: ~~February 21st~~ March 2nd, 2012

Semantic, Instance, and Panoptic Segmentation

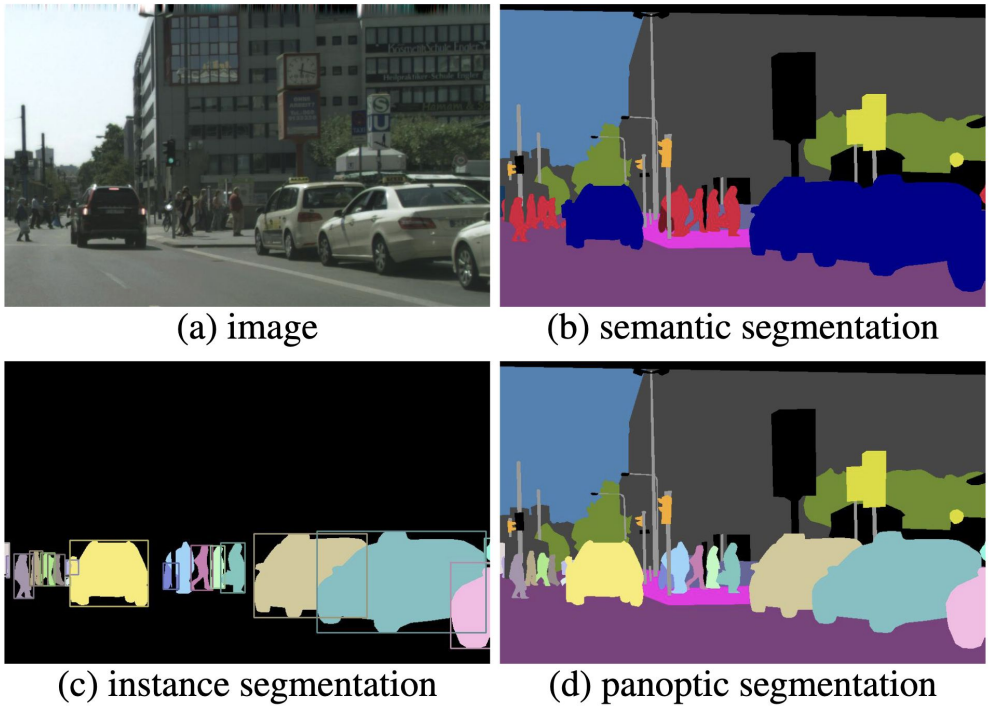
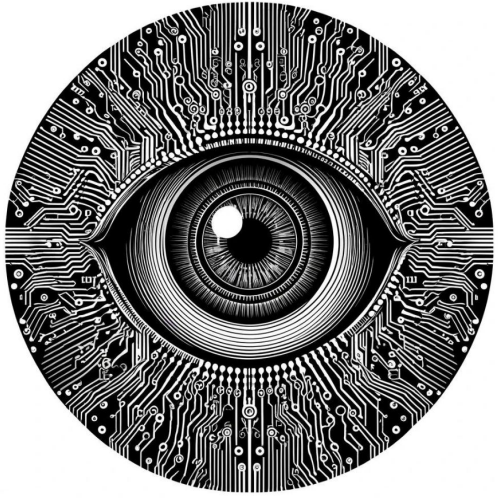


Image from Panoptic Segmentation

Fundamentals of Convolutions



Learning goals

- Understand the role of convolutions in producing image features
- Evaluate effect of stride, padding, and filter size on output images

Convolution filters create new images

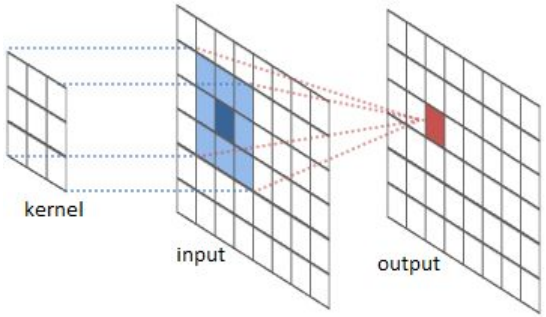
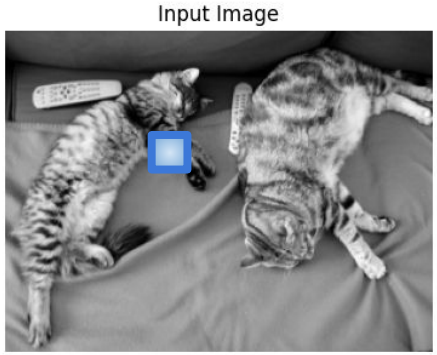
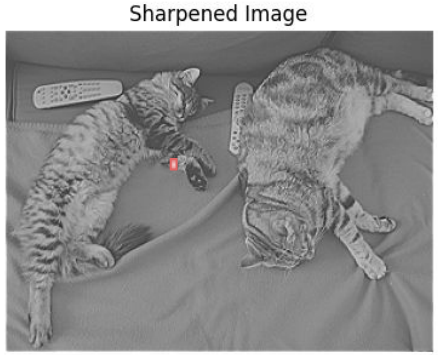


Image from source





$$\text{Sharpening Kernel} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \text{Image Patch (Input)} = \begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{bmatrix}$$

$$\text{Output} = (0 \cdot 10) + (-1 \cdot 20) + (0 \cdot 30) + (-1 \cdot 40) + (5 \cdot 50) + (-1 \cdot 60) + (0 \cdot 70) + (-1 \cdot 80) + (0 \cdot 90)$$

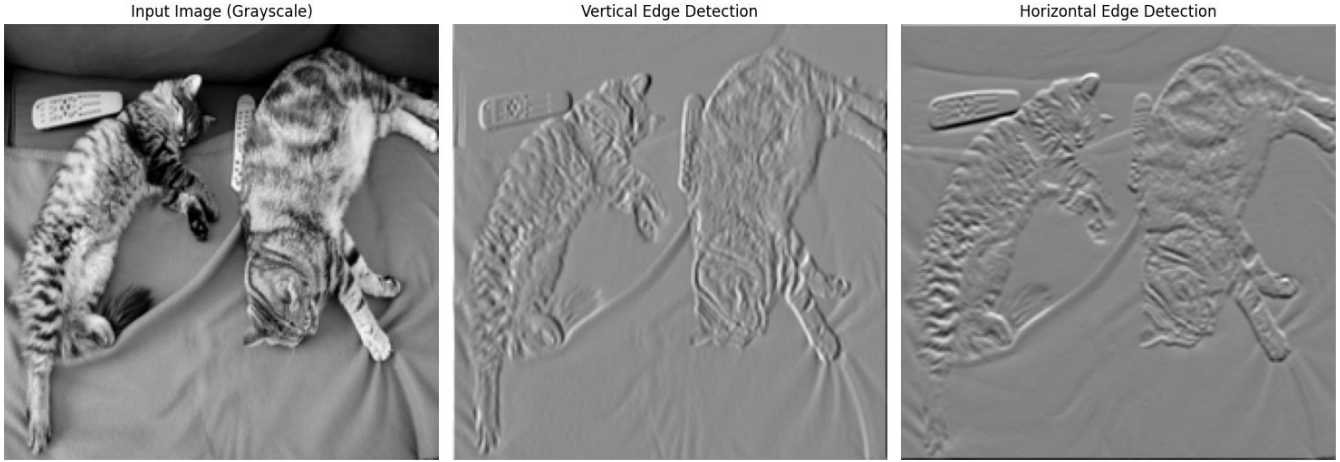
$$\text{Output} = 50$$

Convolutions were ‘traditionally’ handcrafted

$$\begin{bmatrix} 0.111 & 0.222 & 0.111 \\ 0.222 & 0.444 & 0.222 \\ 0.111 & 0.222 & 0.111 \end{bmatrix}$$

Gaussian blur kernel

Handcrafted convolution kernels



Vertical edge detection:

$$K_v = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Horizontal edge detection:

$$K_h = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

The receptive field size controls how many neighboring pixels are considered

Input Image

Blurred Image (3x3)

