Deep learning

Dr. Aissa Boulmerka
a.boulmerka@centre-univ-mila.dz

2023-2024

CHAPTER 8
CONVOLUTIONAL NEURAL NETWORKS (CNNS)
“DEEP CONVOLUTIONAL MODELS CASE STUDIES”

Outline

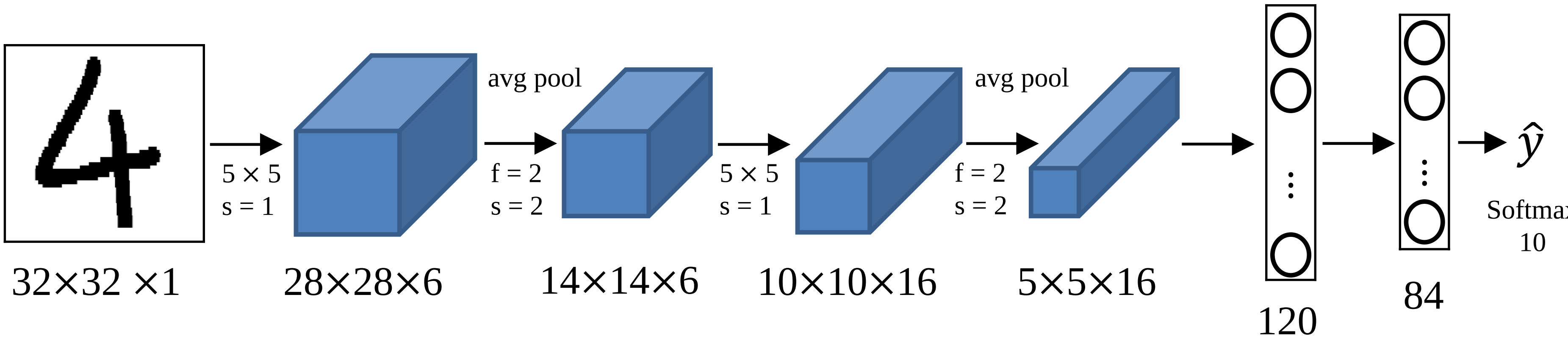
Classic networks:

- LeNet-5
- AlexNet
- VGG

ResNet (152)

Inception

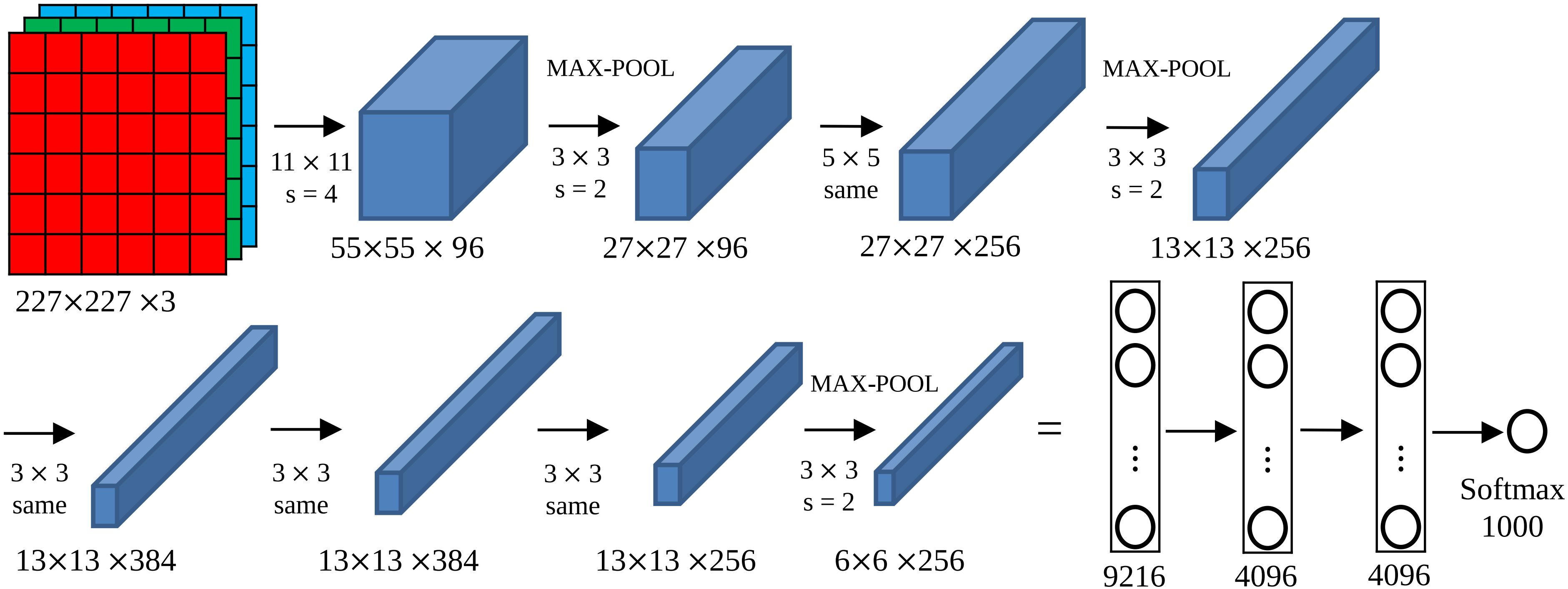
LeNet - 5



- **60K parameters.**
- **CONV-POOL-CONV-POOL-FC-FC-FC-SOFTMAX**
- **Activation: Sigmoid/Tanh   Relu**

[LeCun et al., 1998. Gradient-based learning applied to document recognition]

AlexNet



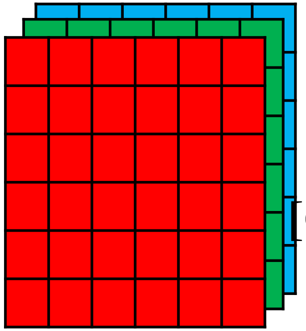
- Similarly to LeNet, but much bigger (60 M parameters).
- ReLU
- Multiple GPUs
- Local Response Normalization (LRN)

[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks]

VGG - 16

CONV = 3×3 filter, s = 1, same

MAX-POOL = 2×2 , s = 2



$224\times224\times3$ $\xrightarrow{\text{[CONV 64]} \times 2}$ $224\times224\times64$ $\xrightarrow{\text{POOL}}$ $112\times112\times64$ $\xrightarrow{\text{[CONV 128]} \times 2}$ $112\times112\times128$ $\xrightarrow{\text{POOL}}$ $56\times56\times128$

$\xrightarrow{\text{[CONV 256]} \times 3}$ $56\times56\times256$ $\xrightarrow{\text{POOL}}$ $28\times28\times256$ $\xrightarrow{\text{[CONV 512]} \times 3}$ $28\times28\times512$ $\xrightarrow{\text{POOL}}$ $14\times14\times512$

$\xrightarrow{\text{[CONV 512]} \times 3}$ $14\times14\times512$ $\xrightarrow{\text{POOL}}$ $7\times7\times512$ $\longrightarrow$ FC 4096 $\longrightarrow$ FC 4096 $\longrightarrow$ Softmax 1000

138 M parameters

[Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition]

Residual Networks (ResNets)

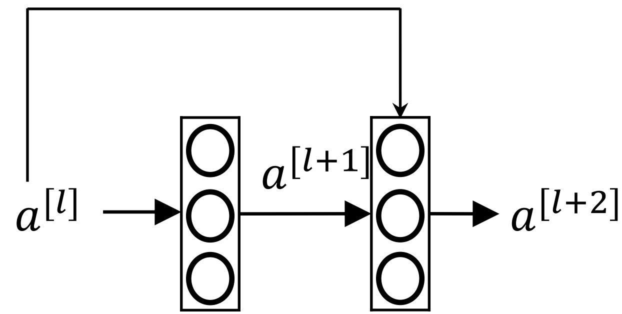
- Very, very deep NNs are difficult to train because of **vanishing and exploding gradients** problems.

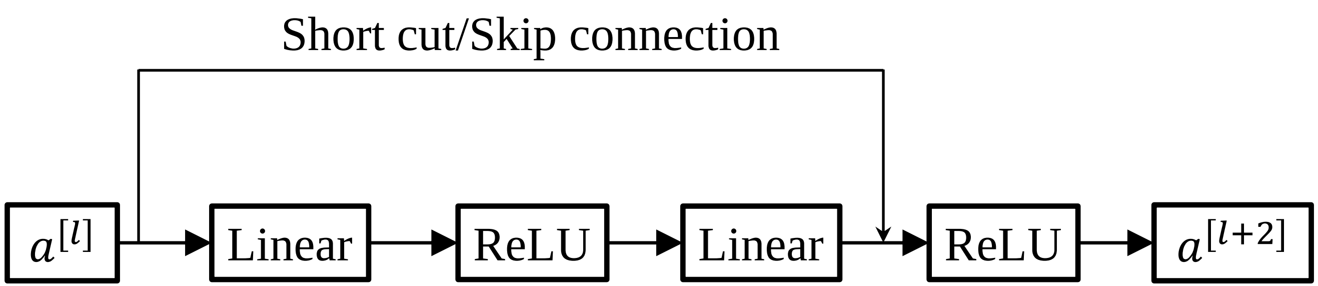
- In this section we will learn about **skip connection** which makes you take the activation from one layer and suddenly feed it to another layer even much deeper in NN which allows you to train large NNs even with **layers greater than 100**.

Residual block

- ResNets are built out of some **Residual blocks**.

