Advanced Machine Learning: Deep Neural Networks

Week 2

Function Approximation

- This course will focus on learning objectives
- E.g. supervised regression learning: optimize $arg\min_{\theta} \sum_{i} \ell(f_\theta(x_i), y_i)$
- In practice, we have to select f
- We can select linear models, boosted trees etc.
- Makes a big difference!

Deep neural networks

- DNNs disrupted the classical ML paradigm on multiple levels
- Deep architectures are **not** the focus of this course
- In practice, DL will be used for implementing our ideas
- This week will be devoted to an overview of DL

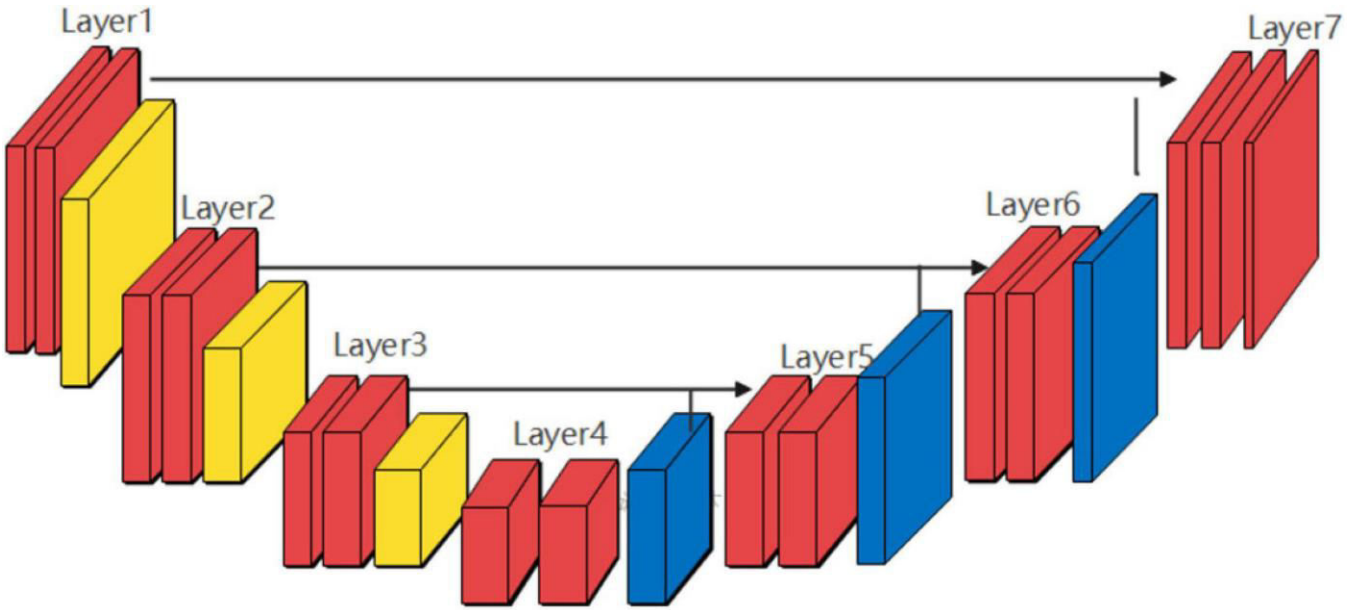
What is deep learning?

- In the past: a neural network with >1 hidden layers
- Yann Lecun: *“DL is constructing networks of parameterized functional modules & training them from examples using gradient-based optimization. That's it. This definition is orthogonal to the learning paradigm: reinforcement, supervised, or self-supervised.”*

Do you agree with Yann?

Deep learning architectures

- DL combines simple operations with learned parameters
- Combinations can be in series or parallel
- Consider this architecture:



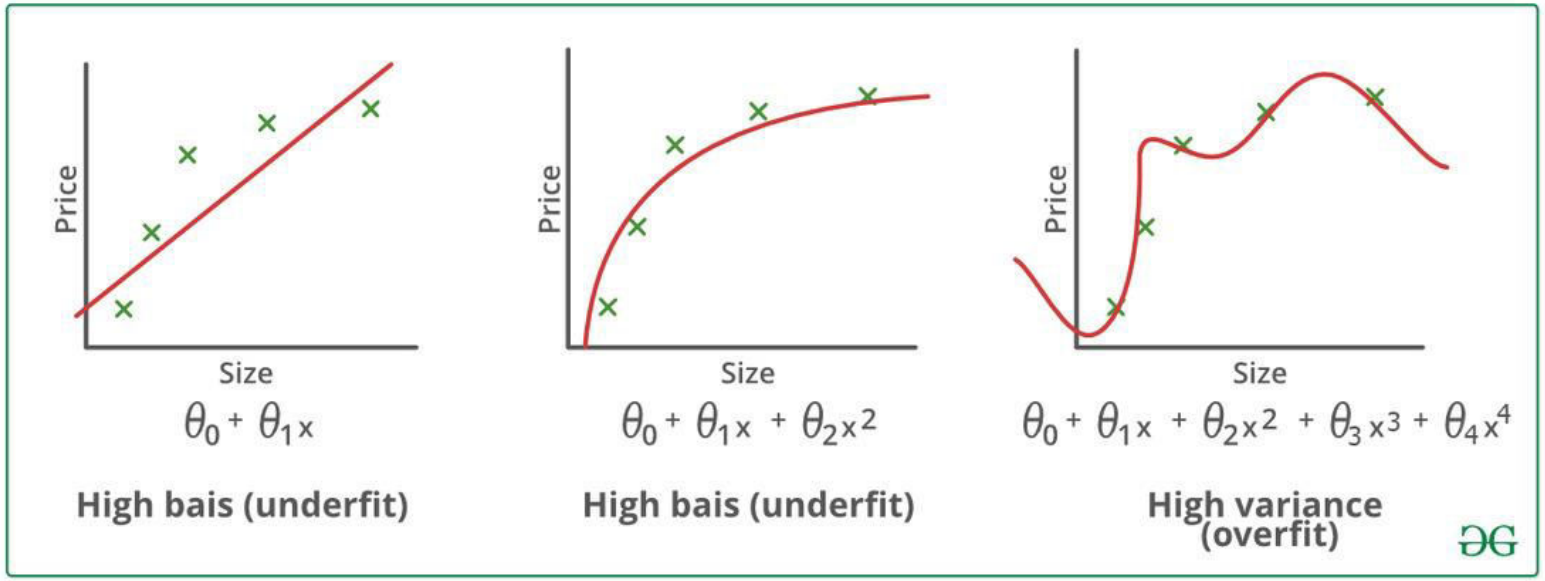
Expressivity

- The required operation must be expressible by the network
- Small networks may not approximate our desired task
- Not just size, need to have the correct type of operations

Which operations are neural networks not good at?

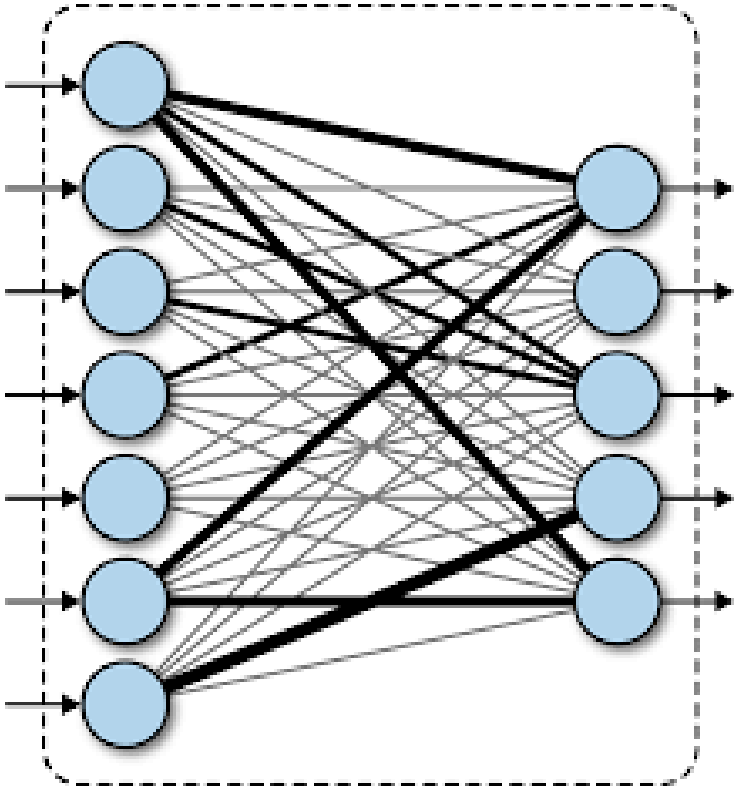
Design principles of deep NNs

1. Maximum expressivity: improving function approximation
2. Minimum unrequired parameters: reducing overfitting



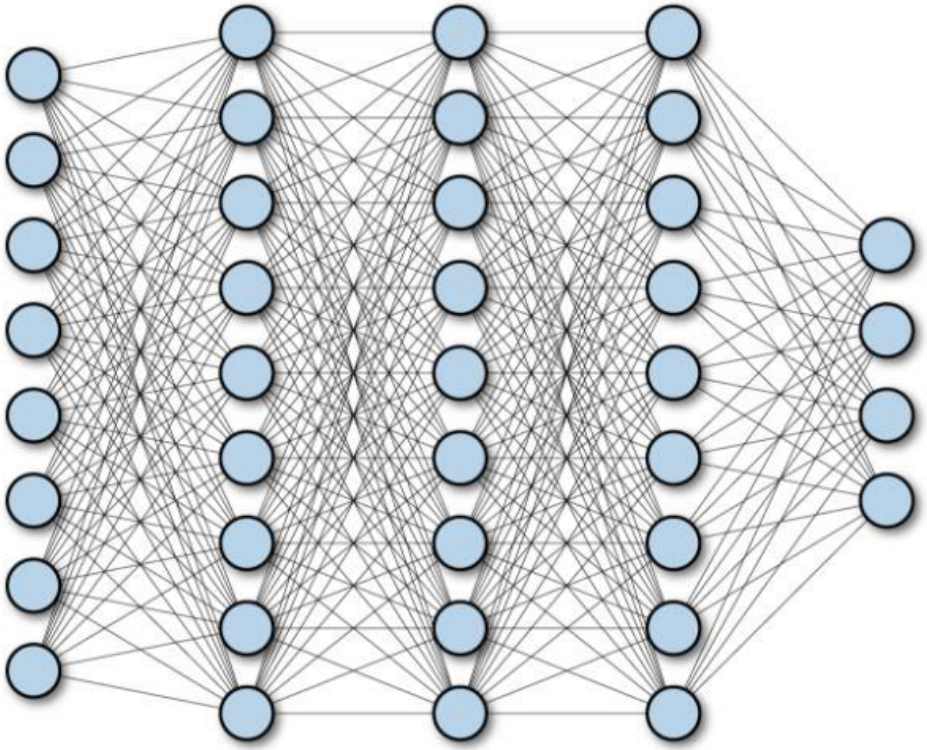
Fully-connected (FC) layer

- Each layer processes data
- It consist of a mathematical operation requiring parameters
- Typically includes a non-linear component
- Fully-connected (FC) layer + RELU: **z** <- W**x** + **b**
- ReLU non-linearity: **z** <- max(**z**, 0)
- Why do we add a non-linearity?



Stacking FC layers

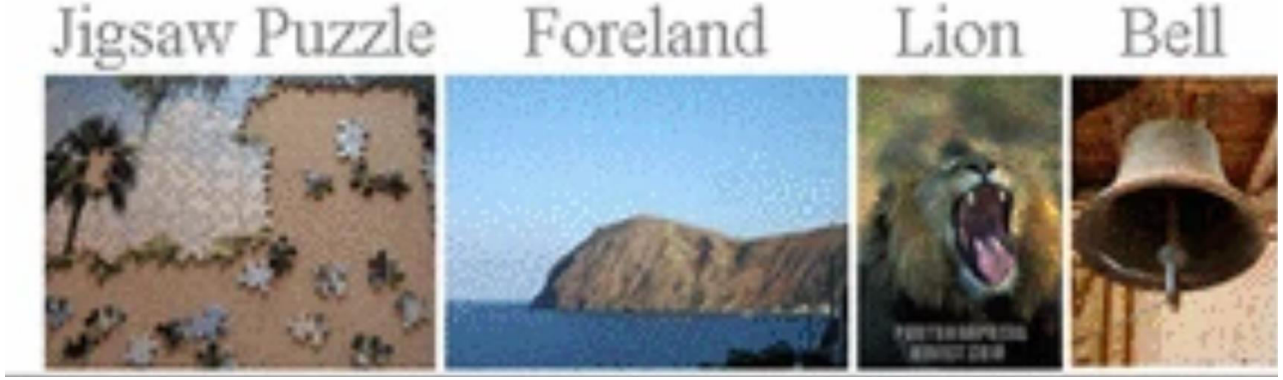
- A series of linear layers + ReLU
- Depth: number of layers
- Width: number of neurons per layer
- Map initial datum to a powerful representation

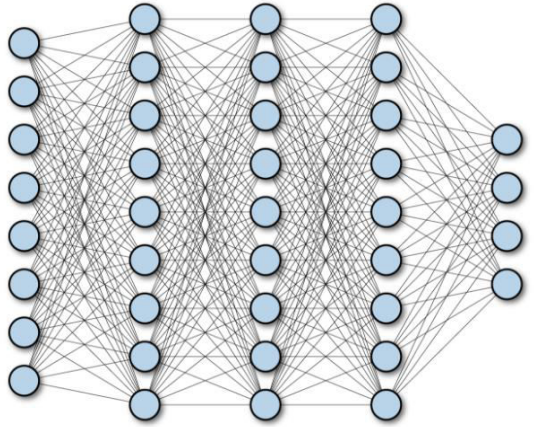


$$f_{W_1,b_1,W_2,b_2}(x) = \max(W_2 \max(W_1 x + b_1, 0) + b_2, 0)$$

What's a good output representation?

- Discuss in depth later in the course
- For now, features that a linear classifier/regressor is effective on
- Consider ImageNet classification:
  - Original pixels – terrible representation (linear classification accuracy <10%)
  - Neural network – much better (linear classification accuracy >90%)





How do we know what weights to use?

