Week 06

Advanced Convolutional Neural Networks

Before 2012

- Some **milestones** before 2012
  - Backprop was proposed in 1986
  - CNN was proposed in 1995
- People **did not believe** that NNs work
  - Could not train NNs for real-world datasets
  - Easy to get overfitting
- In machine learning (computer vision), **hand-crafted feature approach** dominated
  - SIFT (scale-invariant feature transform), SURF (speeded up robust features)

After 2012

- **AlexNet** won **ImageNet competition**
- **Missing Ingredient:** Data
  - In 2009, **the ImageNet dataset** was released, challenging researchers to learn models **from 1 million examples**, **1000 each** from **1000 distinct categories** of objects
- **Missing Ingredient:** Hardware
  - **Graphical processing units (GPUs)** proved to be a game changer in making deep learning feasible.
  - These chips had long been developed to **optimize** for high throughput **4×4 matrix-vector products**, which are needed for many computer graphics tasks and **calculate convolutional layers**
- The **renaissance** of Deep Learning

After 2023

- **ChatGPT** introduced
- **Missing Ingredient:** Prompting
- **Missing Ingredient:** Code fine tuning
- The vibe coding
  - Software Engineering
  - Medicine
  - Arts
  - Security
  - Reasoning
- **What's next?**
  - AI vs HI
  - HI
    - From scratch?
    - Non-vibe, slowdown

Vibe coding: code first, refine late

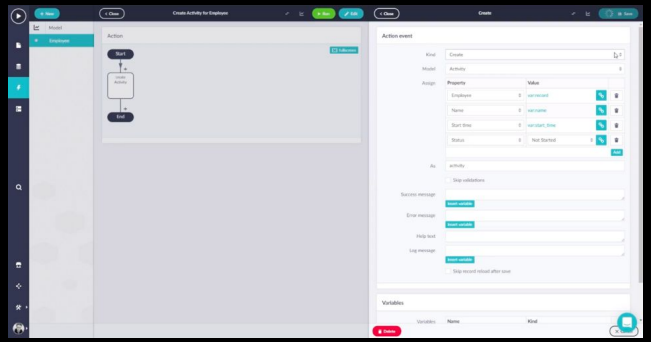
Technical complexity

Code quality and performance issues

Debugging challenges

Maintenance and updates

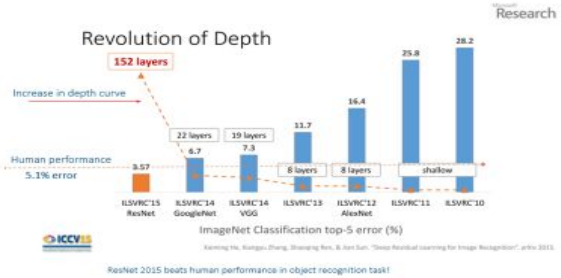
Security concerns



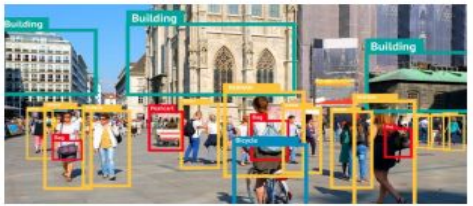
Nocoding 2010

Deep Learning: Good

Achievements of CNNs



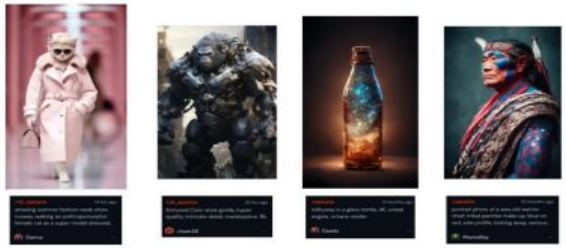
Win human in object recognition on ImageNet dataset