Residual block

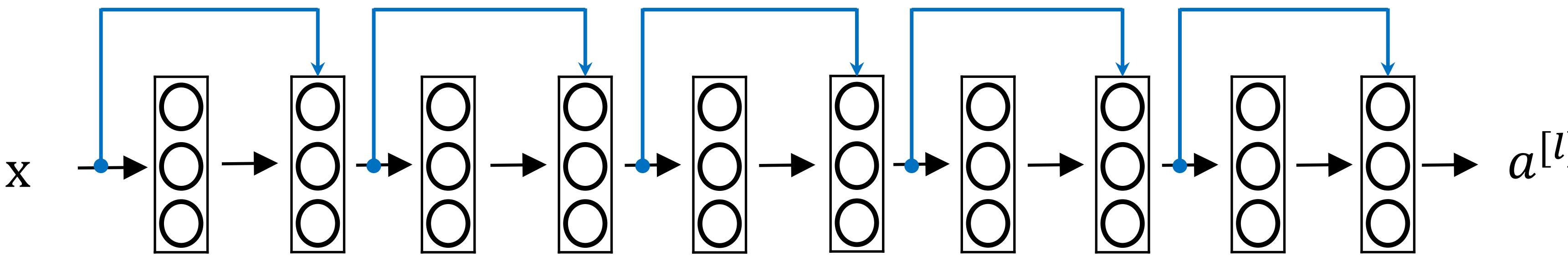




$z^{[l+1]} = W^{[l+1]}\, a^{[l]} + b^{[l+1]}$ $\quad a^{[l+1]} = g(z^{[l+1]})$ $\quad z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$ $\quad a^{[l+2]} = g(z^{[l+2]})$

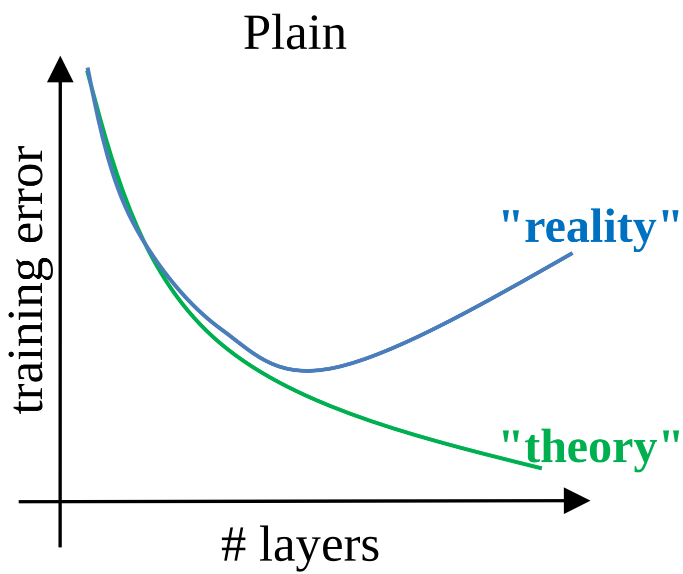
[He et al., 2015. Deep residual networks for image recognition]

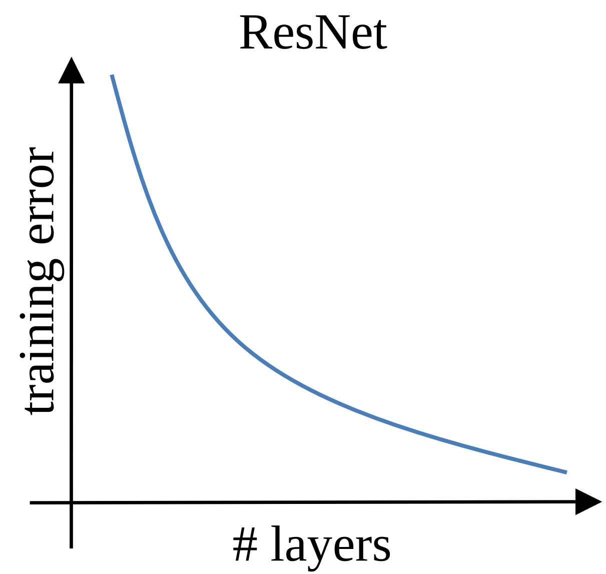
Residual Network



"Plain network"

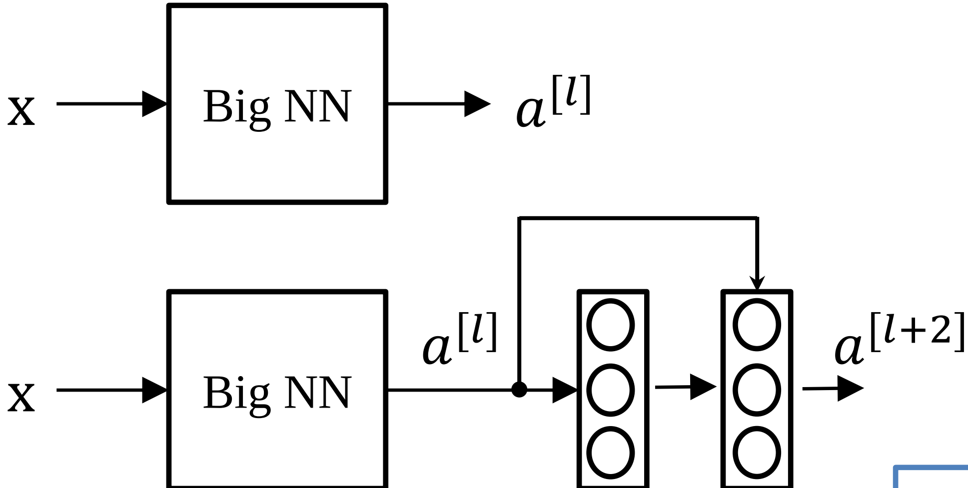
"Residual network"





[He et al., 2015. Deep residual networks for image recognition]

Why do residual networks work?



$ReLU \qquad a \geq 0$

$$a^{[l+2]} = g\left(z^{[l+2]} + a^{[l]}\right)$$
$$= g\left(w^{[l+2]}a^{[l+1]} + b^{[l+2]} + a^{[l]}\right) = g\left(a^{[l]}\right) = a^{[l]}$$

This show that identity function is easy for a residual block to learn, consequently it can train deeper NNs.

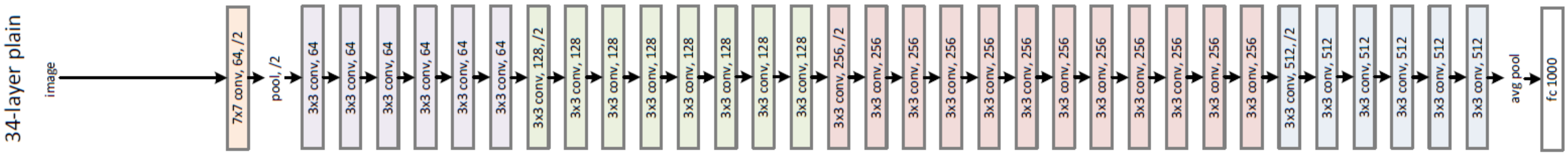
we are using L2 regularization for example, $W^{[l+2]}$ will be zero. Lets say that $b^{[l+2]}$ will be zero too.

Using a skip-connection helps the gradient to backpropagate and thus helps to train deeper networks

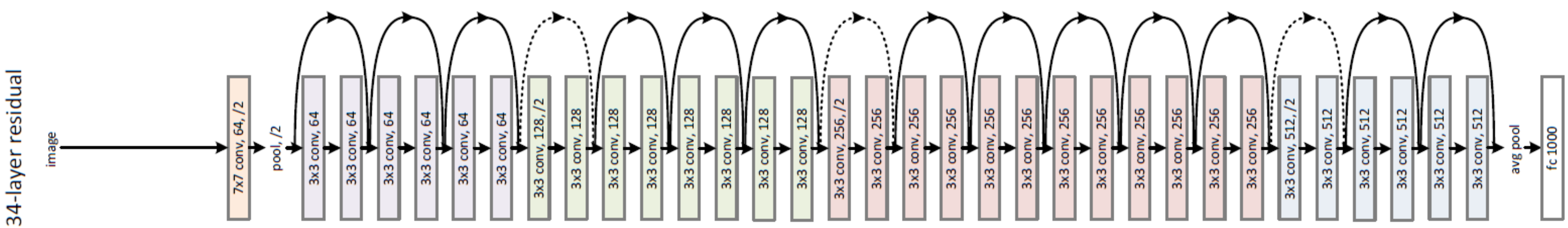
[He et al., 2015. Deep residual networks for image recognition]

ResNet

Plain



ResNet



- All the 3x3 Conv are **same Convs**.
- No **FC layers**, No **dropout** is used.
- The **dotted lines** is the case when the dimensions are different. To solve then they down-sample the input by 2 and then pad zeros to match the two dimensions.

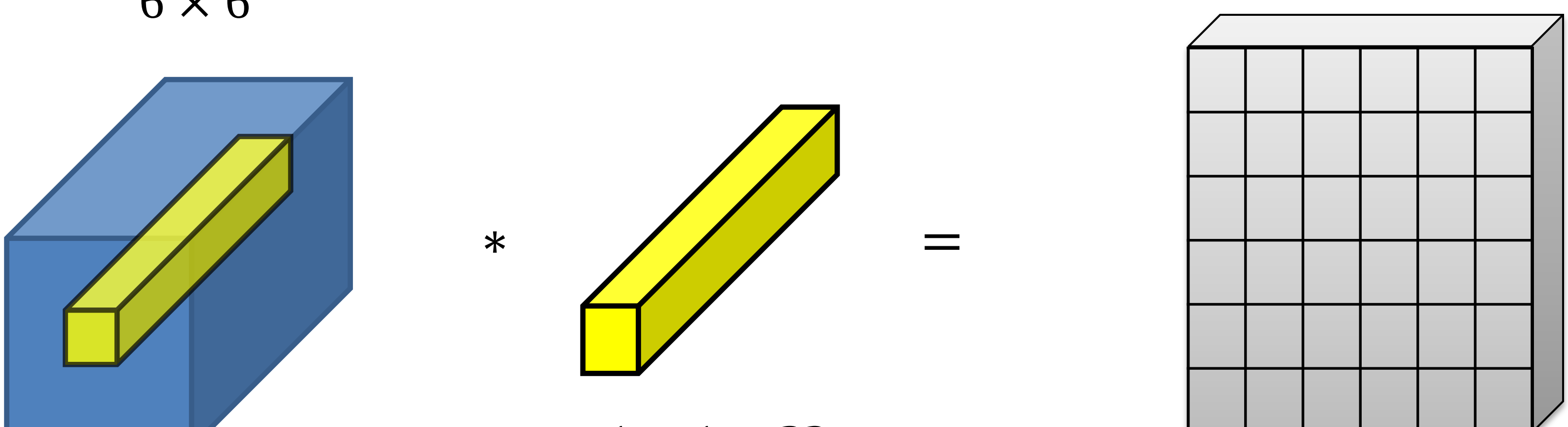
[He et al., 2015. Deep residual networks for image recognition]

Why does a 1×1 convolution do?

