



# DEEP NETWORK DEVELOPMENT

**Imre Molnár**

PhD student, ELTE, AI Department

✉ [imremolnar@inf.elte.hu](mailto:imremolnar@inf.elte.hu)

🌐 [curiouspercibal.github.io](https://curiouspercibal.github.io)

**Tamás Takács**

PhD student, ELTE, AI Department

✉ [tamastheactual@inf.elte.hu](mailto:tamastheactual@inf.elte.hu)

🌐 [tamastheactual.github.io](https://tamastheactual.github.io)

# Lecture 4.

# CNN architectures, Transfer Learning & Autoencoders

---

Budapest, 7<sup>th</sup> March 2025

**1** CNN Architectures

**2** Transfer Learning

**3** Autoencoders

## Previously on Lecture 3

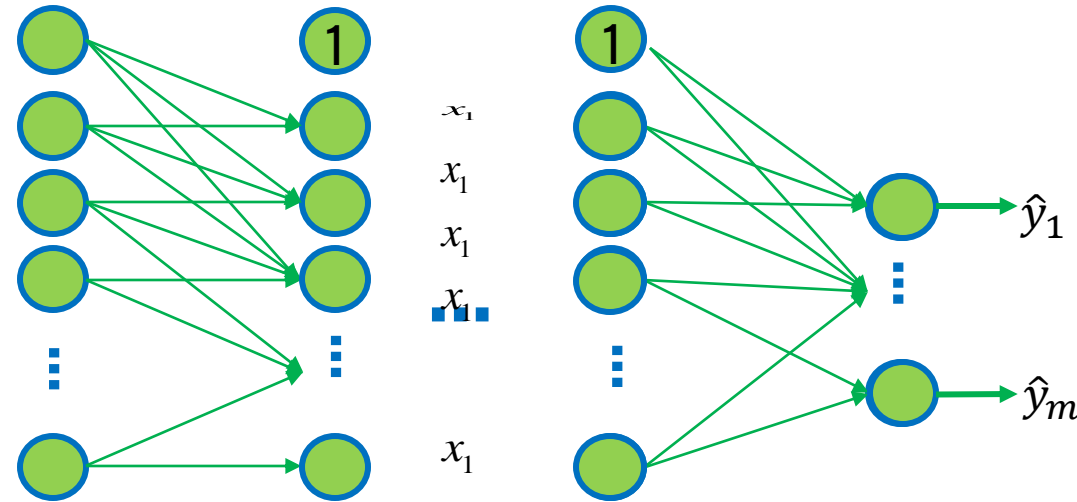
- With ordered input data, data that are close to each other are more closely related, so stronger/more relationships are needed.

### Advantage:

- memory requirements are significantly reduced
- The relationship pattern can be well optimized by exploiting the relationship pattern, thus allowing fast and transparent operation.

### Disadvantage:

- More complex network, but still structured



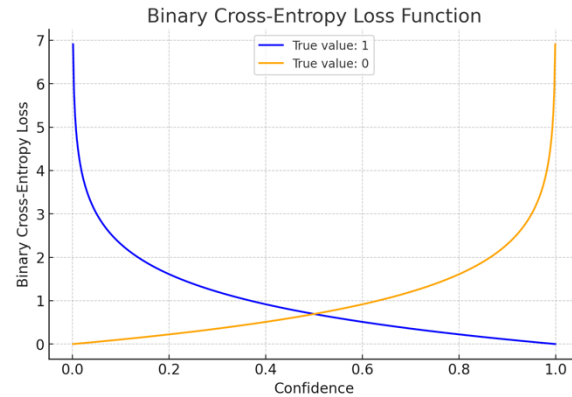
For ordered data, other connection patterns are also possible.

Data that are distant from each other may not even be related (e.g., for time series, causality can be exploited, for images, pixels that are distant from each other are less and less likely to belong to the same object).

# Previously on Lecture 3

## Binary Classification

- Supervised learning
- Have:  $(x, y)$ 
  - $x$ : input
  - $y$ : target  $\in \{0, 1\}$
- *Goal: Learn a function to map  $x \rightarrow y$ .*
  - $h(x) = \hat{y}$
  - $\hat{y} = g(\sum_{j=0}^m w_j x_j + b)$  Add non-linear activation function
- Classification – Predict a discrete set of values:
  - $\hat{y} \in \{0, 1\}$
- Binary cross-entropy loss
 
$$L_{BC}(h(x), y) = -y \log h(x) - (1 - y) \log(1 - h(x))$$



## Multi-class Classification

- With  $K$  classes
- The network will have  $K$  outputs
- Softmax activation for squashing outputs between 0 and 1 (to mimic Probability distribution)
- $\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{i=1}^K e^{z_i}}$

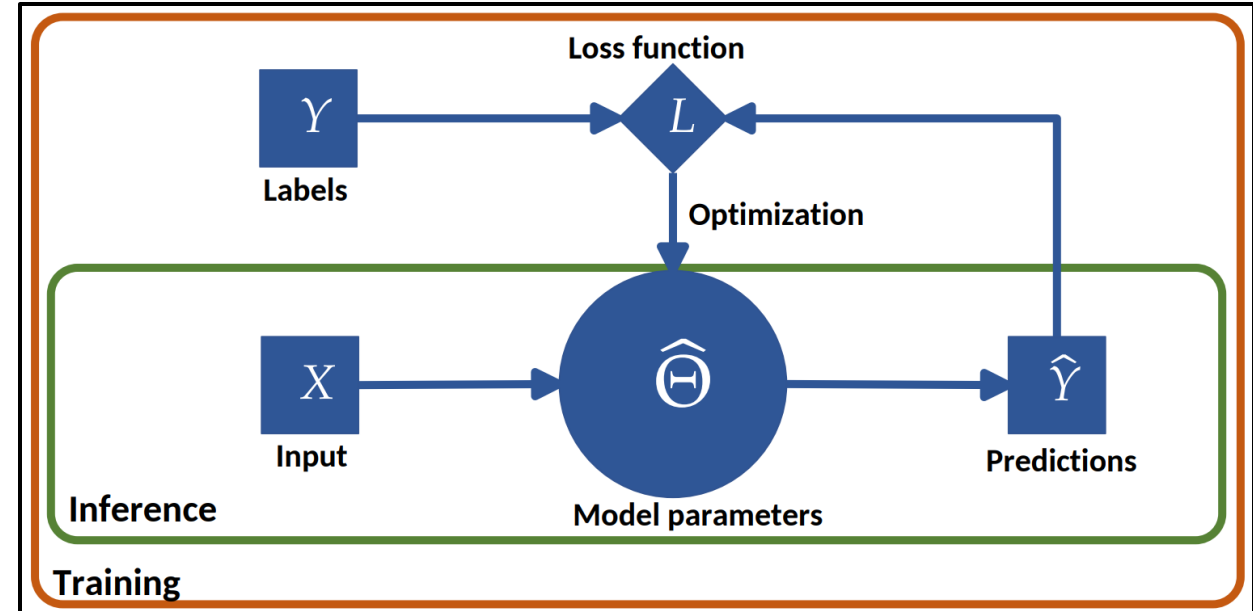
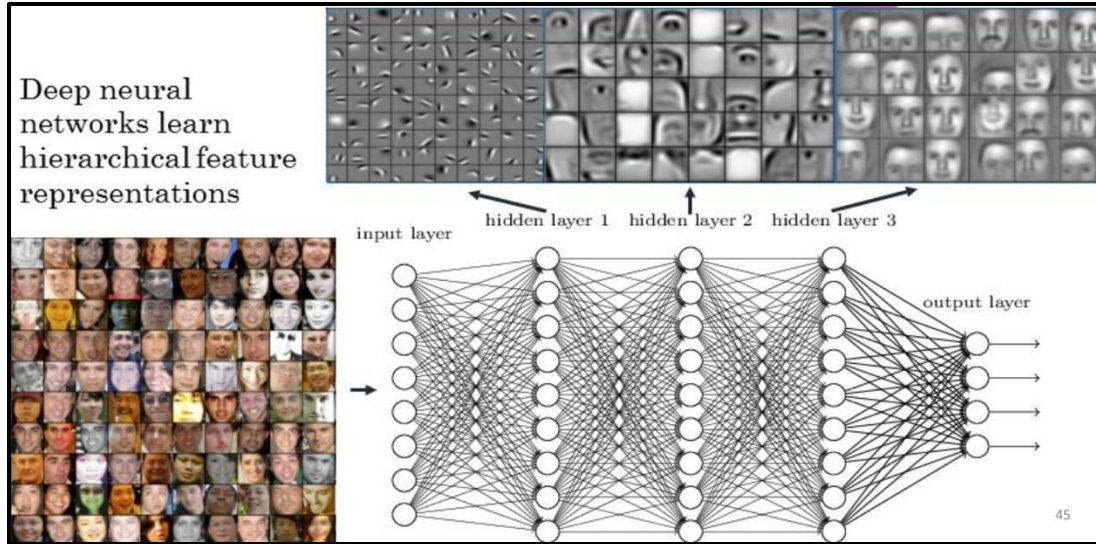
One-hot encoding ( $K=3$ ):

(cat) 1  $\rightarrow$  [1, 0, 0]  
 (dog) 2  $\rightarrow$  [0, 1, 0]  
 (horse) 3  $\rightarrow$  [0, 0, 1]

Categorical Cross-entropy loss

$$L(h(x), y) = - \sum_{i=1}^K y_i \log h_i(x)$$

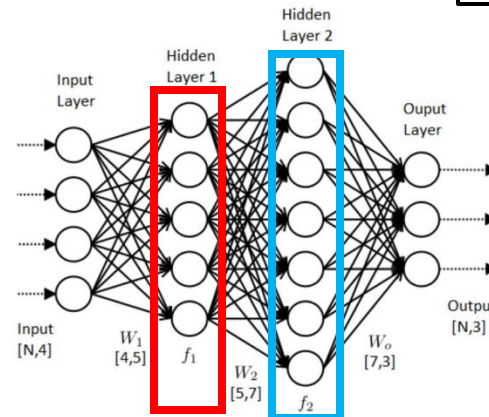
# Previously on Lecture 3



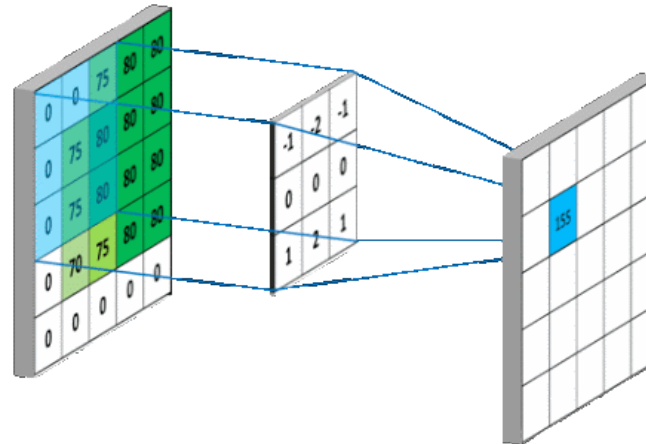
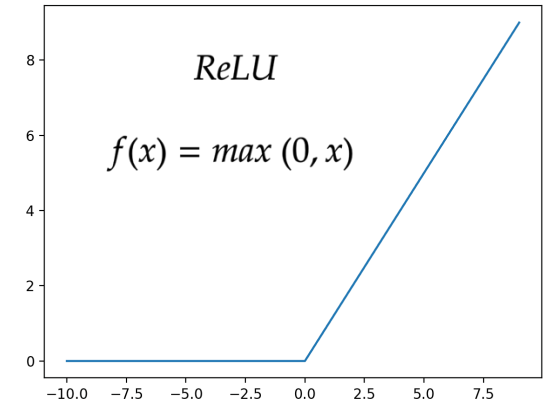
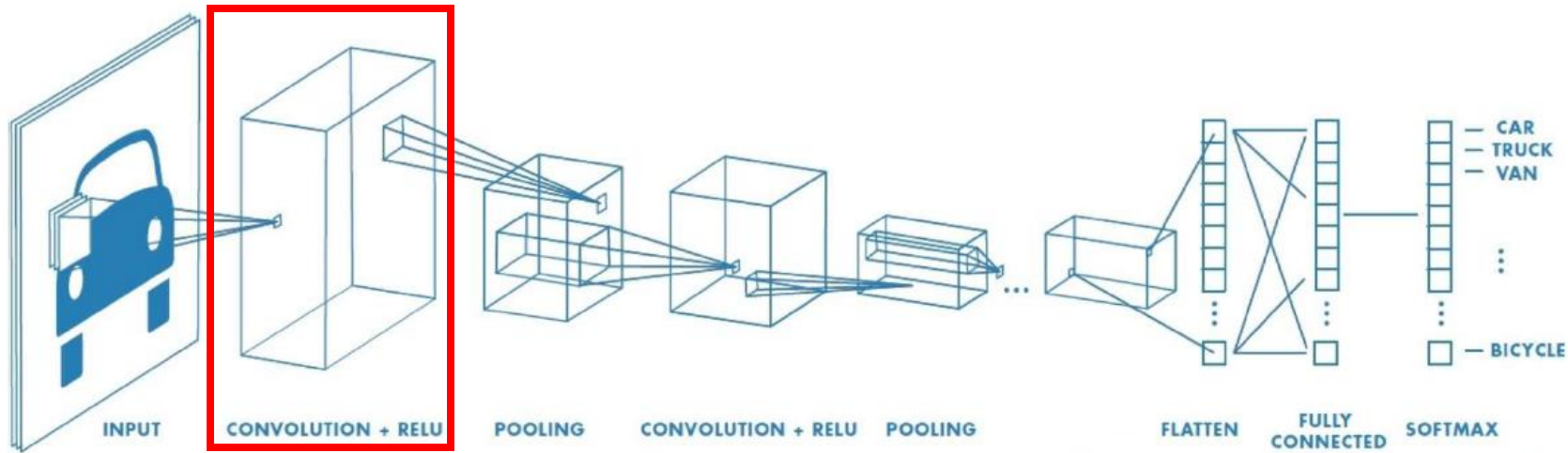
- Given  $h = g(W^{(2)}g(W^{(1)}x + b^{(1)}) + b^{(2)})$
- Update  $W^{(1)}$  :  

$$W_t^{(1)} = W_{t-1}^{(1)} - \alpha \nabla J(W^{(1)})$$
- Update  $W^{(2)}$  :  

$$W_t^{(2)} = W_{t-1}^{(2)} - \alpha \nabla J(W^{(2)})$$
- Same for bias

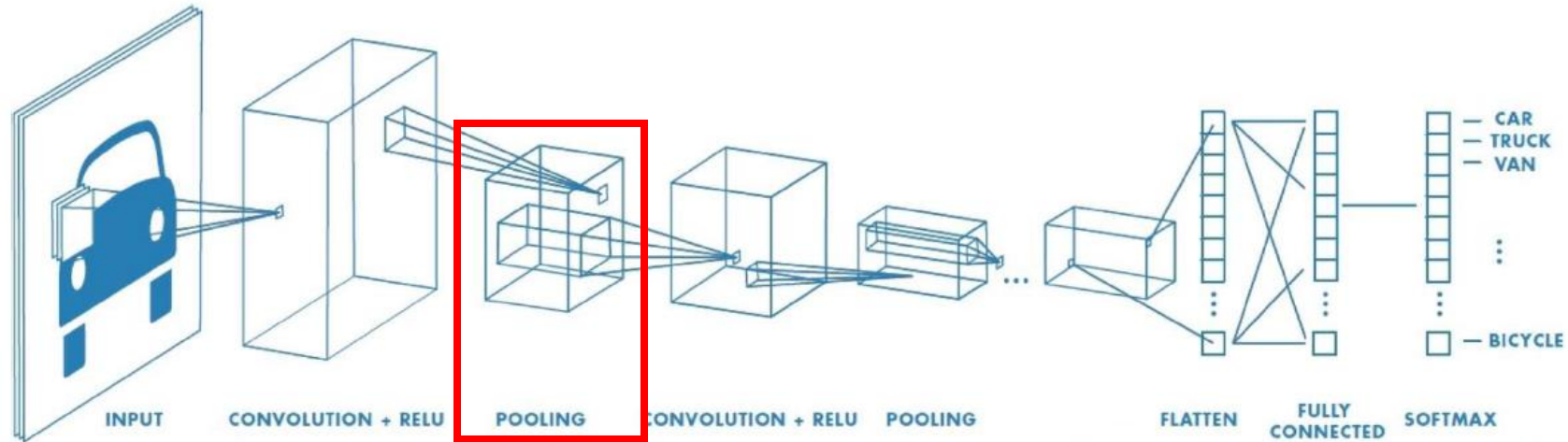


# Previously on Lecture 3





# Previously on Lecture 3



4	6	1	1
1	3	1	3
4	0	0	8
8	5	4	0

Input (4x4)



Output (2x2)

# Lecture 4.



# CNN Architectures

---

Budapest, 7<sup>th</sup> March 2025

**1** CNN Architectures

**2** Transfer Learning

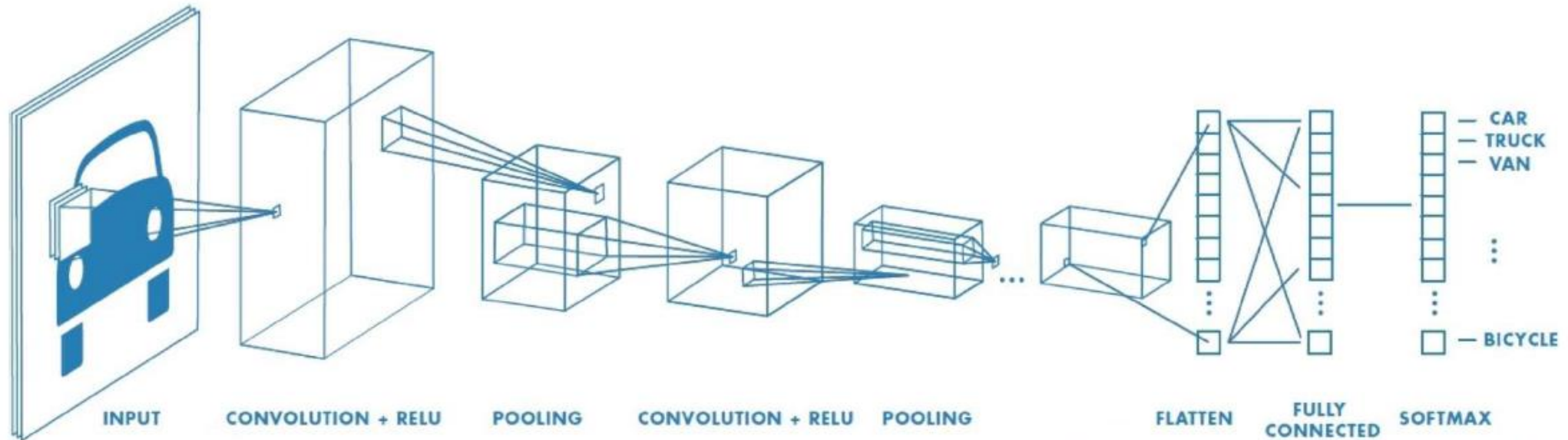
**3** Autoencoders



## CNN architectures

### Visualization:

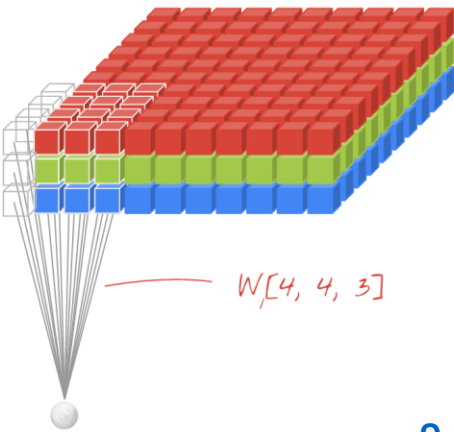
- <https://poloclub.github.io/cnn-explainer/>
- <https://distill.pub/2017/feature-visualization/>
- [https://adamharley.com/nn\\_vis/](https://adamharley.com/nn_vis/)