SOTA in object detection and semantic segmentation

Some fiction and imaginary tasks

Image captioning



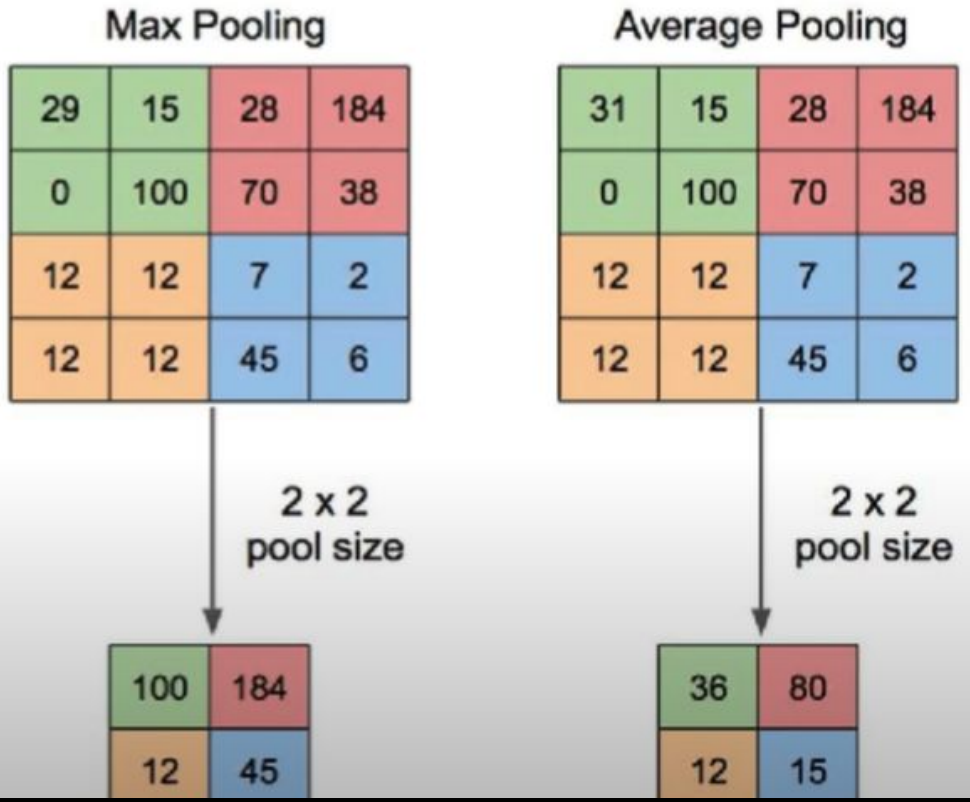
Generative AI
DALL-E/Mid-Journey

Tesla self-driving cars

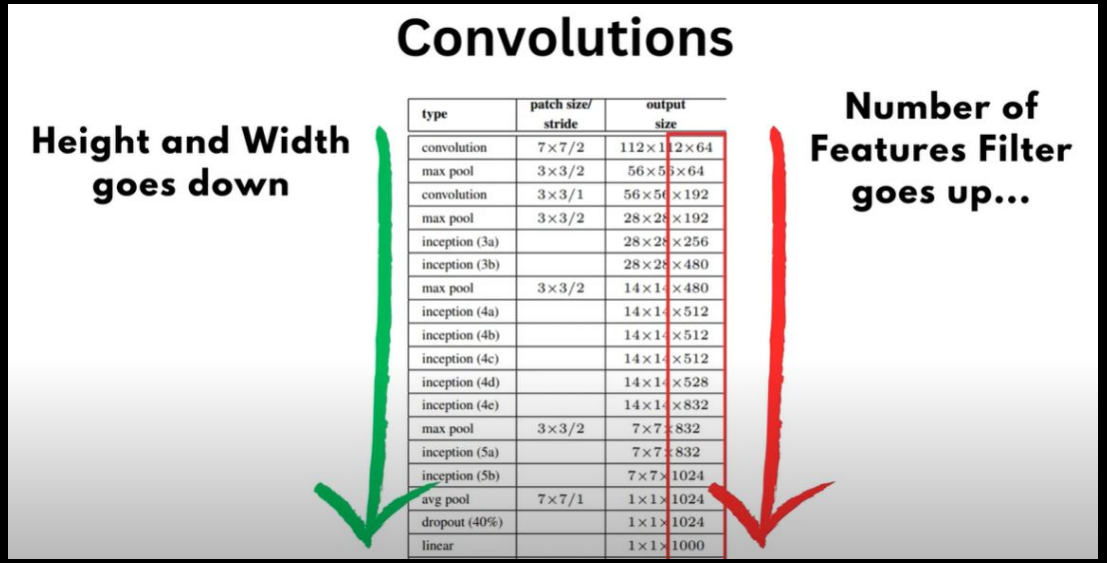
Q & A

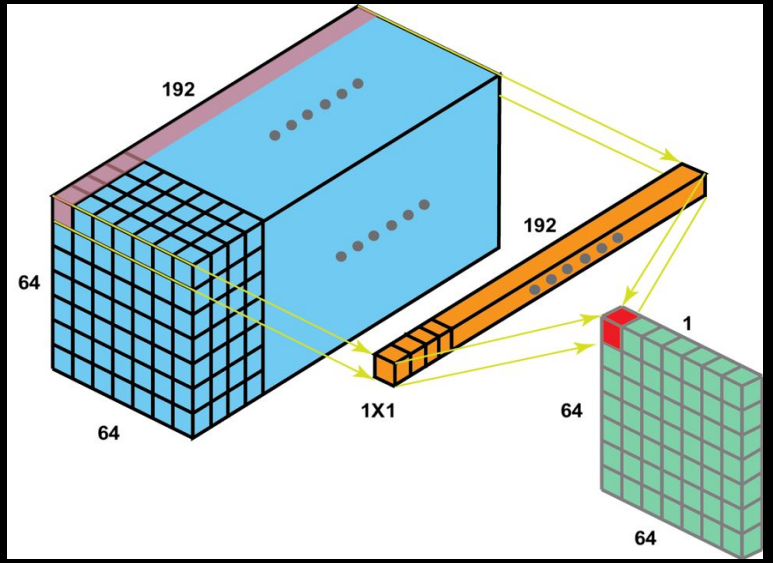
Conv and pooling - high dimension problem

type	patch size/ stride	output size
convolution	7×7/2	112×112×64
max pool	3×3/2	56×56×64
convolution	3×3/1	56×56×192
max pool	3×3/2	28×28×192
inception (3a)		28×28×256
inception (3b)		28×28×480
max pool	3×3/2	14×14×480
inception (4a)		14×14×512
inception (4b)		14×14×512
inception (4c)		14×14×512
inception (4d)		14×14×528
inception (4e)		14×14×832
max pool	3×3/2	7×7×832
inception (5a)		7×7×832
inception (5b)		7×7×1024
avg pool	7×7/1	1×1×1024
dropout (40%)		1×1×1024
linear		1×1×1000