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

6×6

$* \quad \boxed{2} \quad =$

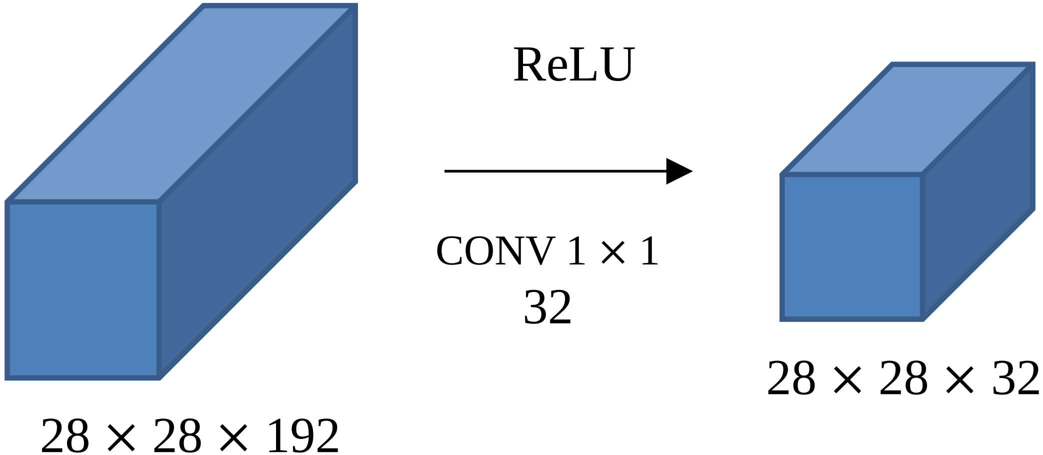


$6 \times 6 \times 32$ $\quad * \quad$ $1 \times 1 \times 32$ $\quad = \quad$ $6 \times 6 \times \#$ filters

32 → # filters $n_c^{[l+1]}$

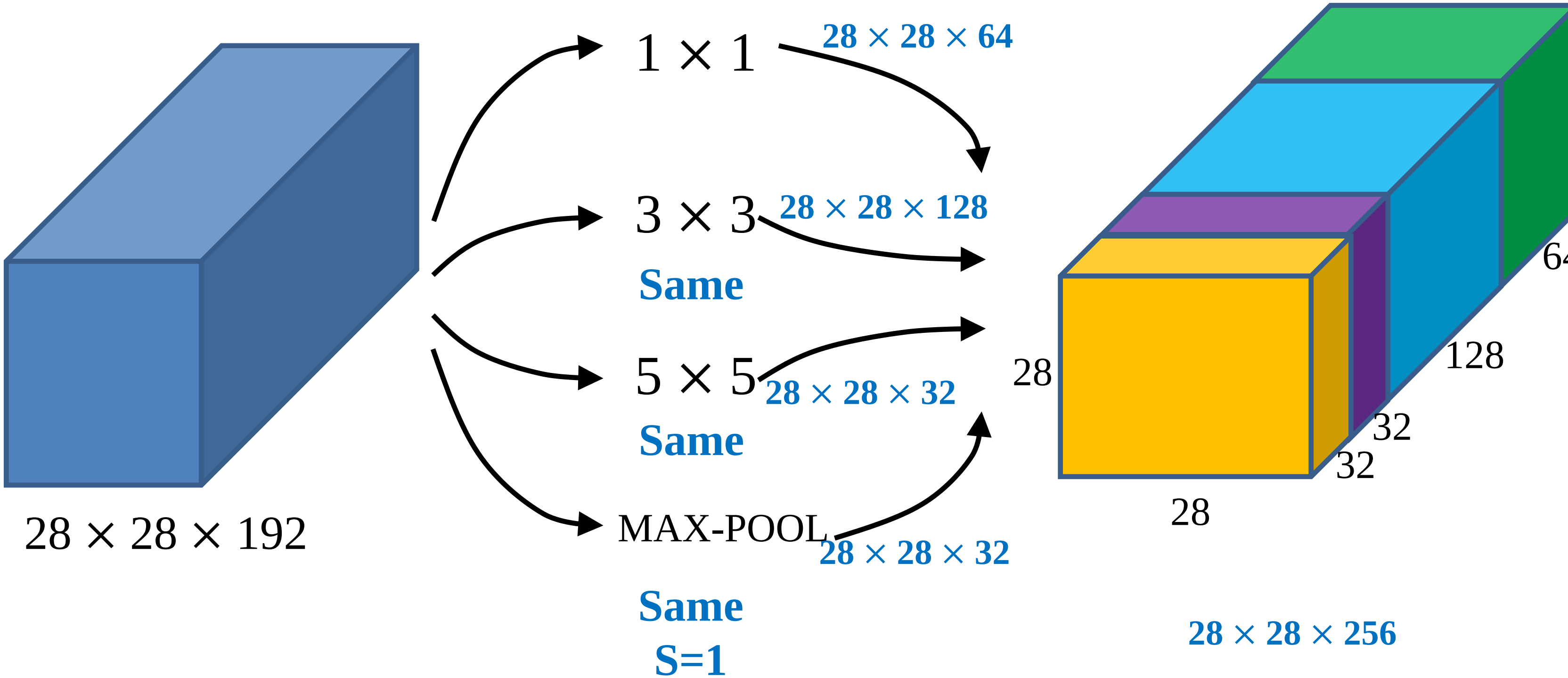
[Lin et al., 2013. Network in network]

Using 1×1 convolutions



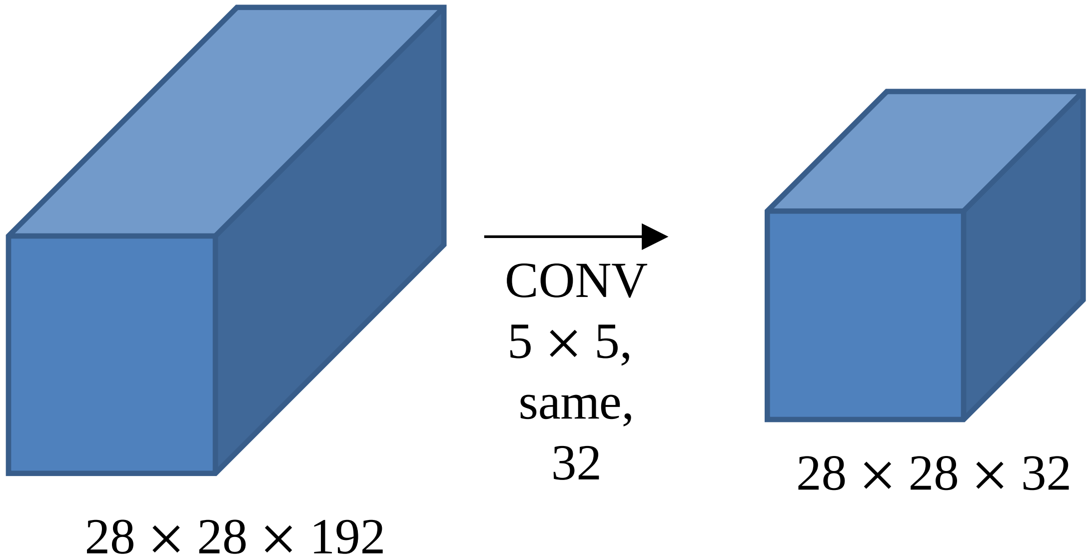
[Lin et al., 2013. Network in network]

Motivation for inception network



[Szegedy et al. 2014. Going deeper with convolutions]

