

# Advanced Workshop on Machine Learning

## Lecture 2: Convolutional Neural Network

# Today's Agenda

## Part 1

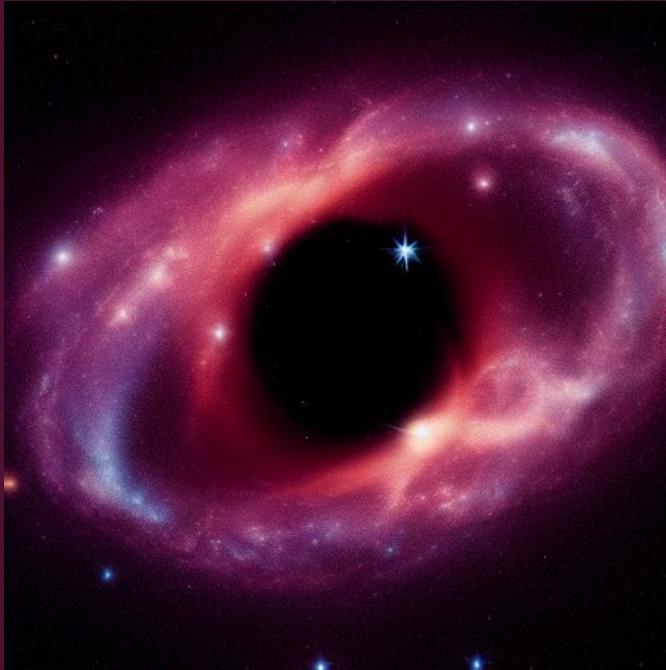
- Convolutional Layer
- Pooling Layer
- Edge Detection

## Part 2

- ImageNet Models
- COCO Models
- Transfer Learning
- Inference as an API

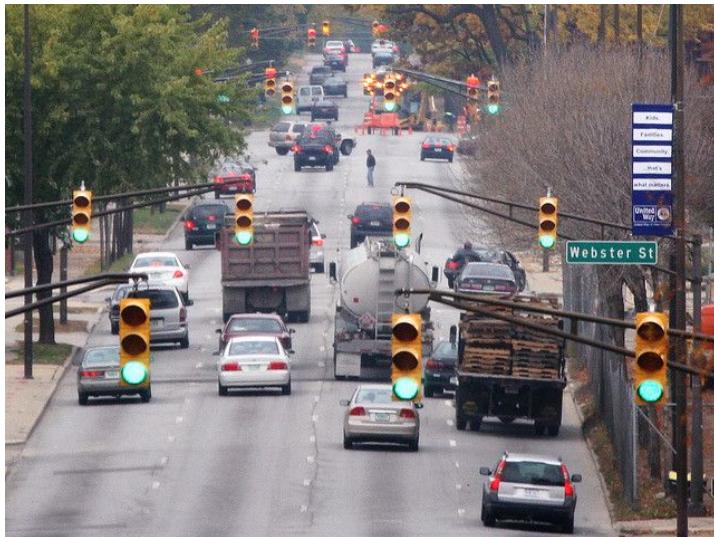
# Part 1

## Convolutions



(Image by a Stable Diffusion model)

# Image Classification Task



Model Input



Class	Probability
Vehicle	100%
Banana	0%
Sheep	0%

Model Output

# Dense Layer for Image Classification?

## #1: Patch Independence

Detecting an object in one region of the image does not require any perception of other regions.

## #2: Parameter Duplication

Detecting an object in one region of the image is the same task as in any other region.



Source: Flickr / License: CC BY 4.0

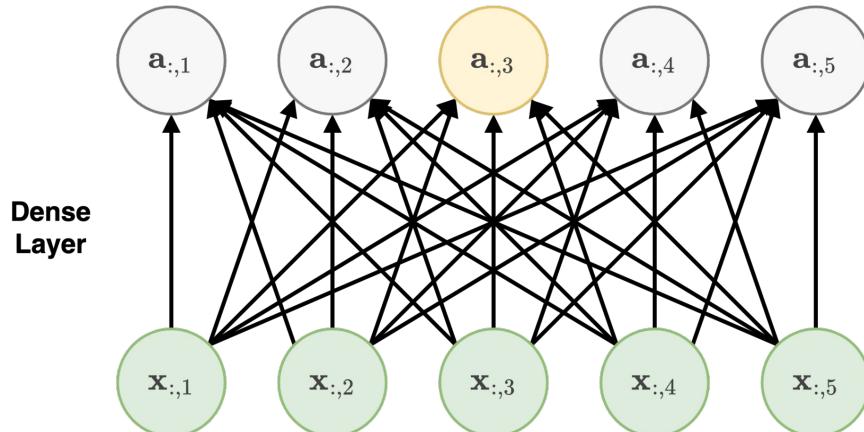
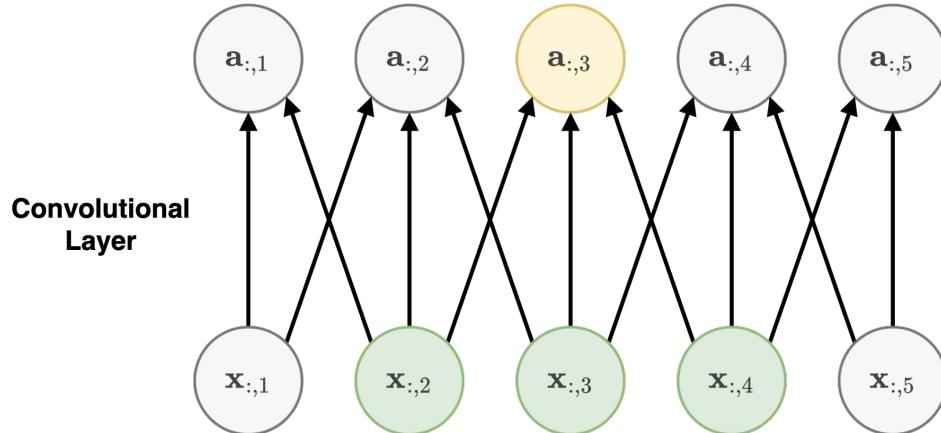
# Sparse Connectivity

## #1: Patch Independence

Detecting an object in one region of the image does not require any perception of other regions.

## Solution

Connect each output node only with a subset of the input.



# Indirect Receptive Field

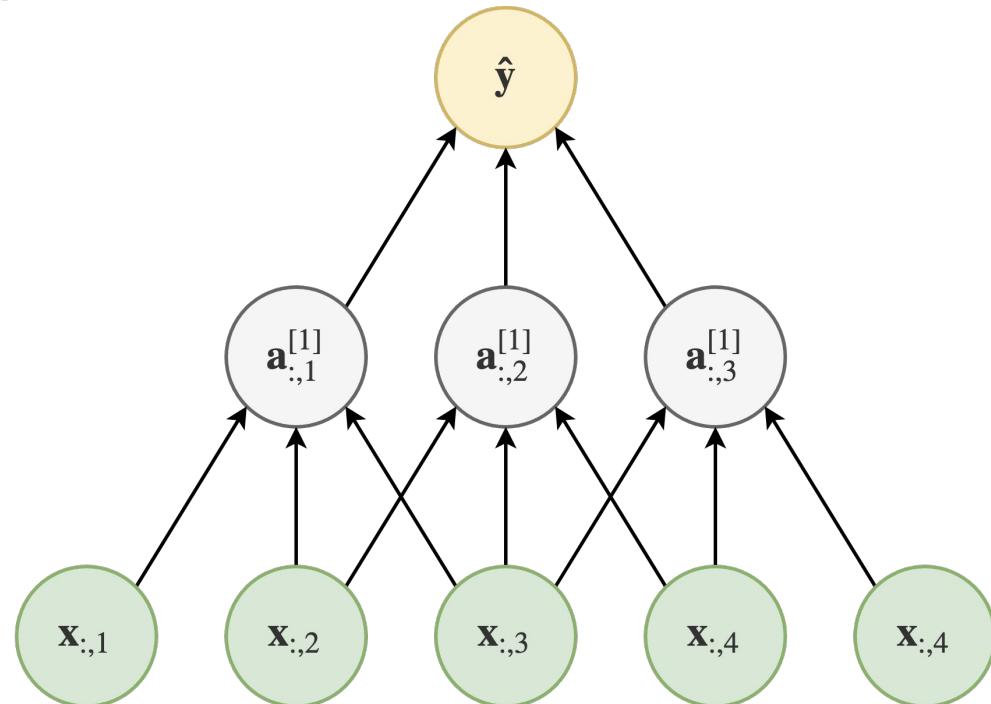
## Question

Can a network with sparse weights see large objects?

## Answer

Indirect **receptive field** is **exponentially increasing** with layer depth.

Deep enough networks have output layer connected to every unit of the input layer.



# Parameter Sharing

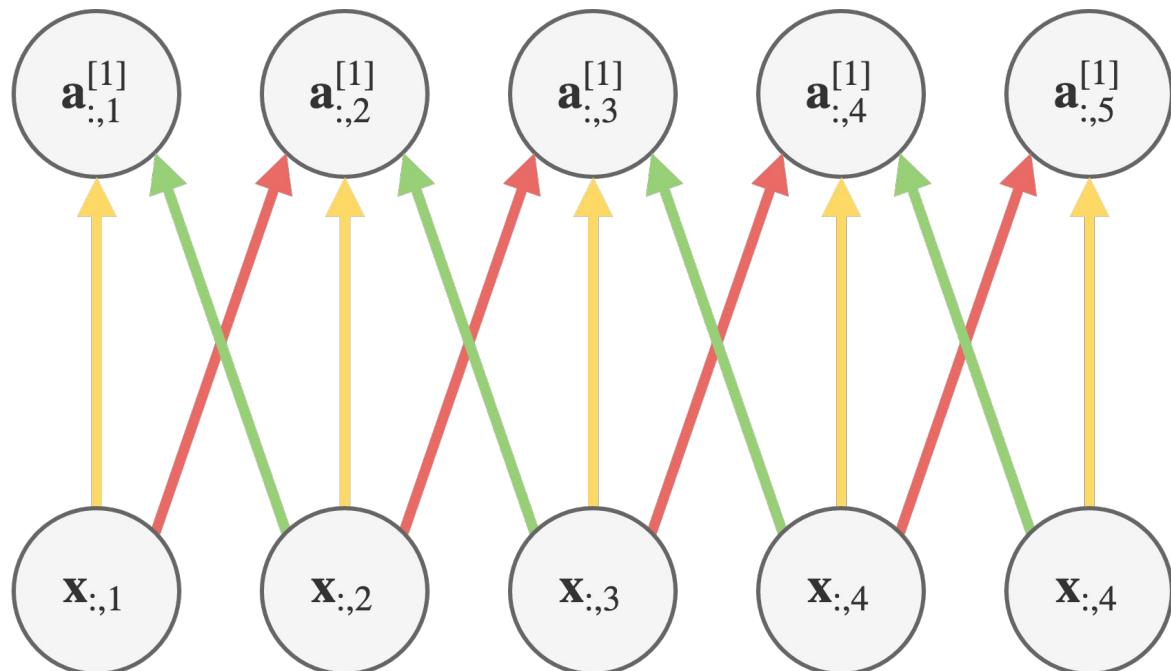
## #2: Parameter Duplication

Detecting an object in one region of the image is the same task as in any other region.

## Solution

Parameters are **shared within layer**.

Equivariance to linear translation of input.