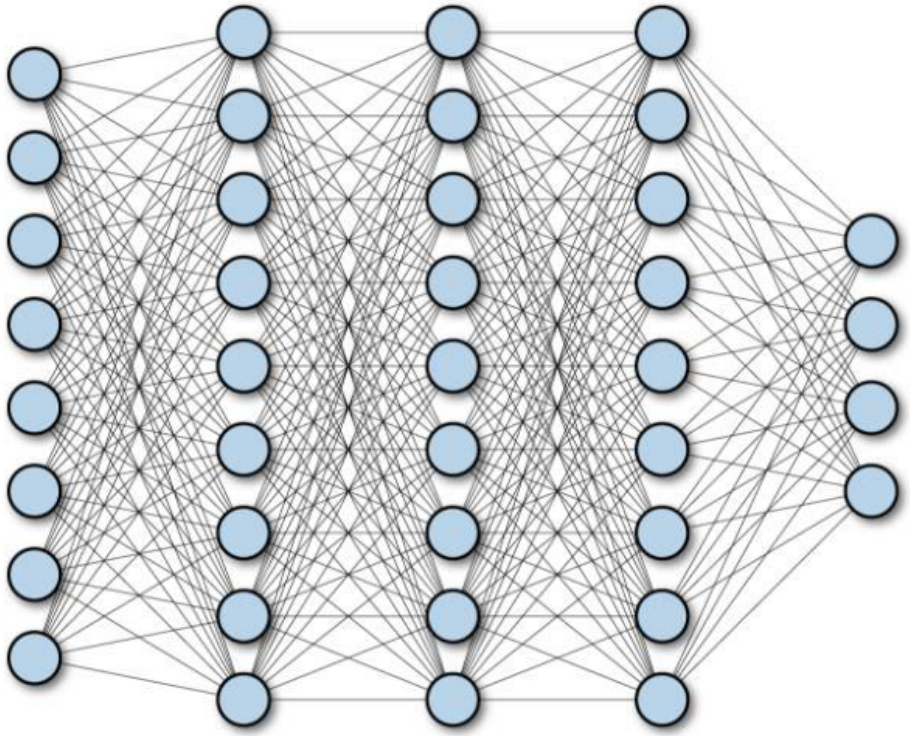
- Find best weights, given network architecture and training dataset
- For example:
  - our encoder is a 2-layer network, params $\mathbf{W}_1,\mathbf{b}_1,\mathbf{W}_2,\mathbf{b}_2$
  - Our linear classifier has parameters $\mathbf{W}_c,\mathbf{b}_c$
- The optimization problem becomes:

$$arg \min_{W_1,b_1,W_2,b_2,W_c,b_c} \sum_i \ell(c(f(x_i)), y_i)$$

- The weights are obtained by optimization (e.g. SGD + momentum)

FC Networks Do Not Scale

- Suppose the input was an image (10^6 pixels)
- Objective: map it to semantic representation



Option #1: Keep Input Dimension

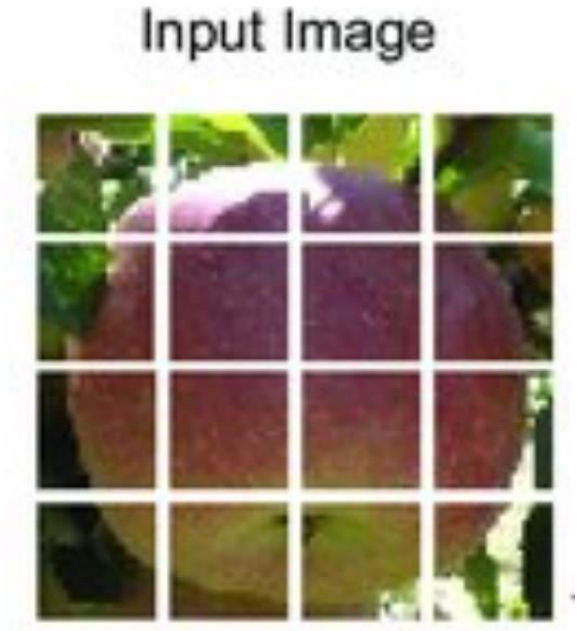
- All layers have output of same order as input
- When input 10^6 pixels and output 10^6 neurons – 10^{12} weights
- Computational cost is extremely high (very slow)
- Memory cost is extremely high (cannot store in RAM or disk)

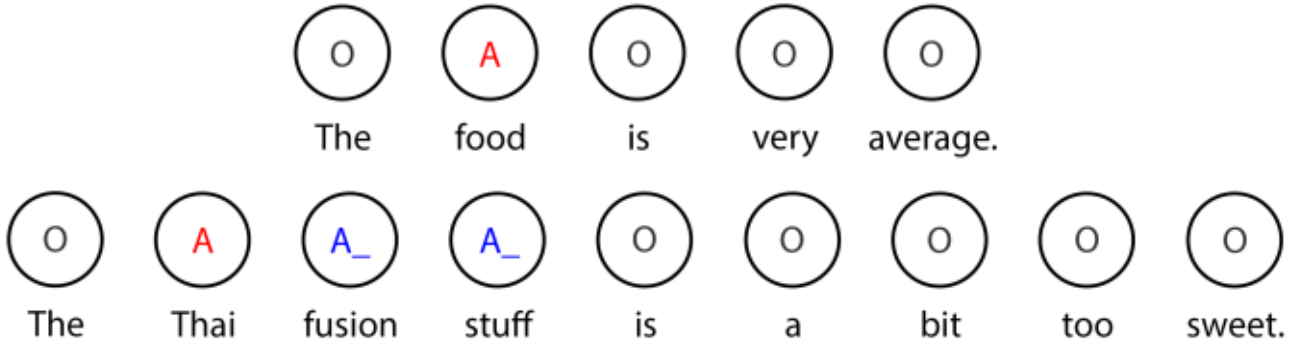
Option #2: Severely Compress Dimension

- Only feasible option: output is much lower dim than input
- First layer already compresses image X1000
- Such heavy compression using a linear layer loses too much info

A Better Approach: Tokenize Input

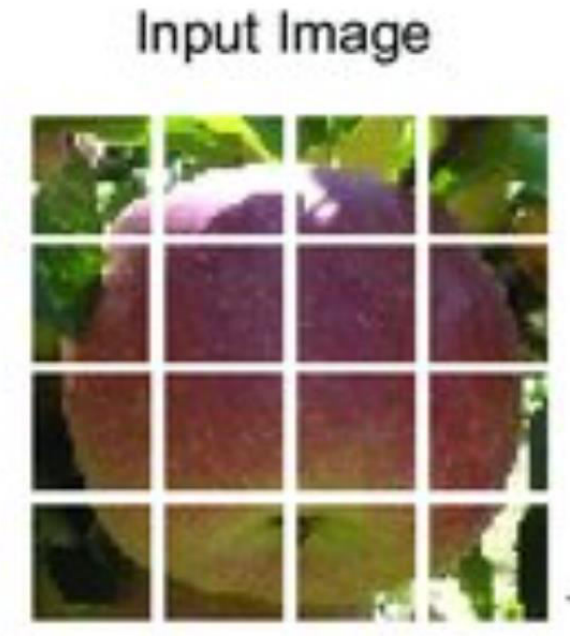
- Do not treat the entire input as one vector
- Insead, split it into multiple “tokens”
- Input features for each tokens





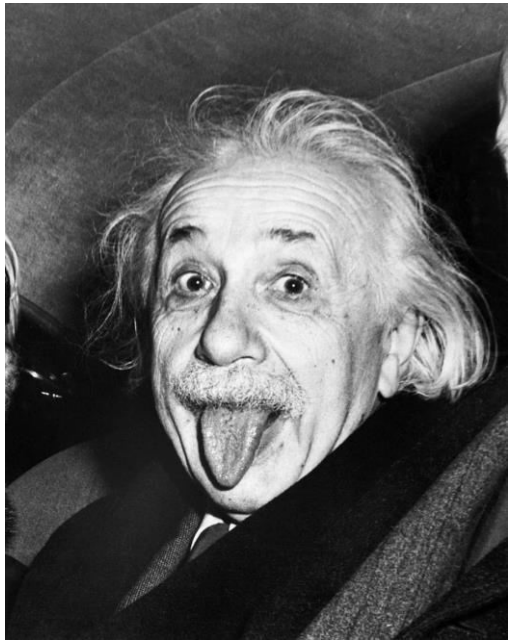
Run FC Network on Each Token

- In each layer, we run a FC layer on each token
- Complexity is ok – params 10^6, compute 10^9
- Cannot model correlations beyond each token



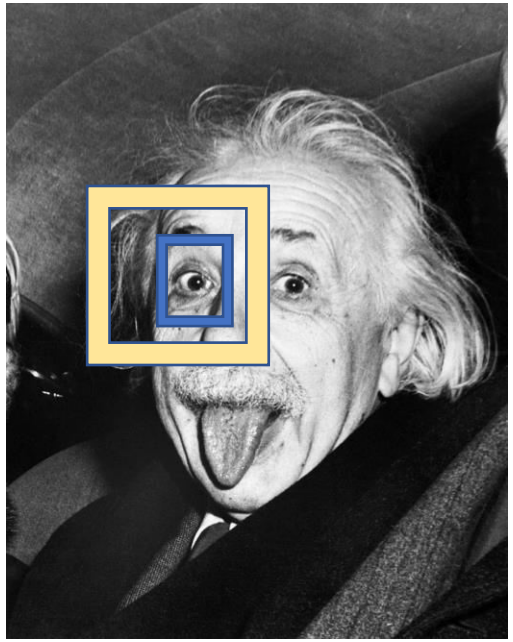
Adding Context Information

- For each token *i*:
  1. Retrieve other tokens relevant to token i - context
  2. Aggregate the information of the context tokens
  3. Add original token i information with context information
  4. Run FC network on this feature



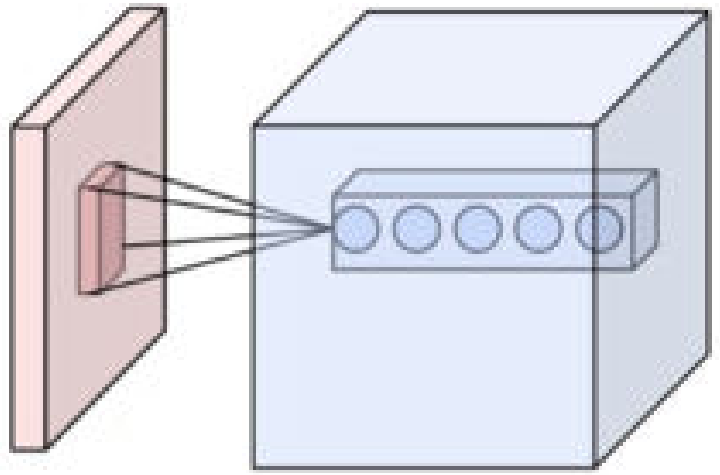
Local Context

- The simplest idea is to use the neighboring M*M tokens
- Motivation: nearest tokens are most relevant for understanding



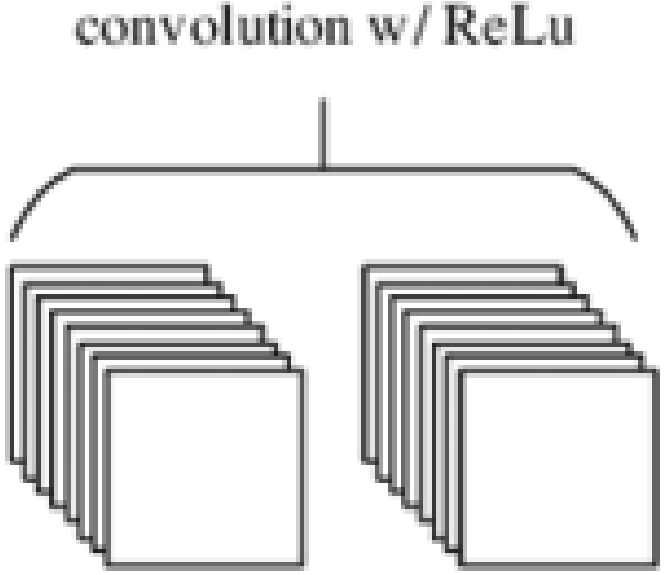
Convolutional Layers

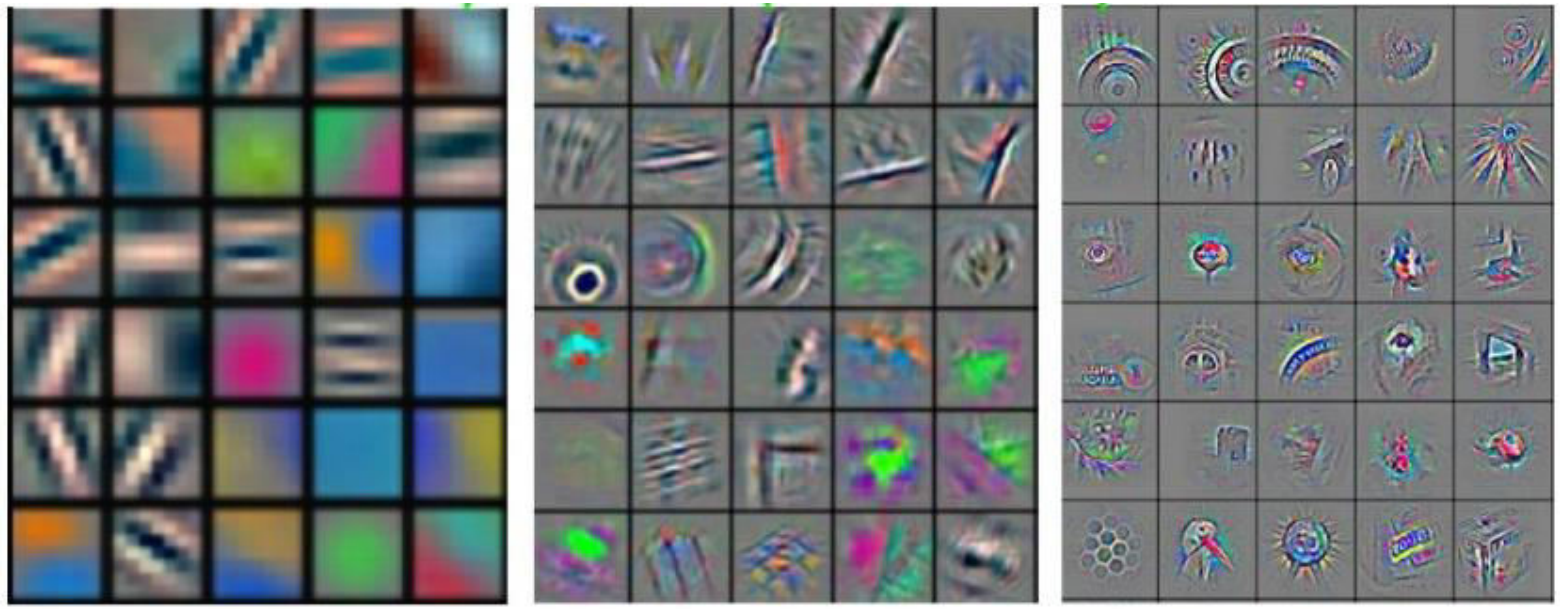
- Per-token context – concatenation of features of M*M nearby tokens
  - $\mathbf{t}(i) = [f_{i-1,j-1}, f_{i,j-1}, f_{i+1,j-1}, f_{i-1,j}, f_{i,j}, f_{i+1,j}, f_{i-1,j+1}, f_{i,j+1}, f_{i+1,j+1}]$
- FC layer operated per-token on the combined feature
- $\mathbf{t}(i) \leftarrow \max(0, W\mathbf{t}(i) + \mathbf{b})$