# Convolutional Layer

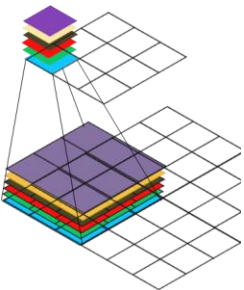
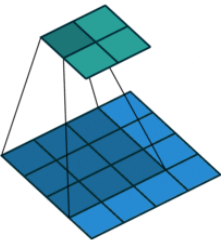
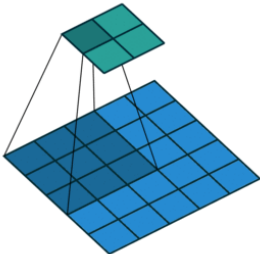
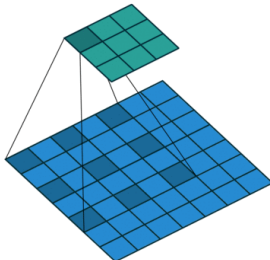
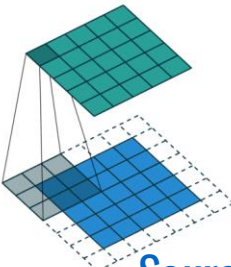
- Purpose: To quantify the correspondence with the characteristics defined by the weights of the filters
- E.g.: how much of a (vertical edge)/nose/wheel/face is in the image part under consideration

Num. of filters	2 filters shown
Kernel size	4x4x3
Stride	1,1
Dilatation	1,1
Padding	0,0,1,2



[Source](#)

- Play

Filters: 6	Kernel size 3x3	Stride 2,2	Dilatation 2,2	Padding 1,1,1,1 (up, down, left, right)
				

[Source](#)

# Convolutional Layer

## Summary of convolutions

$n \times m$  **image**       $f \times f$  **filter**       $c$  **channels**      padding  $p$       stride  $s$

$$\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{m + 2p - f}{s} + 1 \right\rfloor \times c$$

### Example:

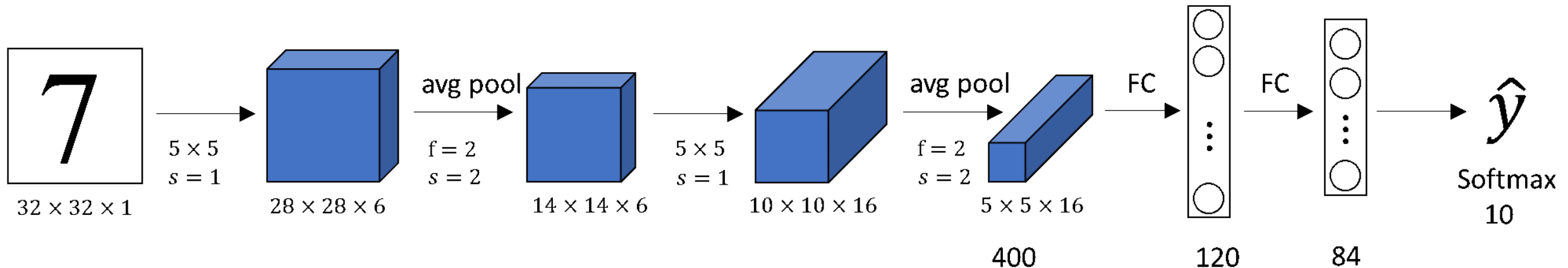
Input Image = 224x224x3

Filter = 3x3x3 (x32)

Output = 222x222x32

$n = 224; p = 0; s = 1; f = 3; m = 32$

## “LeNet 5” Y. Lecun et al. (1998) [2]



[2] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

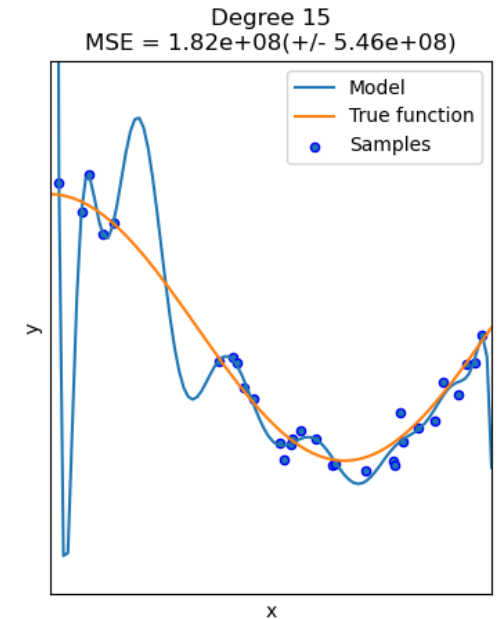
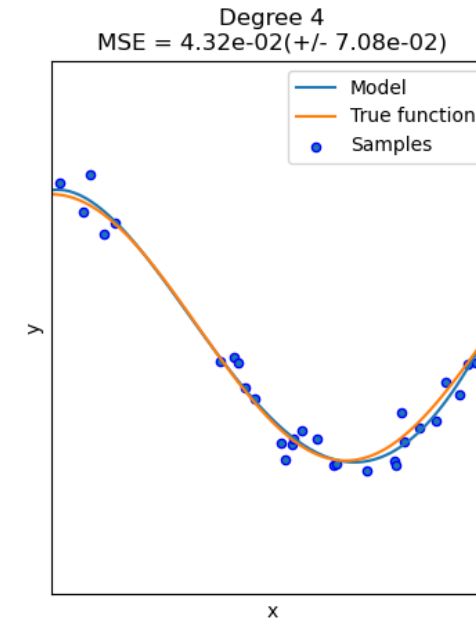
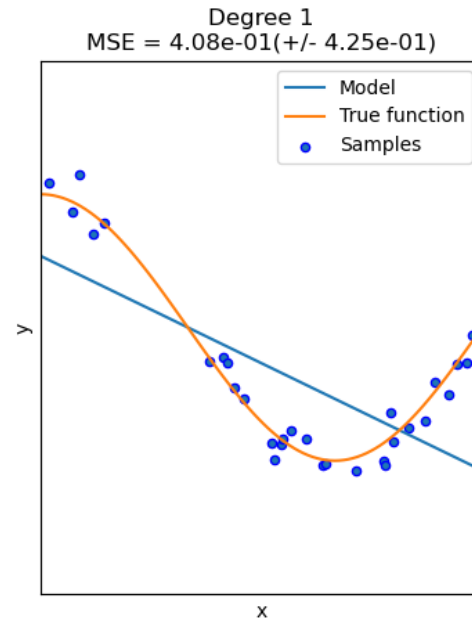
# Going deeper

### What prevents us from creating CNN with arbitrary depth?

- Underfitting/Overfitting
- Vanishing/Exploding gradient
- Generalization
- And many more...

## Underfitting/Overfitting

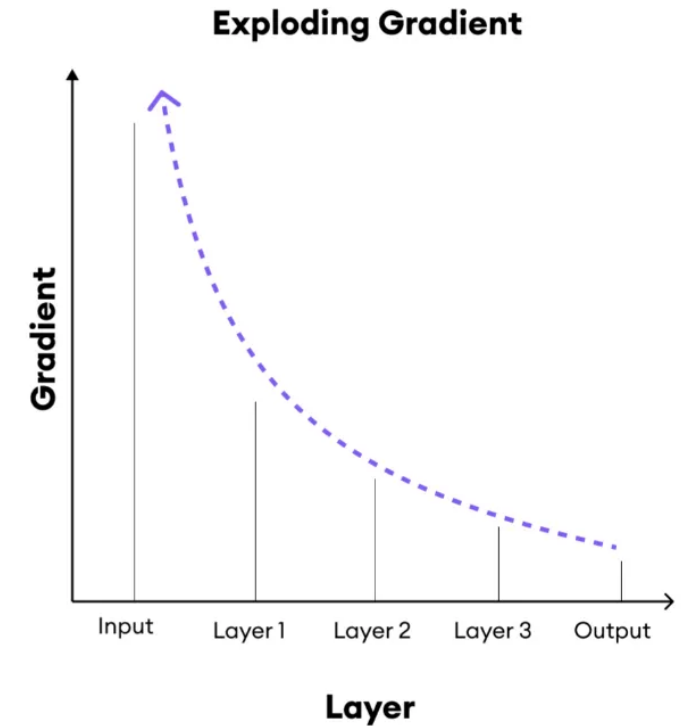
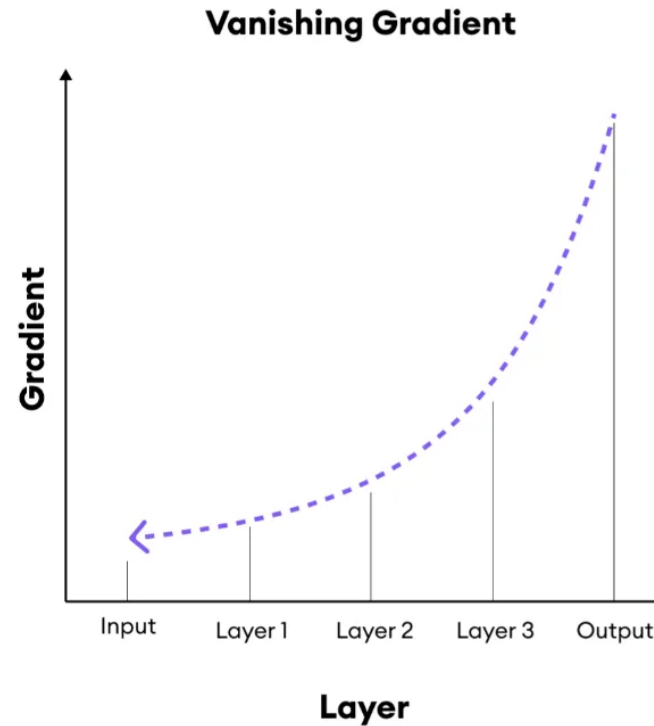
- Underfitting
  - we train the model for more epochs
  - If we have more parameter, we can counter this issue
- Overfitting
  - Regularization (early-stopping, dropout, etc.)
  - If we have less parameters, it is less likely that our model will overfit





# Vanishing/Exploding Gradient

- Weight initialization
- Activation function
- Regularization
  - Batch Normalization
  - Dropout
  - Weight decay
- “Skip connection”
- And many more...



# Weight initialization

- As the name suggest specifies the method how we initialize the weight of our neural network
  - Sample from a uniform distribution in  $[a, b]$
  - Sample from a normal distribution  $N(\mu, \sigma)$  – usually with 0 mean and 1 standard deviation
  - Set it to some constant value
  - Or manually add the initial weights
- In pytorch the weights of the Convolution is sampled from  $U(-\sqrt{k}, \sqrt{k})$ , where  $k = \frac{groups}{C_{in} * \prod_{i=0}^1 kernel\_size[i]}$
- And in Linear layers is sampled from  $U(-\sqrt{k}, \sqrt{k})$ , where  $k = \frac{1}{in\_features}$

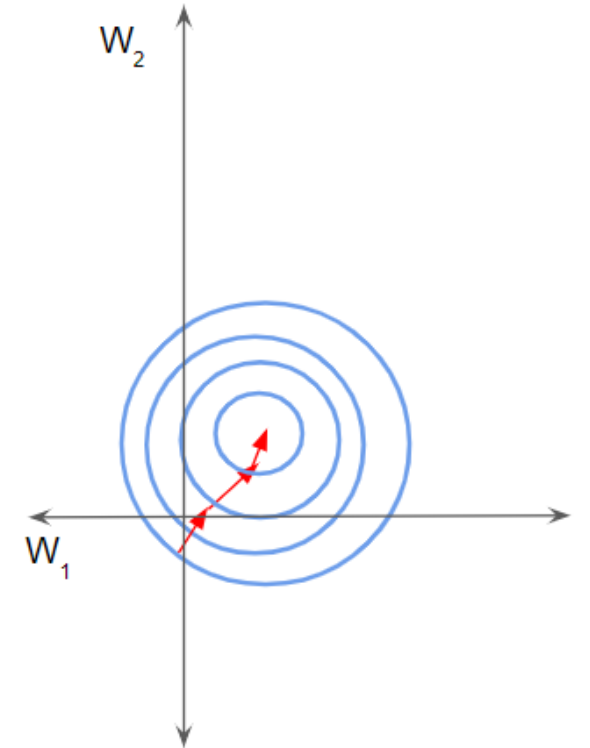
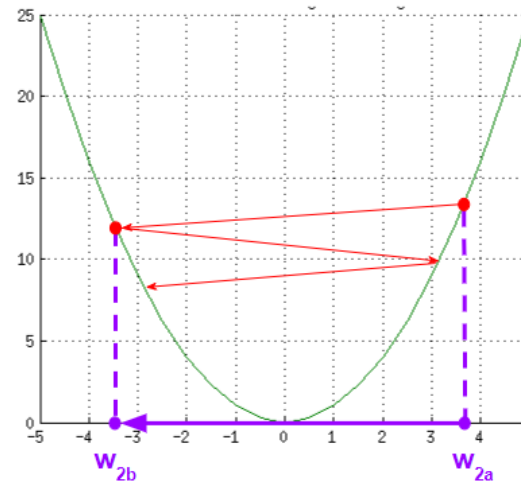
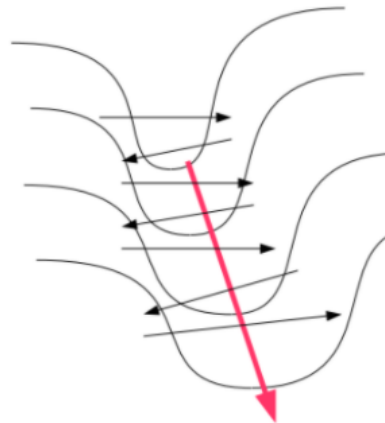
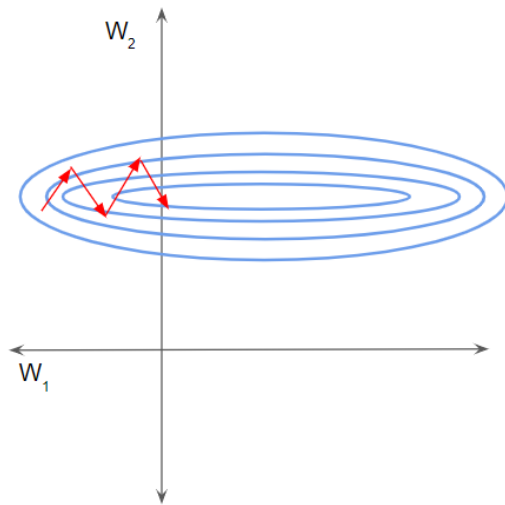
# Regularization

- Weight related regularizations:
  - **L1 regularization** (also called LASSO) leads to sparse models by adding a penalty based on the absolute value of coefficients.
  - **L2 regularization** (also called ridge regression) encourages smaller, more evenly distributed weights by adding a penalty based on the square of the coefficients.
- Early stopping
- Data augmentation
- Batch normalization
- Dropout
- And many more...

## Batch Normalization

- Batch normalization is achieved through a normalization step that fixes the means and variances of each layer's inputs
- It basically normalizes the output of the previous layer in this batch
  - It also keeps track of moving averages, and can do some transformations too

$$y_B = \frac{x_B - \mathbf{E}[x_B]}{\sqrt{\mathbf{Var}[x_B]}}$$



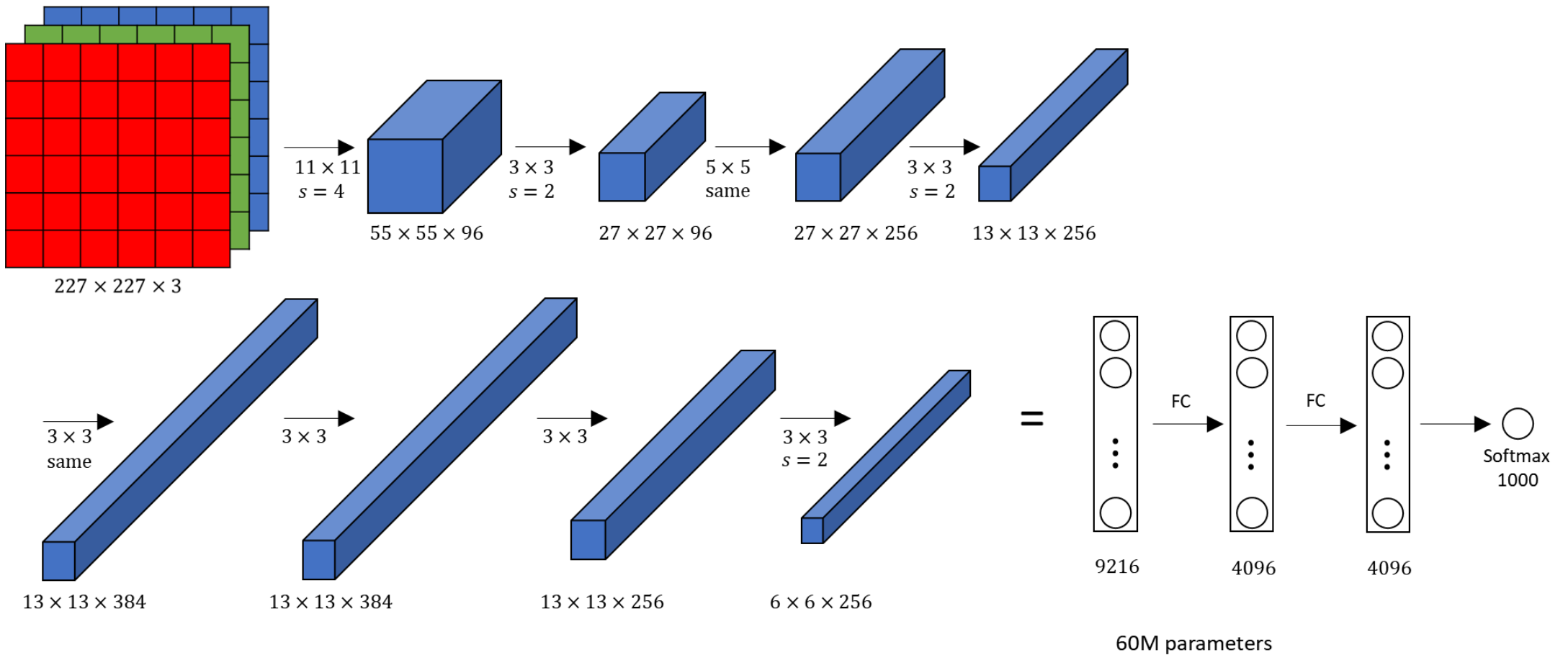
# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [10]

- **1000 classes**
- **1M Images**
- **Serves as the common benchmark for neural nets**

[10] J. Deng, W. Dong, R. Socher, L. -J. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.