# Convolution

Input: 3x3

1	2	3
4	5	6
7	8	9

Filter: 2x2

10	20
30	40



Output: 2x2

370	



Dot  
Product

$$1 \times 10 + 2 \times 20 + 4 \times 30 + 5 \times 40 = 370$$

# Convolution

Input: 3x3

1	2	3
4	5	6
7	8	9

Filter: 2x2

10	20
30	40



Output: 2x2

370	470



Dot  
Product

$$2 \times 10 + 3 \times 20 + 5 \times 30 + 6 \times 40 = 470$$

# Convolution

Input: 3x3

1	2	3
4	5	6
7	8	9

Filter: 2x2

10	20
30	40



Output: 2x2

370	470
670	

Dot  
Product

$$4 \times 10 + 5 \times 20 + 7 \times 30 + 8 \times 40 = 670$$

# Convolution

Input: 3x3

1	2	3
4	5	6
7	8	9



Filter: 2x2

10	20
30	40

Dot  
Product

Output: 2x2

370	470
670	770

$$5 \times 10 + 6 \times 20 + 8 \times 30 + 9 \times 40 = 770$$

# Convolution

Input: 3x3 + 1x1 padding

1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0



Filter: 2x2

10	20
30	40

Dot  
Product

Output: 3x3

370	470	210
670	770	

$$3 \times 10 + 0 \times 20 + 6 \times 30 + 0 \times 40 = 210$$

# Convolution

Input: 3x3 + 1x1 padding

1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0



Filter: 2x2

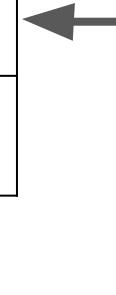
10	20
30	40

Dot  
Product

Output: 3x3

370	470	210
670	770	330

$$6 \times 10 + 0 \times 20 + 9 \times 30 + 0 \times 40 = 330$$



# Convolution

Input: 3x3 + 1x1 padding

1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0



Filter: 2x2

10	20
30	40

Dot  
Product

Output: 3x3

370	470	210
670	770	330
		90

$$9 \times 10 + 0 \times 20 + 0 \times 30 + 0 \times 40 = 90$$

# Convolution

Input: 3x3 + 1x1 padding

1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0

Filter: 2x2

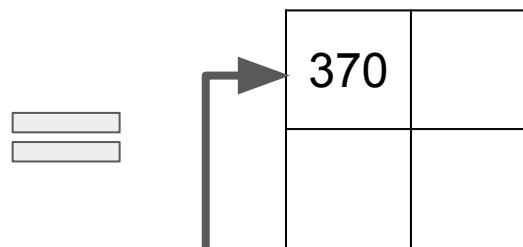


10	20
30	40

Stride: 2x2

Dot  
Product

Output: 2x2



$$1 \times 10 + 2 \times 20 + 4 \times 30 + 5 \times 40 = 370$$

# Convolution

Input: 3x3 + 1x1 padding

1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0

Filter: 2x2

10	20
30	40



Stride: 2x2

Dot  
Product

Output: 2x2

370	210



$$3 \times 10 + 0 \times 20 + 6 \times 30 + 0 \times 40 = 210$$



# Convolution

Input: 3x3 + 1x1 padding

1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0

Filter: 2x2

10	20
30	40



Stride: 2x2

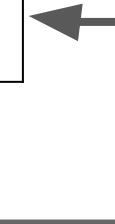
Dot  
Product

Output: 2x2

370	210
	90



$$9 \times 10 + 0 \times 20 + 0 \times 30 + 0 \times 40 = 90$$



# Convolution

Input: 3x3 + 1x1 padding

1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0

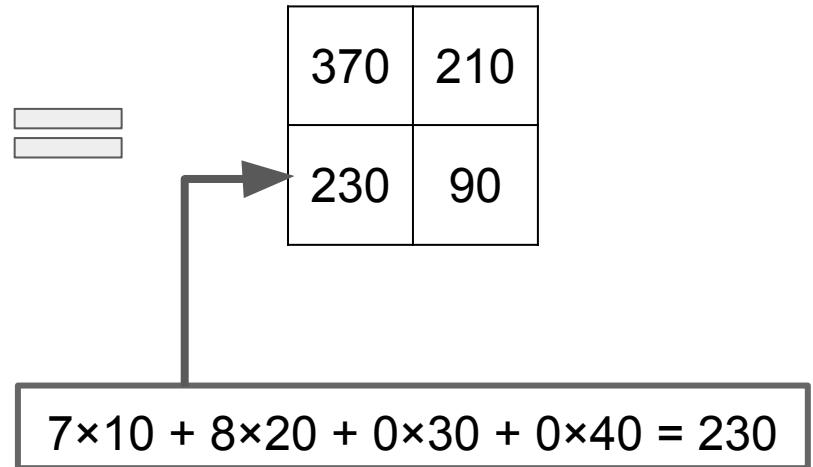
Filter: 2x2

$\times$

10	20
30	40

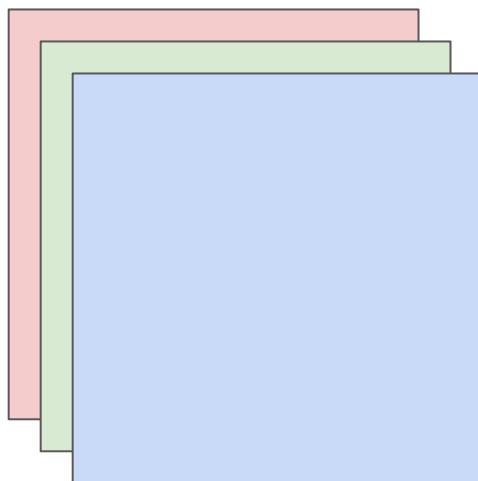
Stride: 2x2

Output: 2x2

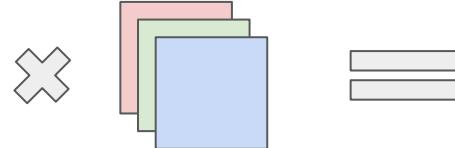


# Convolution (Depth: Many in, 1 out)

$$n_h^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$$



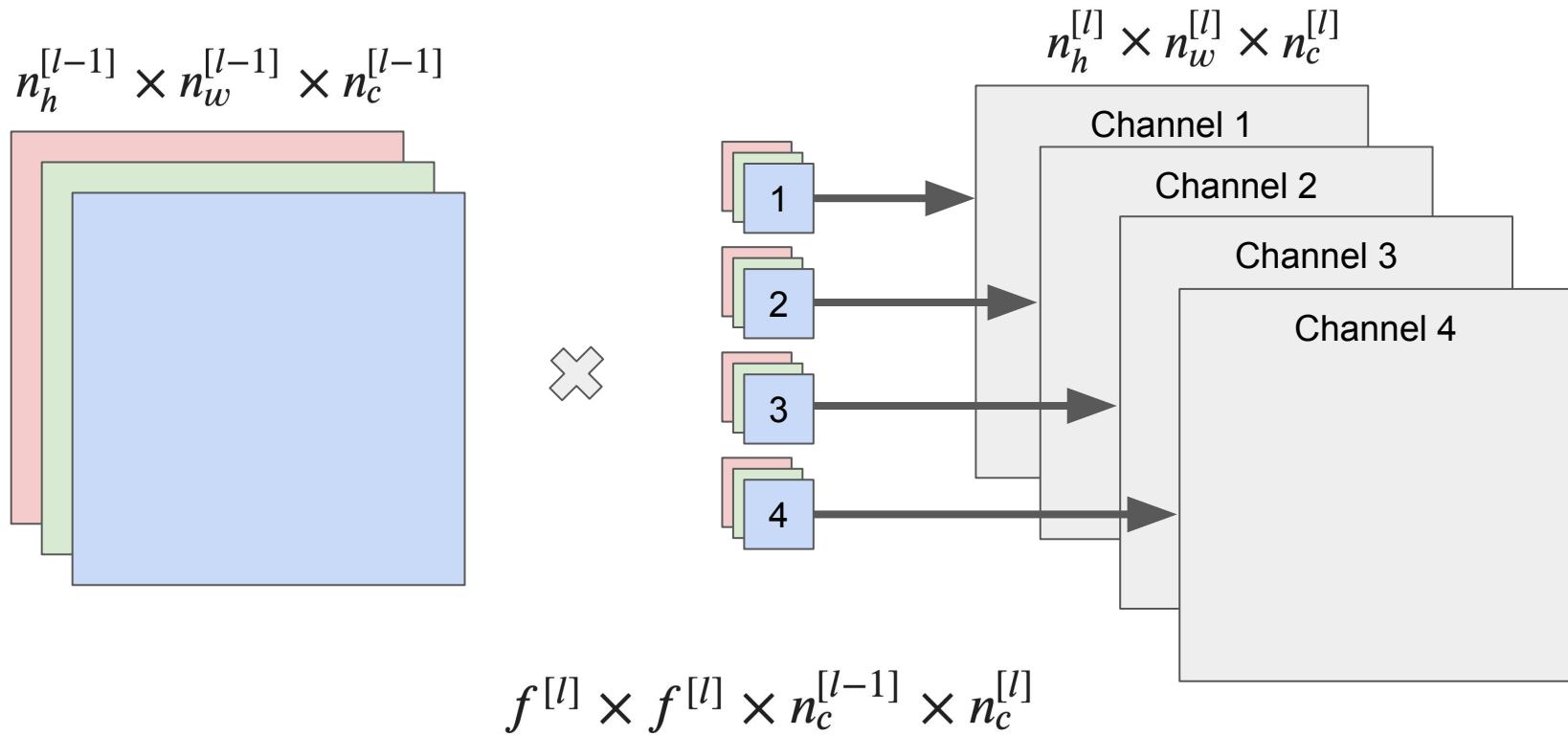
$$n_h^{[l]} \times n_w^{[l]}$$



$$f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$$



# Convolution (Depth: Many in, many out)



# Hierarchy of Visual Information

1. Pixels (RGB)
2. Edges
3. Curves
4. Shapes: circles, boxes, triangles, ...
5. Textures: checkers, gradient, monotonic, ...
6. Parts: wheels, road, windows, ...
7. Objects: cars, cats, dogs, oranges, ...
8. Scenes: crowd of people, fruit garden, ...

# Case Study: Edge Detection

## Image

Artistic Rendering of Mars  
Exploration Rover  
(Spirit and Opportunity)

## Dimensions

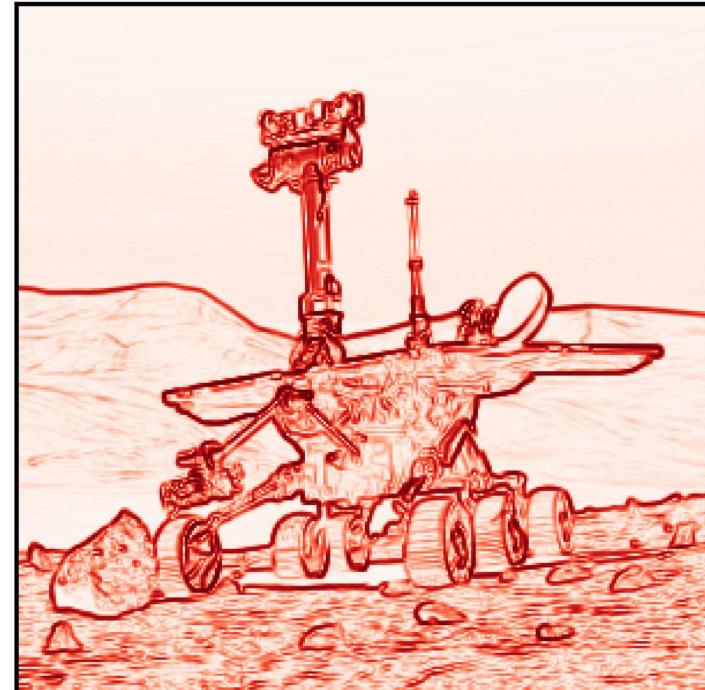
300 x 300 x 3 channels

## Task

Detect all edges at various  
angles: horizontal, vertical,  
tilted, etc.



# What is edge detection?

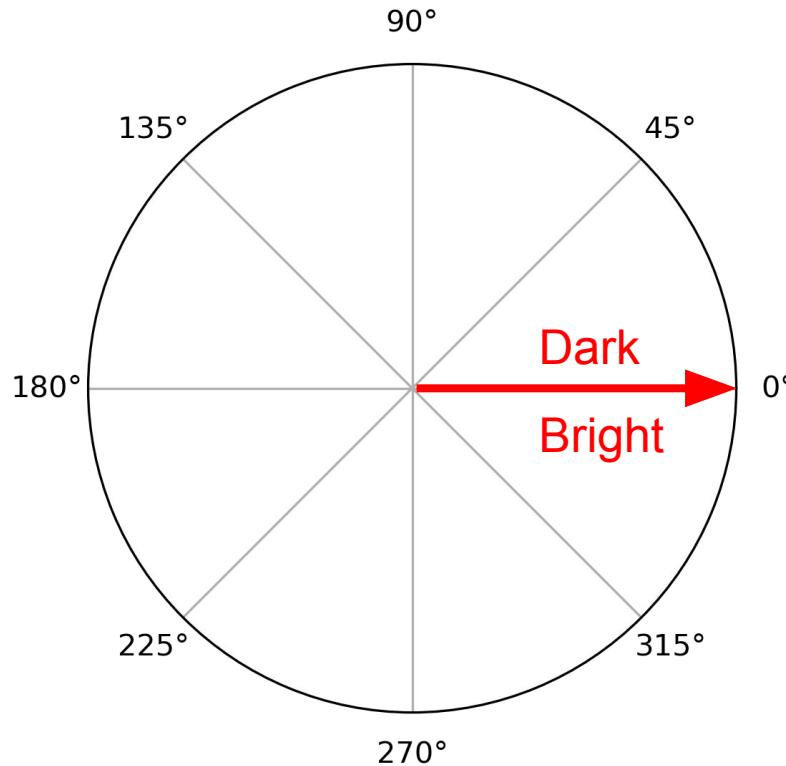


# Edge Detection

Every possible edge from 0 to 360 with 45 degree step.

- Dark side on LHS
- Bright side on RHS

Brightness averaged across RGB channels.

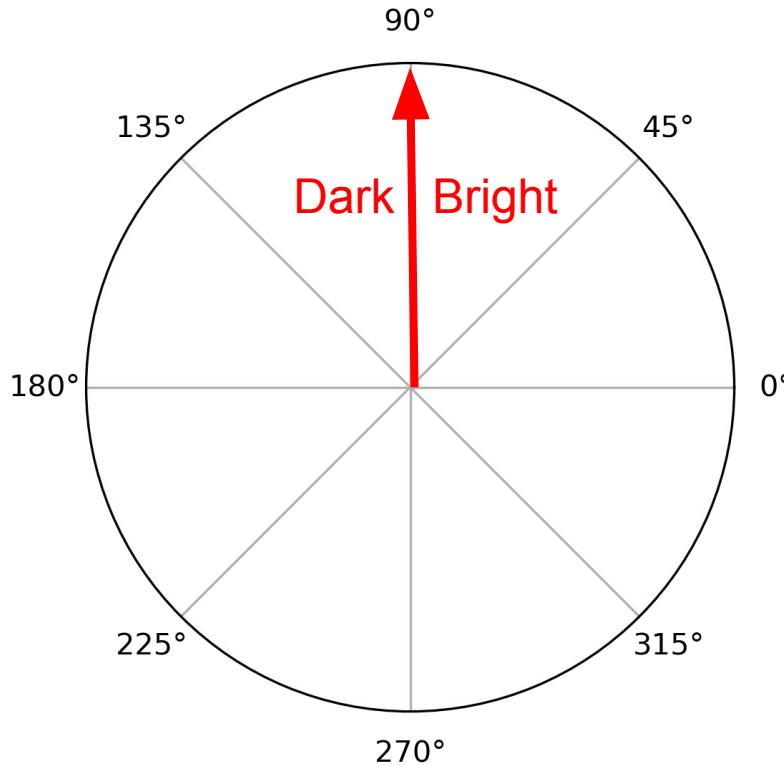


# Edge Detection

Every possible edge from 0 to 360 with 45 degree step.

- Dark side on LHS
- Bright side on RHS

Brightness averaged across RGB channels.

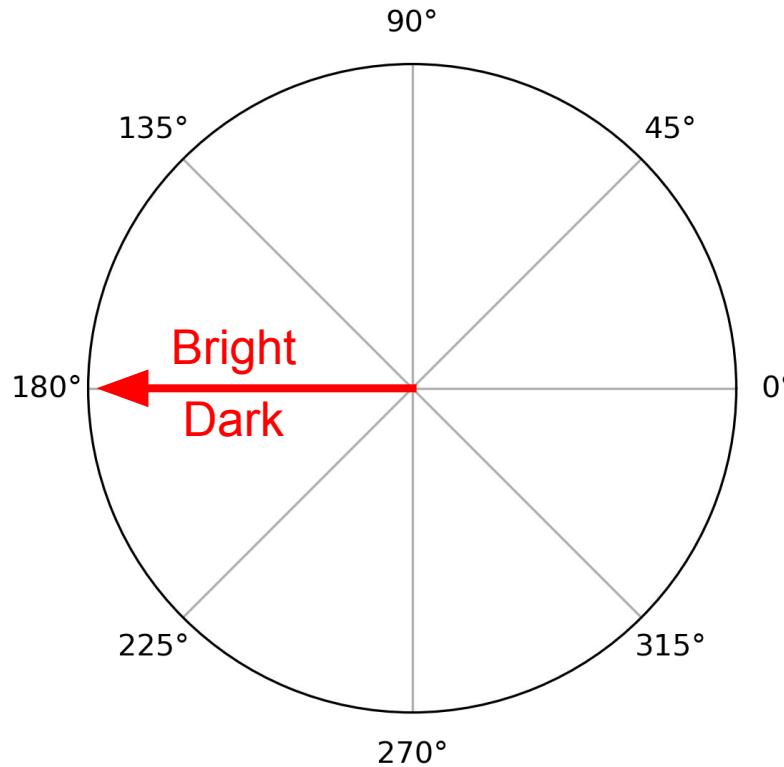


# Edge Detection

Every possible edge from 0 to 360 with 45 degree step.

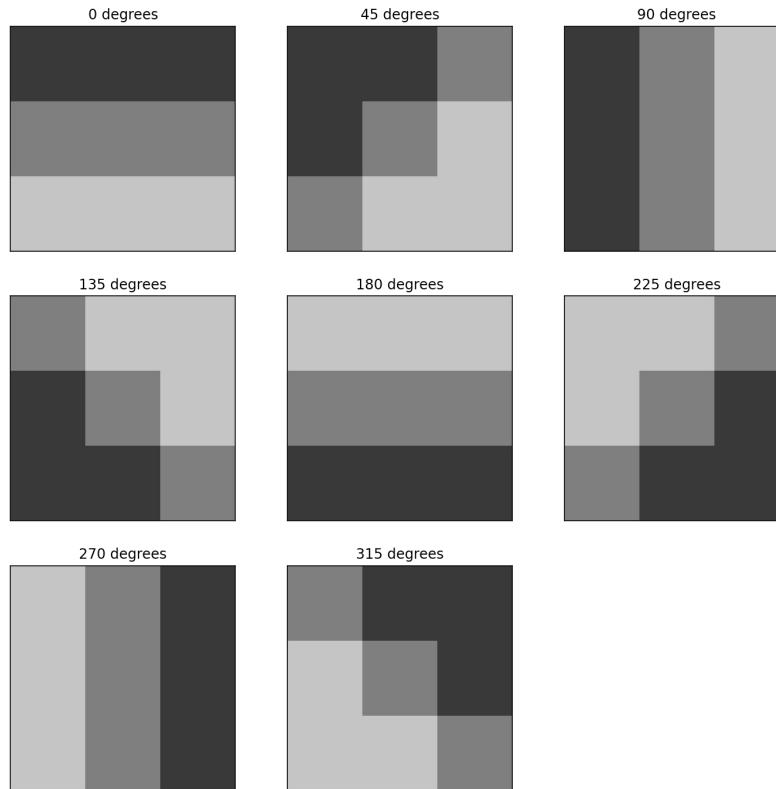
- Dark side on LHS
- Bright side on RHS

Brightness averaged across RGB channels.



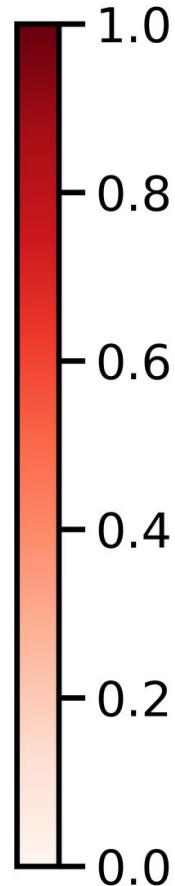
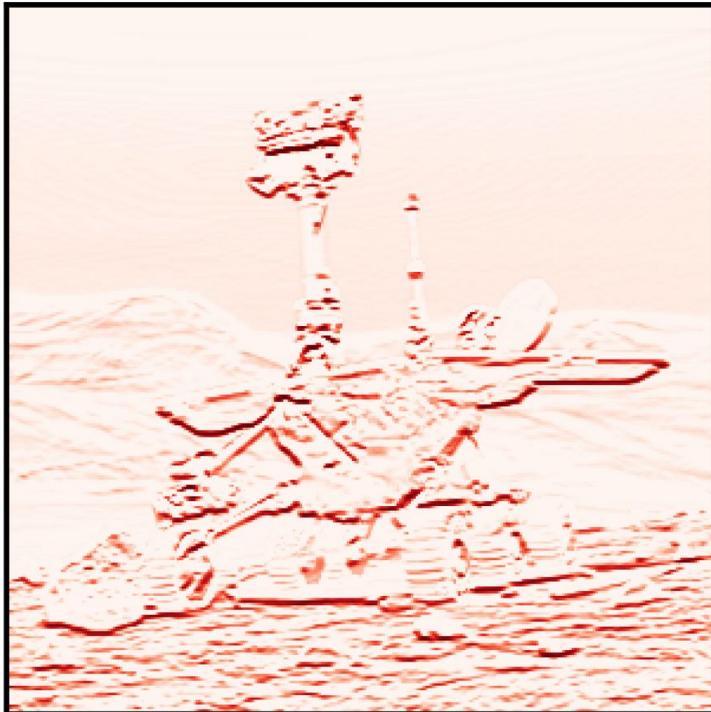
# Filter Visualization

- Manually constructed filters from Edge Detection task.
- $3 \times 3 \times 3 \times 8$
- Rescaled to  $\text{std} = 1$  and transformed via sigmoid for visualization purposes



See [EdgeDetection.ipynb](#) for details and Tensorflow implementation.

