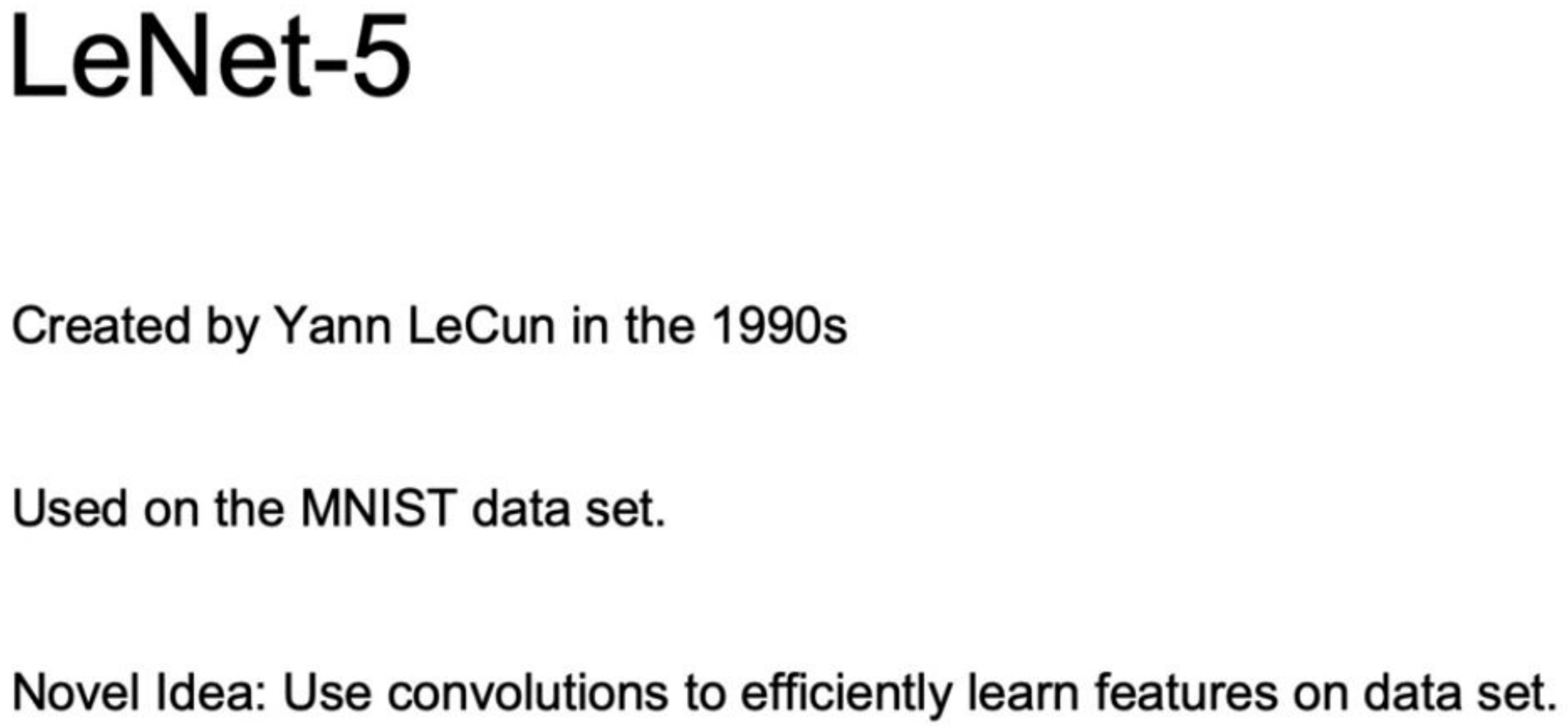


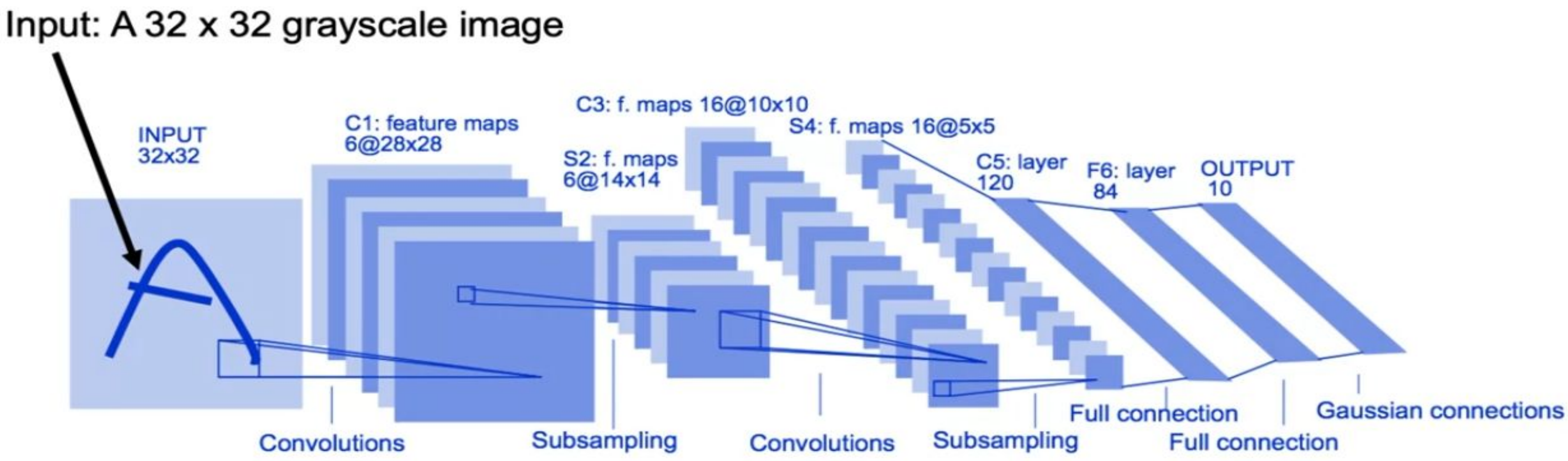
LeNet-5

Created by Yann LeCun in the 1990s

Used on the MNIST data set.

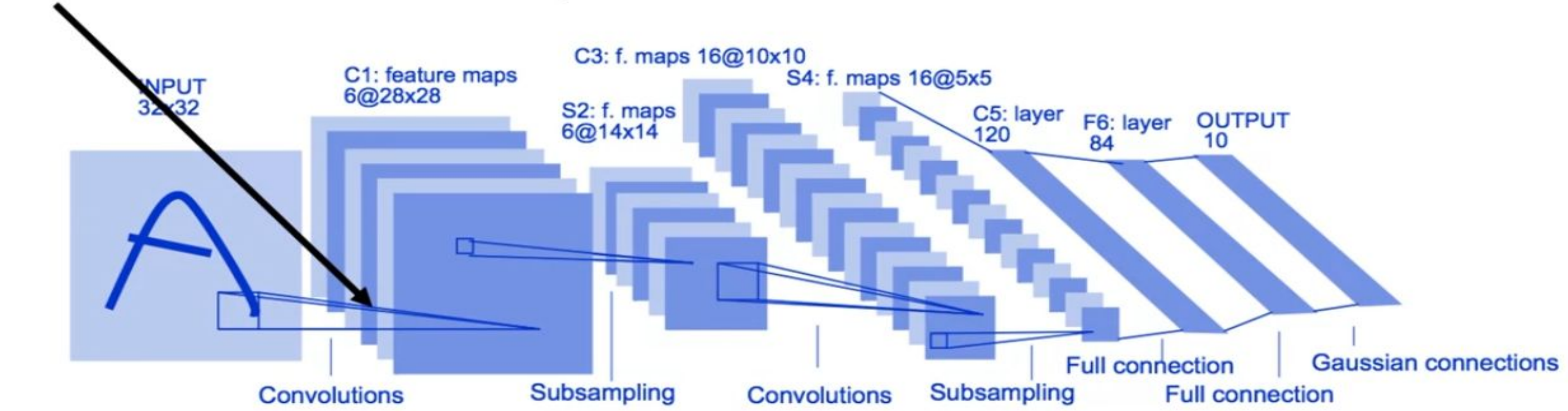
Novel Idea: Use convolutions to efficiently learn features on data set.

LeNet – Structure Diagram



LeNet – Structure Diagram

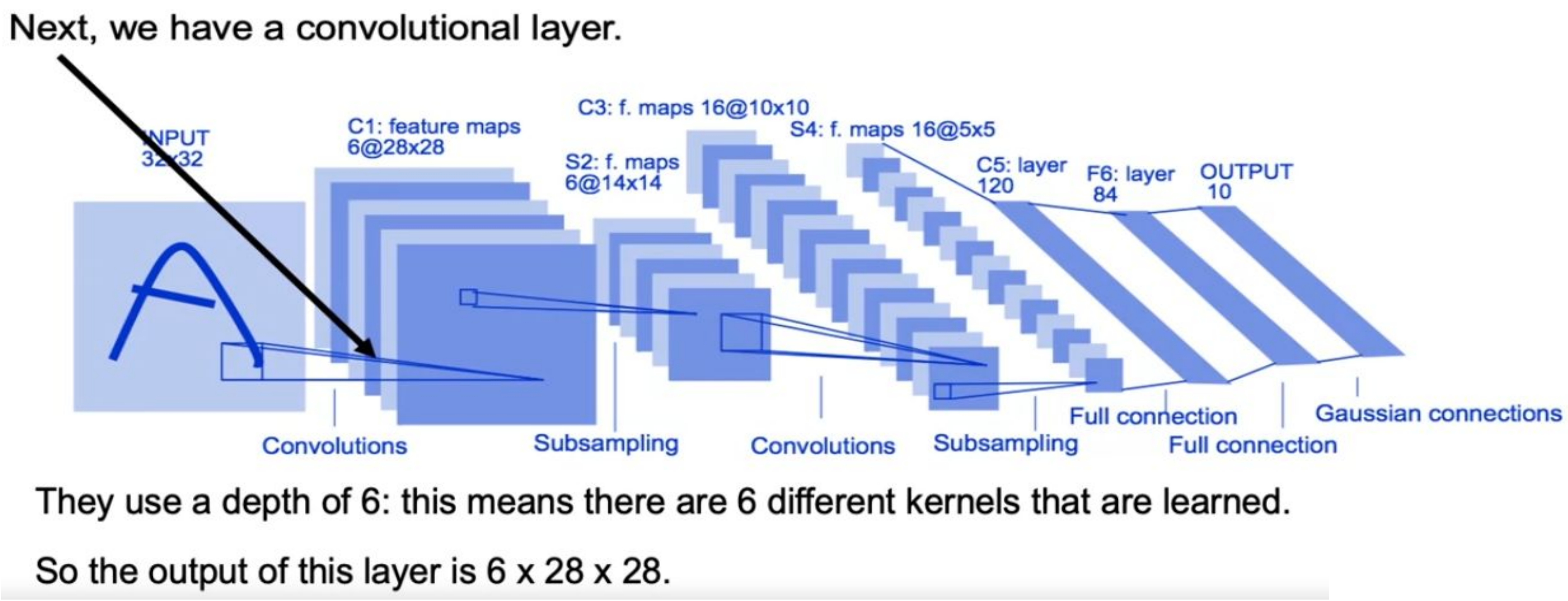
Next, we have a convolutional layer.



This is a 5x5 convolutional layer with stride 1.

This means the resulting output has dimension 28 x 28. (Why?)

LeNet – Structure Diagram



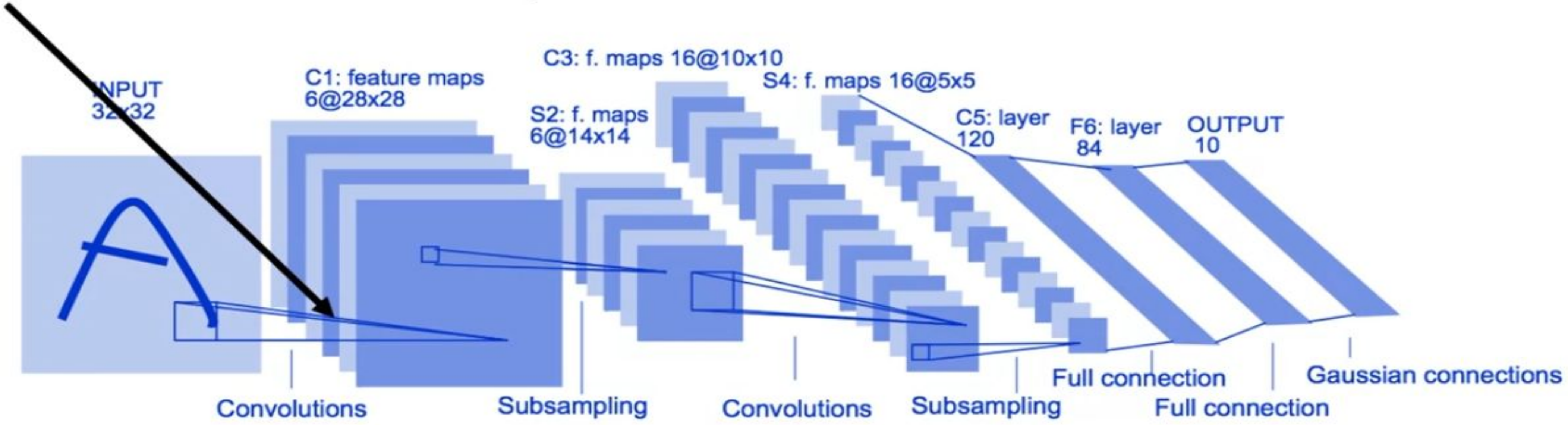
Next, we have a convolutional layer.

