ConvNets and Transfer Learning

Review

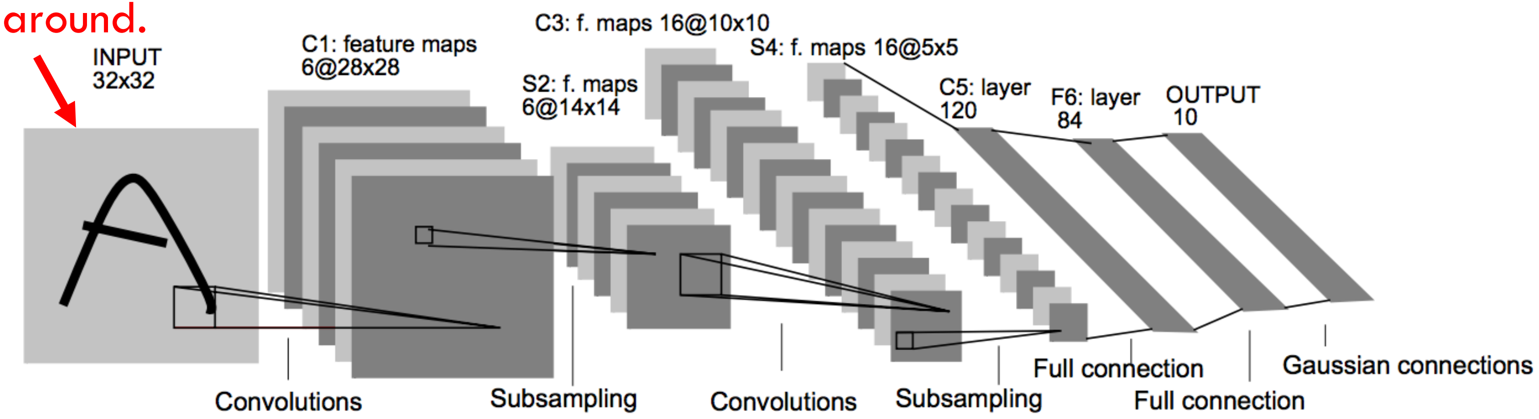
- Do some review of concepts from the last lecture
- We will revisit kernel, stride, and pooling in the context of the Le-Net 5 model.

LeNet-5

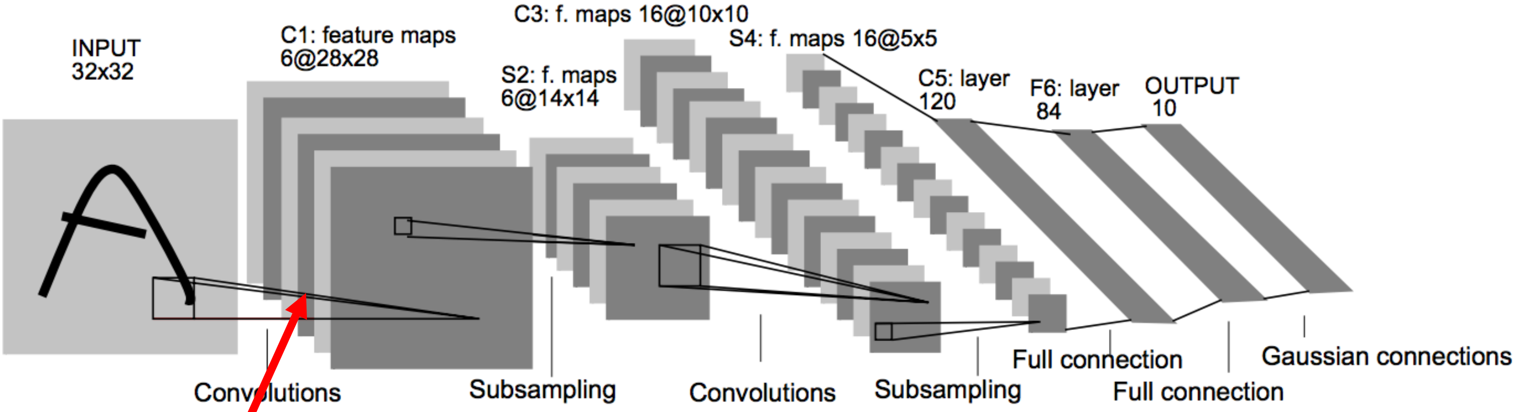
- Created by Yann LeCun in the 1990s
- Used on the MNIST data set.
- Novel Idea: Use convolutions to efficiently learn features on data set.

LeNet – Structure Diagram

Input: A 32 x 32 grayscale image (28 x 28) with 2 pixels of padding all around.

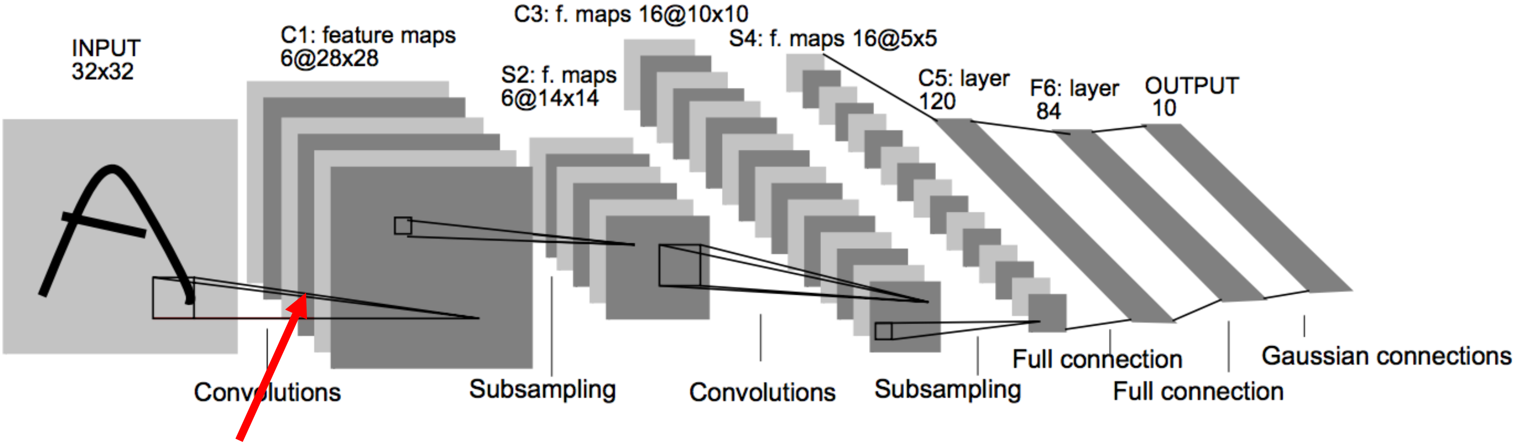


LeNet – Structure Diagram



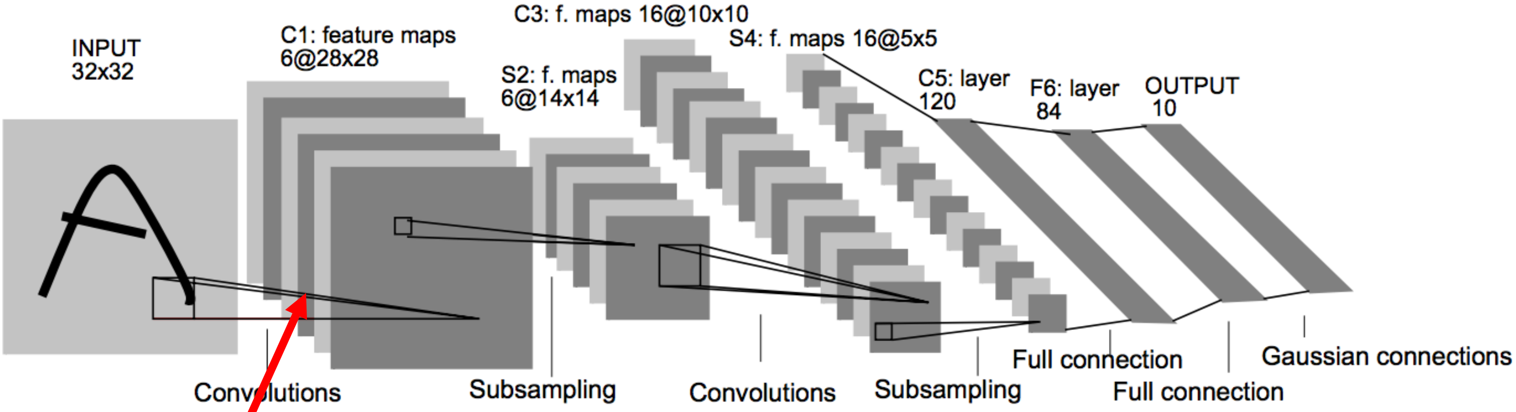
Next, we have a convolutional layer.