0 degrees

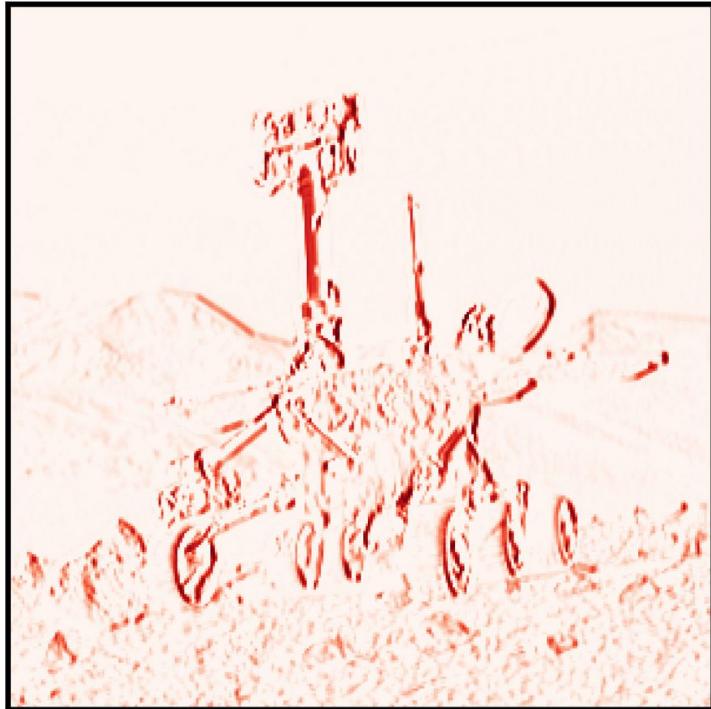


'0 degrees': [  
[-1, -1, -1],  
[ 0, 0, 0],  
[ 1, 1, 1]  
,

- Input =  $300 \times 300 \times 3$
- Filter =  $3 \times 3 \times 3 \times 1$
- Stride = 1
- Padding = same
- Output =  $300 \times 300 \times 1$
- Sigmoid activations for visualization purposes

See [EdgeDetection.ipynb](#) for details and  
Tensorflow implementation.

90 degrees

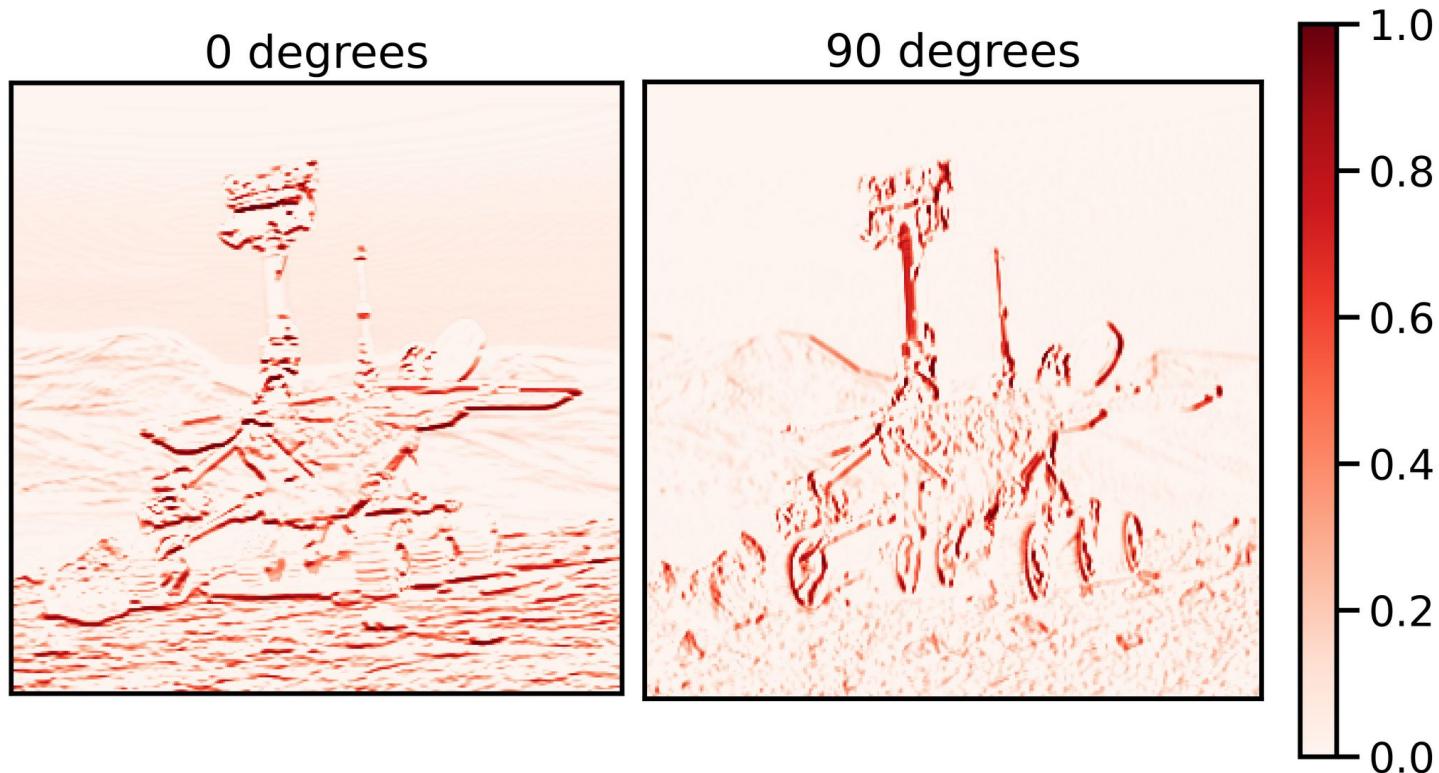


'90 degrees': [ [-1, 0, 1], [-1, 0, 1], [-1, 0, 1] ],

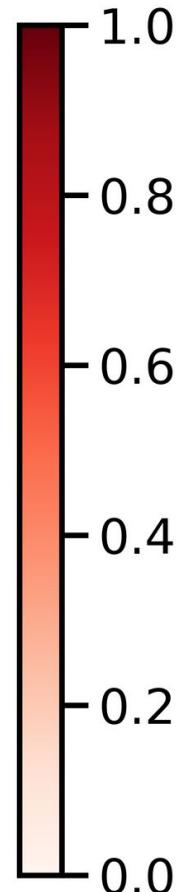
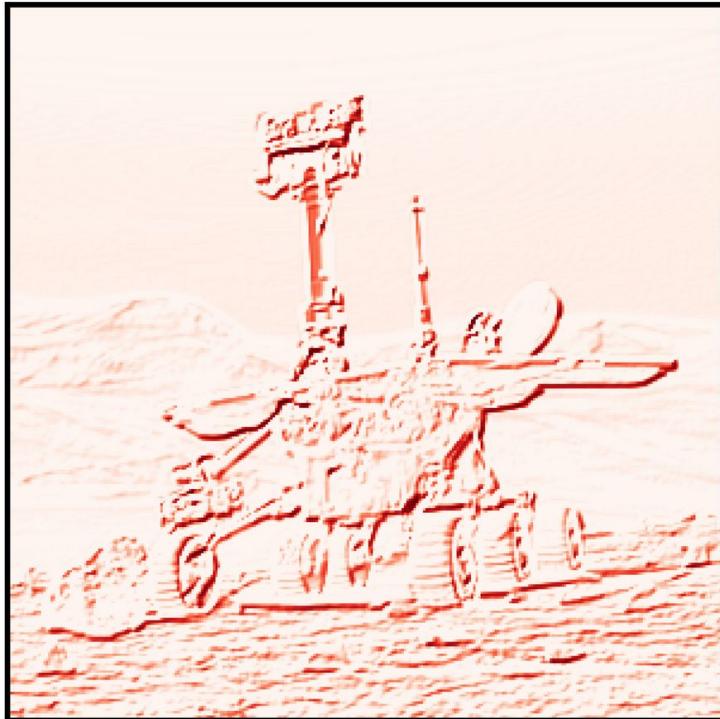
- Input =  $300 \times 300 \times 3$
- Filter =  $3 \times 3 \times 3 \times 1$
- Stride = 1
- Padding = same
- Output =  $300 \times 300 \times 1$
- Sigmoid activations for visualization purposes

See [EdgeDetection.ipynb](#) for details and Tensorflow implementation.

# Comparison of 0- and 90-degree edges



45 degrees



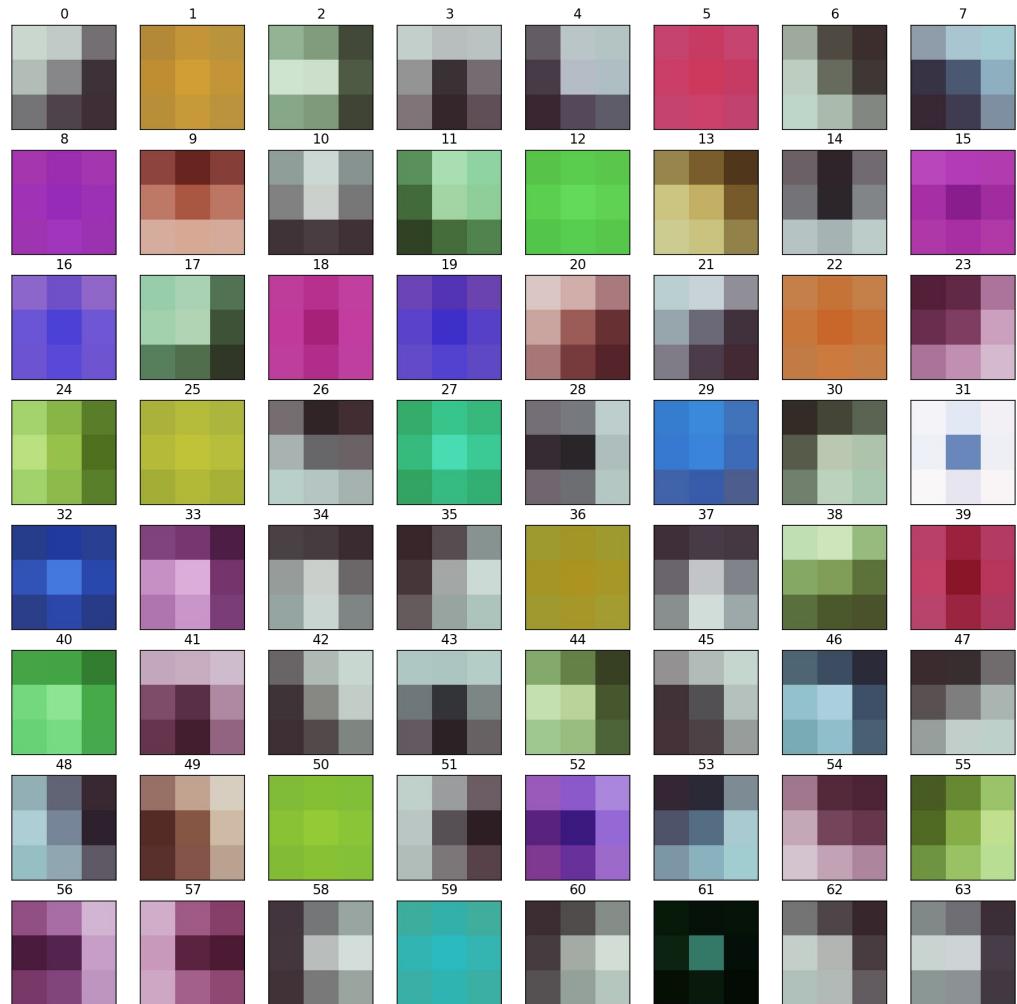
'45 degrees': [  
  [-1, -1, 0],  
  [-1, 0, 1],  
  [ 0, 1, 1]  
,

- Input =  $300 \times 300 \times 3$
- Filter =  $3 \times 3 \times 3 \times 1$
- Stride = 1
- Padding = same
- Output =  $300 \times 300 \times 1$
- Sigmoid activations for visualization purposes

See [EdgeDetection.ipynb](#) for details and Tensorflow implementation.

# Learned Filters

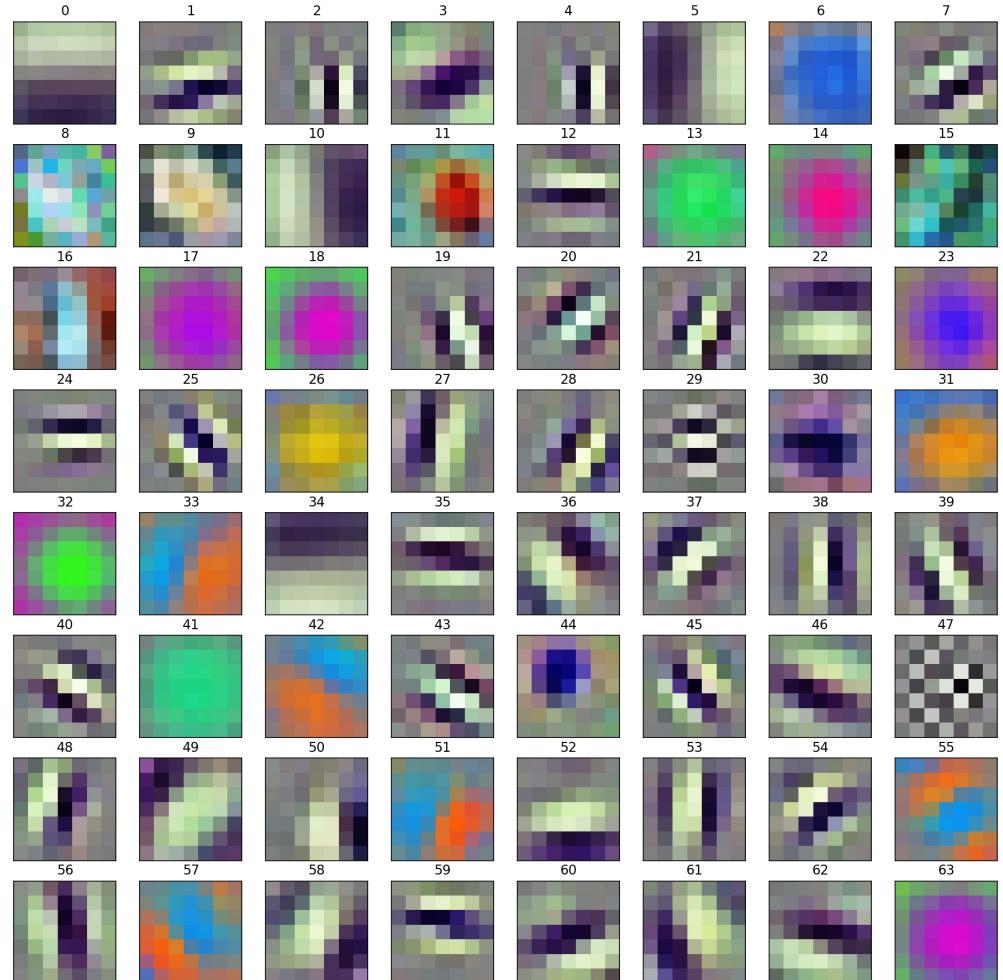
- First layer of VGGNet 16
- $3 \times 3 \times 3 \times 64$
- Trained on ImageNet
- Rescaled to  $\text{std} = 1$  and transformed via sigmoid for visualization purposes



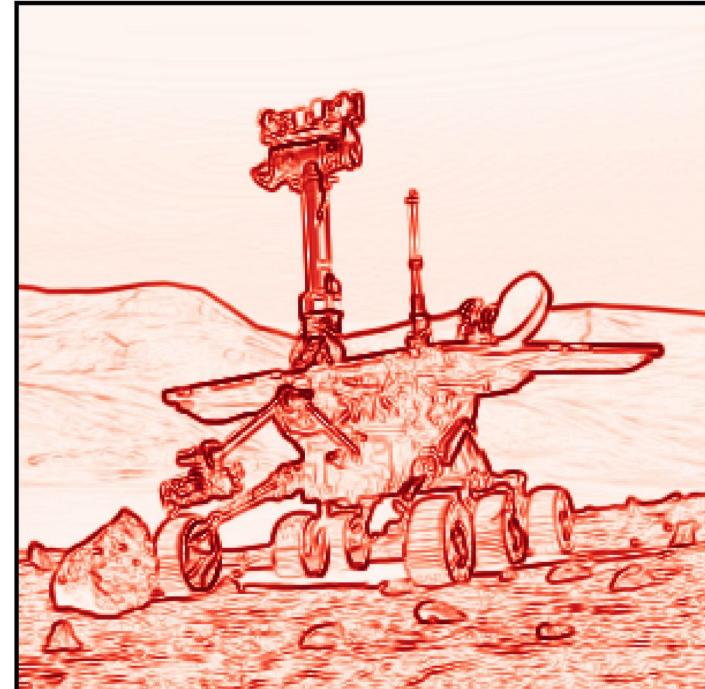
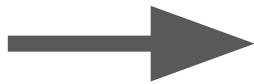
See [VGGDeepDream.ipynb](#) for details and Keras implementation.