# 1. Convolutional Neural Networks

## “AlexNet” A. Krizhevsky et al. (2012) [3]



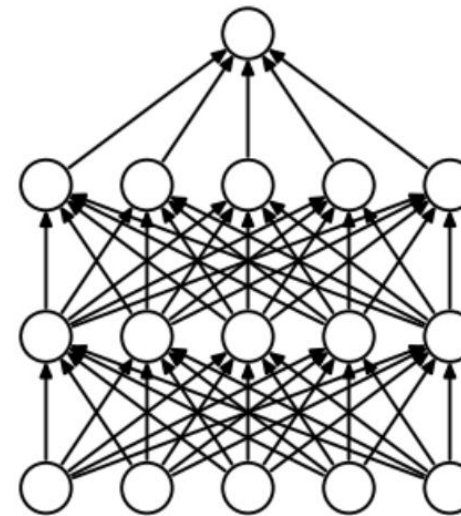
[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12). Curran Associates Inc., Red Hook, NY, USA, 1097–1105.



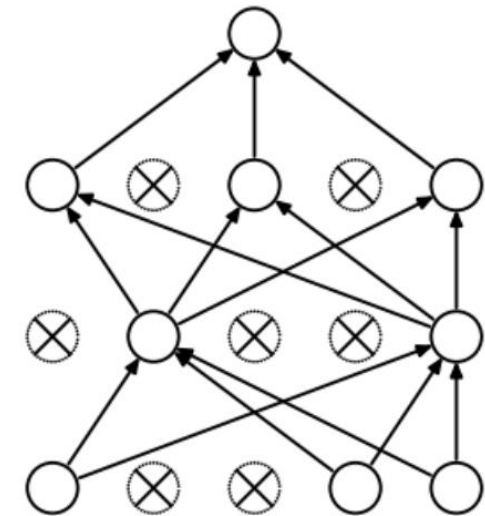
# “AlexNet” A. Krizhevsky et al. (2012) [3]

### Dropout:

- We are dropping some neurons between layers with some probability  $p$ .
- The dropped out neurons do not contribute to the **forward pass** and do not participate in the **backpropagation**.
- “Every time we sample from a different architecture”
- Reduces neuron co-adaptation
- The model forced to learn more robust features



(a) Standard Neural Net



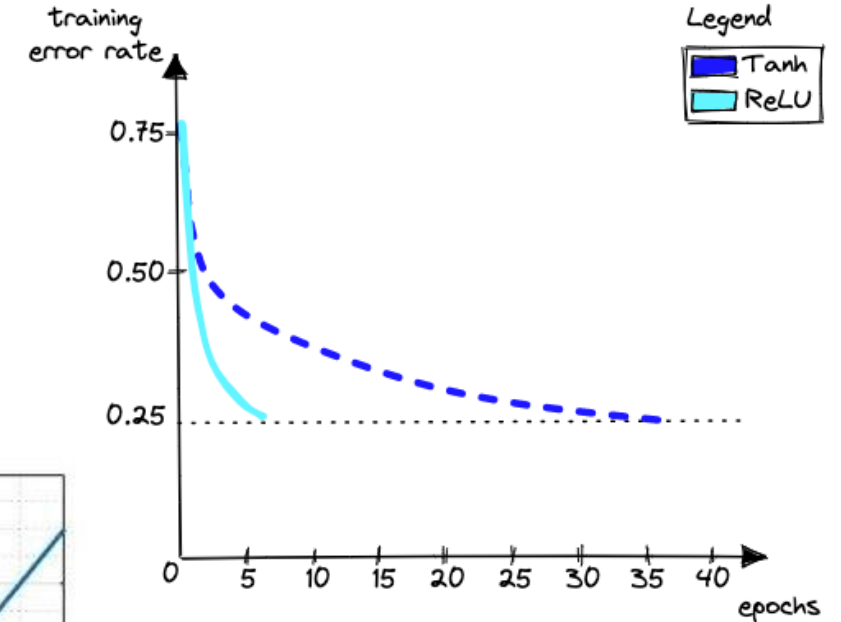
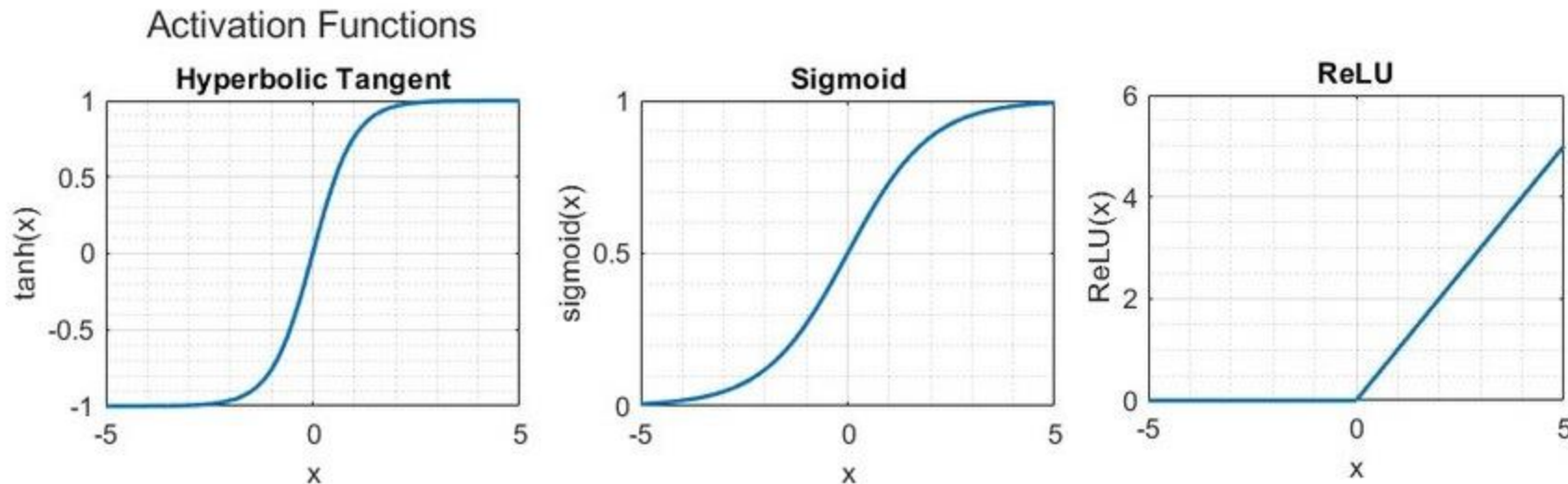
(b) After applying dropout.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12). Curran Associates Inc., Red Hook, NY, USA, 1097–1105.

## “AlexNet” A. Krizhevsky et al. (2012) [3]

### ReLU over Sigmoid or tanh:

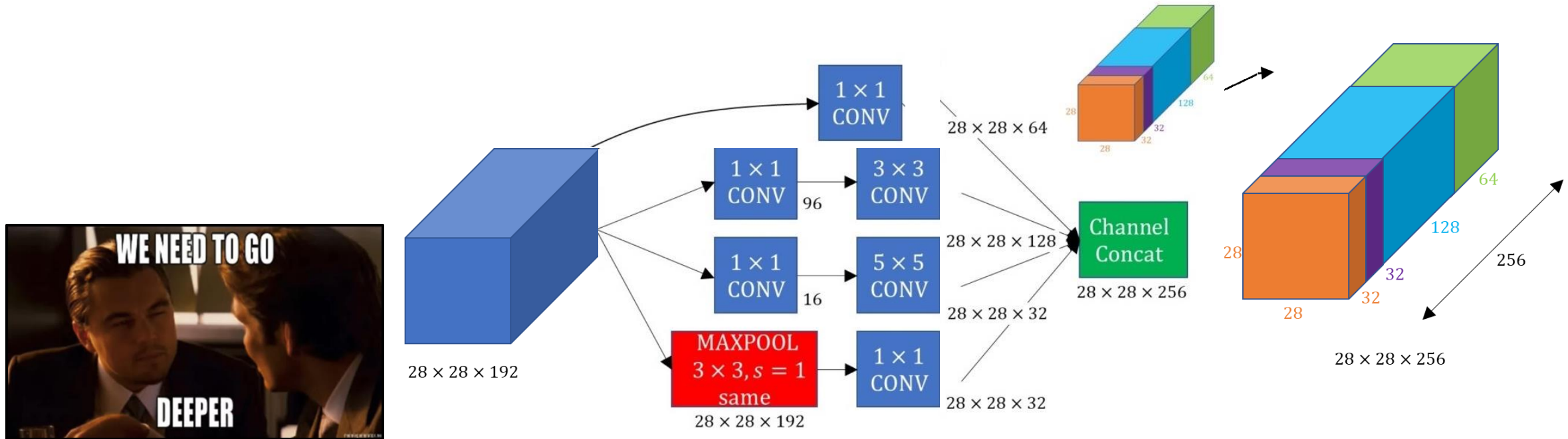
- During training with **Gradient Descent** using **ReLU** (non-saturating nonlinearity) results in less training time compared to **Sigmoid** or **tanh** (saturating nonlinearity).
- Does not require input normalization
- Learning will happen if at least some training examples produce positive input



[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12). Curran Associates Inc., Red Hook, NY, USA, 1097–1105.

## “Inception / GoogLeNet” C. Szegedy et al. (2014) [5]

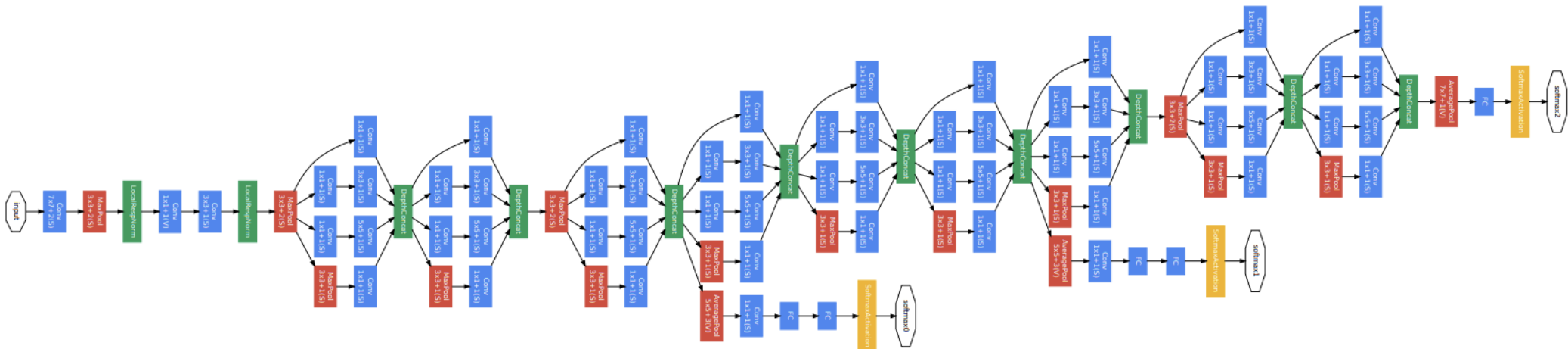
- Aligns with the intuition that visual information should be processed at various scales
- 1x1 Convolution blocks for dimensionality reduction



[5] Szegedy, C., et al. “Going Deeper with Convolutions”, *arXiv e-prints*, 2014. doi:10.48550/arXiv.1409.4842.

## “Inception / GoogLeNet” C. Szegedy et al. (2014) [5]

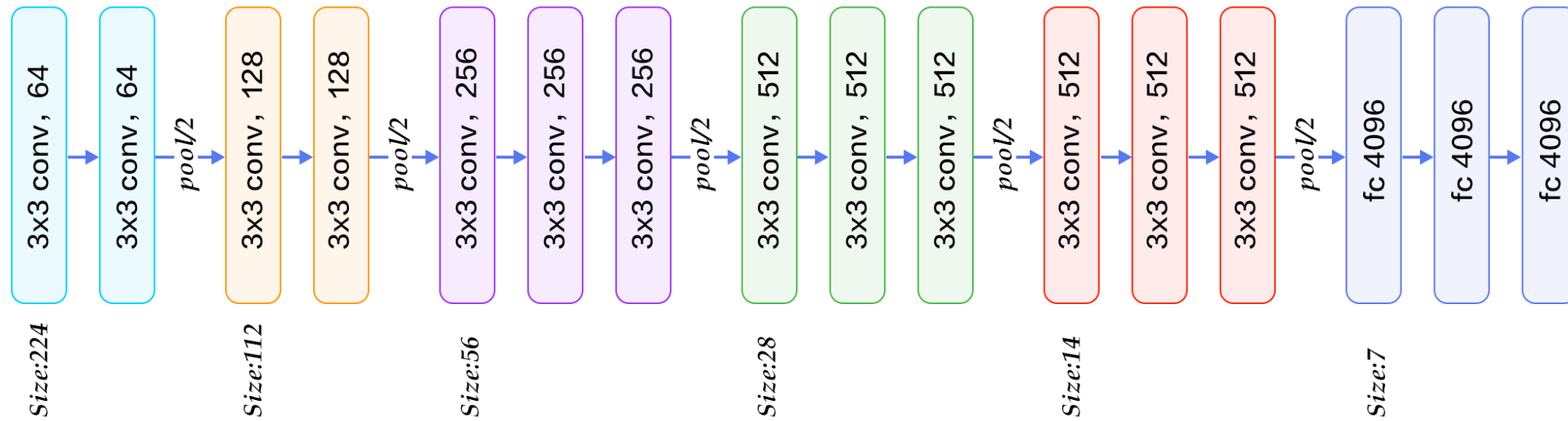
- Attached multiple classifier network during training that were discarded during inference



[5] Szegedy, C., et al. “Going Deeper with Convolutions”, *arXiv e-prints*, 2014. doi:10.48550/arXiv.1409.4842.

## “VGG16” K. Simonyan and A. Zisserman (2014) [4]

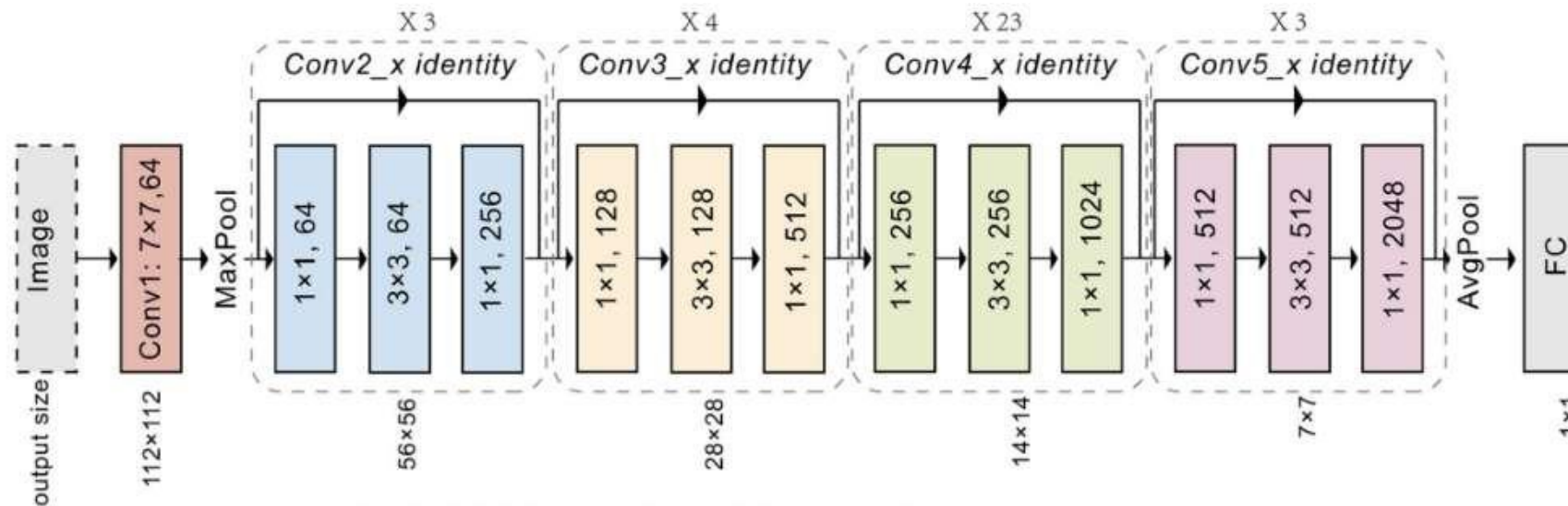
- Increased the depth of the CNN by using smaller convolutions
- This enables the network to extract more complex hierarchical features
- Used multiple **3x3** convolutions instead of larger (**5x5** or **7x7**) convolutions, while introducing more non-linearity



[4] Simonyan, K. and Zisserman, A., “Very Deep Convolutional Networks for Large-Scale Image Recognition”, *arXiv e-prints*, 2014. doi:10.48550/arXiv.1409.1556.