Stacking Conv Layers

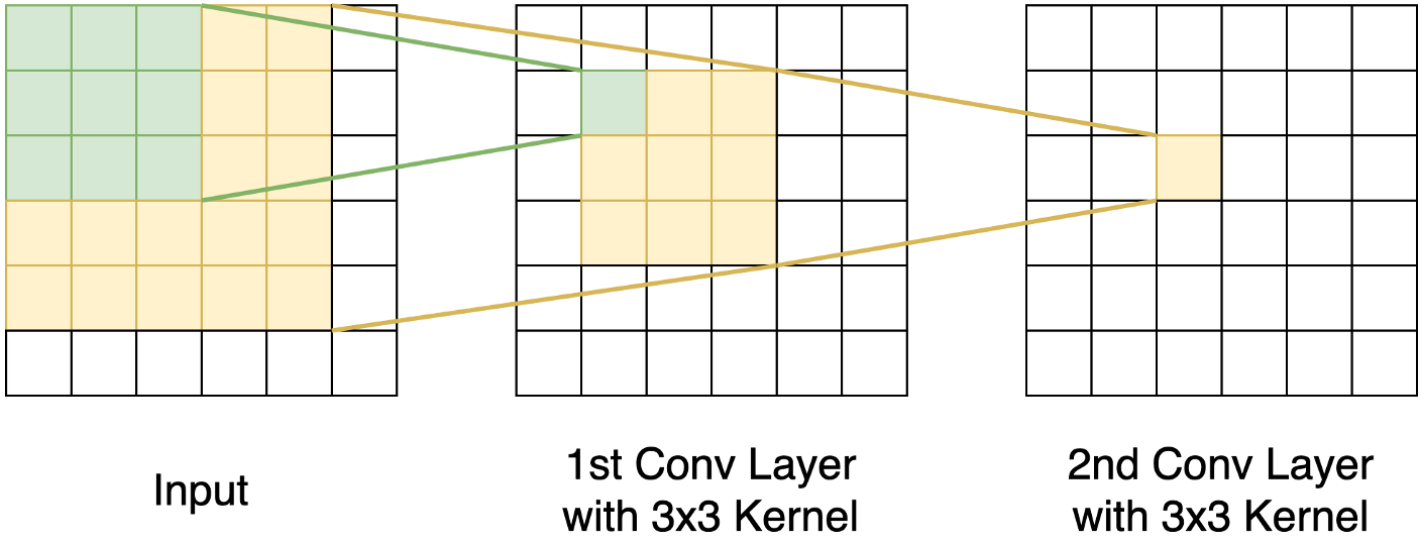
- A single conv layer can only learn simple patterns
- More complex patterns by stacking multiple conv layers
- E.g. first layer detects edges, second layer detects corners etc.

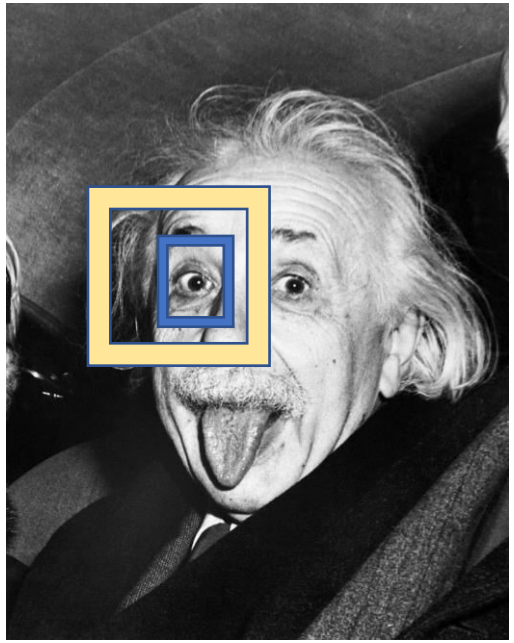




Limited Context

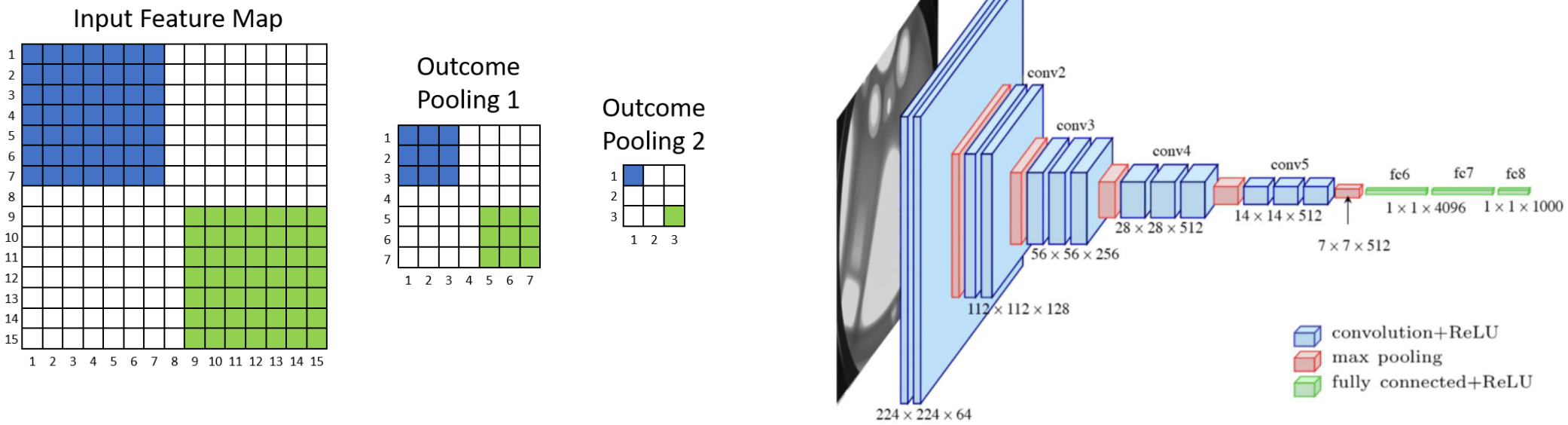
- Stacking up conv layers increases receptive field very slowly
- This means the left eye token is unlikely to see the mouth token





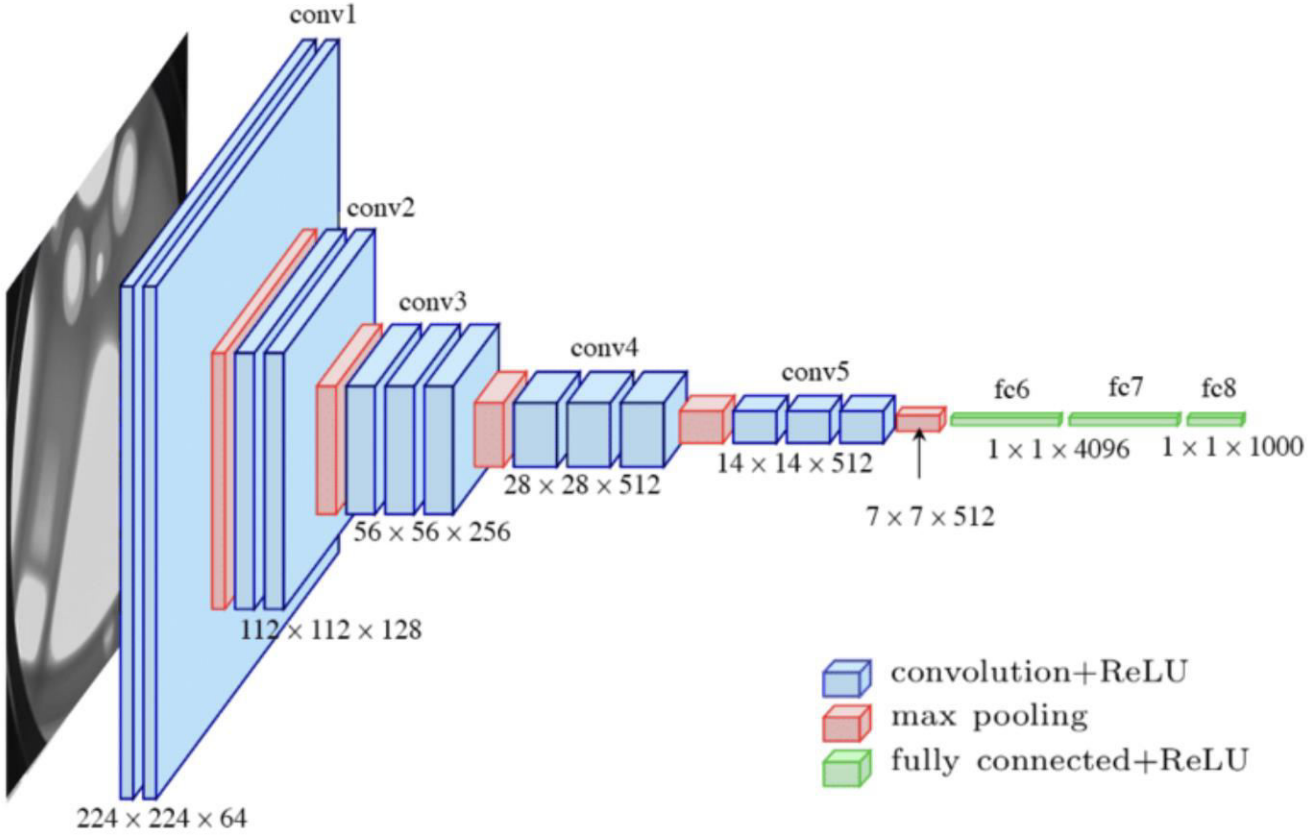
Feature pyramids for largest context

- Idea: pooling (downscaling) the data after a conv stack
- E.g. take the average of every 2X2 patch
- This reduces the number of tokens by 4
- Increases the receptive field by a factor of 2 in each direction



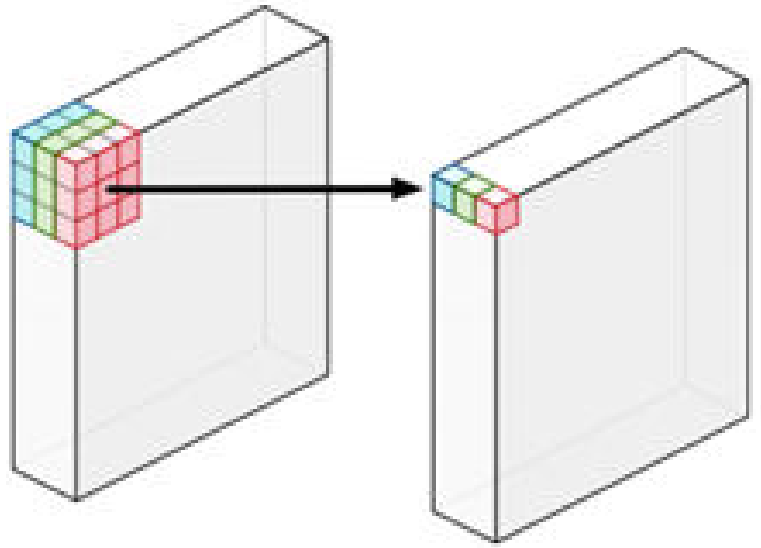
A look at a full conv neural network

- Typical architecture:
  - A stack of convolution layers linear models for small 3X3 patches)
  - A pooling layer
  - [X4]
  - A classifier



Token Averaging Instead of Concat.

- Traditionally, conv layers concatenated features of all neighborhood
- Becomes very expensive to have large neighborhoods, use 3X3
- Recent methods simply compute a learned weighted average
- Denoted “depthwise convolutions” – scales to large neighborhoods

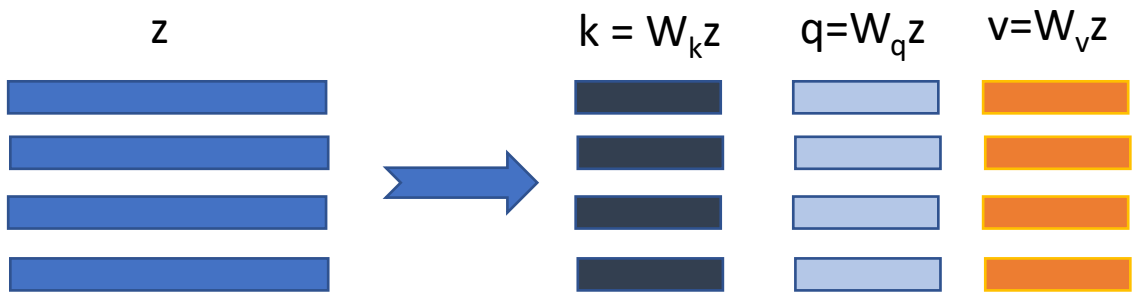


Does Context Have to be Local?

- CNNs use local context, in pyramids
- A good fit for many image tasks, but not so general
- How can we capture global context efficiently?