# Learned Filters

- First layer of DenseNet201
- $7 \times 7 \times 3 \times 64$
- Trained on ImageNet
- Rescaled to  $\text{std} = 1$  and transformed via sigmoid for visualization purposes



# What is edge detection?



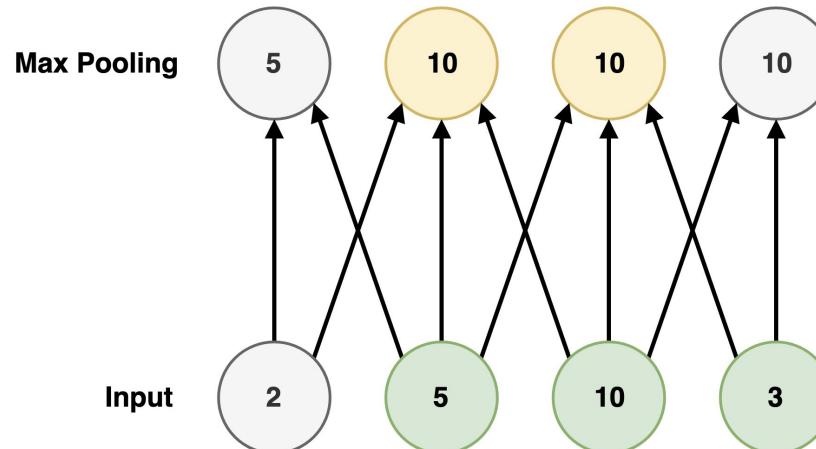
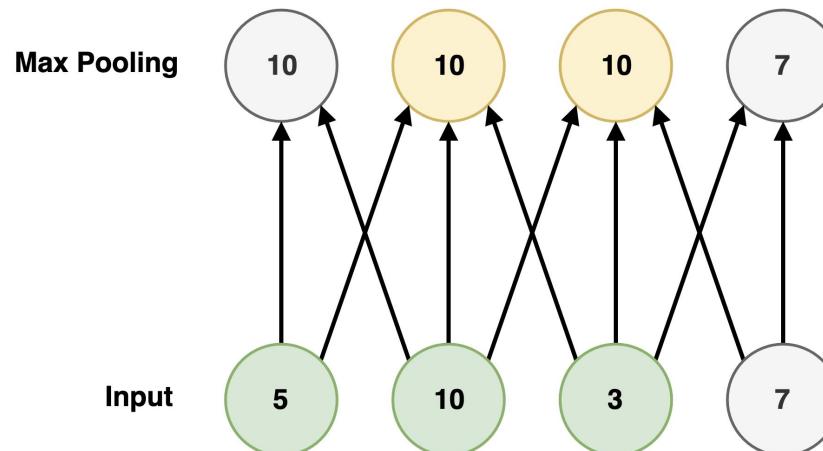
# Max Pooling

## Problem

Convolutions are invariant with respect to linear transformations, but output is still affected.

## Solution

Pooling over spatial positions is invariant with respect to linear translations in the input.



# Max Pooling over Space

Input: 3x3 + 1x1 padding

1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0

Filter  
2x2



Output: 3x3

5	6	6
8	9	9
8	9	9

# Max Pooling over Space

Input: 3x3 + 1x1 padding

1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0

Filter  
2x2



Output: 3x3

5	6	6
8	9	9
8	9	9

# Max Pooling over Space

Input: 3x3 + 1x1 padding

1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0

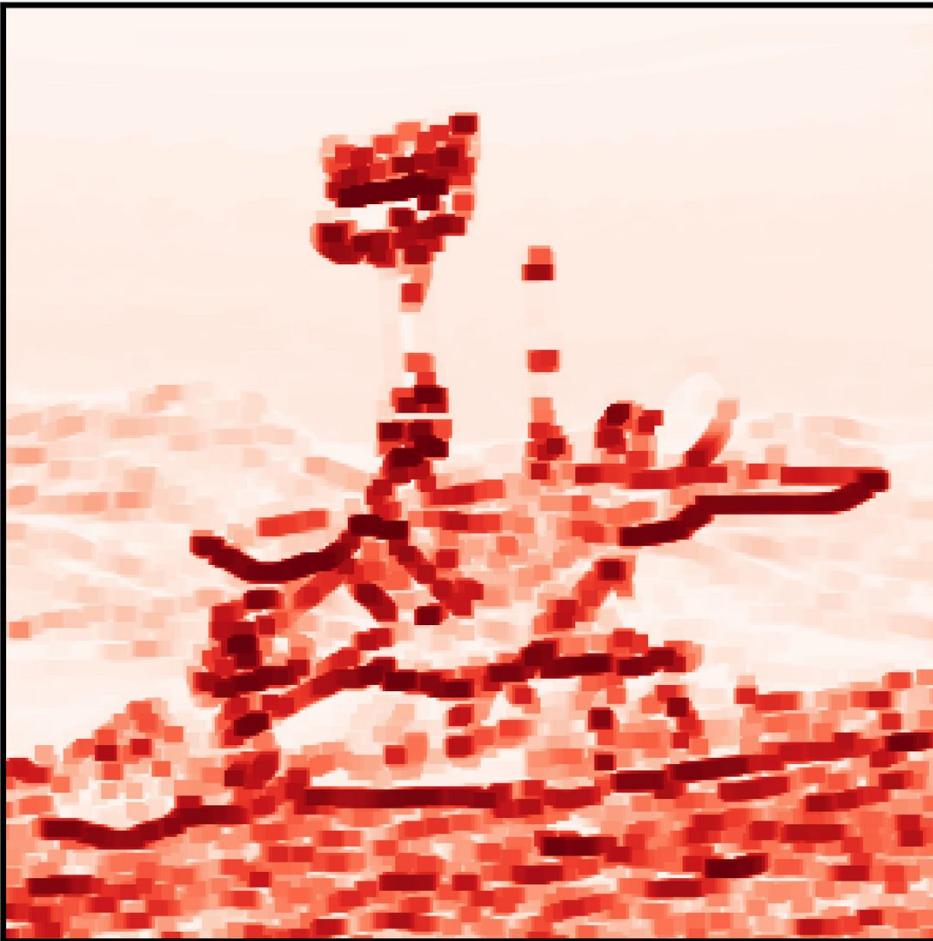
Filter  
2x2



Output: 3x3

5	6	6
8	9	9
8	9	9

Filter:  $5 \times 5 \times 1$



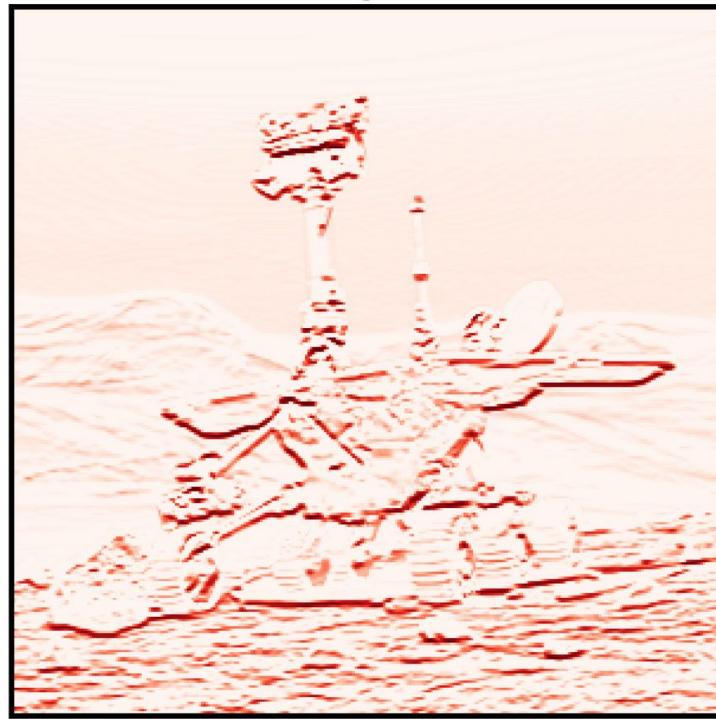
## Max Pooling over Space

- Input:  $300 \times 300 \times 8$   
Convolution → Sigmoid
- Filter =  $5 \times 5 \times 1$
- Stride =  $1 \times 1 \times 1$
- Padding = 0
- Output =  $296 \times 296 \times 8$   
Displaying the channel from the  
0-degree edge activation map

See [EdgeDetection.ipynb](#) for details and  
Tensorflow implementation.

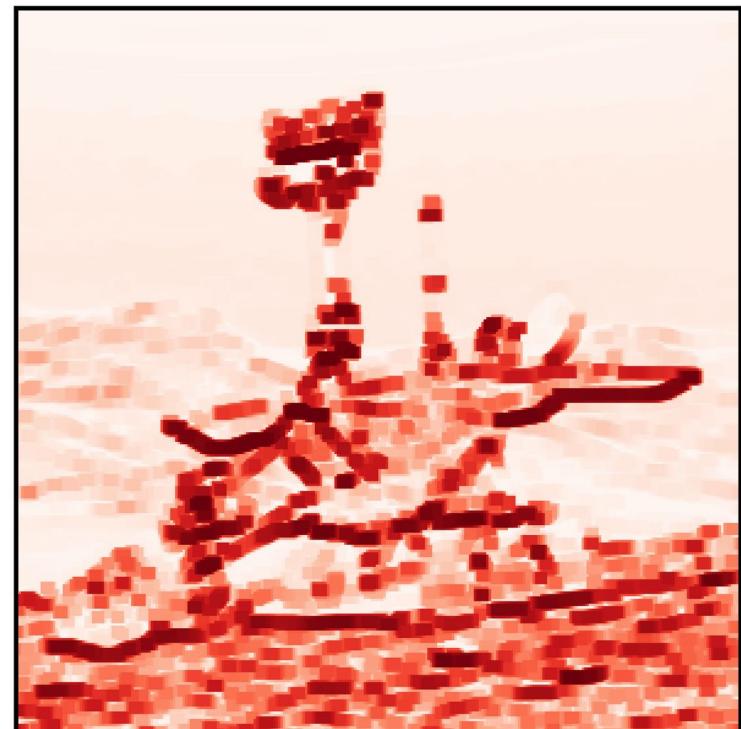
Before pooling

0 degrees

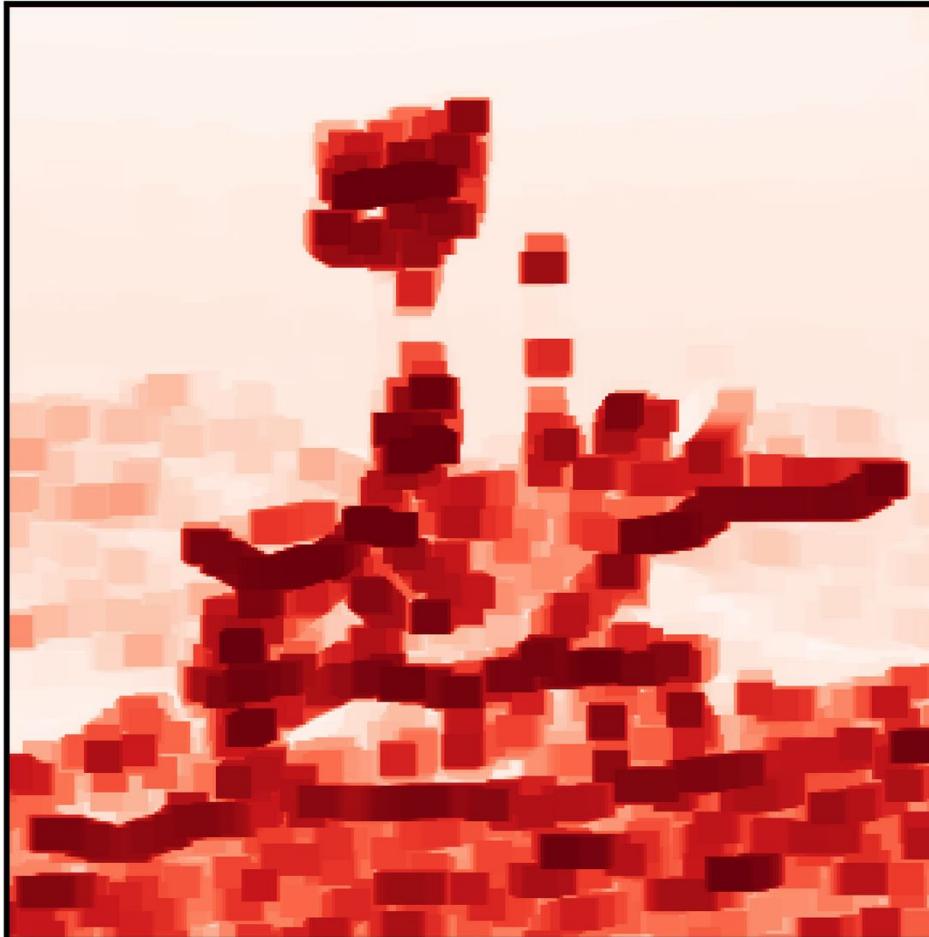


After pooling

Filter:  $5 \times 5 \times 1$



Filter:  $10 \times 10 \times 1$



## Max Pooling over Space

- Input:  $300 \times 300 \times 8$   
Convolution → Sigmoid
- Filter =  $10 \times 10 \times 1$
- Stride =  $1 \times 1 \times 1$
- Padding = 0
- Output =  $291 \times 291 \times 8$
- Displaying the channel from the 90-degree edge activation map

See [EdgeDetection.ipynb](#) for details and  
Tensorflow implementation.

Filter:  $30 \times 30 \times 1$



Max Pooling over Space

- Input:  $300 \times 300 \times 8$   
Convolution → Sigmoid
- Filter =  $30 \times 30 \times 1$
- Stride =  $1 \times 1 \times 1$
- Padding = 0
- Output =  $271 \times 271 \times 8$
- Displaying the channel from the 90-degree edge activation map

See [EdgeDetection.ipynb](#) for details and Tensorflow implementation.

# Max Pooling over Depth

## Problem

Detect features in the same spatial location but in different channels.

## Solution

Take a maximum of the same surface location across all channels.

Input  
 $2 \times 2 \times 3$

10	8
4	40

Output  
 $2 \times 2 \times 1$

2	5
30	6

Filter  
 $1 \times 1 \times 3$



10	8
30	40

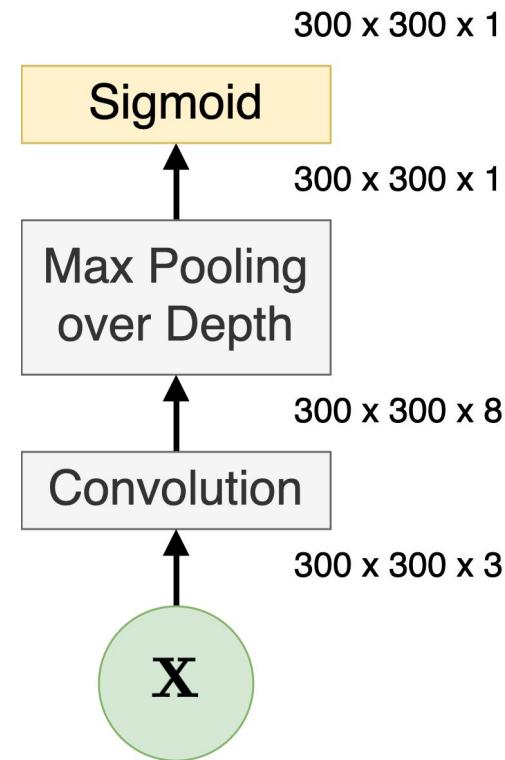
5	0
5	40

# Manual Edge Detection

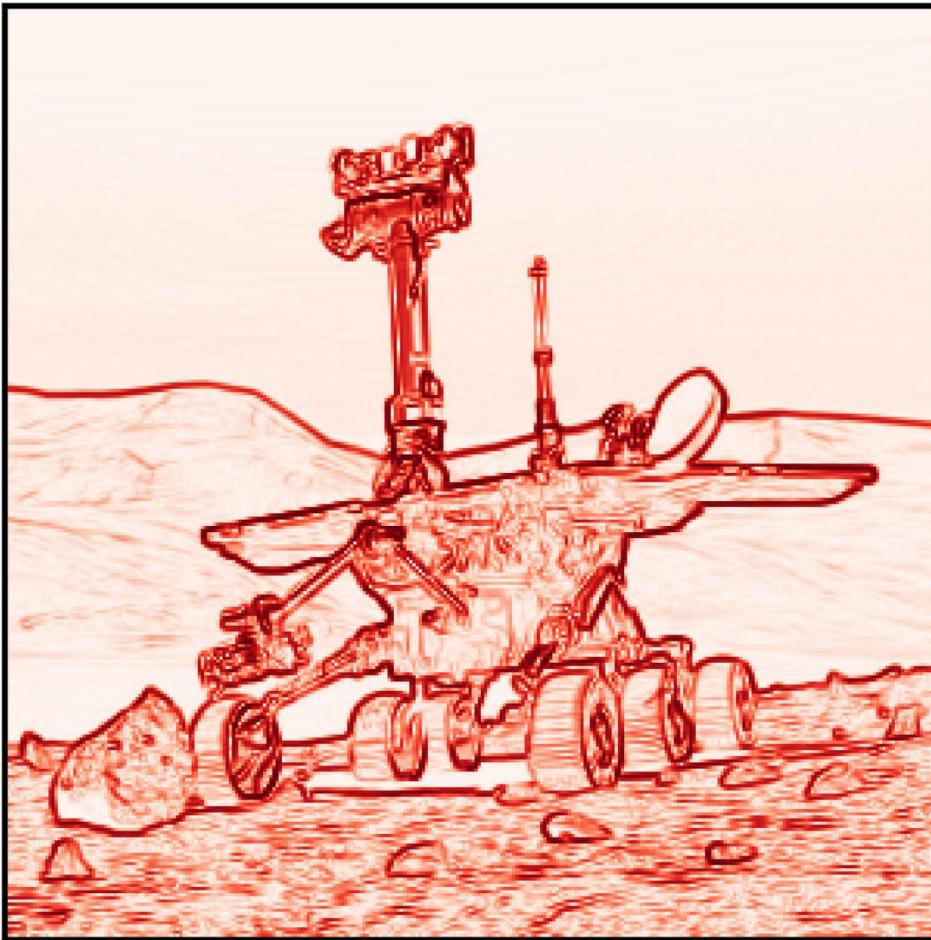
**Sigmoid** for visualization

**Max pooling over depth**  
summarizes activation across  
angles

**Convolution** detects edges with  
specific angle



Filter:  $1 \times 1 \times 8$



Max Pooling over Depth

- Input:  $300 \times 300 \times 8$   
Convolution → Sigmoid
- Filter =  $1 \times 1 \times 8$
- Stride =  $1 \times 1 \times 8$
- Padding = 0
- Output =  $300 \times 300 \times 1$

See [EdgeDetection.ipynb](#) for details and  
Tensorflow implementation.