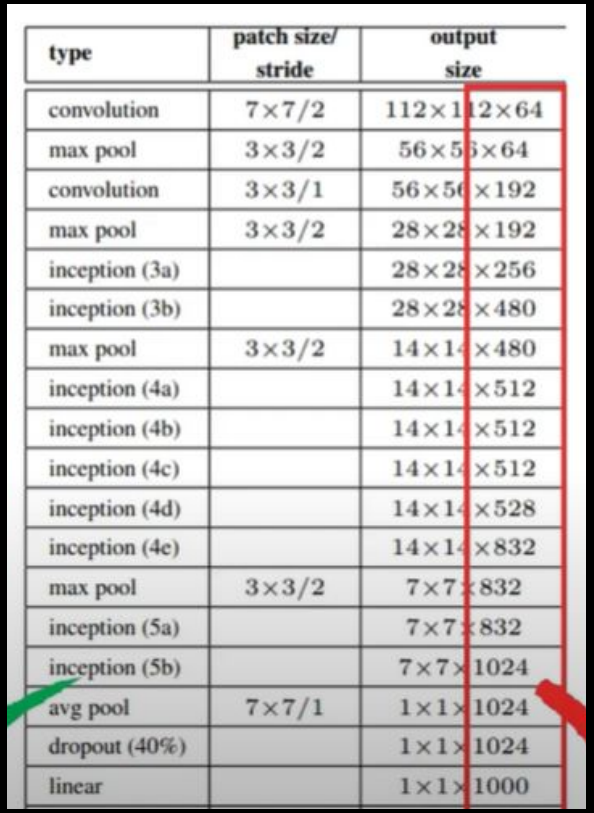


1X1 convolution - filter summarization

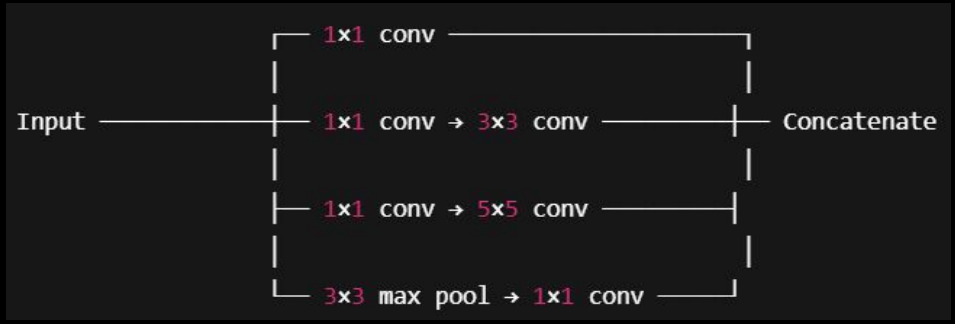




GoogleNet - Inception



type	patch size/ stride	output size
convolution	7×7/2	112×112×64
max pool	3×3/2	56×56×64
convolution	3×3/1	56×56×192
max pool	3×3/2	28×28×192
inception (3a)		28×28×256
inception (3b)		28×28×480
max pool	3×3/2	14×14×480
inception (4a)		14×14×512
inception (4b)		14×14×512
inception (4c)		14×14×512
inception (4d)		14×14×528
inception (4e)		14×14×832
max pool	3×3/2	7×7×832
inception (5a)		7×7×832
inception (5b)		7×7×1024
avg pool	7×7/1	1×1×1024
dropout (40%)		1×1×1024
linear		1×1×1000



1×1 conv: cheap, reduces channel dimensions (bottleneck).

3×3 conv: medium receptive field.

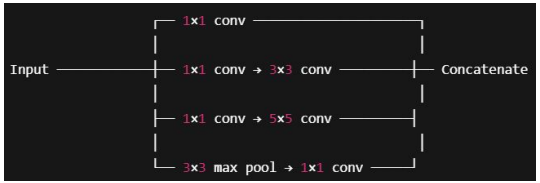
5×5 conv: larger receptive field.

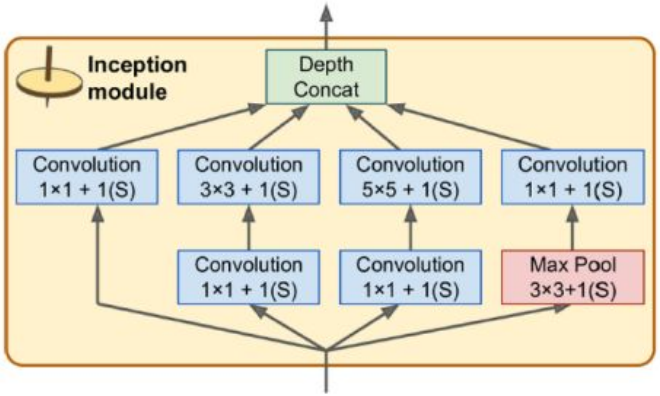
Pooling branch: captures context, then projected back with 1×1 conv.