LeNet – Structure Diagram



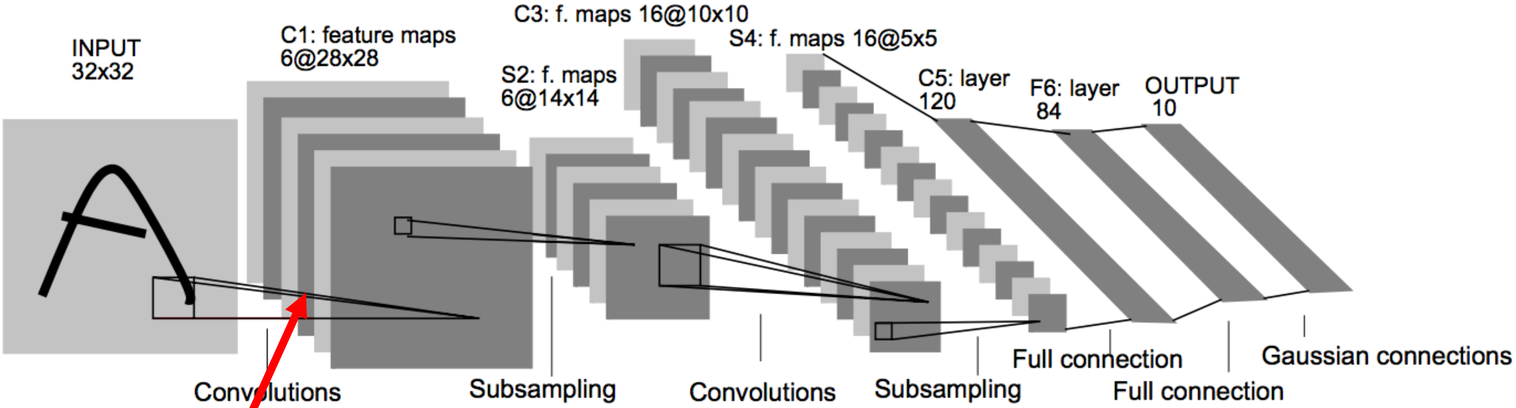
This is a 5x5 convolutional layer with stride 1.

LeNet – Structure Diagram



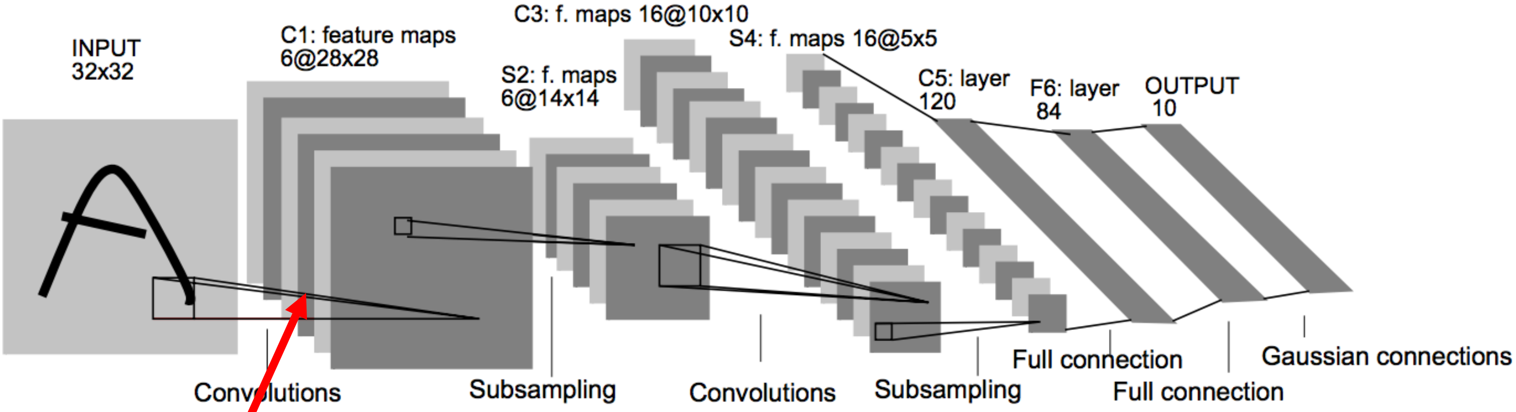
This means the resulting "filter" has dimension 28x28. (Why?)

LeNet – Structure Diagram



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

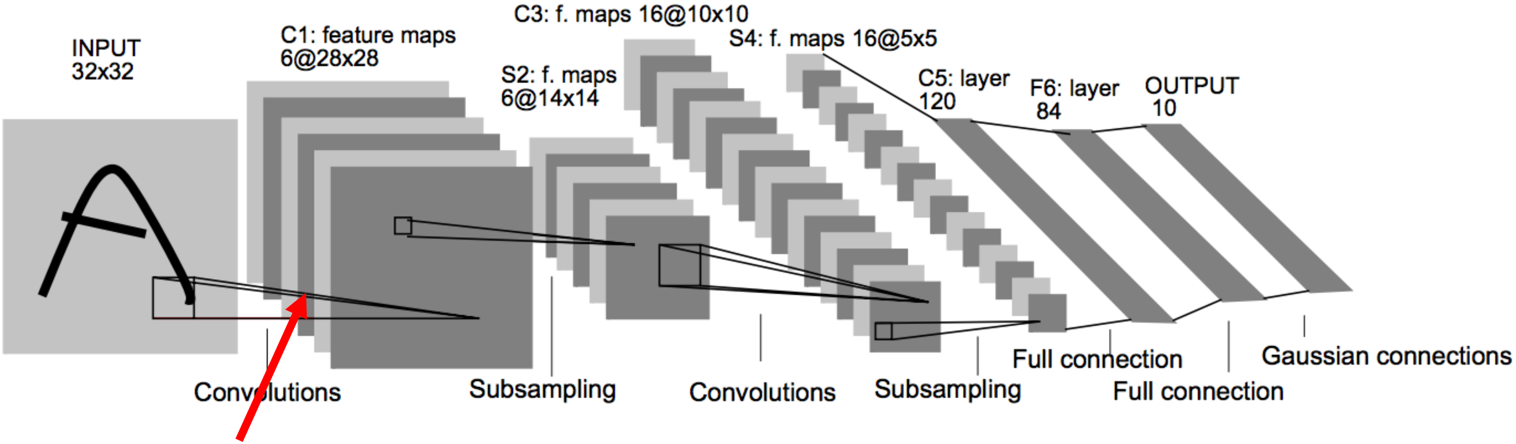
LeNet – Structure Diagram



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

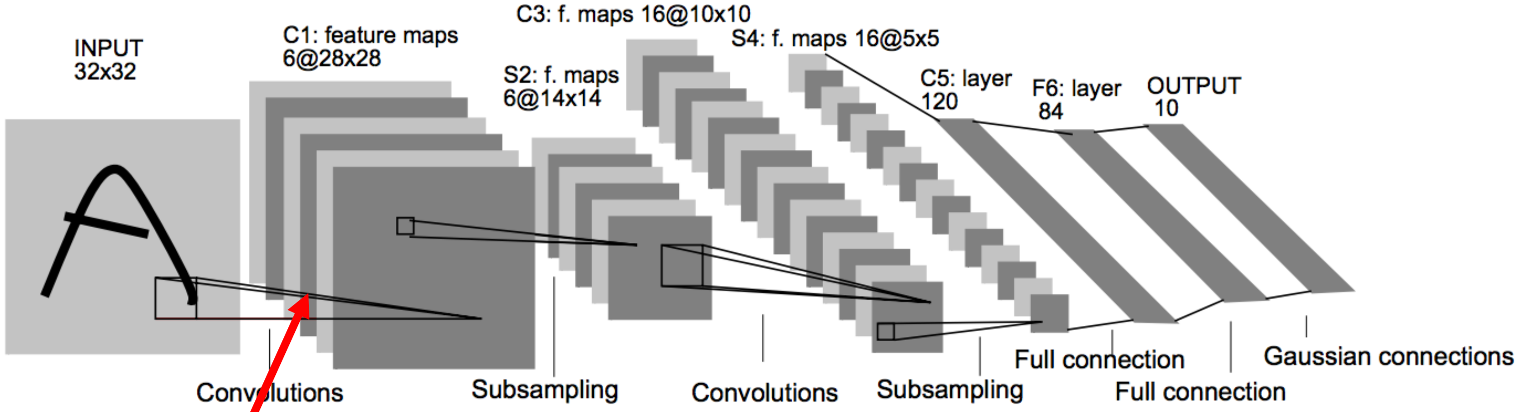
So the output of this layer is 6x28x28.