# “Residual Network / ResNet” K. He et al. (2015) [6]

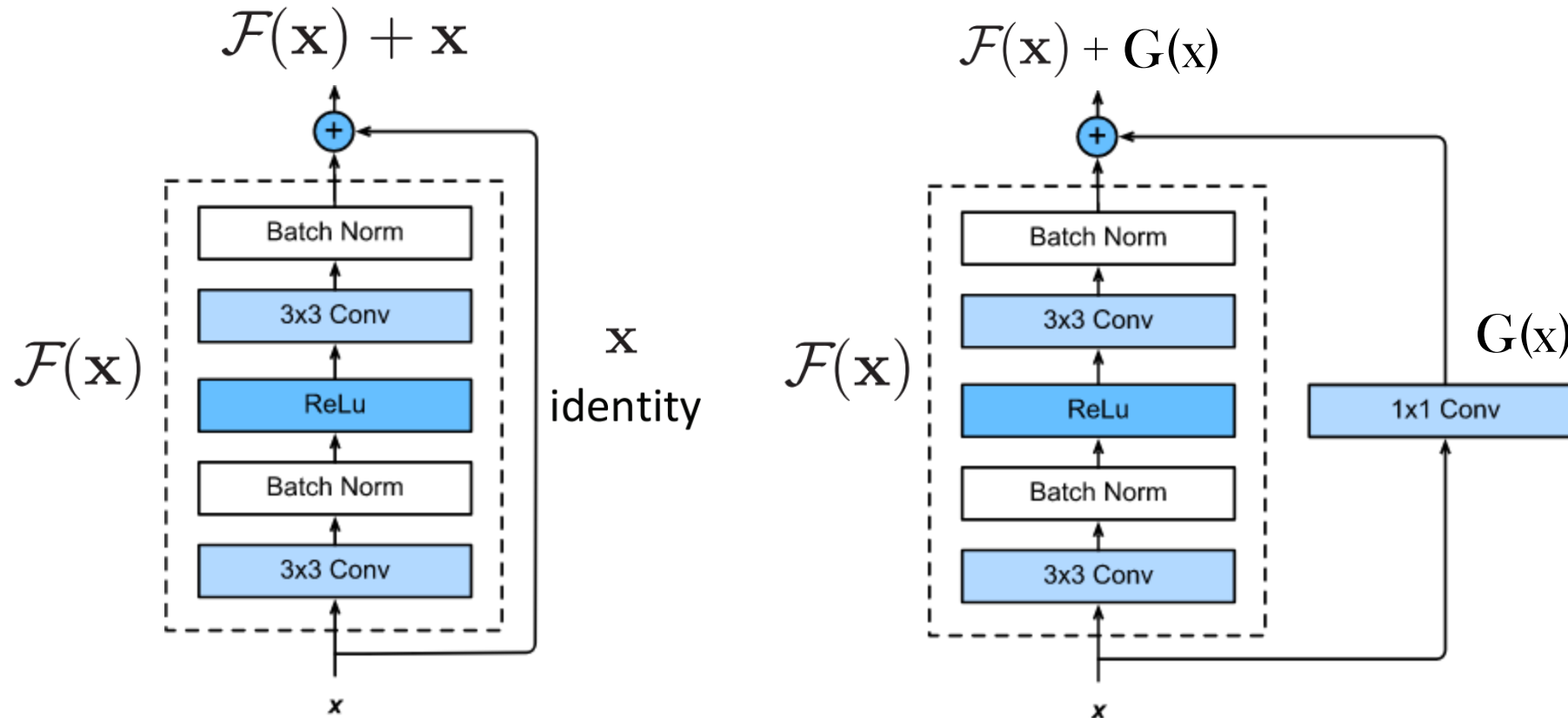
- Learn residual functions reference to layer inputs instead of unreferenced functions
- If we pretrain a smaller network and then we extend it with extra blocks it should have similar or better performance compared to the original network – **Identity mapping**



[6] He, K., Zhang, X., Ren, S., and Sun, J., “Deep Residual Learning for Image Recognition”, *arXiv e-prints*, 2015. doi:10.48550/arXiv.1512.03385.

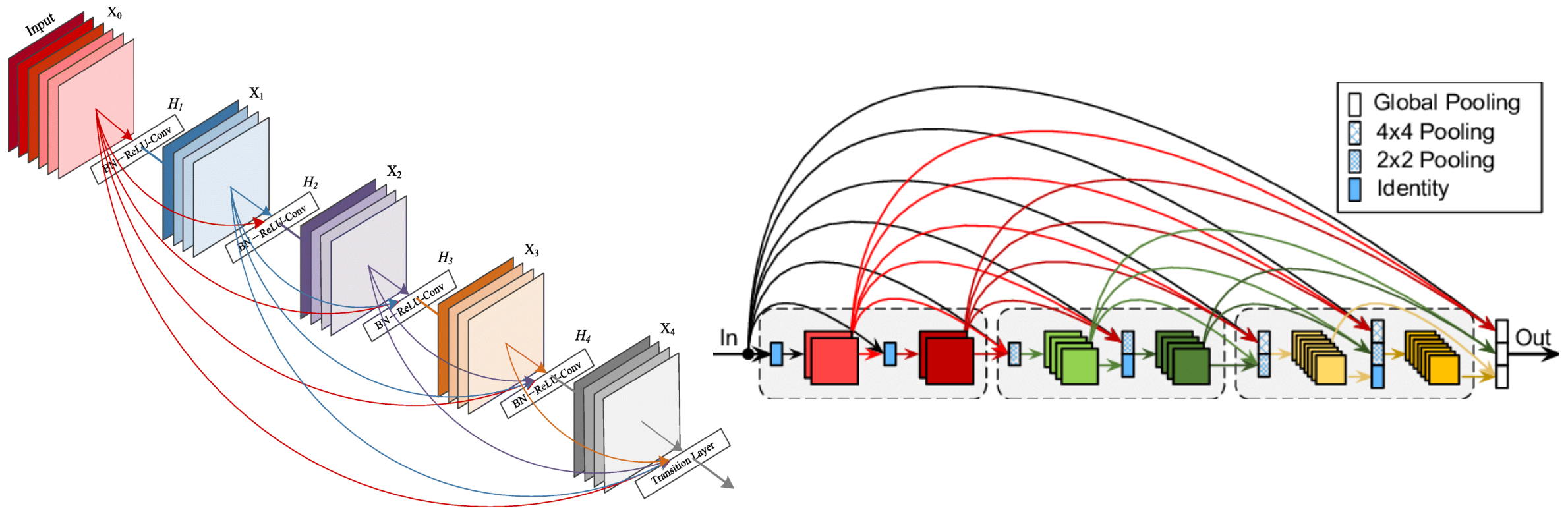


# “Residual Network / ResNet” K. He et al. (2015) [6]



[6] He, K., Zhang, X., Ren, S., and Sun, J., “Deep Residual Learning for Image Recognition”, <i>arXiv e-prints</i>, 2015. doi:10.48550/arXiv.1512.03385.

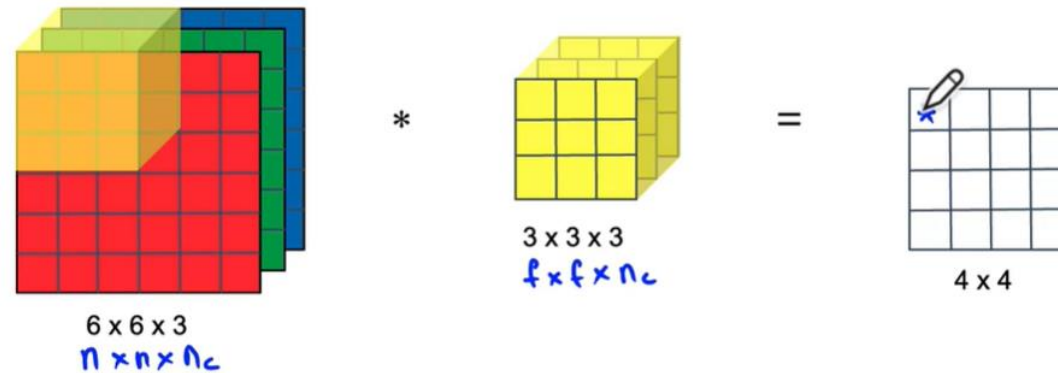
## “DenseNet” G. Huang et al. (2016) [7]



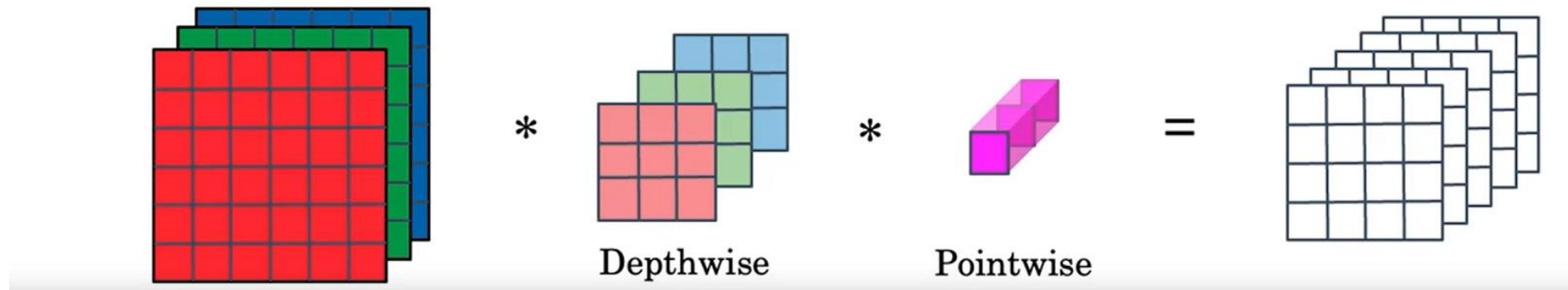
[7] Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q., “Densely Connected Convolutional Networks”, *arXiv e-prints*, 2016. doi:10.48550/arXiv.1608.06993.

## “MobileNet” A. G. Howard et al. (2017) [8]

### Normal Convolution

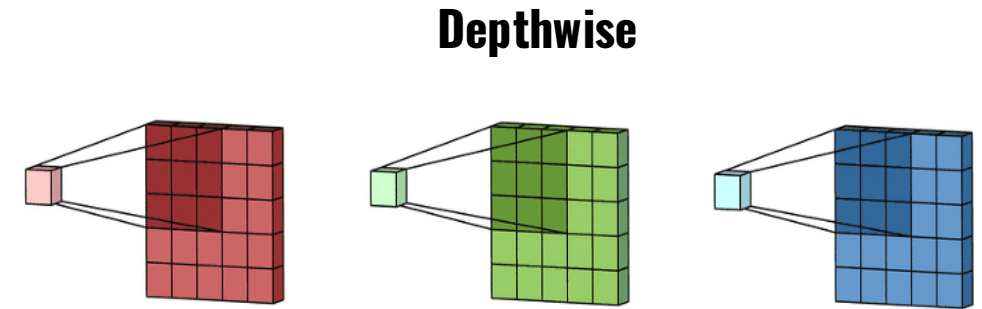
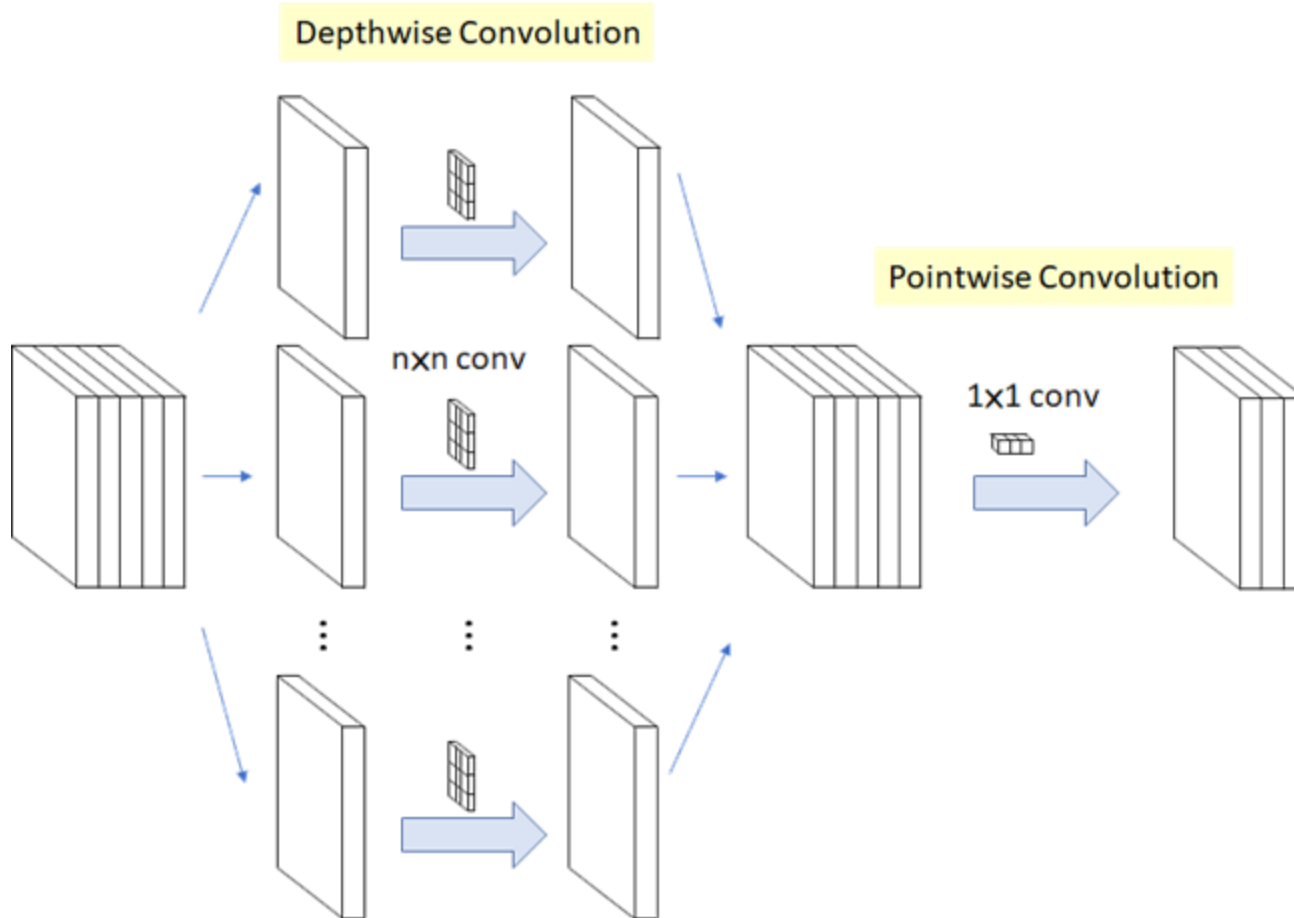


### Depthwise Separable Convolution

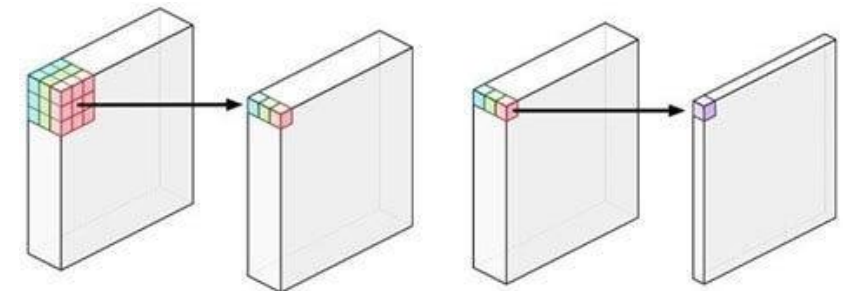


[8] Howard, A. G., et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”, <i>arXiv e-prints</i>, 2017. doi:10.48550/arXiv.1704.04861.

## “MobileNet” A. G. Howard et al. (2017) [8]



### Pointwise



a) depthwise convolution

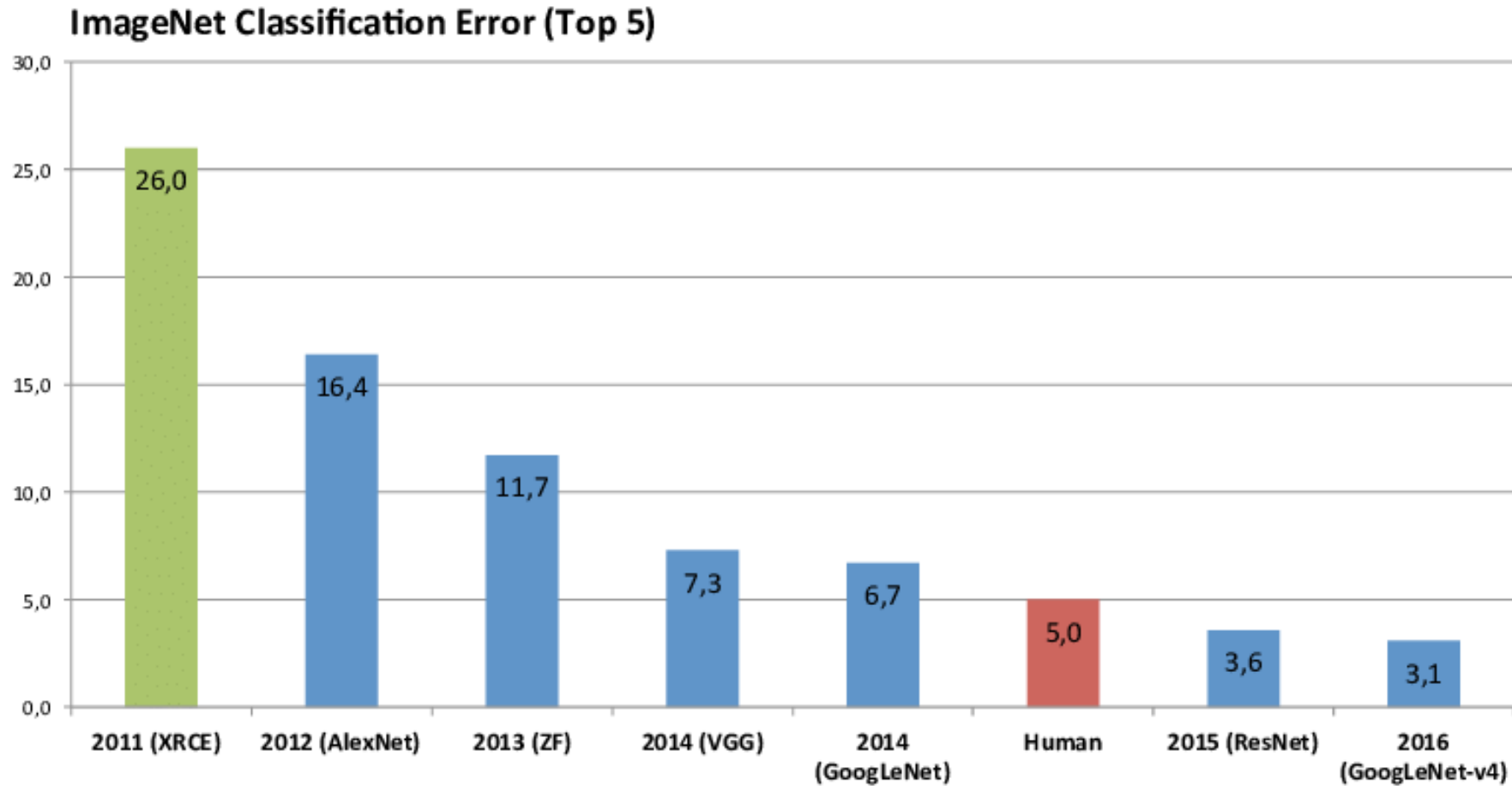
b) pointwise convolution

[8] Howard, A. G., et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”, <i>arXiv e-prints</i>, 2017. doi:10.48550/arXiv.1704.04861.

