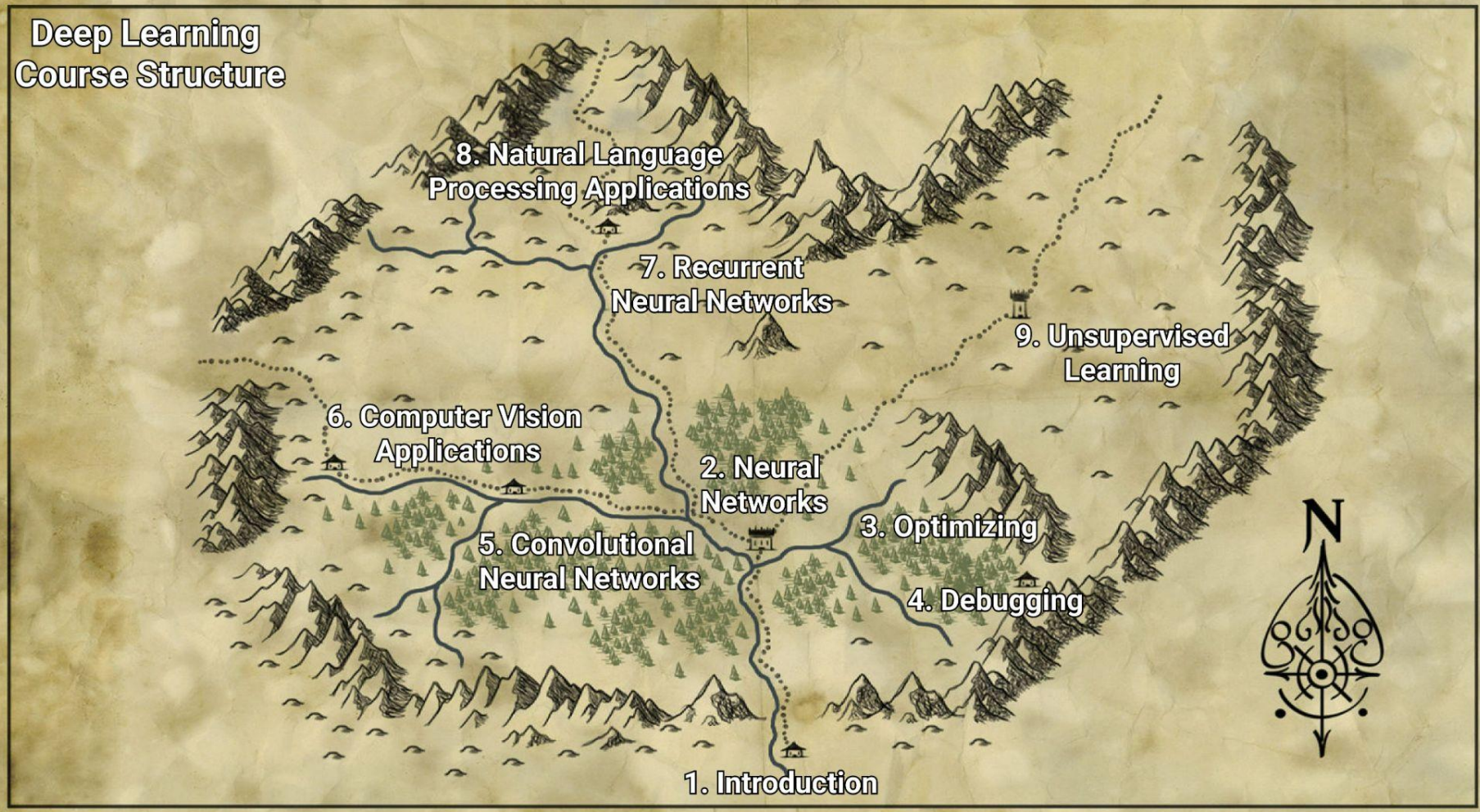
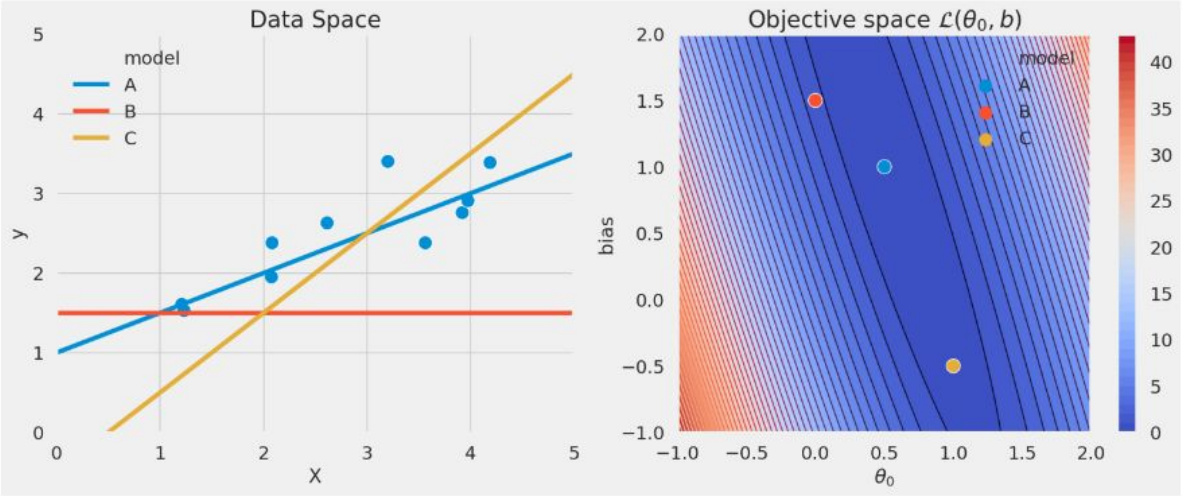
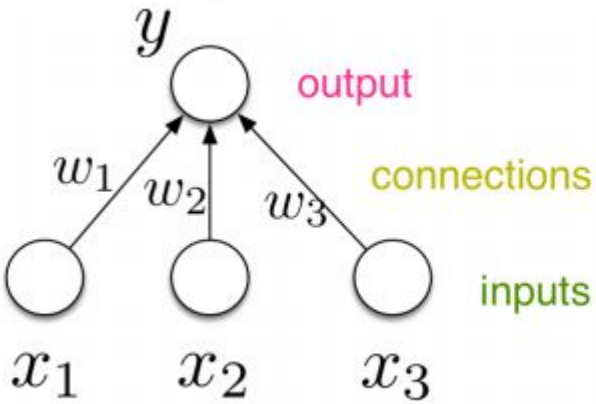
Deep Learning

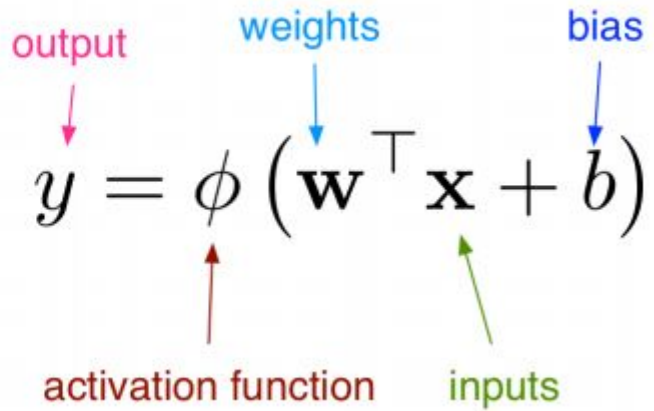
5. Convolutional Networks


Deep Learning
Course Structure
8. Natural Language
Processing Applications
7. Recurrent
Neural Networks
9. Unsupervised
Learning
6. Computer Vision
Applications
2. Neural
Networks
3. Optimizing
5. Convolutional
Neural Networks
4. Debugging
N
1. Introduction

Recap


Data Space
model
A
B
C
X
y
Objective space $\mathcal{L}(\theta_0, b)$
model
A
B
C
bias
θ_0

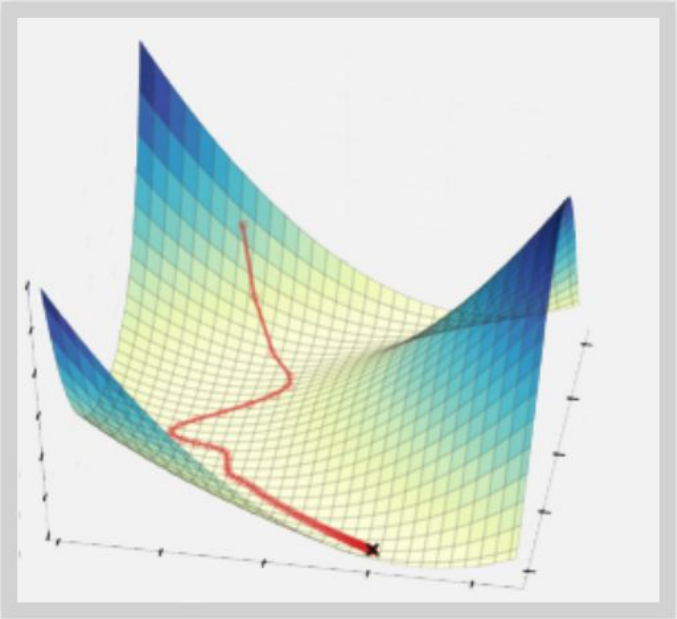

y
output
w_1
w_2
w_3
connections
inputs
x_1
x_2
x_3


output
weights
bias
$y = \phi\left(\mathbf{w}^\top \mathbf{x} + b\right)$
activation function
inputs

Gradient Descent

- Measure the error of our model using $\mathcal{L}(\mathcal{D}, \boldsymbol{\theta})$.
- Compute the gradients w.r.t. $\boldsymbol{\theta}$,
- Find incremental solutions that better explain the data using:

$$\boldsymbol{\theta}_{j+1} \leftarrow \boldsymbol{\theta}_j - \eta \nabla_{\boldsymbol{\theta}_j} \mathcal{L}$$

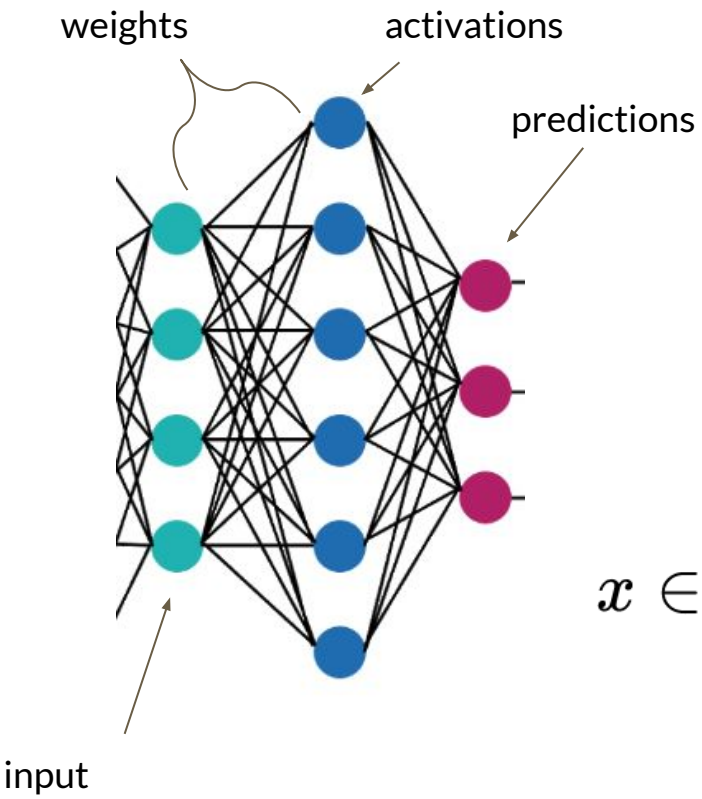


Stochastic Gradient Descent

Consider instead objective functions that are the sum of the losses -- $\mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \mathcal{L}_n(\boldsymbol{\theta})$, resulting in the update:

$$\boldsymbol{\theta}_{j+1} \leftarrow \boldsymbol{\theta}_j - \eta \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}_j} \mathcal{L}_n(\boldsymbol{\theta})$$

Recap: Fully connected networks



$$a = \sigma(W_1 x + b_1)$$

$$p = \sigma(W_2 a + b_2)$$

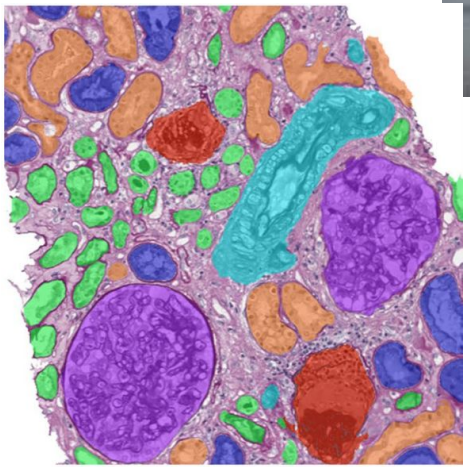
$$x \in \mathbb{R}^N \; W_1 \in \mathbb{R}^{M \times N} \; b_1 \in \mathbb{R}^M \; a \in \mathbb{R}^M$$

$$W_2 \in \mathbb{R}^{O \times M} \; b_2 \in \mathbb{R}^O \; p \in \mathbb{R}^O$$

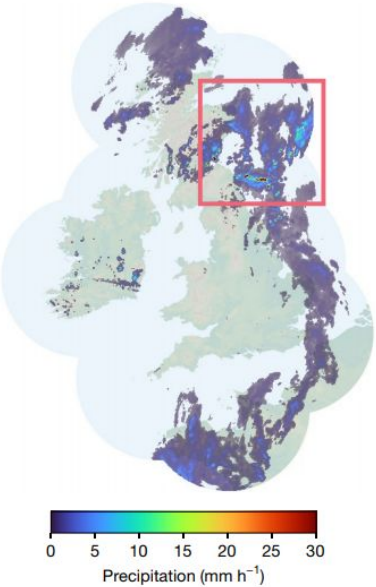
Visual Processing

- Self-driving *vehicles*

- Medical imaging

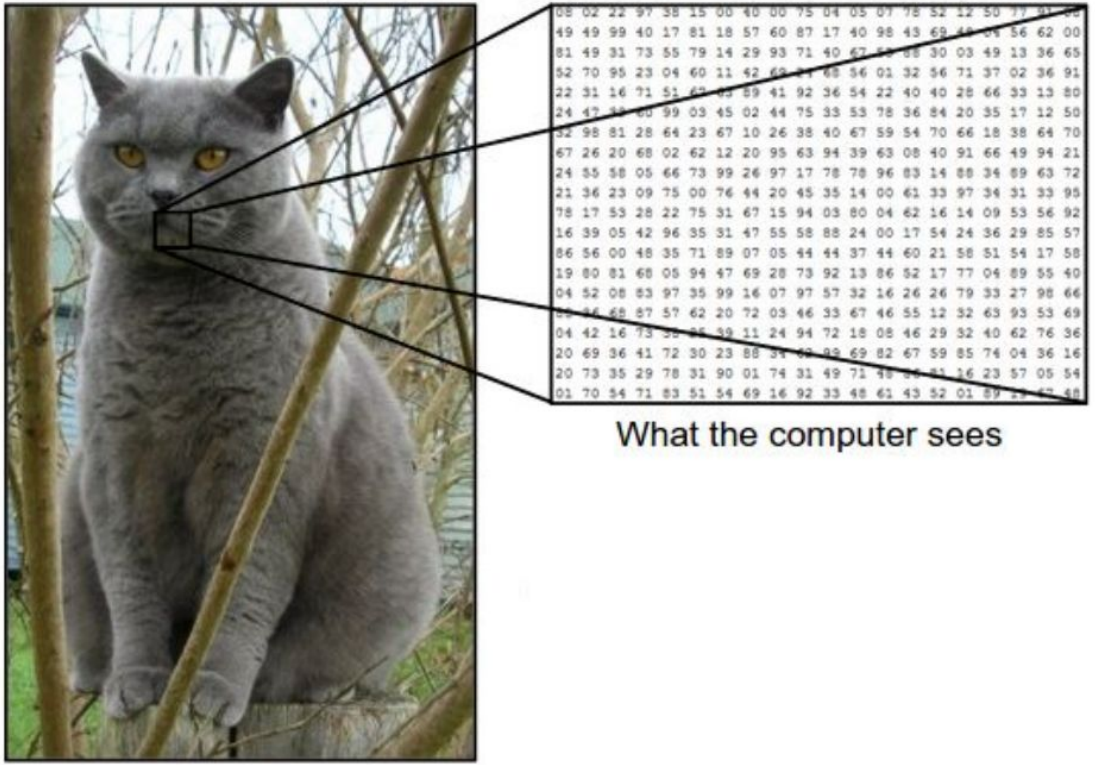


- Weather forecasting



van der Laak et. al. Deep learning in histopathology: the path to the clinic. Nature 2021
Ravuri et al. Skilful precipitation nowcasting using deep generative models of radar. Nature 2021

Process Images



Convolution

- Conv: Element-wise multiply the kernel and each patch and sum the result for every location

- Can be interpreted as:
  - Search for a patch in the image that matches the kernel
  - Compute the matching for every location