Self-attention layers

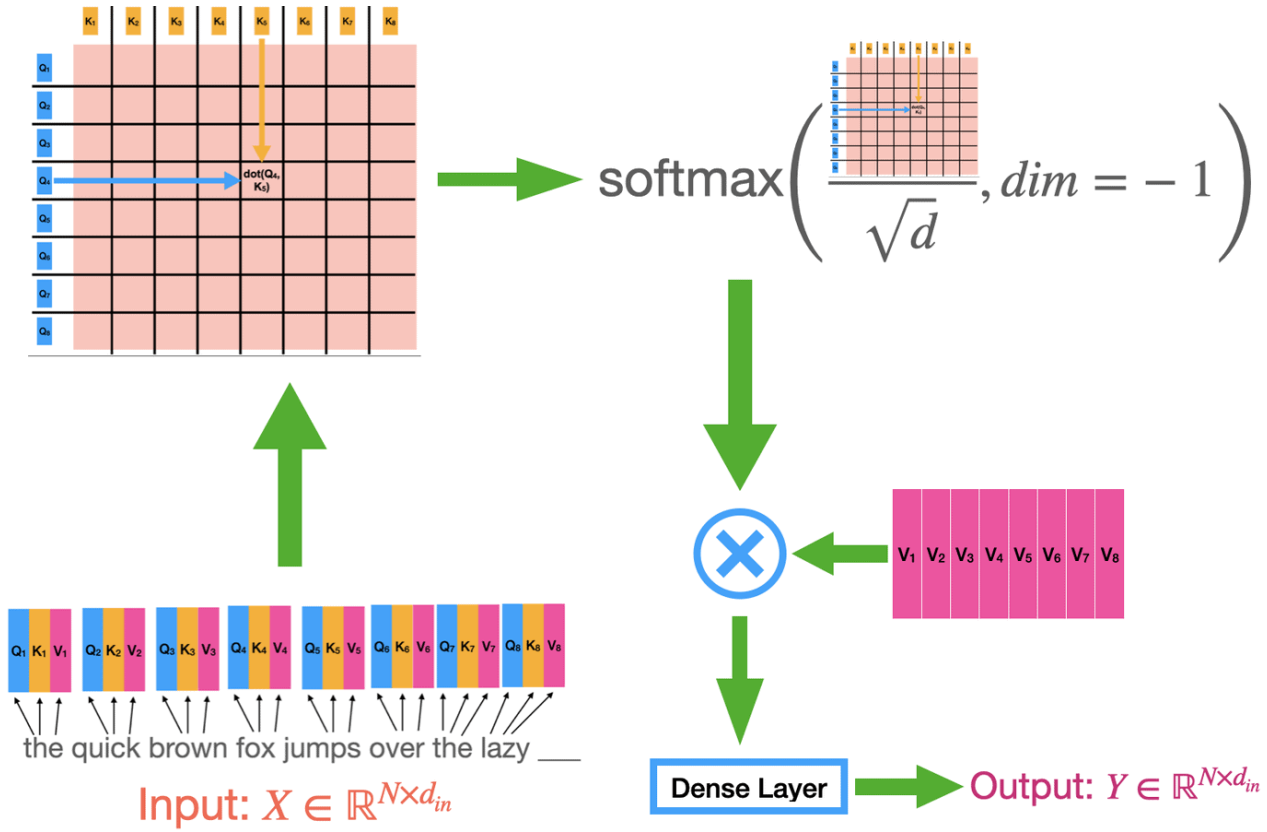
- A more advanced way of looking at large context
- Start with a set of token features
- Map each token feature into three z -> (key , query, value)



Self-attention layers (2)

- Compute self-attention by:
  - Computing the dot product between a token's query and all keys
  - Passing the result through a softmax layer resulting in probability
  - The probability vector is called attention

$$A = \text{softmax}\left(\mathbf{q}\mathbf{k}^\top / \sqrt{D_h}\right)$$

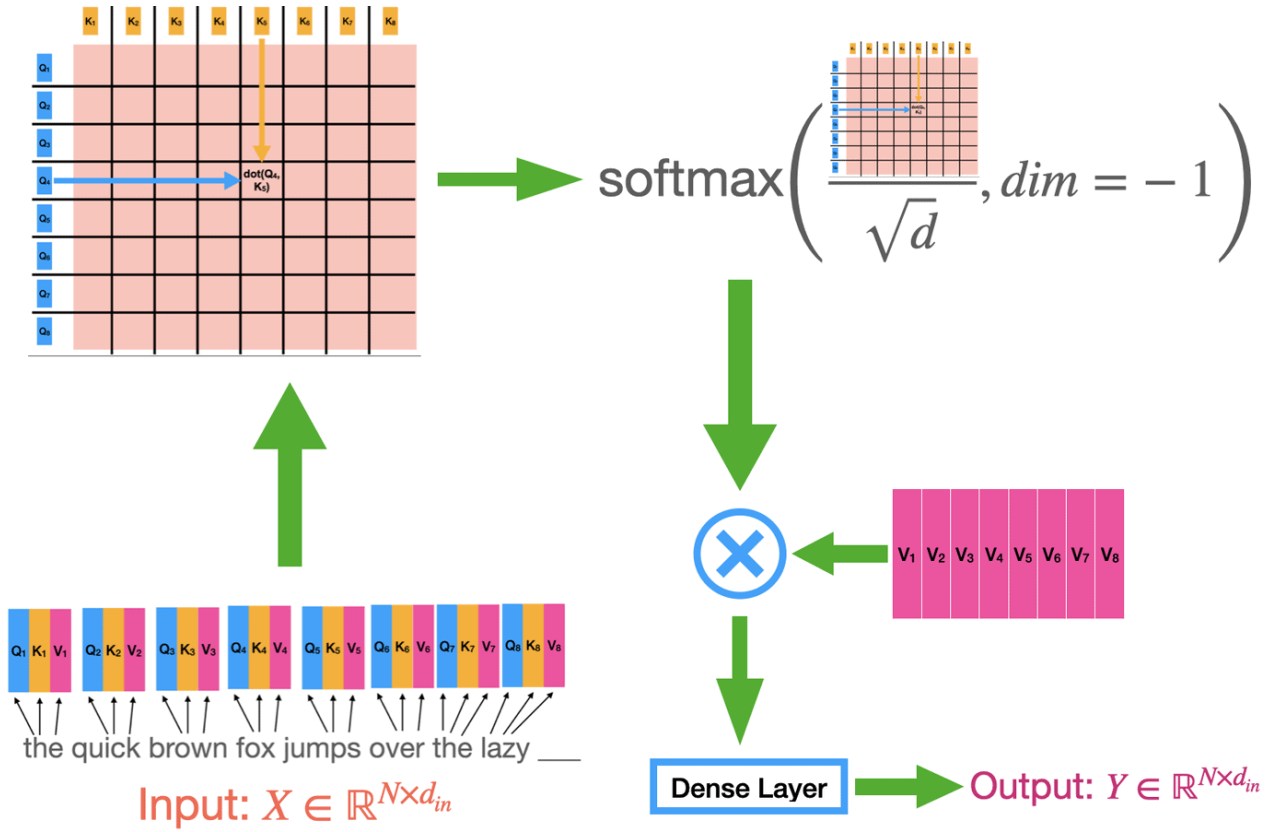


Self-attention layers (3)

- The context for each token is given by a sum of all value vectors, weighted by the attention

$$A = \text{softmax}\left(\mathbf{q}\mathbf{k}^\top / \sqrt{D_h}\right)$$

$$\text{SA}(\mathbf{z}) = A\mathbf{v}\,.$$



Using the Global Context

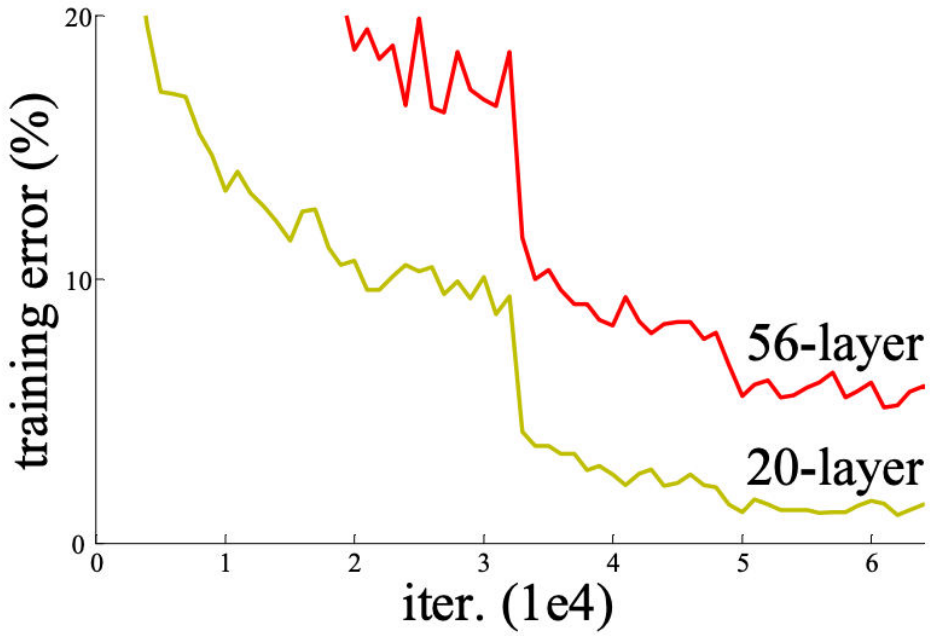
- We can now combine the global context with the token features
- E.g. z + SA(z)
- We now apply the FC layers on the features of each token

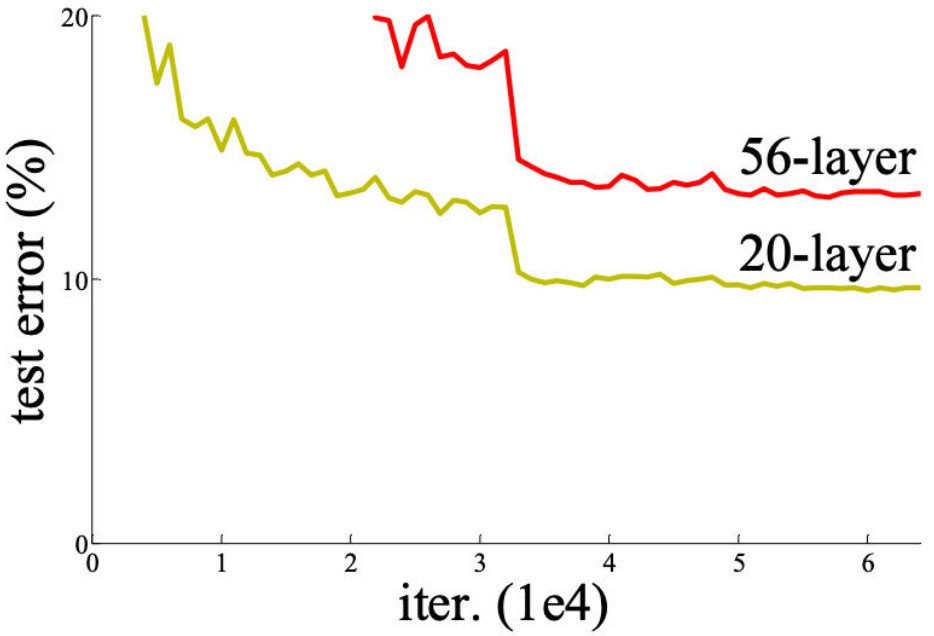
Differences Between the Contexts

- Standard CNN:
  - Small local context becomes global via pyramid
  - Concatenation
- Modern CNNs: large local context
  - Large local neighborhood
  - Weighted averaging (weighting per each dimension, per context token)
- Transformer
  - Adaptive global context
  - Weighted averaging, but weighting in per context token, not per-dimension
  - Last limitation can be mitigated by multiple attention heads, out of scope.

Optimization issues

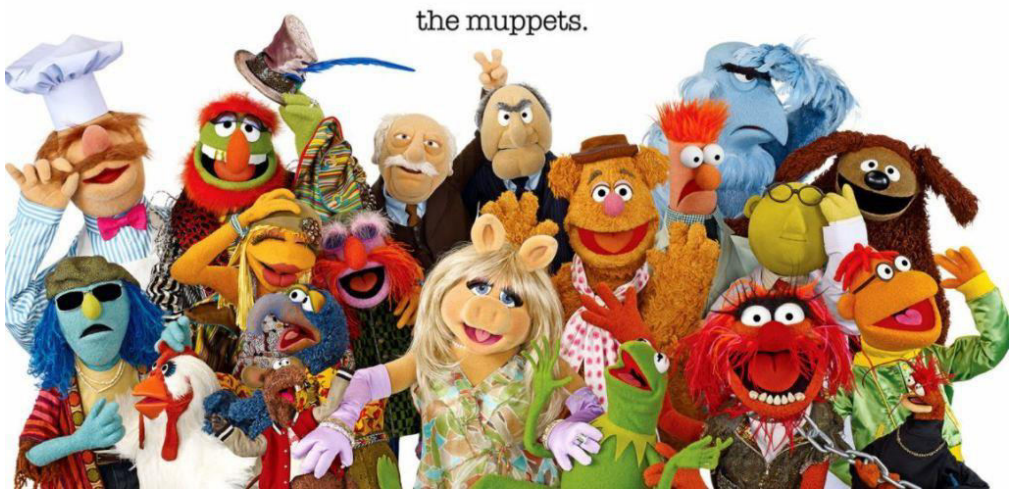
- It turn out that once neural networks get deep optimization fails





Why did that happen?