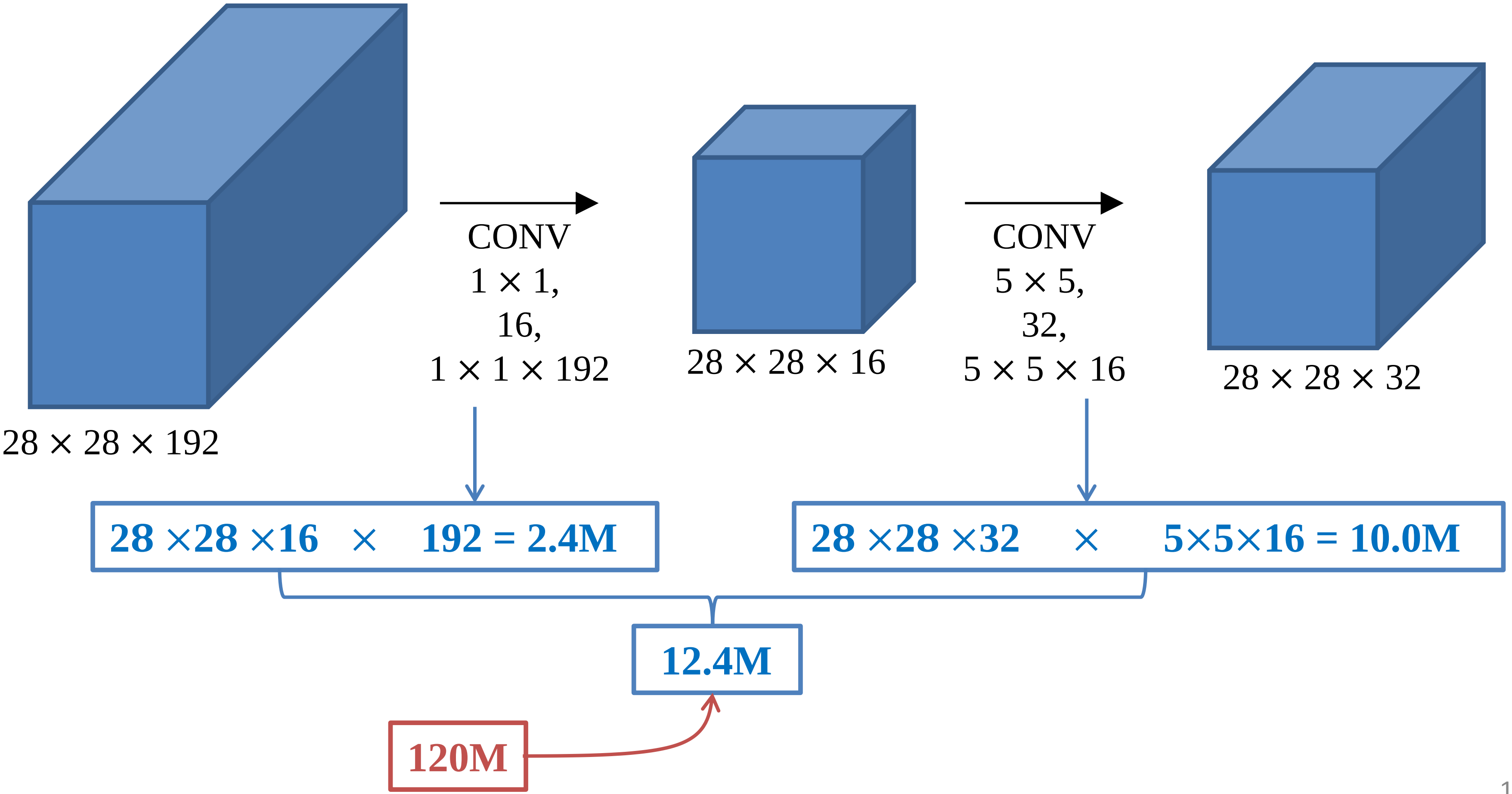
The problem of computational cost



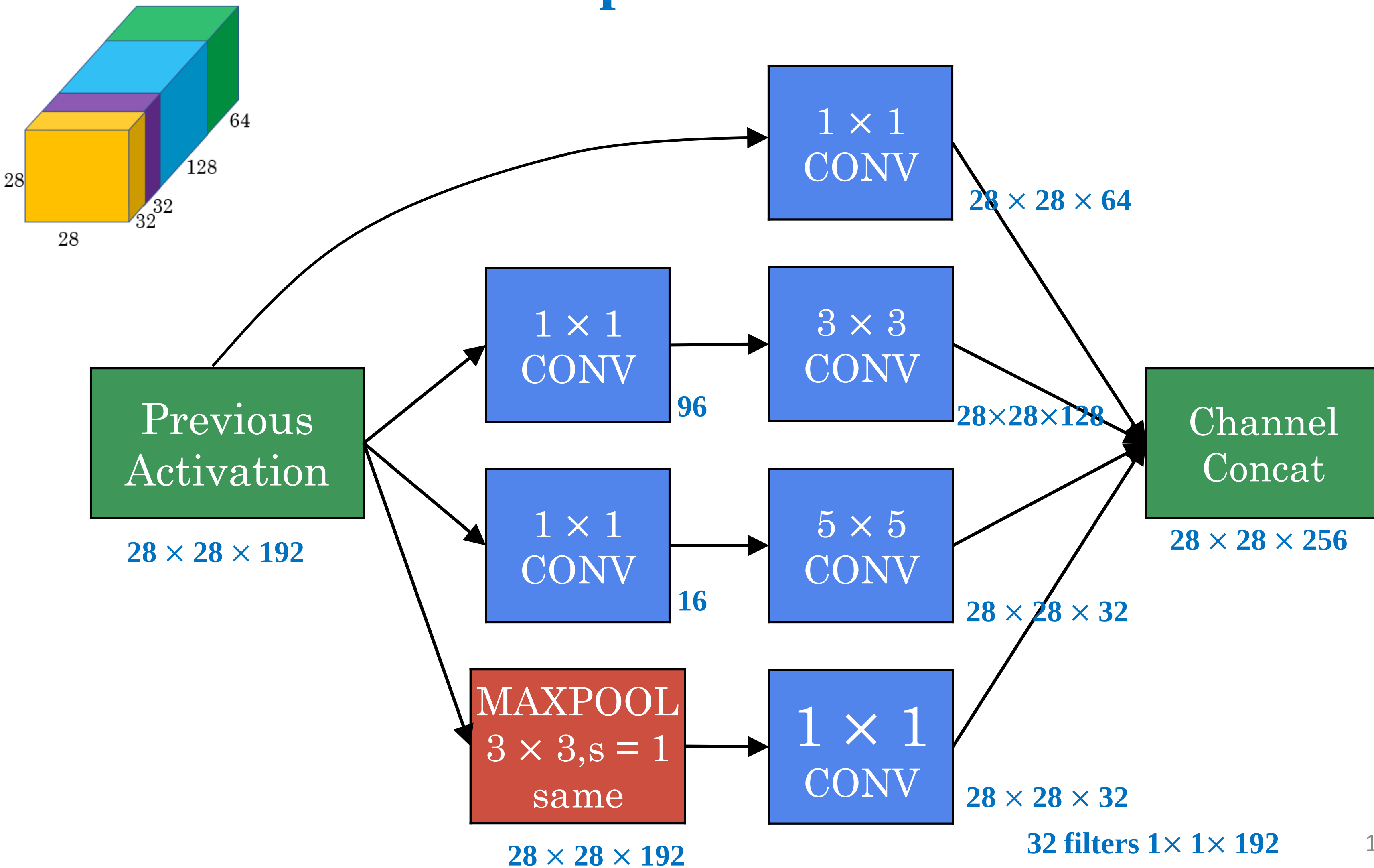
32 filters **filters are 5×5×192**

28 ×28 ×32 × 5×5×192 = 120M

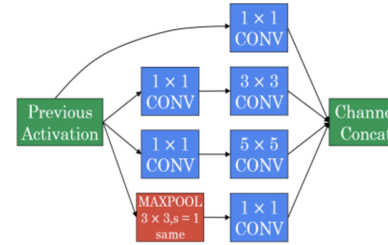
Using 1×1 convolution

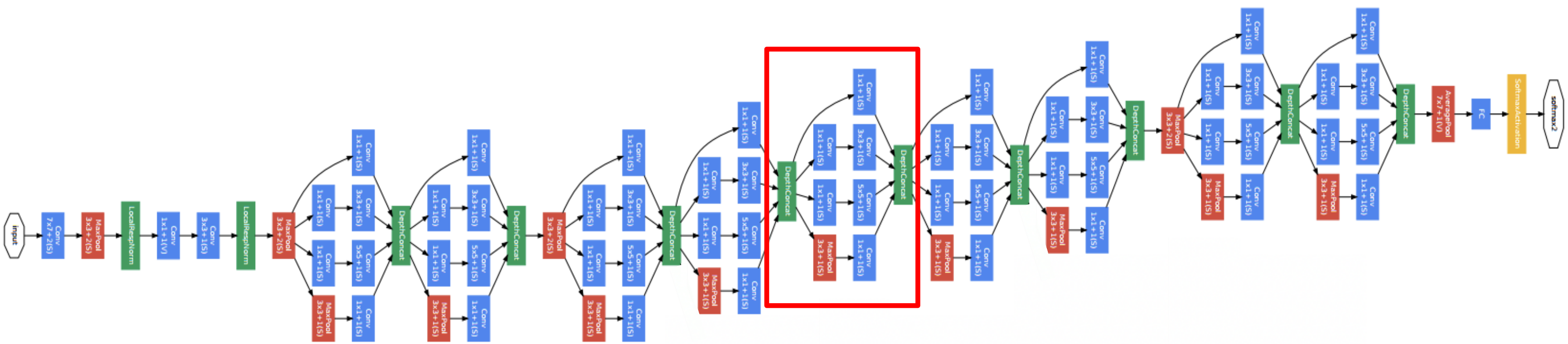


Inception module



Inception network





GooLeNet

[Szegedy et al., 2014, Going Deeper with Convolutions]

