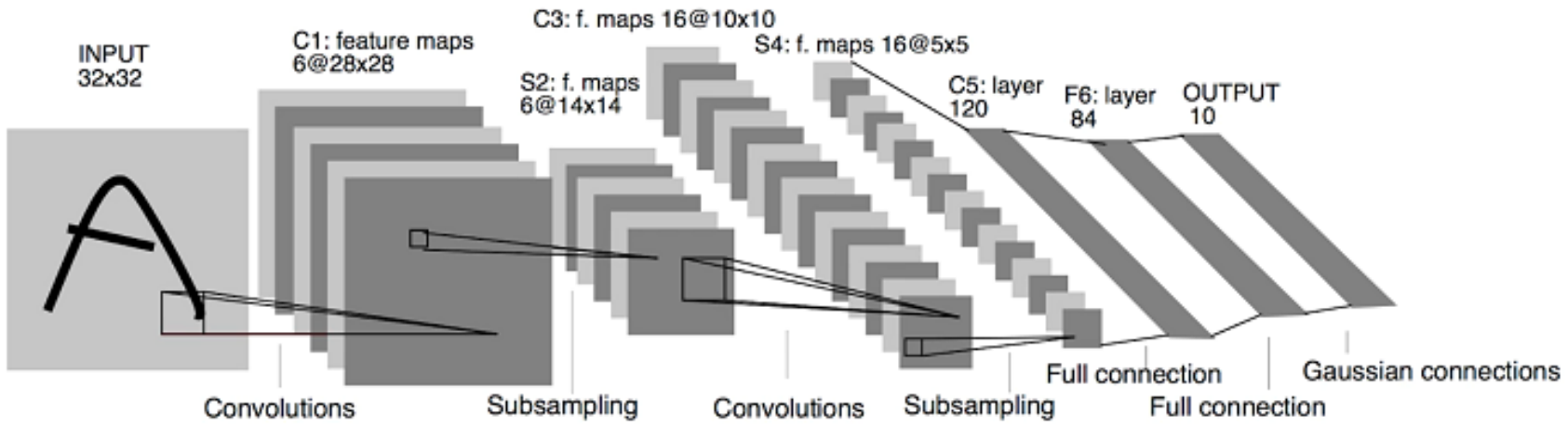