They use a depth of 6: this means there are 6 different kernels that are learned.

So the output of this layer is 6 x 28 x 28.

LeNet – Structure Diagram

Next, we have a convolutional layer.

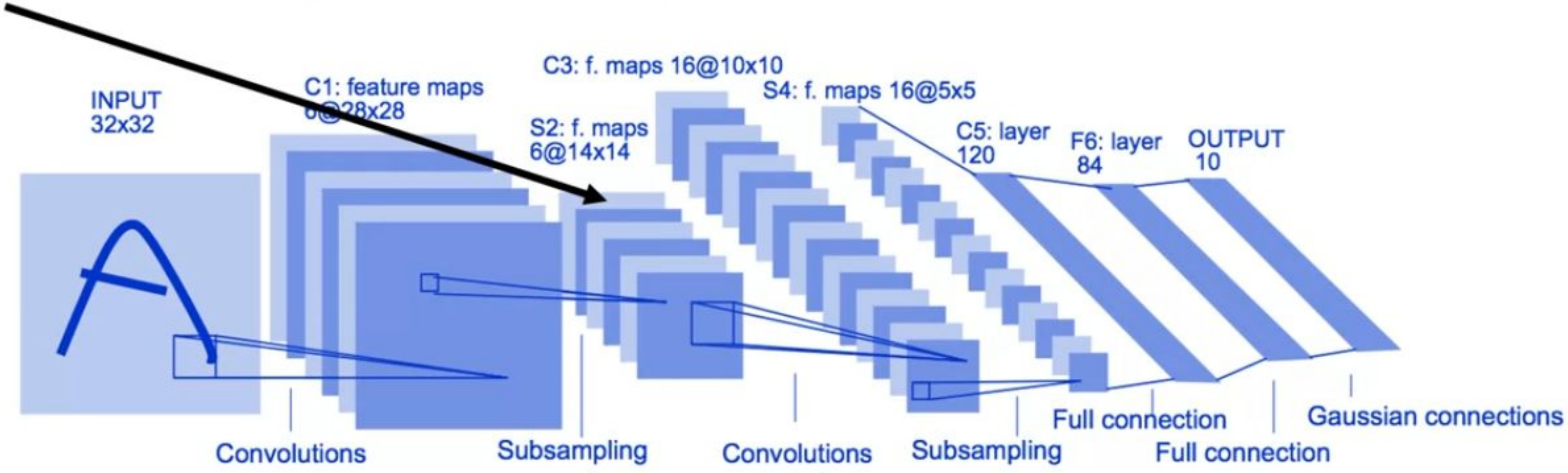


What is the total number of weights in this layer?

Answer: Each kernel has 5 x 5 = 25 weights (plus a bias term, so actually 26 weights).
So total weights = 6 x 26 = 156.

LeNet – Structure Diagram

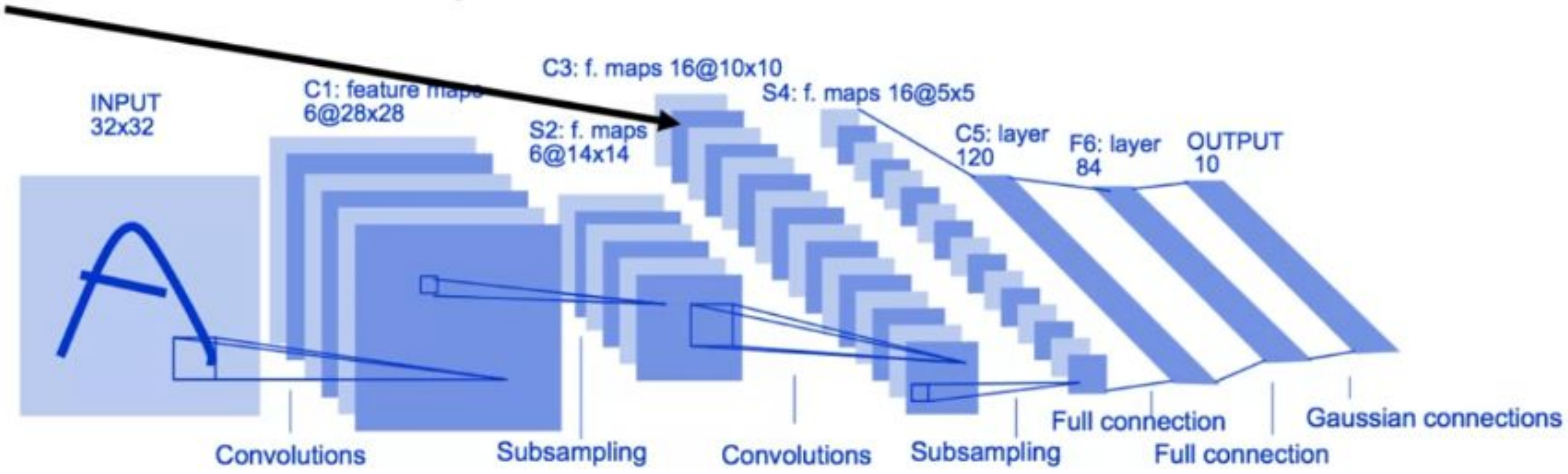
Next is a 2x2 pooling layer (with stride 2).



No weights! (pooling layers have no weights to be learned – it is a fixed operation)

LeNet – Structure Diagram

Another 5x5 convolutional layer with stride 1.

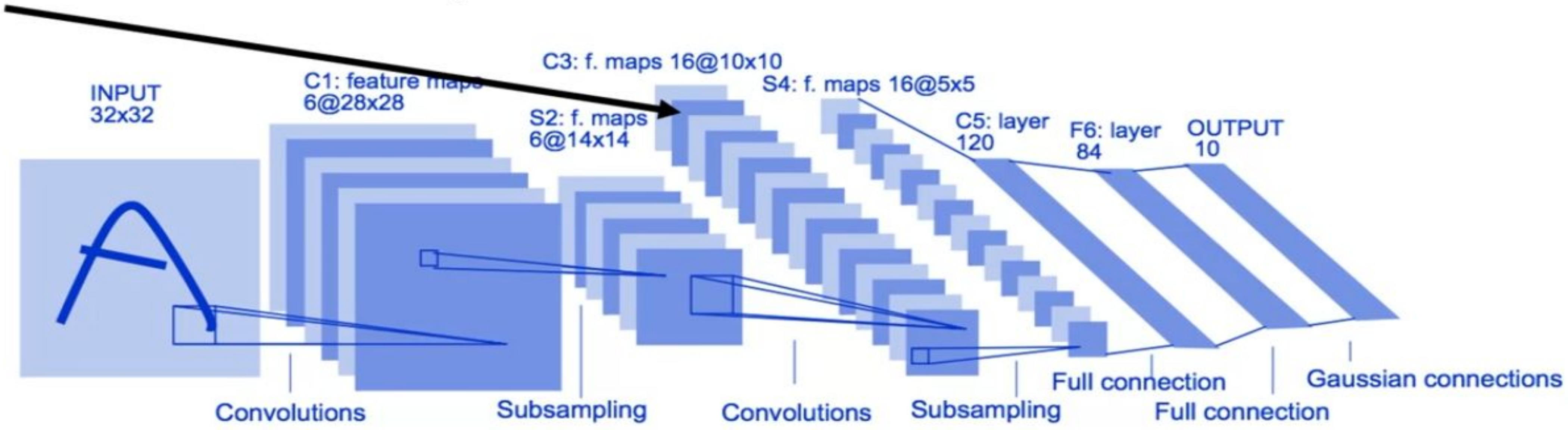


The kernels “take in” the full depth of the previous layer.

So each 5×5 kernel now “looks at” 6×5×5 pixels.

LeNet – Structure Diagram

Another 5x5 convolutional layer with stride 1.

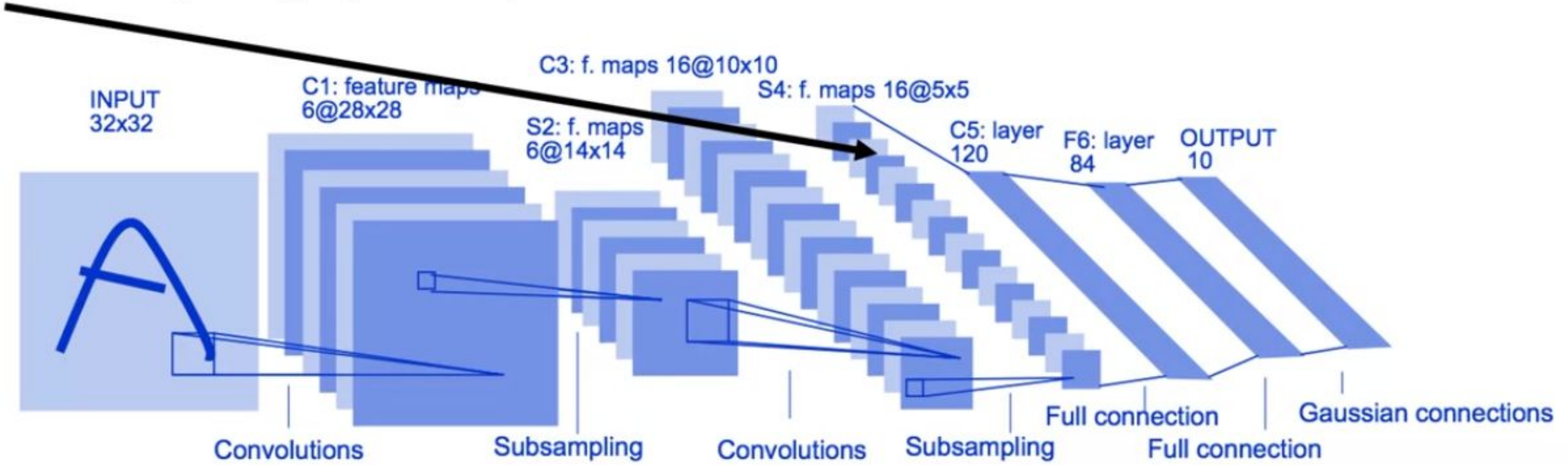


Each kernel has 6×5×5 = 150 weights + bias term = 151.

So, total weights for this layer = 16*151 = 2416.

LeNet – Structure Diagram

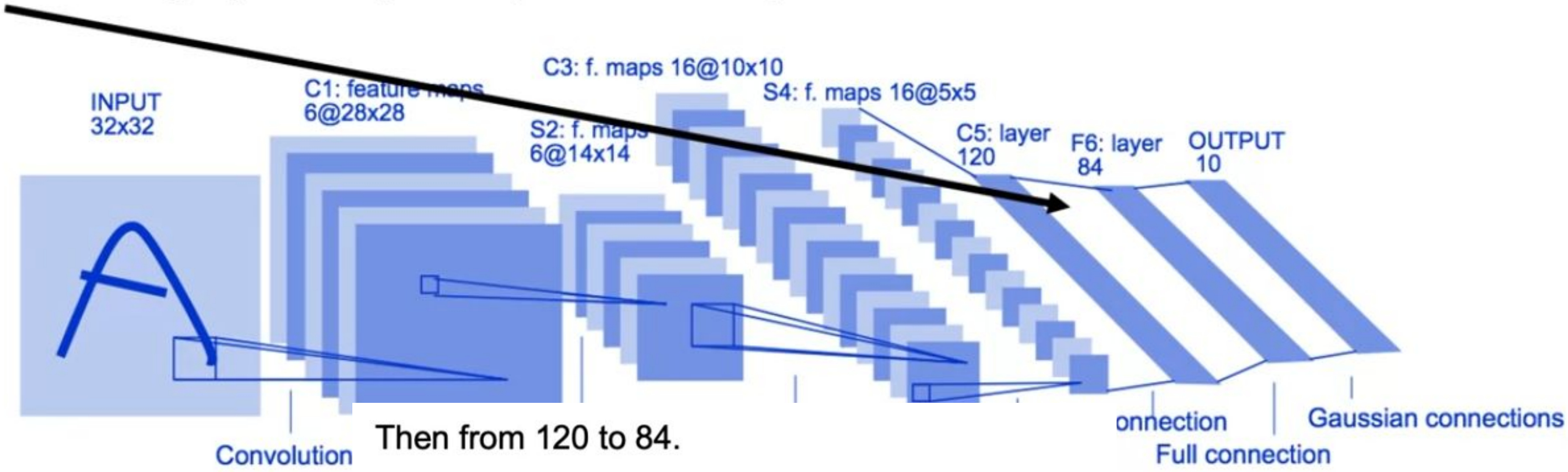
Another 2x2 pooling layer. Output is 16 x 5 x 5.