LeNet – Structure Diagram



What is the total number of weights in this layer?

LeNet – Structure Diagram

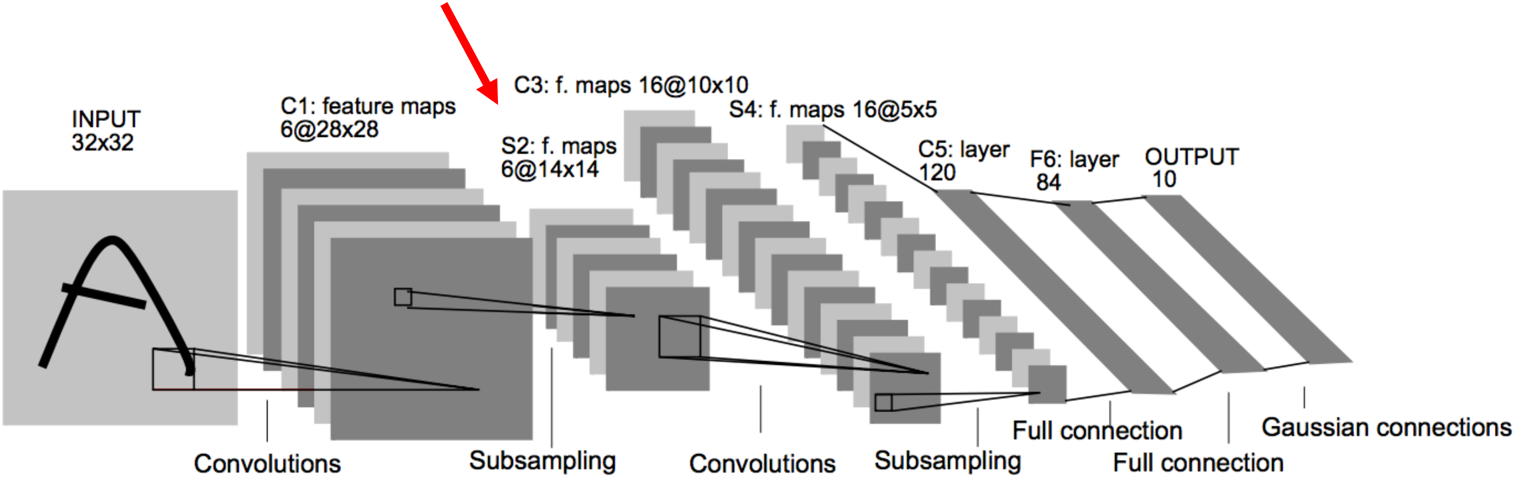


What is the total number of weights in this layer?

Answer: Each kernel has 5x5=25 weights (plus a bias term, so actually 26 weights). So total weights = 6x26 = 156.

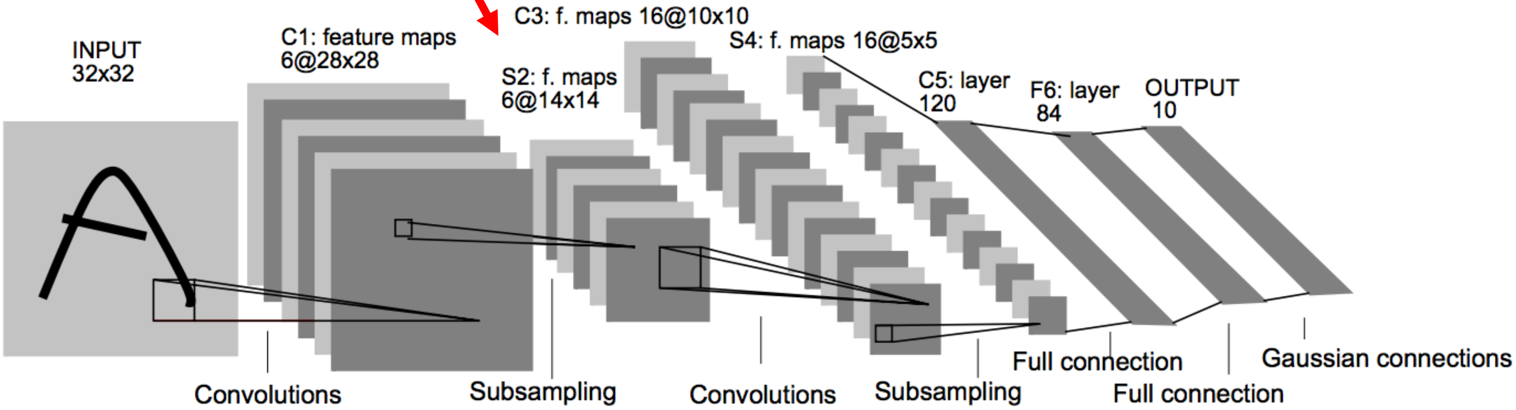
LeNet – Structure Diagram

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



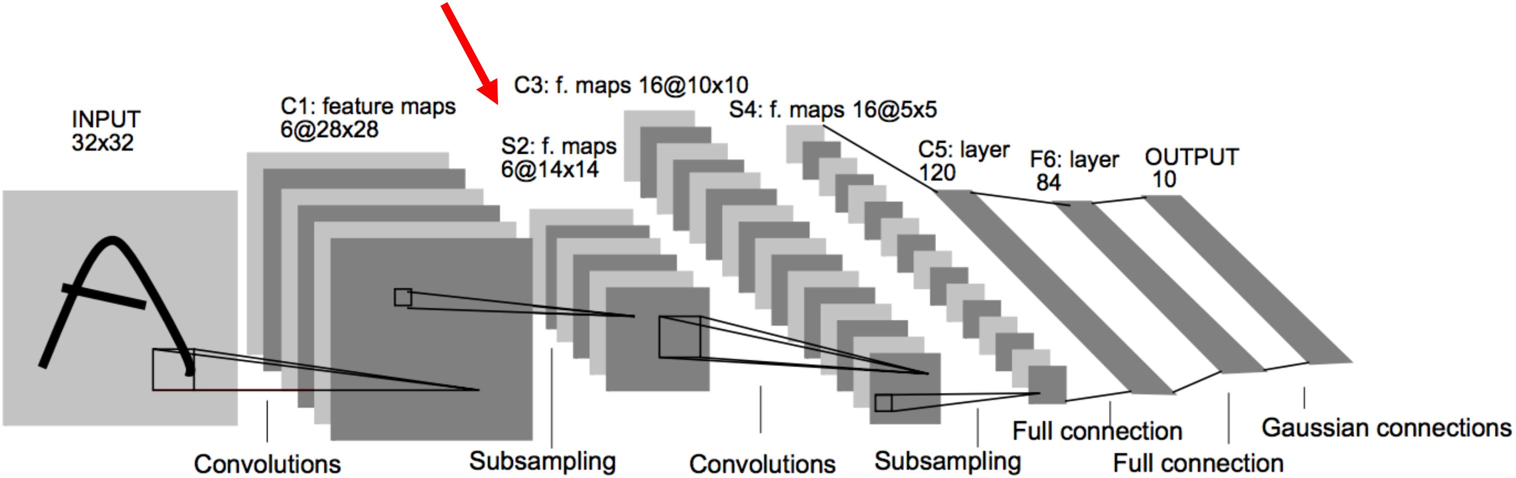
LeNet – Structure Diagram

So output size is 6x14x14 (we downsample by a factor of 2)