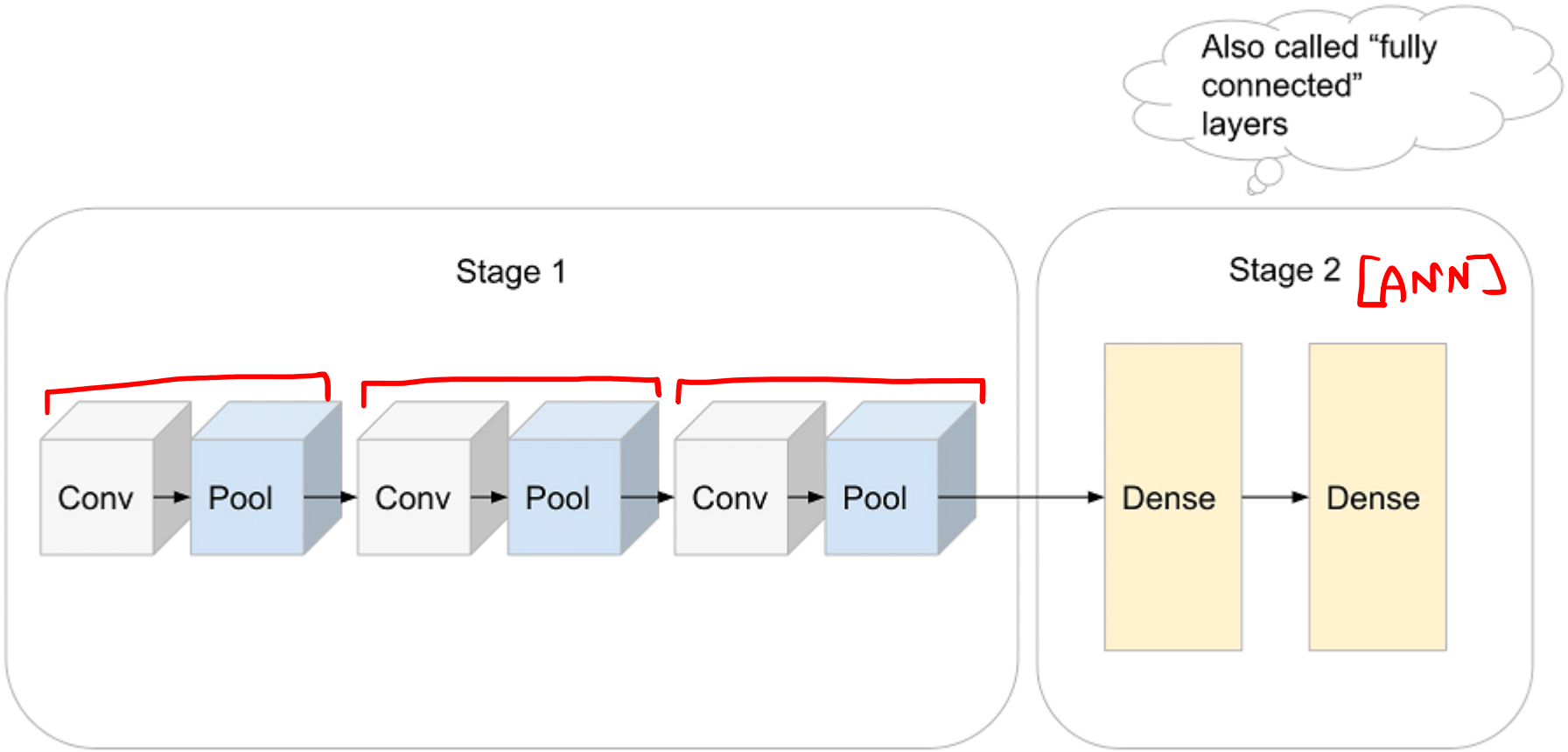
CNN Architecture