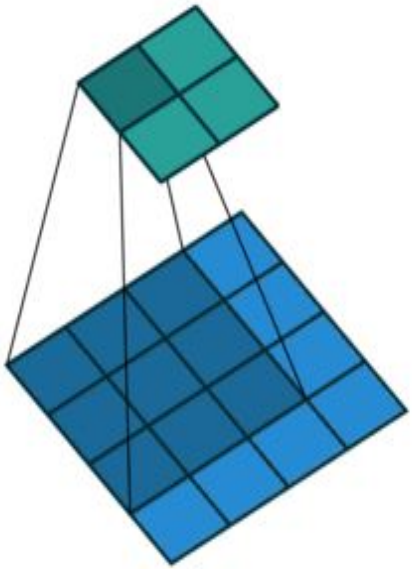
Blurred Image (5x5)

3x3 vs 5x5 uniform blur kernels

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

$$\begin{bmatrix} \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \end{bmatrix}$$

Padding and stride effects



Convolution of 3x3 and stride = 1 without padding

Effect: the output loses one pixel on each dimension

Border padding

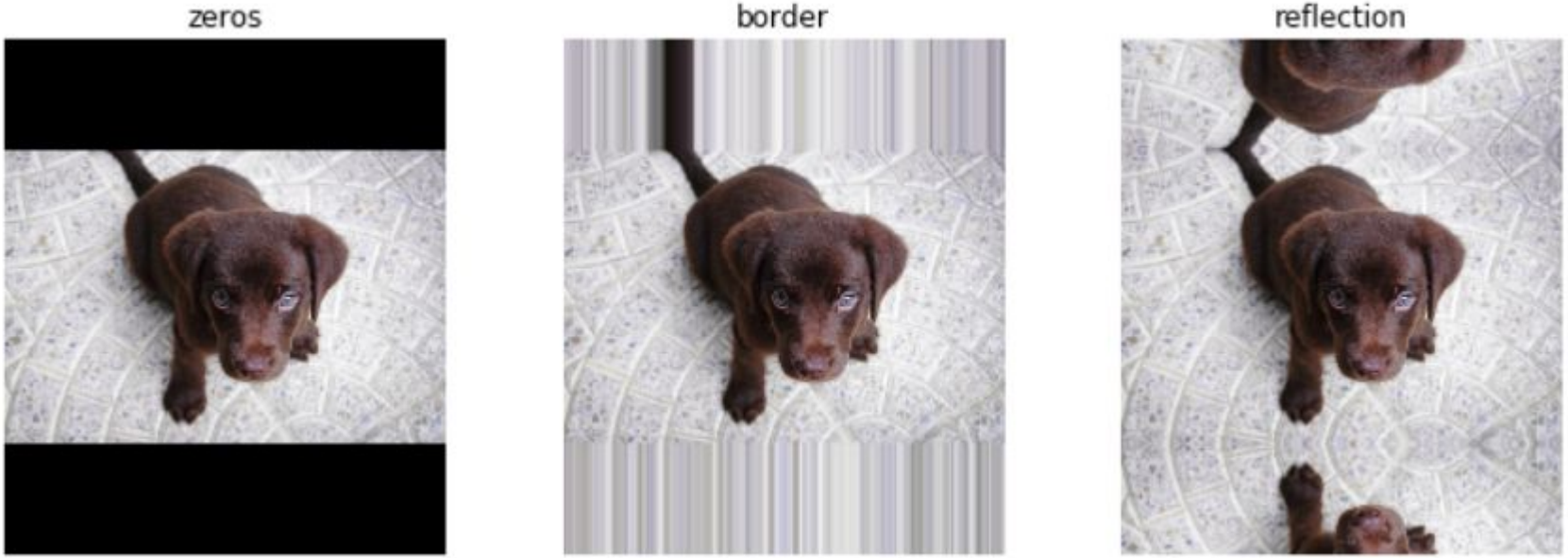
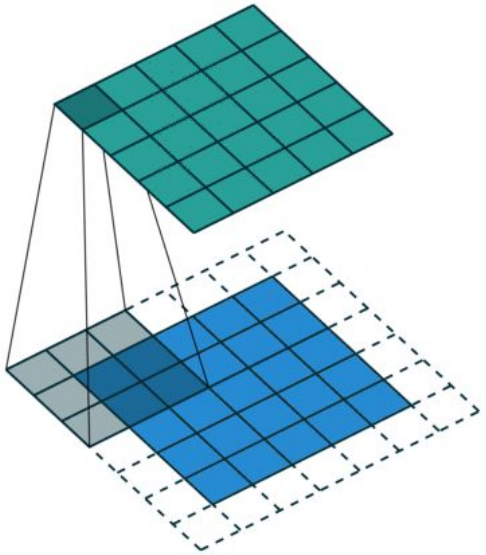


Image from the fastai documentation

“Automatic” feature learning

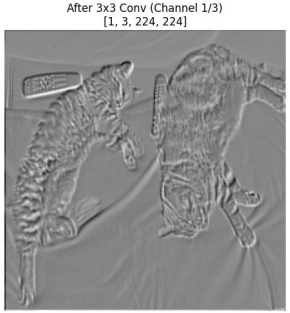


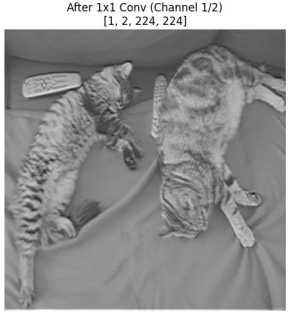
$$\begin{pmatrix} -0.64 & -0.74 & 0.91 \\ -0.96 & 0.48 & -0.46 \\ 0.56 & 0.67 & -0.08 \end{pmatrix}$$

$$\begin{pmatrix} -0.89 & 0.40 & 0.12 \\ -0.45 & 0.38 & 0.67 \\ 0.91 & -0.71 & -0.35 \end{pmatrix}$$

$$\begin{pmatrix} 0.65 & 0.39 & -0.75 \\ 0.36 & -0.57 & 0.34 \\ 0.61 & 0.69 & 0.43 \end{pmatrix}$$

$$\begin{pmatrix} -0.64 & -0.74 & 0.91 \\ -0.96 & 0.48 & -0.46 \\ 0.56 & 0.67 & -0.08 \end{pmatrix}$$





These weights are <u>adjusted to minimize the loss function</u>

$$\mathcal{L} = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(\hat{y}_{ic})$$

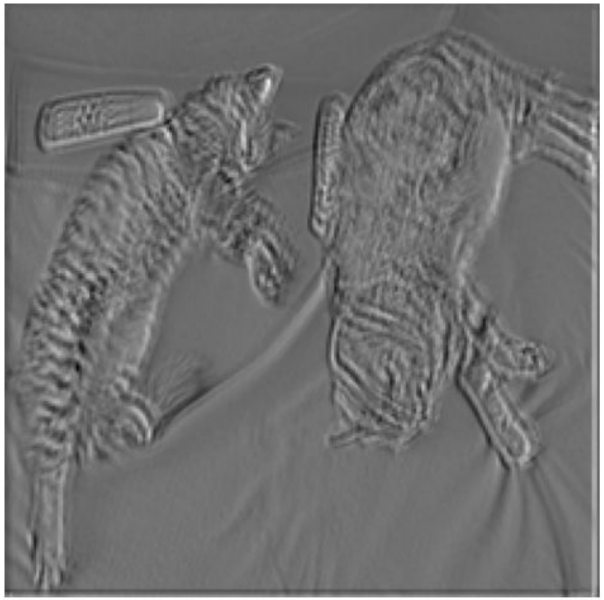
Convolutions of 3x3 and stride = 1 padded with zeros

Effect: the output preserves the original image size while producing a “different-looking” image (based on the values of the weights)