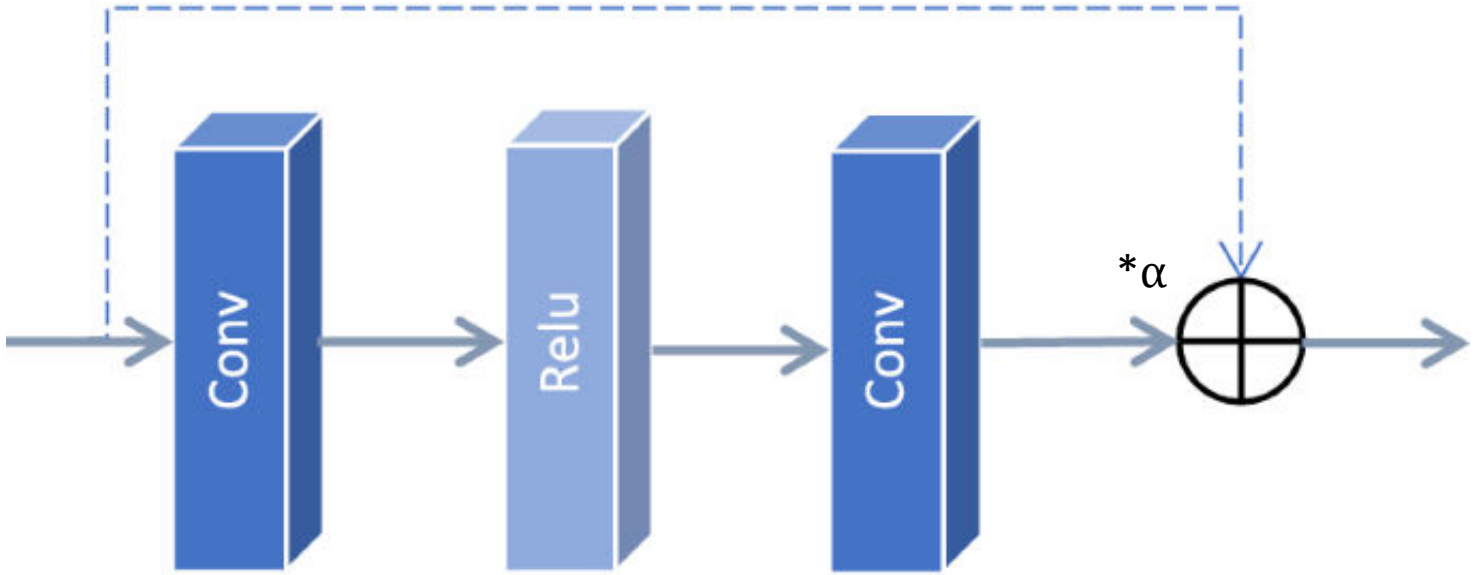
- Credit assignment through poorly intialized layers
- Consider:
  - You are a bad employee in a large factory where everyone is a superhero
  - You are a bad employee in a large factory where everyone is a muppet
- Where would the manager be able to give feedback?



Residual connection

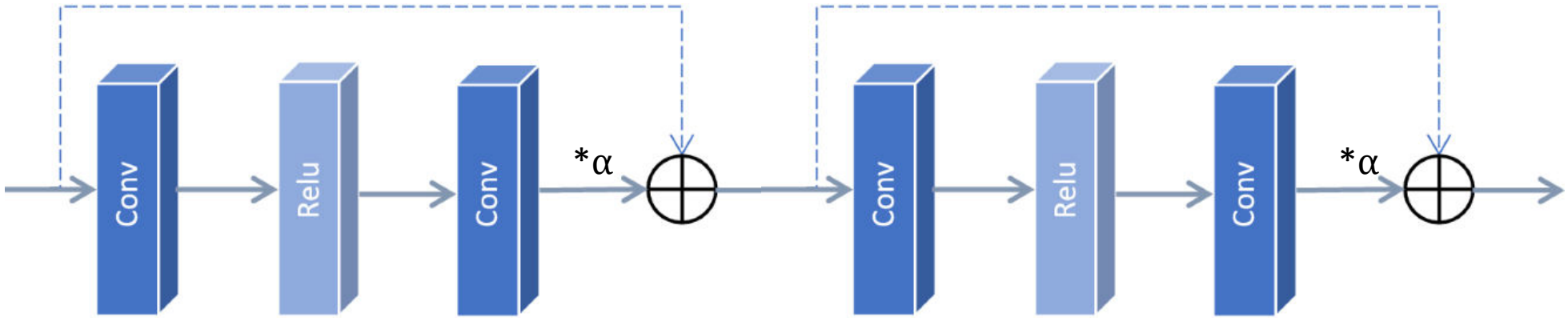
- Idea: provide direct feedback from the output to each weight
- Implemenation: residual (skip) connection
- E.g. $f(\mathbf{U}x) = \alpha \max(\mathbf{W}_2 \max(\mathbf{W}_1(\mathbf{U}x) + \mathbf{b}_1, 0) + \mathbf{b}_2, 0) + \mathbf{U}x$

Layer Scale



Multiple Residual Blocks

- α is initially small
- At first approximation the input to each block is the input x
- A deep ResNet initially approximated as ensemble of shallow blocks
- Easy to optimize!

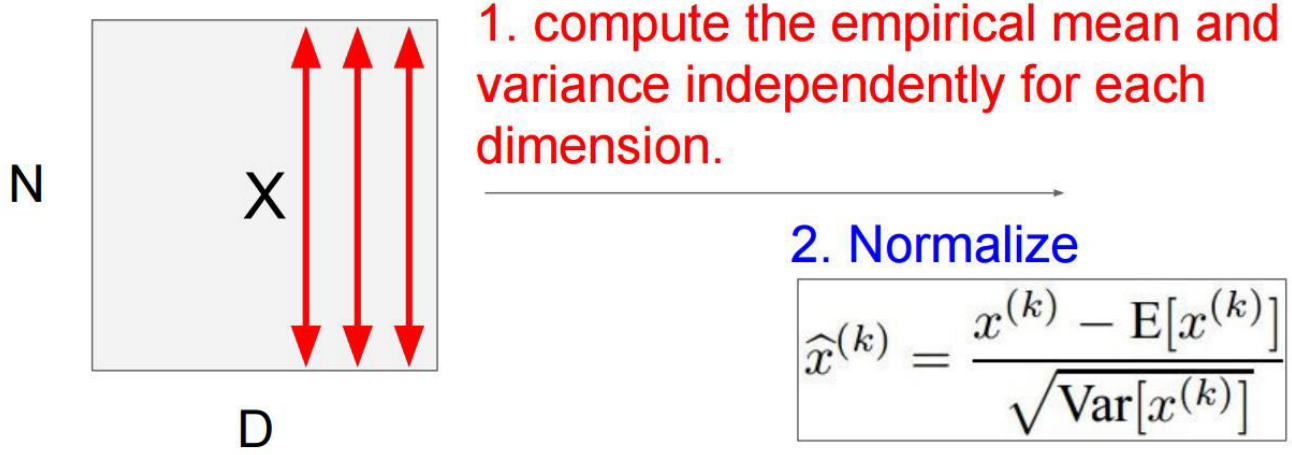


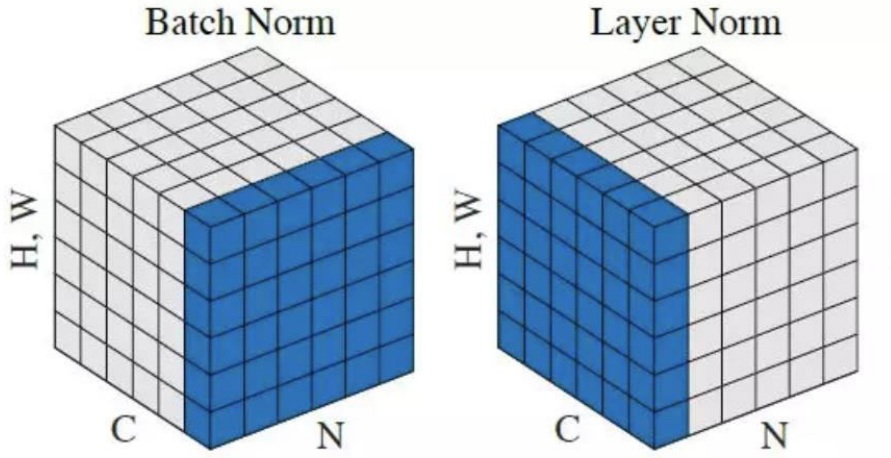
Normalization

- SGD works better for well conditioned data (e.g. normal distribution)
- The cumulative effect of multiple layers can result is large numbers
- Even a single layer blowing up can hurt optimization

Batch/Layer normalization

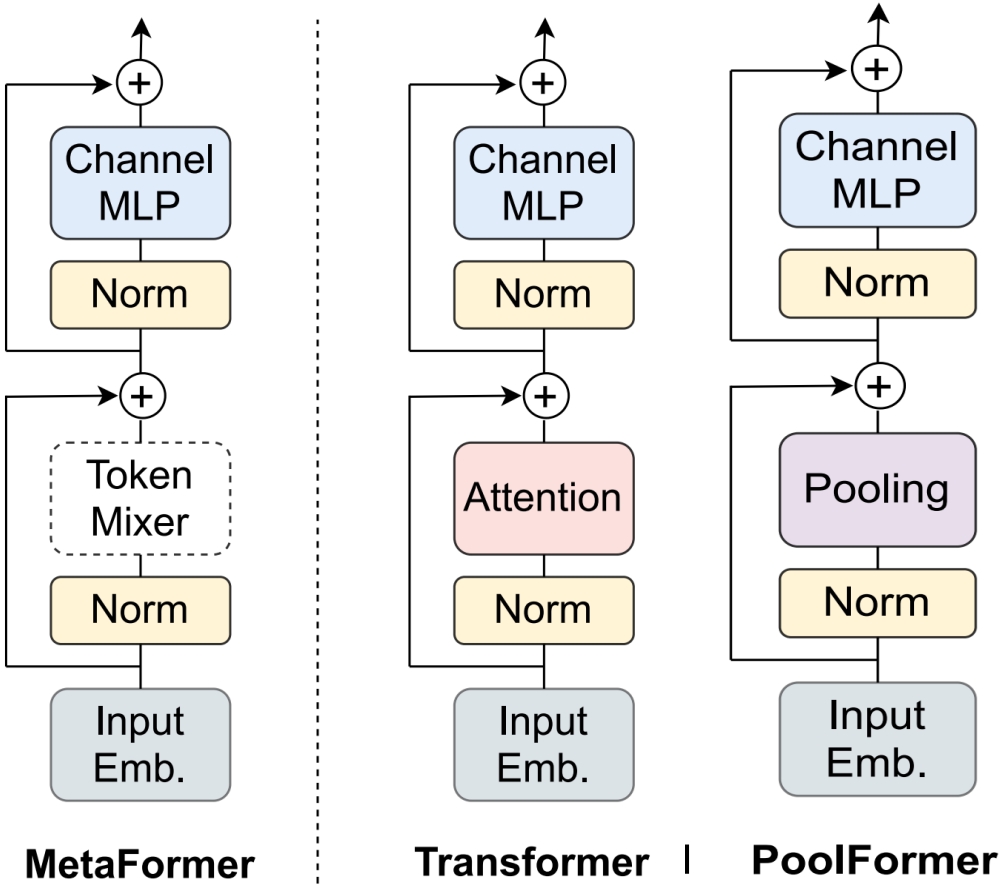
- The output activations for each channel normalized by mean and std
- Layer norm: compute stats across all pixels of a single example
- Batch norm: compute stats across all pixels of a multiple examples

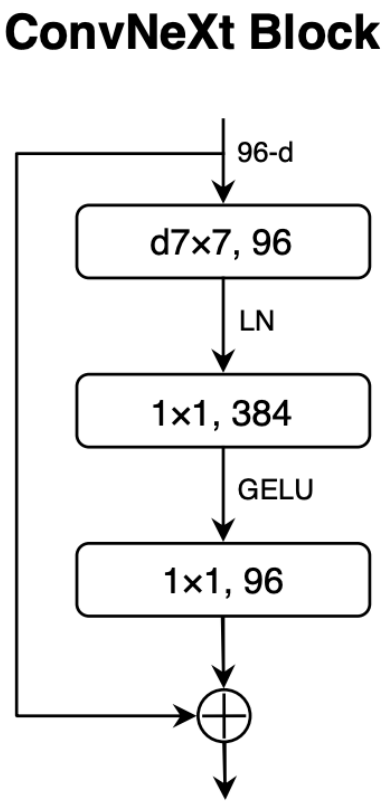




Final Blocks

https://github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/simple_vit.py
https://github.com/facebookresearch/ConvNeXt/blob/main/models/convnext.py

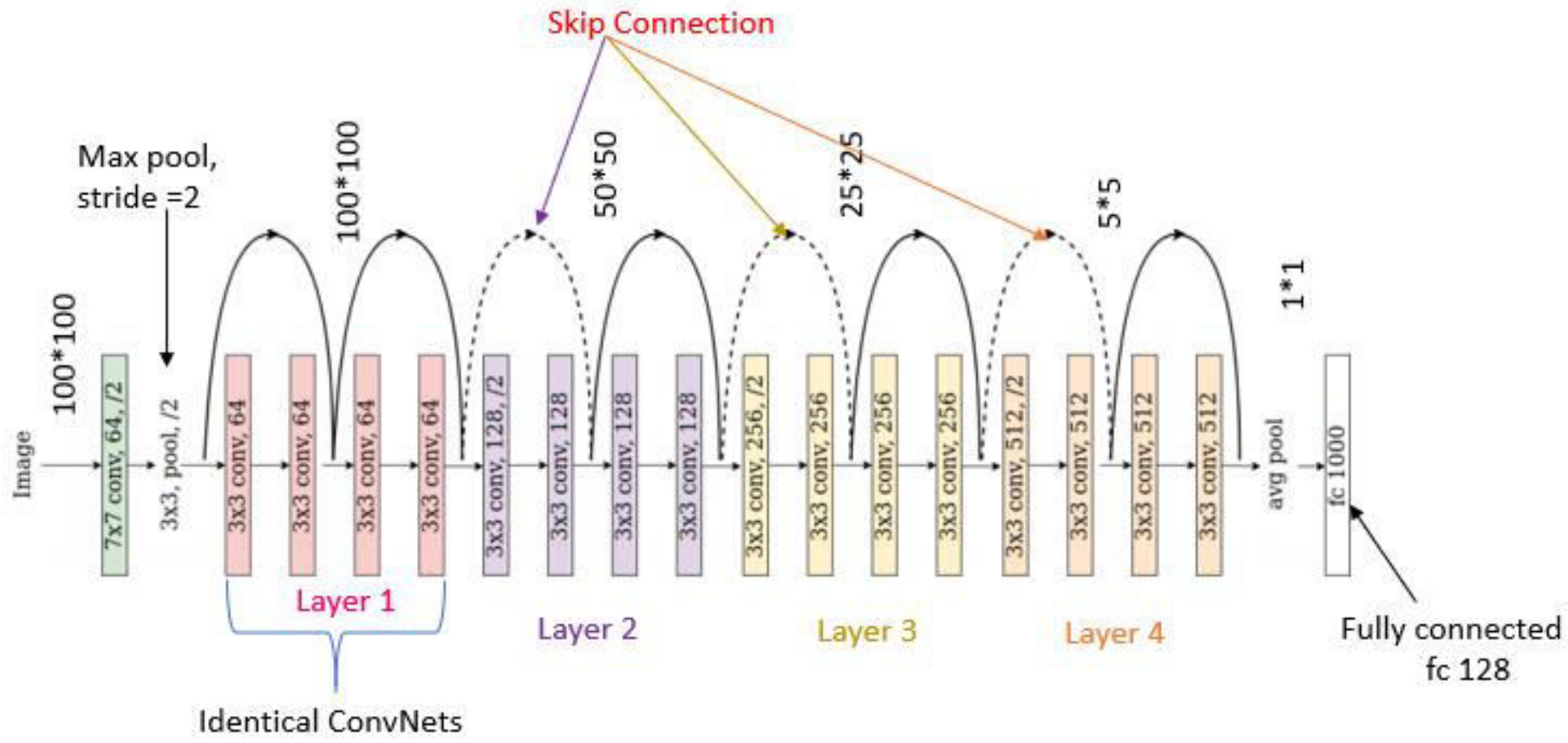




ConvNeXt – uses a very simple weighted average context, so skipped its norm and skip connection