We “flatten” this to a length 400 vector (not shown).

LeNet – Structure Diagram

The following layers are just fully connected layers! From 400 to 120.



Then from 120 to 84.

Then from 84 to 10.

And a softmax output of size 10 for the 10 digits

LeNet-5

How many total weights in the network?

Conv1: $1 \times 6 \times 5 \times 5 + 6 \quad = \quad 156$

Conv3: $6 \times 16 \times 5 \times 5 + 16 \quad = \quad 2416$

FC1: $400 \times 120 + 120 \quad = 48120$

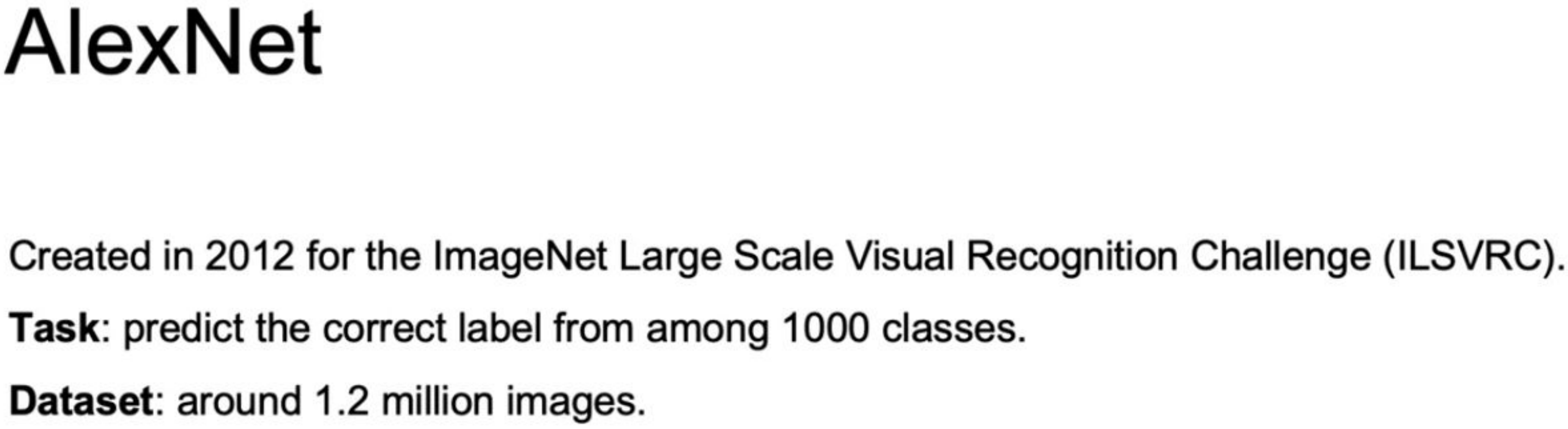
FC2: $120 \times 84 + 84 \quad = 10164$

FC3: $84 \times 10 + 10 \quad = \quad 850$

Total: $= 61706$

Less than a single FC layer with [1200×1200] weights!

Note that Convolutional Layers have relatively few weights.

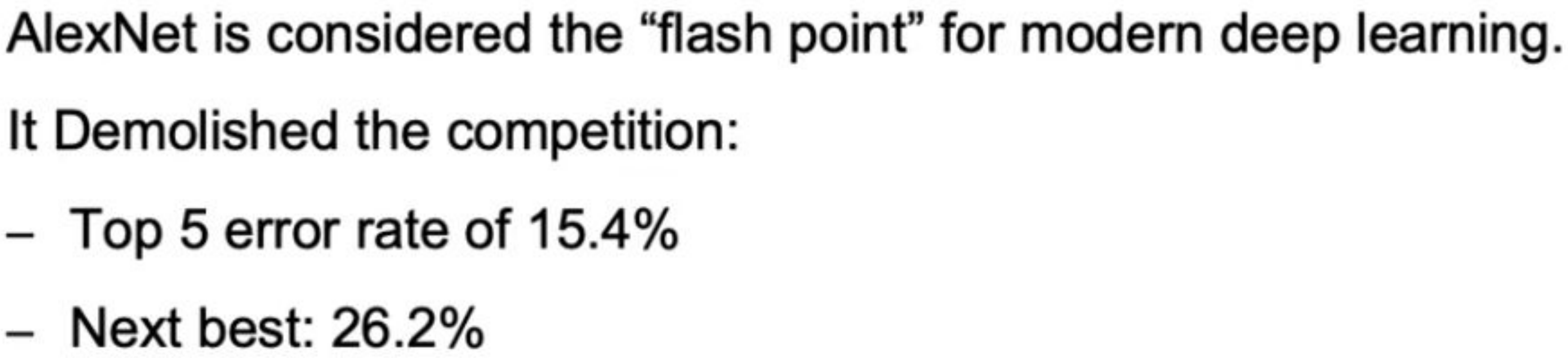


AlexNet

Created in 2012 for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

Task: predict the correct label from among 1000 classes.

Dataset: around 1.2 million images.

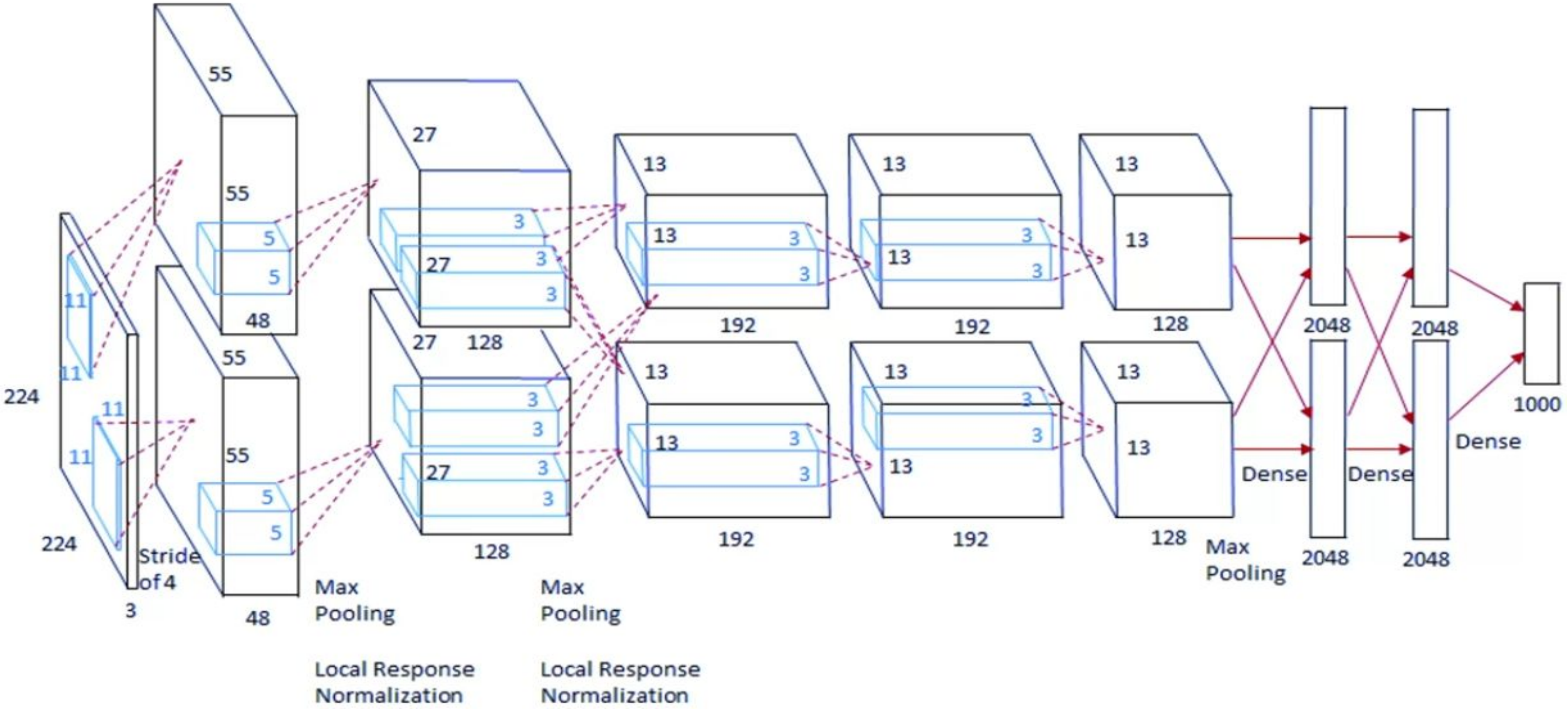


AlexNet is considered the “flash point” for modern deep learning.

It Demolished the competition:

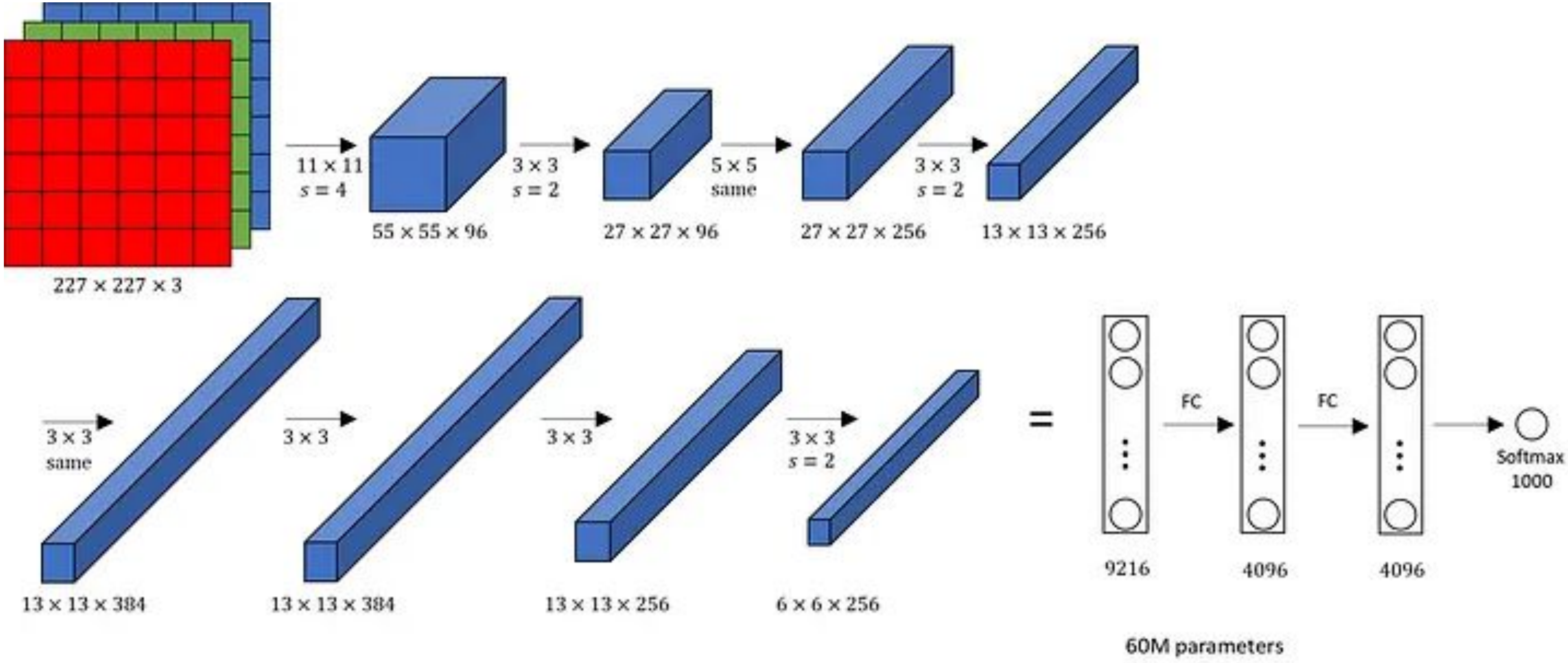
- Top 5 error rate of 15.4%
- Next best: 26.2%