GoogleNet - Inception

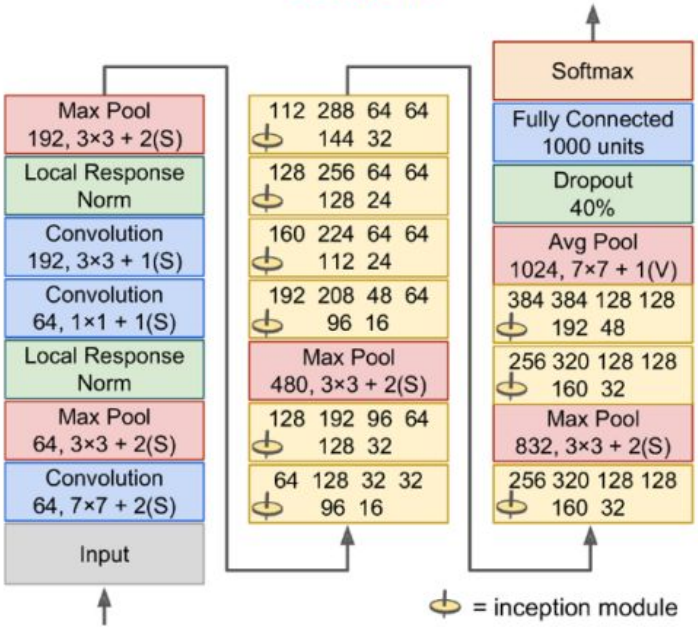
GoogleNet

Szegedy et al. '14





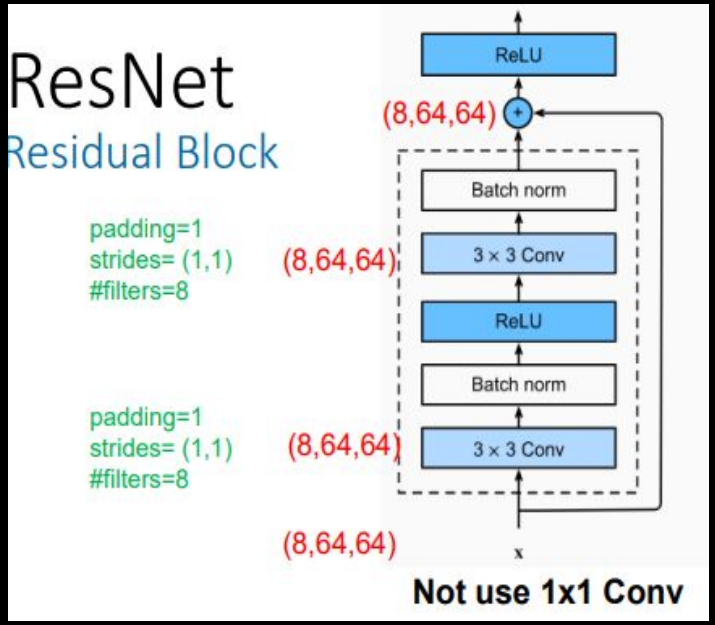
22 layers



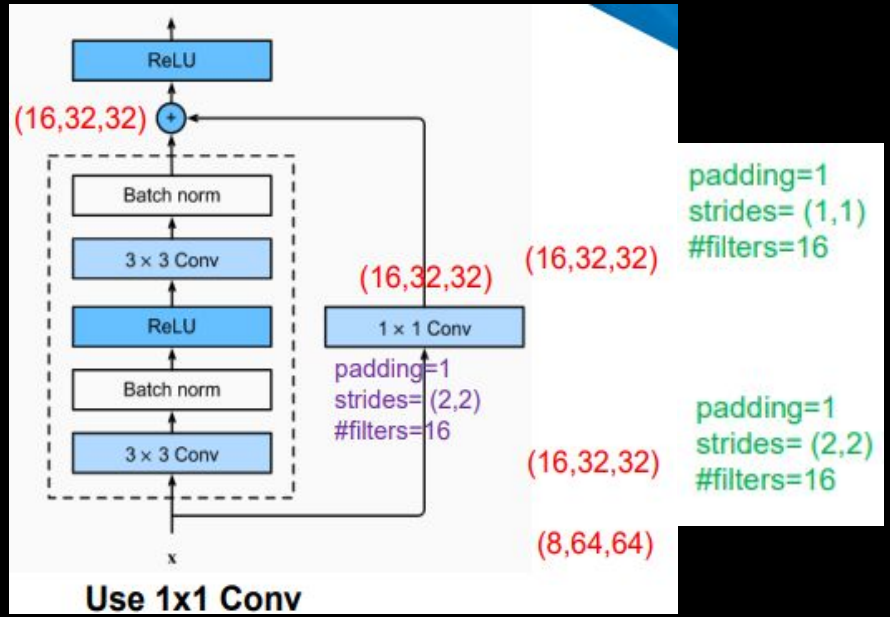
(Source: Hands On, Ch. 15)