# Part 2

## Datasets & Models



(Image by a Stable Diffusion model)

# Project ImageNet

## Dataset

14M images

20K classes

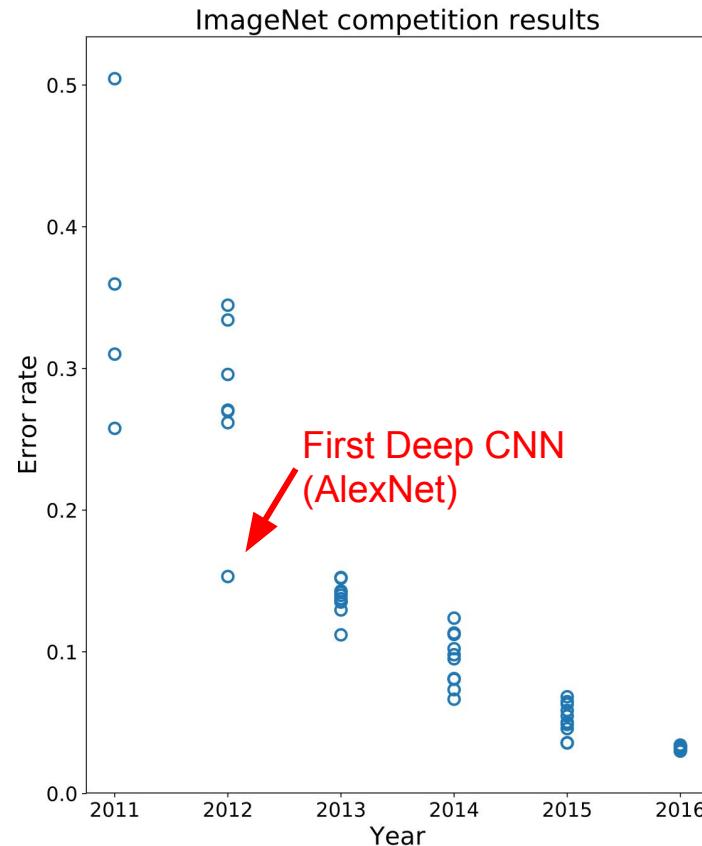
Amazon Mechanical Turk

## Competition

ImageNet Large Scale Visual Recognition  
Challenge (ILSVRC)

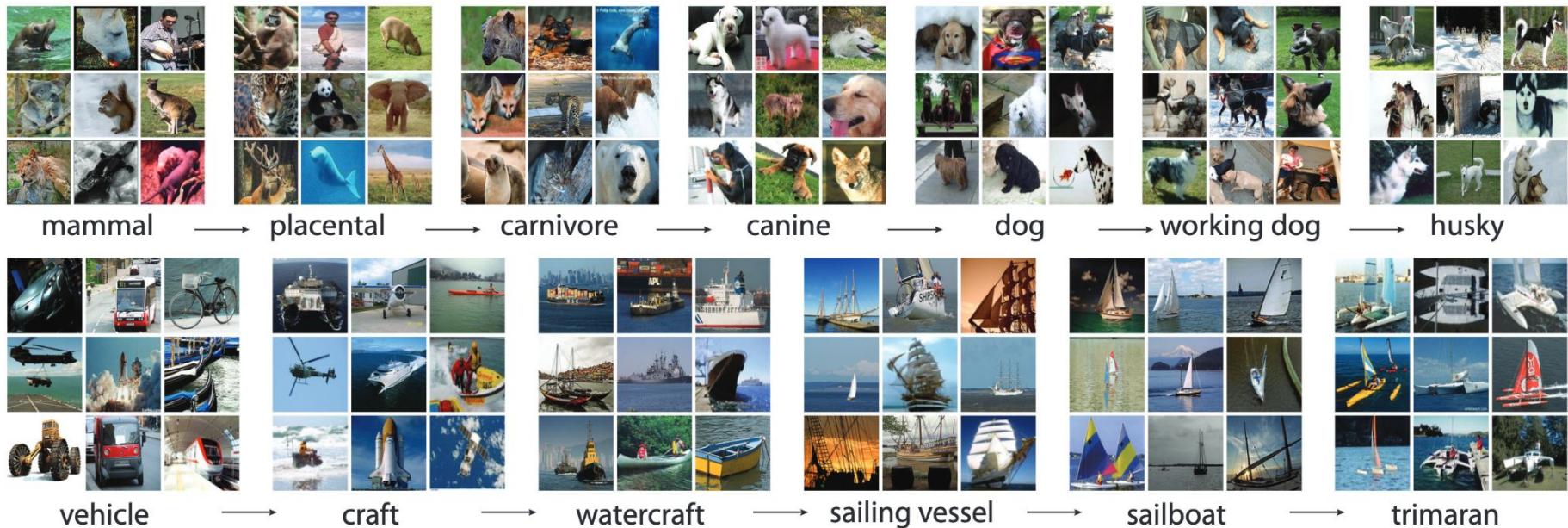
Timeline: 2010 - 2017

Subset: 1K classes



Source: <https://commons.wikimedia.org/w/index.php?title=User:Gkrusze>  
License: CC BY-SA 4.0

# Examples of ImageNet



# AlexNet

## ImageNet Results

Winner of 2012

Top-5 Error: 17.0%

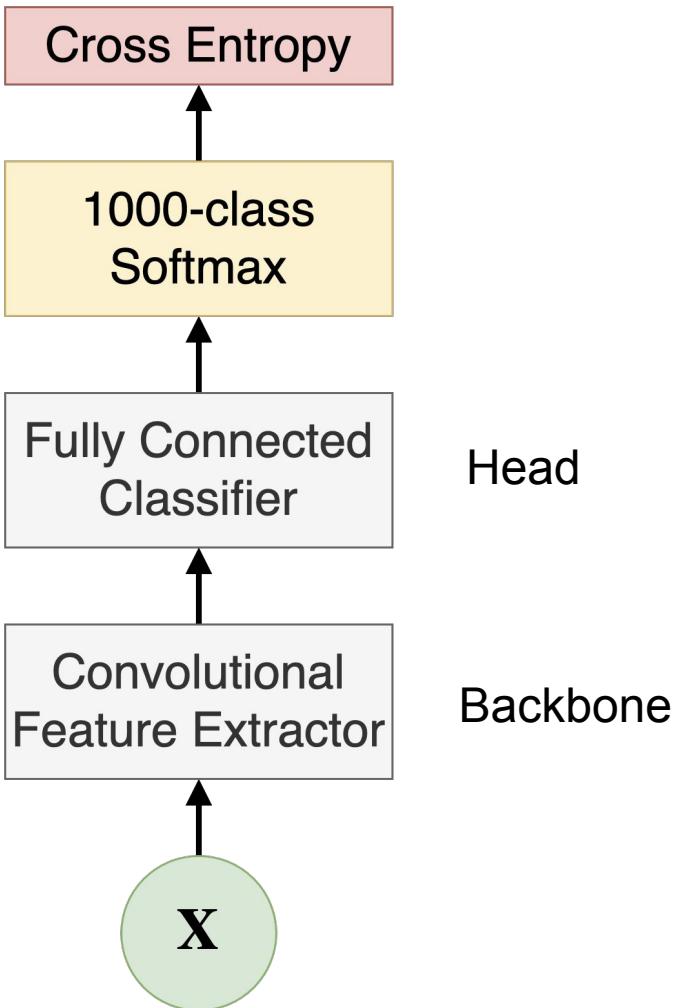
First to successfully use Deep CNN approach in this competition.

## Hyperparameters

Momentum GD

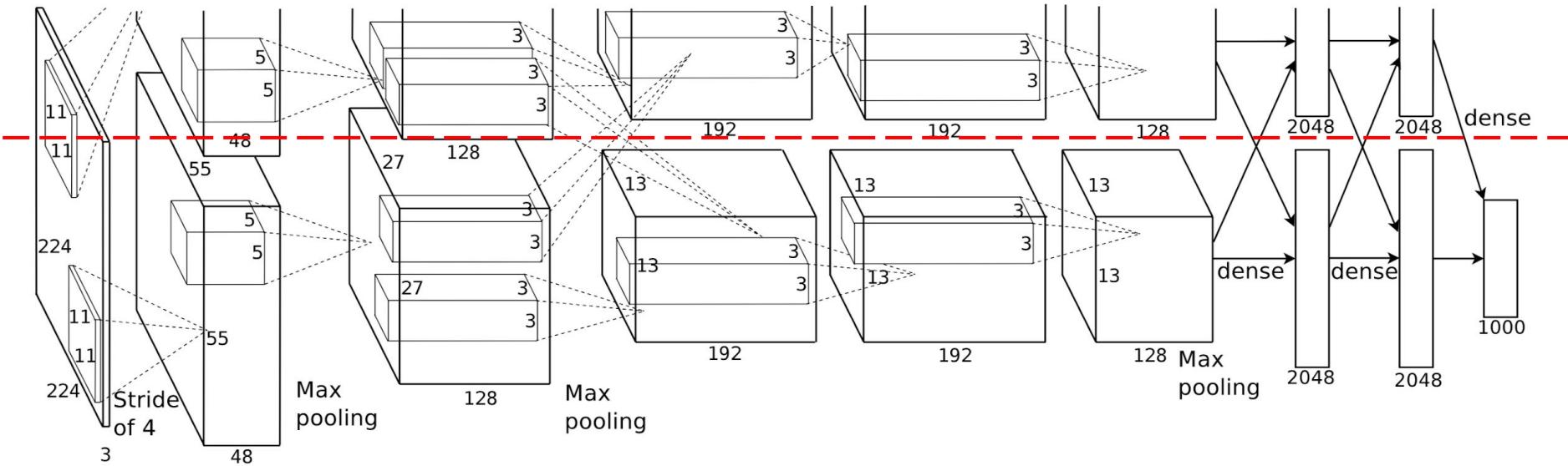
Batch size = 128

L2 Regularization



L	Type	Filter Shape	Padding	Stride	Activation Shape	Activation	Parameters
0	Input				(224 + 3) x (224 + 3)		
1	Convolution	11 x 11 x 3	Valid	4	55 x 55 x (48 + 48)	ReLU + LRN	34,848
2	Max Pooling	3 x 3 x 48	Valid	2	27 x 27 x (48 + 48)		
3	Convolution	5 x 5 x 48	Same	1	27 x 27 x (128 + 128)	ReLU + LRN	307,200
4	Max Pooling	3 x 3 x 128	Valid	2	13 x 13 x (128 + 128)		
5	Convolution	3 x 3 x 128	Same	1	13 x 13 x 384	ReLU	442,368
6	Convolution	3 x 3 x 384	Same	1	13 x 13 x (192 + 192)	ReLU	1,327,104
7	Convolution	3 x 3 x 192	Same	1	13 x 13 x (128 + 128)	ReLU	442,368
8	Max Pooling	3 x 3 x 128	Valid	2	6 x 6 x (128 + 128)		
9	Fully Connected				4096	ReLU	37,748,736
10	Fully Connected				4096	ReLU	16,777,216
11	Fully Connected				1000	Softmax	4,096,000

# AlexNet: Architecture



# ResNet

## ImageNet Results

Winner of 2015

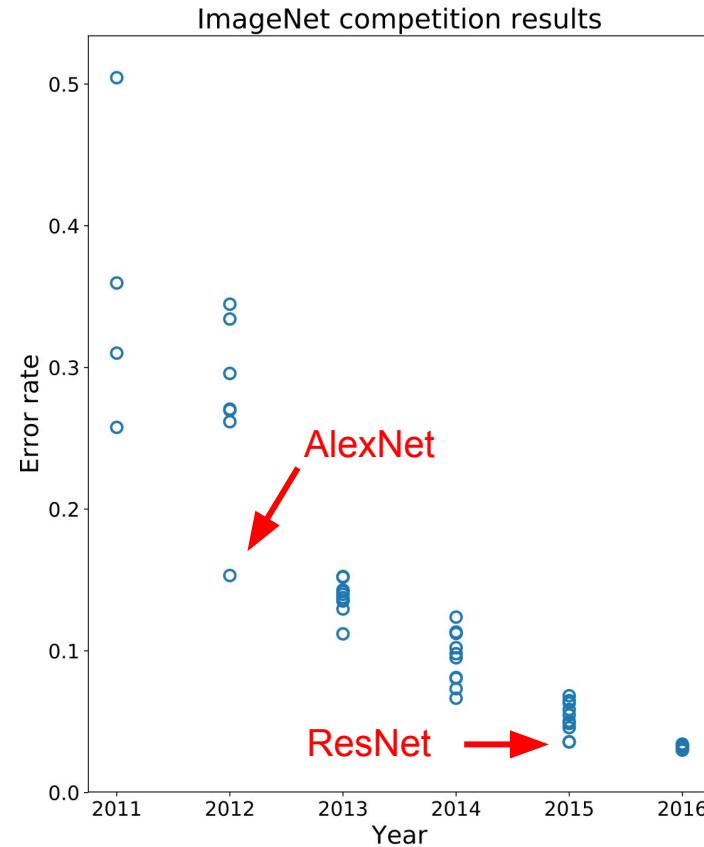
Top-5 Error: 3.57%

## Architecture

152 trainable layers

25M parameters

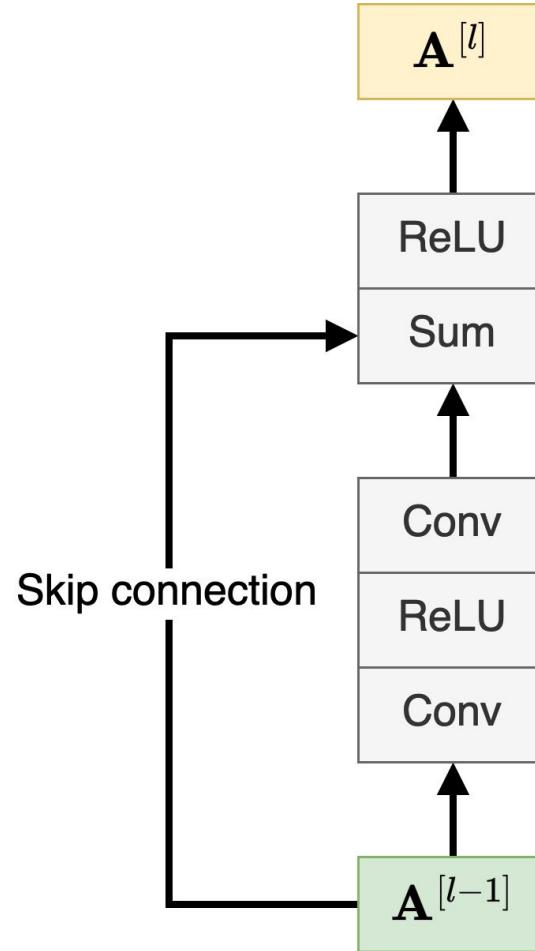
Surpassed human performance.



# Residual Block

Why is this useful?

- 1) Model can learn to turn-off layers
- 2) Reduces **vanishing gradient** problem
  - Question: why?