AlexNet - Model Diagram

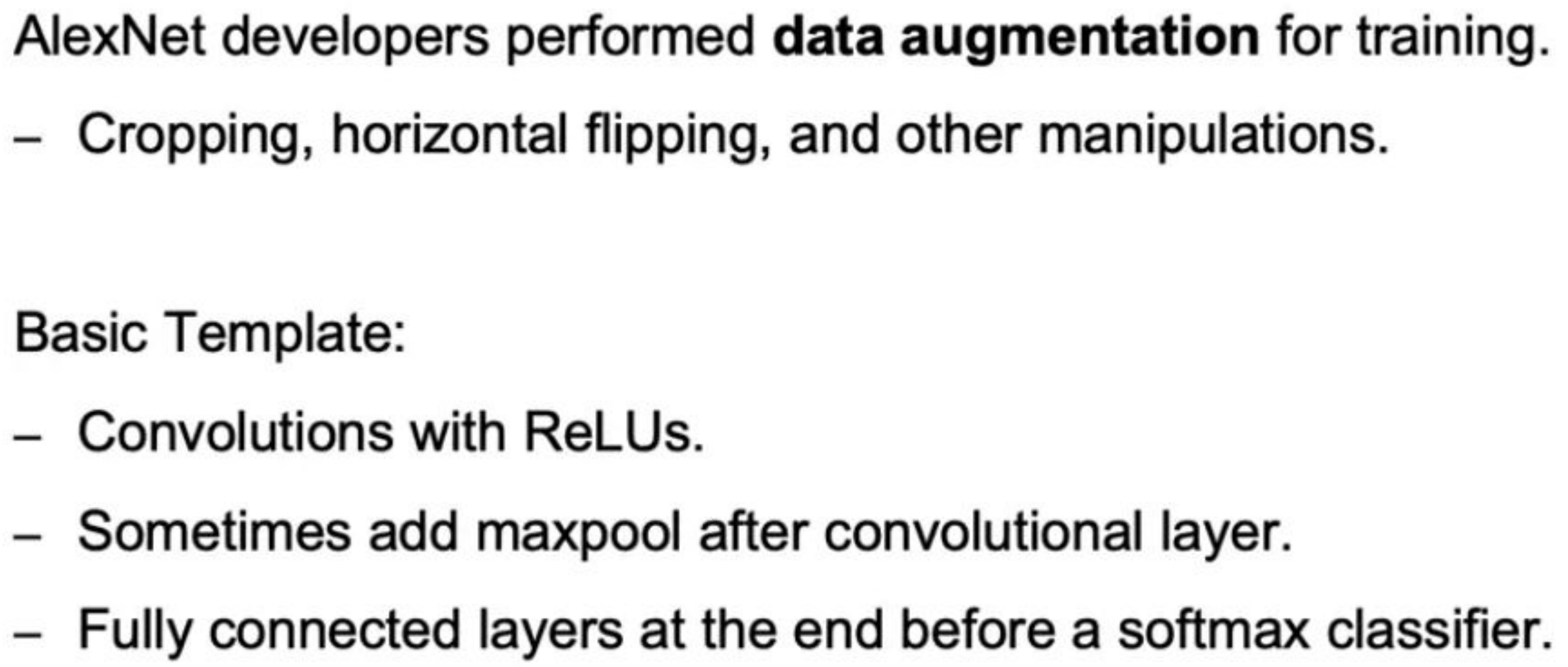


Advanced CNN Architectures

AlexNet



Advanced CNN Architectures



AlexNet developers performed **data augmentation** for training.

- Cropping, horizontal flipping, and other manipulations.

Basic Template:

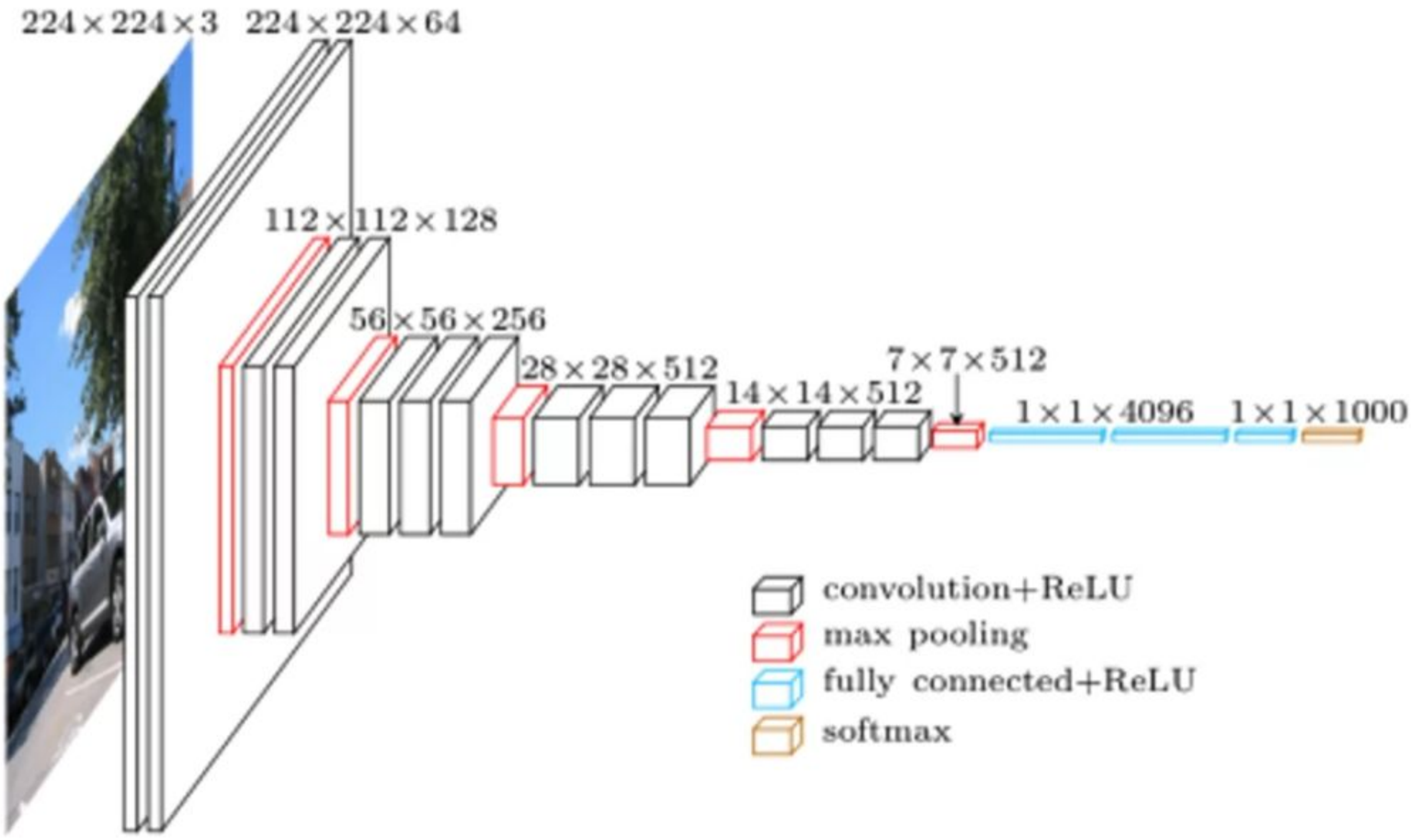
- Convolutions with ReLUs.
- Sometimes add maxpool after convolutional layer.
- Fully connected layers at the end before a softmax classifier.

VGG

Simplify Network Structure:

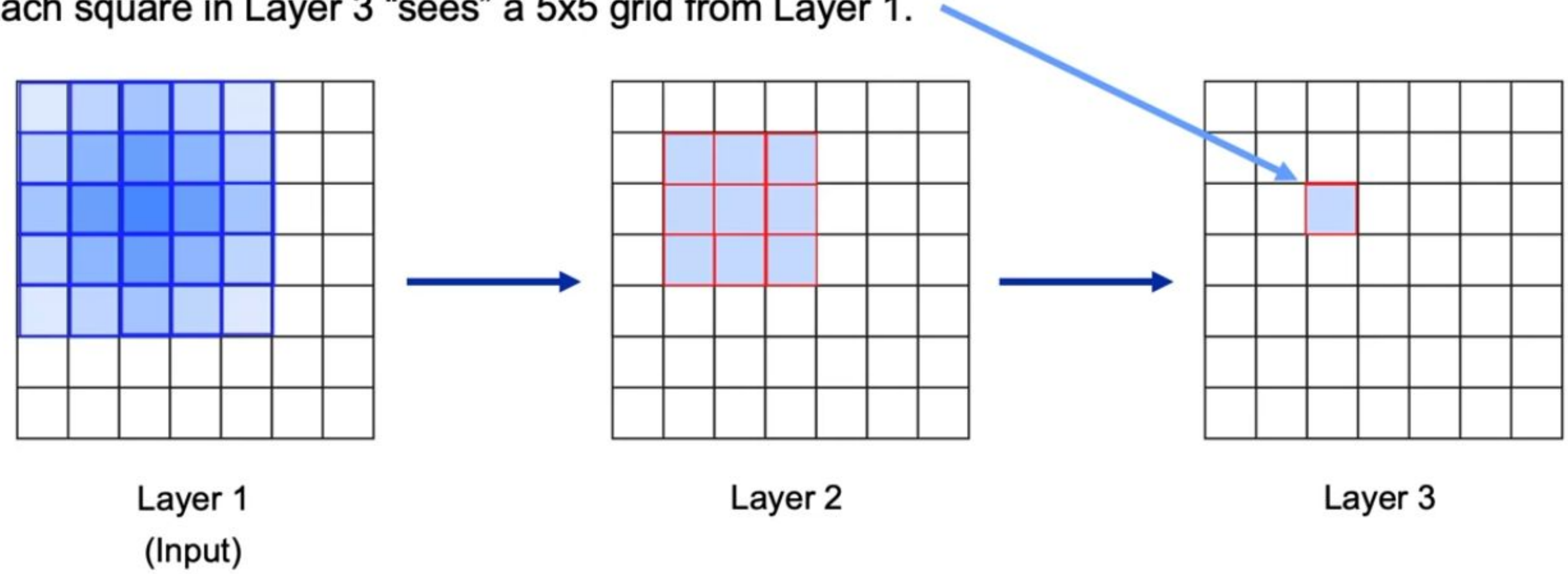
- Avoid Manual Choices of Convolution Size.
- Very Deep Network with 3x3 Convolutions.
- These “effectively” give rise to larger convolutions.

VGG16 Diagram



VGG

Each square in Layer 3 “sees” a 5x5 grid from Layer 1.



VGG

Two 3×3, stride 1 convolutions in a row → one 5×5.

Three 3×3 convolutions → one 7×7 convolution.

One 3x3 layer: $3{\times}3 = 9$

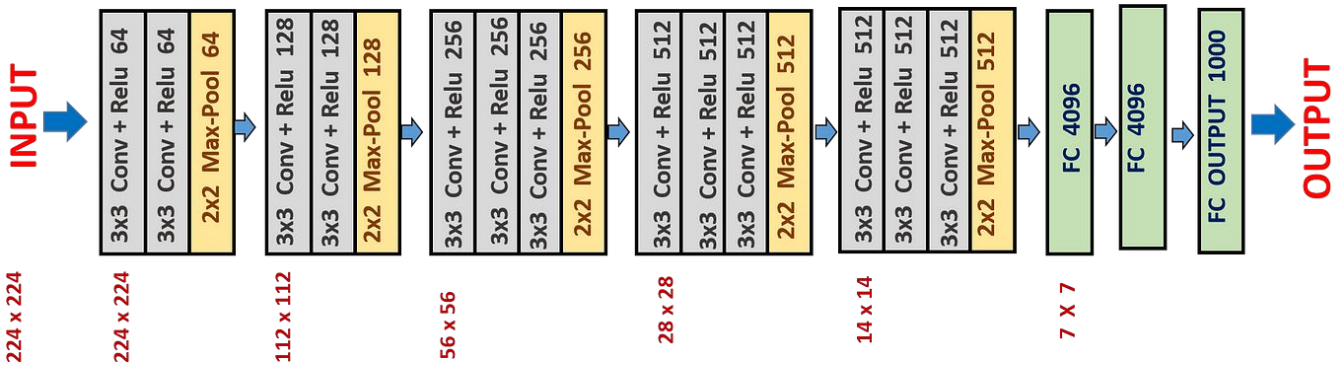
One 7x7 layer: $7{\times}7 = 49$

Three 3x3 layers: $3{\times}(9) = 27$

Benefit: fewer parameters.

$49 \rightarrow 27 \rightarrow \approx 45\%$ reduction!

VGG-16



One of the first architectures to experiment with many layers (More is better!).

– Can use multiple 3x3 convolutions to simulate larger kernels with fewer parameters.

Inception

Szegedy et al 2014.