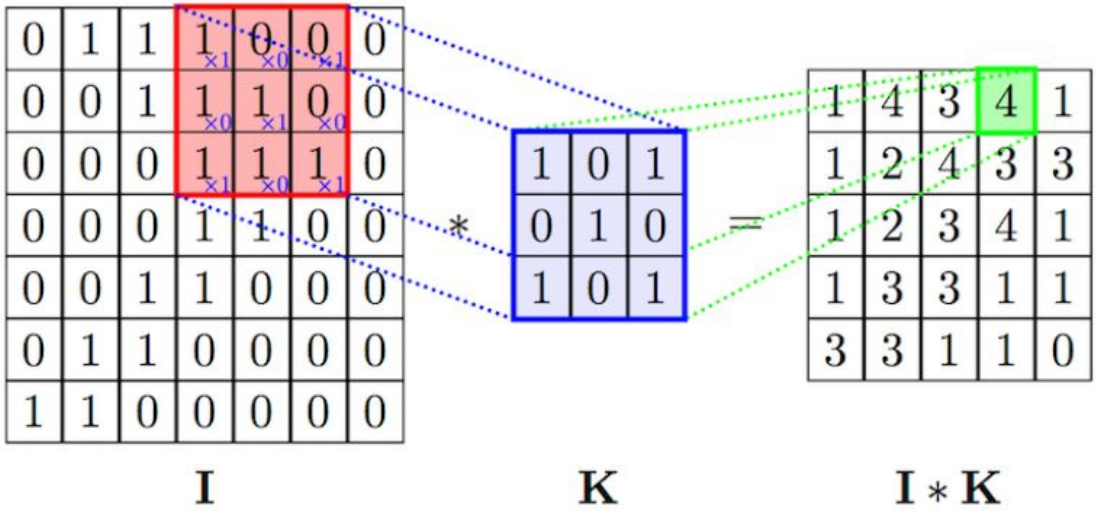
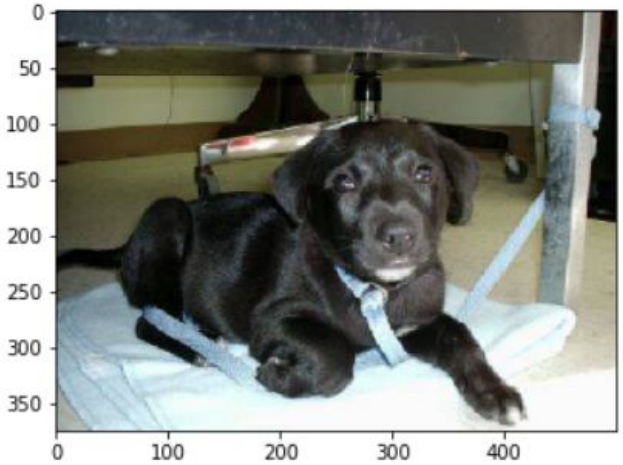


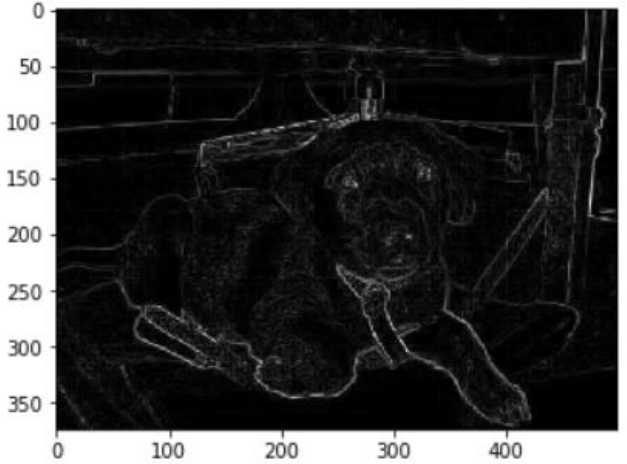
Image from: https://anhvnn.wordpress.com/2018/02/01/deep-learning-computer-vision-and-convolutional-neural-networks/

Convolution

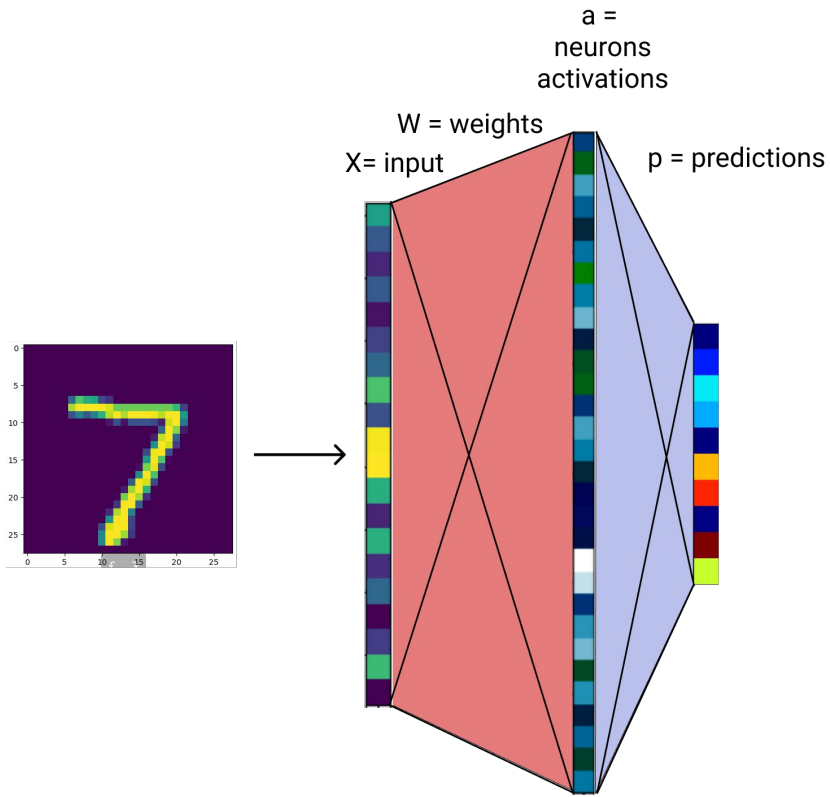


*

1	0	-1
2	0	-2
1	0	-1



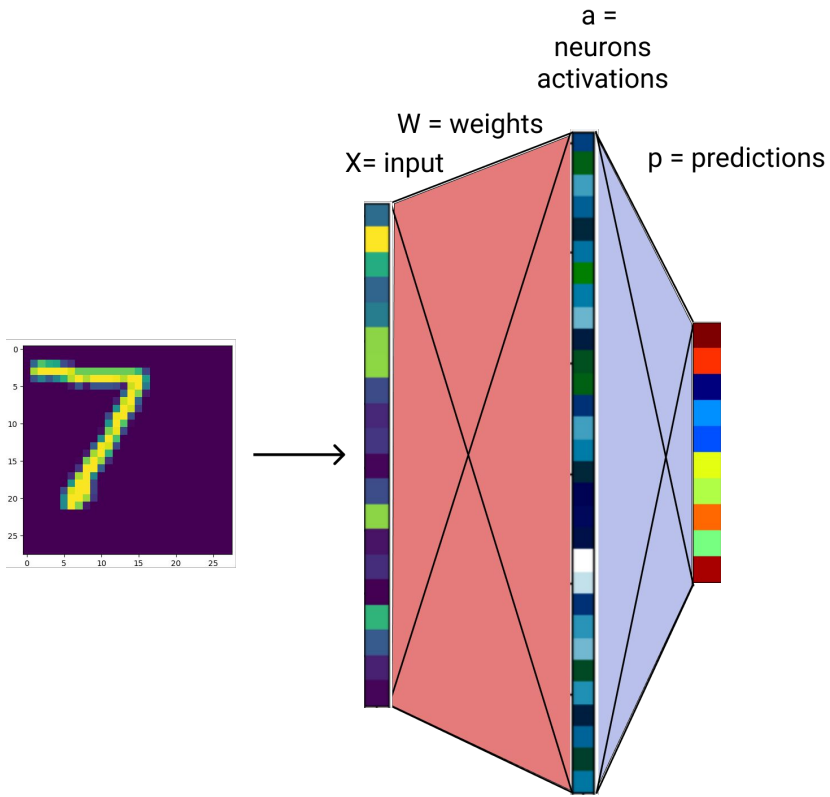
Process Images: Fully connected network



- Linearise image and treat it as a large vector
- Process the image (vector) with fully connected network

- Fully connected networks are universal approximators => they can represent anything

Process Images: Fully connected network



- Linearise image and treat it as a large vector
- Process the image (vector) with fully connected network

- Fully connected networks are universal approximators => they can represent anything

Process Images: Fully connected network

Downsides:

- Too MANY parameters
- Different output for small translations
- Every point is treated differently:
  - Network has to learn the same features for every location in the image
- Inefficient:
  - In **representation**: it has to learn redundant features
  - In **optimization:** hard to train, requires a lot of training examples
- Slow inference

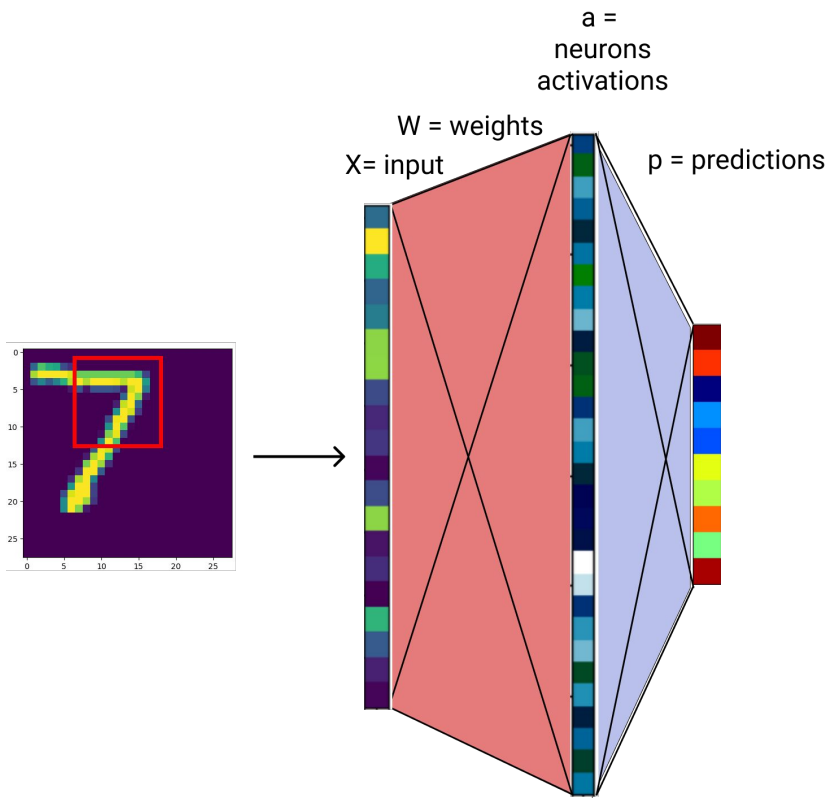
Conclusion: inefficient

Process Images: Useful biases

- Visual data has some useful biases:
  - **Locality**: for a pixel close neighbours are more important than distant ones
  - **Stationarity**: Same pattern has the same meaning regardless of position in the image

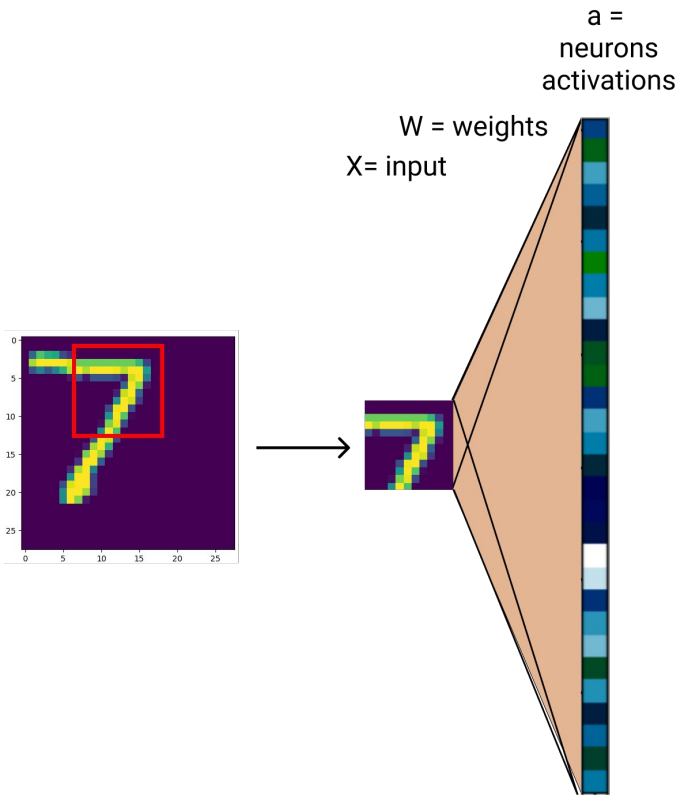
- An architecture suited for processing visual data should incorporate these such inductive biases into its architecture.

Process Images



- Let's tackle locality problem
- Restrict the input of the network to a small path

Process Images



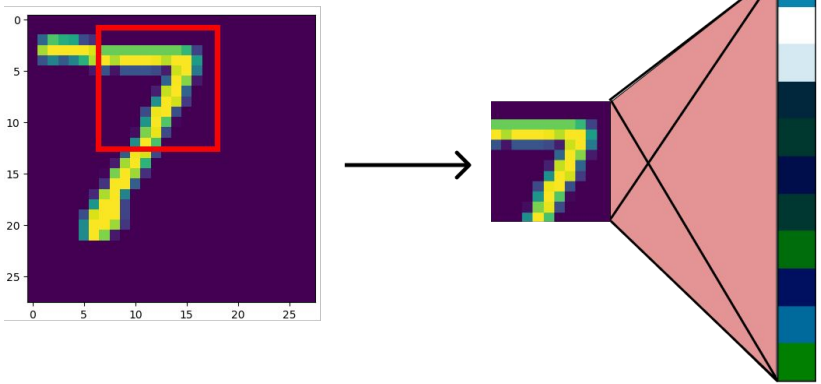
- Let's tackle locality problem
- Restrict the input of the network to a small path

Process Images

a =
neurons
activations

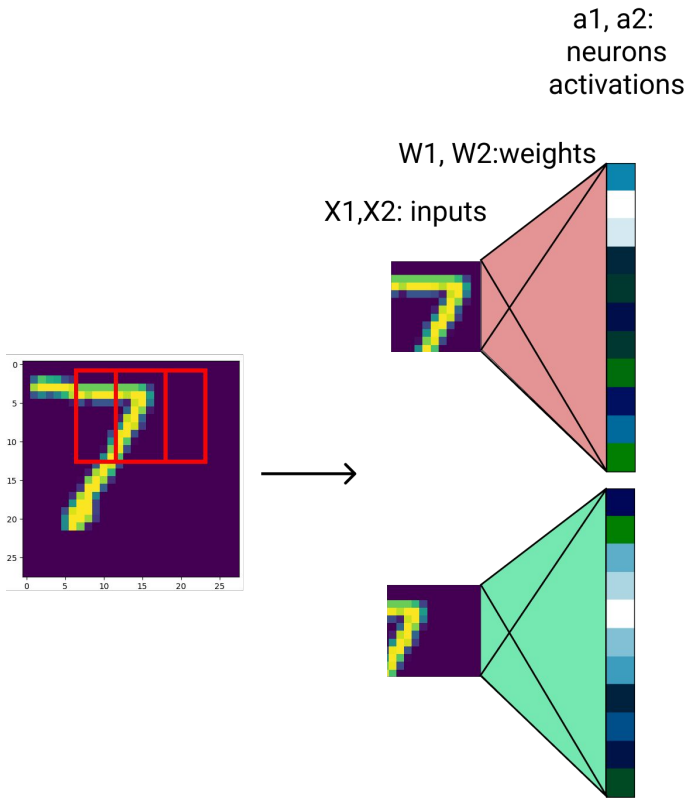
W = weights

X= input



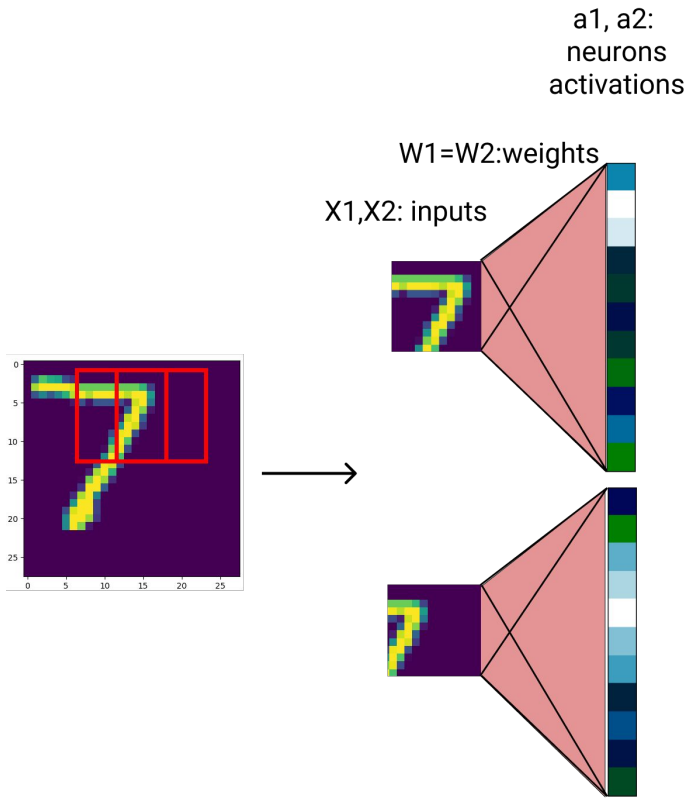
- Let's tackle locality problem
- Restrict the input of the network to a small path
- Small input: easier to model -> use fewer parameters