CNN Architecture

- Now that you understand how exactly a convolution layer works including the bias term and activation function we can now consider the architecture of a convolution or neural network and why it's that way.
- So as a little bit of a history lesson, modern CNN is essentially all originated from the same model, the LeNet.
- This is named after Yann LeCun, one of the original Deep Learning pioneers, along with Geoff Hinton and Yoshua Bengio.

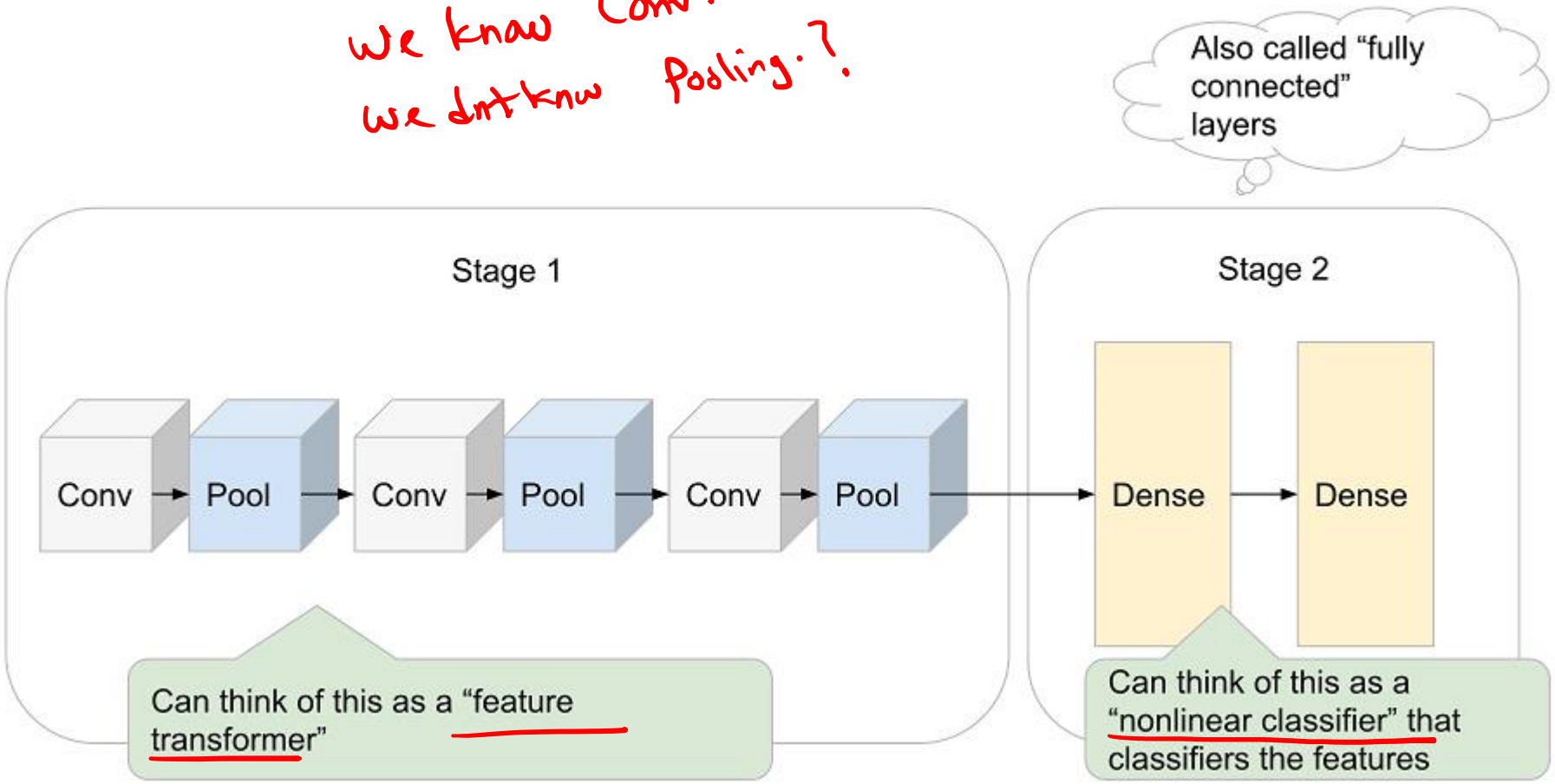


Typical CNN


Also called “fully connected” layers
Stage 1
Stage 2
ANN
Conv
Pool
Conv
Pool
Conv
Pool
Dense
Dense

Typical CNN

we know Conv. ✓
we dnt know Pooling.?