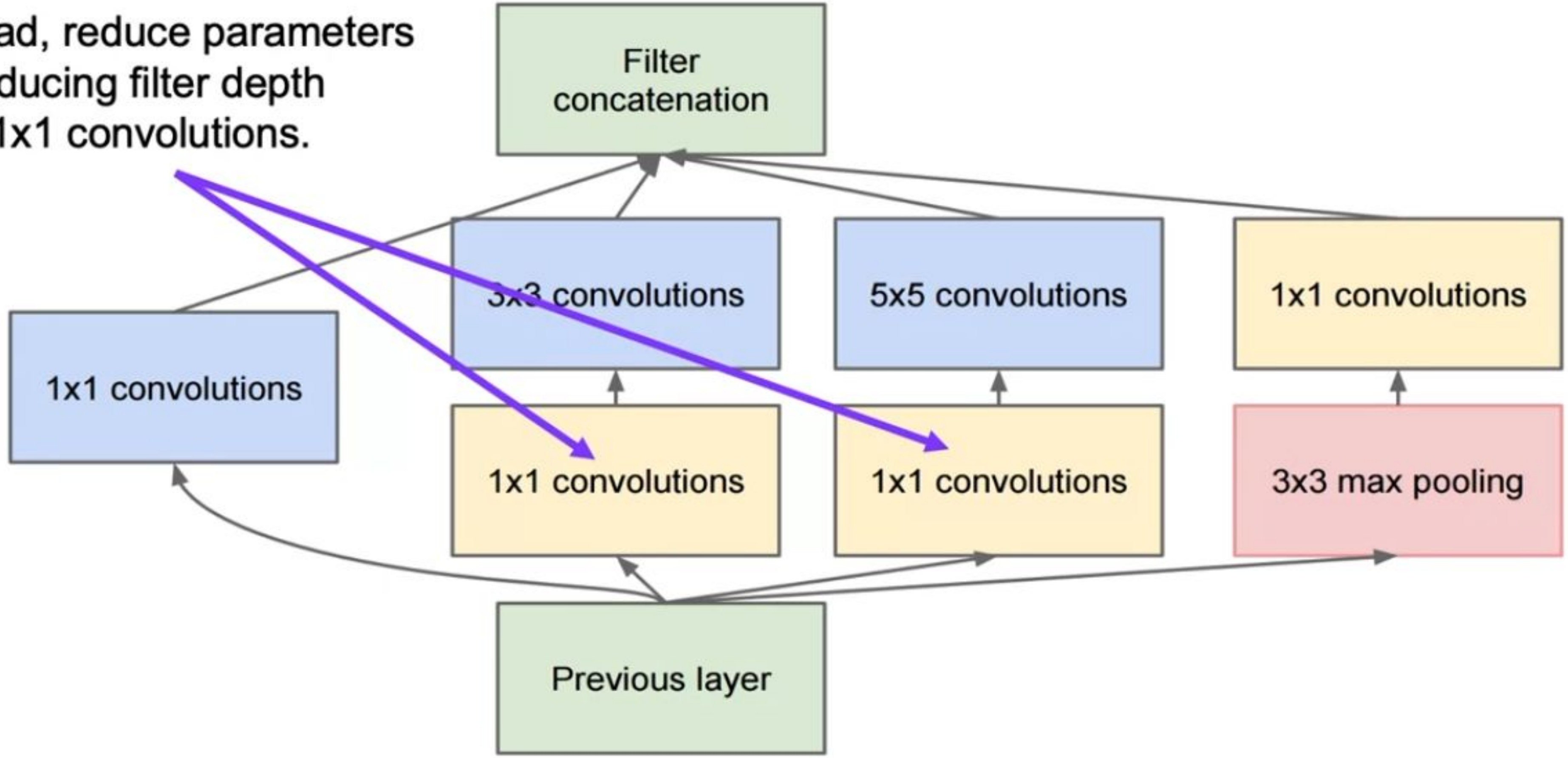
- Idea: network would want to use different receptive fields.
- Want computational efficiency.
- Also want to have sparse activations of groups of neurons.

Hebbian principle: “Fire together, wire together”.

- Solution: Turn each layer into branches of convolutions.
- Each branch handles smaller portion of workload.
- Concatenate different branches at the end.

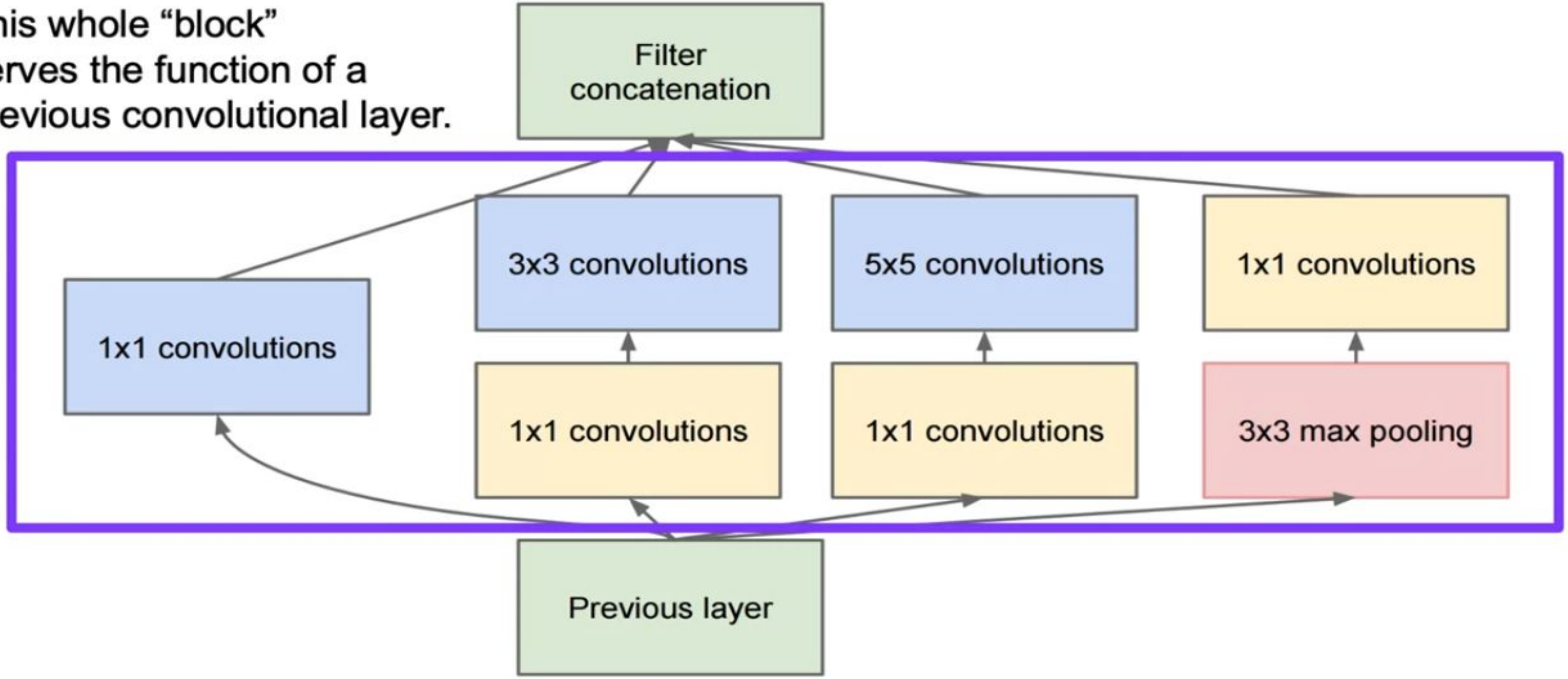
Inception

Instead, reduce parameters by reducing filter depth with 1x1 convolutions.

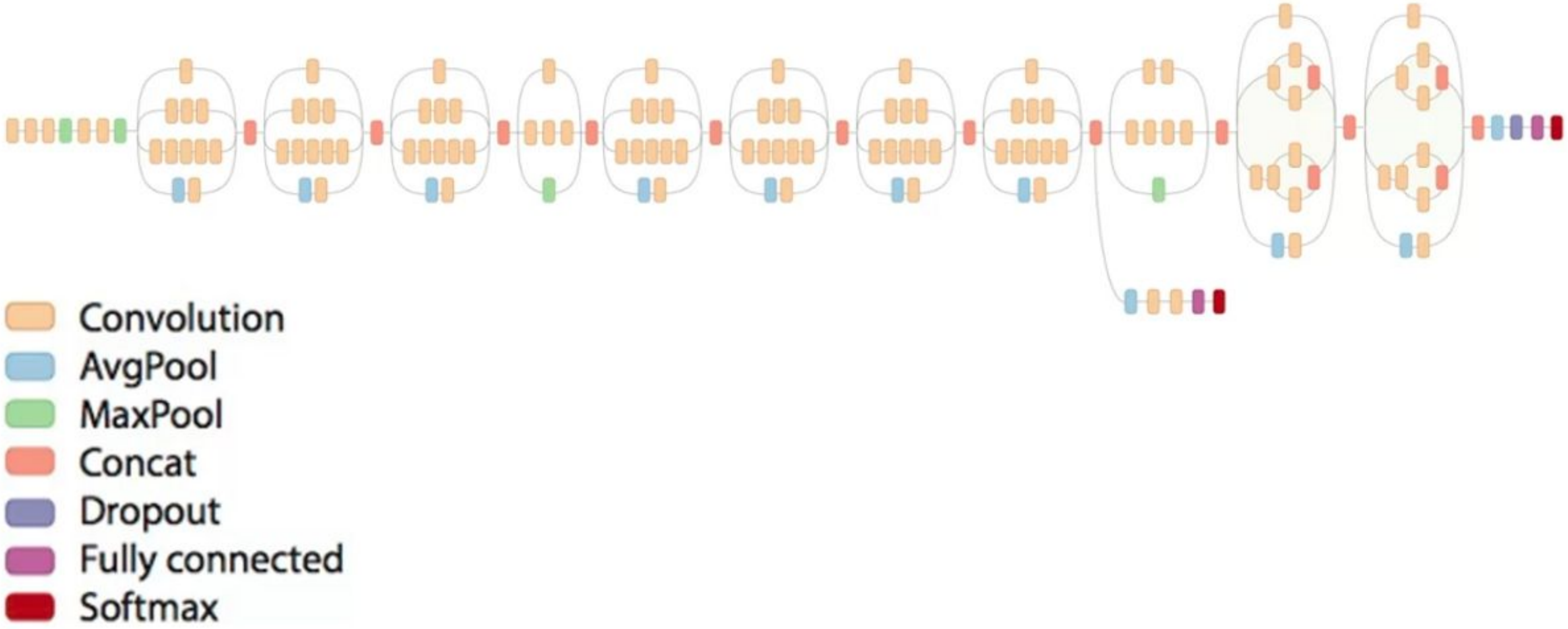


Inception

This whole “block” serves the function of a previous convolutional layer.