Process Images



- Process in a sliding window approach multiple patches in the image

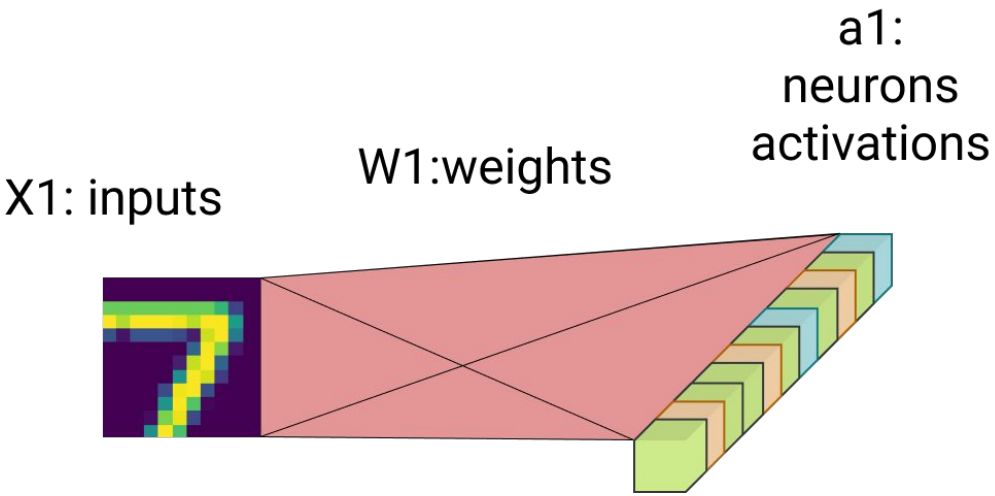
Process Images



- Process in a sliding window approach multiple patches in the image

- By the stationarity assumption: each location should be handled the same. Use the **same** function:
  - same weights

Process Images



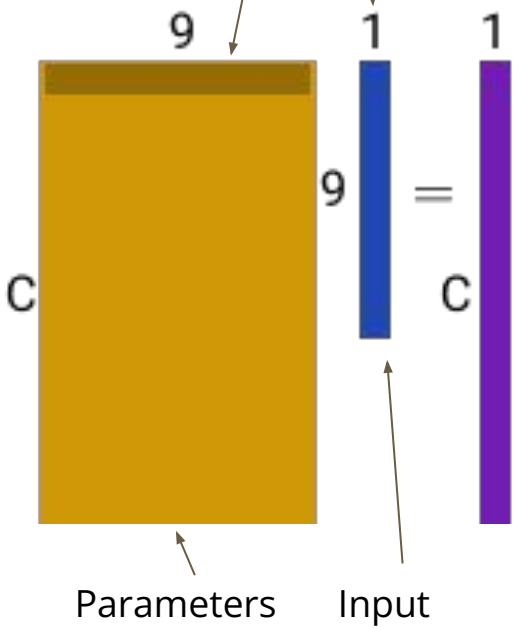
- Every patch has a corresponding output **feature vector**

- x, a - vectors
- W - matrix

$$a = \sigma(W_1 x + b_1)$$

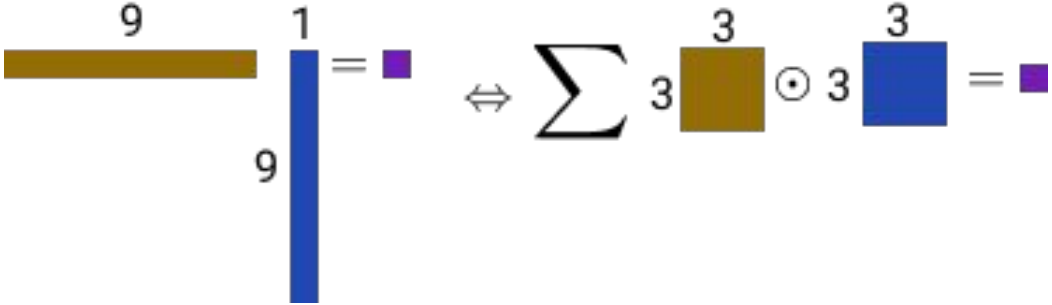
Process Images

$$a = \sigma(W_1 x + b_1)$$



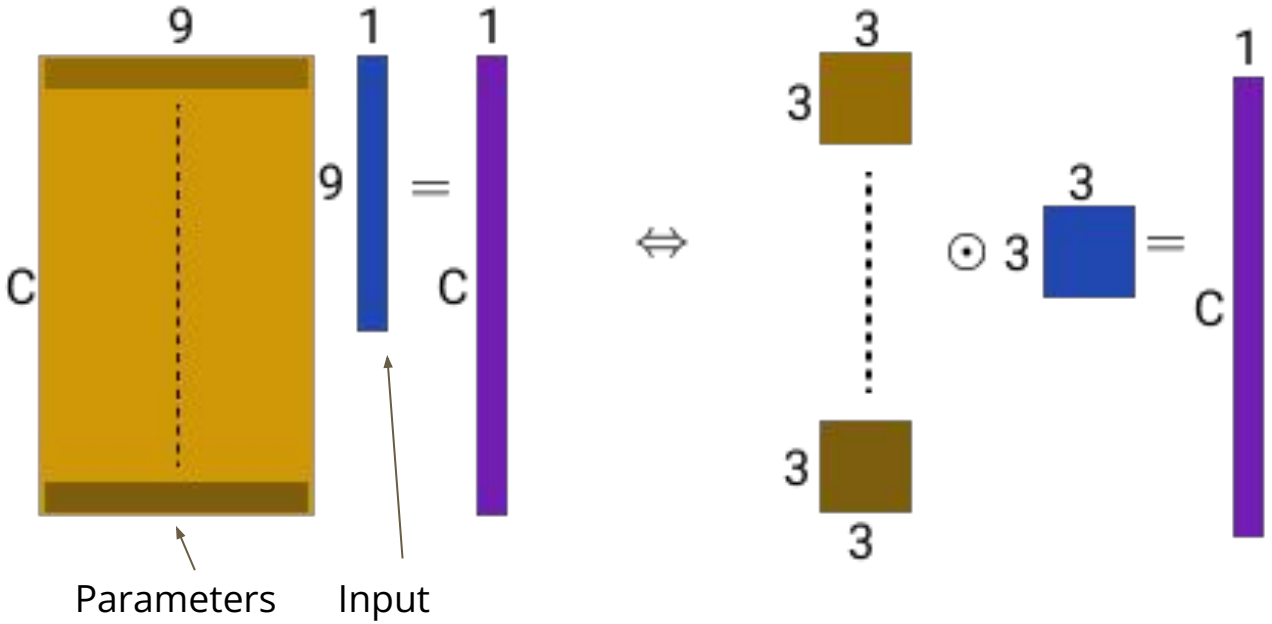
- We can rewrite the linear (+non-linear activation) in a different equivalent form.
  - Same mathematical operation - just different form

- A single dot product between a line in the weights matrix W and the input x can be written as follows:

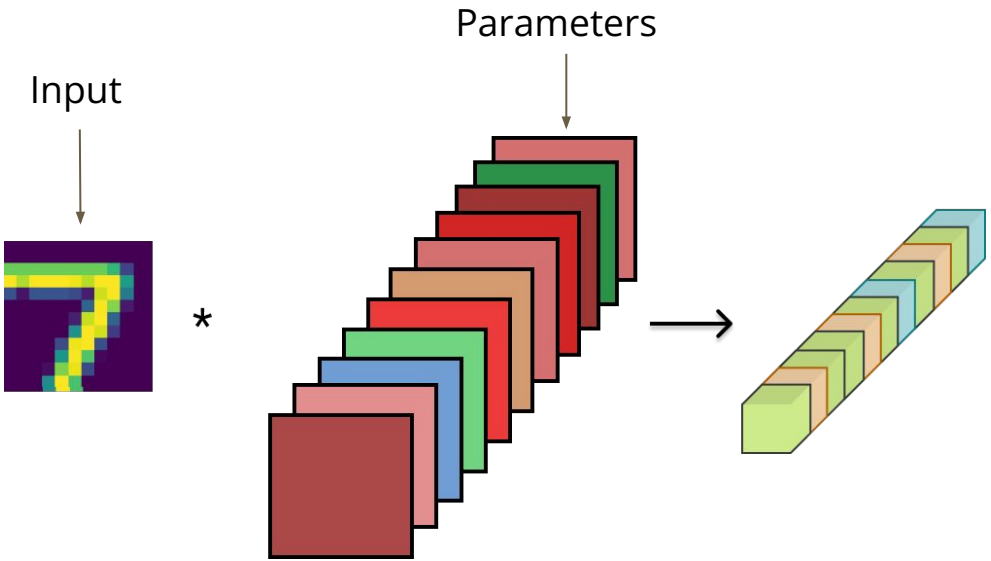


Process Images

$$a = \sigma(W_1 x + b_1)$$



Process Images

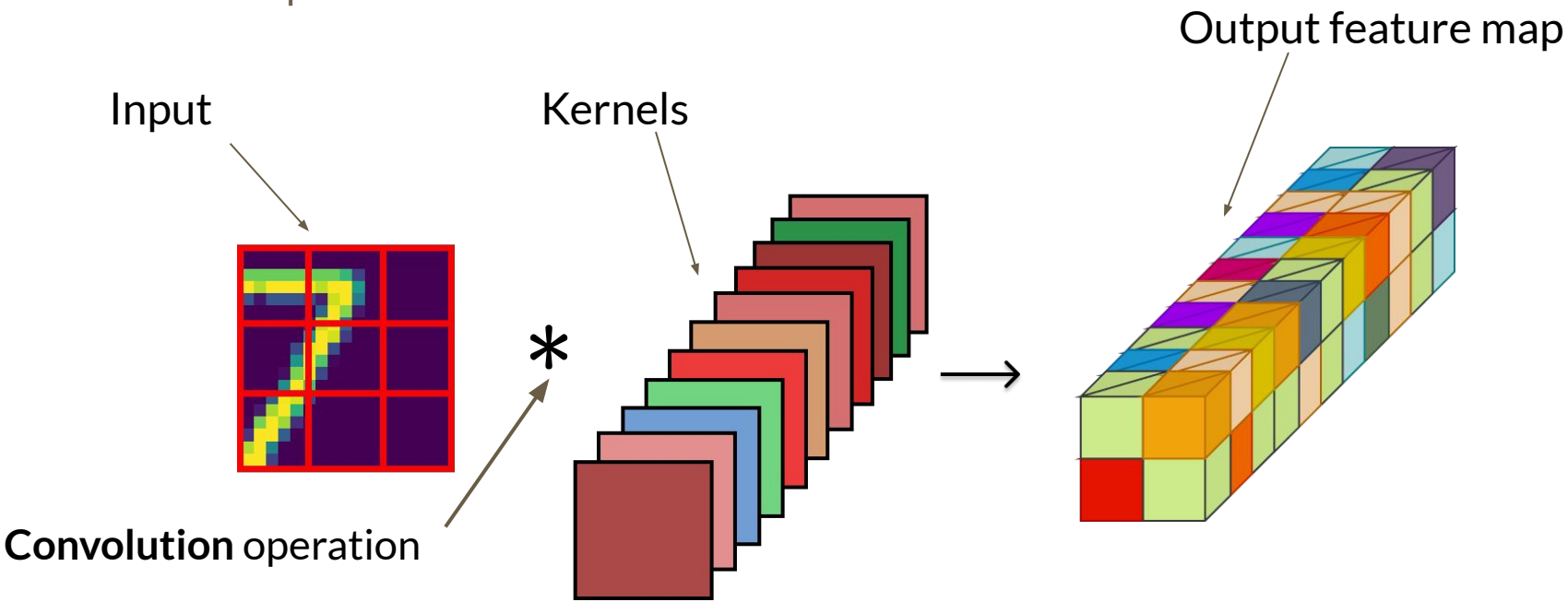


- Rewrite the linear operation Wx+b as elementwise multiplications and global sum
- Parameters <- Kernels = Filters

$$a(n) = \sigma(\sum_n \sum_m W_1(k,n,m) \cdot x(n,m) + b_1(k)), \forall n$$

Convolutional Layer

- Apply the same operation on every image patch



Convolution

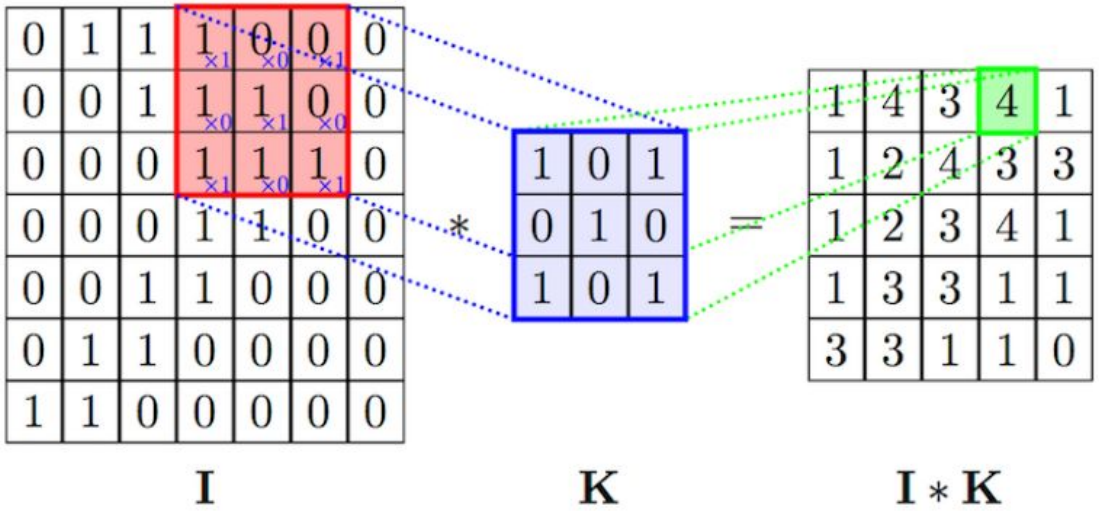
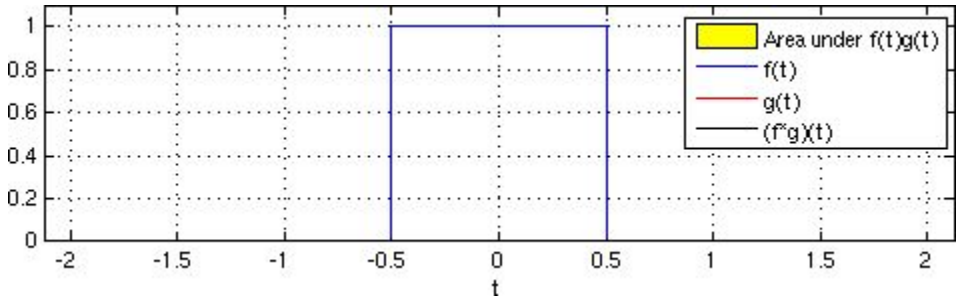


Image from: https://anhvnn.wordpress.com/2018/02/01/deep-learning-computer-vision-and-convolutional-neural-networks/

Convolution: continuous domain

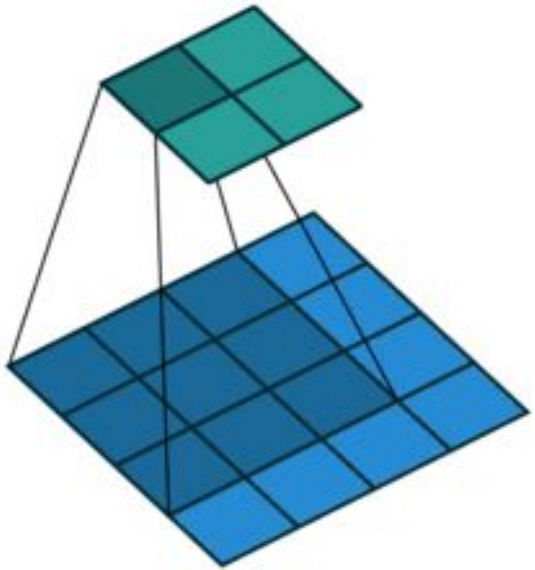
- For 1D continuous signals the convolution is defined as:

$$y(t) = (x * w)(t) = \int_{u=\infty}^{\infty} x(t-a)w(u)du$$



https://upload.wikimedia.org/wikipedia/commons/6/6e/Convolution_of_box_signal_with_itself.gif