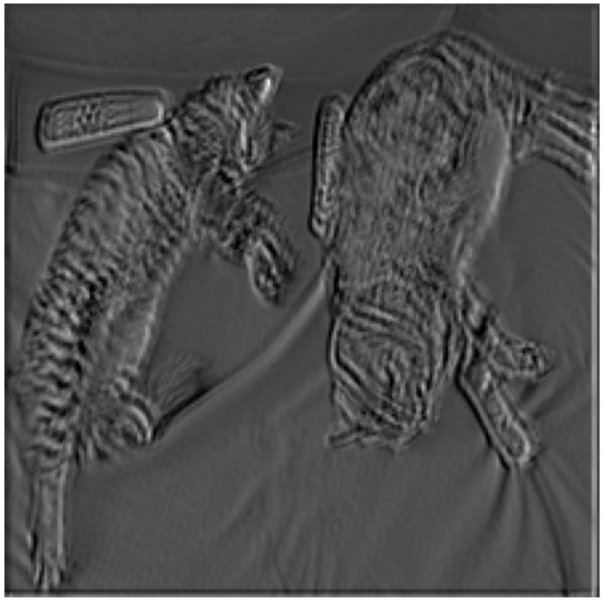
ReLU adds non-linearity to activations

Input with Padding

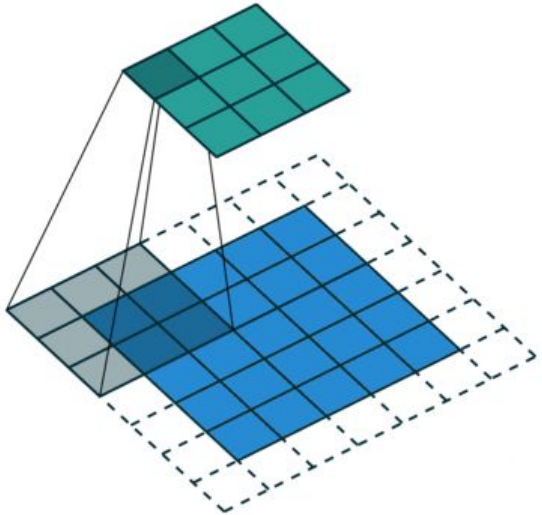
After Conv2d



After ReLU

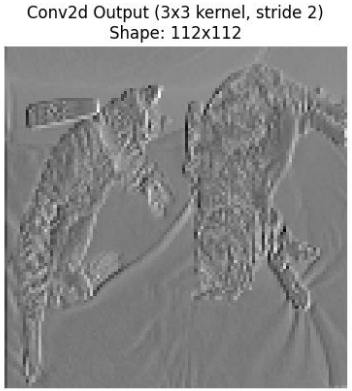


Downsampling images with stride > 1 convolutions



Convolution of 3x3 and stride = 2 padded with zeros

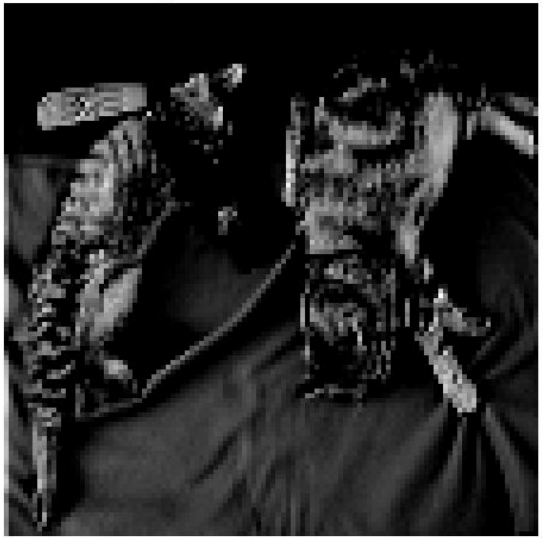
Effect: the output is downsampled to about half its size



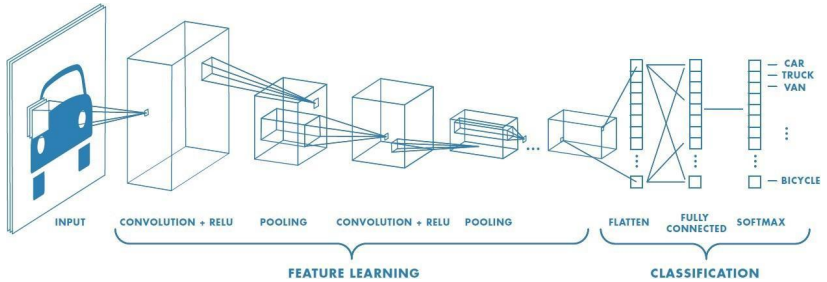
Convolutions are concatenated

Original Image
Shape: torch.Size([224, 224])

After First Conv + ReLU (stride = 2)
Shape: torch.Size([112, 112])

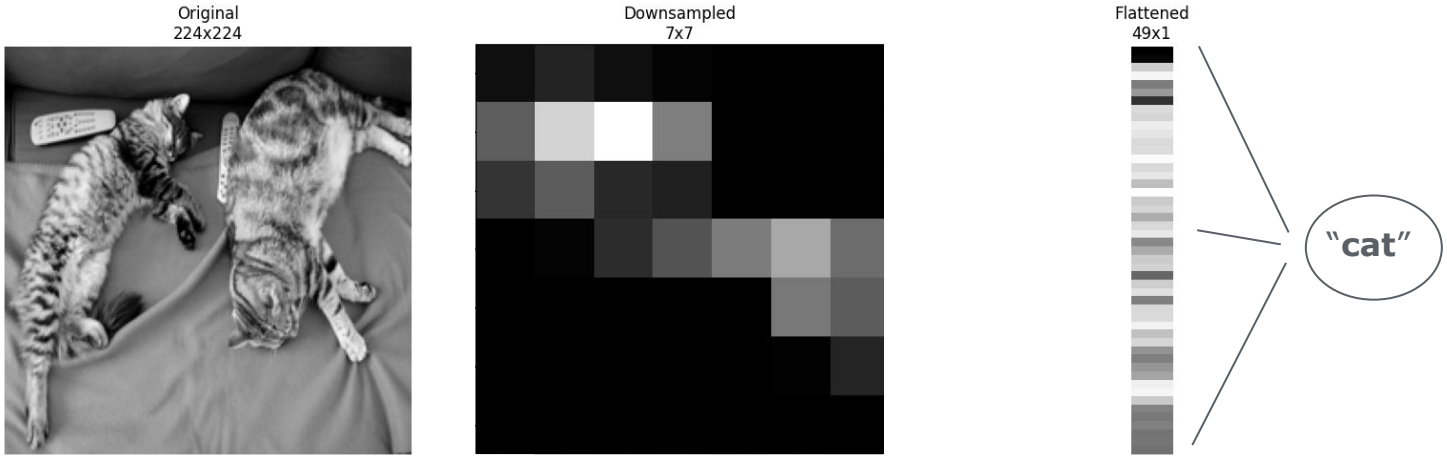


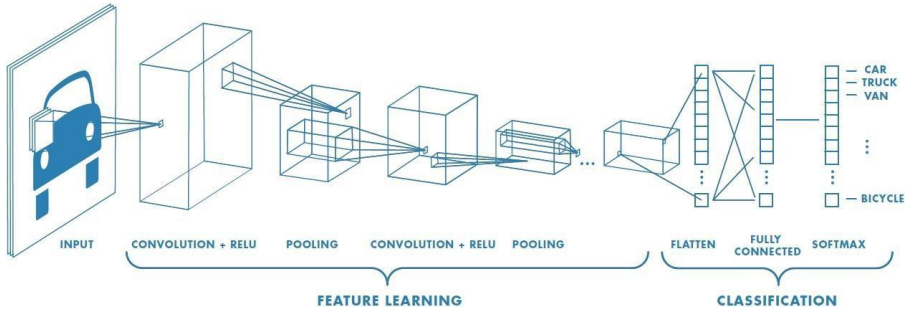
After Second Conv + ReLU (stride = 2)
Shape: torch.Size([56, 56])



$$\mathcal{L} = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(\hat{y}_{ic})$$