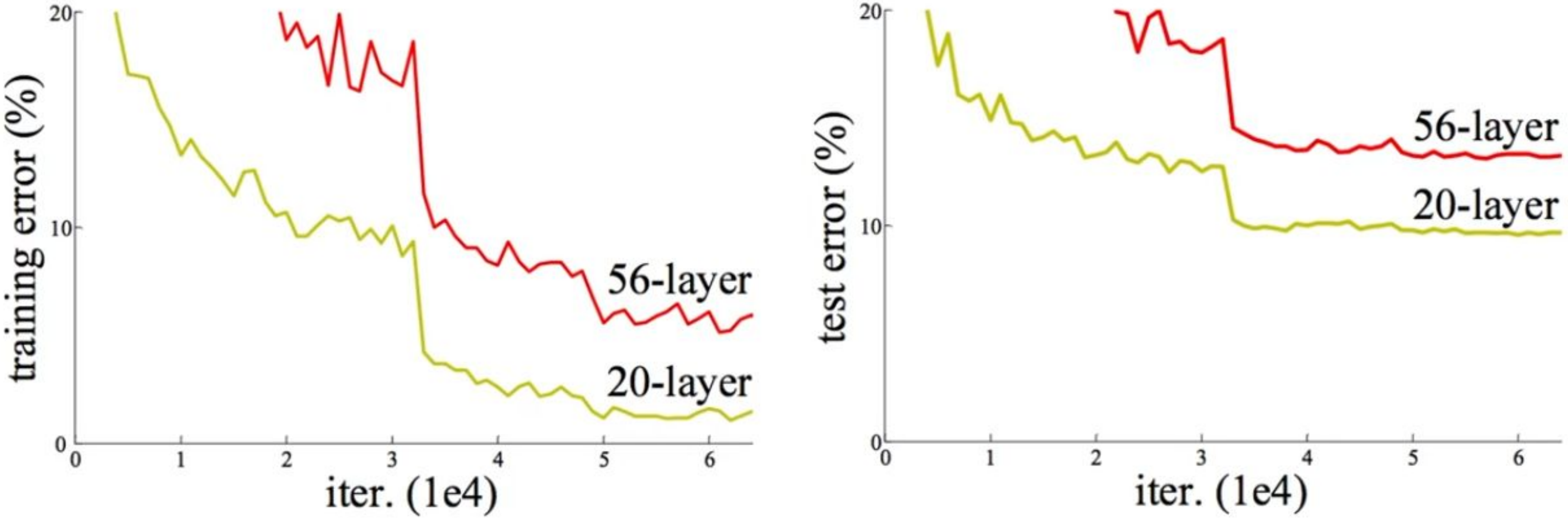


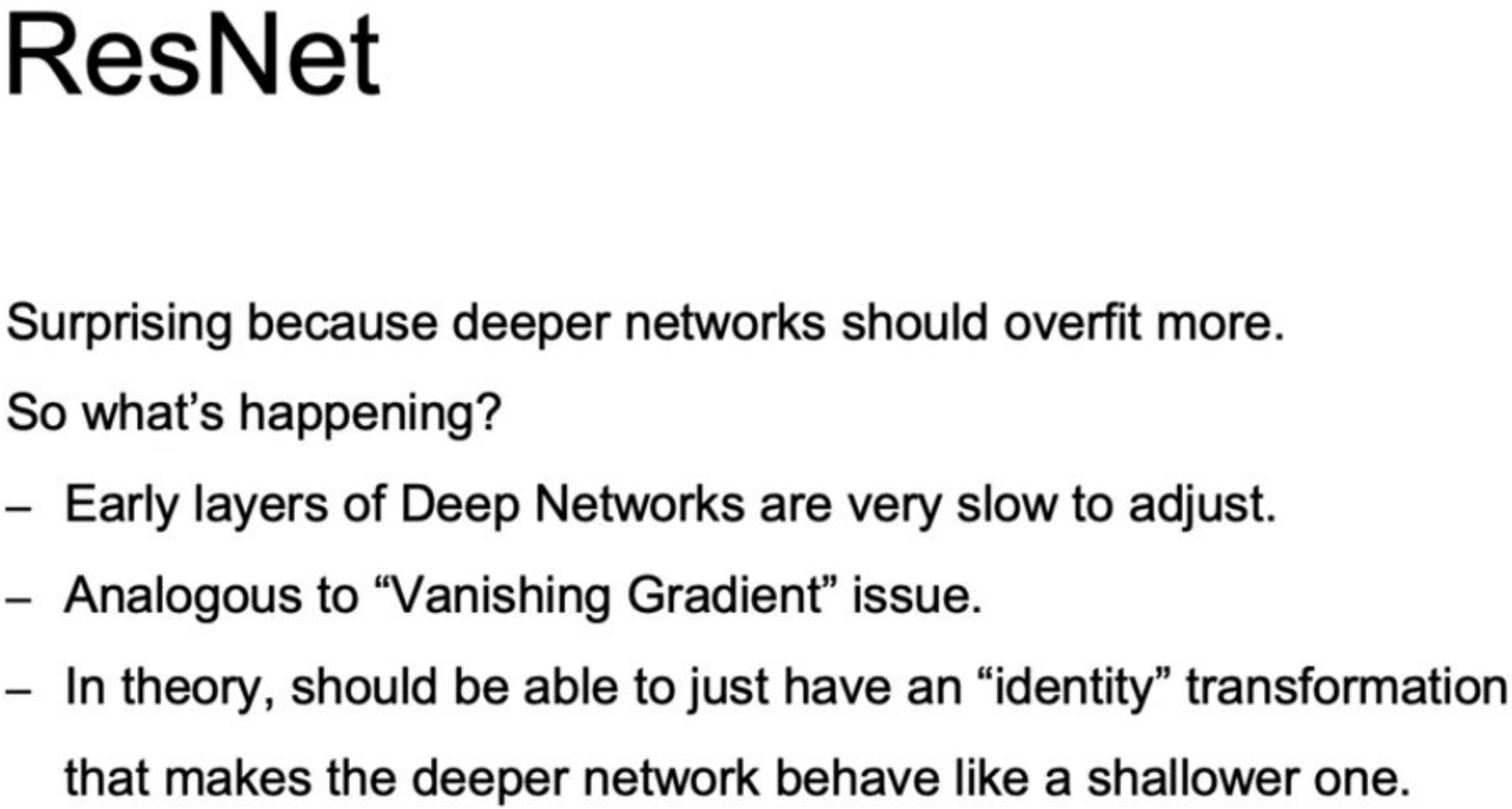
Inception V3 Schematic



ResNet - Motivation

Issue: Deeper Networks performing worse on training data! (as well as test data)





ResNet

Surprising because deeper networks should overfit more.

So what's happening?

- Early layers of Deep Networks are very slow to adjust.
- Analogous to "Vanishing Gradient" issue.
- In theory, should be able to just have an "identity" transformation that makes the deeper network behave like a shallower one.

ResNet

Assumption: best transformation over multiple layers is close to $\mathcal{F}(x) + x$.

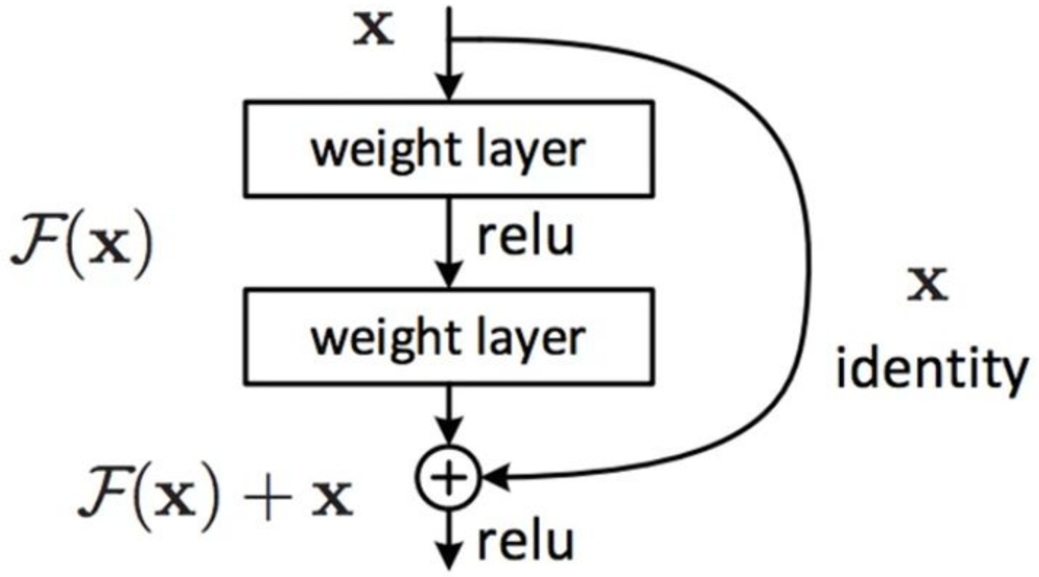
x: input to series of layers.

$\mathcal{F}(x)$: function represented

by several layers

(such as convolutions).

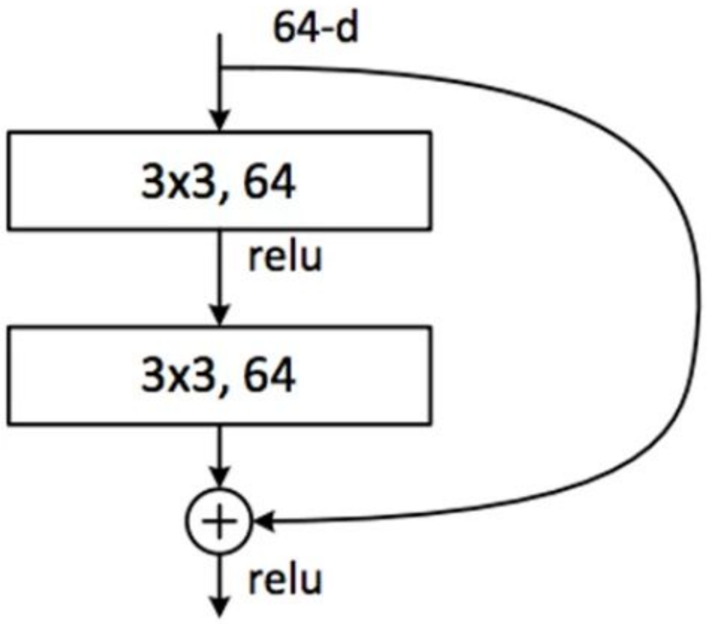
- Enforce "best transformation" by adding "shortcut connections".
- Add the inputs from an earlier layer to the output of current layer.


x
weight layer
relu
weight layer
relu
$\mathcal{F}(\mathbf{x})$
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
x
identity

ResNet

Add previous layer back in to current layer!

Avoids vanishing gradient issue.

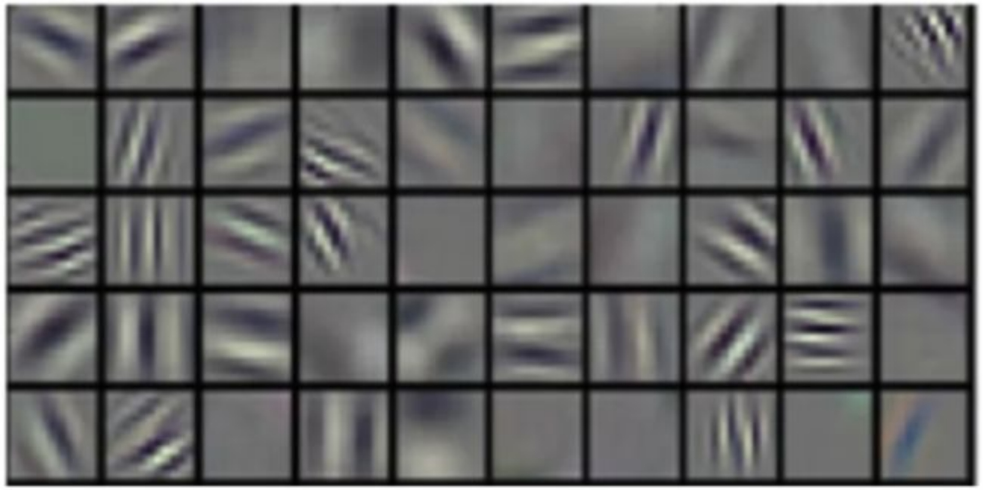


Motivation

Early layers in a Neural Network are the hardest (i.e. slowest) to train.

– Due to vanishing gradient property.

– But these ”primitive” features should be general across many image classification tasks.



Motivation

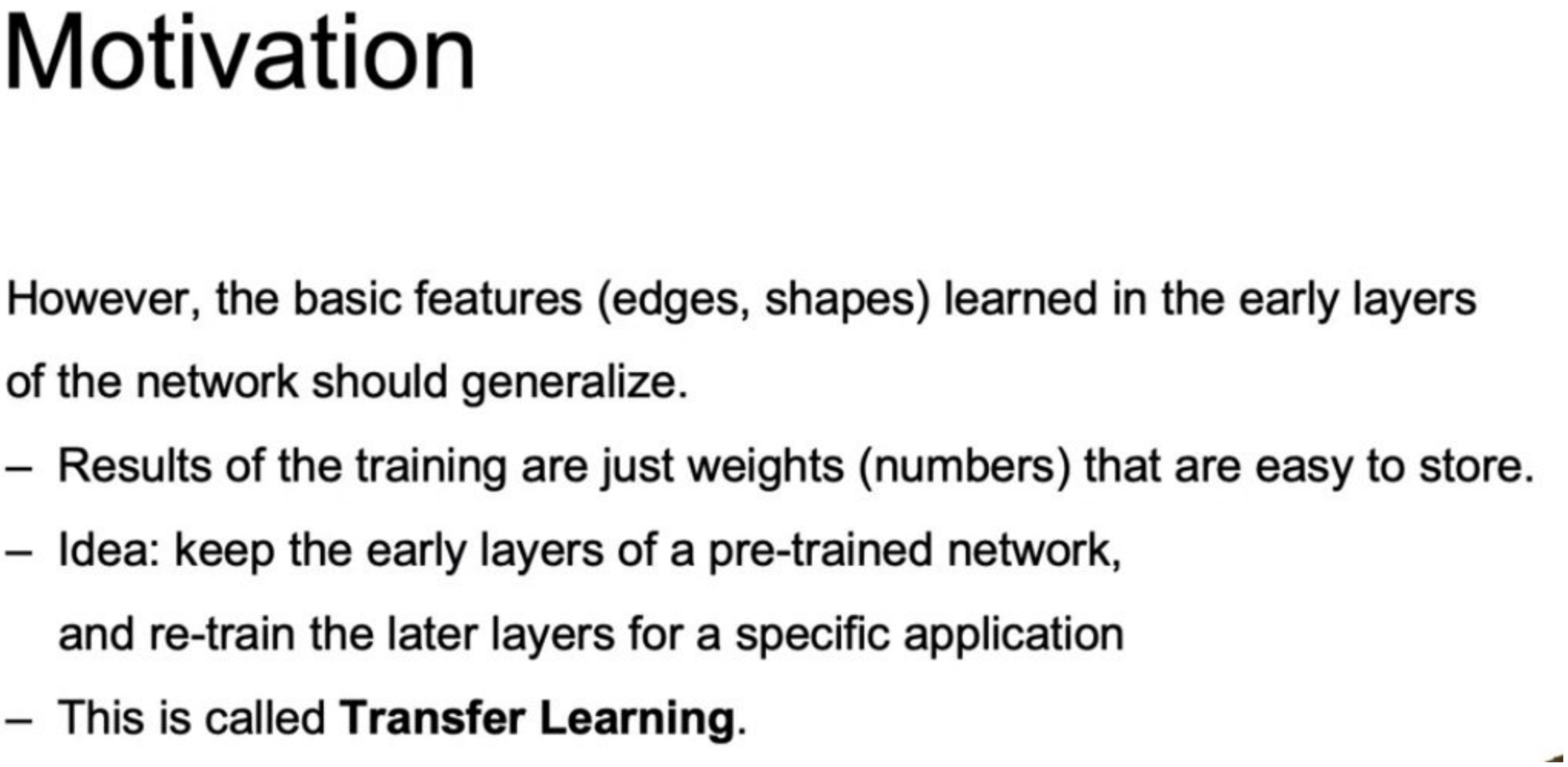
Later layers in the network are capturing features that are more particular to the specific image classification problem.