Convolution



Discrete Case. Input x has size N and the filter w has size K

$$y(i) = (x * w)(i) = \sum_{n=0}^{K-1} x(i-n)w(n)$$

For ease of implementation, we can flip the kernel:

$$y(i) = (x * w)(i) = \sum_{n=0}^{K-1} x(i+n)w(n)$$

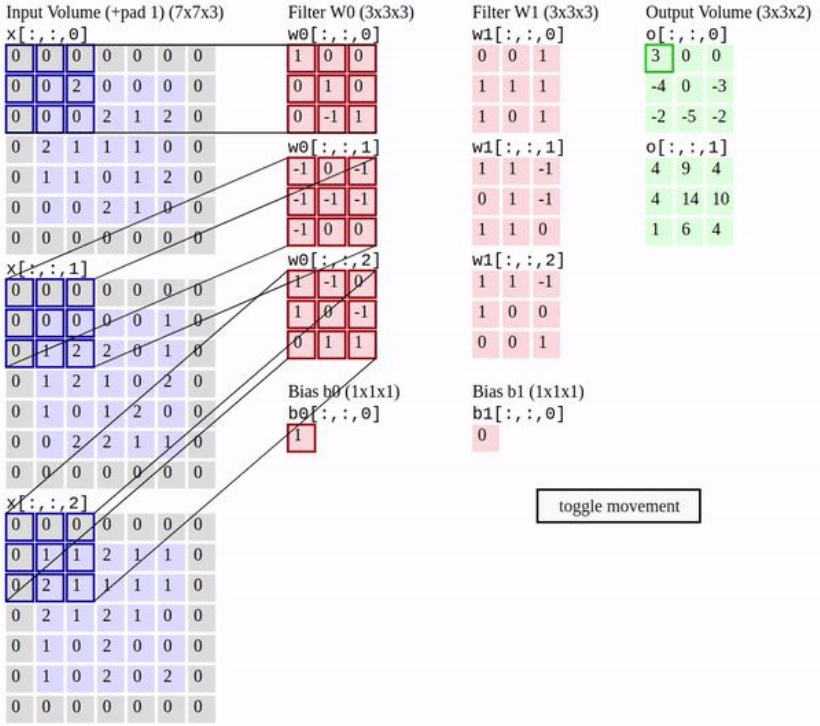
For 2D data, input has size $H \times W$ and the filter w has size $K \times K$

$$y(i,j) = (x * w)(i,j) = \sum_{n=0}^{K-1} \sum_{m=0}^{K-1} x(i+n, j+m)w(n,m)$$

Figure from https://github.com/vdumoulin/conv_arithmetic

Convolutional Layer

- The standard case for neural networks is having 2D inputs with **multiple** channels $\boldsymbol{x} \in \mathbb{R}^{C_{in} \times H \times W}$ and kernels $\boldsymbol{w} \in \mathbb{R}^{C_{out} \times C_{in} \times K \times K}$



$$y(d,i,j) = (x * w)(d,i,j) = \sum_{c=0}^{C_{in}-1} \sum_{n=1}^{K-1} \sum_{m=0}^{K-1} x(c, i+n, j+m) w(d,c,n,m)$$

Convolution

- If the input has C channels then each convolutional filter has C channels
- Here we have 2 filter, each having size C x K x K = 3 x 3 x 3
- In total:
  Filter W has size 2 x C x K x K
  Bias b has size 2

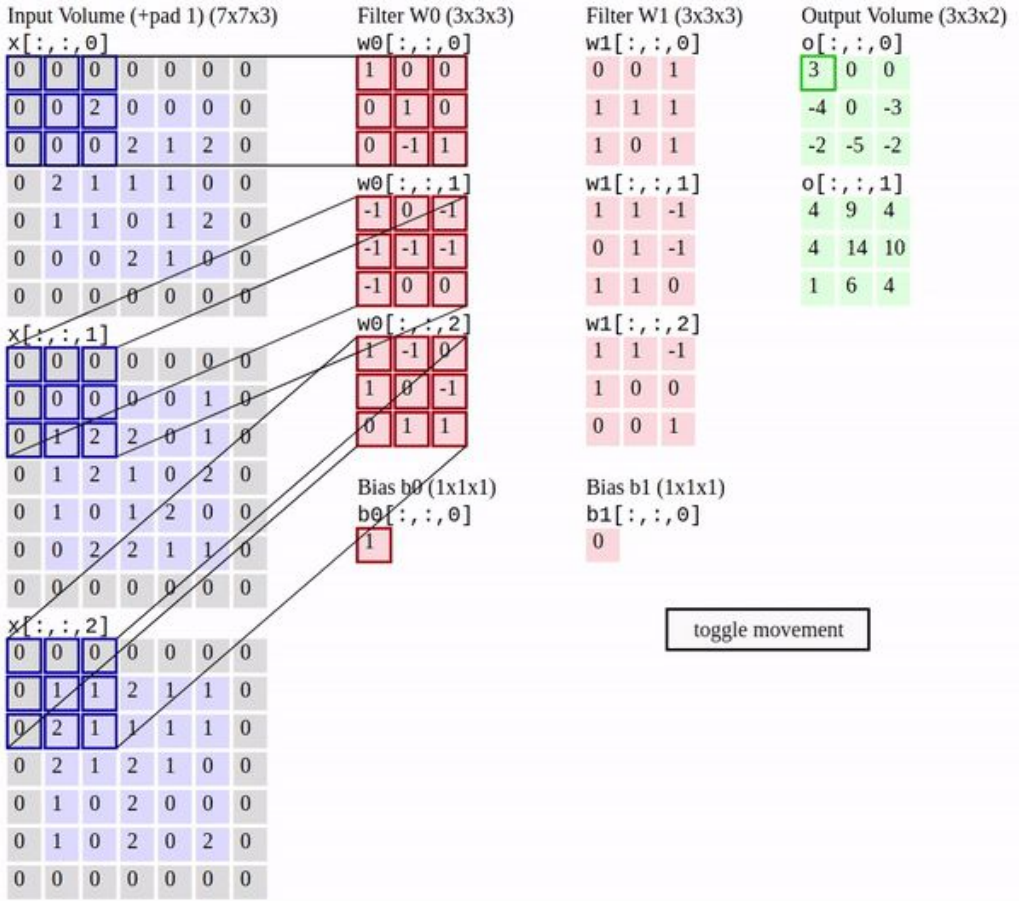
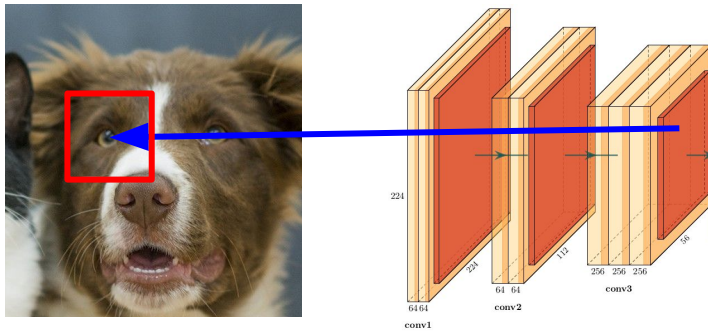


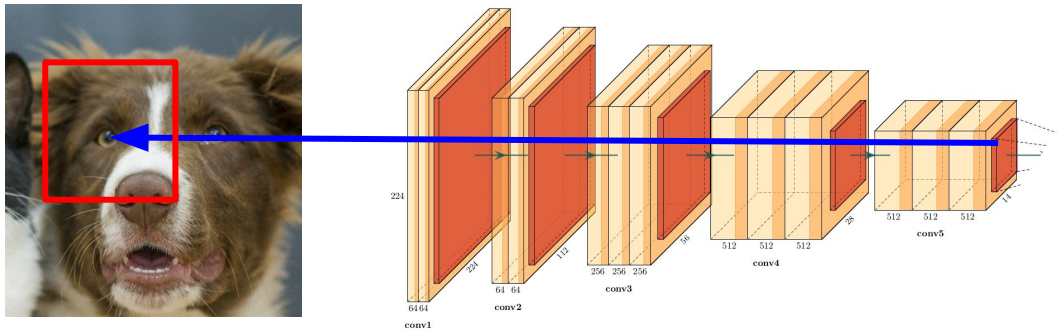
Image from: http://cs231n.github.io/convolutional-networks/

Convolutional Networks



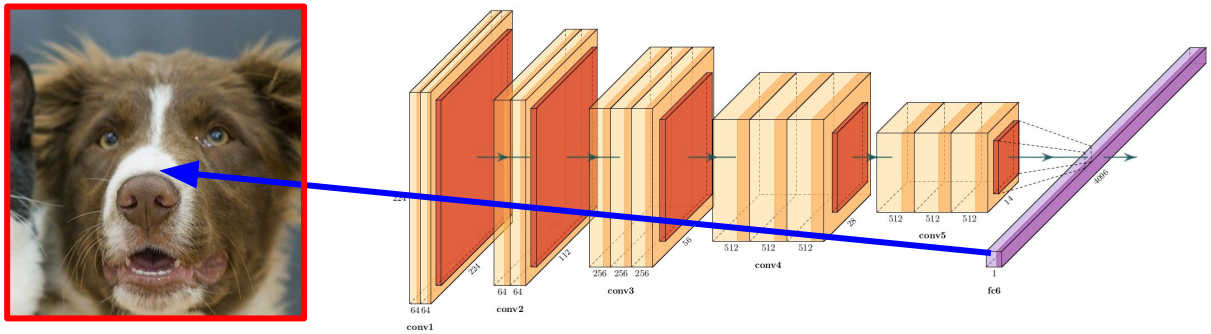
- A convolutional network is formed by stacking multiple convolutional layers, with non-linear activations in between. At the end, fully connected layers could be used.
- Going from the first layers to the last, the features obtained by the network have different meanings. In the first layers they represent low-level features such as edges, textures, while in the upper layers the features could represent entities such as parts or whole objects
    - Edges -> textures -> entities

Convolutional Networks



- A convolutional network is formed by stacking multiple convolutional layers, with non-linear activations in between. At the end, fully connected layers could be used.
- Going from the first layers to the last, the features obtained by the network have different meanings. In the first layers they represent low-level features such as edges, textures, while in the upper layers the features could represent entities such as parts or whole objects
    - Edges -> textures -> entities

Convolutional Networks

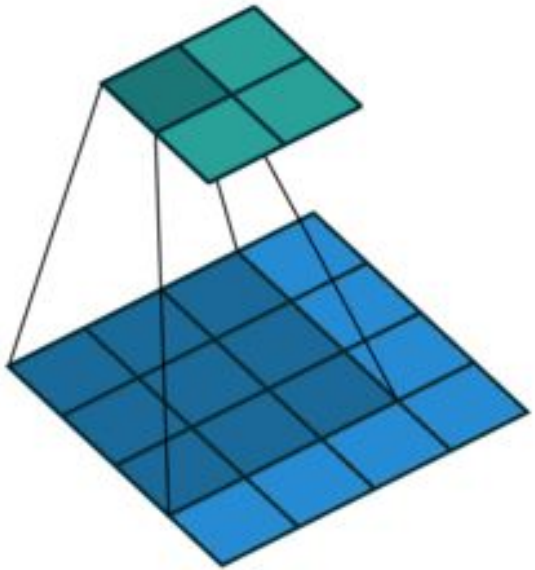


- A convolutional network is formed by stacking multiple convolutional layers, with non-linear activations in between. At the end, fully connected layers could be used.
- Going from the first layers to the last, the features obtained by the network have different meanings. In the first layers they represent low-level features such as edges, textures, while in the upper layers the features could represent entities such as parts or whole objects
    - Edges -> textures -> entities

Recap - Convolutional operation

- It is **efficient** because it takes advantage of properties of the data
  - **Locality** and **stationarity**
- Appropriate for cases where the data has these properties:
  - Images, videos
  - Audio
  - Sequence of words
  - Any problem where spatial structure plays a key role
- Keywords:
  - Learnable parameters: Kernel = filter + bias

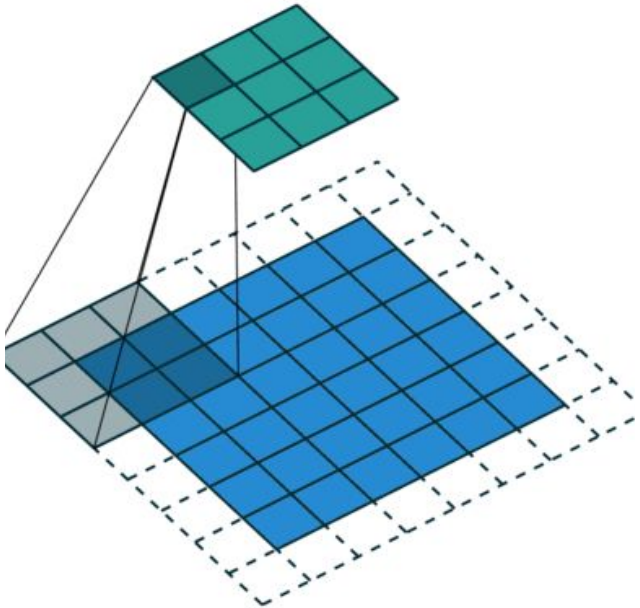
Convolution



- We will see how convolutions are applied in standard convolutional networks.
- Introduce some key concepts

Figure from https://github.com/vdumoulin/conv_arithmetic

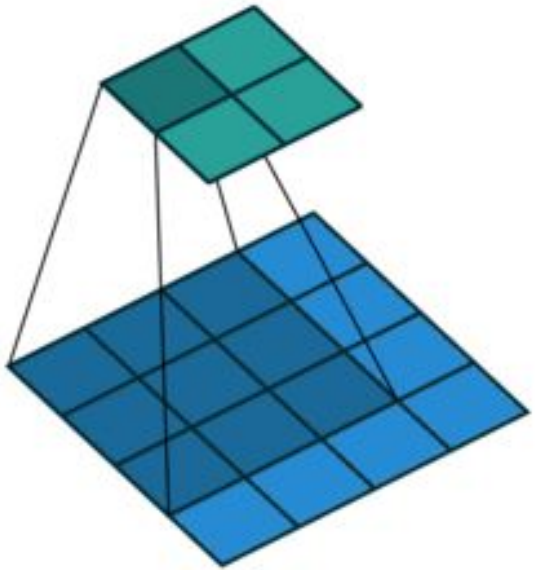
Convolution: Stride



- Skip computations with a stride
- Stride k: move each patch by k pixels
- Reduces the resolution of the output

Figure from https://github.com/vdumoulin/conv_arithmetic

Convolution: Padding



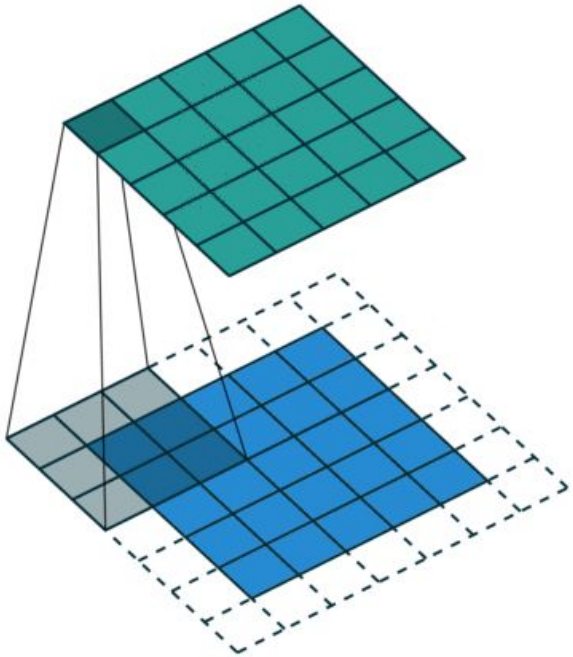
- Lose information at margins
  - The output features maps are smaller than the input

Solutions:

- Process margins separately with different kernels
- **Apply padding to the input**

Figure from https://github.com/vdumoulin/conv_arithmetic

Convolution: Padding

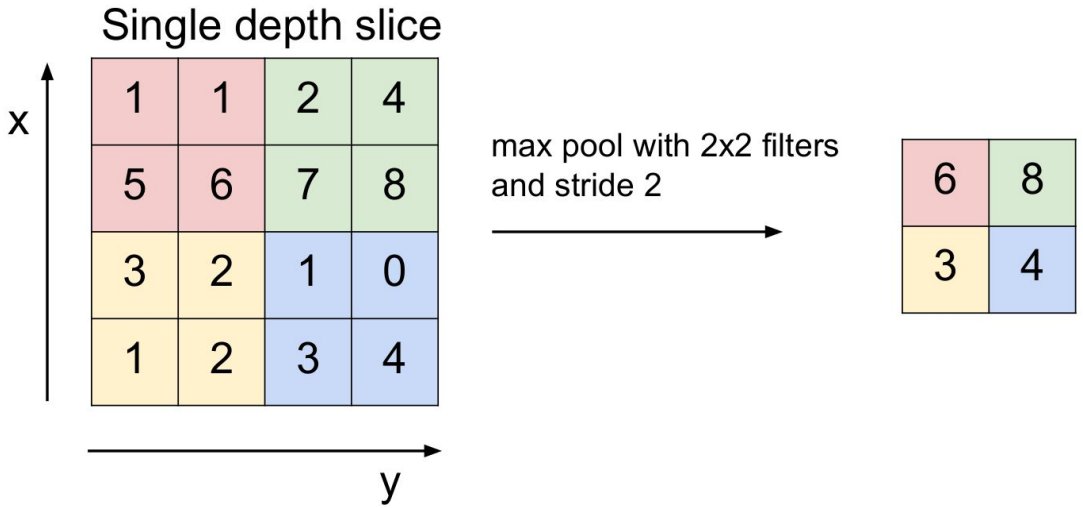


- Use padding half the filter size, to keep the output the same size as the input