LeNet – Structure Diagram

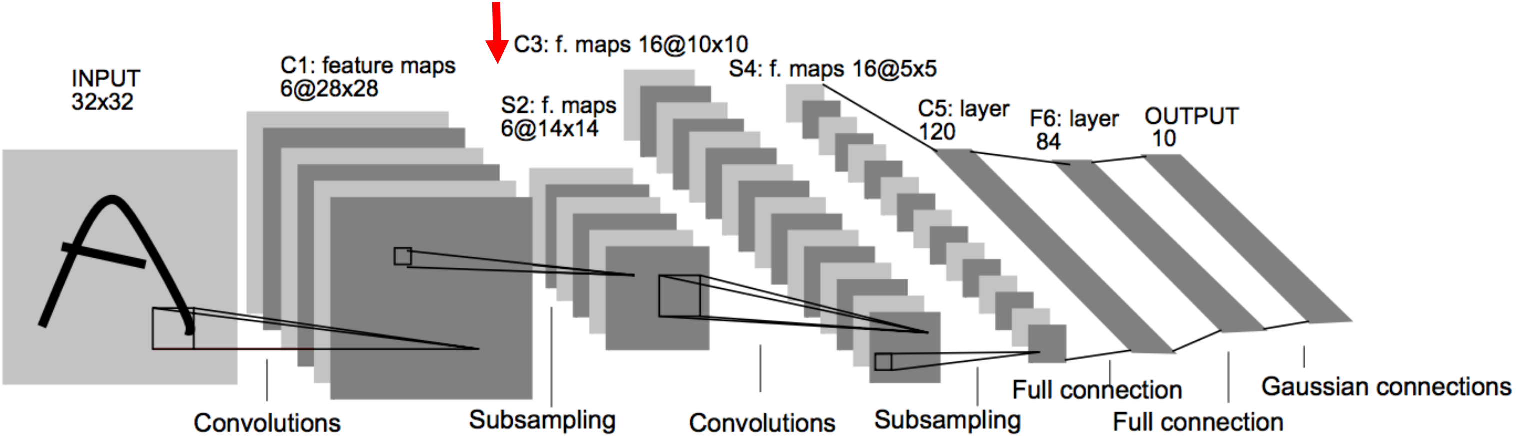
So output size is 6x14x14 (we downsample by a factor of 2)



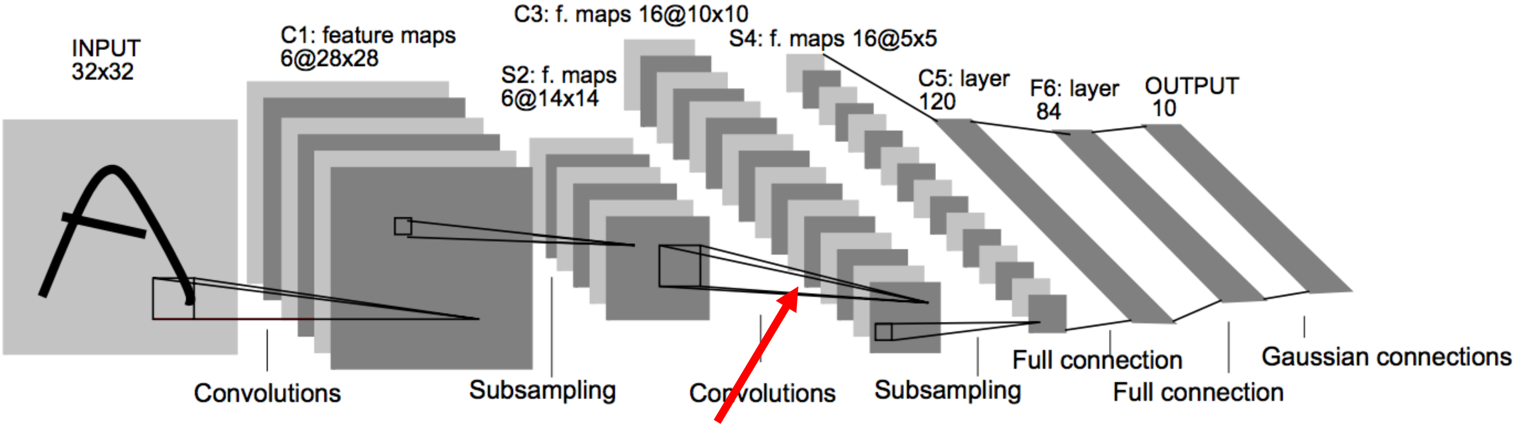
Note: The original paper actually does a more complicated pooling then max or avg pooling, but this is considered obsolete now.

LeNet – Structure Diagram

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



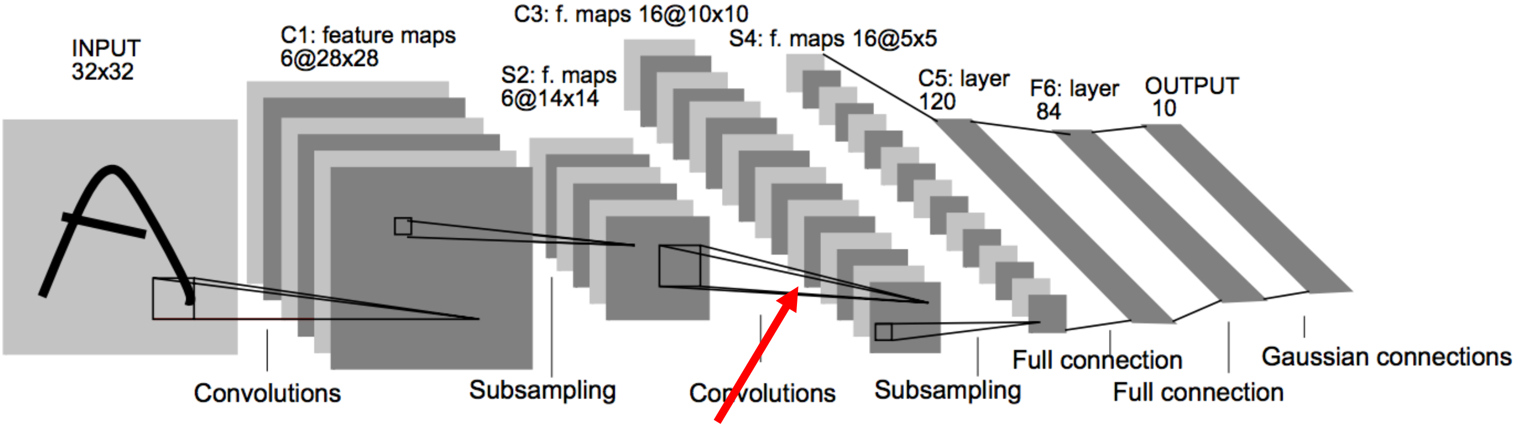
LeNet – Structure Diagram



Another 5x5 convolutional layer with stride 2.

This time the depth is 16.