- Later layers are easier (quicker) to train since adjusting their weights has a more immediate impact on the final result.

Famous, Competition-Winning Models are difficult to train from scratch:

- Huge datasets (like ImageNet)
- Long number of training iterations
- Very heavy computing machinery
- Time experimenting to get hyper-parameters just right

Transfer Learning



Motivation

However, the basic features (edges, shapes) learned in the early layers of the network should generalize.

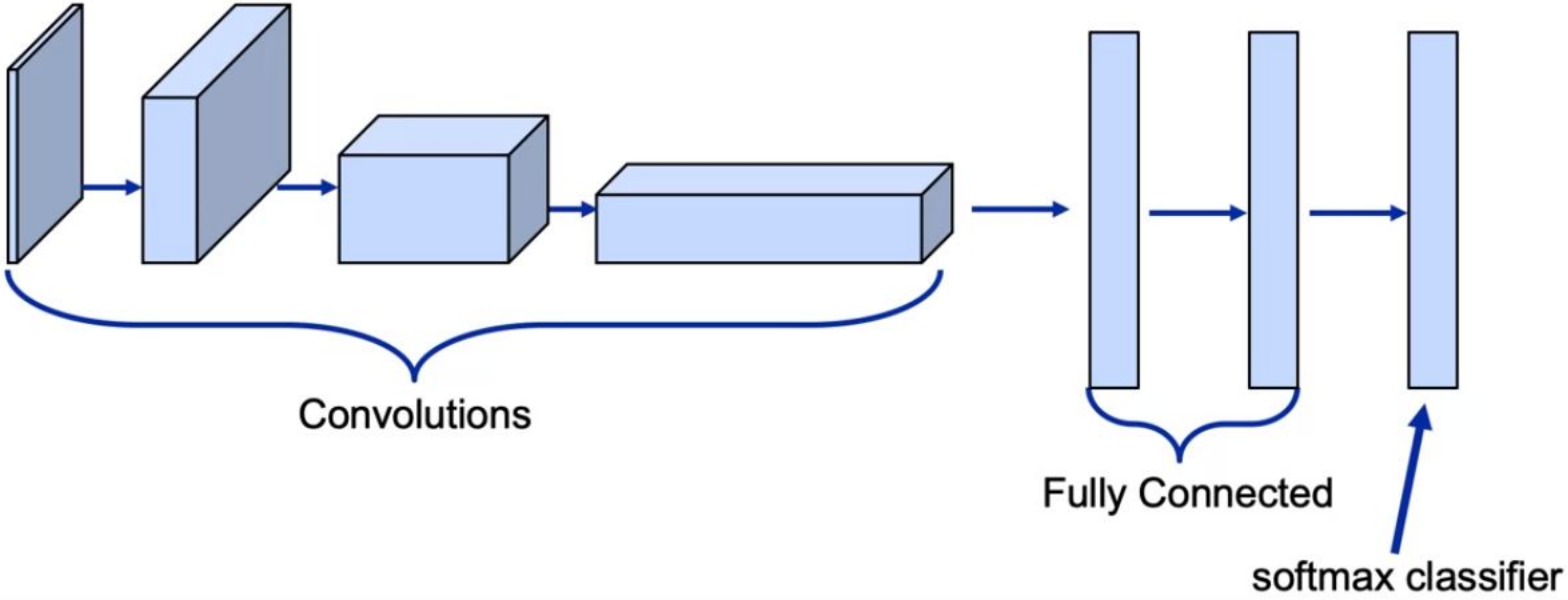
– Results of the training are just weights (numbers) that are easy to store.

– Idea: keep the early layers of a pre-trained network, and re-train the later layers for a specific application

– This is called **Transfer Learning**.

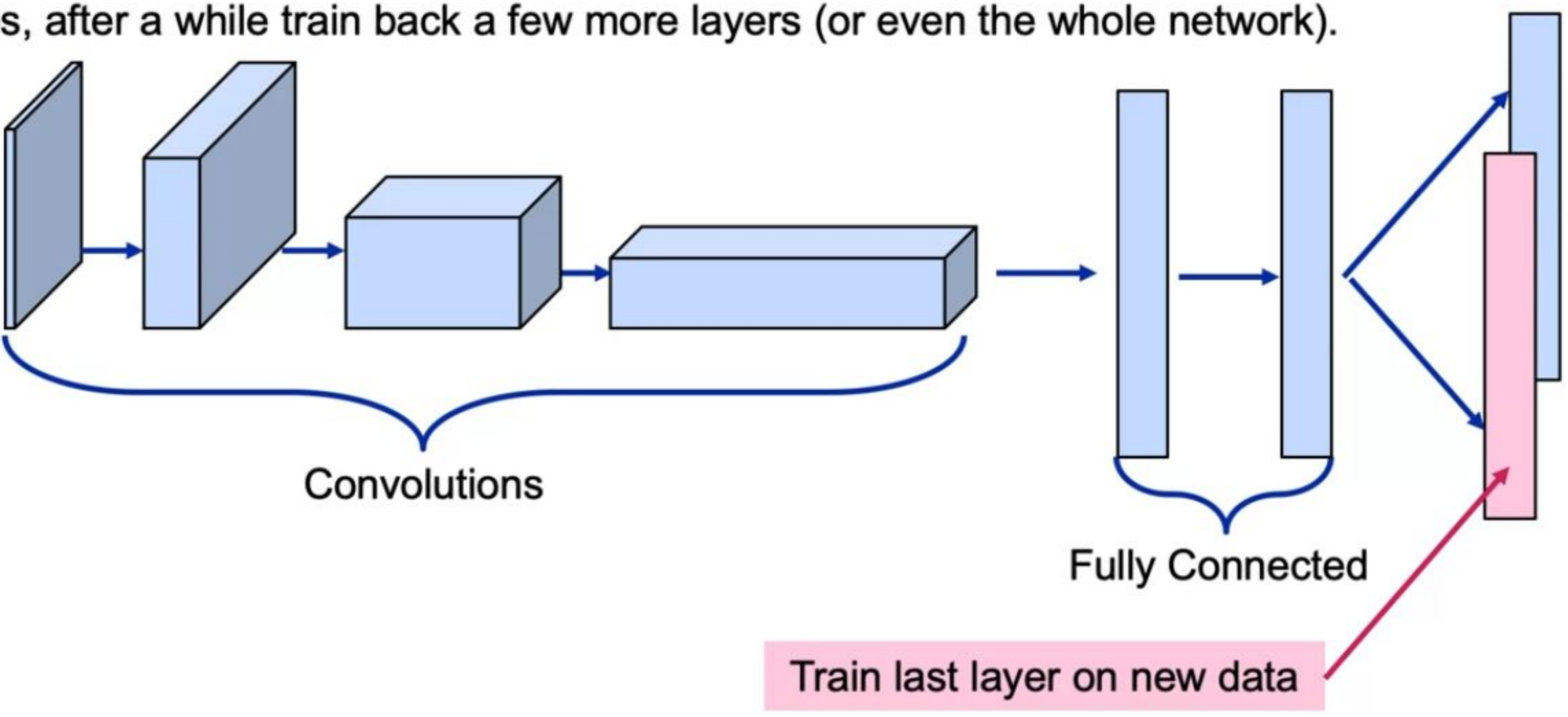
Transfer Learning

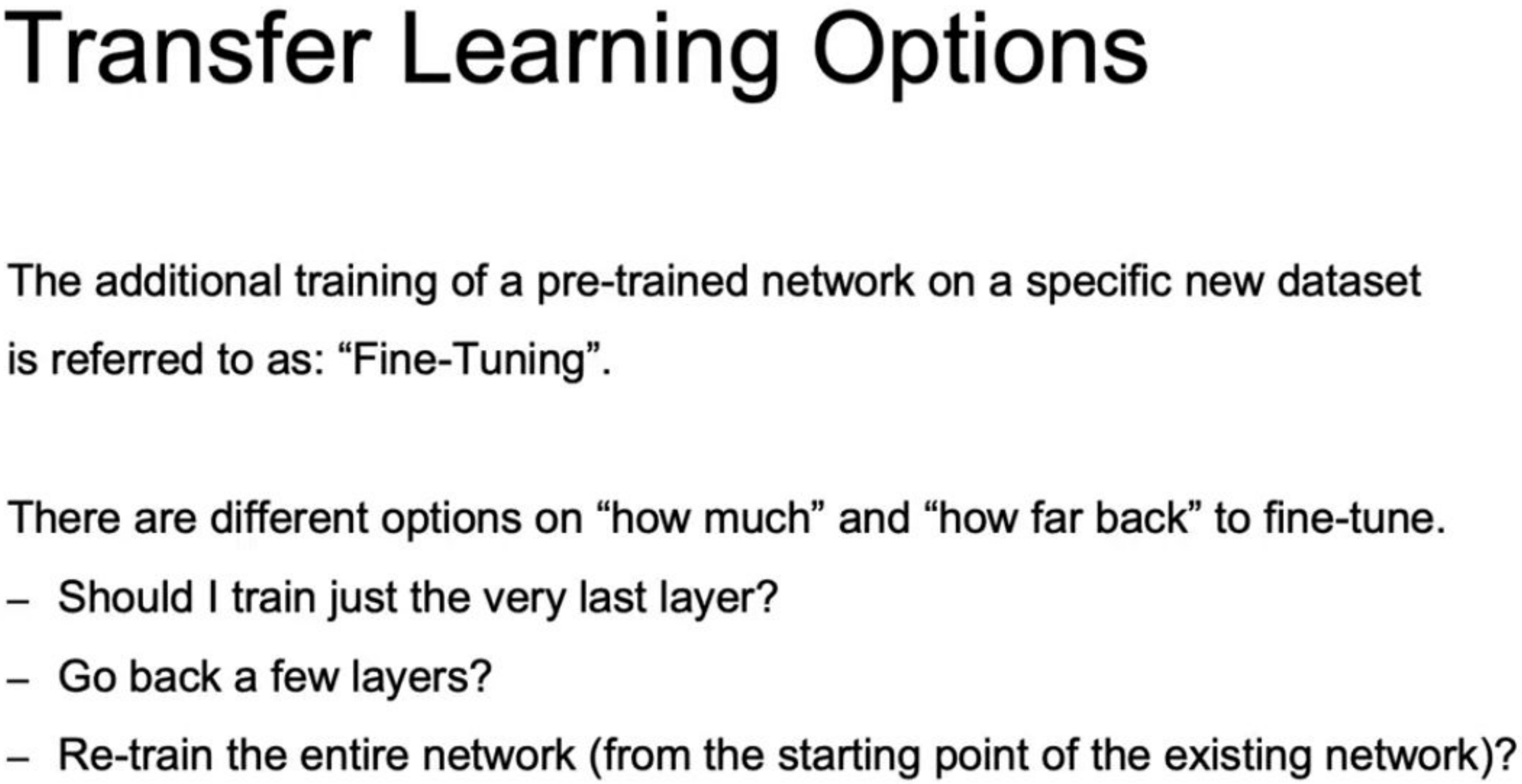
Transfer Learning



Transfer Learning

Perhaps, after a while train back a few more layers (or even the whole network).





Transfer Learning Options

The additional training of a pre-trained network on a specific new dataset is referred to as: “Fine-Tuning”.