- Different kinds of padding:
  - **Zero**, reflexive

Figure from https://github.com/vdumoulin/conv_arithmetic

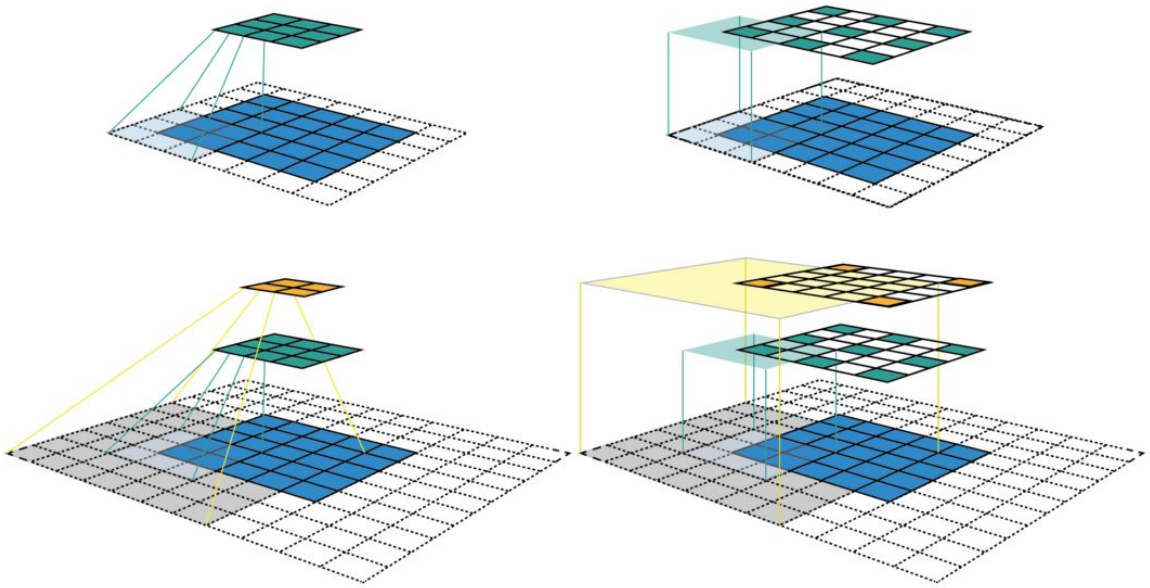
Pooling



- From every patch output one value
- Reduce output resolution:
  - Output is invariant to small local translations
  - Make the computation lighter
- Pooling types: max, average

Figure from: CS231n Stanford Course

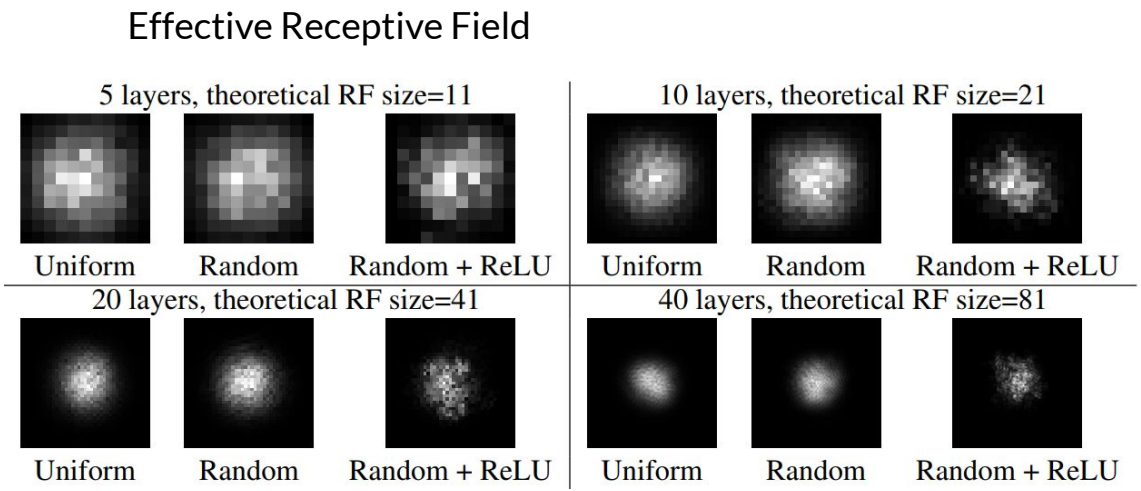
Receptive field



- **Receptive field** of a point in a feature map:
  - Region of the image that influences that point
- Receptive field grows:
  - linear with the number of layers with stride one
  - multiplicatively by using layers with stride > 1
  - linear with size of filters

Figure from: https://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807

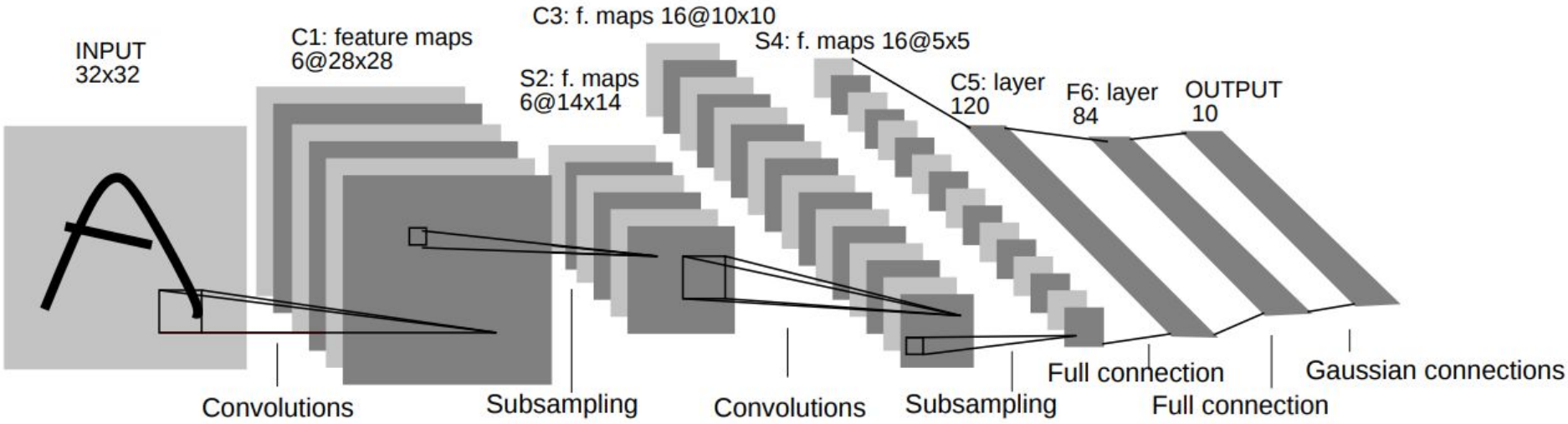
Effective Receptive field



- **Effective Receptive field** of a point in a feature map:
  - Region of the image that influences that point in a **non-negligible** way
- effective receptive field only takes up a **fraction** of the full theoretical receptive field
- distribution of effective receptive field: asymptotically **Gaussian**

Figure from: Luo et. al: Understanding the effective receptive field in deep convolutional neural networks

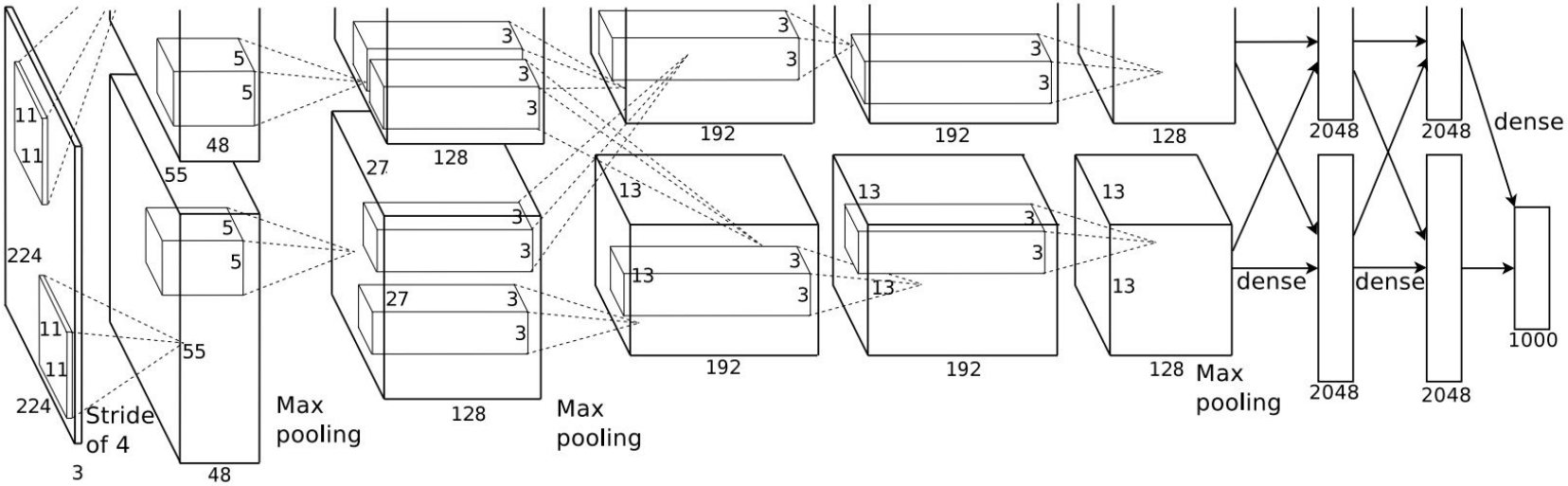
Convolutional Networks in 1998: LeNet-5



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

Convolutional Networks in 2012: AlexNet - DL revolution

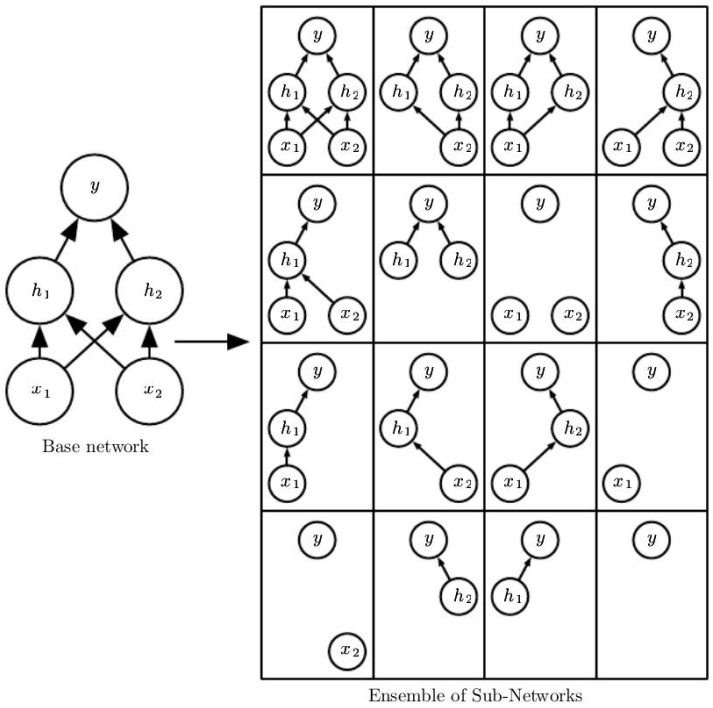
- fast conv nets implemented on GPU started to obtain superior results: DanNet [2011] - AlexNet [2012]
- By winning the 2012 ImageNet competition AlexNet is considered to have triggered a revolution in ML, starting the current wave of Deep Learning methods



Krizhevsky et al. [2012]: Imagenet classification with deep convolutional neural networks

Dan Cireșan, U Meier, J Schmidhuber [2011]: Multi-column Deep Neural Networks for Image Classification

Dropout



- At each step in the optimisation algorithm make 50% of neuron activations zero
- At testing time (inference) use all the output but scaled with 0.5

- Reduces **overfitting** by **regularise** the network
- Make each neuron not relying on fixed features
  - => more robust model
- Create implicit ensemble of subnetworks

Image from: I. Goodfellow et. al: Deep Learning book

Regularization

- Any strategy used to **reduce its generalization error** but **not** the training error
- Constraints and penalties designed to encode **prior knowledge** or to **impose preference** towards simpler models in order to promote generalization

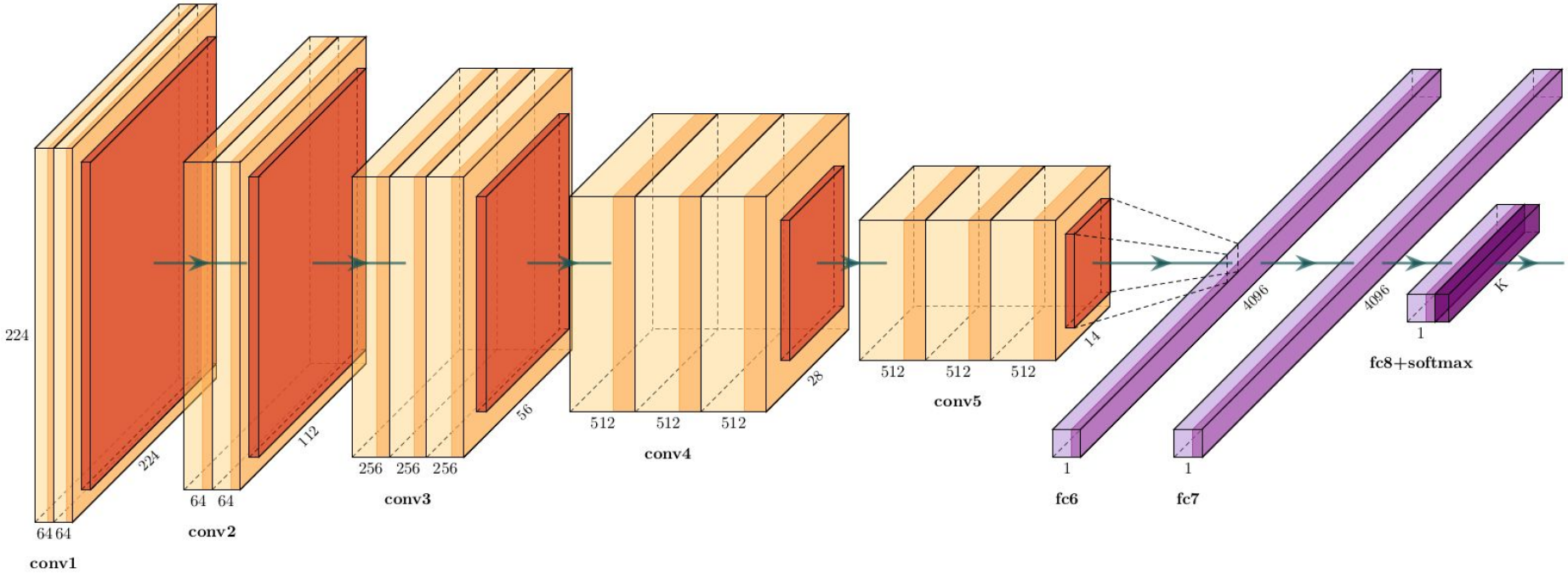
General Regularizations:

- Weight decay: L2 or L1 penalties of parameters norm
- Dropout
- More data / Data augmentation
- Adding noise: to input / activations / weights / labels
- Early Stopping

Implicit Convolutional Regularizations:

- **Weight sharing** across locations
- **Local** / **gaussian** effective receptive fields

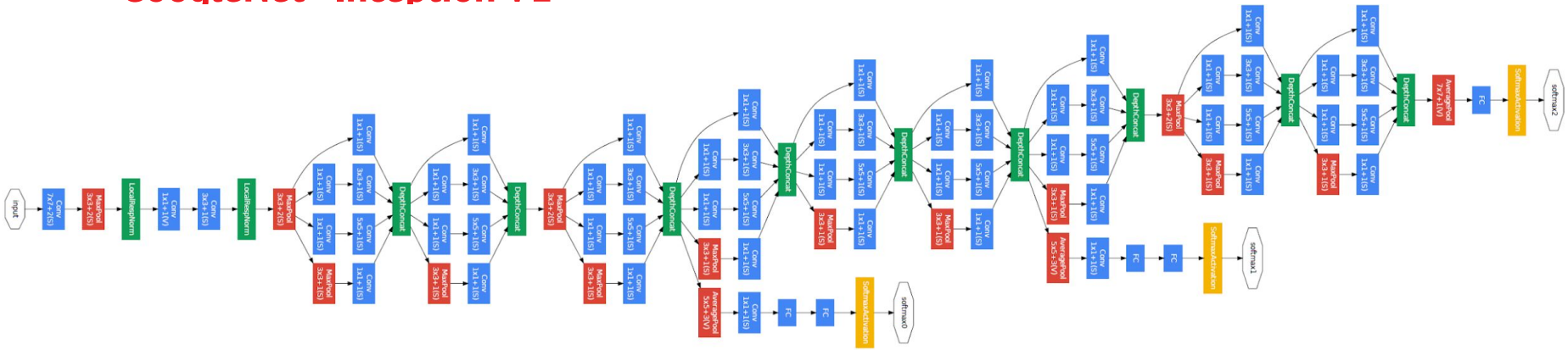
Convolutional Networks in 2014: VGG



Simonyan and Zisserman [2014]: Very deep convolutional networks for large-scale image recognition.