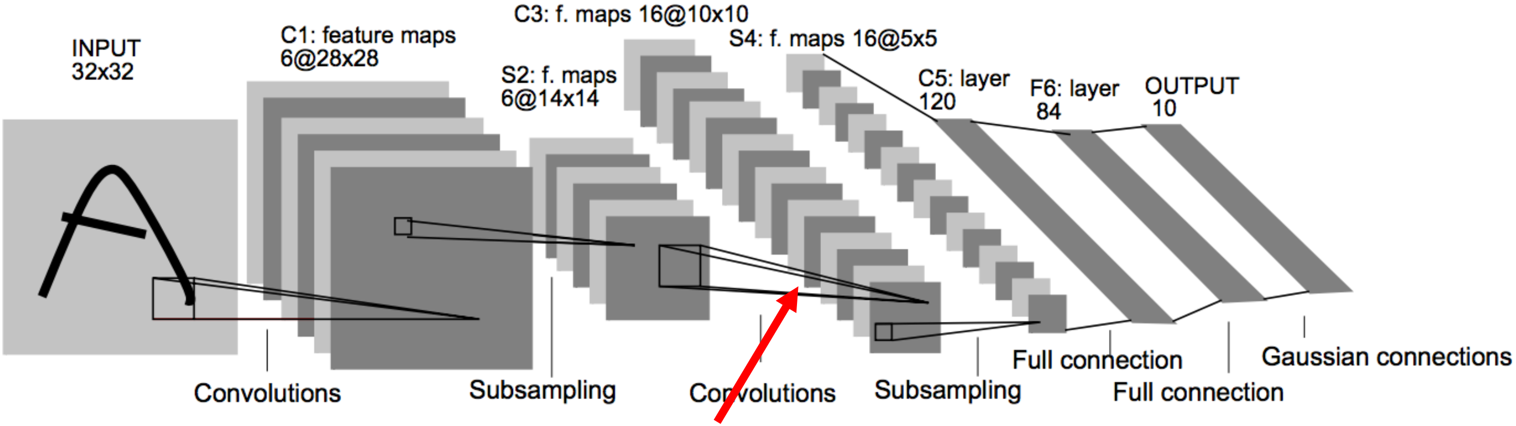
LeNet – Structure Diagram



Output size: 16 x 10 x 10

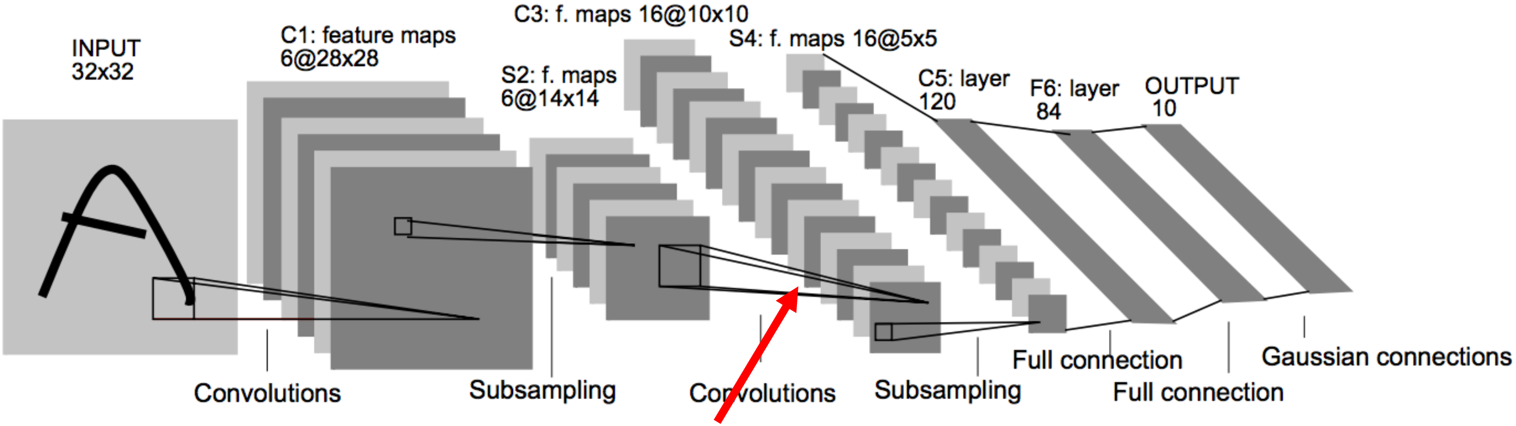
How many weights? (tricky!)

LeNet – Structure Diagram



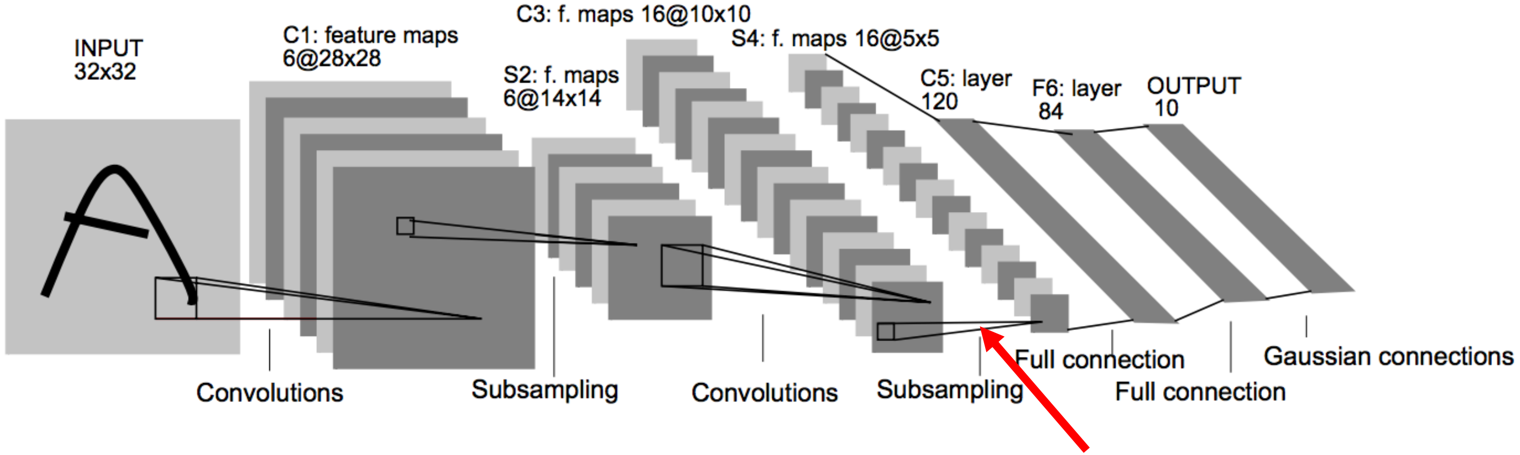
The kernels "take in" the full depth of the previous layer.
So each 5x5 kernel now "looks at" 6x5x5 pixels.
Each kernel has 6x5x5 = 150 weights + bias term = 151.

LeNet – Structure Diagram



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

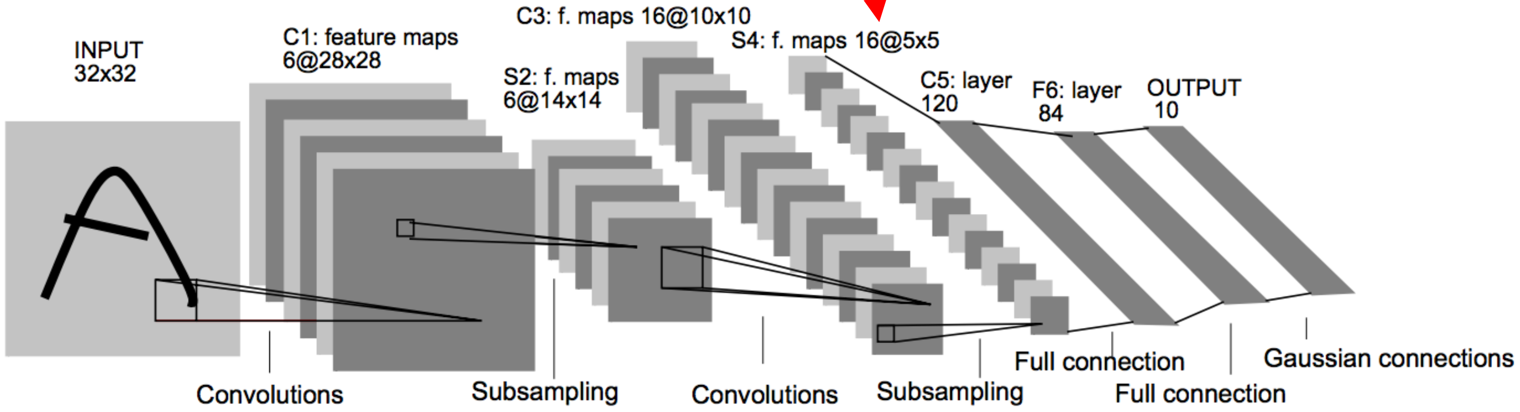
LeNet – Structure Diagram



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

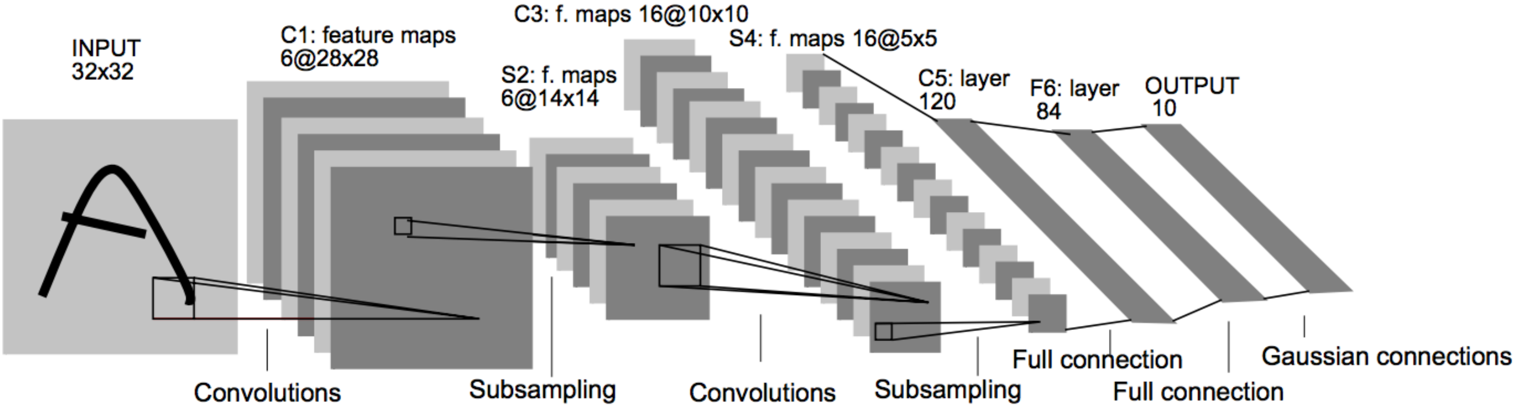
LeNet – Structure Diagram

We "flatten" this to a length 400 vector (not shown)