# ResNet Family

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer	
conv1	$112 \times 112$			$7 \times 7, 64, \text{stride } 2$			
conv2_x	$56 \times 56$			$3 \times 3 \text{ max pool, stride } 2$			
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	
conv3_x	$28 \times 28$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$	
conv4_x	$14 \times 14$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$	
conv5_x	$7 \times 7$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	
	$1 \times 1$			average pool, 1000-d fc, softmax			
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$	

Source: He, K., et al. (2015)

# EfficientNet

## ImageNet Results

Top-1 Accuracy: 84%

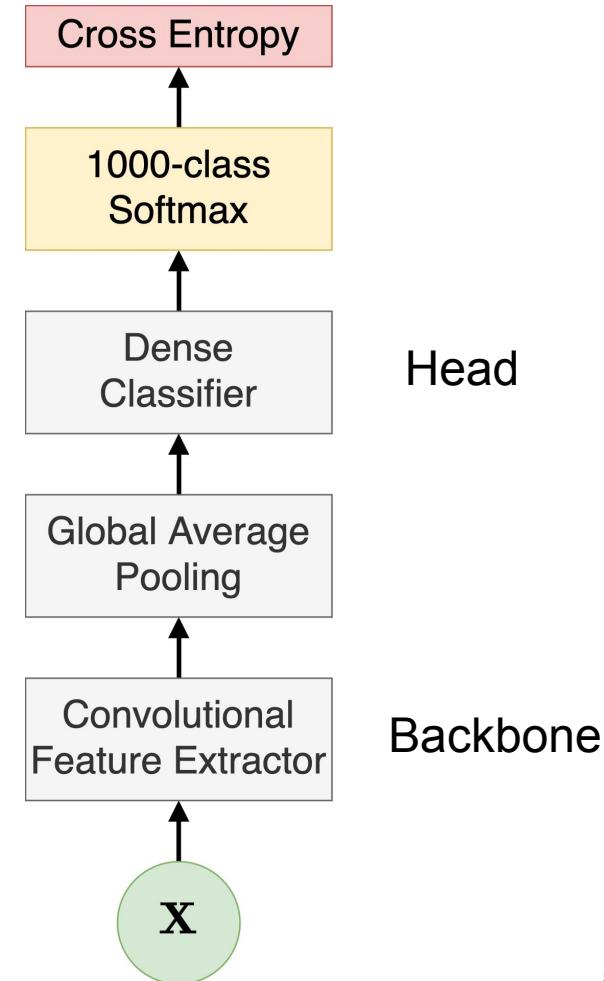
Top-5 Accuracy: 97%

## Architectural Insight

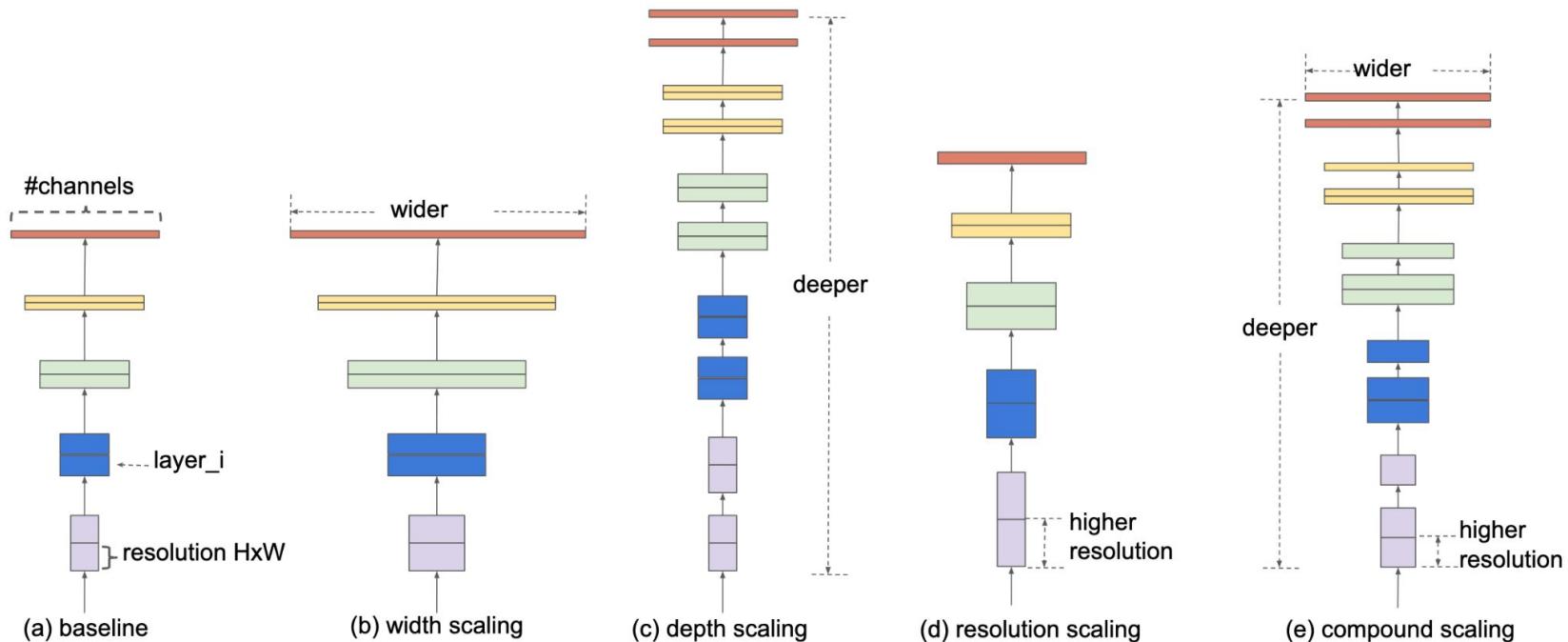
(1) Use **Neural Architecture Search** to automate experimentation (RNN + RL).

(2) Find an **optimal scaling ratio** for layer count, convolution width, and depth.

Source: Tan, M., et al. (2021)



# Efficient Model Scaling



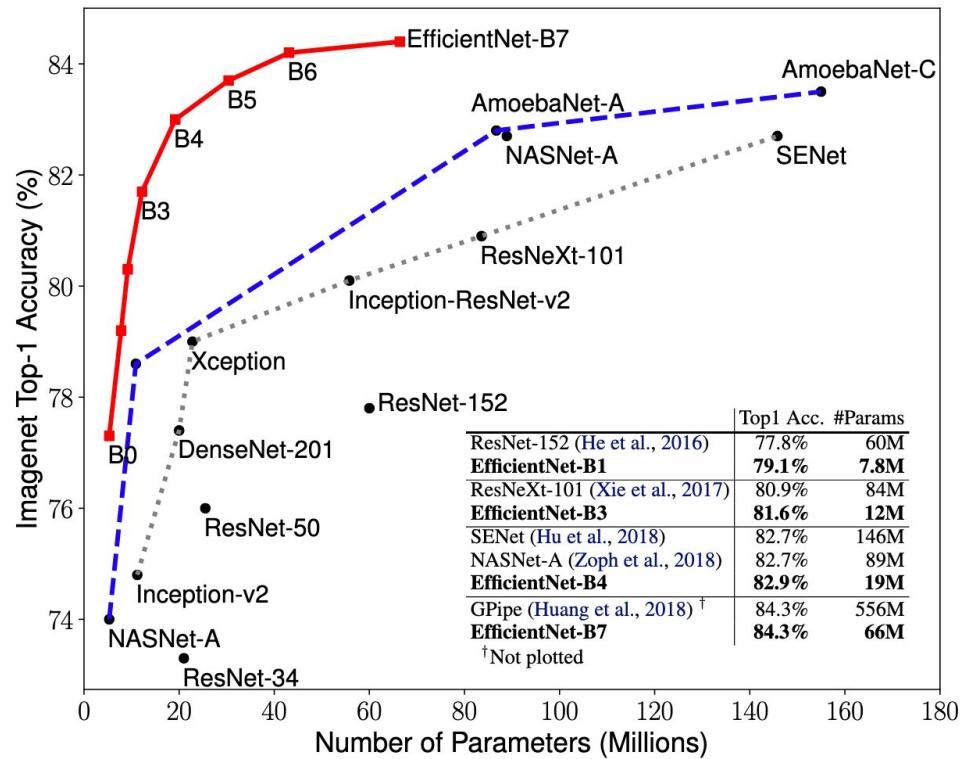
Source: Tan, M., et al. (2021)

# Compute/Accuracy Trade-off

**FLOPS** = floating point operations (per second)

**NAS** optimizes for accuracy and FLOPS, which leads to better trade-off points.

**EfficientNet** offers both best accuracy and faster inference.



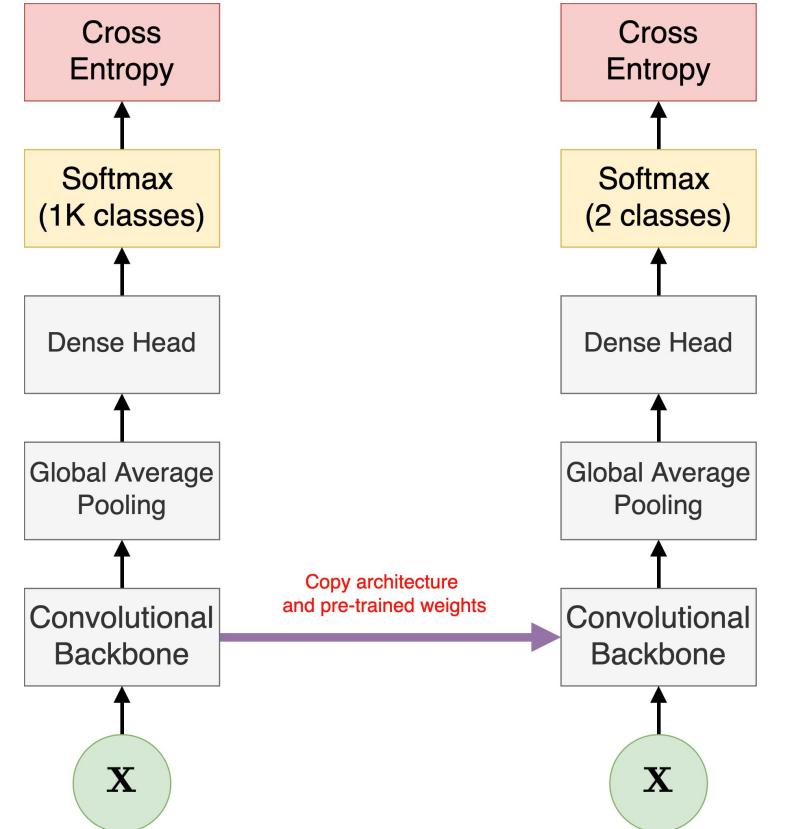
# Demo

EfficientNetClassification.ipynb

# Transfer Learning

## Example

A new model is initialized from a **pre-trained** convolutional backbone.

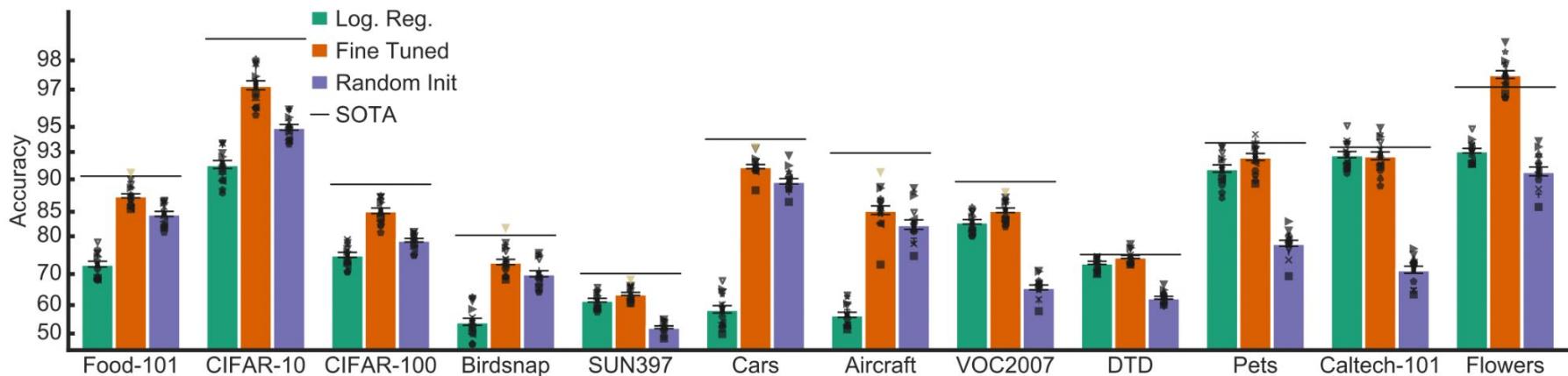


# Transfer Learning

A few design choices:

1. Whether to **copy all or some** of the convolutional layers?
2. Whether to **fine-tune** the pre-trained convolutional layers?
3. Whether to **add new** convolutional layers after the copies ones?
4. How many **dense layers** to put into the new head?
5. ...

# Transfer Learning from ImageNet



Source: Do Better ImageNet Models Transfer Better? (Kornblith, S, et al, 2018)

# Demo

AssignmentTemplate.ipynb

# Assignment #2

- Learning Objectives:
  - Inspecting pre-trained models
  - Transfer learning to a new task
- Deadline: **EOD 11/24**
- Instructions: **AssignmentTemplate.ipynb**
- If Accuracy @ 0.5 threshold > 60%, then
  - + 10 points flat
  - + 10 points proportionally to the rank within class
  - In 2023, the median accuracy was 87%
- If Accuracy @ 0.5 threshold <= 60%
  - Zero points

# Assignment Deliverables

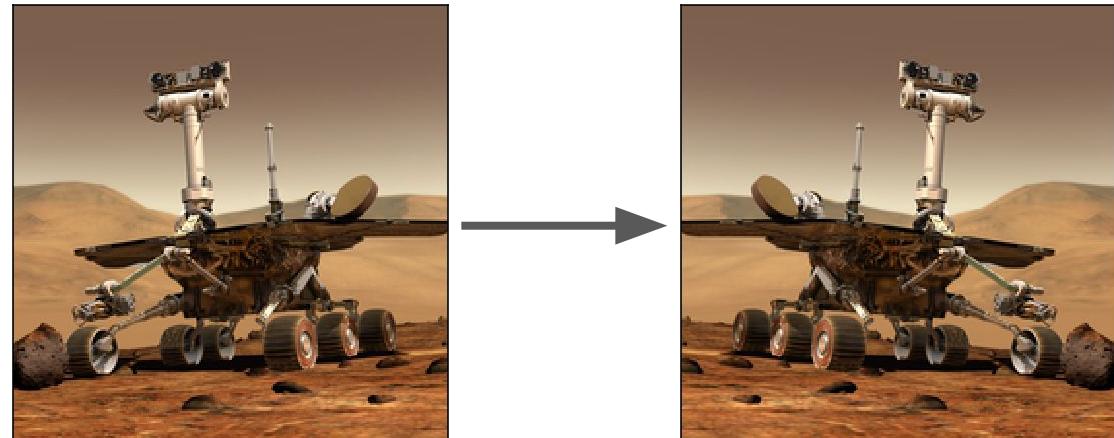
1. Jupyter notebook and/or Python script
  - a. Data preprocessing
  - b. Construction and training of the final model
  - c. Inference on the Score segment
2. Model architecture
  - a. Serialization in JSON format
3. Model parameters
  - a. Serialization in Keras format
4. Probabilities of Person Label for the Score segment
  - a. Shape = 20K x 1
  - b. Data Type = float32 in [0, 1]
  - c. File Type = Parquet

# Regularization

## Data Augmentation

- Flip
- Scale
- Color shift
- Rotate
- Crop

Any change that produces  
a different image but  
preserves the label.