Image from:
https://github.com/HarisIqbal88/PlotNeuralNet

Convolutional Networks in 2014: GoogleNet - Inception V1

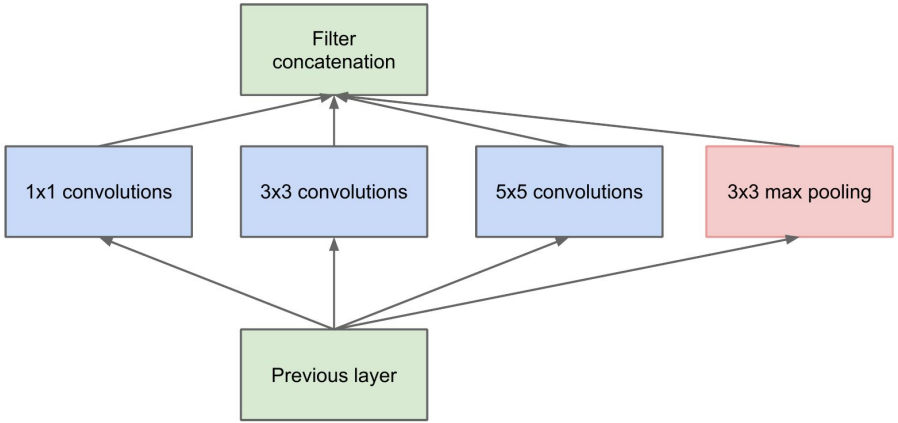


- Core principles:
  - efficient utilization of the computing resources
  - multi-scale processing inside the network

- Results
  - ImageNet ILSVRC14 winner (6.7% top 5 error)
  - 12x less parameters than AlexNet while significantly more accurate

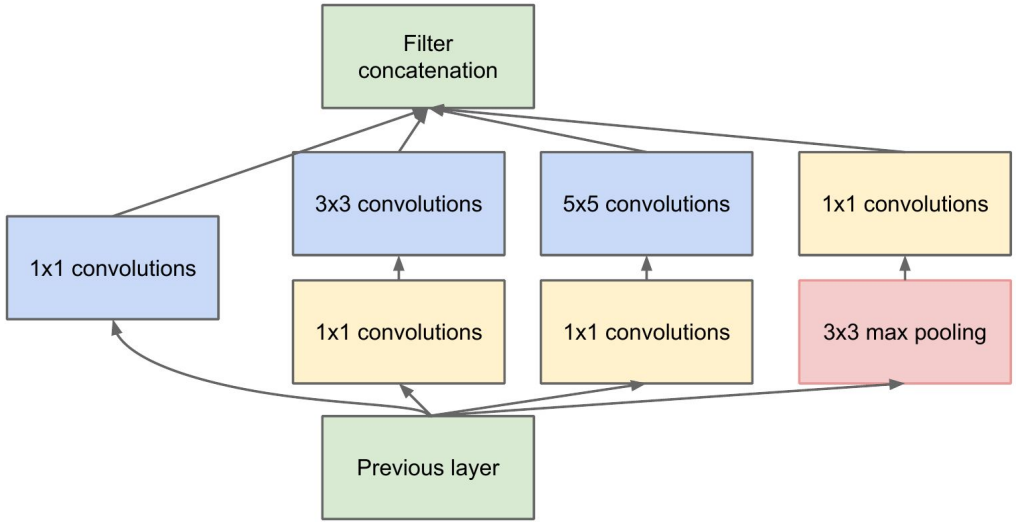
Szegedy et al. [2015]: Going deeper with convolutions

Inception Module



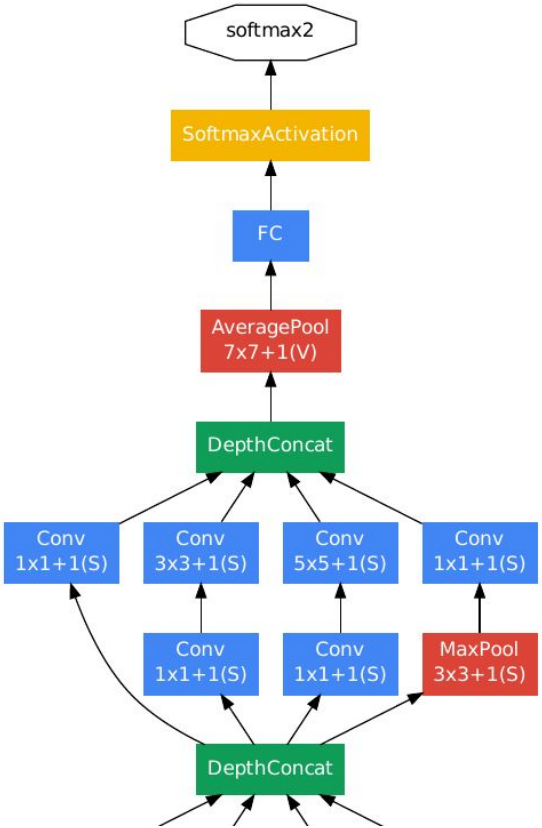
- Process at different spatial scales
  - 1x1 convs
  - 3x3 convs
  - 5x5 convs
- Concatenate the result of all branches
- Drawback:
  - Large computations

Inception Module



- Use 1x1 convolutions to reduce the dimensions
- 1x1 convolutions project every pixel information into a lower dimensional space
  - I : 512 x 28 x 28 ->
    O: 64 x 28 x 28
- The resulting embedding act as a **compressed** information

Final Layers

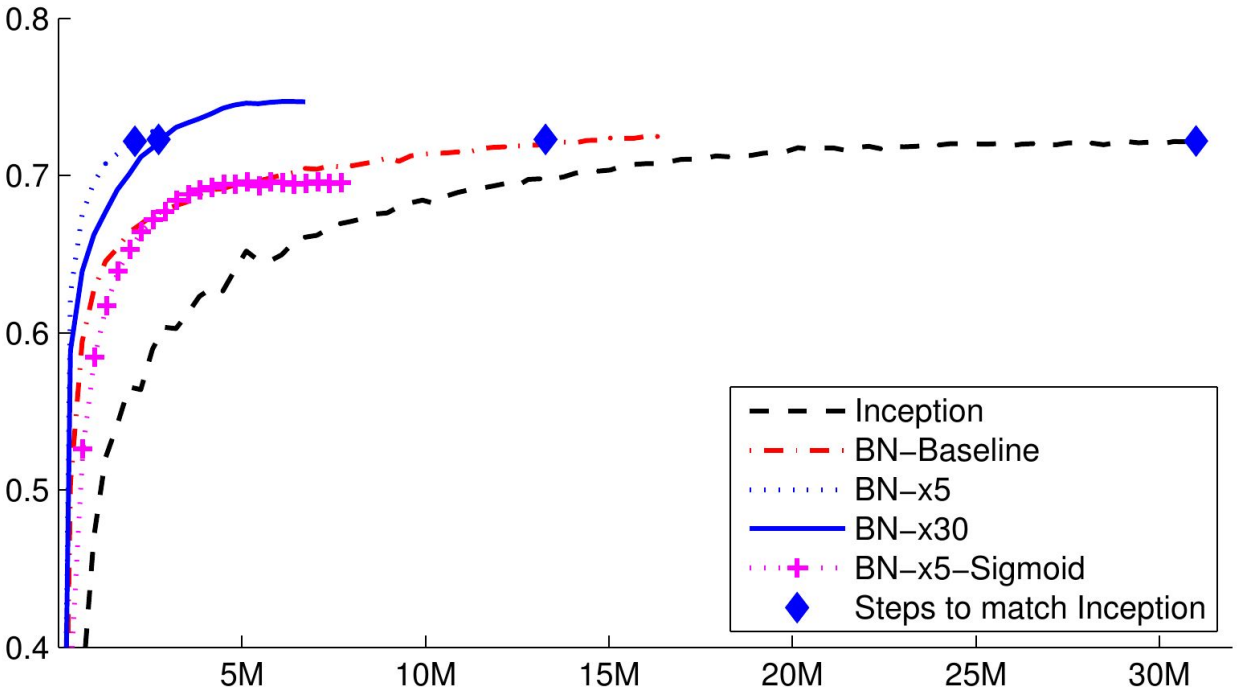


- After the last inception module simply use **average pooling**
- Final linear layer + softmax

- By replacing fully connected with average pooling:
  - Removes many parameters from the last fully connected layers
  - improved the top-1 accuracy by about 0.6%

- For comparison: VGG16 has 73% (103M / 138M) just in the first fully connected layer after the convolutional part

Convolutional Networks in 2015: Inception - v2



- Use **Batch Norm**

Ioffe and Szegedy [2015]: Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Batch Norm

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

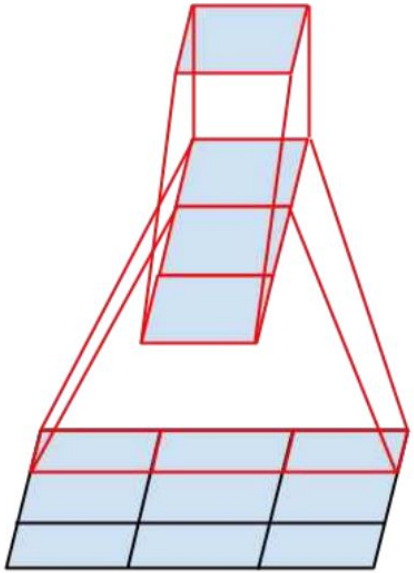
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

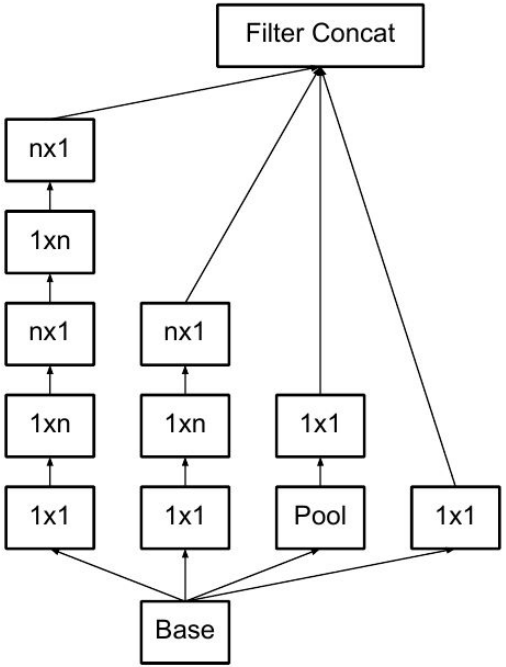
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- Different kinds of normalisation
- Batch Norm for Vision - ConvNets
- Layer Norm for NLP & CV - recurrent Nets, Transformers, GNNs
- Spectral Norm for RL

Ba et.al. Layer Normalization. 2016
Gogianu et. al: Spectral Normalisation for Deep Reinforcement Learning: An Optimisation Perspective. ICML 2021

Convolutional Networks in 2016: Inception - v3



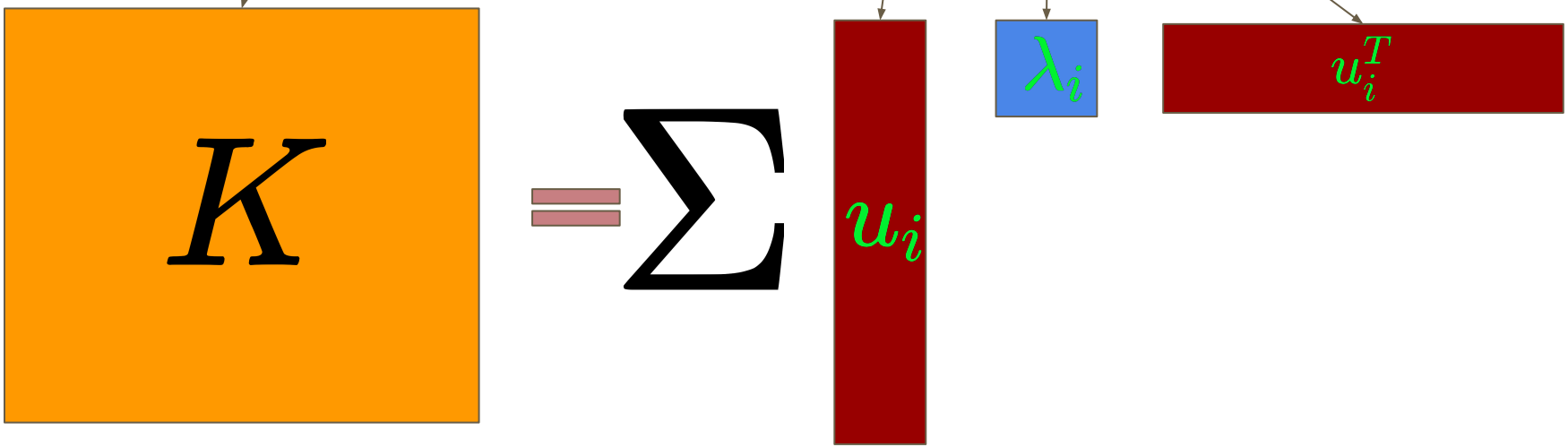


- Factorise the convolutions
- Replace N x N kernels with N x 1 and 1 x N kernels
- Reduces computations
- Acts as a regularization

Szegedy et al. [2016]: Rethinking the inception architecture for computer vision

Kernel Factorisation: SVD motivation

$$K = U\Sigma U^T = \sum_i u_i * \lambda_i * u_i^T$$



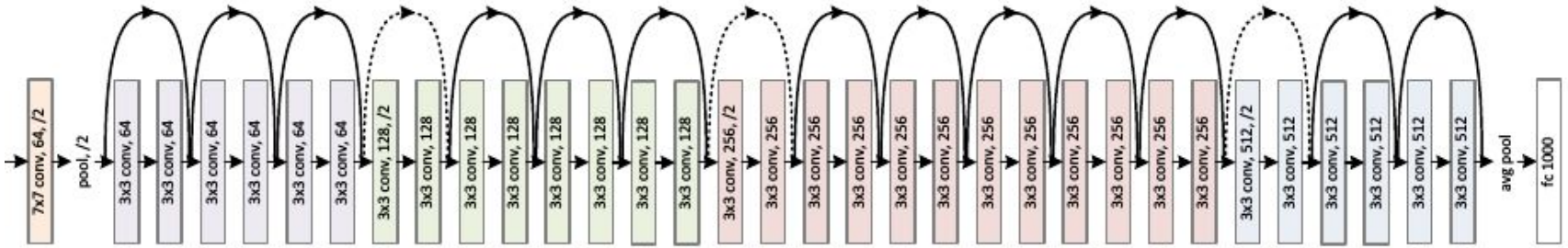
Kernel Factorisation: SVD motivation

$$K = U\Sigma U^T = \sum_i u_i * \lambda_i * u_i^T$$

- An N x N matrix is decomposed in **maximum N** vector outer products

- By using fewer we arrive at a low rank matrix:
  - Fewer computations
  - Better regularisation
  - In practice, same representation power

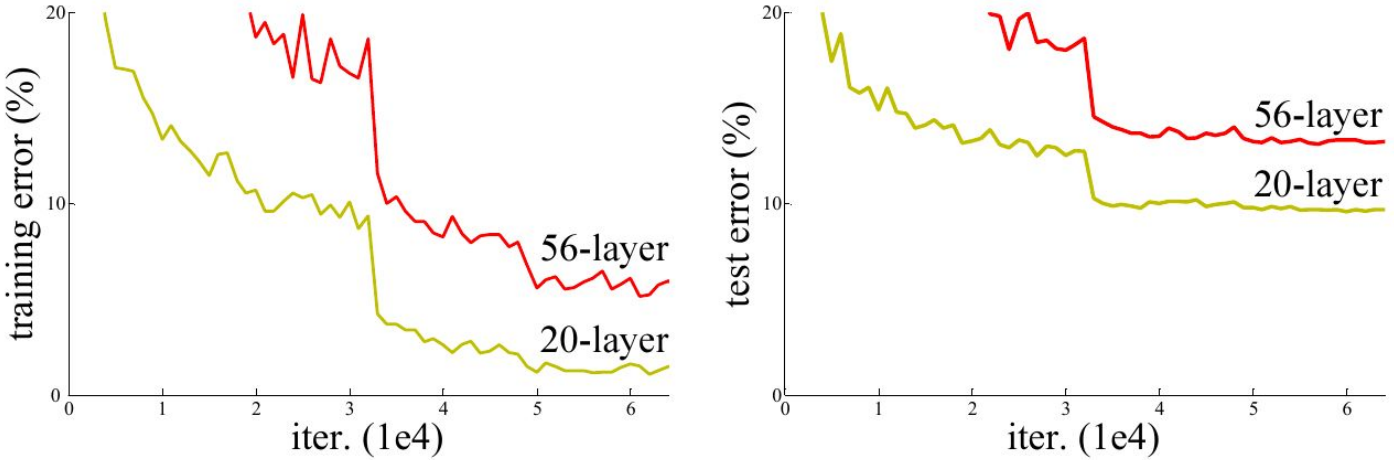
Convolutional Networks in 2016: ResNet



- Core principles:
  - Train deeper models
  - Easily optimise by allowing direct paths between lower and upper levels