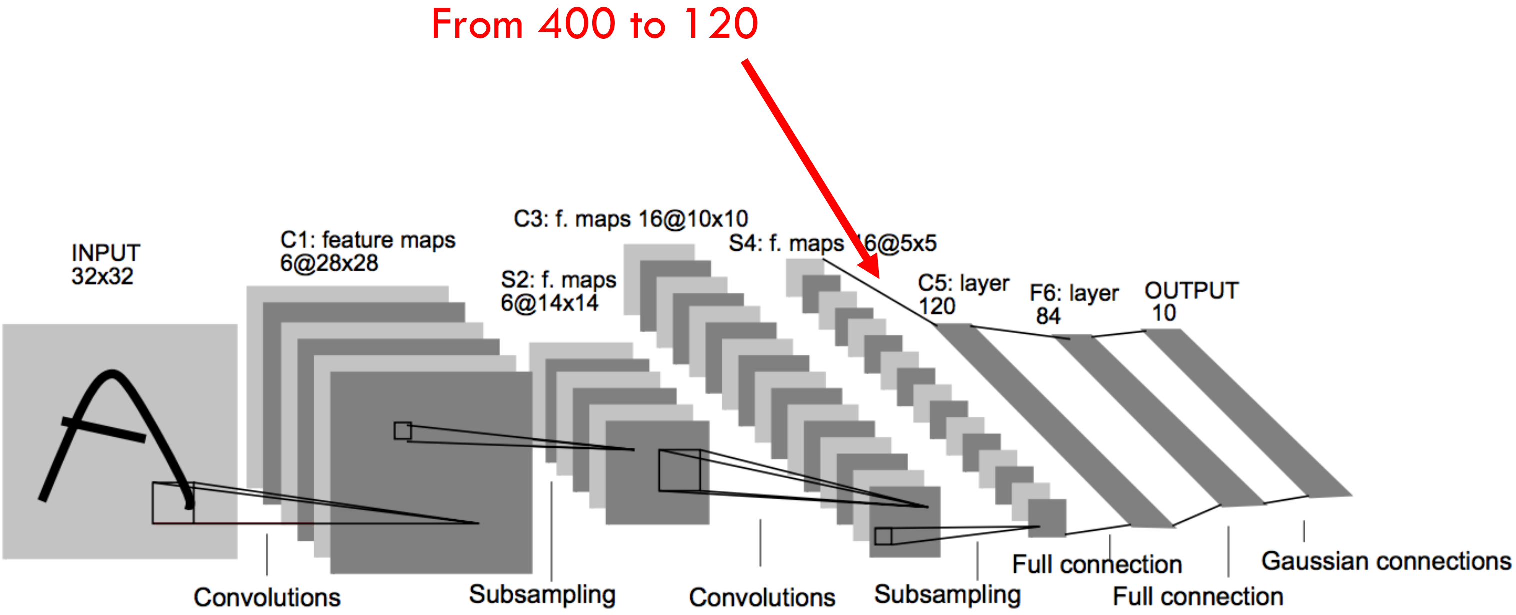


LeNet – Structure Diagram

The following layers are just
fully connected layers!