Example: Horizontal Flip

# Notable Papers

- NASNet (Zoph, B, et al, 2018)
- Inception-ResNet (Szegedy, C., et al, 2017)
- DenseNet (Huang, G, et al, 2018)
- EfficientNet (Tan, M, et al, 2020)
- EfficientNetV2 (Tan, M, et al, 2021)
- ConvNeXt (Liu, Z, et al, 2022)

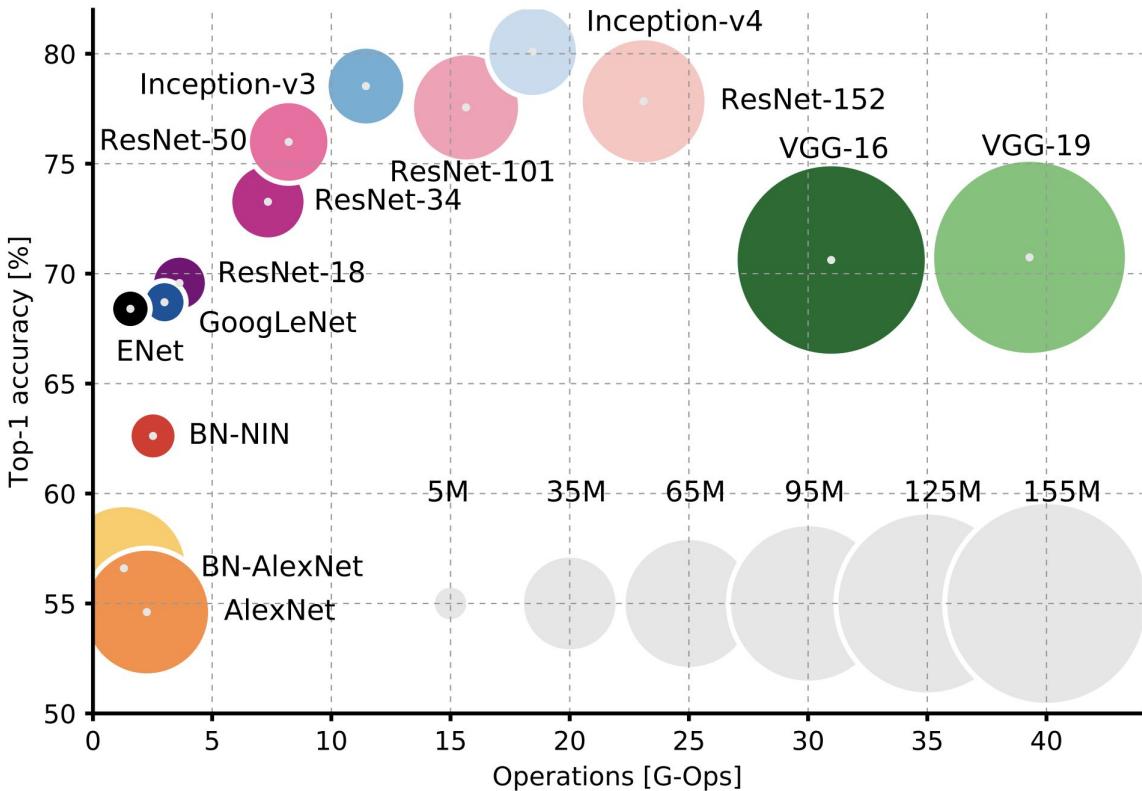
Pre-trained models available in Keras Hub.

# Model Zoo

**Trade-off Dimensions**  
Inference time  
Memory consumption  
Accuracy

**Strong Preferences**  
ResNet  
Inception

Note: includes classifier parameters.



# Project COCO

## Dataset

200,000 images and 80 categories

Browse online! <https://cocodataset.org/#explore>

## COCO Explorer

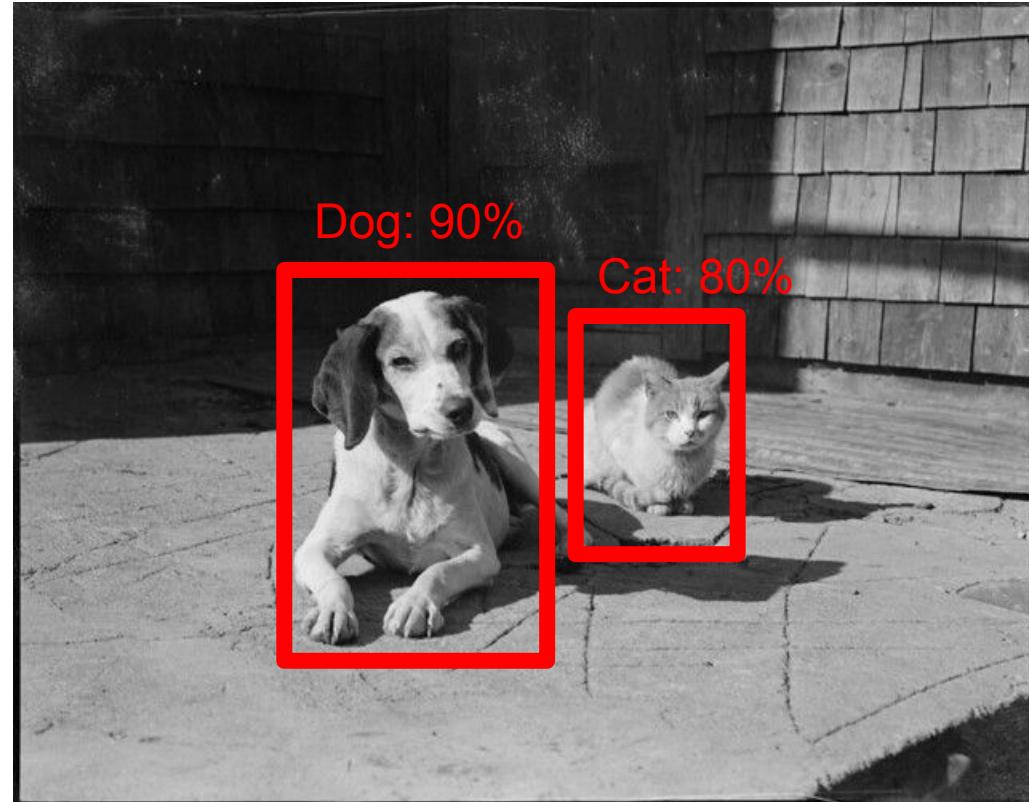
COCO 2017 train/val browser (123,287 images, 886,284 instances). Crowd labels not shown.



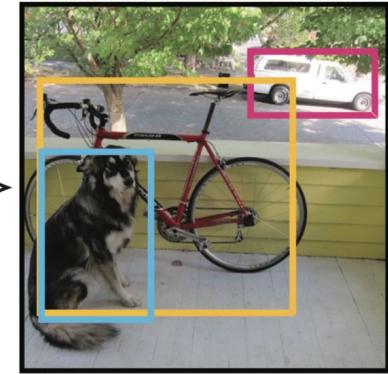
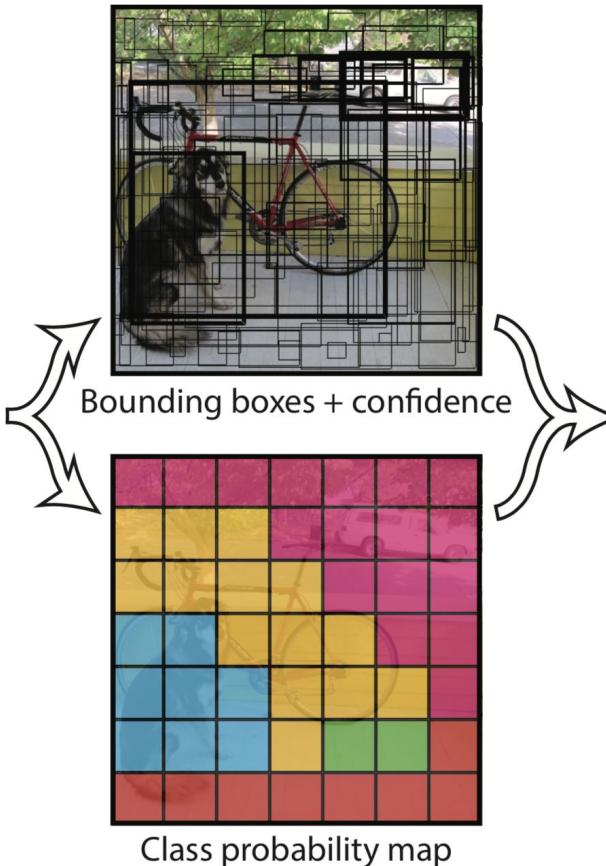
# Object Detection and Localization

## Problem

Identify any number of objects by their **class probability** and a **bounding box** providing their location on the image.



# You Only Look Once



Each cell is responsible for detecting an object's center.

# YOLO: Model Output

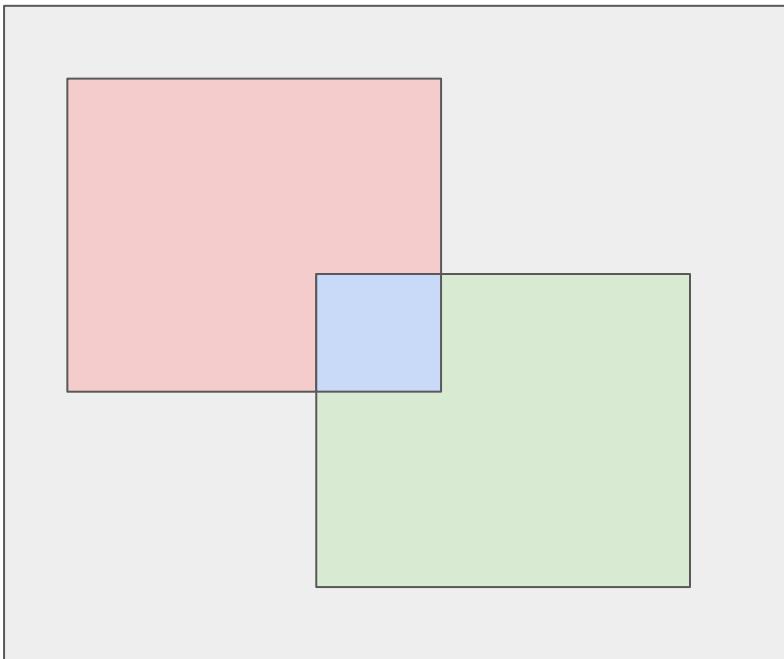
Each of **SxS** cells predicts **B** boxes and a **distribution over C classes**.

Each of **B** boxes consists of:

- **Confidence score** = Probability of object x IOU
- **Location** = the object's center relative to the cell's top-left corner
- **Dimensions** = the object's dimensions relative to the image size

The cost function is a multi-part MSE.

# YOLO: Intersection over Union



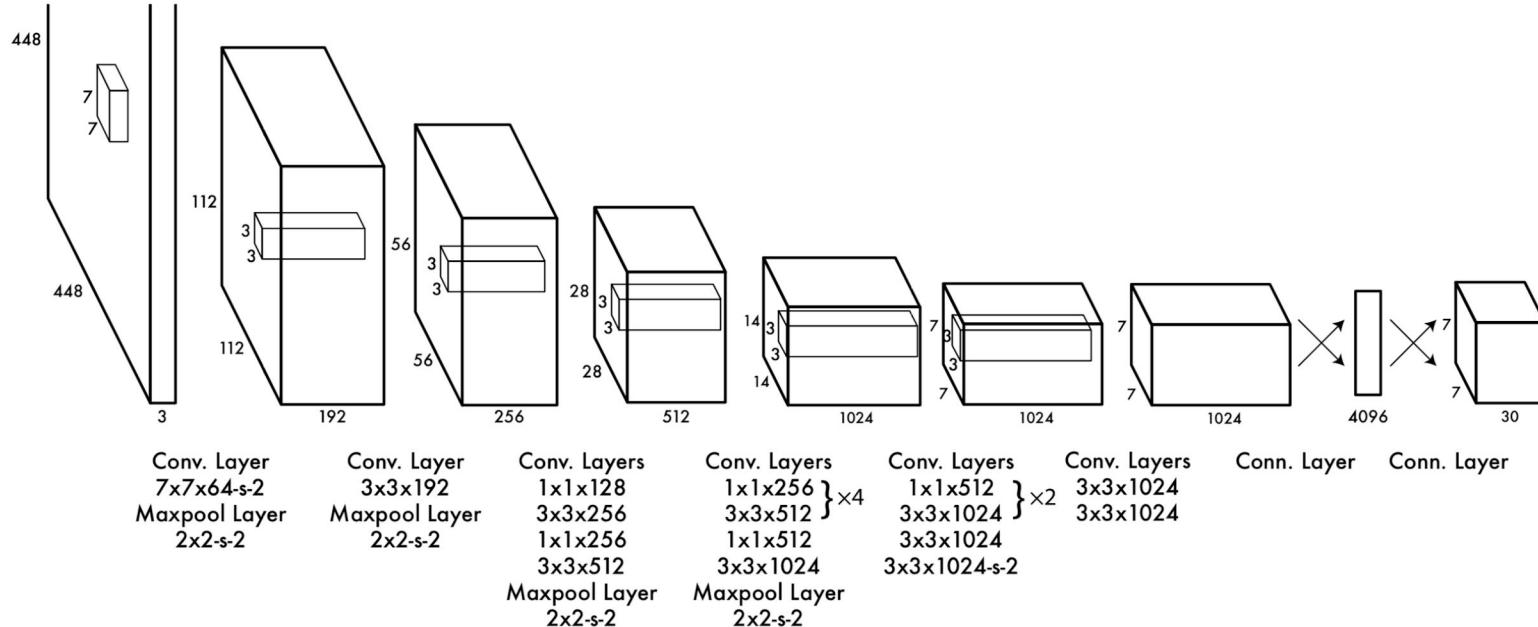
$\text{IOU} = \text{Area of Intersection} / \text{Area of Union}$   
 $\text{IOU} \in [0, 1]$

Typically,  $\text{IOU} > 0.5$  is accepted as a correct bounding box.

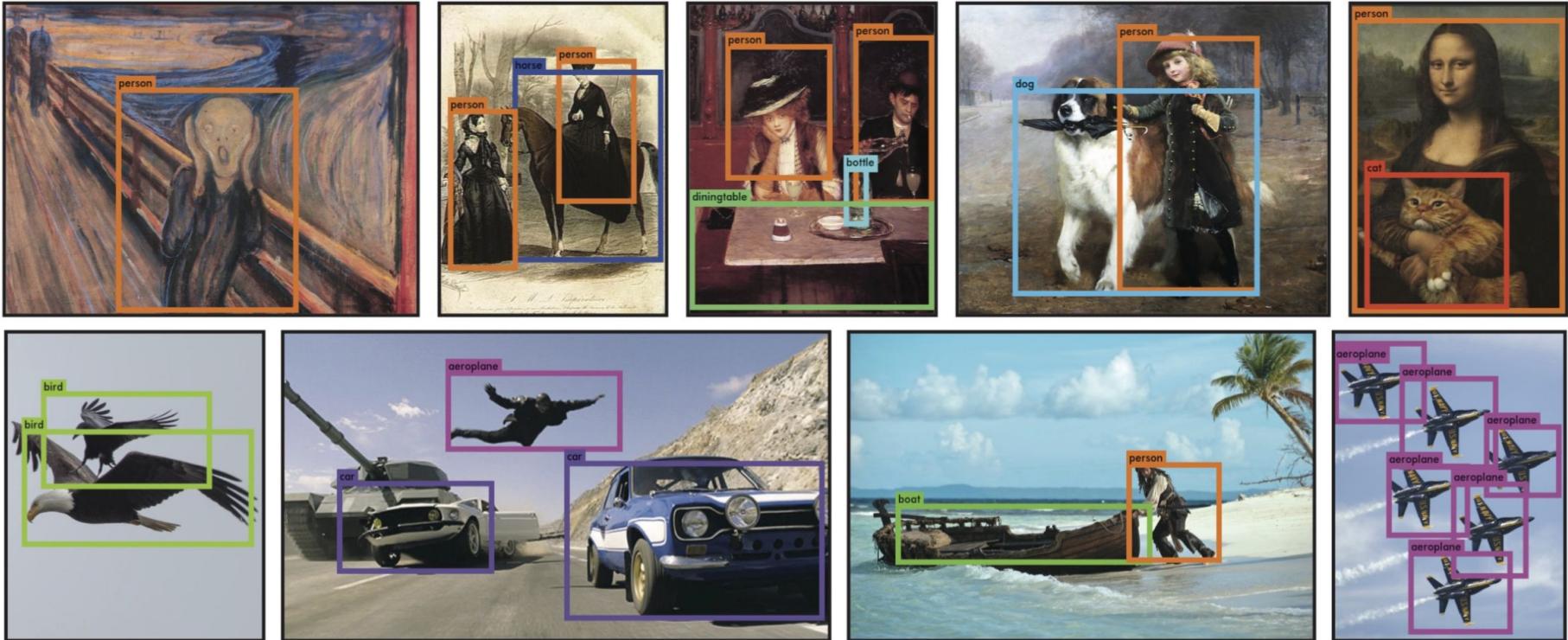
Example:

$$\text{IOU} = \text{Blue} / (\text{Red} + \text{Green} + \text{Blue})$$

# YOLO: Architecture



Source: You Only Look Once: Unified, Real-Time Object Detection  
Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi (2015)



Source: You Only Look Once: Unified, Real-Time Object Detection  
Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi (2015)

# Demo

YOLO V3

<https://danylo-ucla.s3.us-west-2.amazonaws.com/YOLOv3.mp4>

# Demo

YoloObjectDetection.ipynb

# Instance Segmentation Task



## Problem

Detect object class and pixels that belong to individual instances of that class.

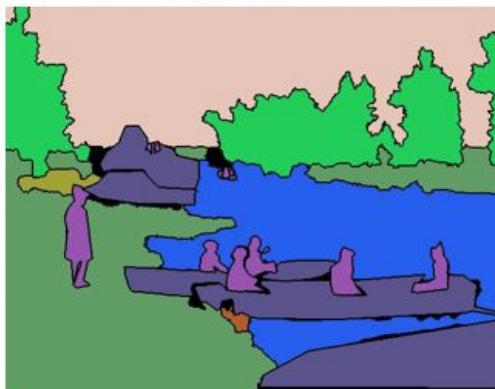
# Stuff Segmentation Task



## Problem

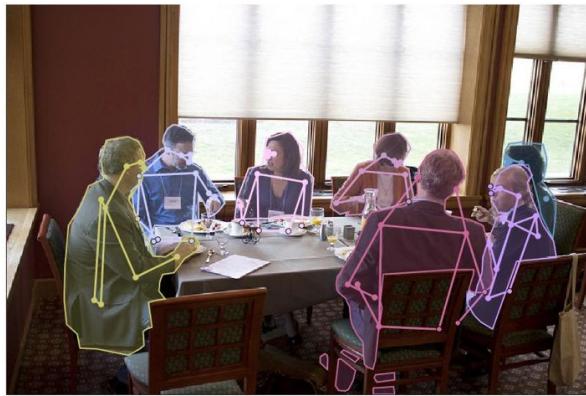
Detect non-object classes and pixels that belong to image regions of that class.