We use convolutions as “feature extractors” for classifiers





‘Unrolling’ convolutions

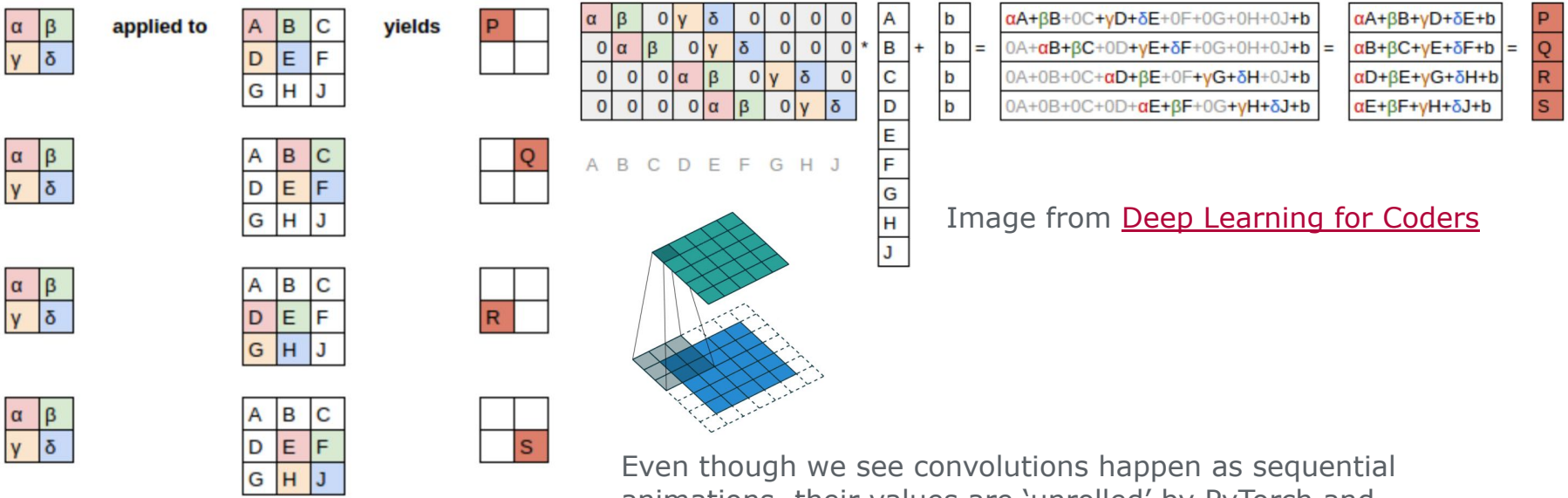


Image from [Deep Learning for Coders]

Even though we see convolutions happen as sequential animations, their values are ‘unrolled’ by PyTorch and applied as parallel matrix multiplications **simultaneously**

Summary

Convolutions and feature learning

- Convolutional layers create new images (“new features”) through learnable weights
- Convolutions were traditionally hand-crafted, now we use the neural network to set their weights
- Classifiers flatten these learned features before feeding them to a feedforward network
- Convolutions are “unrolled” by PyTorch in order to do parallel processing

Effects of receptive field size, padding, and stride

- The size of the receptive field influences how many pixels we consider
- Convolutions with padding preserve spatial dimensions
- Strided convolutions allow us to do learnable downsampling

Further reading and references

A guide to convolution arithmetic for deep learning

- https://arxiv.org/abs/1603.07285

Network in network (1x1 convolutions)

- https://arxiv.org/abs/1312.4400

Hypercolumns for object segmentation and fine-grained localization

- https://openaccess.thecvf.com/content_cvpr_2015/papers/Hariharan_Hypercolumns_for_Object_2015_CVPR_paper.pdf

Summary

Dataset objects wrap images and labels together

- Provide a standard way to access data through `__getitem__` and `__len__` methods

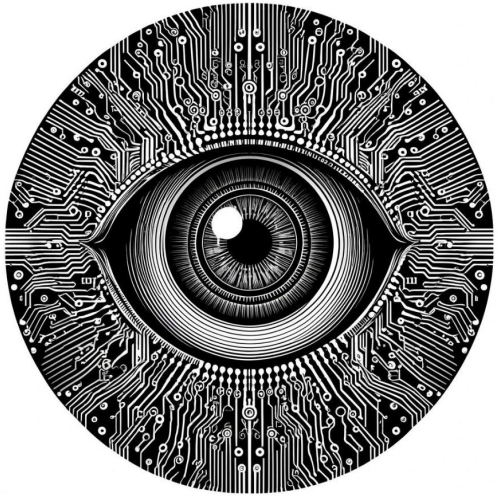
Dataloaders split datasets in batches

- Dataloaders shuffle the training data and maintain sequential order for validation and test sets

Dataloaders are fundamental to implement Stochastic Gradient Descent

- Their random sampling and shuffling of training samples provide the ‘stochastic’ part of Stochastic Gradient Descent

Skip Connections



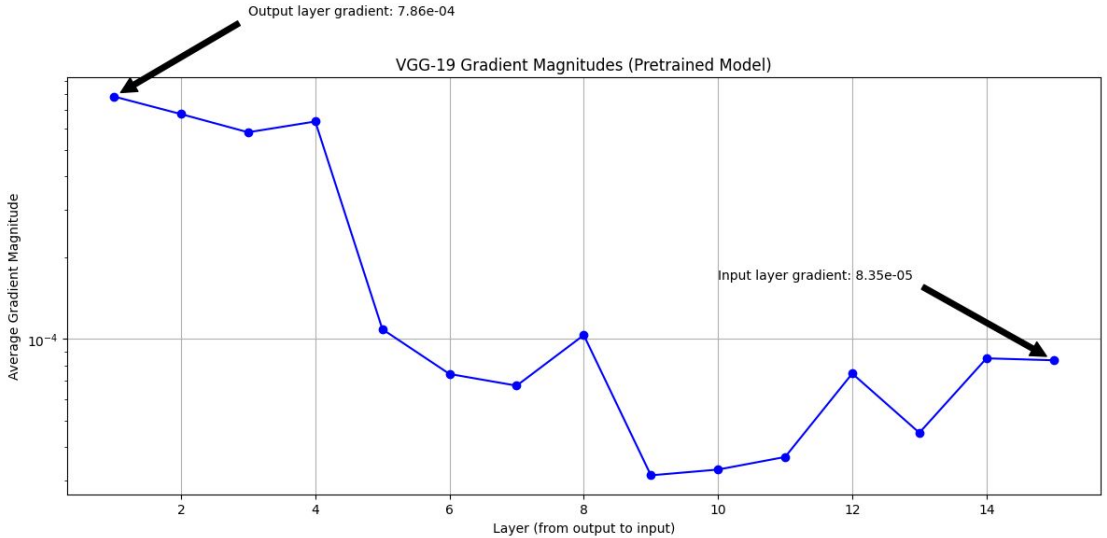
Antonio Rueda-Toicen

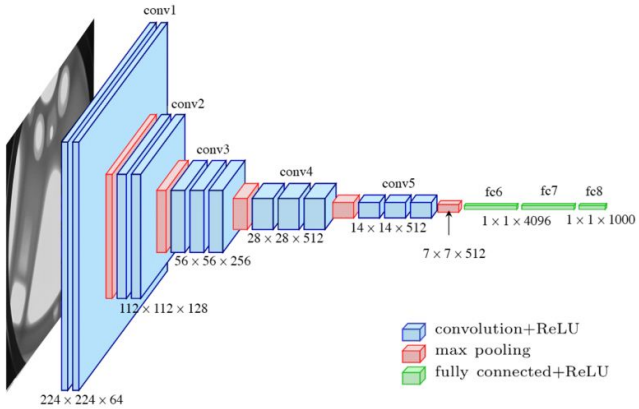
Learning goals

- Understand the vanishing gradient as a numerical problem
- Implement skip connections as element-wise addition or concatenation of activation maps

The vanishing gradient

$$w_{ij} = w_{ij} - (\text{learning rate} * \frac{dL}{dw_{ij}})$$

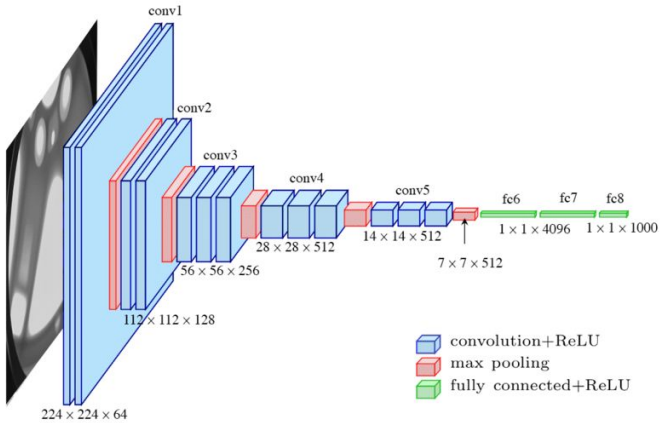




VGG-19 network (source)