Using Open-Source Implementation

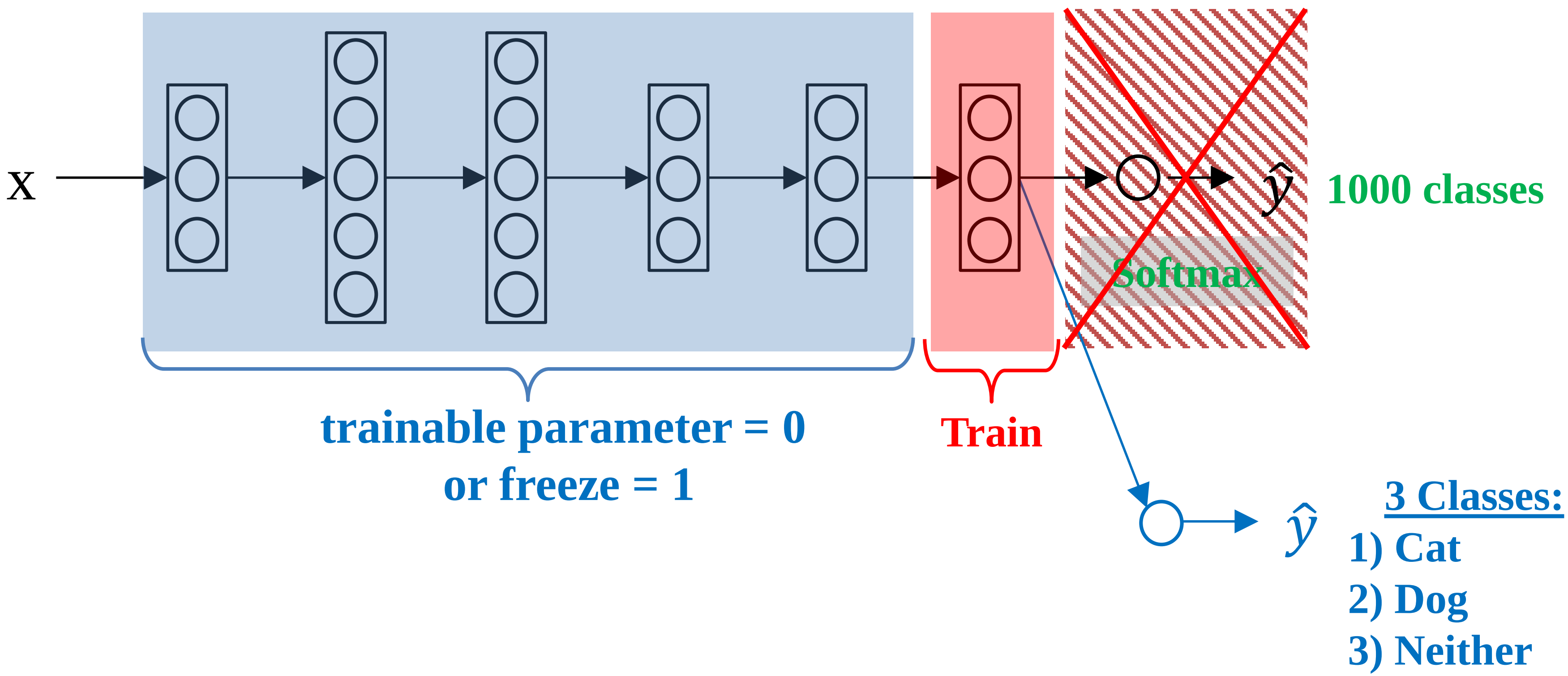
- Lot of **convolutional neural network architectures** are difficult to replicated. because there are some details that may not presented on its papers. There are some other reasons like:
    - Learning decay.
    - Parameter tuning.
- A lot of deep learning researchers are **opening source** their code into Internet on sites like Github.
- If you see a research paper and you want to build over it, the first thing you should do is to look for an **open source implementation** for this paper.
- Some advantage of doing this is that you might download the **network implementation** along with its parameters/weights. The author might have used **multiple GPUs** and spent some weeks to reach this result and its right in front of you after you download it.

Transfer Learning

- If you are using a specific neural network architecture that has been trained before, you can use this **pretrained parameters/weights** instead of random initialization to solve your problem.
- It can help you **boost the performance** of the neural network.
- The **pretrained models** might have been trained on a large datasets like **ImageNet**, **Ms COCO**, or **Pascal** and took a lot of time to learn those parameters/weights with optimized hyperparameters.
- This can save you a **lot of time**.

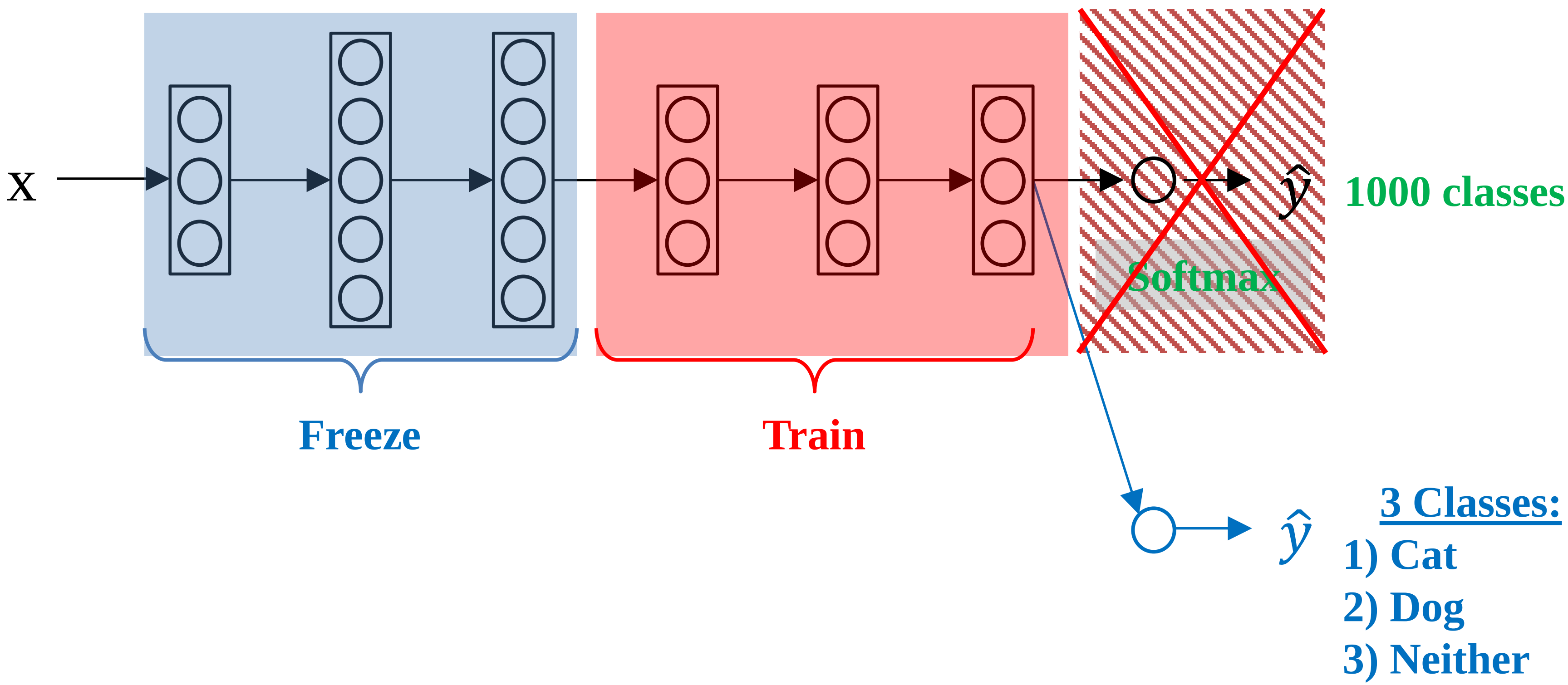
Transfer learning

Example 1:



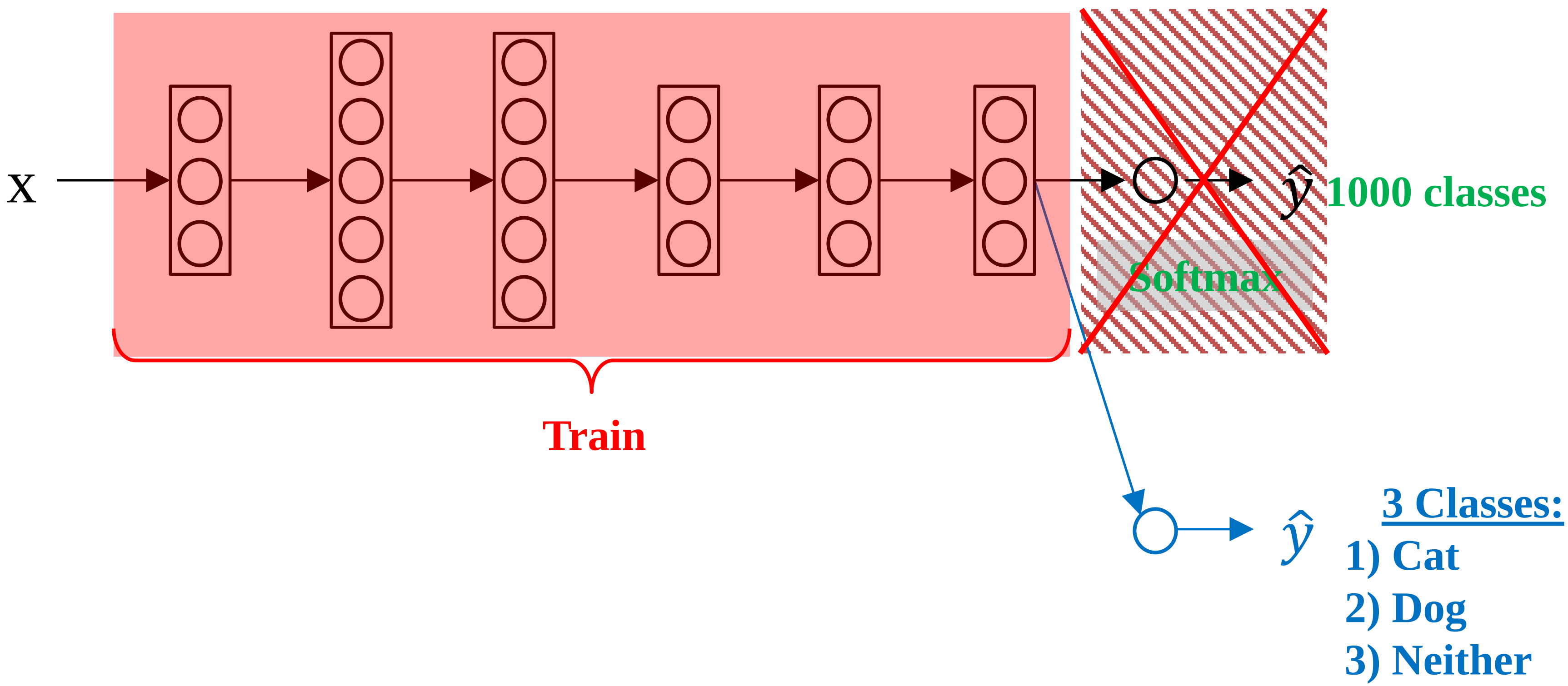
Transfer learning

Example 2:



Transfer learning

Example 3:

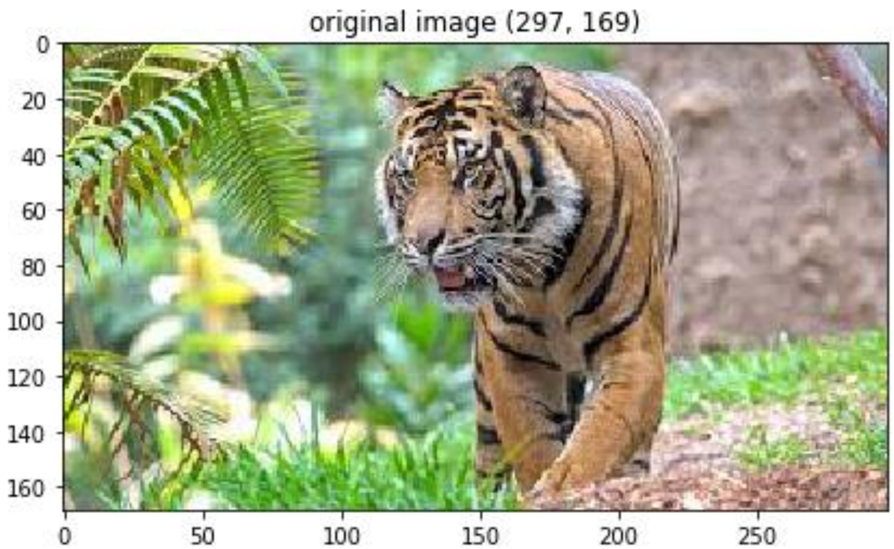


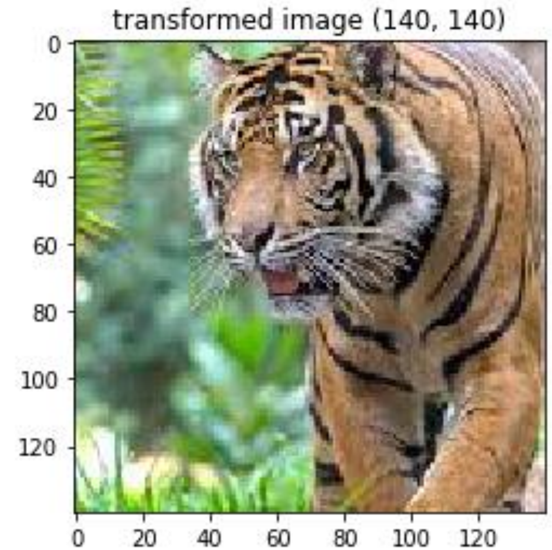
Data Augmentation

- If data is increased, your deep neural network will perform better.
- **Data augmentation** is one of the techniques that deep learning uses to **increase the performance** of deep neural networks.
- The majority of **computer vision applications** needs more data right now.
- Common **augmentation methods**:
  - Mirroring.
  - Cropping.
  - Rotation.
  - Shearing.
  - Local warping.
  - Color shifting.
  - ….