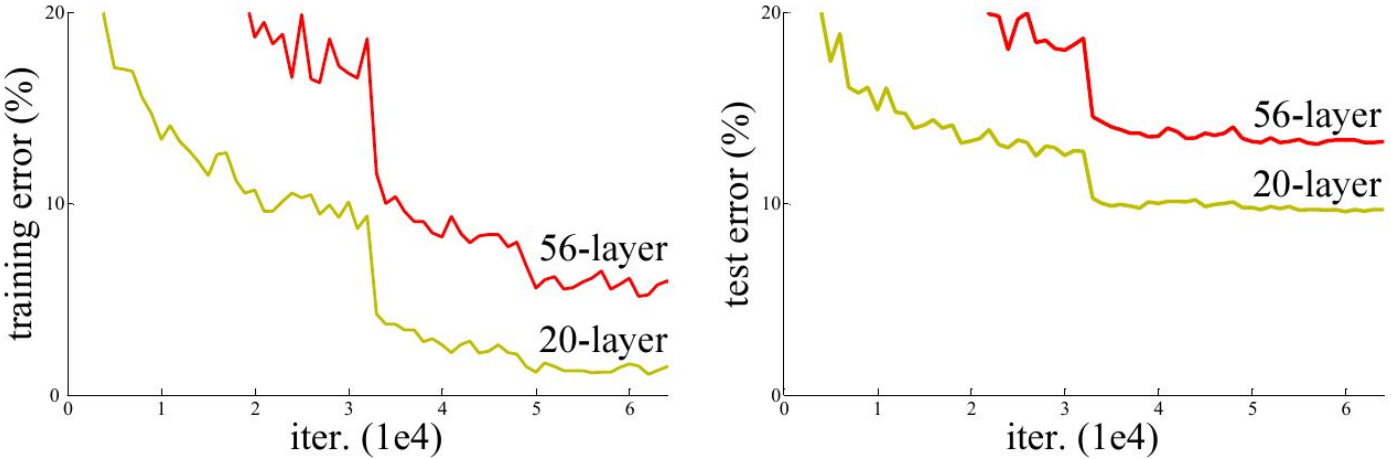
He et al. [2016]: Deep residual learning for image recognition

Convolutional Networks in 2016: ResNet



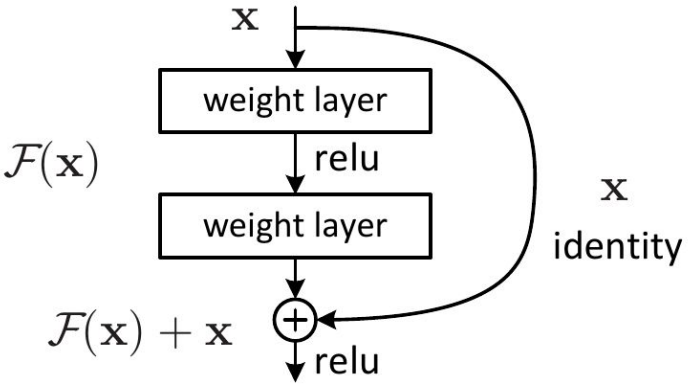
- Deep ‘plain’ networks obtain worse results by increasing the number of layers
- Cause:

Convolutional Networks in 2016: ResNet



- Deep ‘plain’ networks obtain worse results by increasing the number of layers
- Cause: Overfit? NO
  - **Optimization problem**

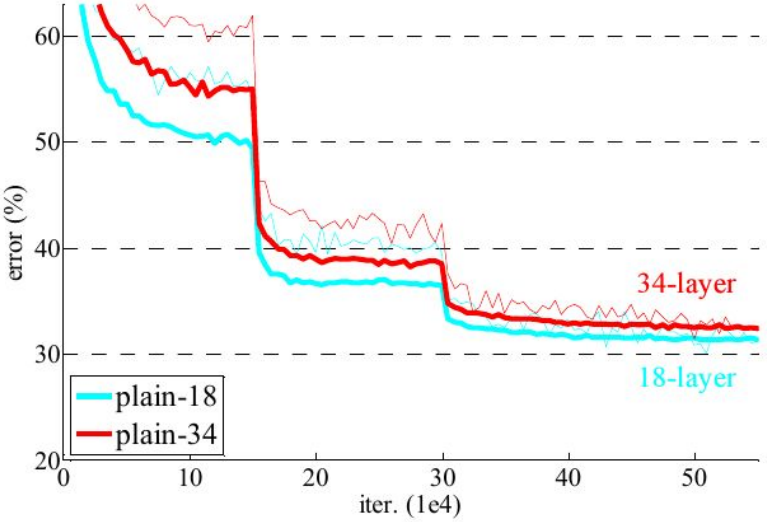
Convolutional Networks in 2016: ResNet



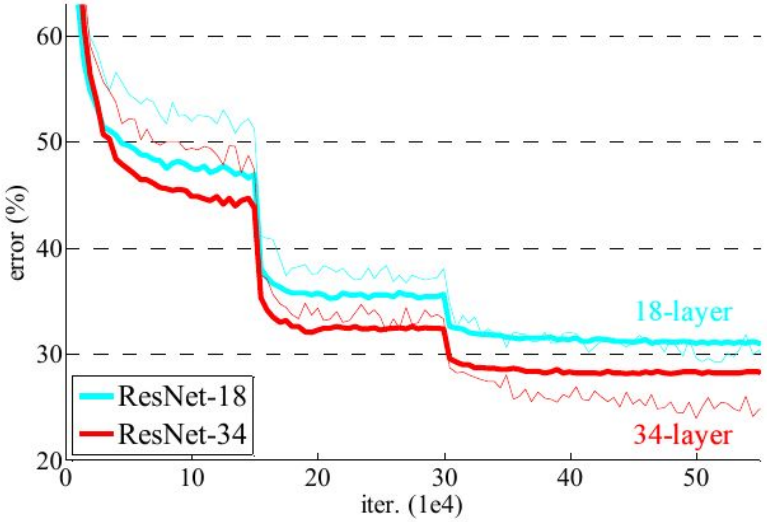
- Add skip connections between between layers
- The representation power of the network remains the same
- The network can easily learn identity mapping
- The optimisation is easier

Convolutional Networks in 2016: ResNet

Without BN



With BN



- Effect of the residual connections
- Deeper models can be trained and can improve shallower models

Recent Convolutional Nets Performance

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

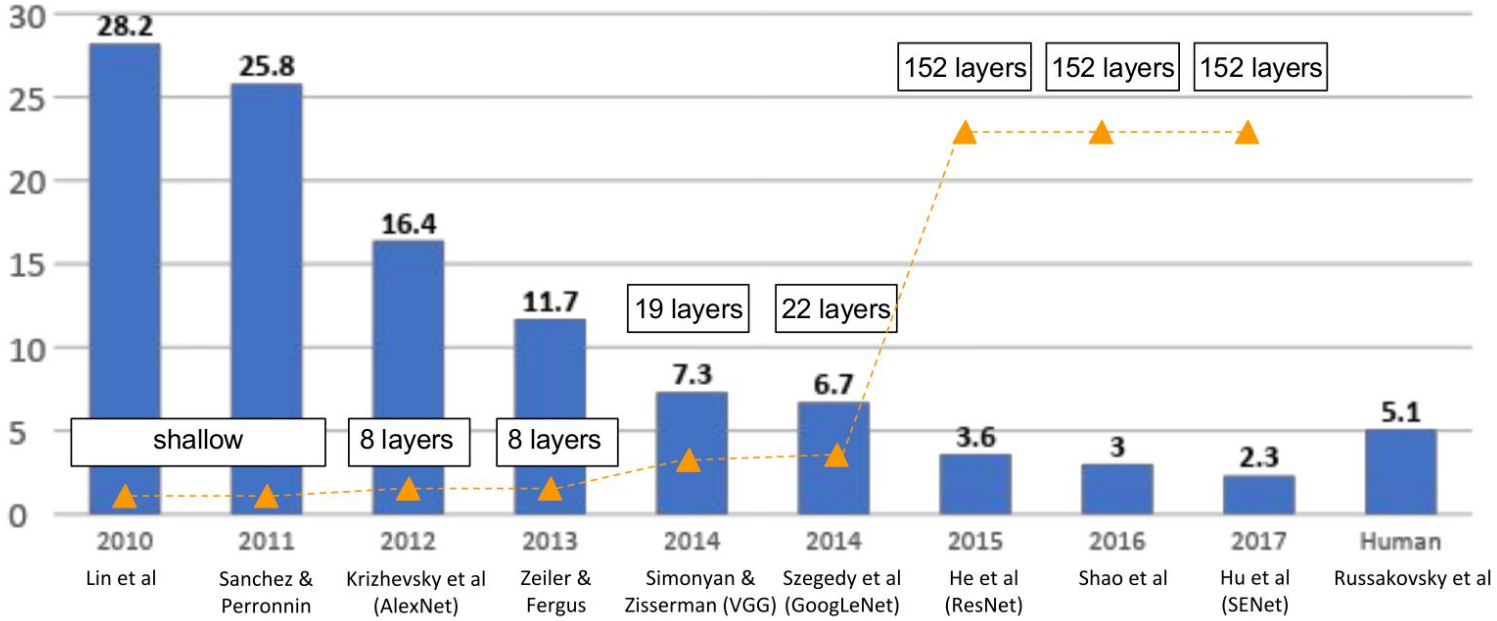
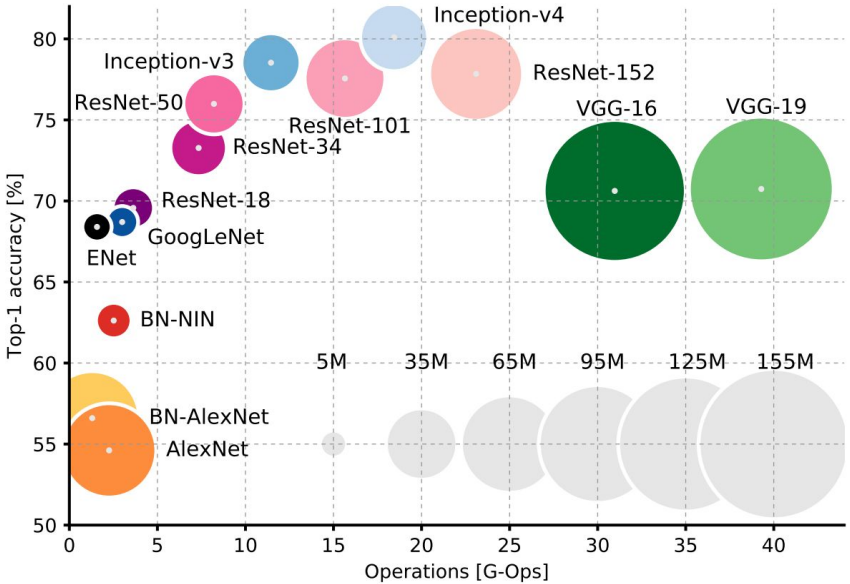
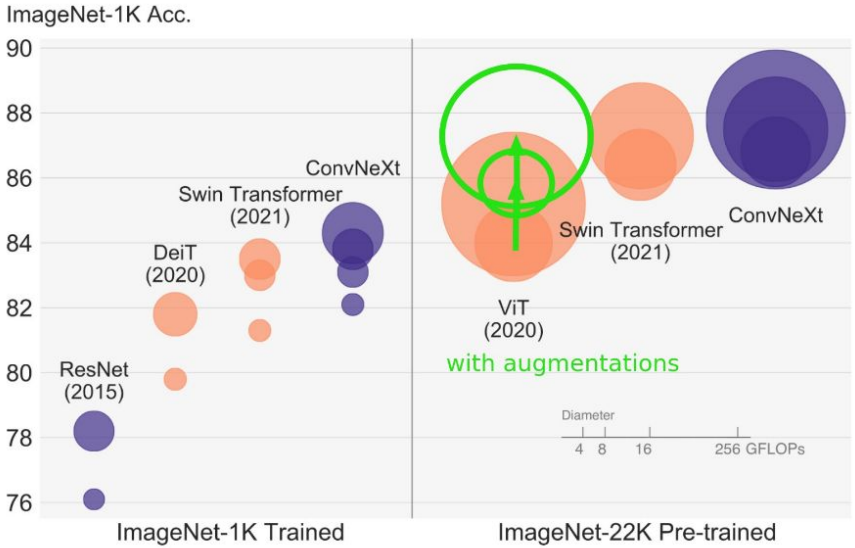


Figure from: CS231n Stanford Course

Convolutional Networks Analysis



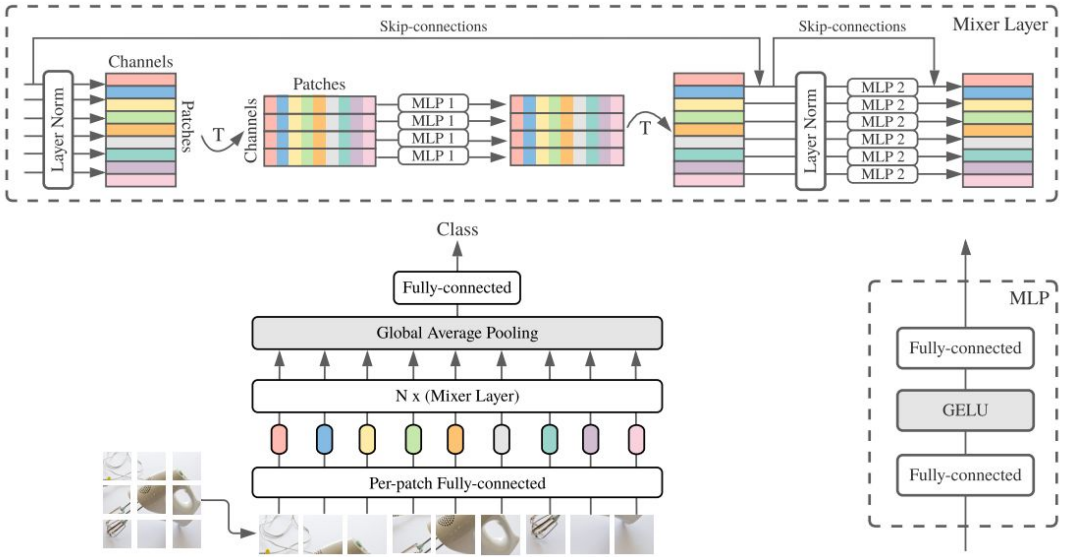


Canziani et al. [2016]: An analysis of deep neural network models for practical applications.
Liu et. al. [2022]: A ConvNet for the 2020s + Lucas Beyer correction https://twitter.com/giffmana/status/1481054929573888005

Future of Convolutions in Visual Processing

- Newer methods like Transformers and MLP-Mixers can bring benefits like long-range connection

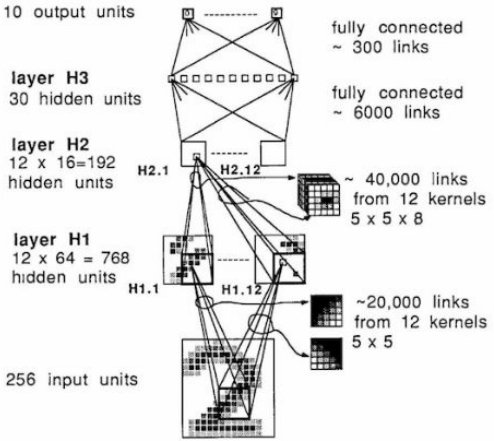
- Are still based on the same principles of
  - Locality
  - Stationarity

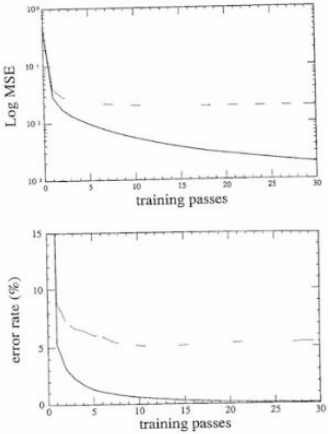


Tolstikhin et al. [2021]: MLP-Mixer: An all-MLP Architecture for Vision

Recap:

- Basic components remained stable
- In the last 10 years we added
  - Deeper and bigger **architectures** trained by SGD on GPUs
  - Better regularisation: **dropout**, **batch norm**
  - Better optimisation: **skip** connection, **adam**
  - More efficient use of params: **factorisations**





- Deep Neural Nets: 33 years ago and 33 years from now
  https://karpathy.github.io/2022/03/14/lecun1989/

New task: Classify image as cat or dog

Kaggle competition: https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition

Cats vs Dogs

- Deep ConvNets to the rescue
- First approach:
  - Train Resnet or Inception from scratch on all available dogs and cats data
- Downsides:
  - Require large amount of data
  - Long training time

Transfer Learning

- When learning a new skill, humans reuse knowledge and practices that they already learned before.
  - when learning physics we use knowledge learned from mathematics
  - when learning to play ukulele we benefit from already playing guitar
- How can we made ML models benefit from previously learned tasks?
  - Start with **good representations** of data
  - This could be given by a model already learned on a previous task
- Build **new** representations for a new problem, by building on the old representations