Numerical underflow in neural networks

```python
import numpy as np
from scipy.signal import convolve2d

# Example data (any image or 2D array)
image = np.ones((8,8), dtype=np.float32)

# A 3×3 kernel with sum=0.8
kernel = np.array([[0.05, 0.10, 0.05],
                   [0.10, 0.20, 0.10],
                   [0.05, 0.10, 0.05]], dtype=np.float32)

for i in range(1000):
    image = convolve2d(image, kernel, mode='same', boundary='fill', fillvalue=
    # Underflow can show up when values drop below np.finfo(np.float32).tiny
    if (image > 0).sum() == 0:
        print("All values underflowed to 0 at iteration", i)
        break
```



VGG-19 network (source)

Numerical underflow

```python
import numpy as np

a = 1e-8 # Equal to 1 x 10 ** -8
b = 2

print(np.float32(a) ** b)  # Gives a value close to 1e-16
print(np.float16(a) ** b)  # Underflows to 0.0
```

$$(0.00000001)^2 = (1 \times 10^{-8})^2 = 10^{-16}$$

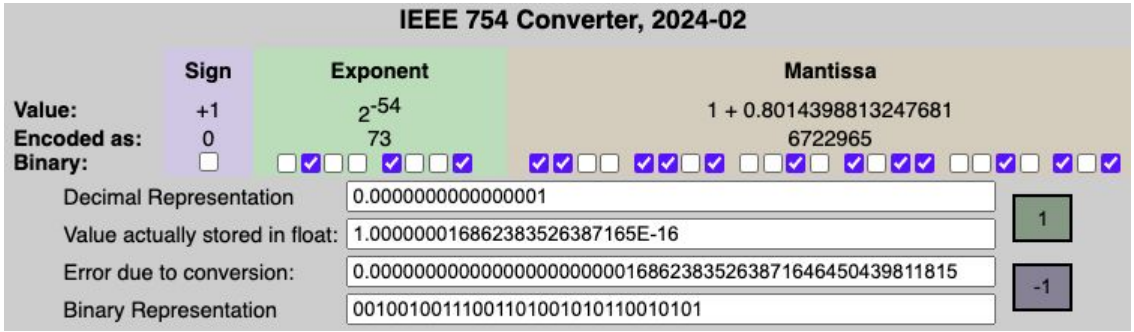
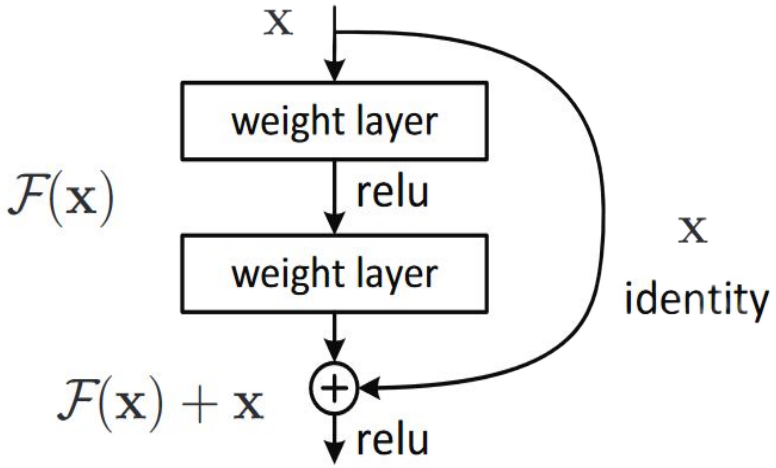


Image from [IEEE-754 Floating Point Converter]

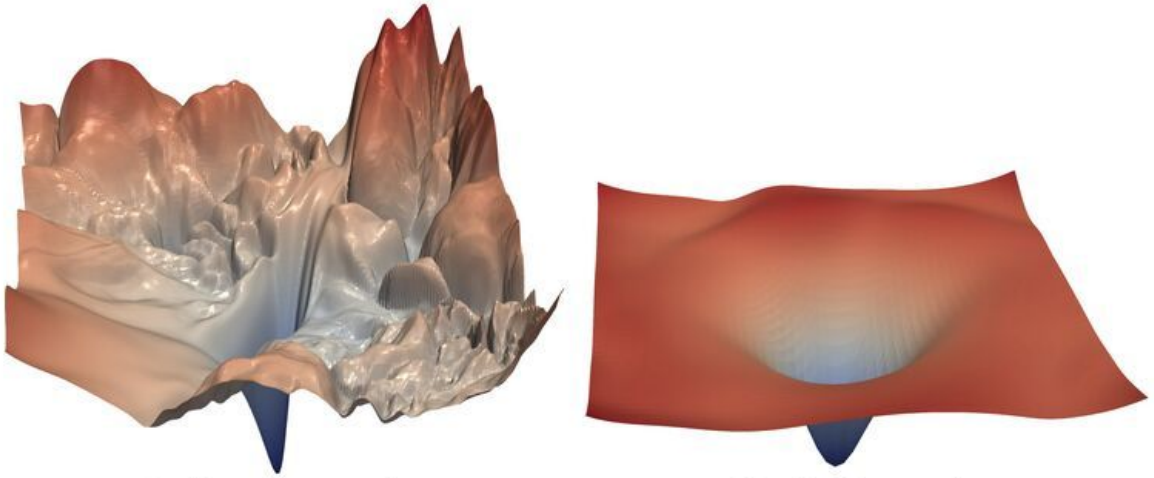
Skip connections on Resnet

```python
import torch.nn as nn

class ResidualBlock(nn.Module):
    def __init__(self, channels):
        super().__init__()
        # Main path - "city route"
        self.conv1 = nn.Conv2d(channels, channels, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(channels)
        self.conv2 = nn.Conv2d(channels, channels, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(channels)
        self.relu = nn.ReLU()

    def forward(self, x):
        # Save input for skip connection - "highway route / checkpoint"
        identity = x

        # Main path through convolutions
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)

        # Add skip connection - "merging highway with city route (adding checkpoint)"
        out += identity

        # Final activation
        out = self.relu(out)

        return out
```



$$\underbrace{\begin{bmatrix} -0.12 & 0.24 \\ 0.35 & -0.25 \end{bmatrix}}_{\text{Identity Path (x)}} + \underbrace{\begin{bmatrix} 0.25 & -0.14 \\ -0.45 & 0.35 \end{bmatrix}}_{\text{Main Path (F(x))}} = \underbrace{\begin{bmatrix} 0.13 & 0.10 \\ -0.10 & 0.10 \end{bmatrix}}_{\text{Output (F(x) + x)}}$$

Effects on the loss landscape



(a) without skip connections (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