# Panoptic Task



Source: COCO (CC BY 4.0) / Flickr

# Keypoint Task



Source: COCO (CC BY 4.0) / Flickr

# DensePose Dataset



Examples of Annotations

# Detectron2

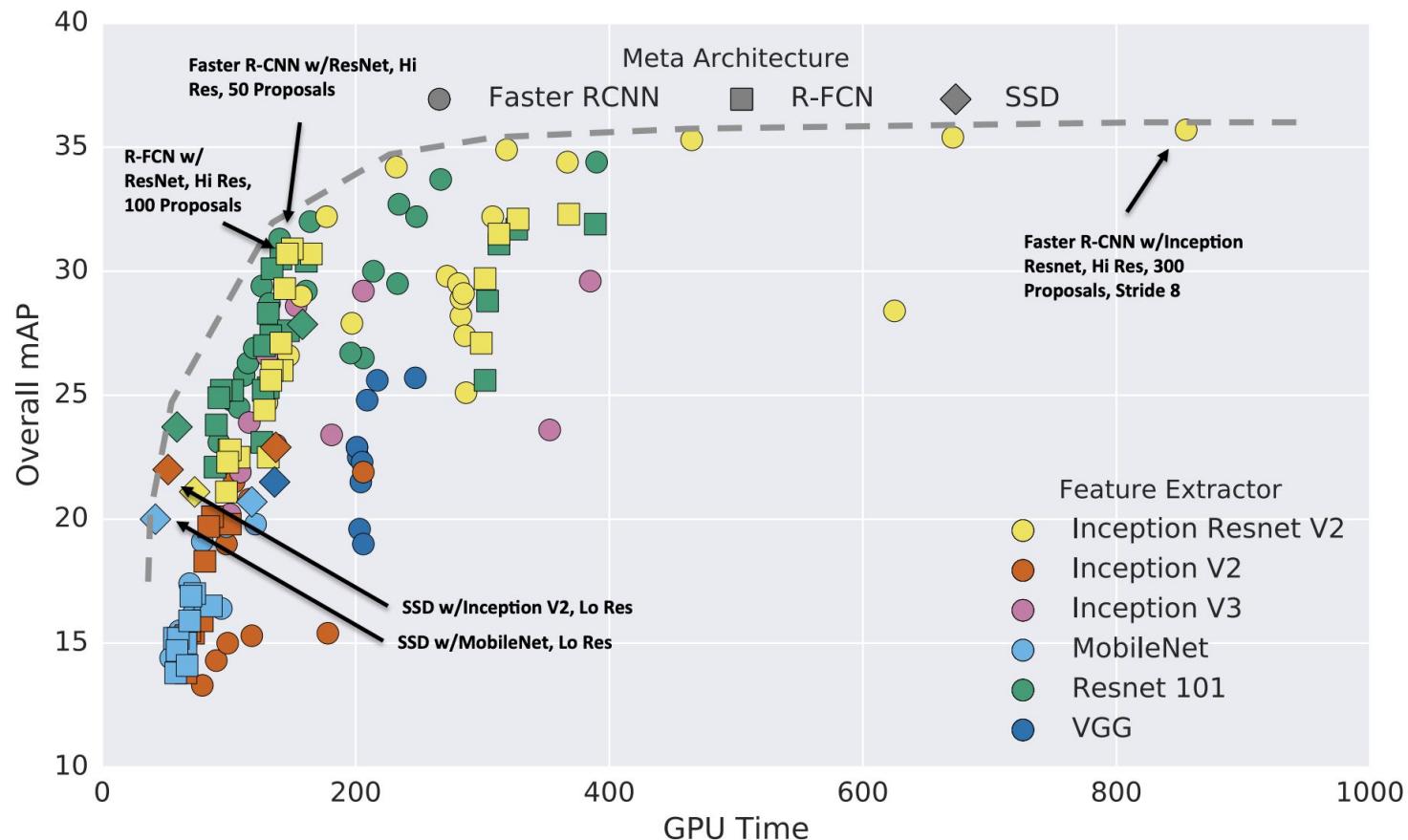
Integrated Collection of Models:

- Bounding boxes
- Segmentation (Instance + Stuff)
- Keypoint
- Dense pose

Facebook AI Research  
PyTorch



Facebook (2019), <https://github.com/facebookresearch/detectron2>



# Other Datasets

- <http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html>
- <https://visualgenome.org>
- <http://ufldl.stanford.edu/housenumbers/>
- <https://storage.googleapis.com/openimages/web/index.html>
- <http://places2.csail.mit.edu>
- <https://www.yf.io/p/lsun>
- <https://www.lvisdataset.org/>

# Face Verification

## Problem

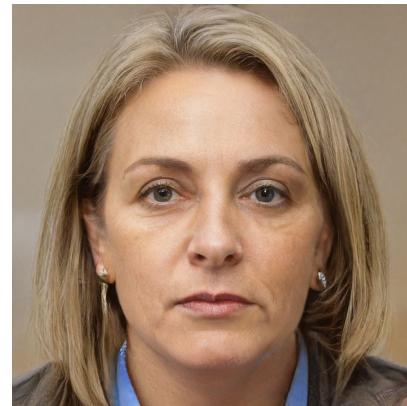
Verify whether a previously unseen photo contains the face of a particular person.

## One Shot Learning

- 1)** None of the test persons appear in the training dataset.
- 2)** Only one known photo per person in the testing dataset.



Known Photo of Person A



Is this a photo of Person A?

# Face Recognition



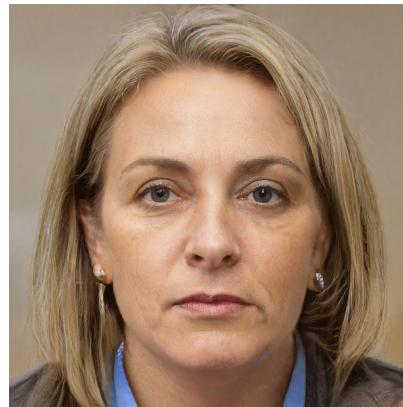
Known Persons

## Problem

Identify a person on a previously unseen photo from a list of persons with known photos.

## One Shot Learning

- 1) None of the test persons appear in the training dataset.
- 2) Only one known photo per person in the testing dataset.



Which known person is this photo of, if any?

# AWS Rekognition

## Product

~ \$0.001 / image

Available as a convenient API

## Features

Object/scene detection

Face recognition and analysis

Both photo and video input

Image to text



# Demo

AWS Rekognition

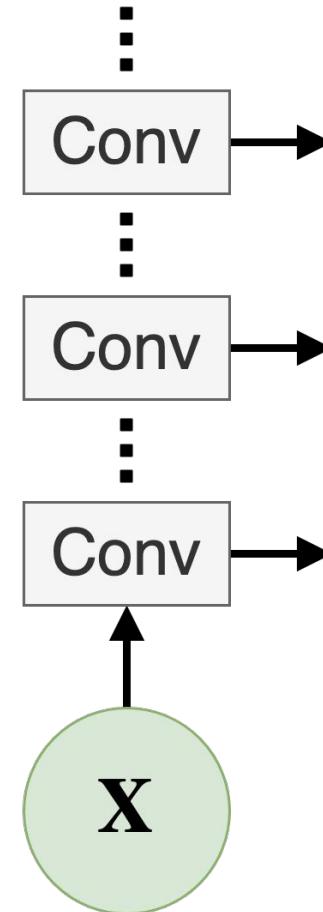
# Visualizing Deep Activations

## Problem

How to gain intuition about behavior of convolutional filters beyond the first layer?

## Solution

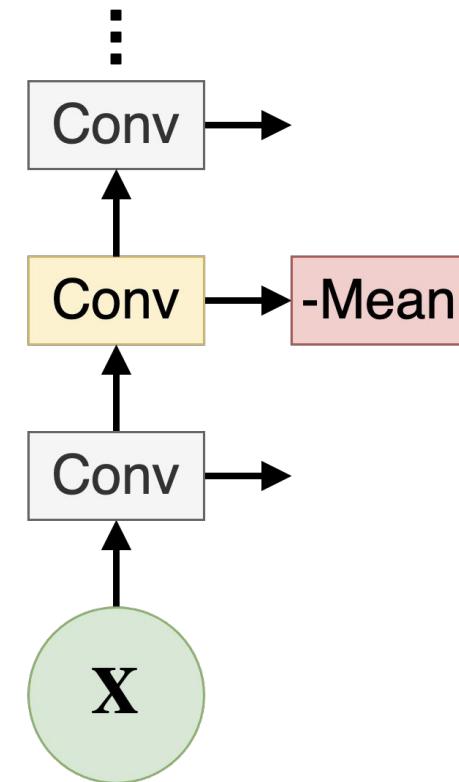
Feed an image through the network and plot activation maps of the deep convolutional layers.



# Optimizing Image for Deep Activation

In order to understand what a deep filter is looking for:

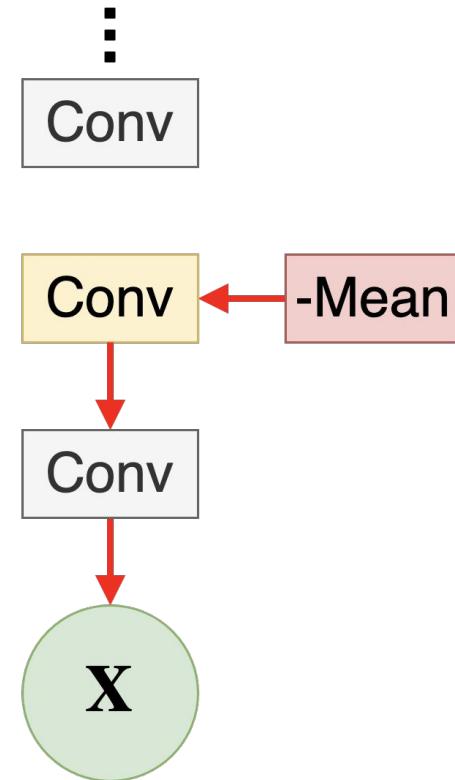
1. Take a pre-trained model, and choose a particular activation map,
2. Maximize activation map by finding gradient with respect to the image's pixels.



# Optimizing Image for Deep Activation

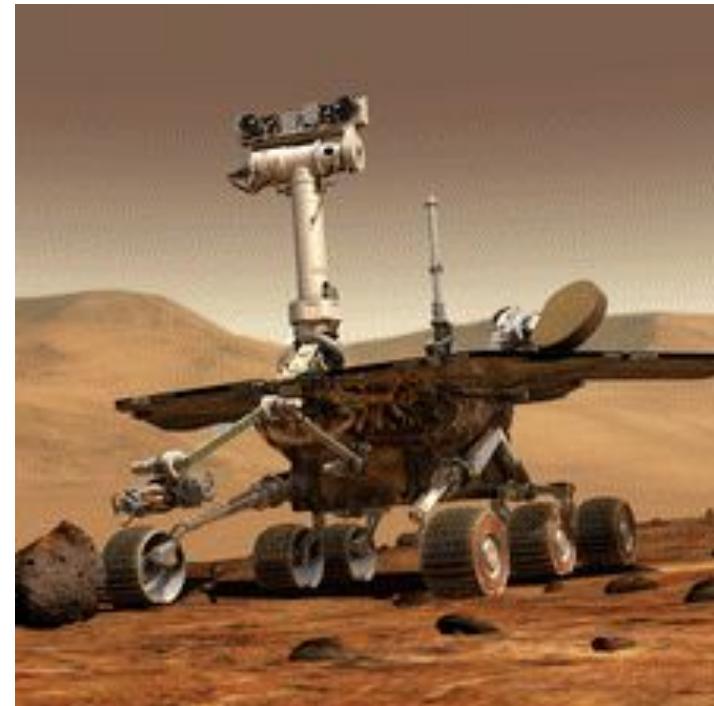
In order to understand what a deep filter is looking for:

1. Take a pre-trained model, and choose a particular activation map,
2. Maximize activation map by finding gradient with respect to the image's pixels.

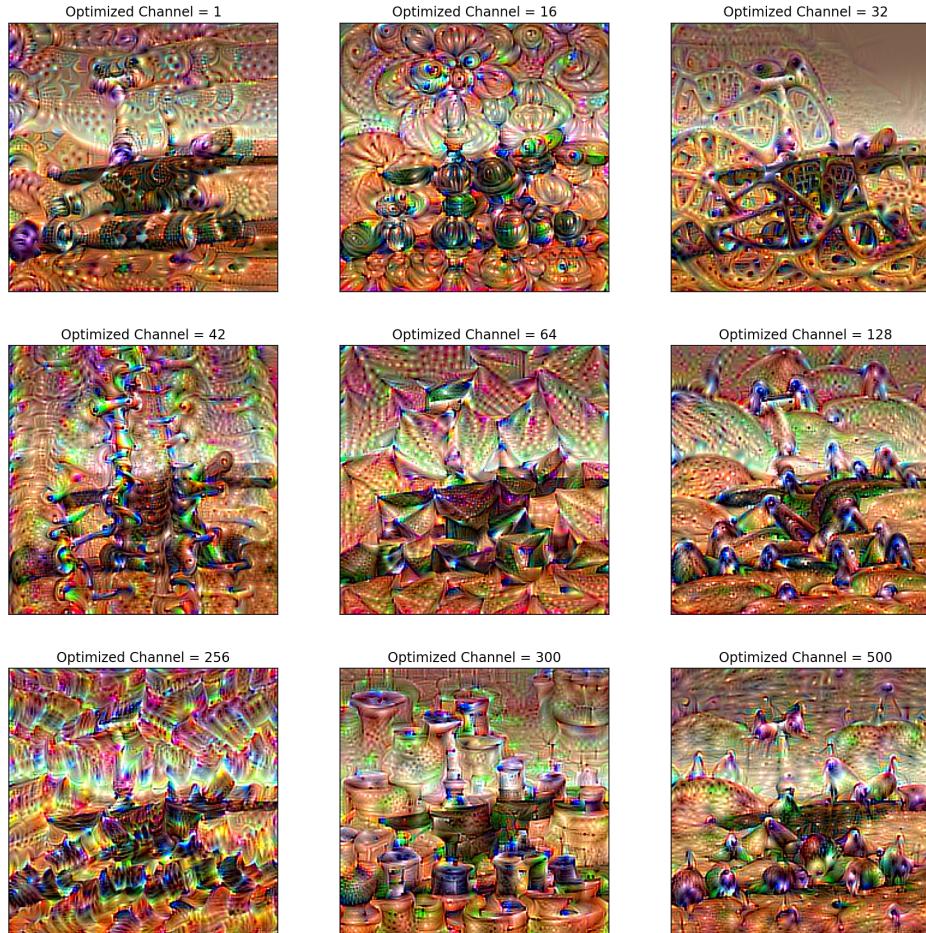


# Deep Dream with VGG16

1. Load pre-trained VGG16
  2. Layer block5\_conv2
  3. Channel 42
  4. Average Activation across H/W
- 
1. Setup Gradient Descent to **find** an image that maximizes the average activation.
  2. Start with the rover image
  3. Iterate for 50 epochs



# Deep Dream with VGG16



# References on CNN Visualizations

- Visualizing and Understanding Convolutional Networks
  - (Zeiler, M, and Fergus, R., 2013)
- <https://github.com/yosinski/deep-visualization-toolbox>
- <https://distill.pub/2017/feature-visualization/>
- <https://distill.pub/2019/activation-atlas/>

# References

1. A Survey on Deep Transfer Learning (Tan, C., et al., 2018)
2. **Deep Learning (Goodfellow, I., Bengio, Y, and Courville, A., 2016)**
3. Deep Residual Learning for Image Recognition (He, K., et al., 2015)
4. DensePose: Dense Human Pose Estimation In The Wild (Riza, A., G., et al., 2018)
5. Do Better ImageNet Models Transfer Better? (Kornblith, S., et al., 2018)
6. Going deeper with convolutions (Szegedy, C., et al., 2014)
7. ImageNet: A Large-Scale Hierarchical Image Database (Deng, J., et al., 2009)
8. ImageNet Classification with Deep Convolutional Neural Networks (Krizhevsky, A., et al., 2012)
9. Speed/accuracy trade-offs for modern convolutional object detectors (Huang, J., et al., 2017)
10. EfficientNetV2: Smaller Models and Faster Training (Tan, M., et al., 2021)
11. You Only Look Once: Unified, Real-Time Object Detection (Redmon, J., et al., 2016)

# Thanks

+ Q&A Time



(Image by a Stable Diffusion model)