Common augmentation methods

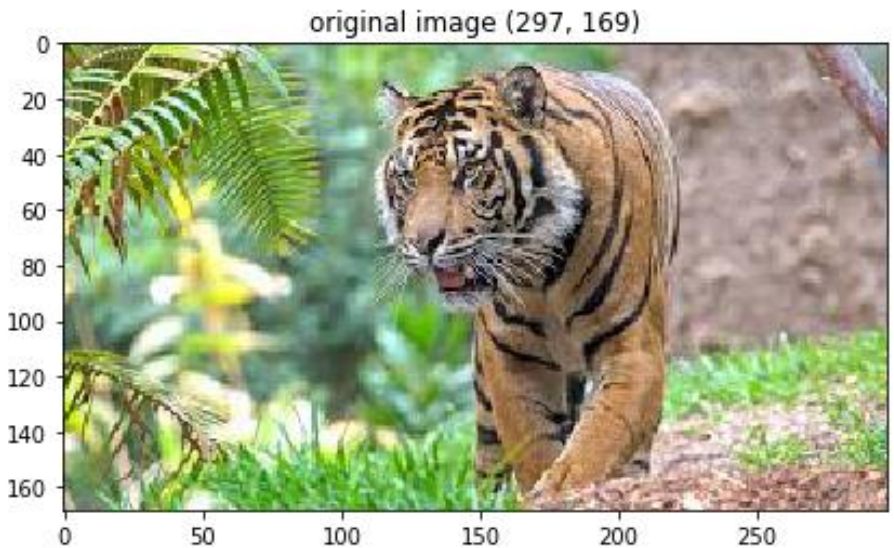
- **Cropping :**

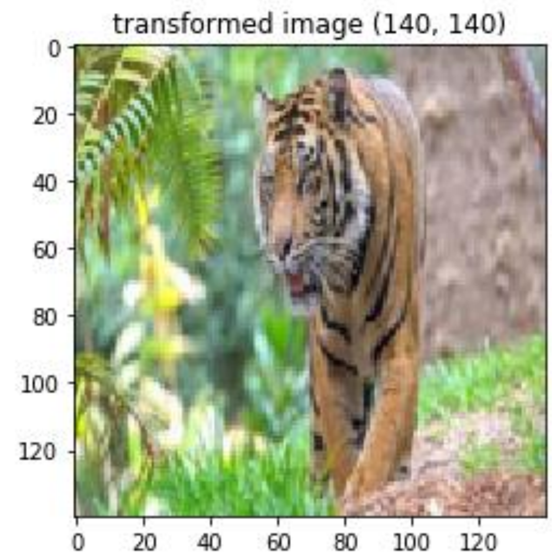




Common augmentation methods

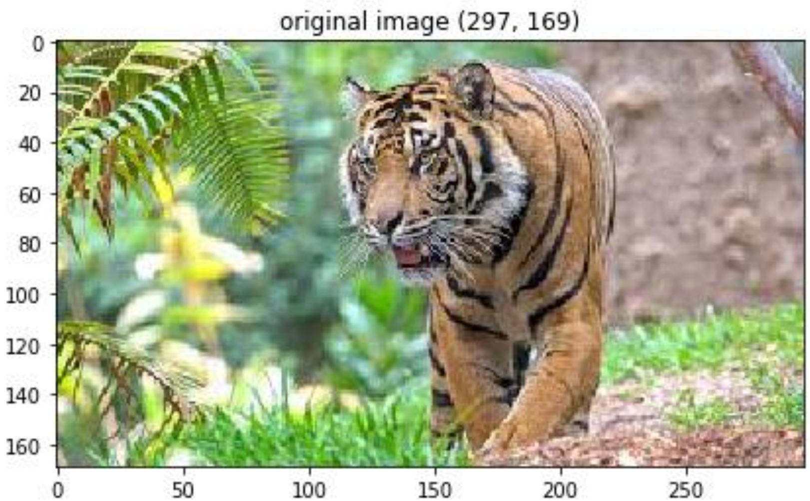
- **Scaling:**

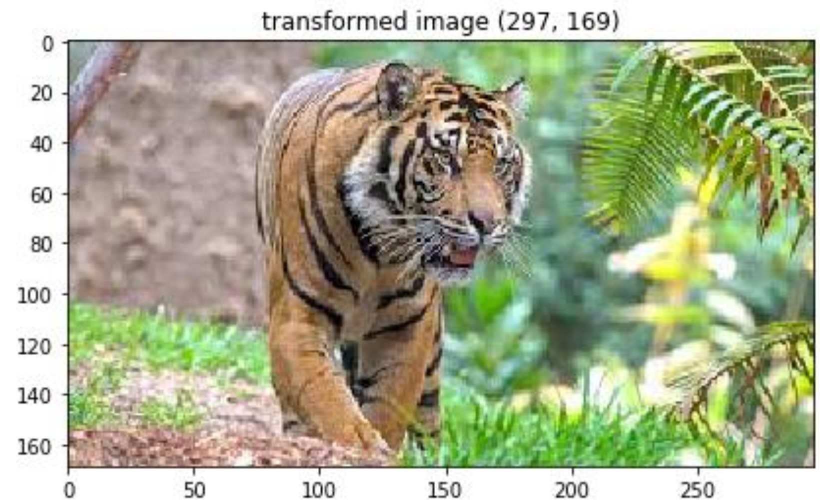




Common augmentation methods

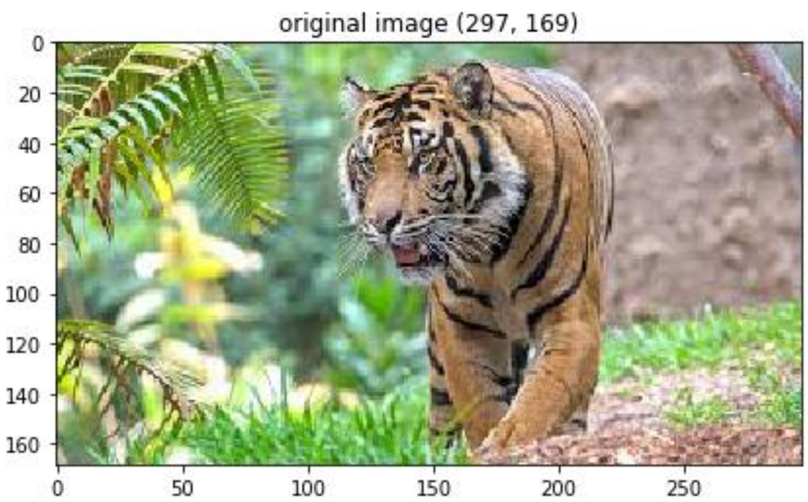
- **Flipping :**





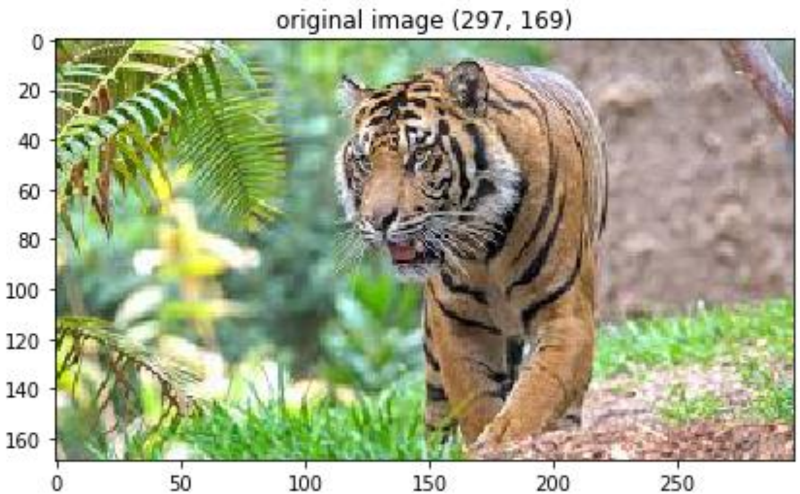
Common augmentation methods

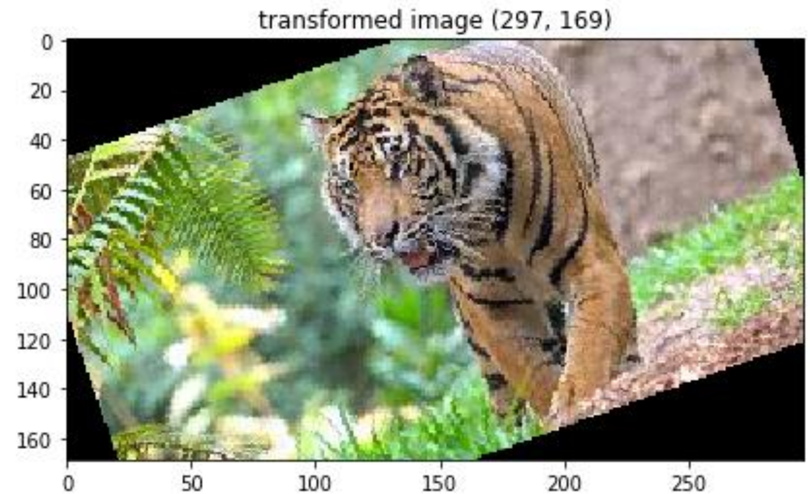
- **Padding :**



Common augmentation methods

- **Rotation :**





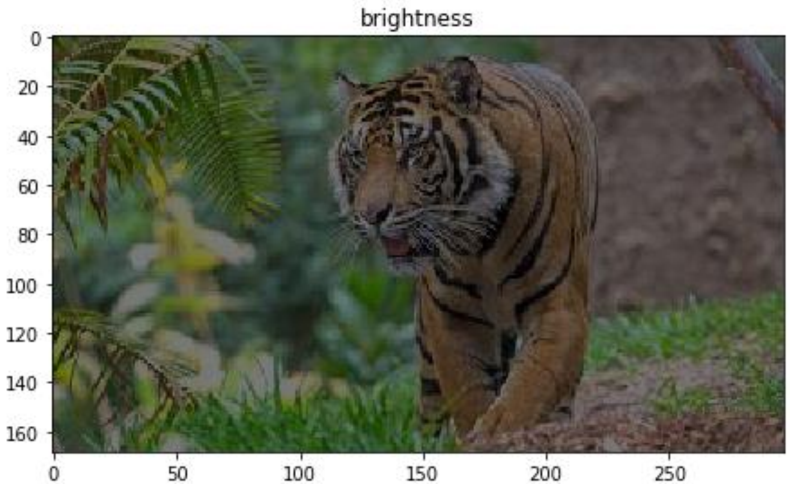
Common augmentation methods

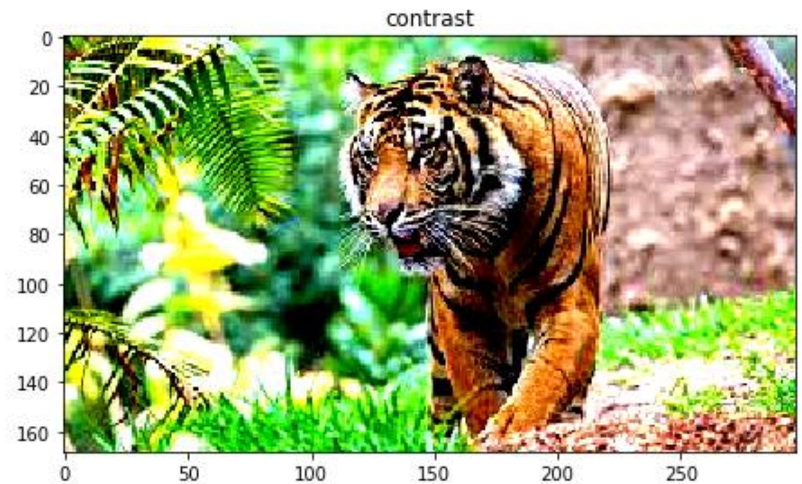
- **Affine transformation :**

- For example, we add to R, G, and B some distortions that will make the image identified as the same for the human but is different for the computer.

Common augmentation methods

- **Color augmentation (brightness, contrast):**

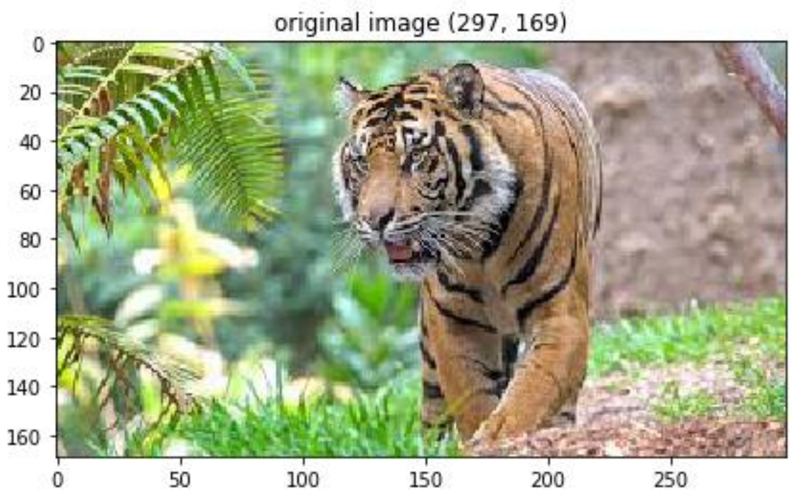


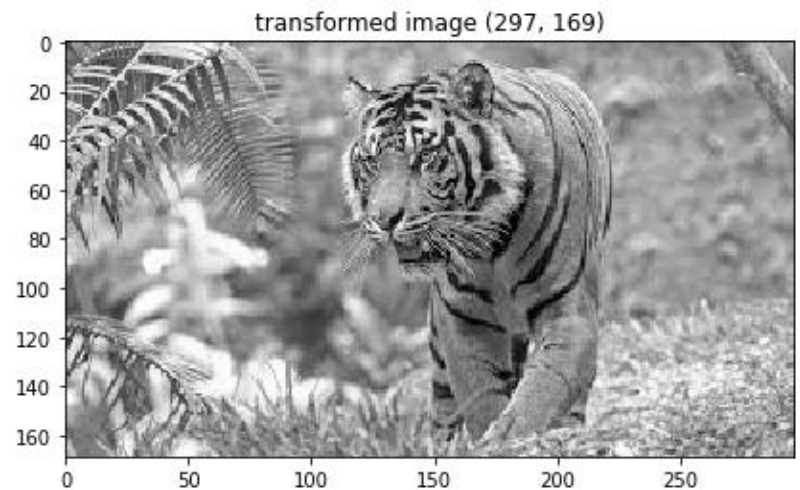


- There are an algorithm which is called **PCA color augmentation** that decides the shifts needed automatically.

Common augmentation methods

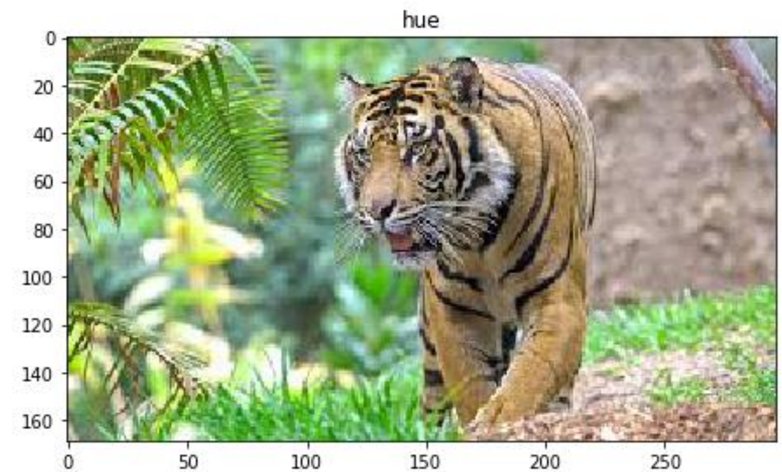
- **Color augmentation (Grayscale):**





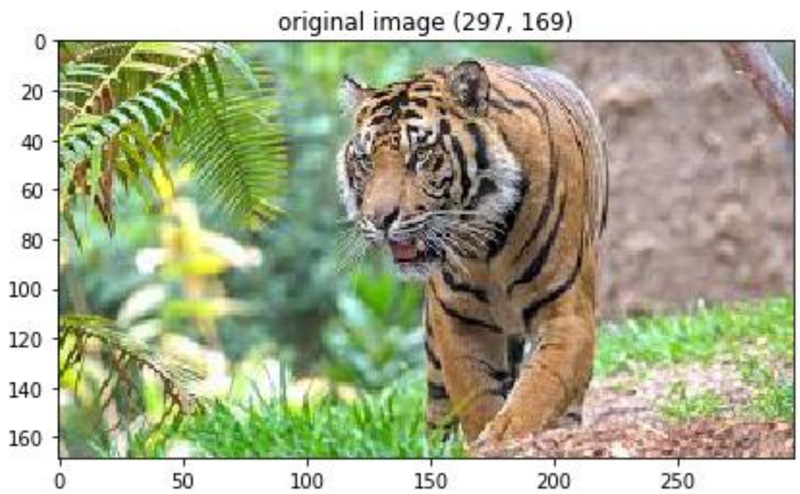
Common augmentation methods

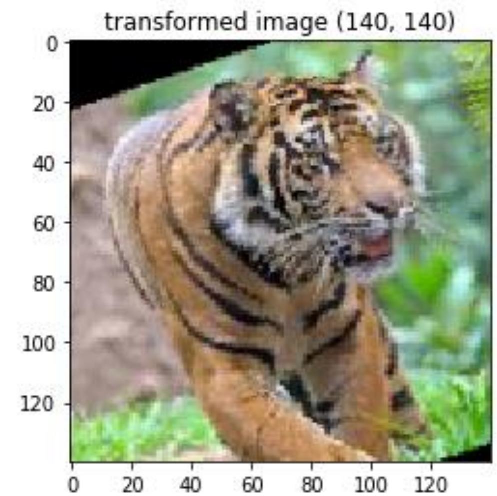
- **Color augmentation (saturation, hue):**