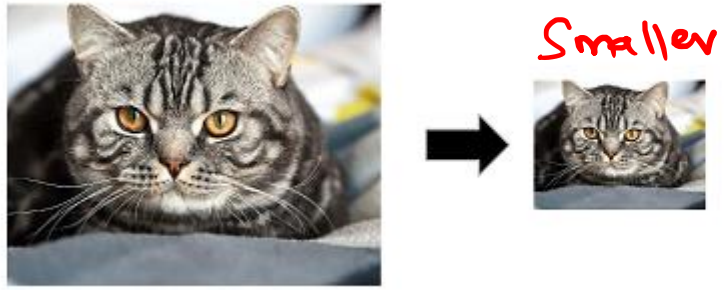
Pooling → Downsampling

If input image = 100 x 100

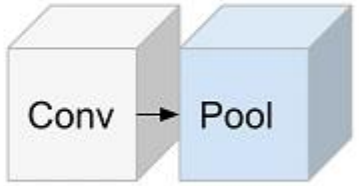
POOL SIZE = 2

output image = 50 x 50

Bigger Image



Smaller



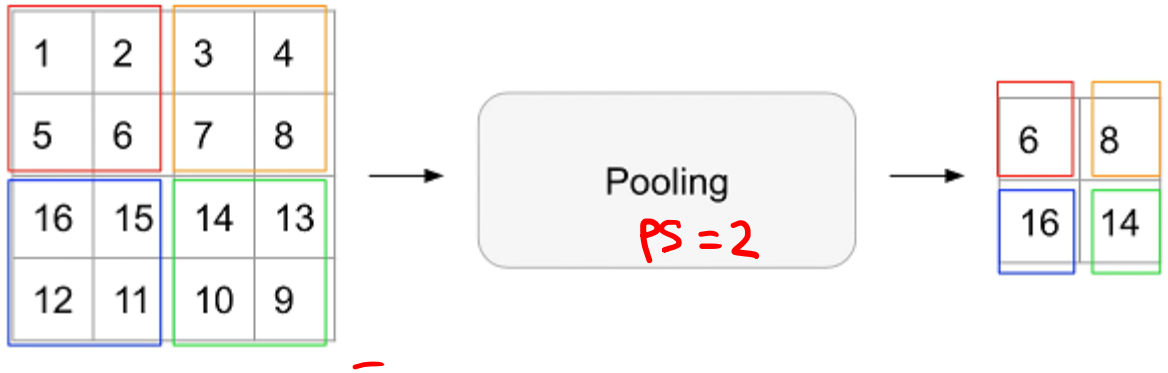
Types of Pooling

- There are two types of Pooling:

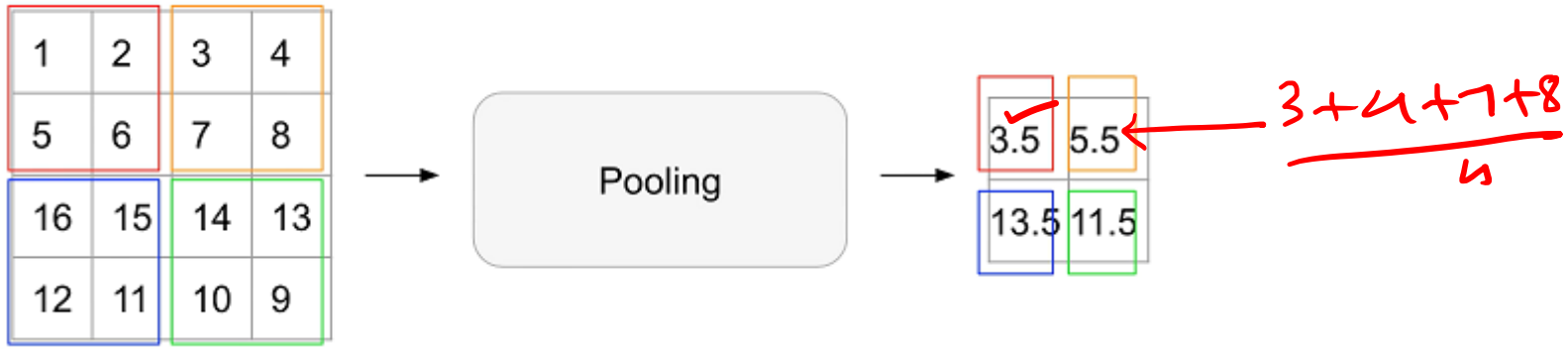
1. Max pooling
2. Average pooling

- Which one to use is a Hyperparameter choice.