ResNet

Residual Block



Not use 1x1 Conv



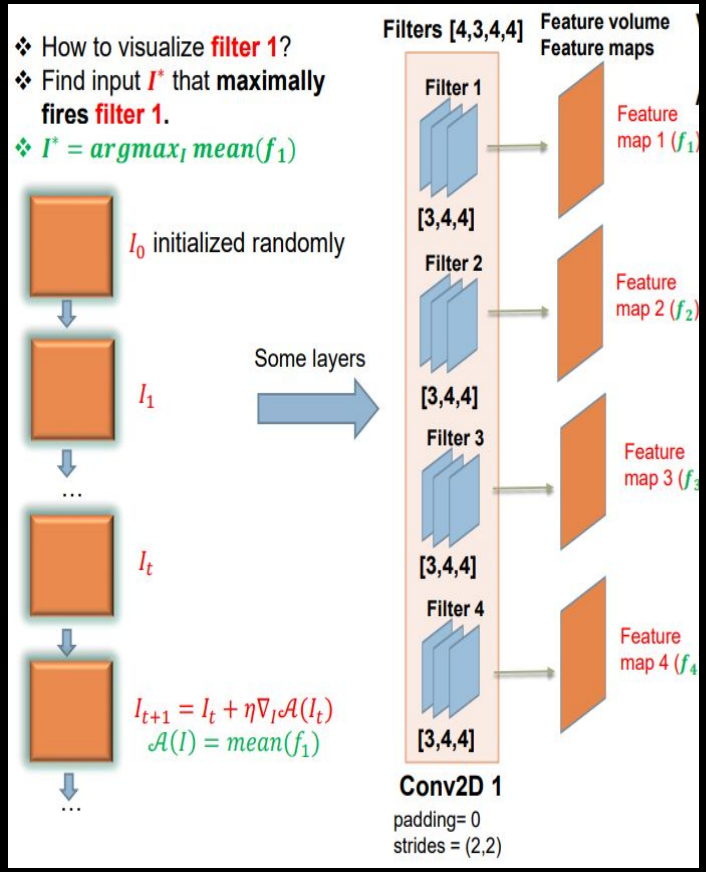
Use 1x1 Conv

Learn $f(x) = g(x) + x$ where $g(x) = f(x) - x$

- The **model expressiveness** is the **same** as previous
- The **gradient** $\nabla f(x) = \nabla g(x) + \mathbf{1}$ **looks better** where $\mathbf{1}$ is the vector of **all** 1 with the same shape as x

Q & A

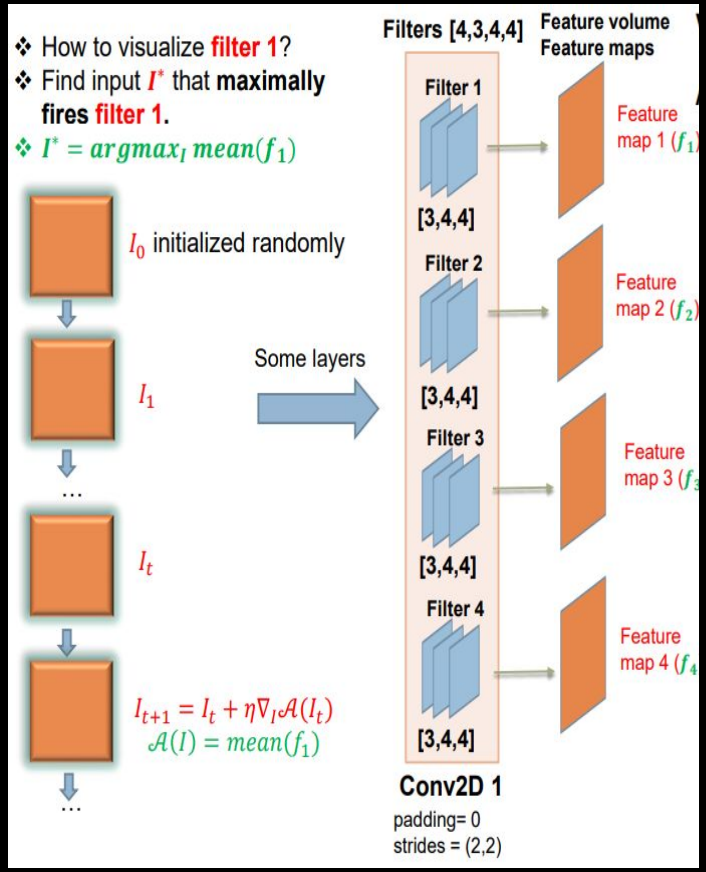
Feature Visualization




```python
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.models as models
import torchvision.transforms as transforms
import matplotlib.pyplot as plt

# ----------------------
# 1. Load Pretrained Model
# ----------------------
model = models.alexnet(pretrained=True)  # You can also try resnet18, vgg16, etc.
model.eval()

# Choose a convolutional layer & filter index to visualize
target_layer = model.features[0]  # first conv layer
filter_idx = 5                    # visualize filter #5
```

```python
# ----------------------
# 2. Create Random Image
# ----------------------
img = torch.randn(1, 3, 64, 64, requires_grad=True)  # input: [1,3,64,64]
```