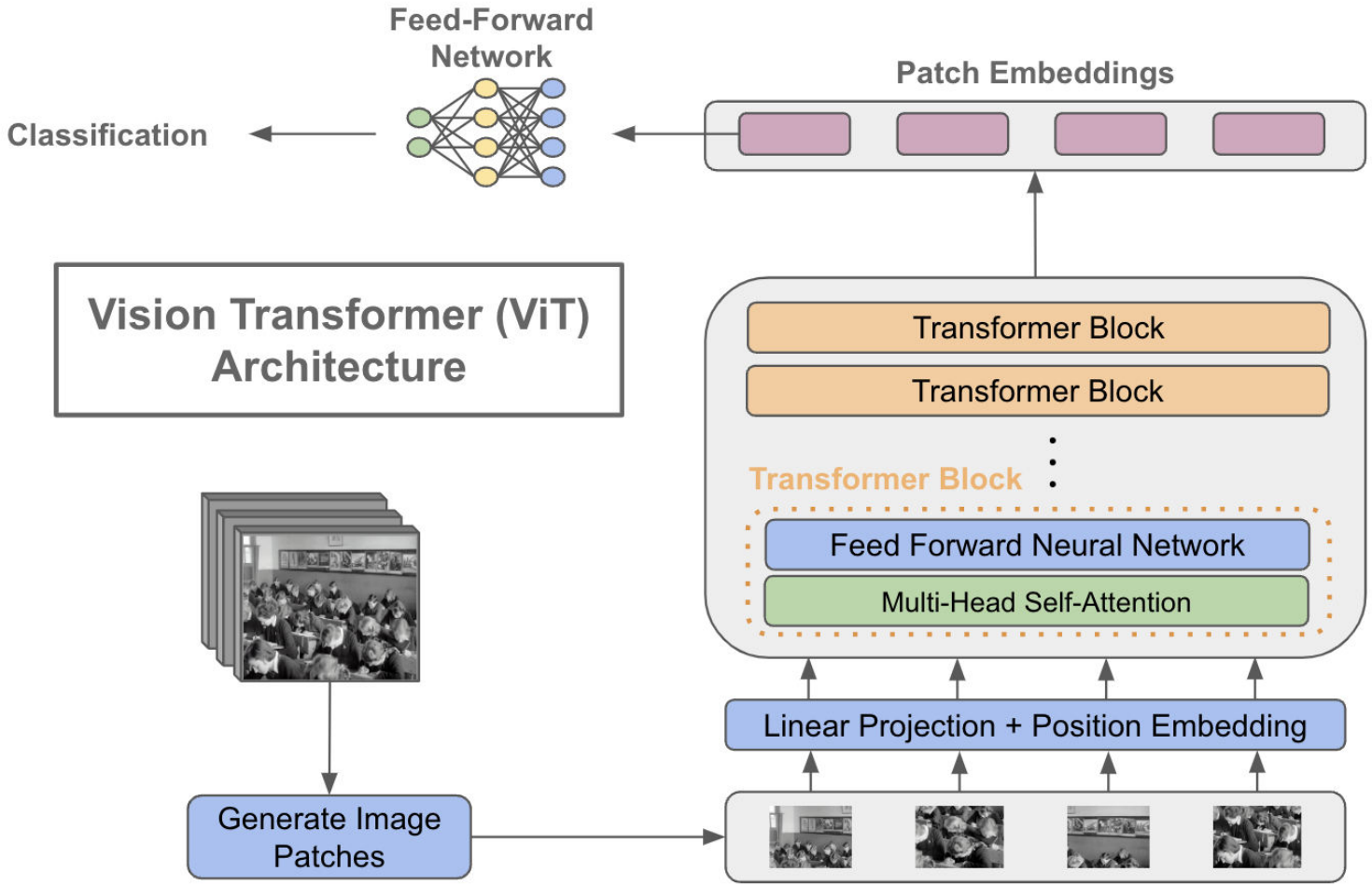
A full residual neural network

- This network has 18 layers (it's a small one, it can have >1000 layers)



A look at a full transformer

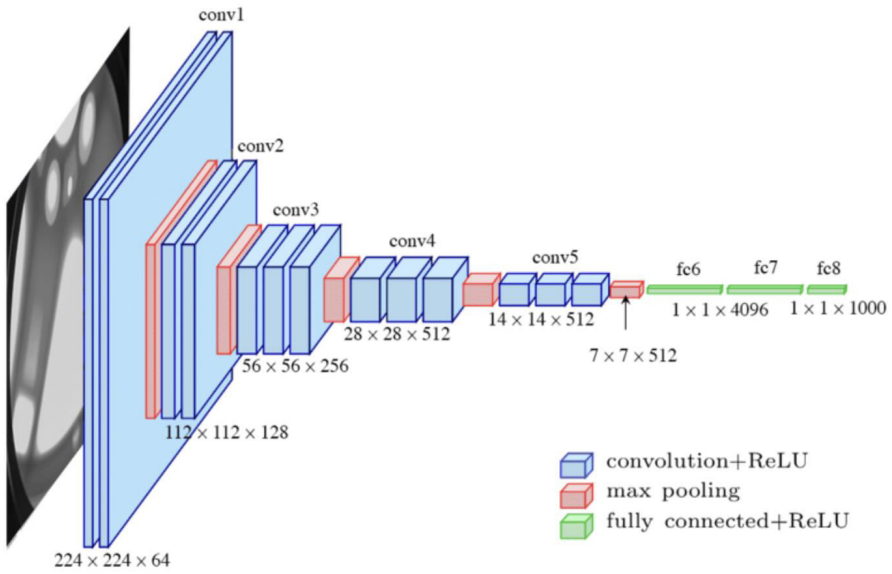
- Stack of SA + MLP
- Classifier on final token
- Single resolution

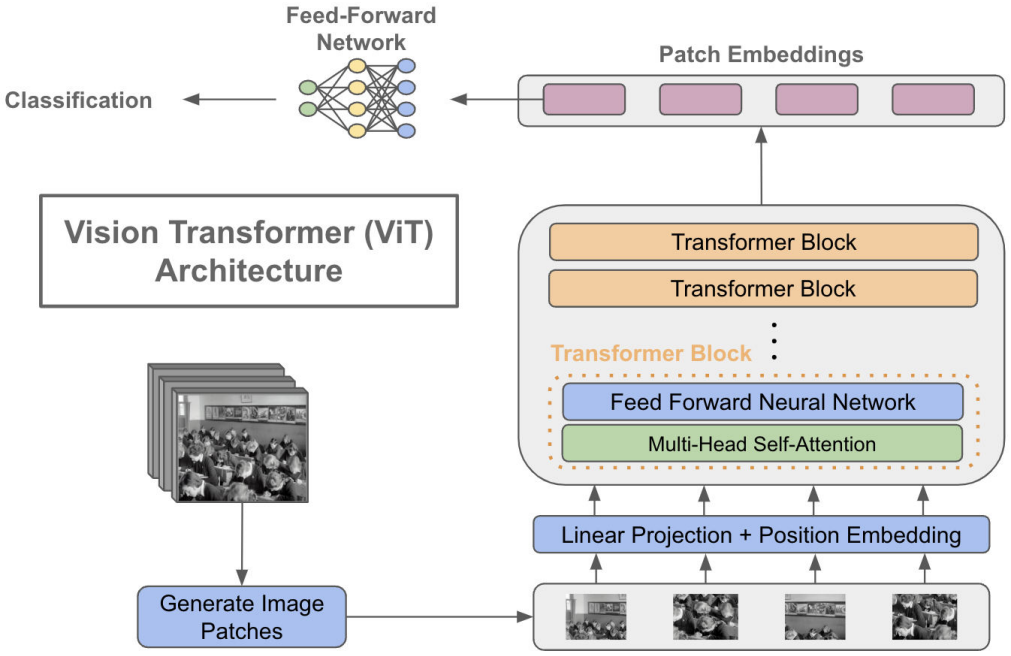


Summary: global context modeling

- Two ways of reaching global context with few params

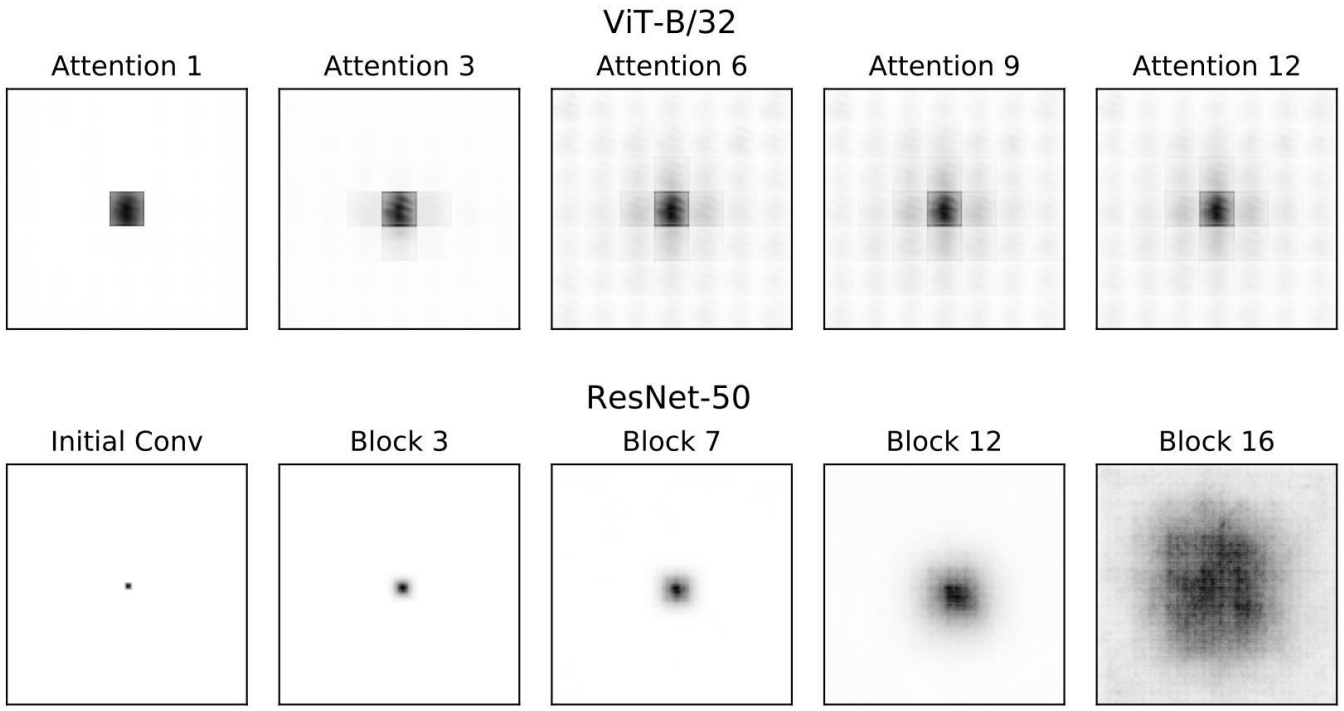
1. CNN: look at local patches,use multiscale for larger context
2. Transformer: use self-sttention to encode the relevant context





Visualization of the receptive fields

- Transformer receptive field is global from early on
- CNN receptive field grows gradually

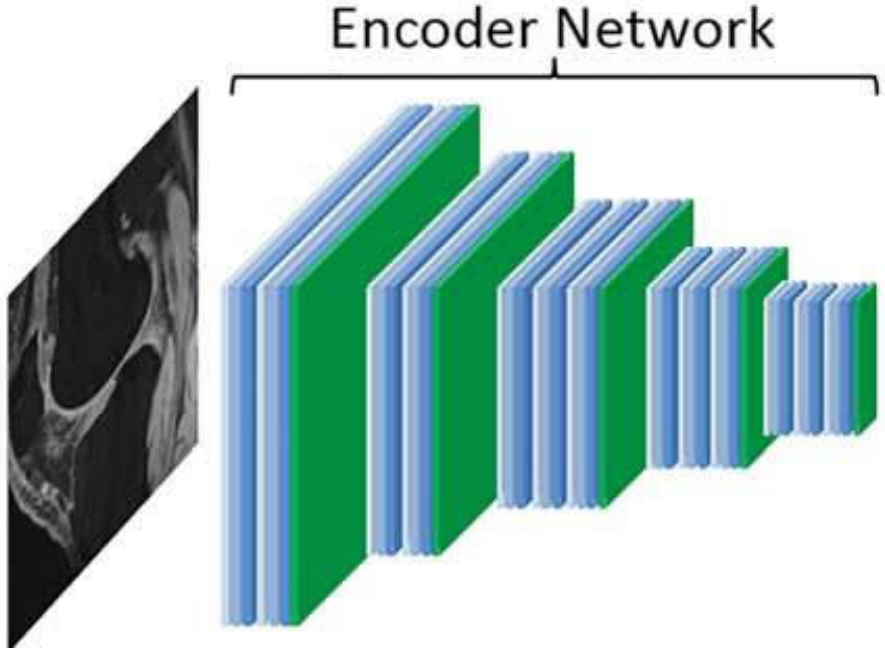


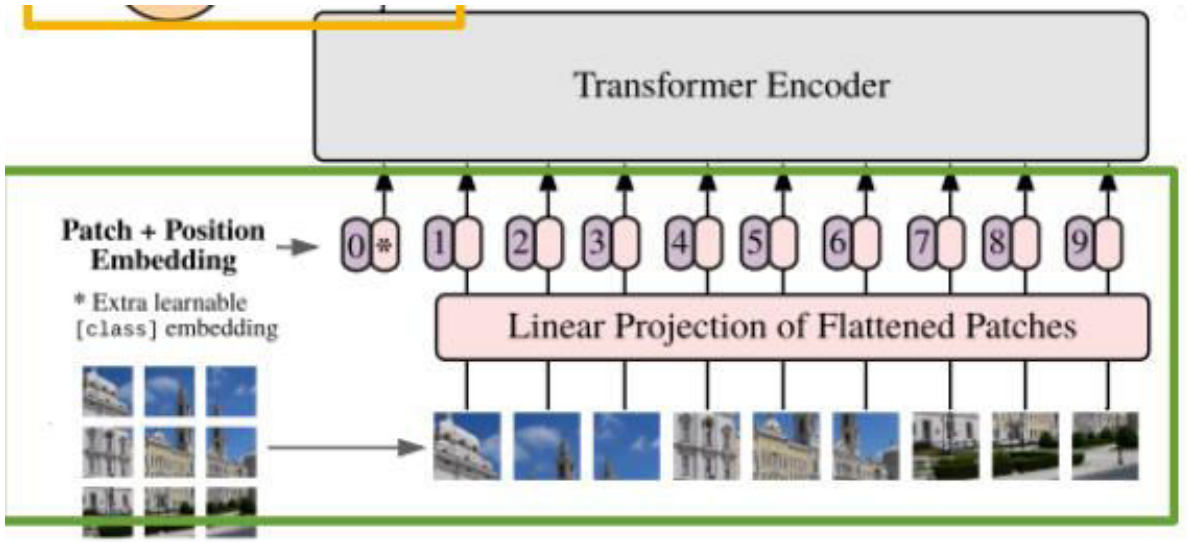
Inductive Bias vs. Expressivity

- Transformers: very expressive
- CNNs: special case of transformers
- Too much expressivity: overfitting
- The right kind of expressivity: **positive inductive bias**
- CNNs: better on small image datasets, transformer on larger ones