Image from Visualizing the Loss Landscape of Neural Nets

Relevance on current architectures

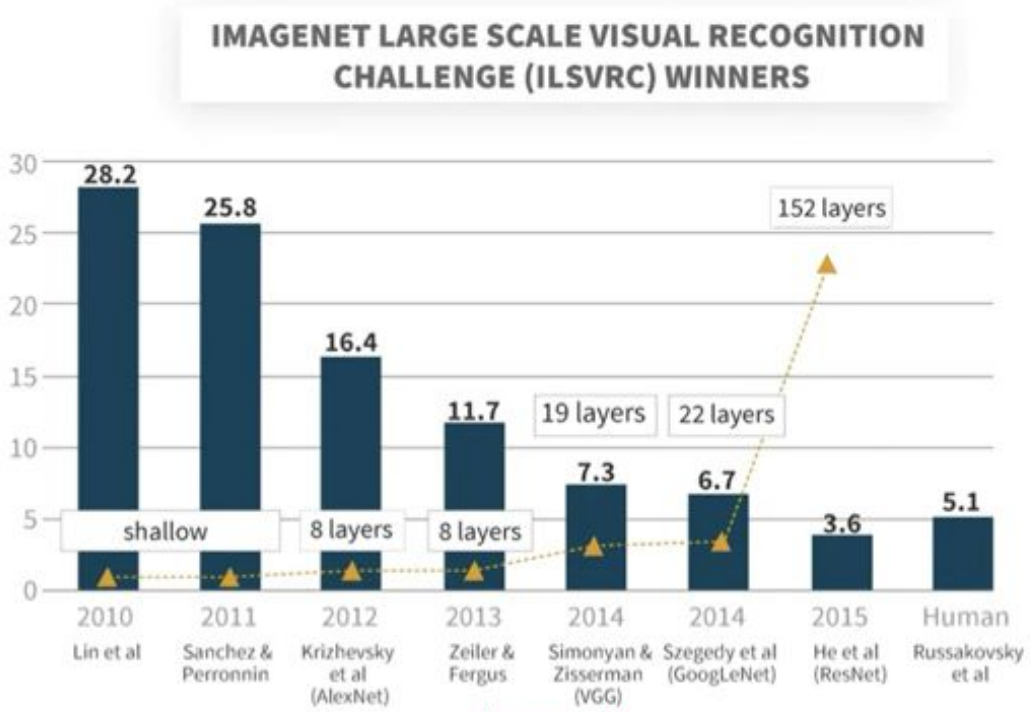
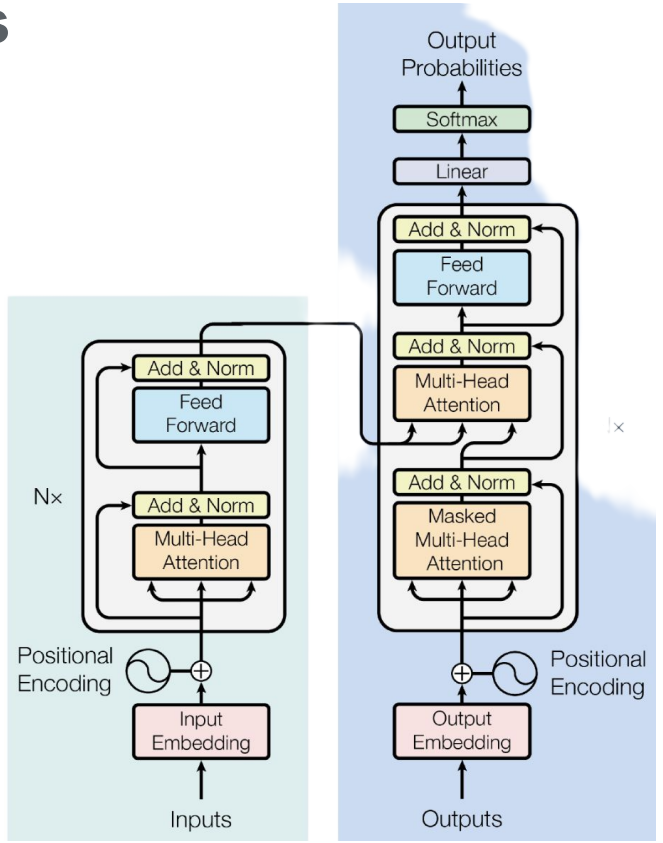


Image source



Transformer architecture from source

Skip connections on Densenet

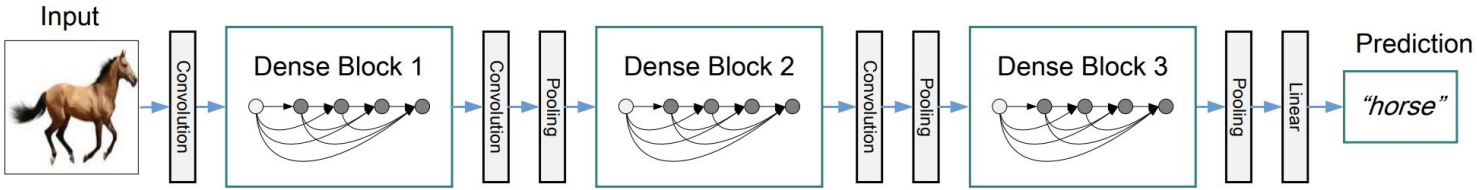
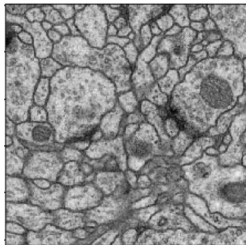
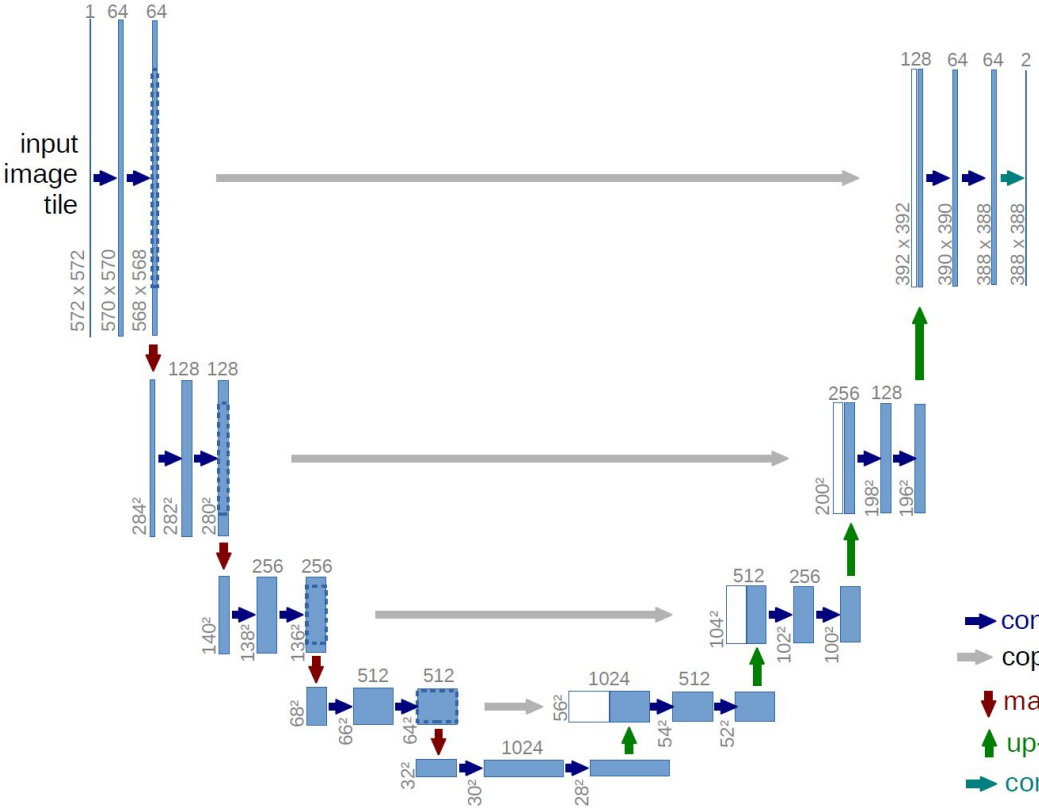


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

```
# Feature maps are concatenated instead of added
# We can control the number of feature maps by using 1x1 convolutions
torch.cat(features, dim=1)
```

Skip connections on U-net





Summary

The vanishing gradient is a numerical problem

- Computers have limited precision to represent small numbers

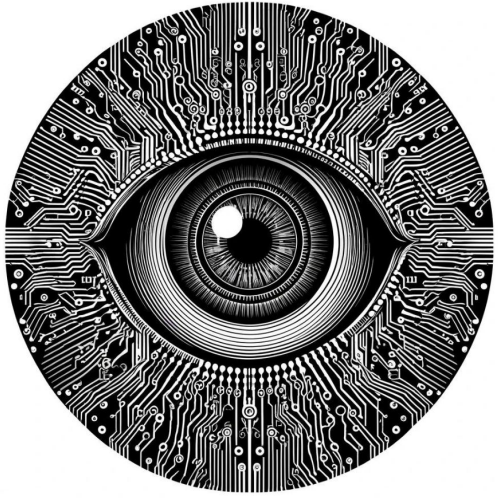
Skip connections serve as “checkpoints” for what the model has learned

- A skip connection gives us the chance to preserve information that could have been destroyed due to numerical underflow
- Skip connections are what allow neural networks to be deep and increase their number of parameters while avoiding vanishing gradients

Two types of skip connections: addition and concatenation

- We use either element wise addition or concatenation of feature maps as skip connections