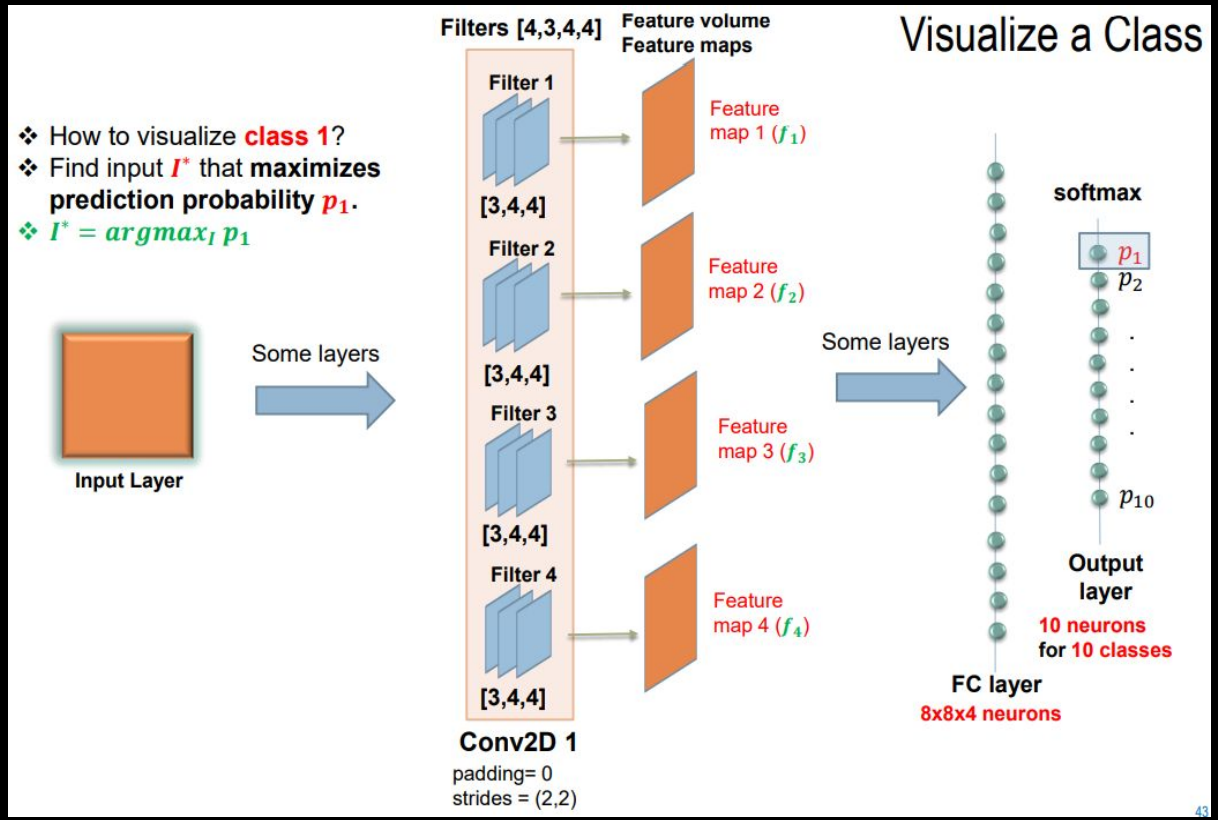
Feature Visualization



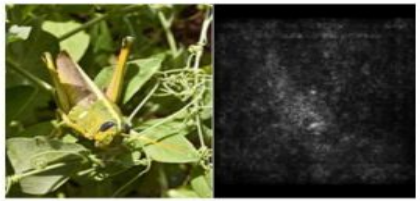

```python
# Choose target convolutional layer & filter index
conv_layer = model.features[0]    # first conv layer
filter_idx = 5                    # visualize filter #5
```

```python
# 4. Gradient Ascent Loop
# -----------------------
for step in range(50):
    optimizer.zero_grad()

    # Forward pass only through the target conv layer
    activation = conv_layer(img)  # shape [1, num_filters, H, W]

    # Select chosen filter's mean activation
    loss = -activation[0, filter_idx].mean()

    # Backward (gradient ascent on input image)
    loss.backward()
    optimizer.step()

    # Keep pixel values bounded
    img.data = img.data.clamp(-1.5, 1.5)

    if step % 10 == 0:
        print(f"Step {step}, Loss: {-loss.item():.4f}")
```

Class Visualization



Class Visualization



Salience/Heat map

Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps

Karen Simonyan Andrea Vedaldi Andrew Zisserman
Visual Geometry Group, University of Oxford
{karen,vedaldi,az}@robots.ox.ac.uk

Link: https://arxiv.org/pdf/1312.6034

Which **part** of images **mostly contribute** to the **model prediction**?

GradCam Model Explainability

$p = [p_1, \dots, p_{\hat{y}}, \dots, p_{1000}]$

$\hat{y}$ is a **predicted truth label**

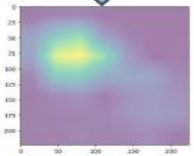
A layer with **3D tensor** feature maps [100,7,7]

$A = [A_1, A_2, \dots, A_{100}]$ with shape [100,7,7]

A_i with shape [7,7] is a feature map.

$heatmap = \sum_{i=1}^{1000} sum\left(\frac{\partial p_{\hat{y}}}{\partial A_i}\right) \times A_i$ has shape [7,7]

Normalize *heatmap* and **resize** to [224, 224]



Salience/Heat map

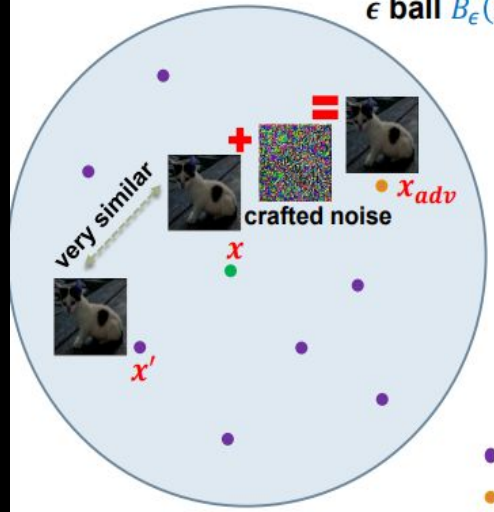
Grad-CAM:
Visual Explanations from Deep Networks via Gradient-based Localization

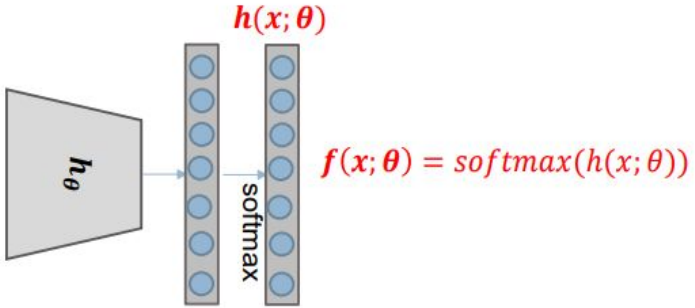
Ramprasaath R. Selvaraju[1]* Michael Cogswell[1] Abhishek Das[1] Ramakrishna Vedantam[1]*
Devi Parikh[1,2] Dhruv Batra[1,2]
[1]Georgia Institute of Technology [2]Facebook AI Research
{ramprs, cogswell, abhshkdz, vrama, parikh, dbatra}@gatech.edu

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

Adversarial examples

ϵ **ball** $B_\epsilon(x) = \{x' : \|x' - x\| \leq \epsilon\}$ with a **small** $\epsilon > 0$.