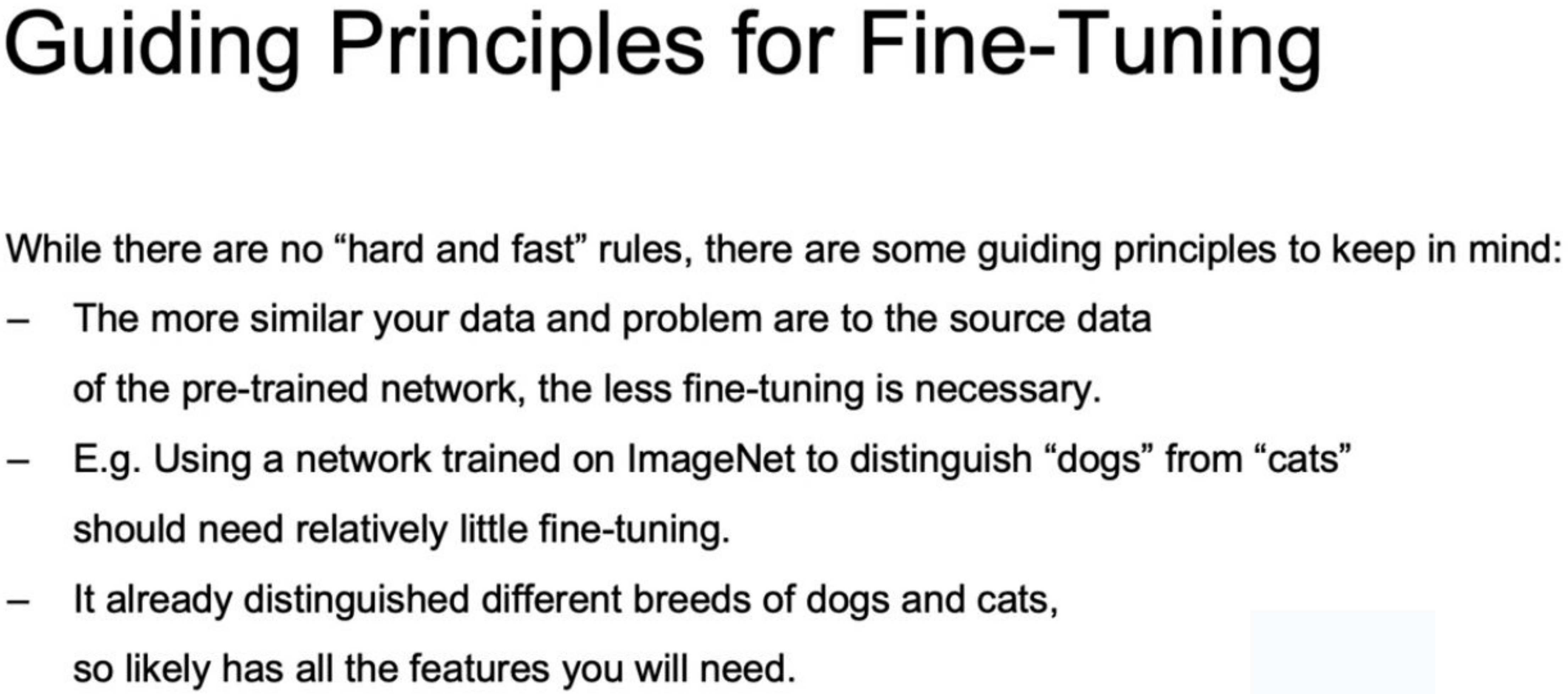
There are different options on “how much” and “how far back” to fine-tune.

– Should I train just the very last layer?

– Go back a few layers?

– Re-train the entire network (from the starting point of the existing network)?

Transfer Learning



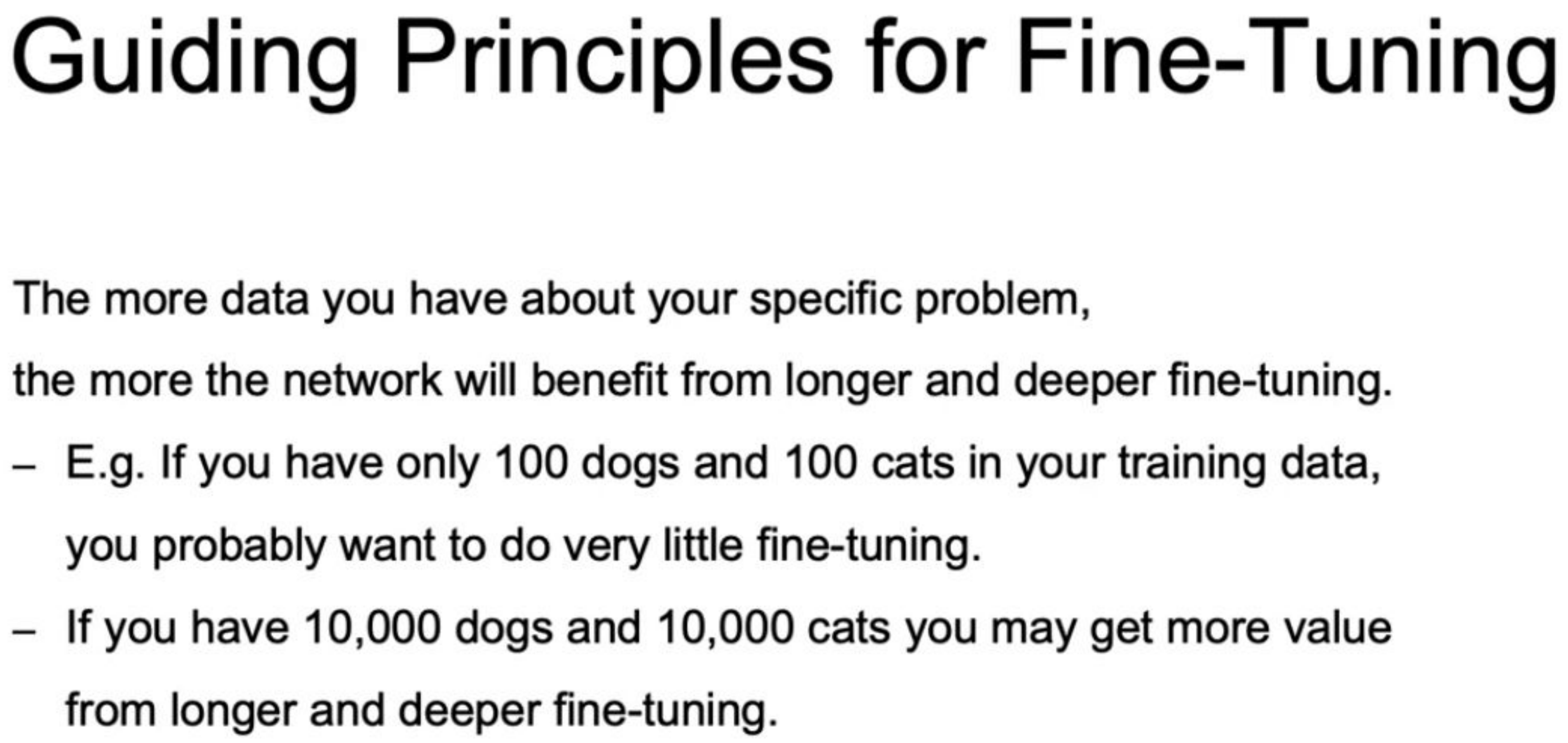
Guiding Principles for Fine-Tuning

While there are no “hard and fast” rules, there are some guiding principles to keep in mind:

– The more similar your data and problem are to the source data of the pre-trained network, the less fine-tuning is necessary.

– E.g. Using a network trained on ImageNet to distinguish “dogs” from “cats” should need relatively little fine-tuning.

– It already distinguished different breeds of dogs and cats, so likely has all the features you will need.



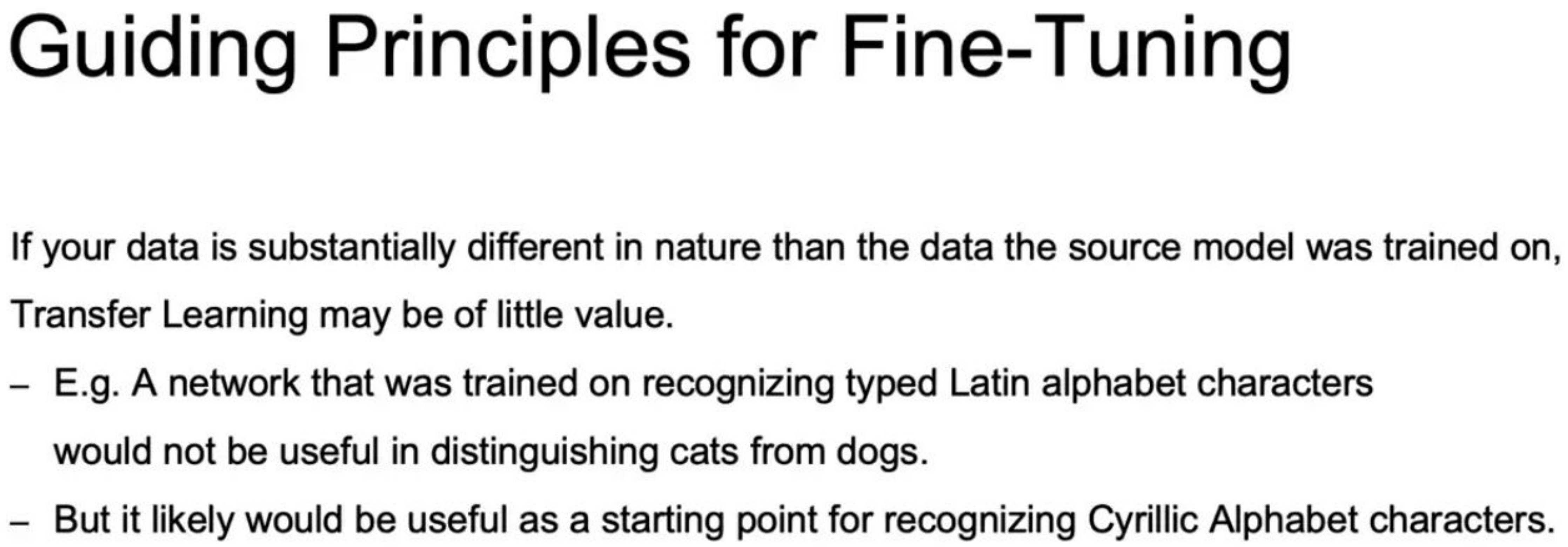
Guiding Principles for Fine-Tuning

The more data you have about your specific problem,

the more the network will benefit from longer and deeper fine-tuning.

- E.g. If you have only 100 dogs and 100 cats in your training data, you probably want to do very little fine-tuning.
- If you have 10,000 dogs and 10,000 cats you may get more value from longer and deeper fine-tuning.

Transfer Learning



Guiding Principles for Fine-Tuning

If your data is substantially different in nature than the data the source model was trained on, Transfer Learning may be of little value.

- E.g. A network that was trained on recognizing typed Latin alphabet characters would not be useful in distinguishing cats from dogs.
- But it likely would be useful as a starting point for recognizing Cyrillic Alphabet characters.

Summaries

Transfer Learning

The main idea of Transfer Learning consists of keeping early layers of a pre-trained network and re-train the later layers for a specific application.
Last layers in the network capture features that are more particular to the specific data you are trying to classify.
Later layers are easier to train as adjusting their weights has a more immediate impact on the final result.

Guiding Principles for Fine Tuning

While there are no rules of thumb, these are some guiding principles to keep in mind:

- The more similar your data and problem are to the source data of the pre-trained network, the less intensive fine-tuning will be.
- If your data is substantially different in nature than the data the source model was trained on, Transfer Learning may be of little value.

CNN Architectures

LeNet-5

- Created by Yann LeCun in the 1990s
- Used on the MNIST data set.
- Novel Idea: Use convolutions to efficiently learn features on data set.

AlexNet

- Considered the “flash point” for modern deep learning
- Created in 2012 for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC).
- Task: predict the correct label from among 1000 classes.
- Dataset: around 1.2 million images.

AlexNet developers performed data augmentation for training.

- Cropping, horizontal flipping, and other manipulations.

Basic AlexNet Template:

- Convolutions with ReLUs.
- Sometimes add maxpool after convolutional layer.
- Fully connected layers at the end before a softmax classifier.

Summaries

VGG

Simplify Network Structure: has same concepts and ideas from LeNet, considerably deeper.

This architecture avoids Manual Choices of Convolution Size and has very Deep Network with 3x3 Convolutions.

These structures tend to give rise to larger convolutions.

This was one of the first architectures to experiment with many layers (More is better!). It can use multiple 3x3 convolutions to simulate larger kernels with fewer parameters and it served as ”base model” for future works.

Inception

Ideated by Szegedy et al 2014, this architecture was built to turn each layer of the neural network into further branches of convolutions. Each branch handles a smaller portion of workload. The network concatenates different branches at the end. These networks use different receptive fields and have sparse activations of groups of neurons.

Inception V3 is a relevant example of an Inception architecture.

ResNet

Researchers were building deeper and deeper networks but started finding these issues:

In theory, the very deep (56-layer) networks should fit the training data better (even if they overfit) but that was not happening.

Seemed that the early layers were just not getting updated and the signal got lost (due to vanishing gradient type issues).

These are the main reasons why adding layers does not always decrease training error:

- Early layers of Deep Networks are very slow to adjust.
- Analogous to “Vanishing Gradient” issue.
- In theory, should be able to just have an “identity” transformation that makes the deeper network behave like a shallow one.

In a nutshell, a ResNet:

- Has several layers such as convolutions
- Enforces “best transformation” by adding “shortcut connections”.
- Adds the inputs from an earlier layer to the output of current layer.
- Keeps passing both the the initial unchanged information and the transformed information to the next layer.