Neural network patterns - encoder

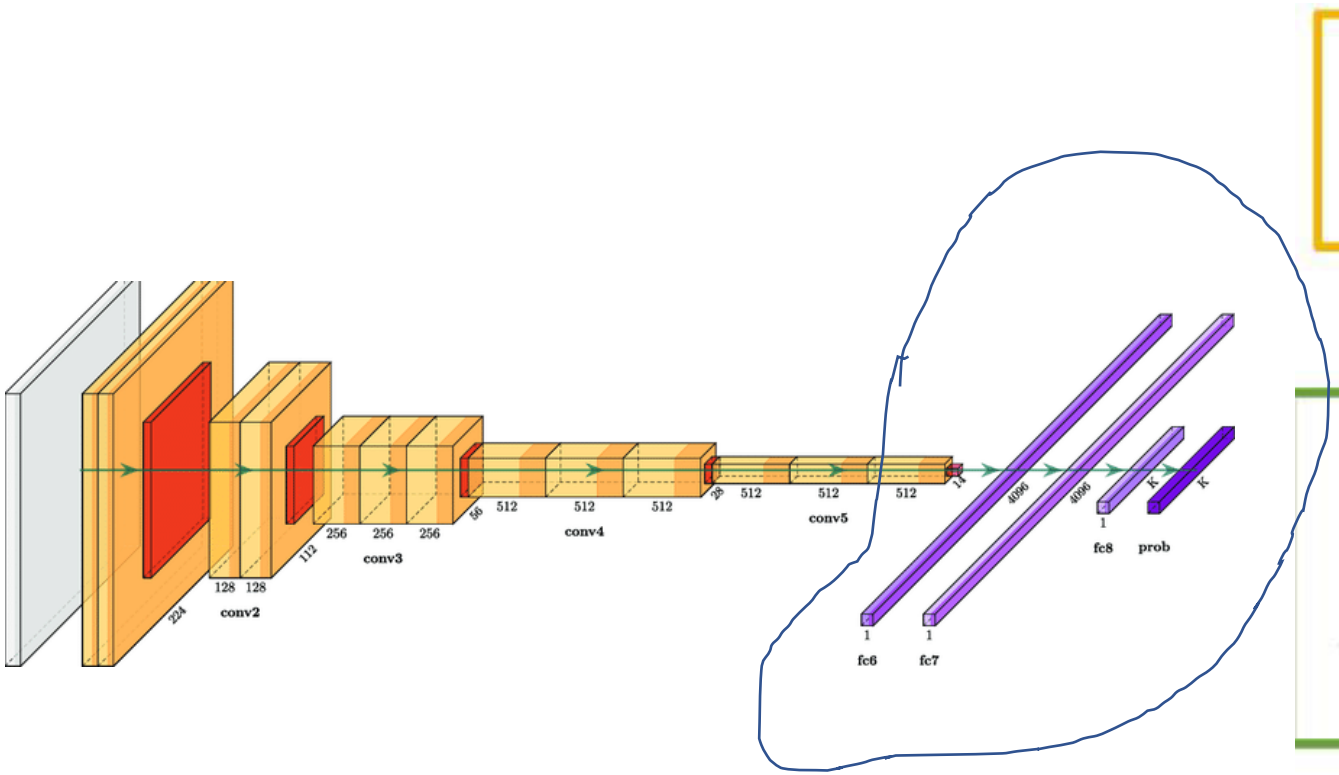
- Map raw representation to semantic deep representation
- In CNN typically also downscaling

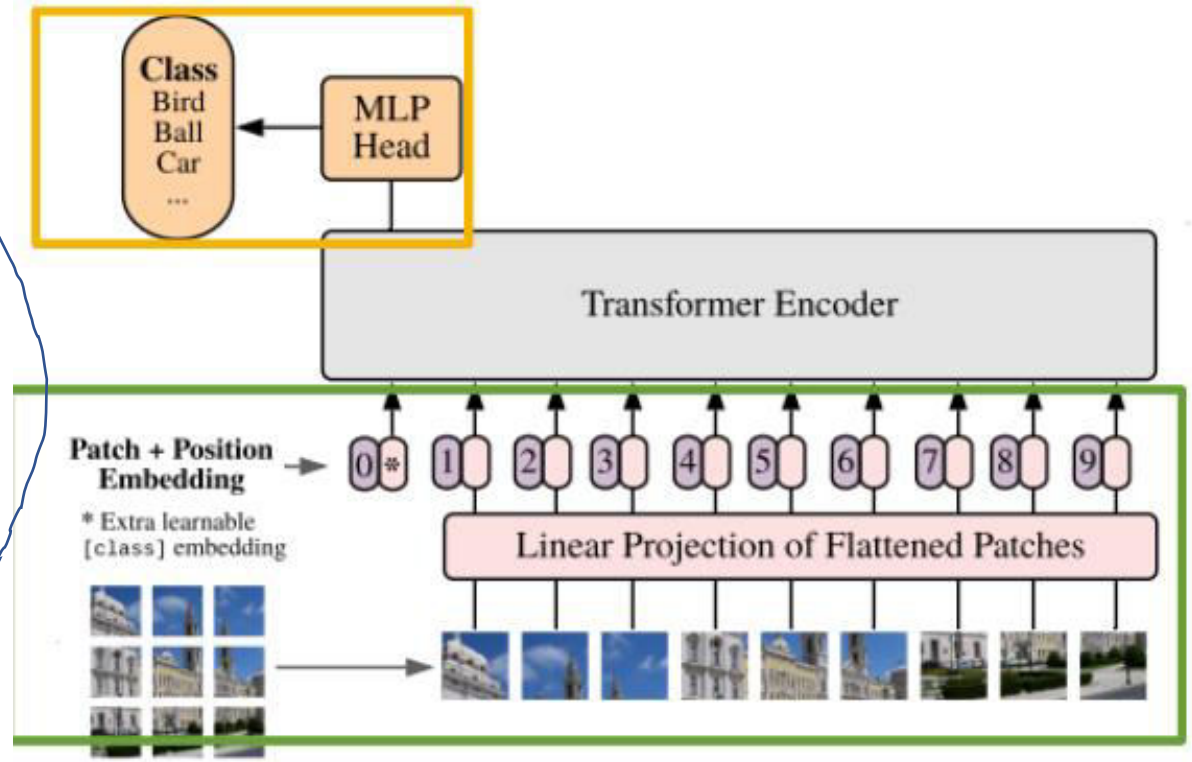




Neural network patterns - classifier

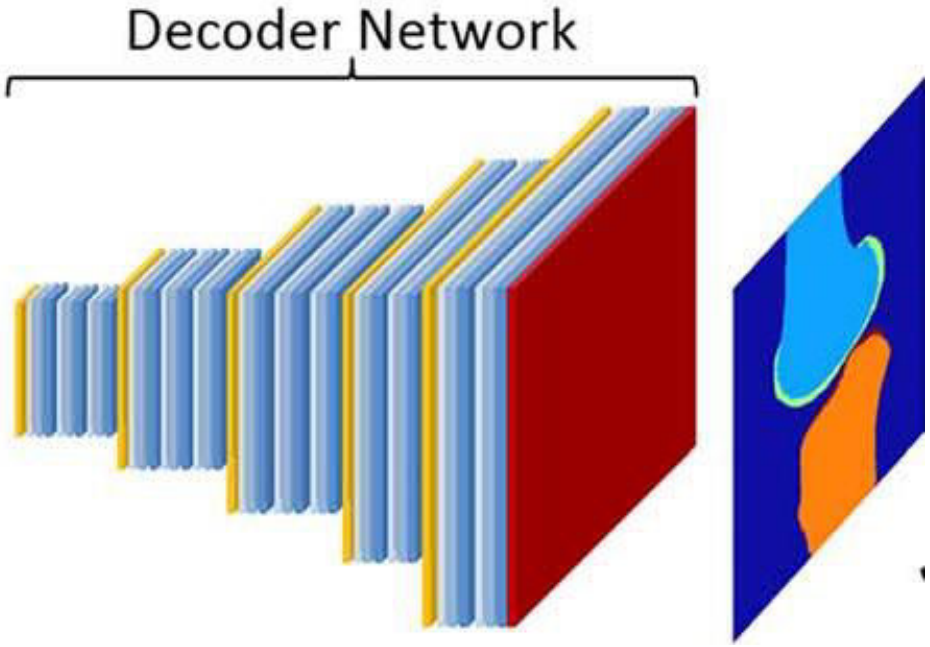
- One or more FC layers mapping deep representation to class

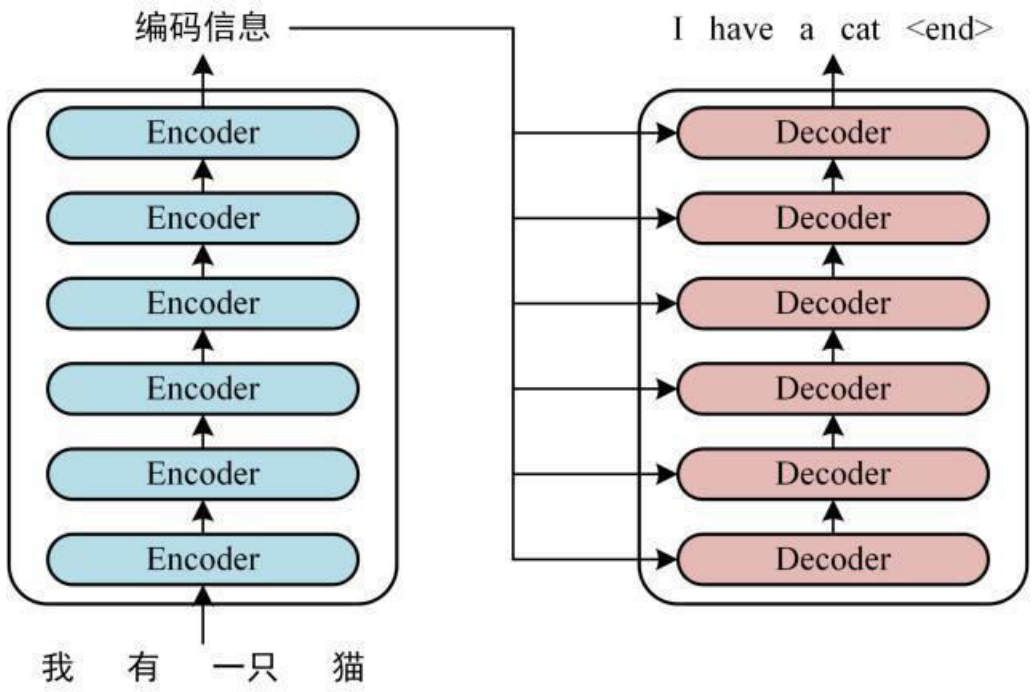




Neural network patterns - decoder

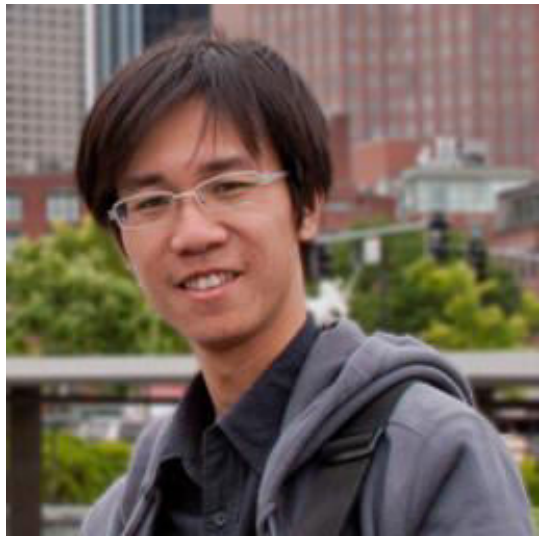
- Map deep representation to output data
- In CNN there is typically also upscaling