Transfer Learning

In practice, for computer vision tasks

- Start from a pretrained model on another task
  - large amount of data
  - Similar with the new task
- Remove the prediction layer
  - Usually, it is useful just for the old task
- Build a new model by adding one or more layer to the model
- Train the new model on the new task
  - Learn only the new layers (if the task is very similar) or
  - Learn the parameters of the entire model (fine-tunning)

Unsupervised learning? Self-supervised learning

- Humans and especially babies learn on their own (without supervision) by **observing** and **interacting** with the world. E.g. babies:
  - Observe and imitate their parents, without knowing what they are doing
  - Interact with their environment: play / taste objects
- Can ML models emulate this?
  - Design tasks where the goal is to predict something about the data itself
- Usually the methods are very similar or the same as used in supervised learning but the task has to be constructed from the data alone, without having access to any kinds of labels
- Generally to approaches: **generative** and **discriminative**

Generative methods

- **Generative:** learn to generate some parts of the input
- Common model: **auto-encoder**
  - Learn a model that reconstructs the input
  - Use the middle, bottleneck representations in a downstream task

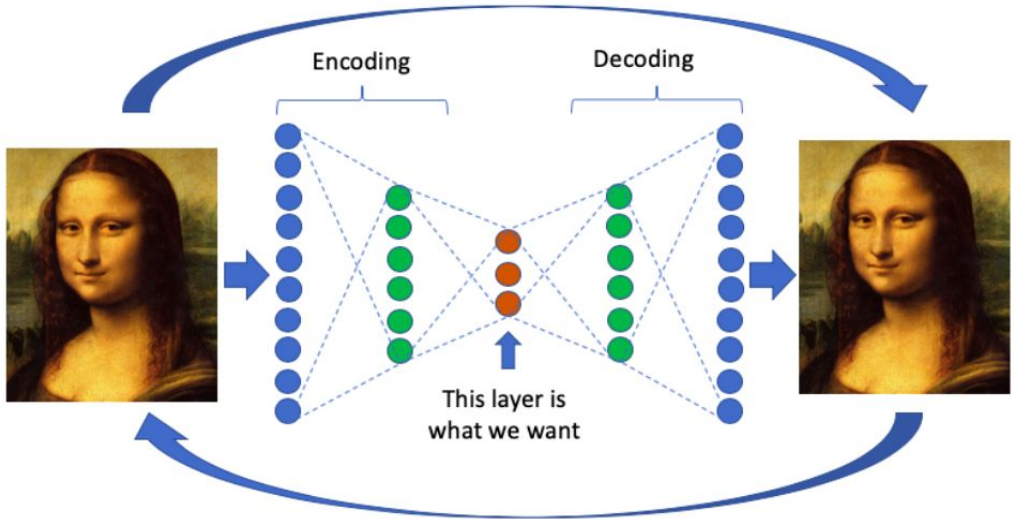
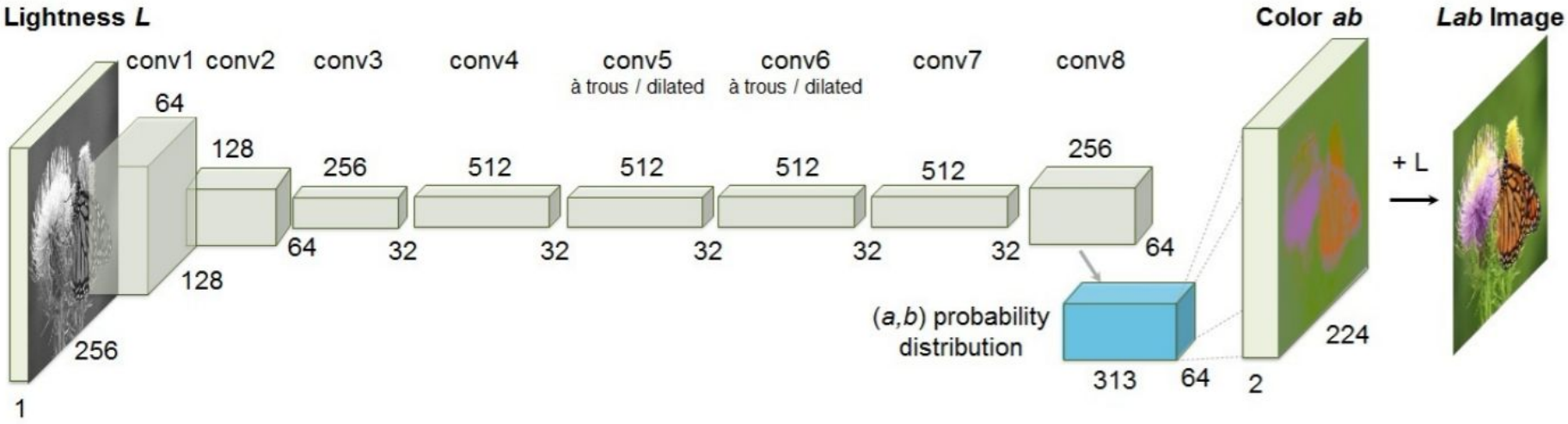


Image from: https://towardsdatascience.com/anomaly-detection-with-autoencoder-b4cdce4866a6

Generative methods: image colorization

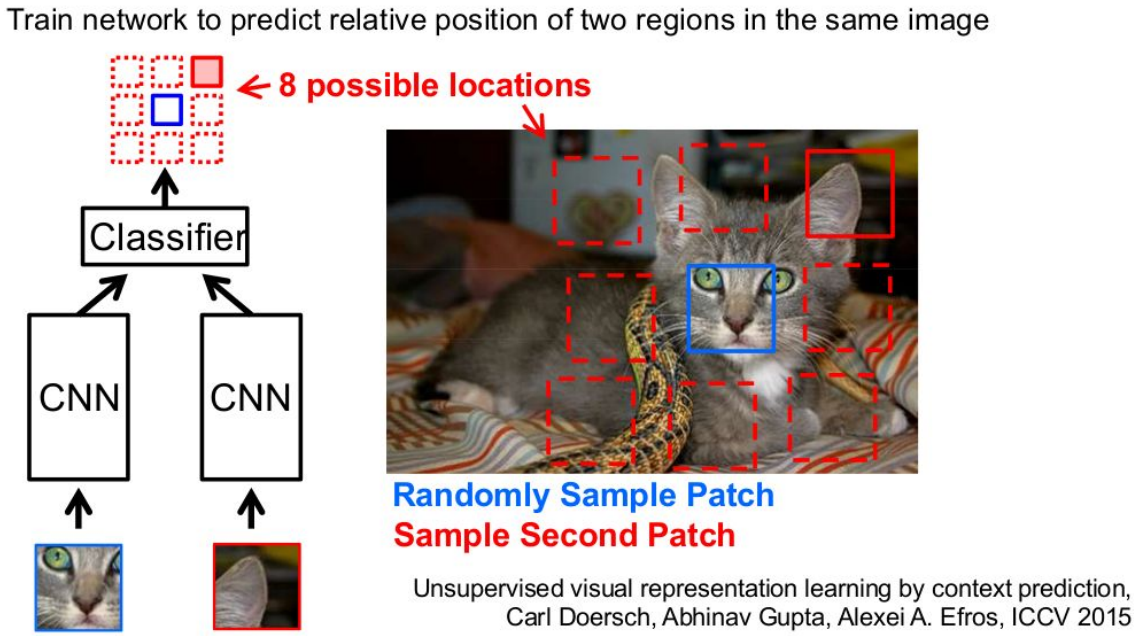


- Train a convolutional network to predict the colors from a gray image
- Re-use part of the model to obtain good representations

Zhang et al. "Colorful image colorization." *ECCV* 2016

Discriminative methods

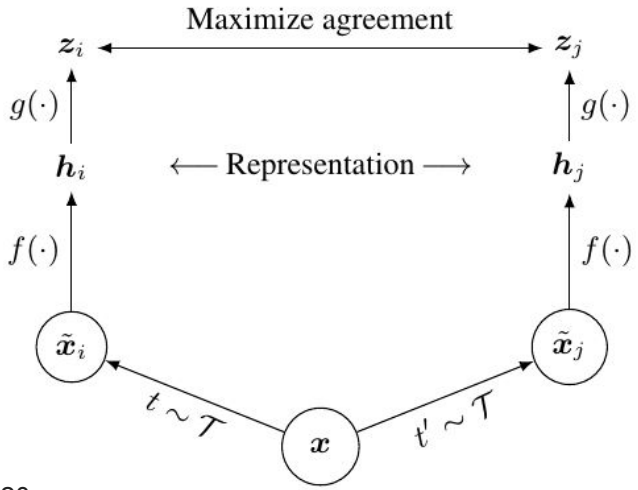
- **Discriminative:** design a task where the model should distinguish between some modifications of the data.



Slide from Andrew Zisserman's presentation

Contrastive Methods

- Current successful discriminative approaches: **Contrastive** methods
- Design a task to distinguish between **positive and negative pairs**
- Difficulty: finding meaningful categorisation in positive and negative pairs
  - Positive:
    - different parts of the same image
    - the same image transformed in different ways
  - Negative:
    - Random pairs of images



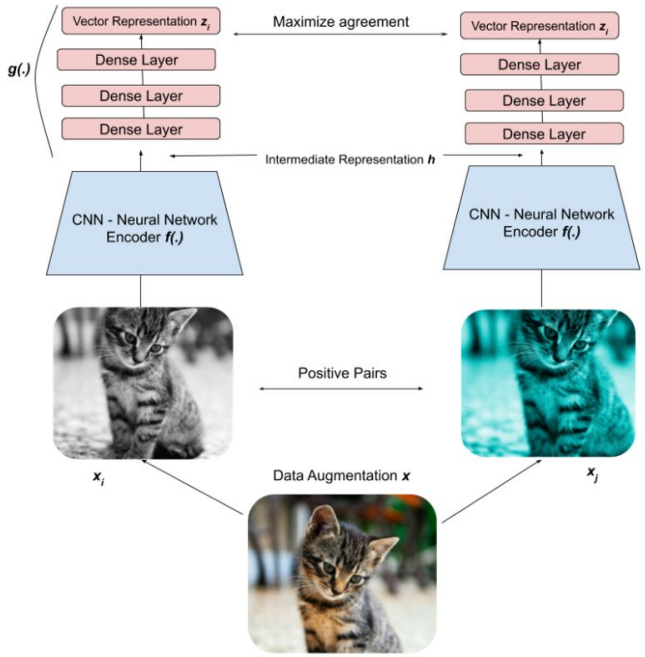
Chen et al. "A simple framework for contrastive learning of visual representations." *ICML* 2020.

Contrastive Methods

- Minimise generic Loss:

$$\mathcal{L} = -\sum_{(x,y)\in positive} \text{sim}(f(x), f(y)) + \sum_{(x,y)\in negative} \text{sim}(f(x), f(y))$$

- A generic contrastive method should learn
  - make the representations of positive pairs more similar
  - Make the representation of negative pairs less similar

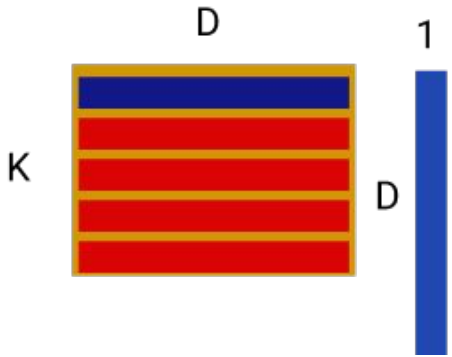


Chen et al. "A simple framework for contrastive learning of visual representations." *ICML* 2020.

Contrastive Methods

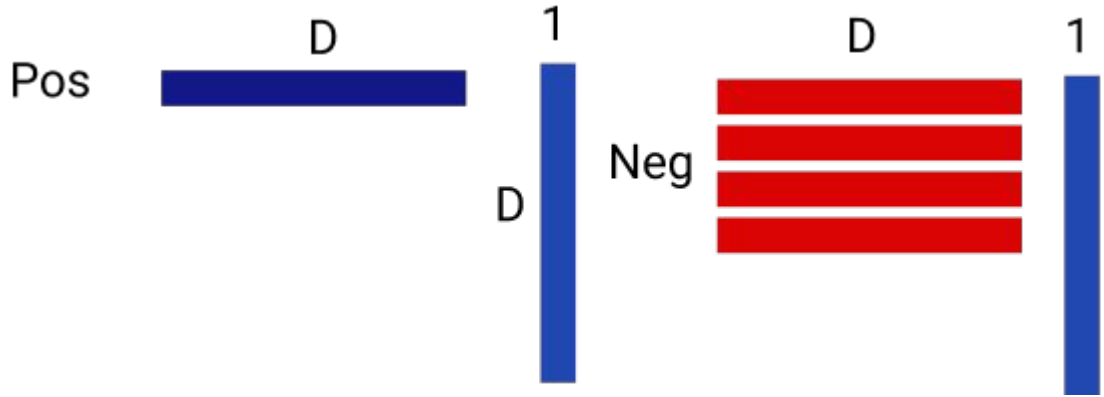
- Supervised cross-entropy loss

$$\mathcal{L}_{x,i} = -\log \frac{\exp(w_i f(x))}{\sum_{k \in 1..K} \exp(w_k f(x))}$$



- Self-supervised contrastive loss

$$\mathcal{L}_{x,y} = -\log \frac{\exp(f(y)^T f(x))}{\sum_{(x,z) \in neg} \exp(f(z)^T f(x))}$$



For more details you could see: https://gidariss.github.io/self-supervised-learning-cvpr2021/

Inductive biases of CNNs

- Convolutional networks are suited for visual tasks because they have inductive biases implicit in their architecture, that makes representing and learning the visual data more easily
  - Locality
  - Translation equivariance - (same output regardless of position)

- For visual data, CNN are more efficient than fully connected layer

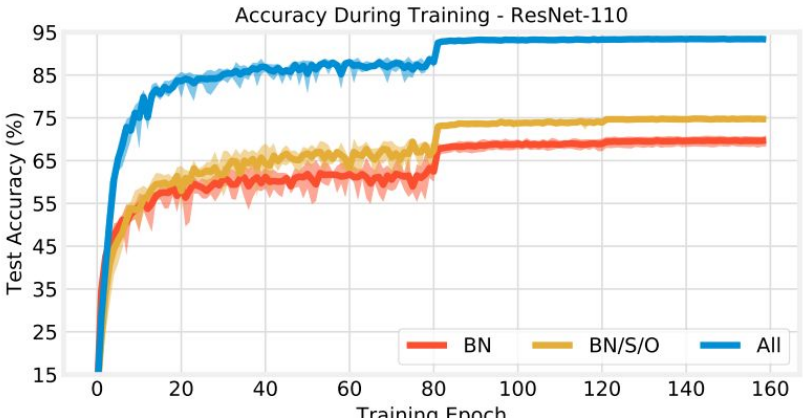
- But how much training is really necessary?
  - In some cases, tasks could be solved by learning just a small fraction of the parameters (but this is not the norm..)

Training BatchNorm and Only BatchNorm

Batch Norm

$$\hat{x}^{(i)} = \gamma \frac{x^{(i)} - \mu}{\sqrt{\sigma^2}} + \beta$$

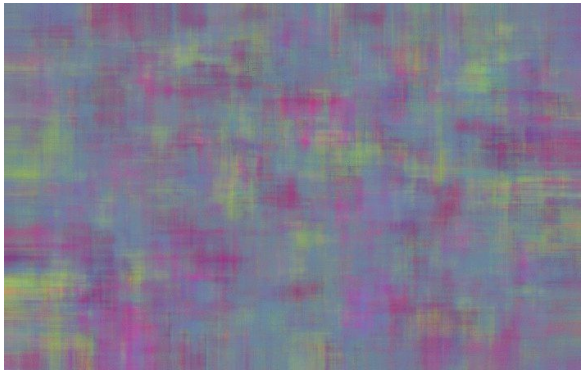
- Train only the scale and shift (gamma, beta) parameters of BN and keep randomly initialized all the other parameters of the network
    - ‘Surprisingly’ good results



Frankle et al. [2020] Training BatchNorm and Only BatchNorm: On the Expressive Power of Random Features in CNNs

Restore Images without training

- Generate an image starting from random noise vector
- Optimize a convolutional network to generate an image close to the input
  - **Without training** on any other images, just optimize on the **single** input
- Takes advantage of the inductive biases of the model to produce probable images



Ulyanov et al. [2018] Deep Image Prior

Thank you!

(Next: NN applications for Computer Vision)

Other Resources

http://cs231n.github.io/convolutional-networks/

http://cs231n.github.io/transfer-learning/

References

- A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. arXiv preprint arXiv:1605.07678, 2016.
- Frankle, J., Schwab, D.J. and Morcos, A.S., 2020. Training BatchNorm and Only BatchNorm: On the Expressive Power of Random Features in CNNs. arXiv preprint arXiv:2003.00152.
- I. Goodfellow, Y. Bengio, and A. Courville. Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.
- K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.
- S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37,ICML'15, pages 448–456. JMLR.org, 2015.
- A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.
- Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

References

- W. Luo, Y. Li, R. Urtasun, and R. Zemel. Understanding the effective receptive field in deep convolutional neural networks. In Advances in neural information processing systems
- K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.
- C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1–9, 2015.
- C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2818–2826, 2016.
- Ulyanov, D., Vedaldi, A. and Lempitsky, V., 2018. Deep image prior. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 9446-9454).