# More recent CNN architectures

- **ZfNet**
  - **Xception**
  - **InceptionV2, V3, V4**
  - **Inception-ResNet**
  - **MobileNet V3**
  - **FractalNet**
  - **WideResNet**
  - **PyramidalNet**
  - **Residual Attention Net**
  - **EfficientNet**
  - **Etc**
- 
- Currently, there are many models using the Transformer architecture (we will explore this later)

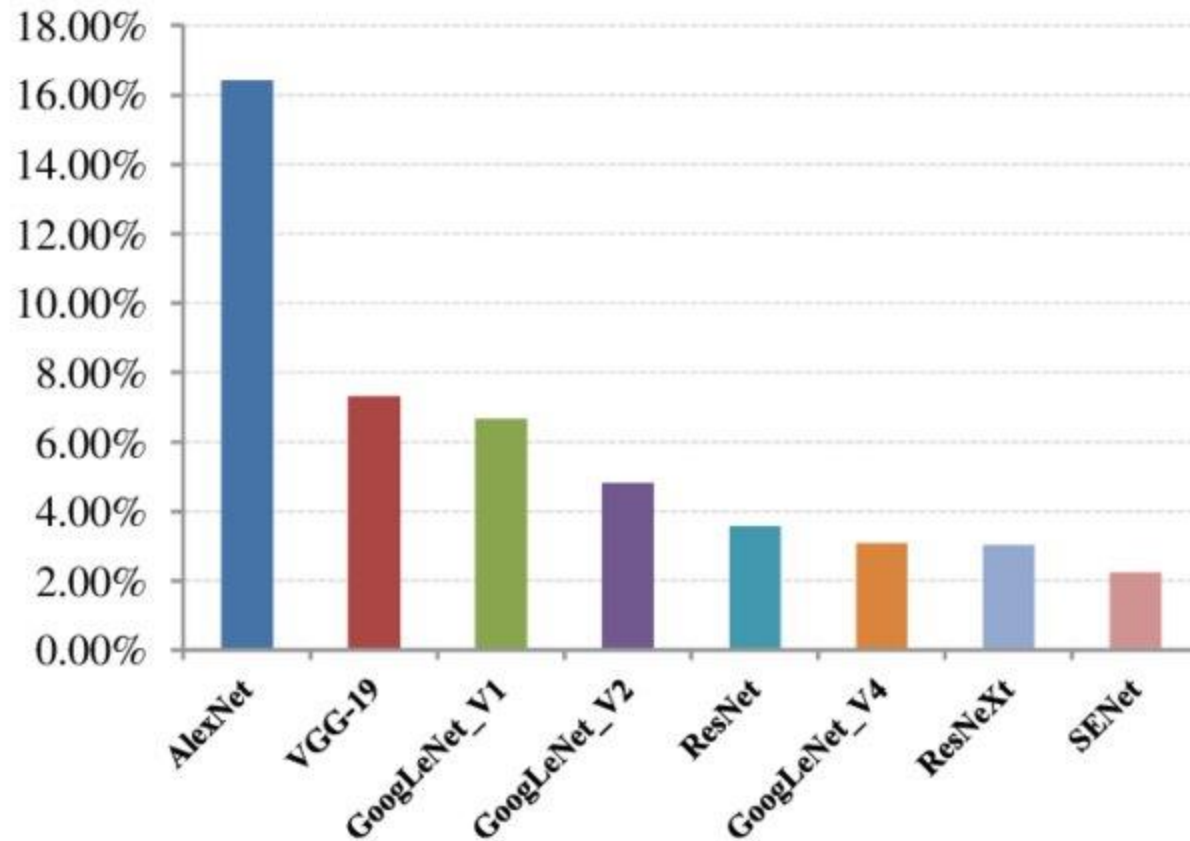
# ImageNet – Architectures Top 5 error (2016)





# ImageNet – Architectures Top 5 error (2019)

Top-5 error rate



- **More recently:**  
<https://paperswithcode.com/sota/image-classification-on-imagenet>
- **ImageNet website:**  
<https://www.image-net.org/>

# Lecture 4.



# Transfer Learning

---

Budapest, 7<sup>th</sup> March 2025

**1** CNN Architectures

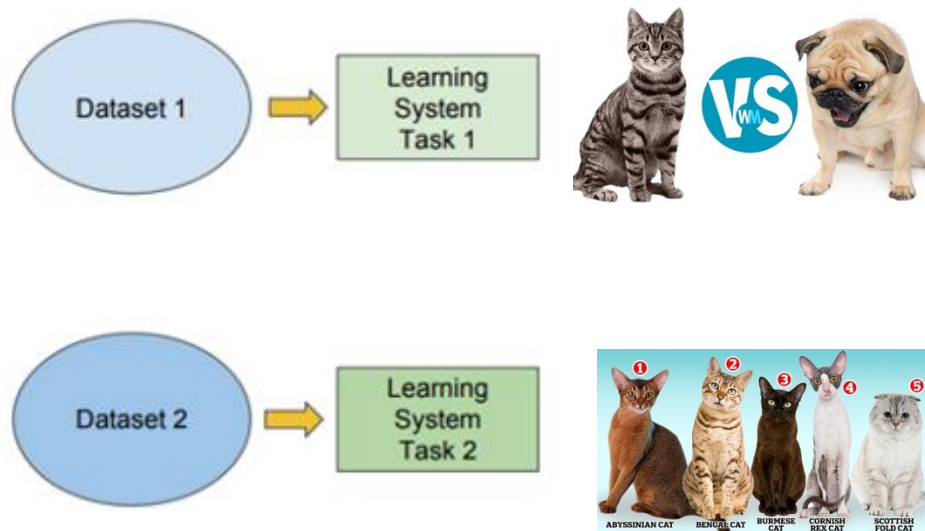
**2** Transfer Learning

**3** Autoencoders

# Traditional ML vs Transfer Learning

## Traditional ML

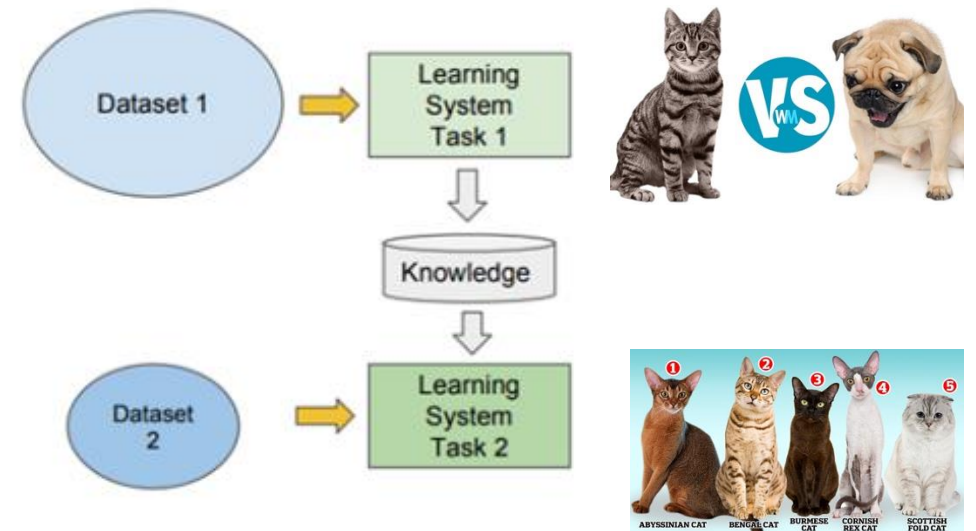
- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



vs

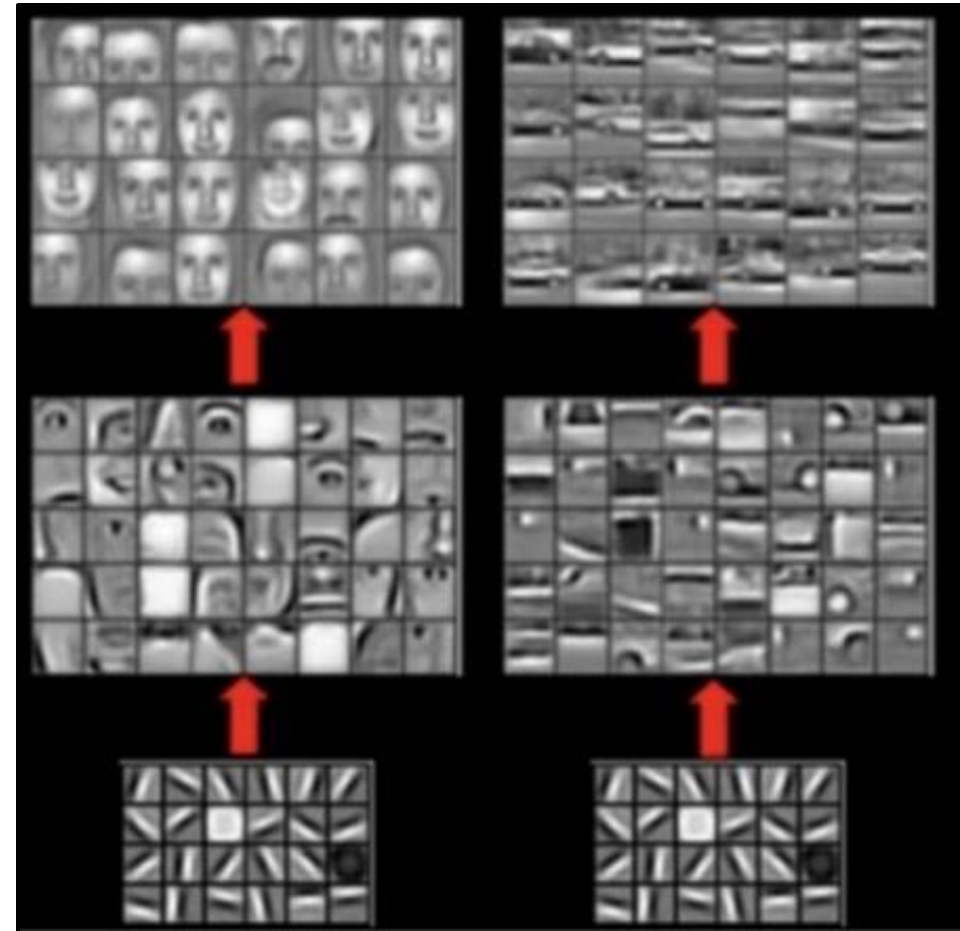
## Transfer Learning

- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data

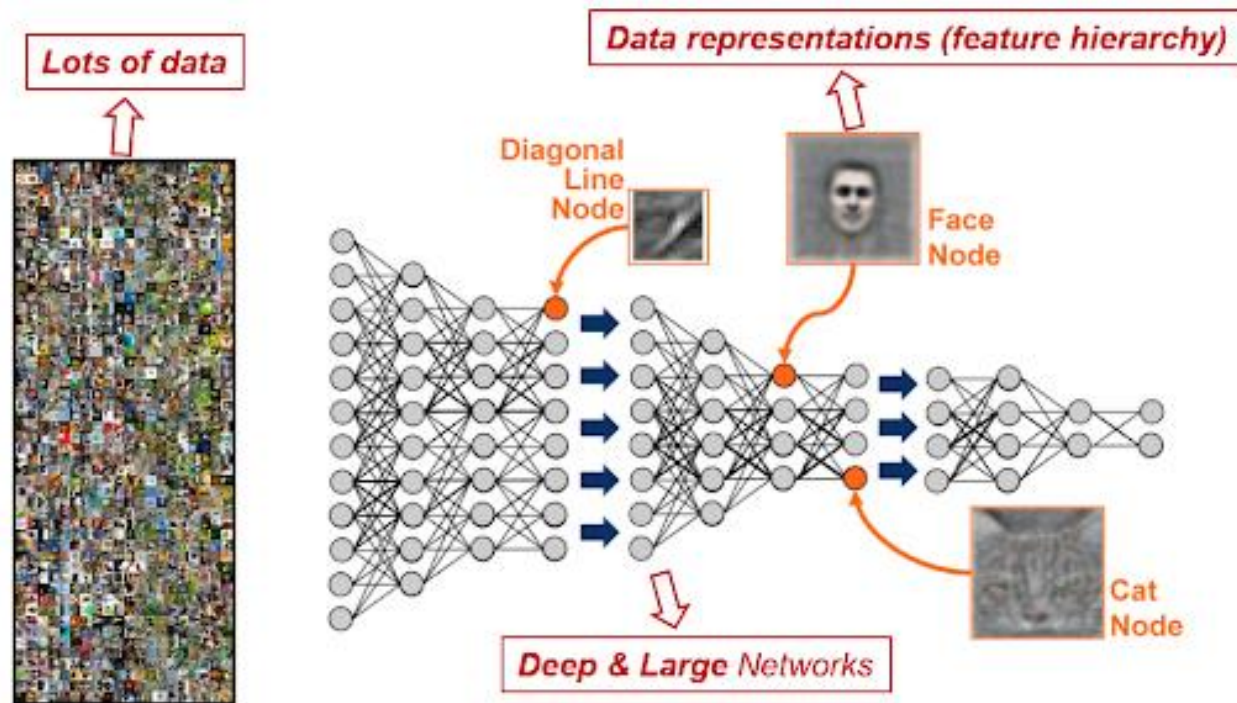


# Learned / Transferrable features

- The characteristics are built up in a pyramidal way
- Initial edge filtering with different orientation, "color" and cut-off values
- Simpler shapes, forms
- Part components (nose, eye, wheel,...)
- Faces, cars