Upsampling and Channel Mixing with Convolutions

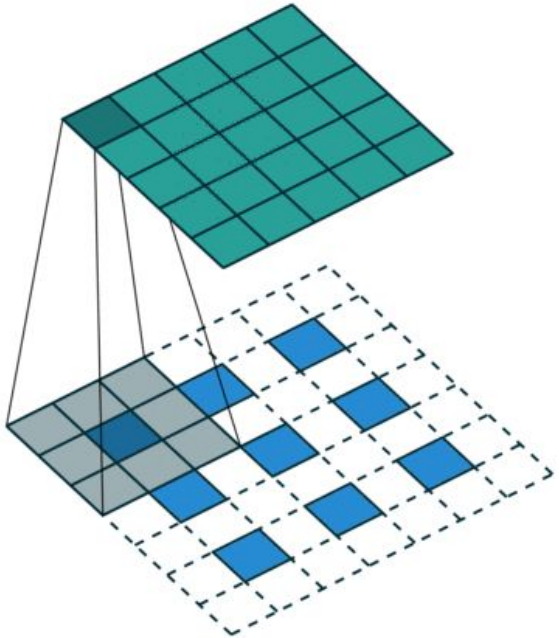


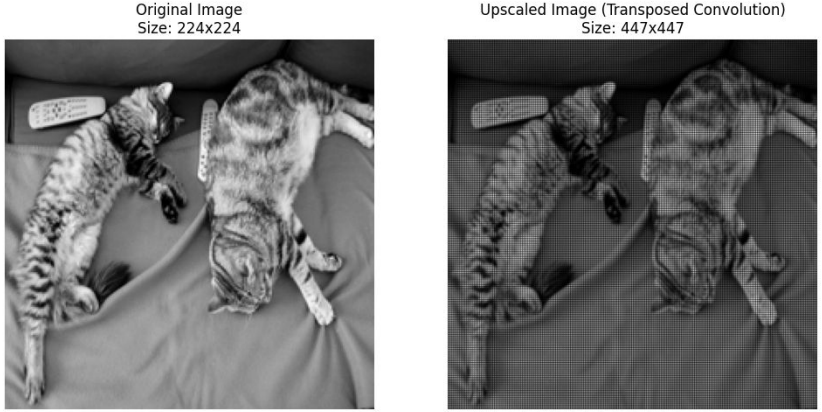
Antonio Rueda-Toicen

Learning goals

- Master upsampling and channel mixing with convolutions
- Use bilinear interpolation to resize images
- Understand hypercolumns and feature combination for image generation

Upsampling filters (aka transposed convolution / upconvolution / atrous convolution / deconvolution)



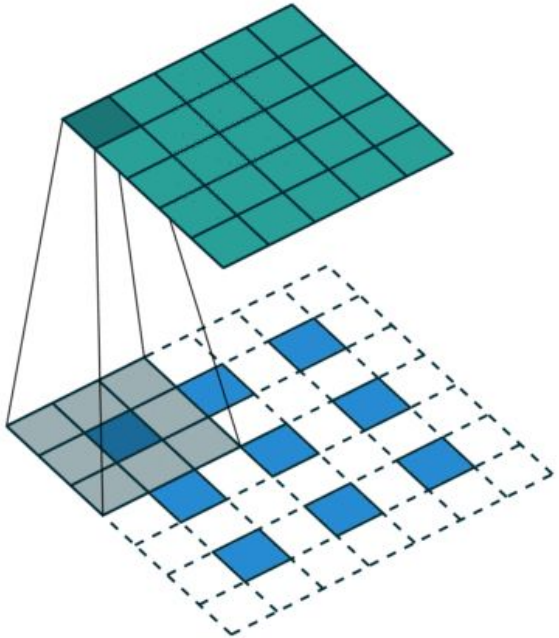


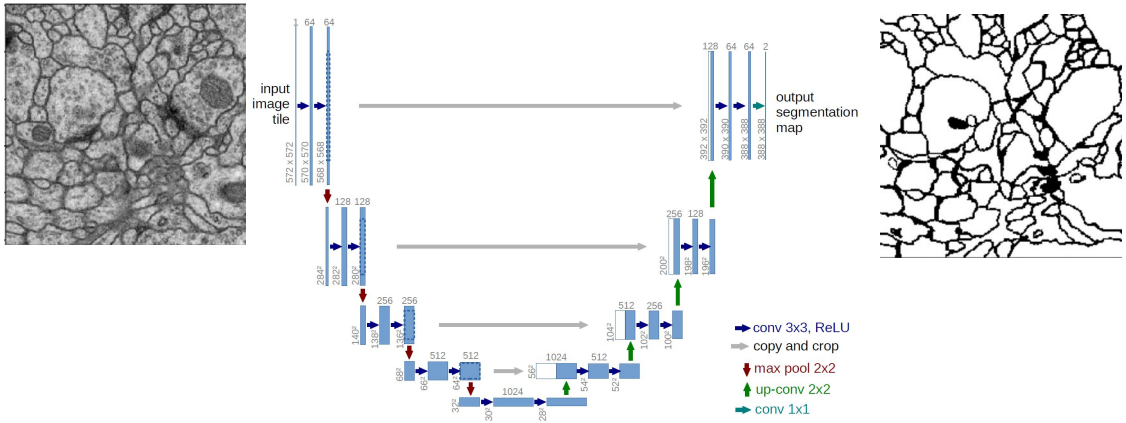
Upconvolution of 3x3 and stride = 1 padded with zeros

Effect: the output is a 5x5 image with ‘synthetic’ pixels

Activity: Conv2D in Pytorch (Colab notebook)

Usage of ‘up-convolution’: decoder paths in image generation models





Upconvolution of 3x3 and stride = 1 padded with zeros

Effect: the output is a 5x5 image with ‘synthetic’ pixels

Activity: Conv2D in Pytorch (Colab notebook)

The checkerboard artifact from “deconvolution”

Deconv in last two layers.
Other layers use resize-convolution.
Artifacts of frequency 2 and 4.

Deconv only in last layer.
Other layers use resize-convolution.
Artifacts of frequency 2.

All layers use resize-convolution.
No artifacts.

Image from [Deconvolution and Checkerboard artifacts]

Original Image
Size: 224x224

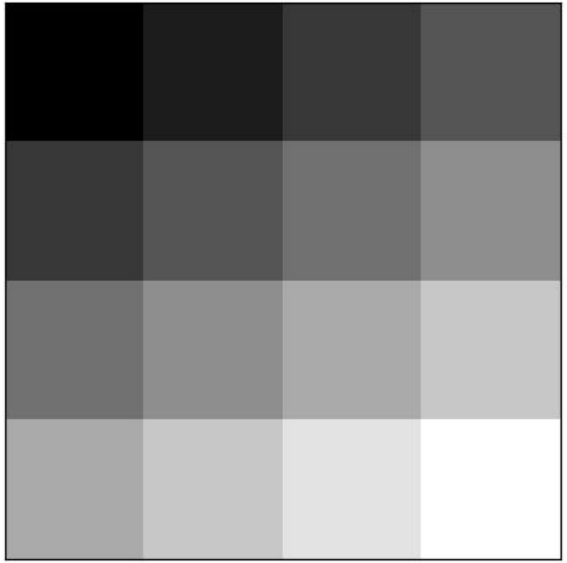
Upscaled Image (Transposed Convolution)
Size: 447x447

Problem: “checkerboard” artifacts

Bilinear interpolation

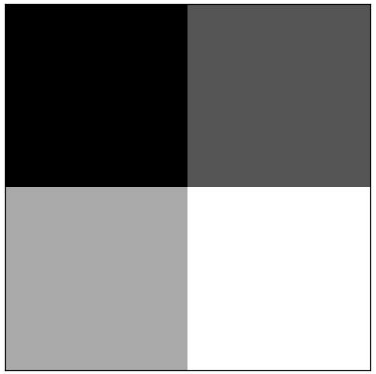
$$w_x = \frac{x - x_1}{x_2 - x_1}$$

output



$$w_y = \frac{y - y_1}{y_2 - y_1}$$

input



$(x_1, y_1) = (0, 0)$ coordinates of Q_{11}

$(x_2, y_2) = (1, 1)$ coordinates of Q_{22}

$(x, y) = (0.6, 0.7)$ target point

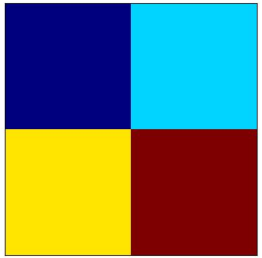
$$Q = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

$$f(x, y) = \begin{bmatrix} 1 - w_x & w_x \end{bmatrix} \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} 1 - w_y \\ w_y \end{bmatrix}$$