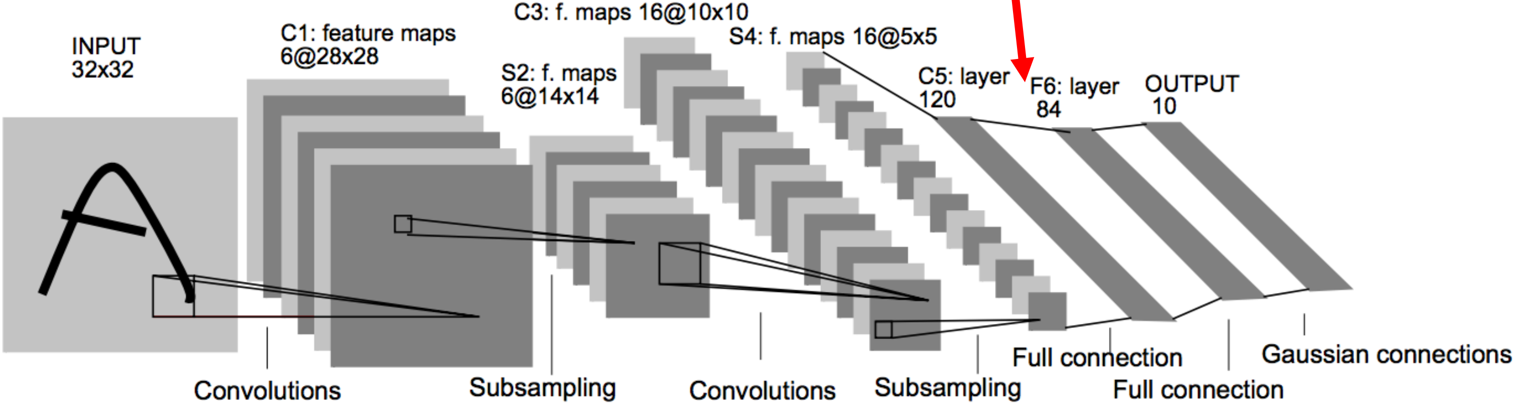


LeNet – Structure Diagram



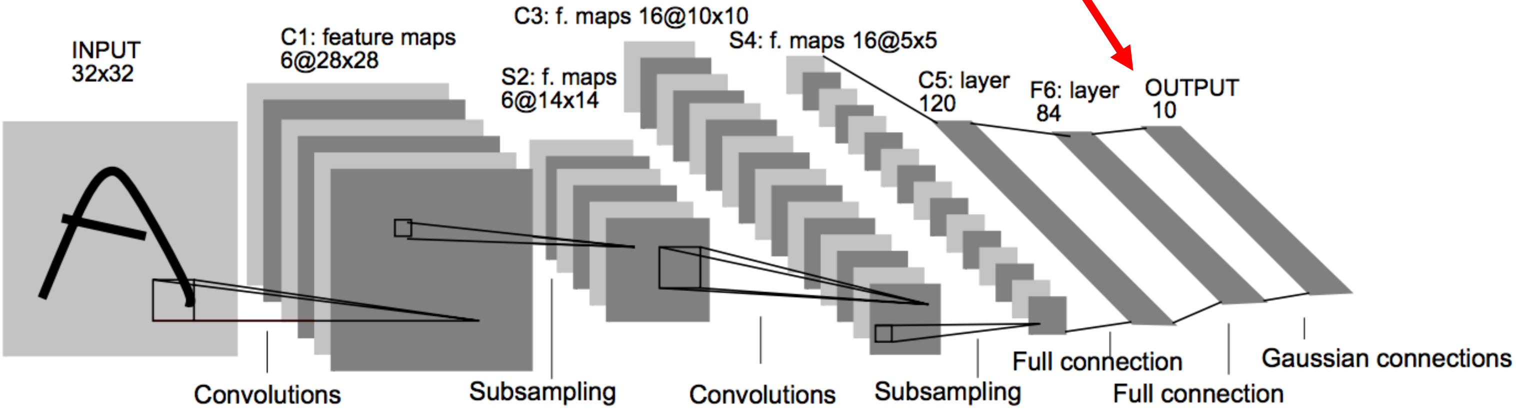
LeNet – Structure Diagram

Then from 120 to 84



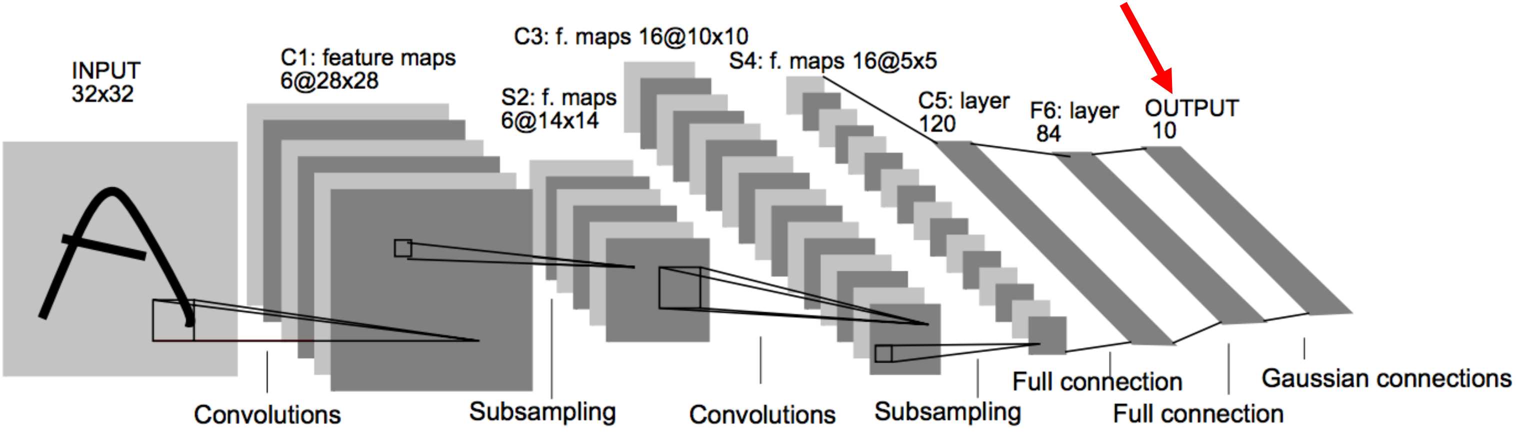
LeNet – Structure Diagram

Then from 84 to 10



LeNet – Structure Diagram

And a softmax output of size 10 for the 10 digits



LeNet-5

How many total weights in the network?

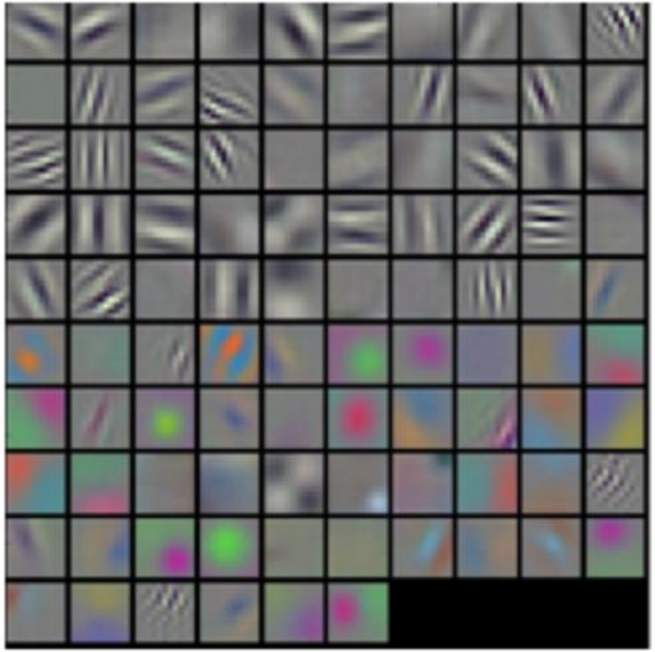
Conv1: 1*6*5*5 + 6 = 156
Conv3: 6*16*5*5 + 16 = 2416
FC1: 400*120 + 120 = 48120
FC2: 120*84 + 84 = 10164
FC3: 84*10 + 10 = 850
Total: = 61706

Less than a single FC layer with [1200x1200] weights!
Note that Convolutional Layers have relatively few weights.

Transfer Learning

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.

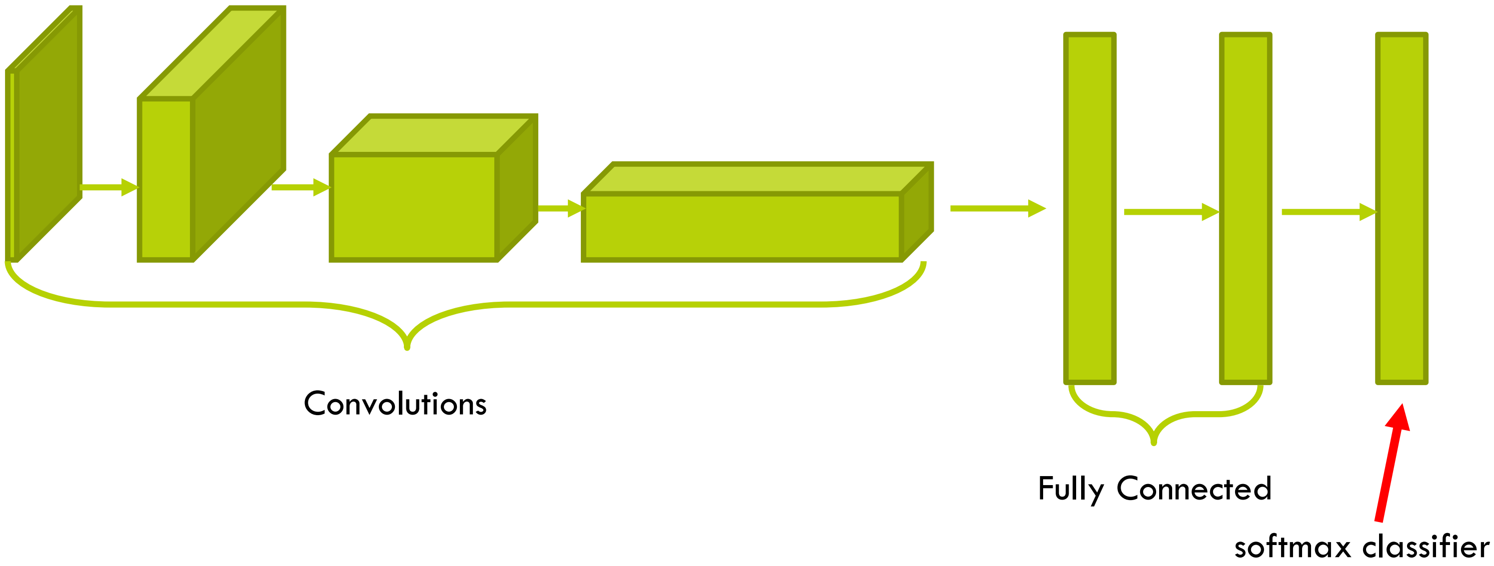
Motivation

- 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

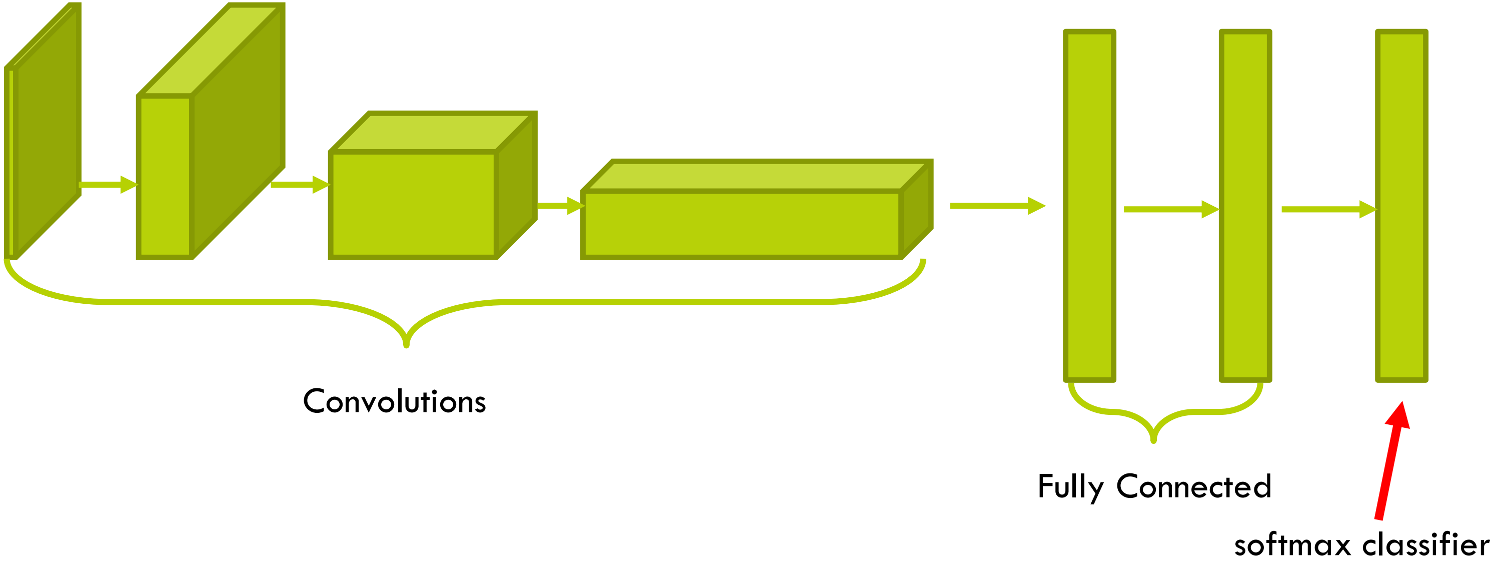
Transfer Learning

- 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***.