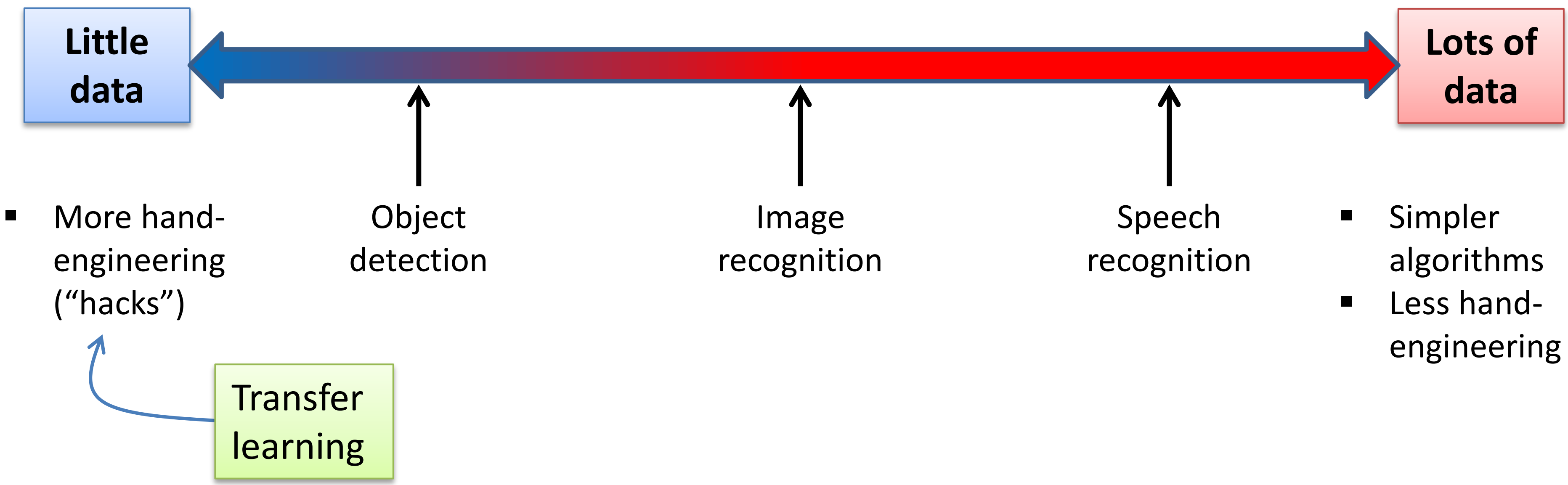
Common augmentation methods

- **Combination e.g. cropping after resizing:**





Data vs. hand engineering

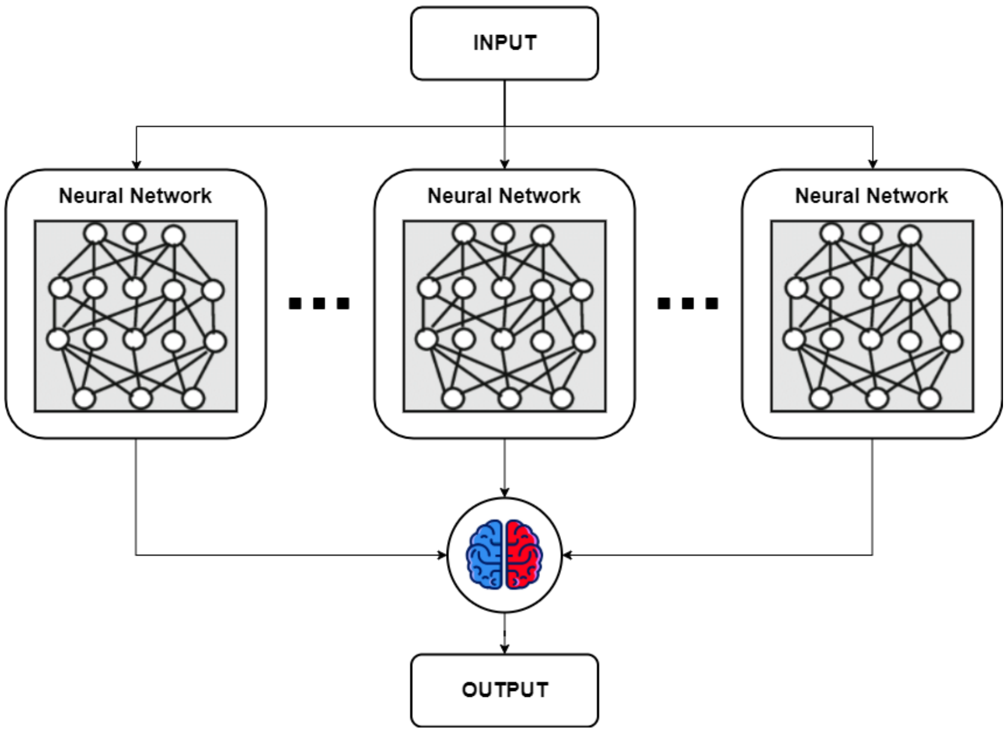


Two sources of knowledge:

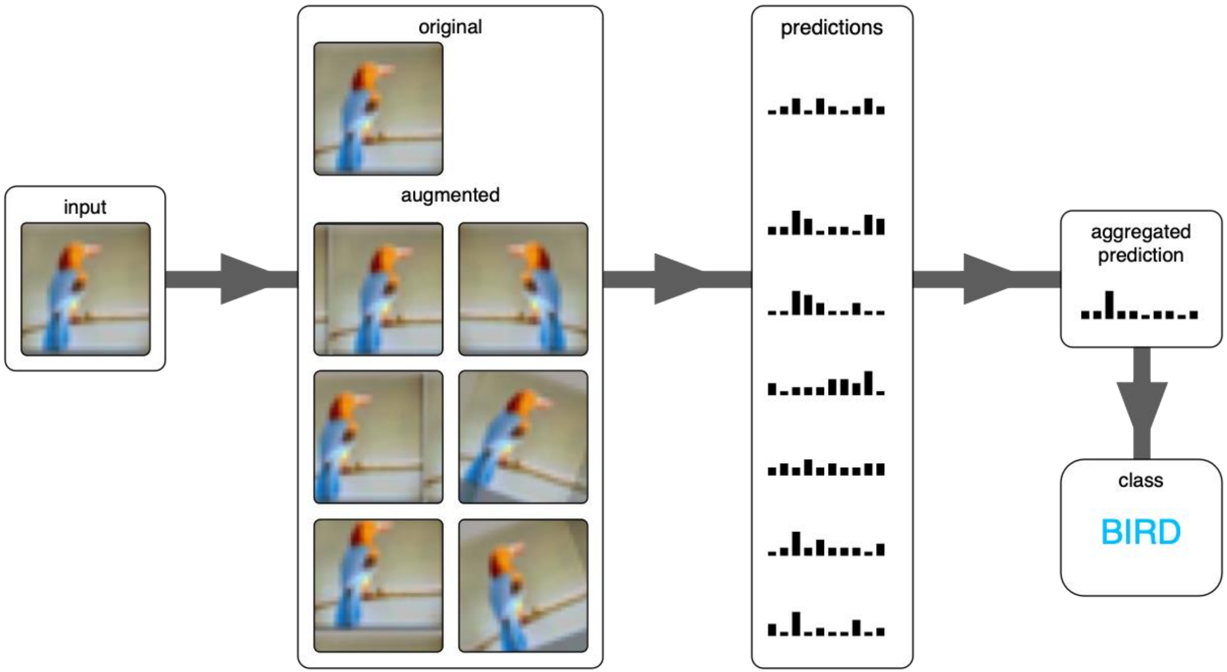
- Labeled data
- Hand engineered features/network architecture/other components.

Ensembling

- Train **several networks independently** and **average** their outputs.
- After choosing the best architecture, **initialize** some of that randomly and **train** them independently.
- This can give improve results by **2%**
- **Slow down production** by the number of the ensembles. Also it takes **more memory** as it saves all the models in the memory.
- Can be used in **competitions** but not in a real **productions**.



Multi-crop at test time



- Run classifier on multiple versions of test images and average results.
- There is a technique called **10 crops** that uses this.
- This can give a better result in the production.

Use open source code

- Use **architectures** of networks published in the literature.
- Use open source **implementations** if possible.
- Use **pretrained models** and fine-tune on your dataset.

References

- Andrew Ng. Deep learning. Coursera.
- Geoffrey Hinton. Neural Networks for Machine Learning.
- Kevin P. Murphy. Probabilistic Machine Learning An Introduction. MIT Press, 2022.
- MIT Deep Learning 6.S191 (http://introtodeeplearning.com/)