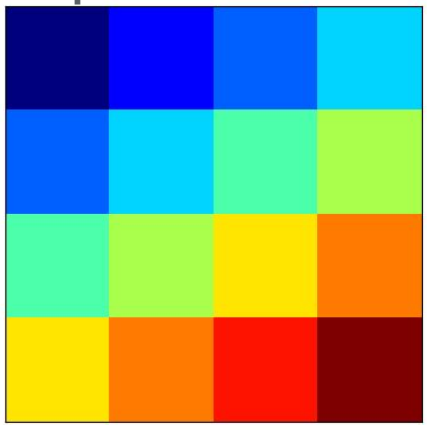
Bilinear interpolation in PyTorch

```
import torch
import torch.nn.functional as F

# Create a 1x1x2x2 tensor (batch_size x channels x height x width)
# Notice that the batch size and channel dimensions are created by wrapping
# the height and width tensor with two pairs of extra square brackets
input = torch.tensor([[[[10, 20],
                        [30, 40]]]],
                     dtype=torch.float32)

# Upscale to 4x4
output = F.interpolate(input, size=(4, 4), mode='bilinear',
align_corners=True)

import matplotlib.pyplot as plt
# The colormap is just for illustration of corner alignment
plt.imshow(input.squeeze(), cmap="jet")
plt.imshow(output.squeeze(), cmap="jet")
```

input (with colormap)
shape: 2x2



output (with colormap)
shape: 4x4



1x1 convolutions mix image channels

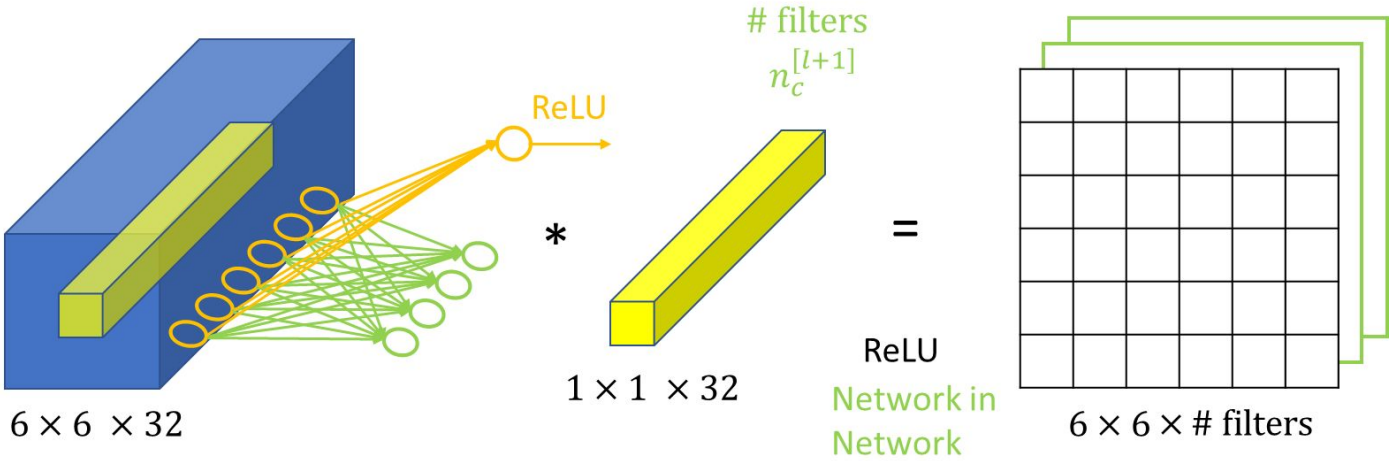


Image from "What does a 1x1 convolution do?" by Andrew Ng

Upsampling and channel mixing with 1x1 convs

Original Image
Size: 224x224

Sample Convolution Outputs (4 of 64)
Size: 224x224

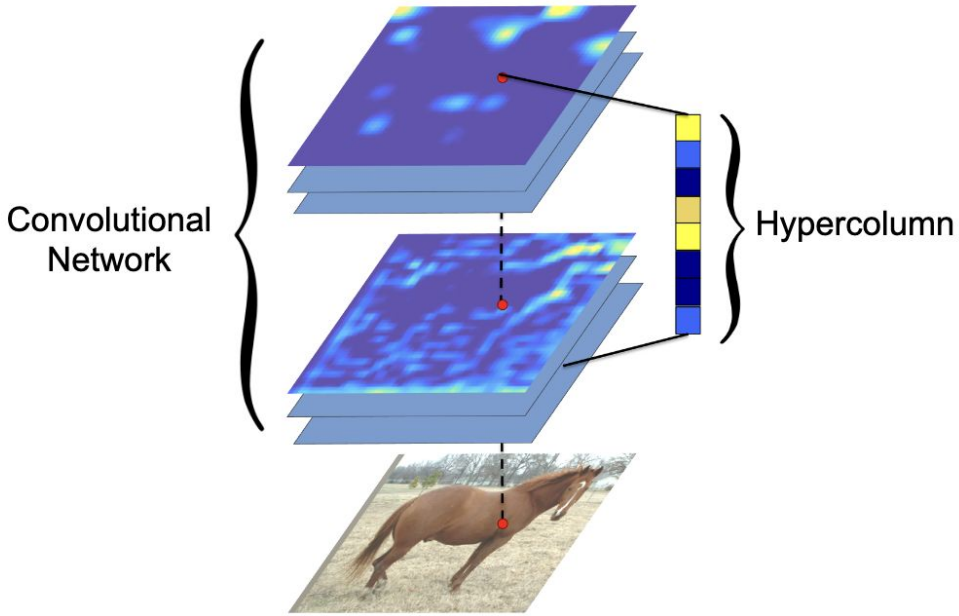
Sample Upsampled Outputs
Size: 448x448

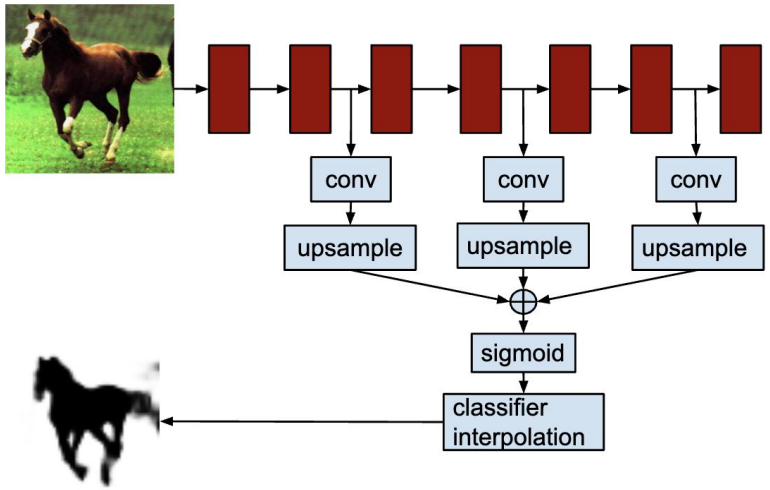
Final Output (After 1x1 Conv)
Size: 448x448

```
layers = nn.Sequential(
    nn.Conv2d(
        in_channels=1,
        out_channels=64,
        kernel_size=3,
        padding=1,
        bias=False
    ),
    nn.Upsample(
        scale_factor=2,
        mode='bilinear',
        align_corners=True
    ),
    nn.Conv2d(
        in_channels=64,
        out_channels=1,
        kernel_size=1,
        bias=False
    )
)
```

Note: no checkerboard artifacts

Combining “hypercolumns” is a common use of 1x1 convolutions





Images from “Hypercolumns for Object Segmentation and Fine-grained Localization”

Summary

1x1 convolutions mix channel information

- Mixing channels allows us to create new images by combining learned features

Transposed convolutions enable upsampling (with checkerboard artifacts)

- Bilinear interpolation and channel mixing is a better alternative

Further reading and references

A guide to convolution arithmetic for deep learning

- https://arxiv.org/abs/1603.07285

Network in network (1x1 convolutions)

- https://arxiv.org/abs/1312.4400

Hypercolumns for object segmentation and fine-grained localization

- https://openaccess.thecvf.com/content_cvpr_2015/papers/Hariharan_Hypercolumns_for_Object_2015_CVPR_paper.pdf

Resources

- Github Repository
- YouTube playlist
- Discord channel
  #practical-computer-vision-workshops