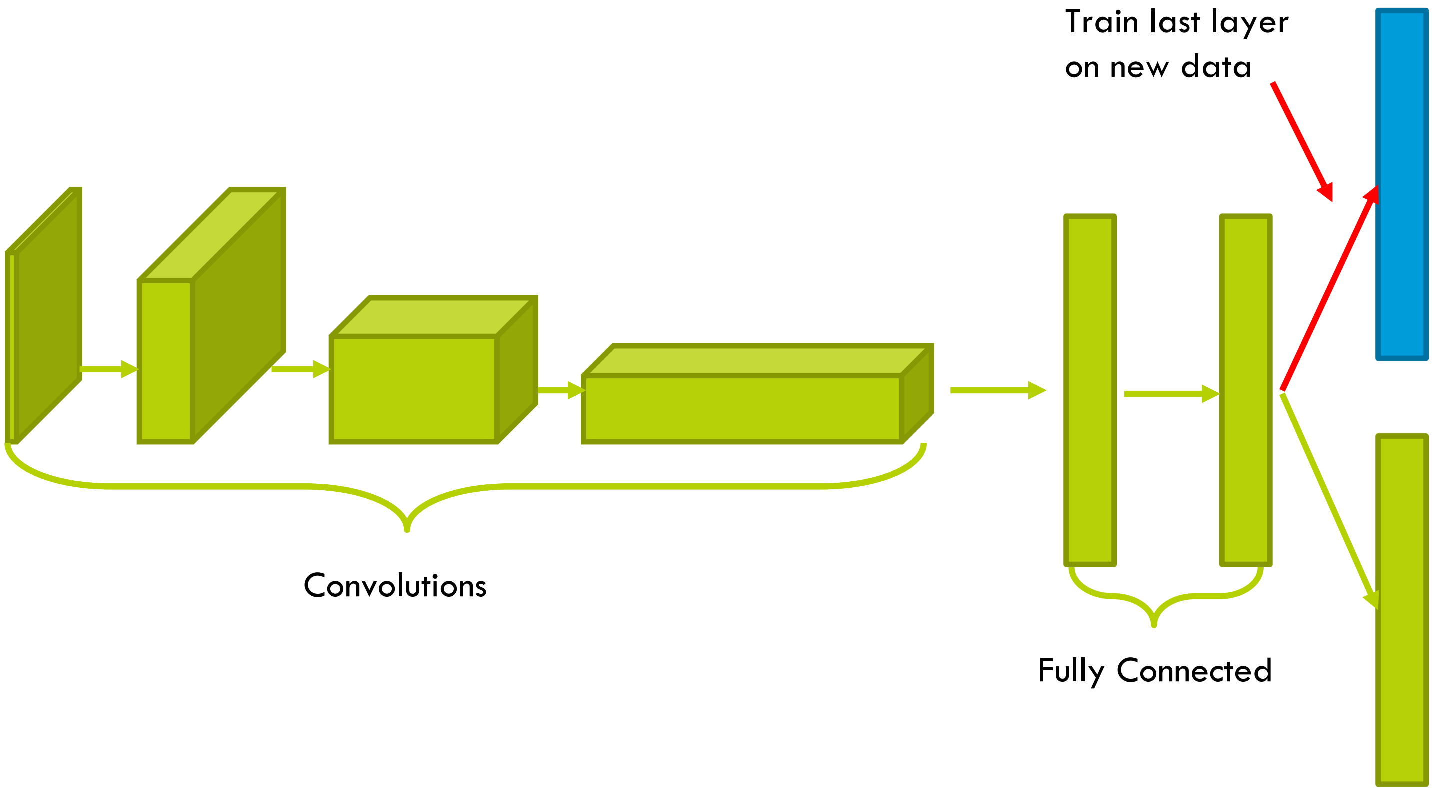
Transfer Learning



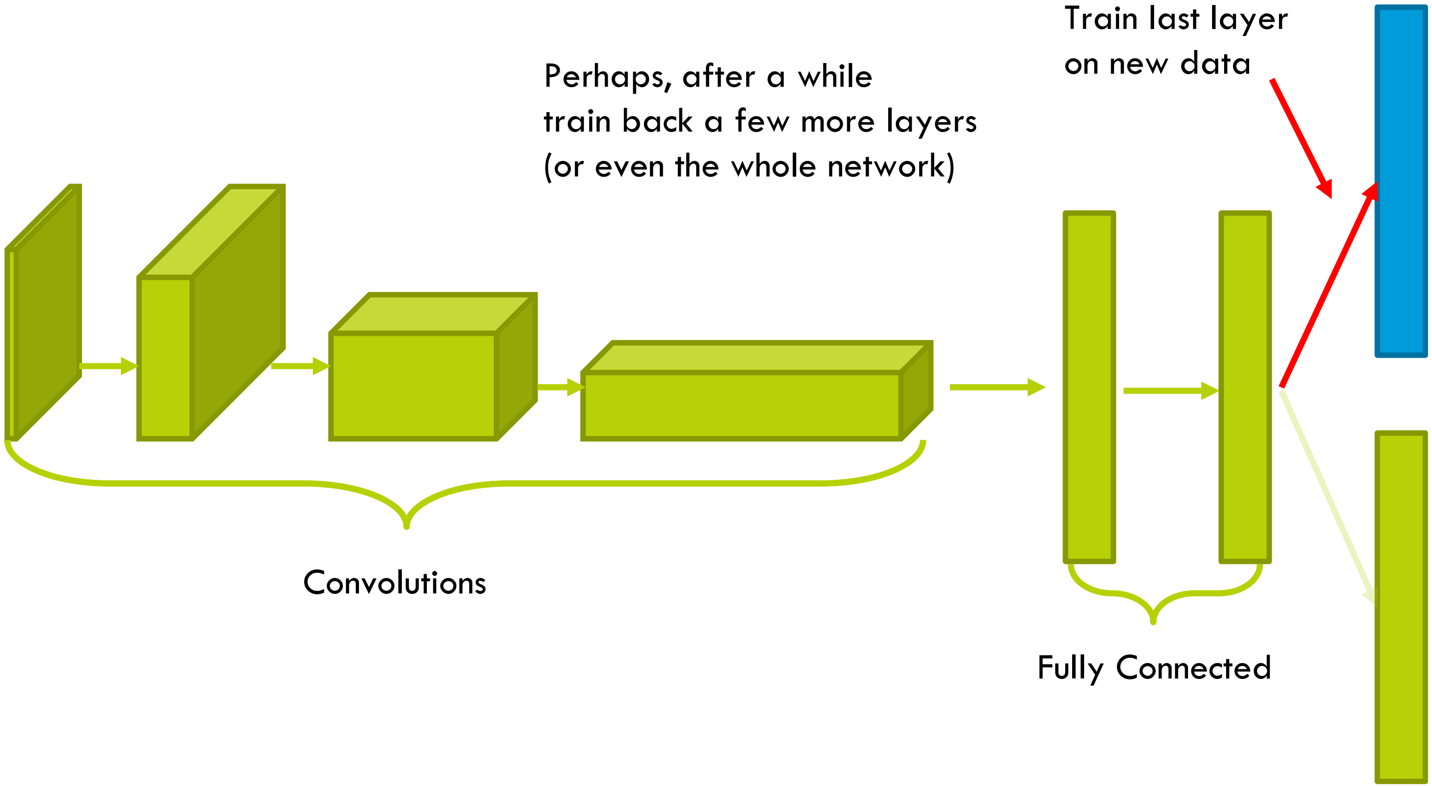
Transfer Learning



Transfer Learning



Transfer Learning



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)?

Guiding Principles for Fine-Tuning

- While there are no "hard and fast" rules, there are some guiding principles to keep in mind.

1) 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

2) 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.

Guiding Principles for Fine-Tuning

3) 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.