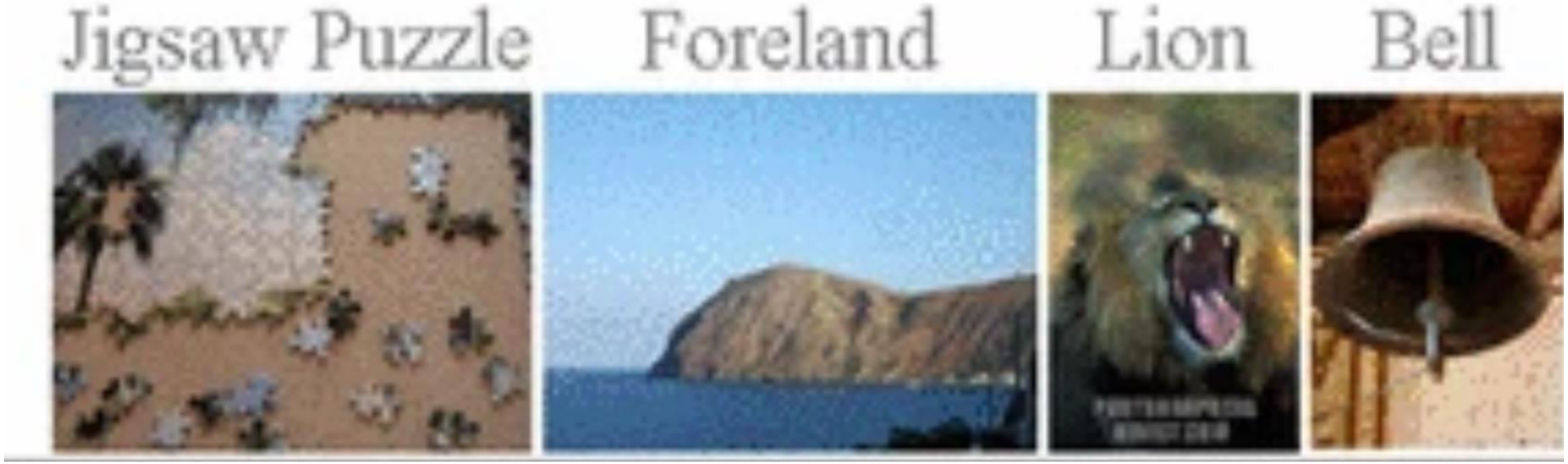
Choosing Neural Architectures in Practice

- Don’t invent new architectures unless you have to
- Much better to borrow architectures from similar problems
- There are very few architecture that passed the test of time
- Probably more than 10000 were proposed, so chances aren’t good



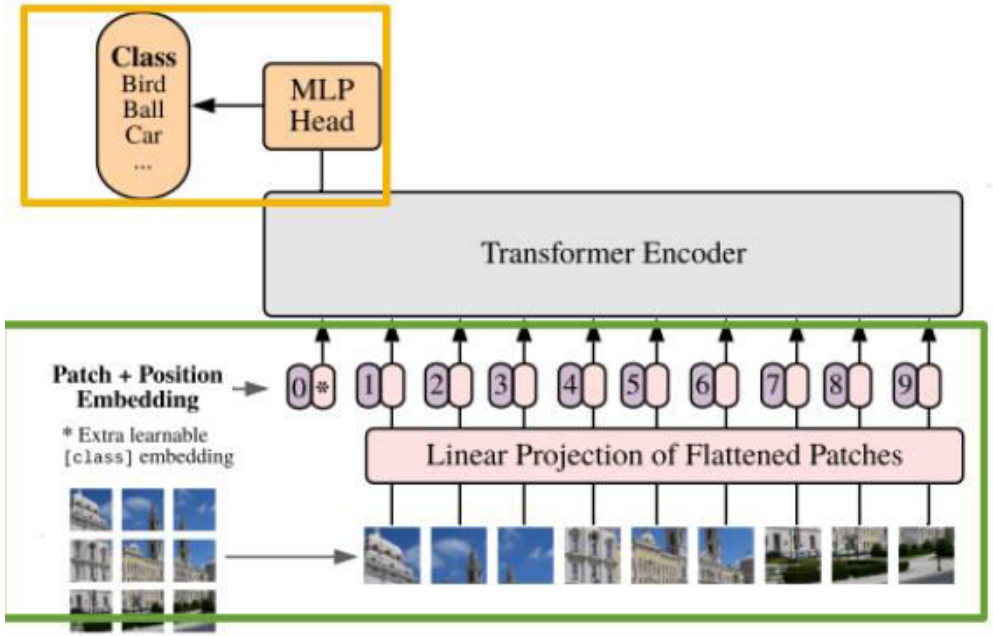
Example: Supervised Image Classication

- A transformer for image classification – for large datasets
- Input: ImageNet dataset (1M images), label 1 of 1000 object classes



Challenges: Using Transformers for Images

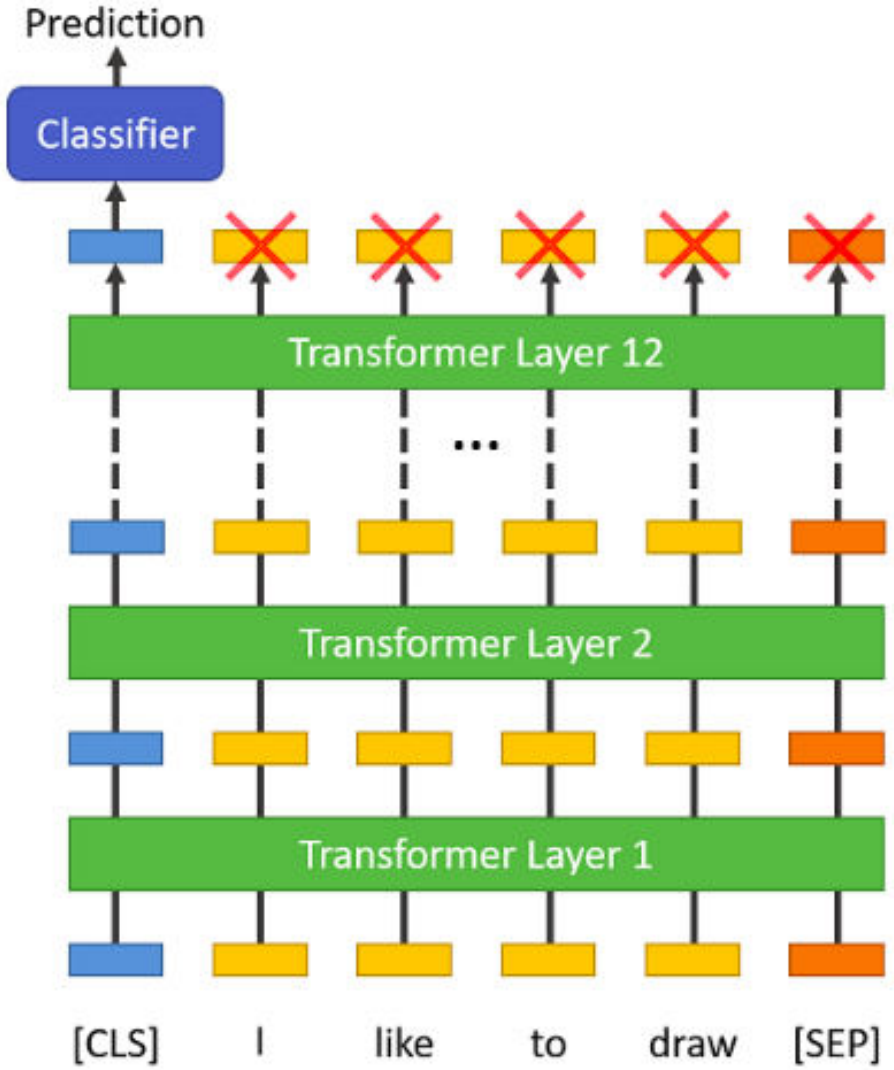
- Input to transformer should be no more than 1000 tokens
  - Solution: split image into 16X16 tokens (each is a 14X14 image patch)
- Output is a set of token activations – how to combine classifier?
  - Solutions: i) average token embedding ii) add special solution token



Example: Supervised Text Classication

- A transformer for image classification – for large datasets
- Input: sentence dataset, label 1 of 10 sentiments
- Tokenization: either every word is a token, or sub-word units

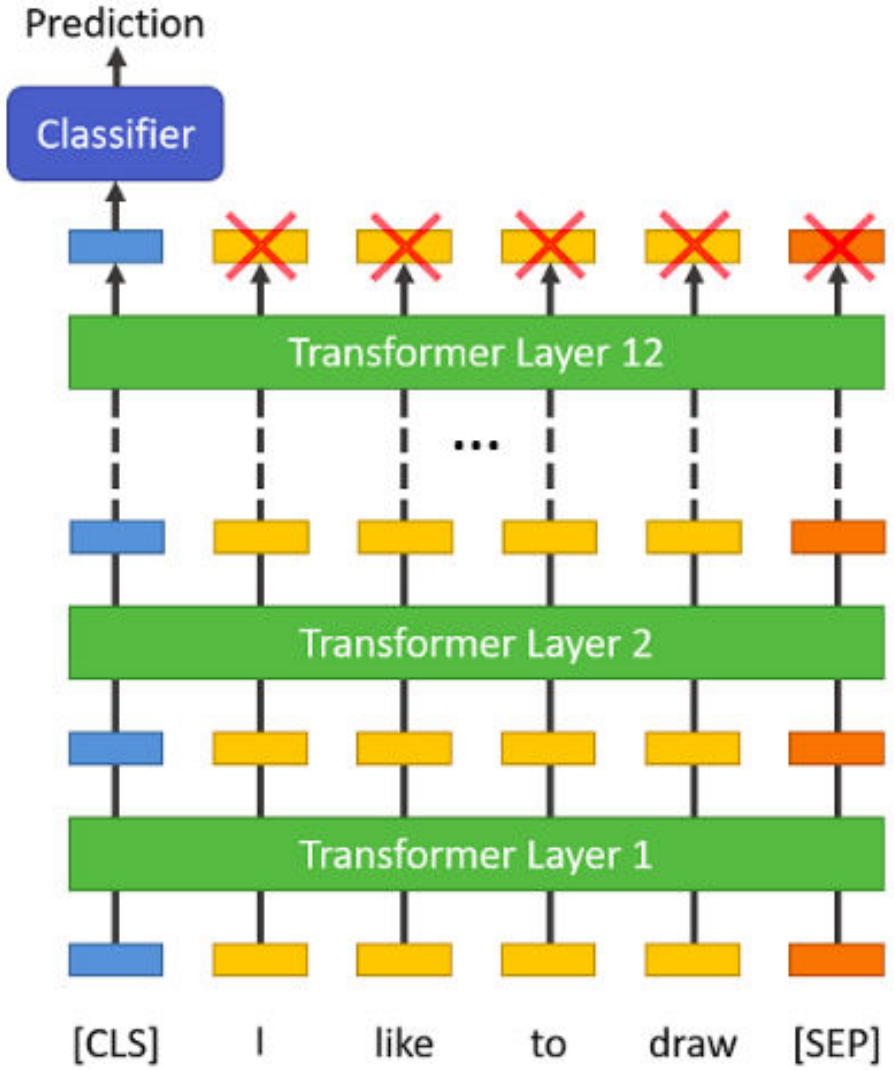
$$arg \min_{W_1,b_1,W_2,b_2,W_c,b_c} \sum_i \ell(c(f(x_i)), y_i)$$



Example: Supervised Speech Classication

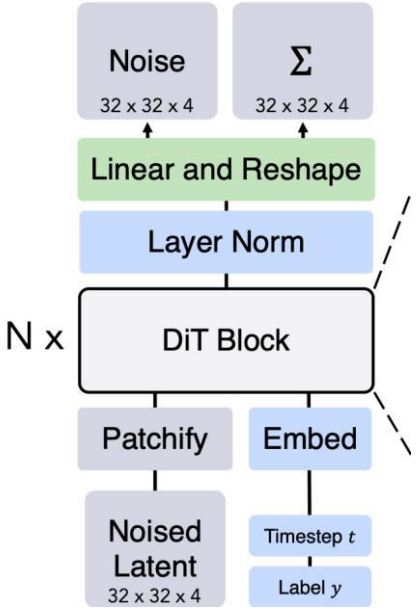
- Input: speech command dataset, label 1 of 10 commands
- Tokenization: extract spectrogram features from every 100 ms of audio

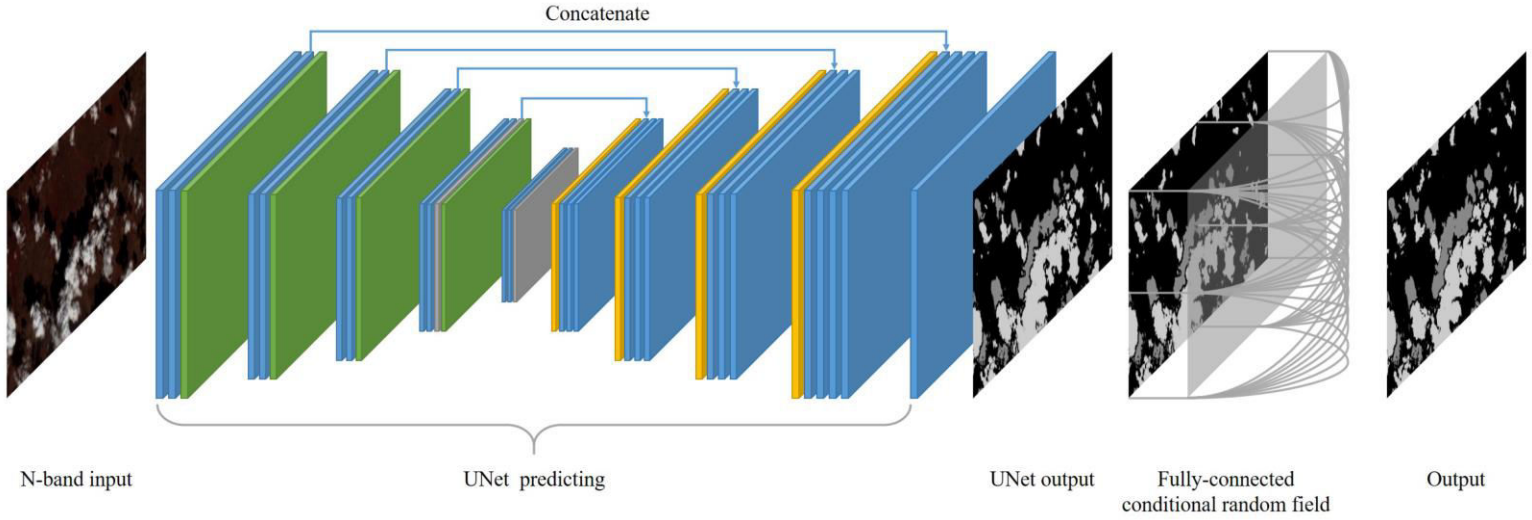
$$arg \min_{W_1,b_1,W_2,b_2,W_c,b_c} \sum_i \ell(c(f(x_i)), y_i)$$



Example: Image-to-Image

- Challenge: architecture when input and output are images
- Two ideas:
  - Tokenization – followed by transformer
  - Unet – encoder-decoder with skip connections





Conclusion

- Deep architectures are converging
- Global context without too many parameters
- Lots of effort required for hyperparameter tuning
- New tasks require clever tokenization