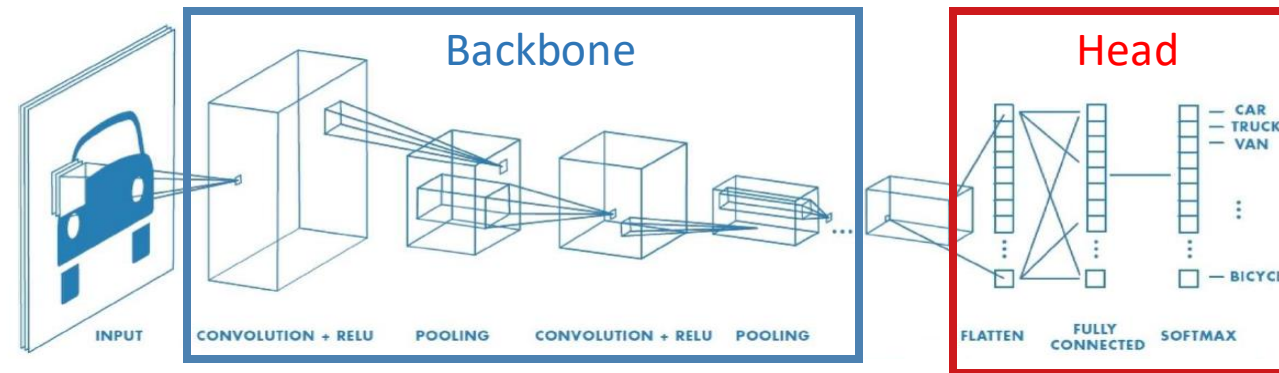


# Learned / Transferrable features

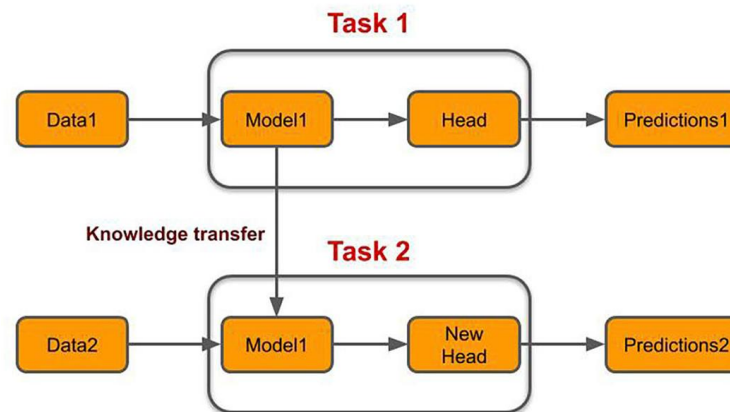




# Backbone vs head

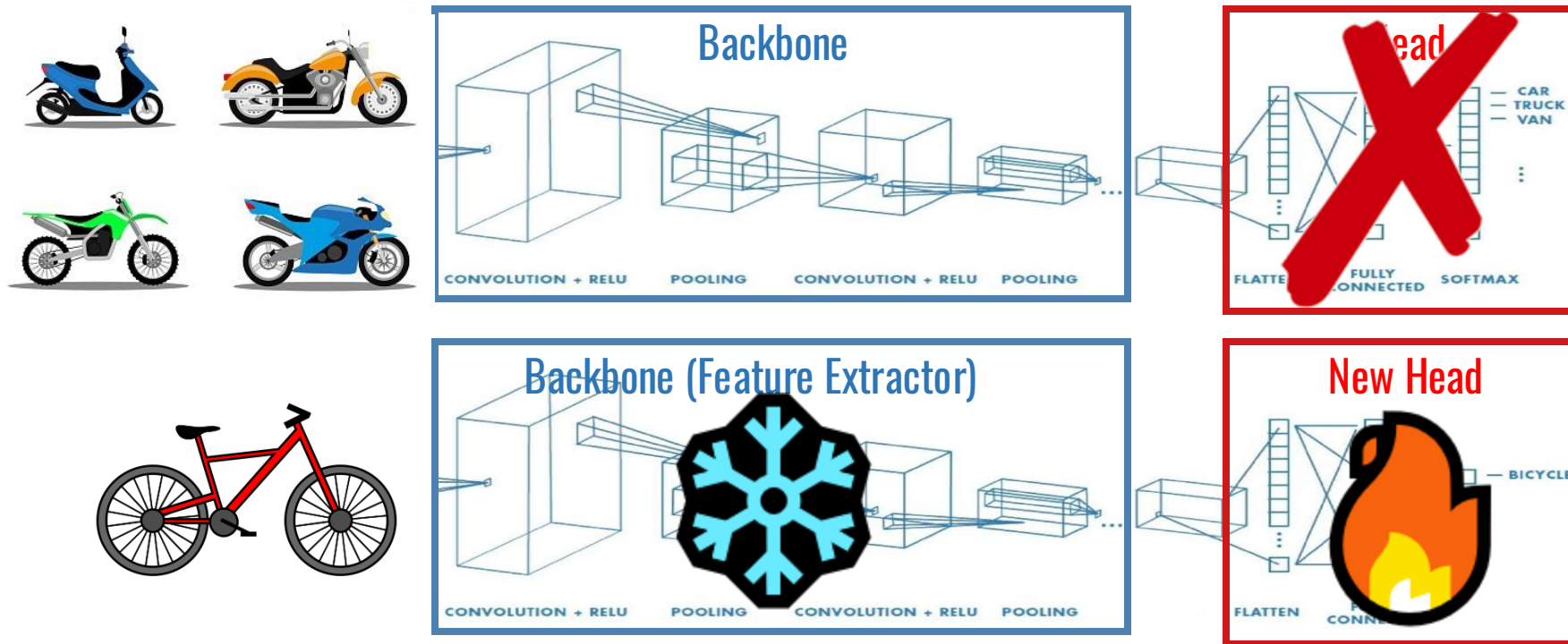


## Transfer Learning



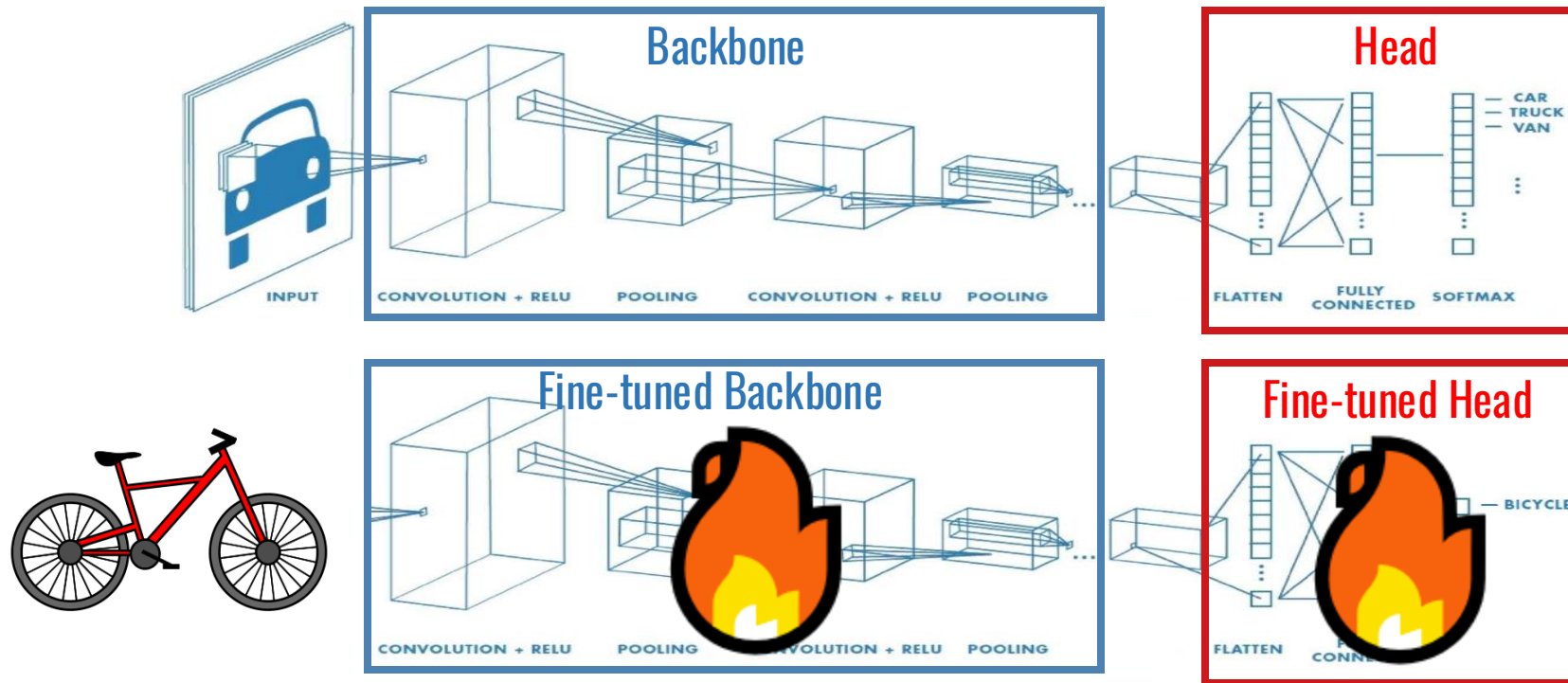
# Feature extraction

- We initialize the network with pre-trained weights from a pre-trained model. **We freeze all the weights** from the backbone, and we replace the head with a new one, initialized with random weights. We only train the new head, and not the backbone.



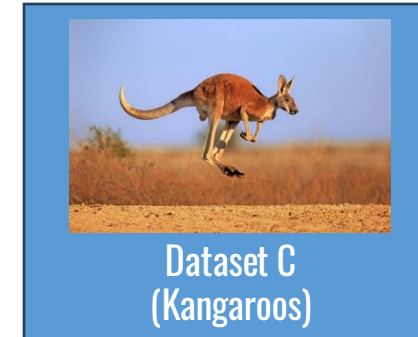
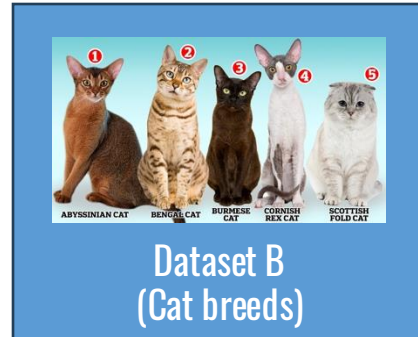
# Fine-tuning

- Instead of randomly initializing the weights for training, we initialize the network with pretrained weights from a pretrained model trained on a bigger dataset such as ImageNet (1M images and 1K class) and then we can fine-tune (train) the whole network.





# Performance comparison



Model A  
(trained on dataset A)

Model B  
(Using Model A for Feature  
Extraction)

Good performance;  
Fast Training;  
Less data needed.

Model D  
(Using Model A for Feature  
Extraction)

Worse  
performance

Model C  
(Fine-Tuned from Model A)

Good performance;  
Slower Training;  
More data needed.

Model E  
(Fine-Tuned from Model A)

Good performance;  
Slow Training;  
More data needed.

# When to use Transfer Learning?

- Task A and B have the same input  $x$
- When you have a lot of data for the problem you are transferring from (A) and few data for the problem you are transferring to (B)
- Low level features from A could be helpful for learning B
- Faster training. Use pre-trained weights as initialization point whether than randomly initializing weights