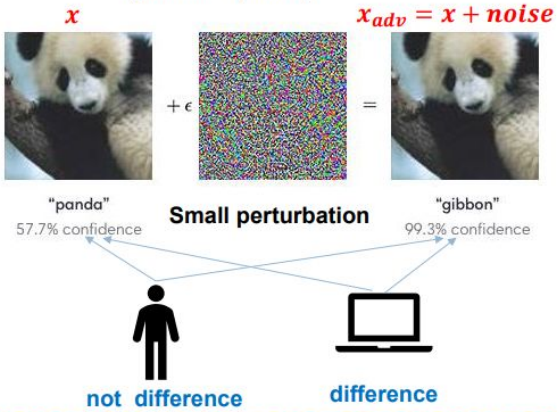
$$f(x;\theta) = softmax(h(x;\theta))$$

$$argmax_{1\leq i\leq M} f_i(x;\theta) = \hat{y} \neq \hat{y}_{adv} = argmax_{1\leq i\leq M} f_i(x_{adv};\theta)$$

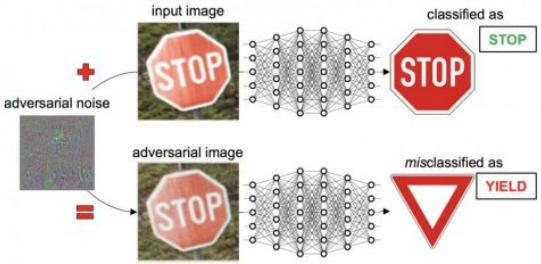
- **Good** example which has **same predicted label** as x.
- **Adversarial** example which has **different predicted label** as x.

- The **true sense** of a **robust model**
  - If x' and x are **close**, the **predictions** of x' and x must be the **same**.
  - If $x' \in B_\epsilon(x)$ (i.e., $\|x' - x\| \leq \epsilon$), $argmax_{1\leq i\leq M} f_i(x;\theta) = \hat{y} = \hat{y}' = argmax_{1\leq i\leq M} f_i(x';\theta)$.
- Unfortunately, we can **easily** find **adversarial examples** $x_{adv} \in B_\epsilon(x)$ that can **fool** the classifier
  - $argmax_{1\leq i\leq M} f_i(x;\theta) = \hat{y} \neq \hat{y}_{adv} = argmax_{1\leq i\leq M} f_i(x_{adv};\theta)$

(Source: OpenAI)



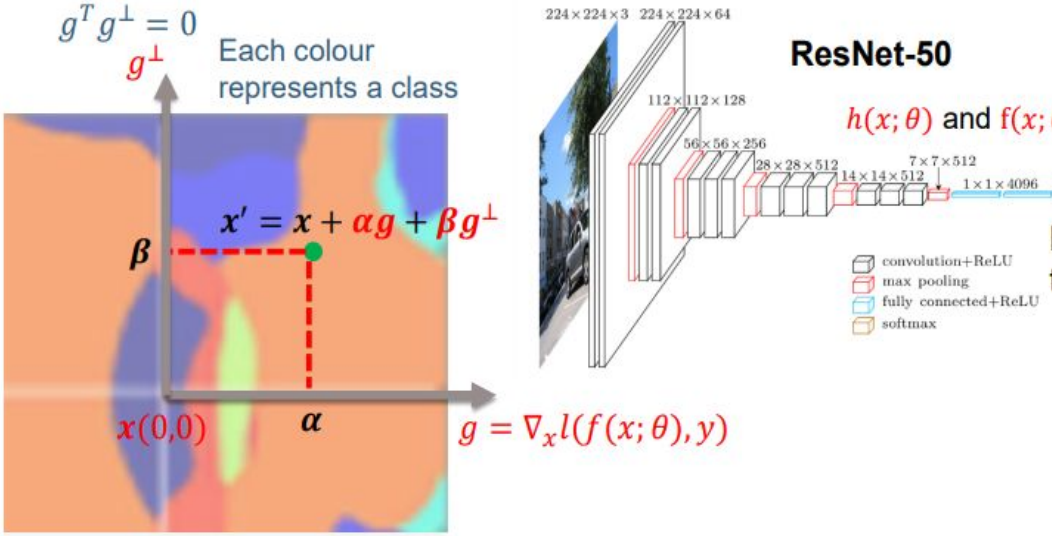
(Source: Internet)



Misleading autonomous car driving system

- Adversarial example x_{adv} of a clean data example x w.r.t a model $f(.;\theta)$
  - $d(x_{adv}, x) = \|x_{adv} - x\| \leq \epsilon$ where $\|.\|$ is a norm (e.g., $\|.\|_1, \|.\|_2, \|.\|_\infty$)
  - f predicts x and x_{adv} with **different labels**, i.e., $argmax_i f_i(x;\theta) \neq argmax_i f_i(x_{adv};\theta)$

Visualization of decision regions



ResNet-50

$h(x;\theta)$ and $f(x;\theta) = softmax(h(x;\theta))$

Predicts $x' = x + \alpha g + \beta g^\perp$ to **decide** the **colour**

Following the **gradient direction** is easy to **get out the blue region**.