COCO Base Classes 1-40

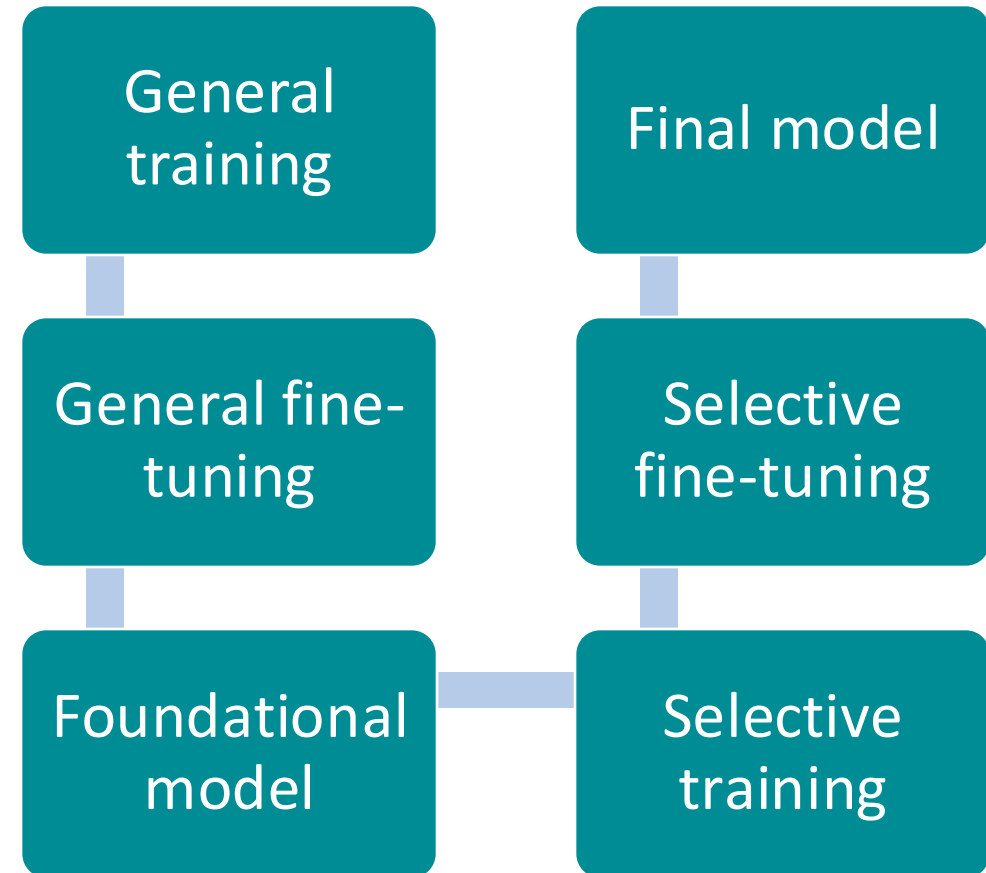


COCO Incremental Classes 41-80



# Deep Learning common steps - Parameter learning

- General
  - large dataset
  - Training
  - Fine-tuning
  - Foundational model



# Lecture 4.



# Autoencoders

---

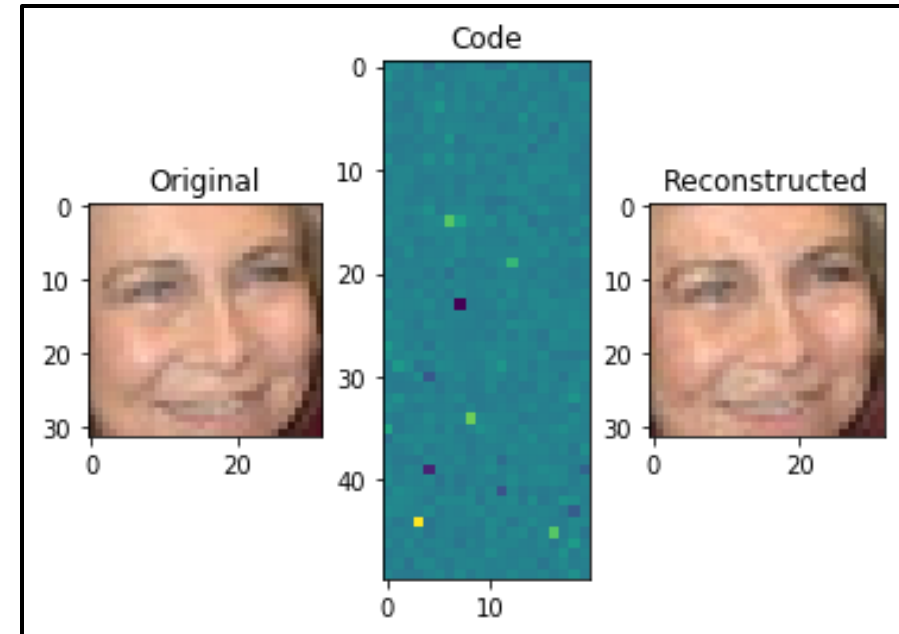
Budapest, 7<sup>th</sup> March 2025

**1** CNN Architectures

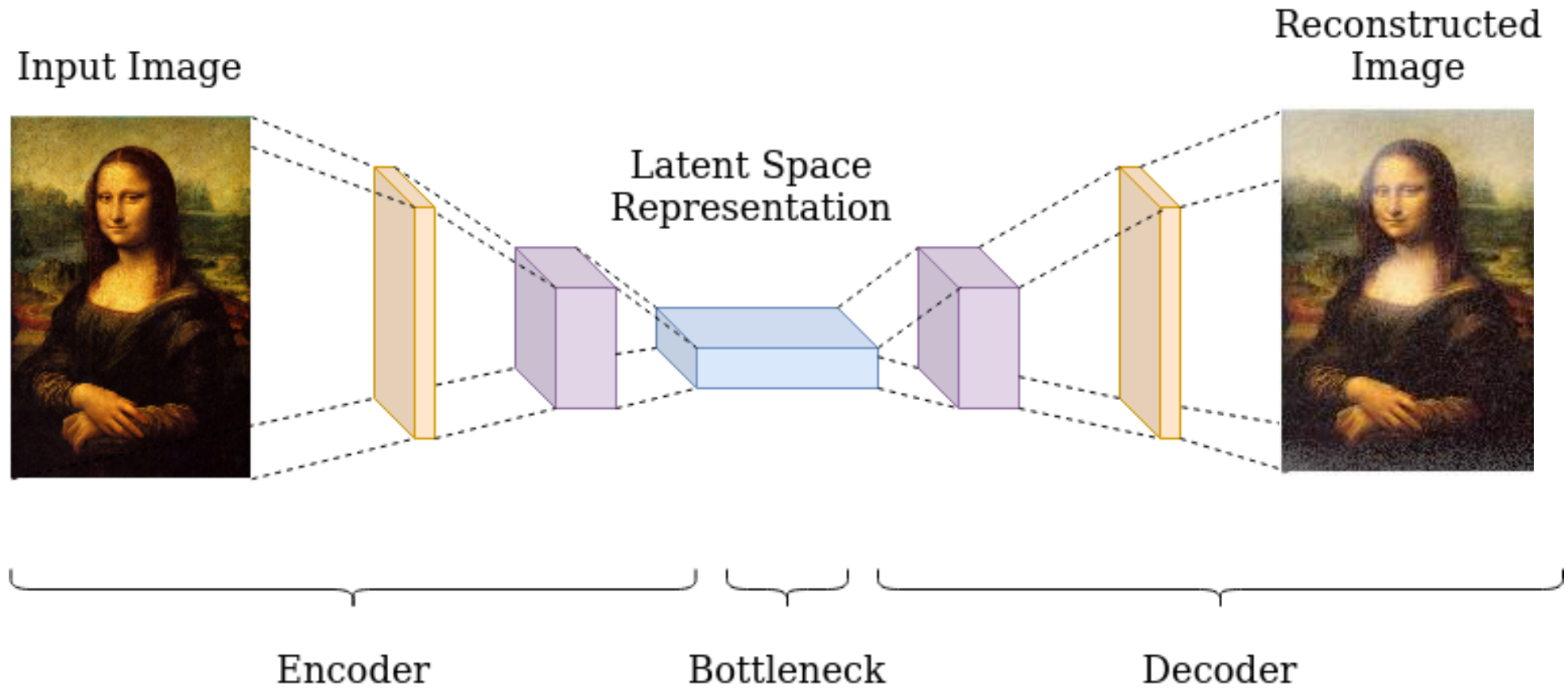
**2** Transfer Learning

**3** Autoencoders

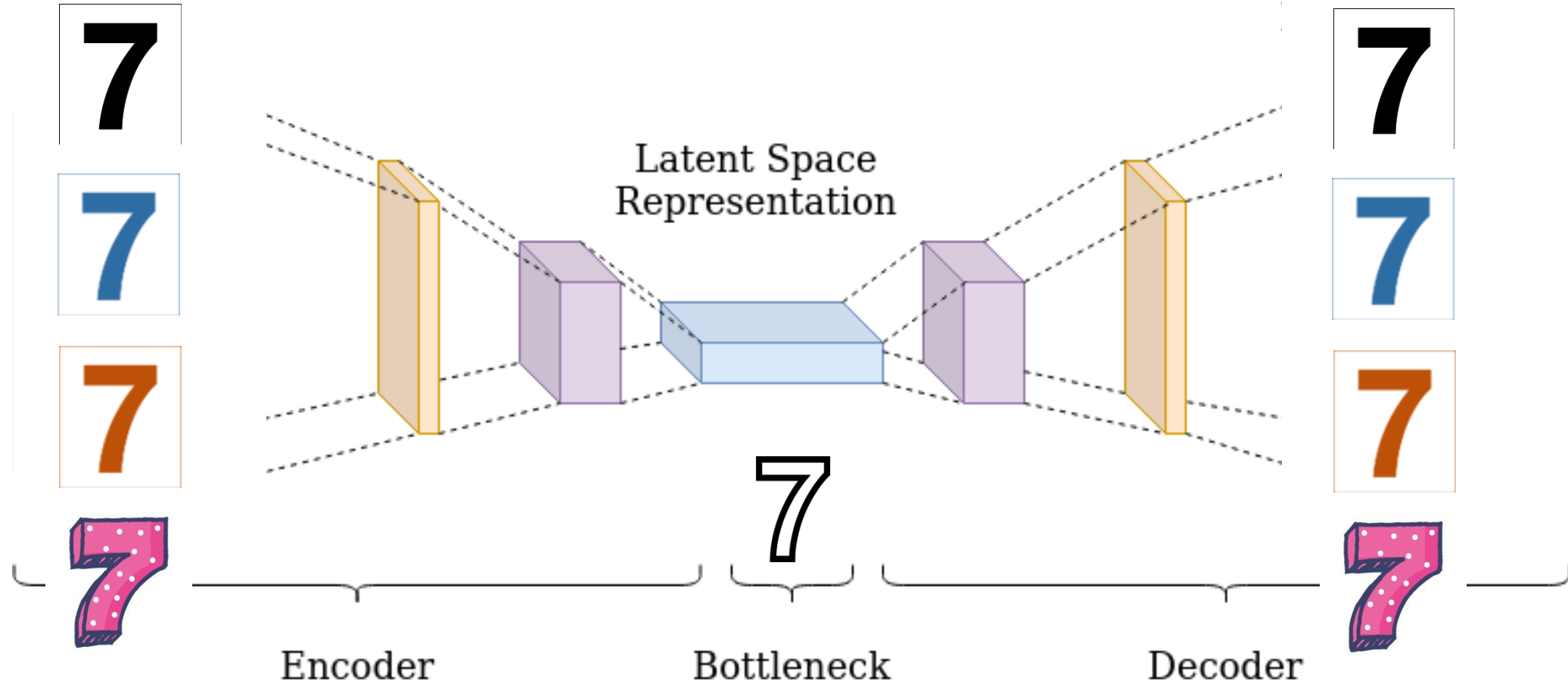
# Feature learning: Autoencoders



# Reconstruction Task



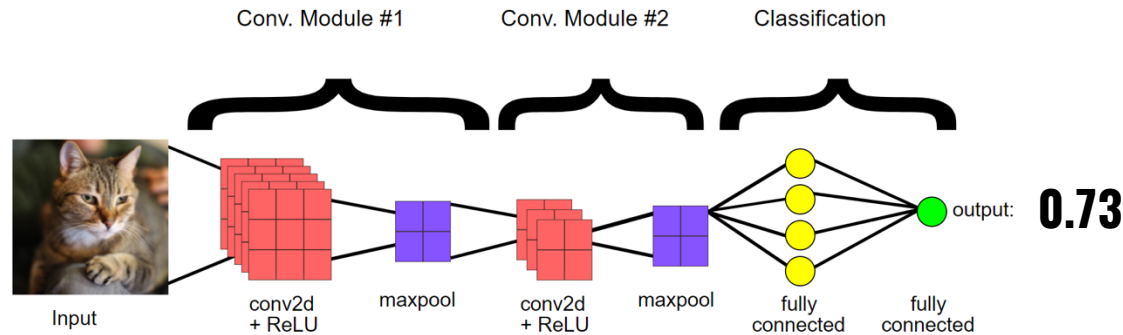
## Reconstruction Task





# Image Classification: Supervised Learning (labelled data is needed)

Data:  
**X = image**  
**Y = label / target (1)**

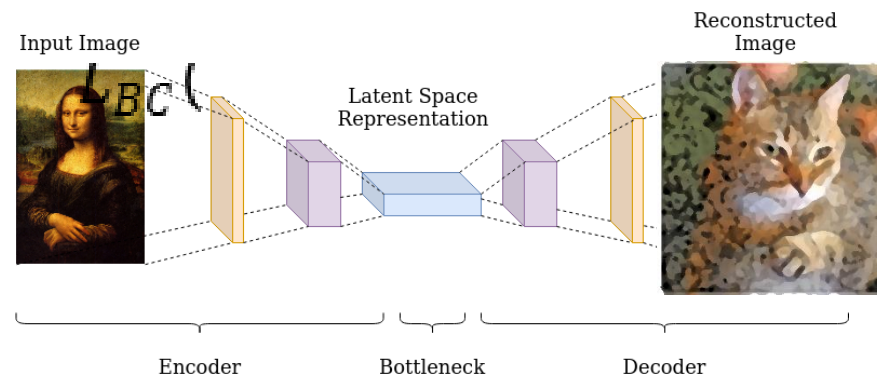


Loss between prediction and target

$$\begin{aligned}
 L_{BC}(h(x), y) &= -y \log h(x) - (1 - y) \log(1 - h(x)) \\
 L_{BC}(0.73, 1) &= -1 \log 0.73 - (1 - 1) \log(1 - 0.73) \\
 L_{BC}(0.73, 1) &= -1 * -0.13667714 - (0) \log(0.23) \\
 L_{BC}(0.73, 1) &= 0.13667714 \\
 L_{BC}(0.73, 1) &= 0.14
 \end{aligned}$$

Reconstruction: Self-Supervised Learning (no need for labelled data)

Data:  
**X = image**  
**Y = X**



Loss between prediction and input

$$\left( \text{Input Image} - \text{Reconstructed Image} \right)^{**2} = 0.2$$

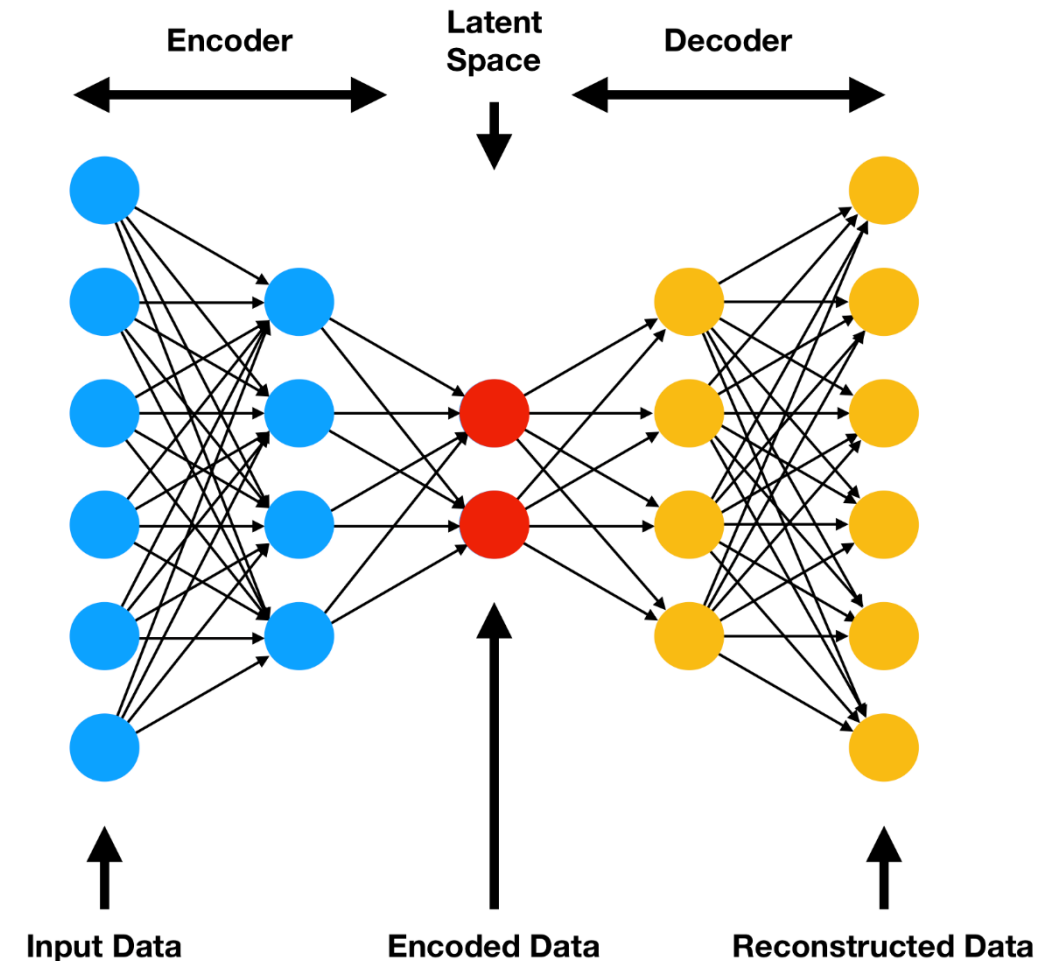


# Autoencoder structure

**Autoencoder** is a machine learning algorithm that **learns a compressed representation of an input**. Because it requires no label it is an unsupervised method<sup>1</sup>. However, the **output is a reconstruction of the input**, thus it is also considered as a self-supervised method.

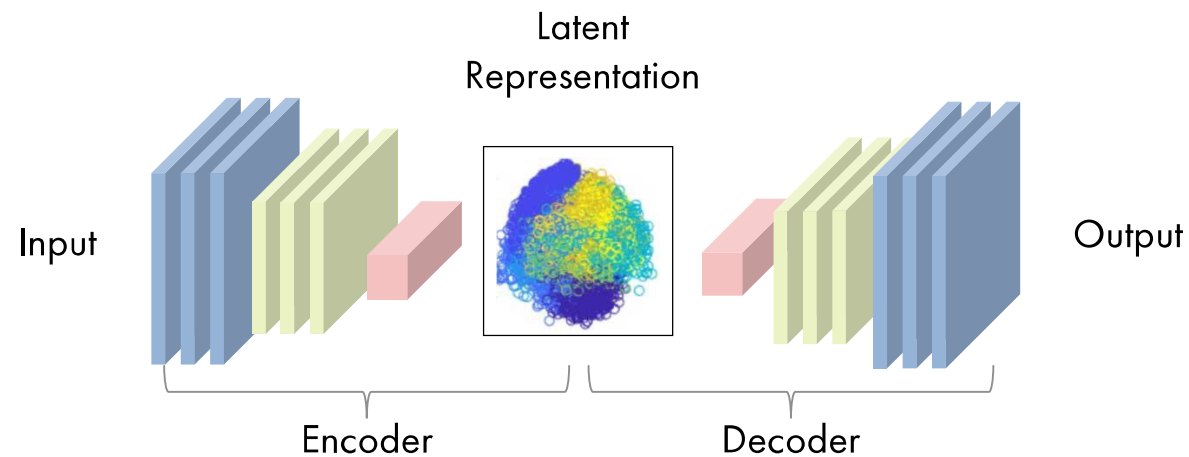
An autoencoder consists of three parts:

- **Encoder**: the part of the network that compresses the input into a latent-space representation of reduced dimension. It can be represented by an encoding function  $h=f(x)$ .
- **Latent space (bottleneck/code)**: the part of the network which contains the reduced representation of the input.
- **Decoder**: the part that aims to reconstruct the input from the latent space representation. It has a similar structure to the encoder and can be represented by a decoding function  $r=g(h)$ .



## Autoencoder structure

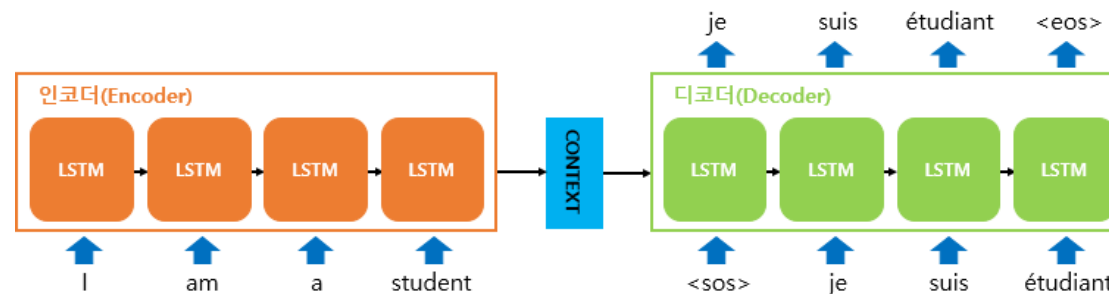
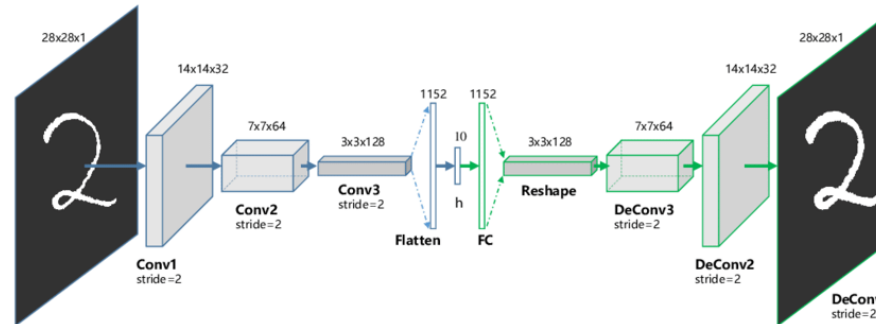
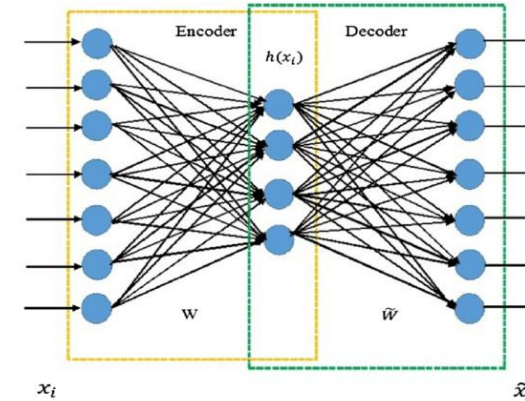
- Autoencoders are learned automatically from data examples, **using the input as output target**, making them easy to train. However, they do not generalize to new data.
- If the only purpose of autoencoders was to copy the input to the output, they would be useless. The main idea is that by training the autoencoder to copy the input to the output, **the latent representation will learn the most important features of the input**
- The bottleneck is designed in such a way that the maximum information possessed by an input is captured in it, therefore, the bottleneck **forms a knowledge-representation of the input**.



# Autoencoder structure

The input and output layers of the autoencoder can be formed with:

- Feed Forward neural networks (Dense / Fully connected layers)
- Convolutional neural networks
- Recurrent neural networks
- Others (Restricted Boltzmann Machine, ...)

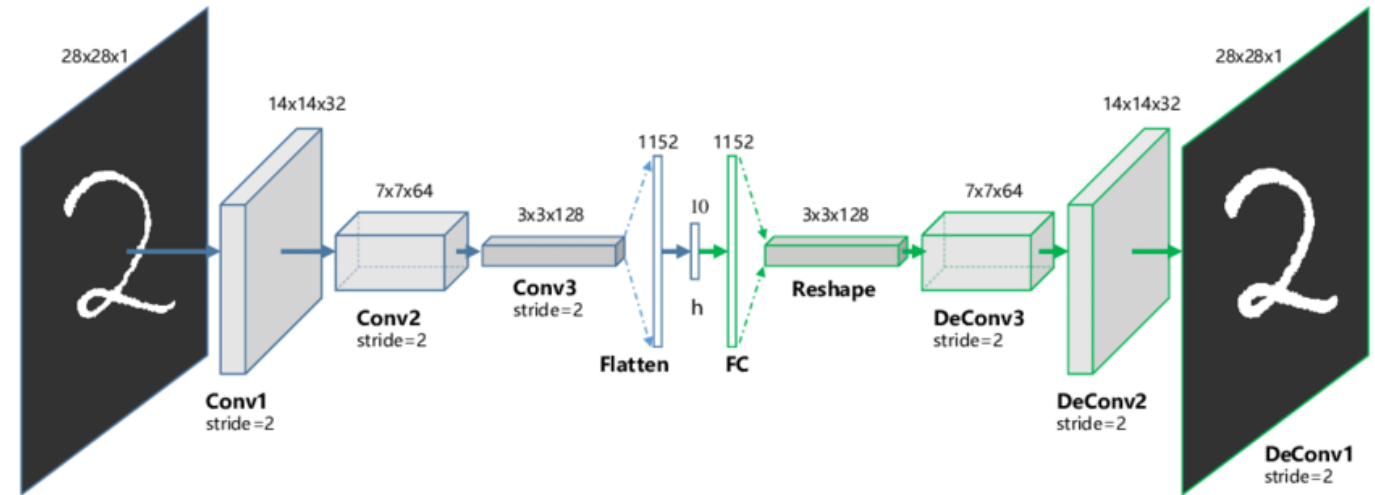


## Autoencoder Types

- Convolutional Autoencoder
- Denoising Autoencoder
- Sparse Autoencoder
- Variational Autoencoder
- and more

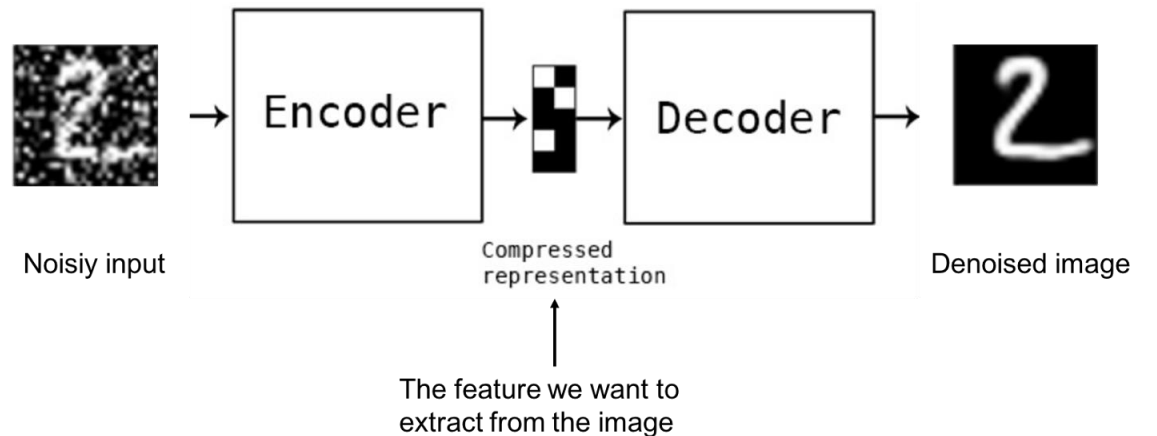
# Convolutional Autoencoders

- Specifically designed for image data.
- They employ convolutional layers in both the **encoder** and **decoder** parts of the network.
- This architecture allows them to capture spatial dependencies and hierarchical features effectively.
- The reconstruction of the input image is often blurry and of lower quality due to compression during which information is lost.



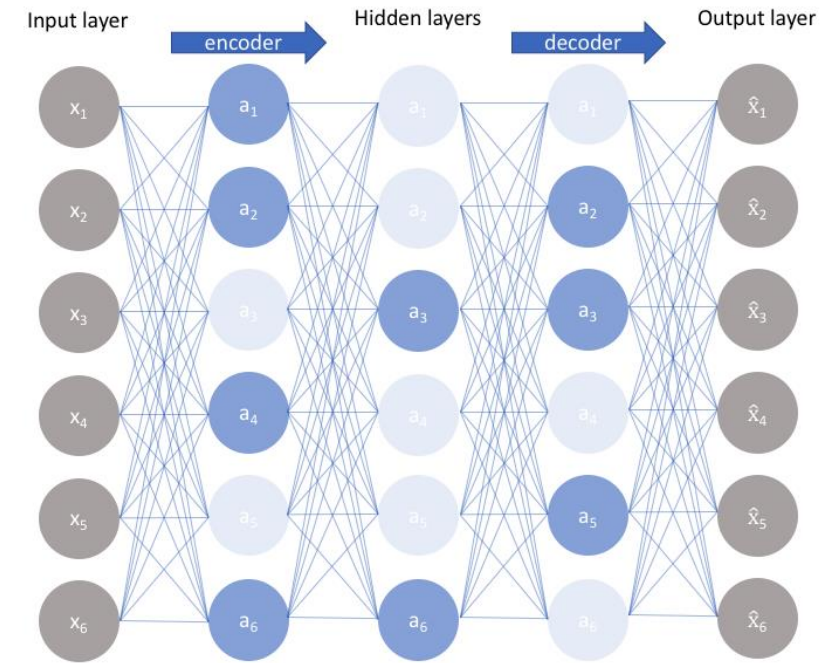
## Denoising Autoencoders

- Designed to remove noise from input data.
- Noise can be added to the input, by modifying different parts of the input:
  - Gaussian noise; Salt-and-Pepper noise; Random Masking; Perturbation noise; etc
- During training, a noisy version of the input is provided, and the network learns to reconstruct the clean, noise-free data.
- This encourages the model to capture robust and meaningful features while filtering out irrelevant or noisy information.



# Sparse Autoencoders

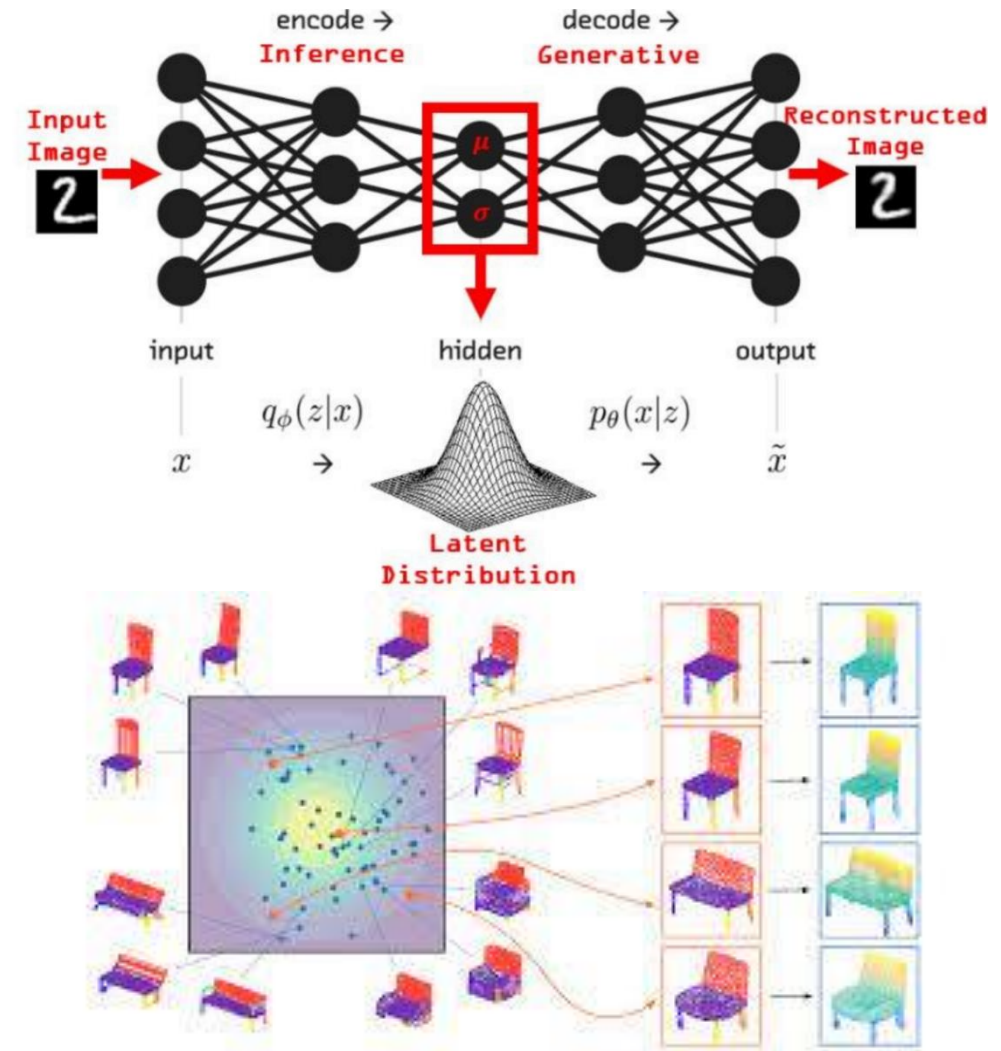
- Designed to impose sparsity constraints on the representations learned by the autoencoder.
- Sparsity means that only a small subset of neurons in the hidden layer is active at a time
- They take the highest activation values in the hidden layer and zero out the rest of the hidden nodes. This prevents autoencoders to use all the hidden nodes at a time and forcing only a reduced number of hidden nodes to be used.
- Sparse autoencoders have a sparsity penalty, a value close to zero but not exactly zero. Sparsity penalty is applied on the hidden layer and to the reconstruction error. This prevents overfitting.





## Variational Autoencoders

- Variational Autoencoders (VAE) are generative autoencoders that **can generate new instances**
- Different from traditional autoencoders, instead of learning a mapping function for each input to a number in the latent space, they **learn a probability distribution representation of the data**.
- By sampling from the learned probability distribution, a point within it is selected, which represents a potential encoding of the input. This randomness during sampling is what makes VAEs generative, enabling to generate diverse outputs consistent with the input's learned distribution.
- <https://xnought.github.io/vae-explainer/>





# Applications

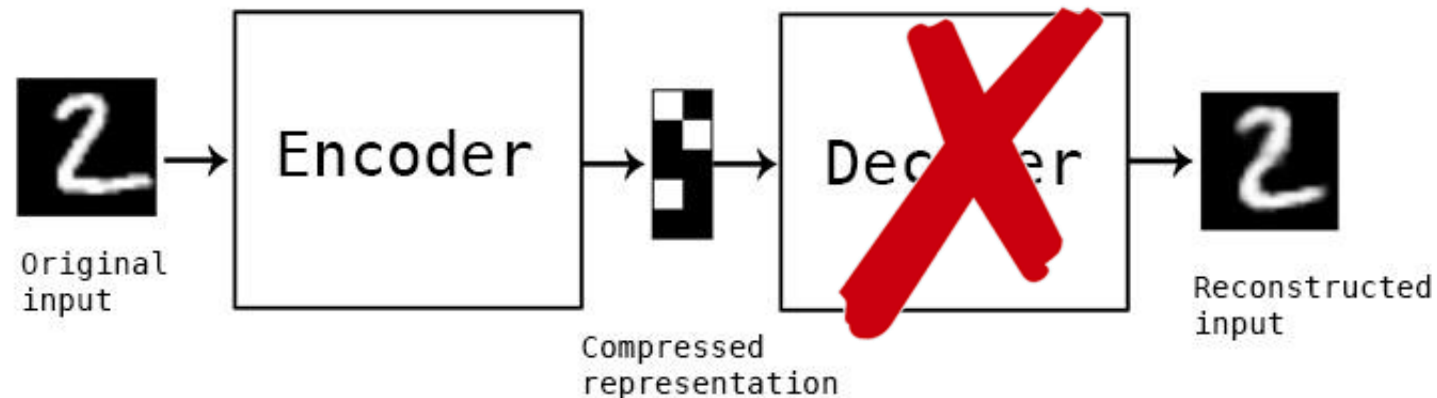
## (Not for) Data Compression

- Although autoencoders can compress the input into a latent representation, it is **usually not used for data compression**. The reasons are:
  - **Lossy compression**: The output of the autoencoder is not the same as the input, it is a close but degraded representation because information is lost in the latent representation.
  - **Data-specific**: Autoencoders are only able to meaningfully compress data like what they have been trained on. Since they learn features specific for the given training data. Hence, we can't expect an autoencoder trained on handwritten digits to compress landscape photos.

# Applications

## Dimensionality Reduction

- The autoencoders convert the input into a **reduced representation which is stored in the latent representation space**. This is where the information from the input has been compressed. Thus by **removing the decoder part**, an autoencoder can be used for dimensionality reduction with **the output being the latent representation vector**.

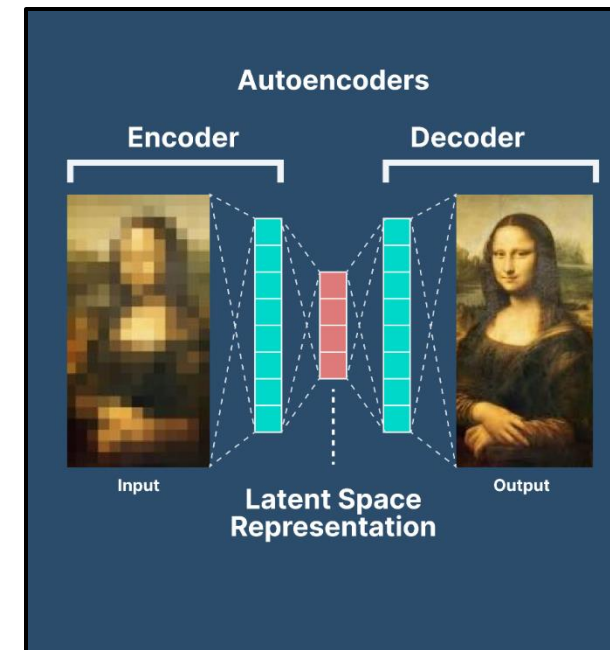
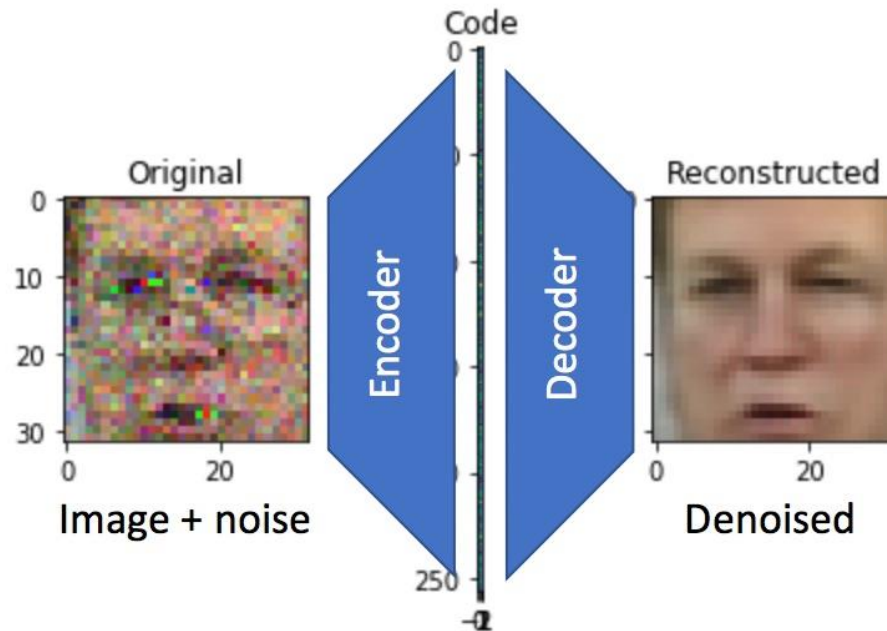


# Applications

## Image Denoising

- Example:

- <https://huggingface.co/spaces/Xintao/GFPGAN>
- <https://huggingface.co/spaces/aryadytm/photo-colorization>

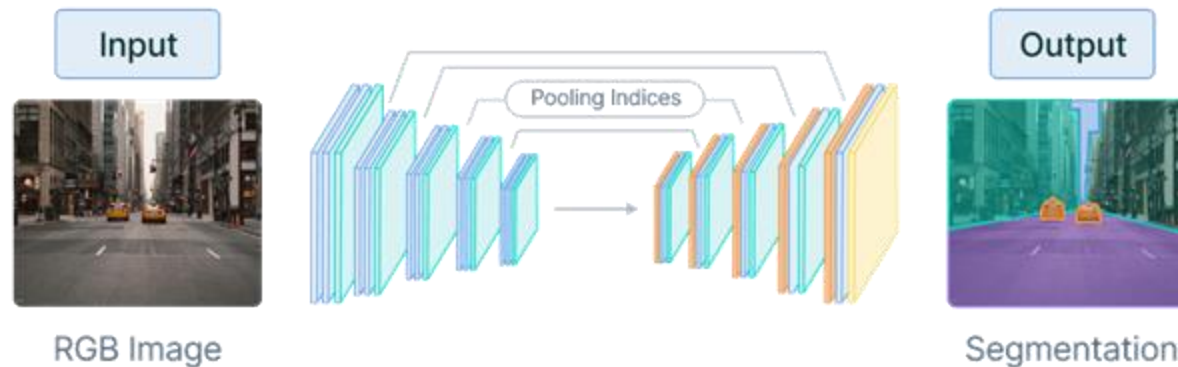


# Applications

## Image Segmentation

- Image segmentation is the process of partitioning an image into multiple segments each belonging to a class. The goal is to simplify and/or change the representation of an image by **grouping pixel values according to the class they belong**.

### Convolutional encoder-decoder

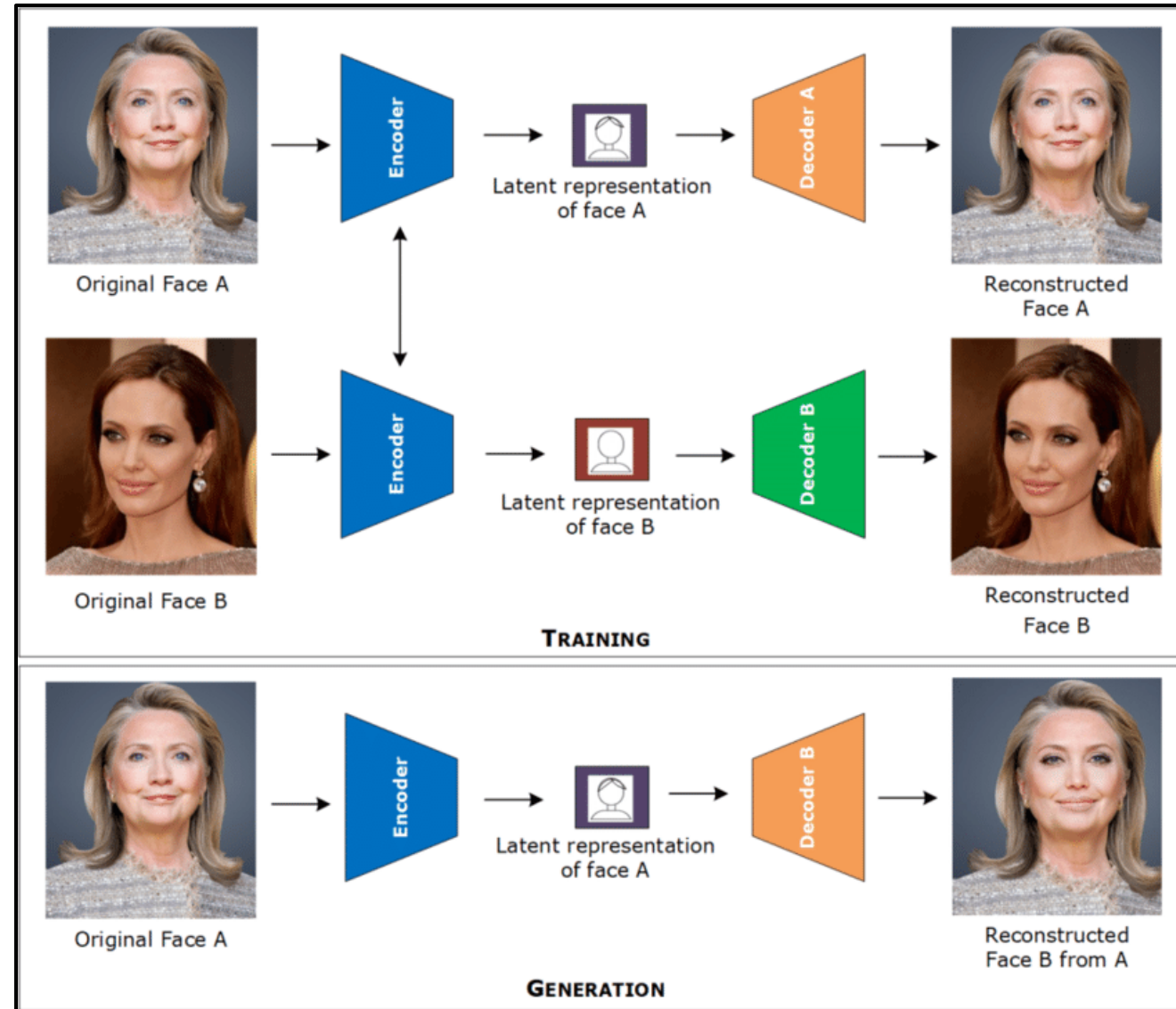


# Applications

## Image Generation

Encoder-Decoder swap

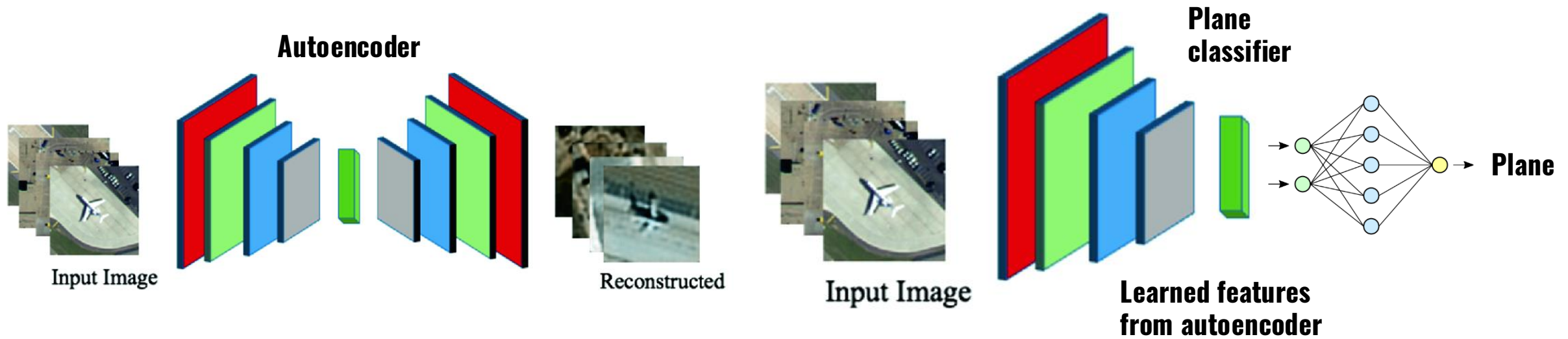
Latent Space Manipulation



# Applications

## Feature Extraction

- The Encoding part of Autoencoders helps to **learn important hidden features** present in the input data, in the process to reduce the reconstruction error. During encoding, a new set of combinations of original features is generated. **By removing the decoder, we can use the encoded features as input features for a network to use**, for example, in the classification task.



## Summary

- Different CNN hyperparameter choices result in different CNN architectures
- Transfer Learning is used when we want to utilize the learned features from a model A, that can be useful for a task that model B is trying to solve.
- We can retrain the whole model or freeze part of the model and only retrain a small part
- Autoencoders are unsupervised / self-supervised methods
- By reconstructing the input, they learn to encode the input into a lower dimensional latent space
- By removing the decoder from a trained autoencoder, the encoder can be used for feature learning

# Resources

## Books:

- Courville, Goodfellow, Bengio: Deep Learning  
Freely available: <https://www.deeplearningbook.org/>
- Zhang, Aston and Lipton, Zachary C. and Li, Mu and Smola, Alexander J.: Dive into Deep Learning  
Freely available: <https://d2l.ai/>

## Courses:

- Deep Learning specialization by Andrew NG
- <https://www.coursera.org/specializations/deep-learning>



# That's all for today!