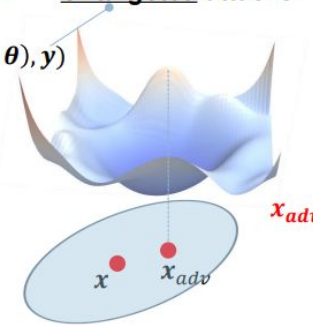
- Start from $x = x + 0g + 0g^\perp$ (coordinate $(\mathbf{0}, \mathbf{0})$), if we **go along with** the **gradient direction** $g = \nabla_x l(f(x;\theta), y)$, we can reach $x_{adv} = x + \alpha g + 0g^\perp$ such that
  - x_{adv} is very close to x → human cannot tell how they **are different** → the **same class** from human **perceptron**
  - x_{adv} is predicted to **different class** from x by the classifier
  - x_{adv} is called **adversarial example** of x

Adversarial attack

How to run untargeted attack?

Untargeted **attacks**

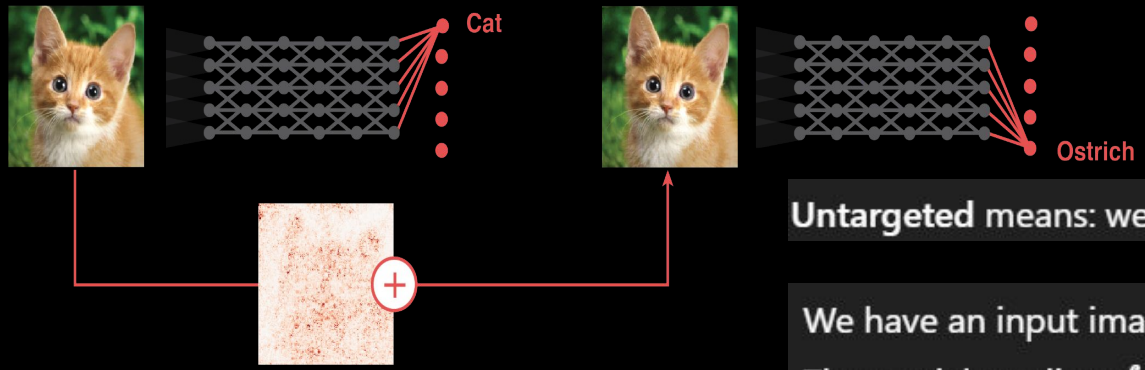
Loss surface $l(f(x';\theta), y)$



$B_\epsilon(x) = \{x' : \|x' - x\| \leq \epsilon\}$

x has true label $y \in \{1, 2, \ldots, M\}$

$x_{adv} = argmax_{x' \in B_\epsilon(x)} l(f(x';\theta), y)$



Untargeted means: we don't care *which wrong class* the model predicts

We have an input image x with true label y.

The model predicts $f(x;\theta)$.

Adversarial attack: add a **small perturbation** δ to create $x_{adv} = x + \delta$

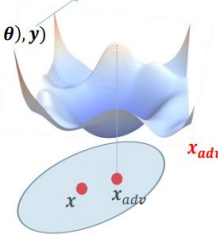
Quiz - what's the difference ?

Adversarial attack

How to run untargeted attack?

***Untargeted* attacks**

Loss surface $l(f(x';\theta), y)$



$B_\epsilon(x) = \{x' : \|x' - x\| \le \epsilon\}$

x has true label $y \in \{1, 2, \dots, M\}$

$x_{adv} = argmax_{x' \in B_\epsilon(x)} l(f(x';\theta), y)$

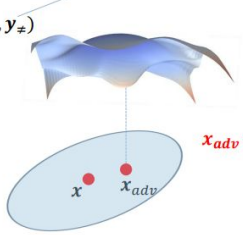
Adversarial attack

How to run targeted attack?

***Targeted* attacks**

Loss surface $l(f(x';\theta), y_{\neq})$

$y_{\neq}$ is different from y



$B_\epsilon(x) = \{x' : \|x' - x\| \le \epsilon\}$

x has true label $y \in \{1, 2, \dots, M\}$

$x_{adv} = argmin_{x' \in B_\epsilon(x)} l(f(x';\theta), y_{\neq})$

- **Projected Gradient Descent (Ascent)** (PGD)
  - $x_0 = x + Uniform([-\epsilon, \epsilon])$
  - $\tilde{x}_{t+1} = x_t + \eta \nabla_x l(f(x_t;\theta), y)$
  - $x_{t+1} = Proj_{B_\epsilon(x)}(\tilde{x}_{t+1})$
  - Run in k steps ($k = 20$), $\eta > 0$ is the learning rate
  - $x_{adv} = x_k$

- **Projected Gradient Descent (Ascent)** (PGD)
  - $x_0 = x + Uniform([-\epsilon, \epsilon])$
  - $\tilde{x}_{t+1} = x_t - \eta \nabla_x l(f(x_t;\theta), y_{\#})$
  - $x_{t+1} = Proj_{B_\epsilon(x)}(\tilde{x}_{t+1})$
  - Run in k steps ($k = 20$), $\eta > 0$ is the learning rate
  - $x_{adv} = x_k$