Max Pooling



Average Pooling



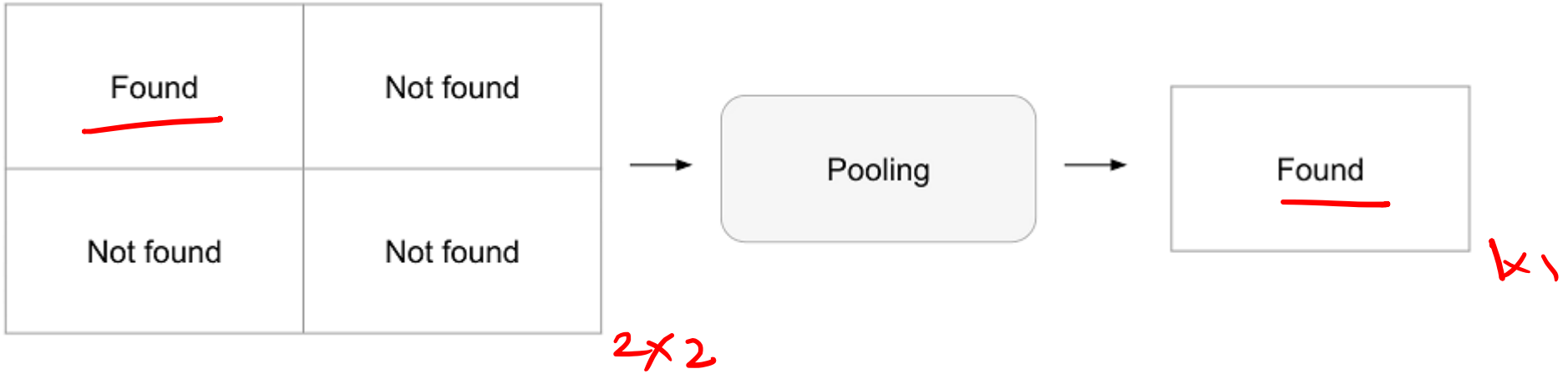
Why to use Pooling?

- Practical: If we shrink the image, we have less data to process.

- Translational Invariance: I don’t care where in the image the feature occurred, I just care that it did.

What is the Max Pooling doing?

- **Recall: Convolution is a "pattern finder" (the highest number is the best matching location)**

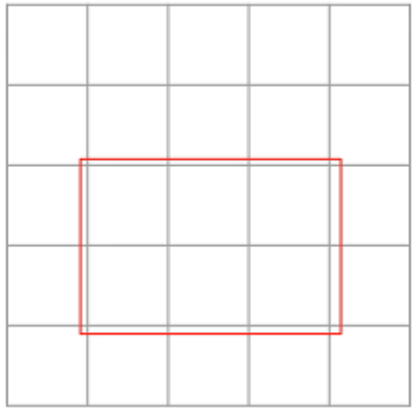


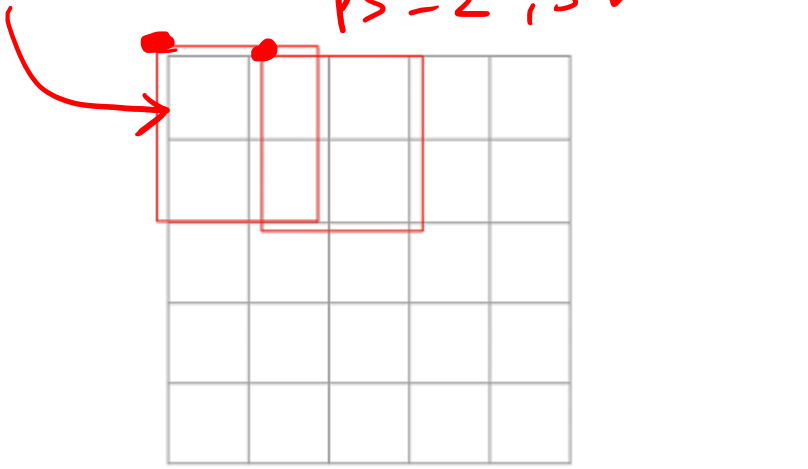
What about Average Pooling?

- Average pooling is the same idea but Max pooling is a little more intuitive in this regard.

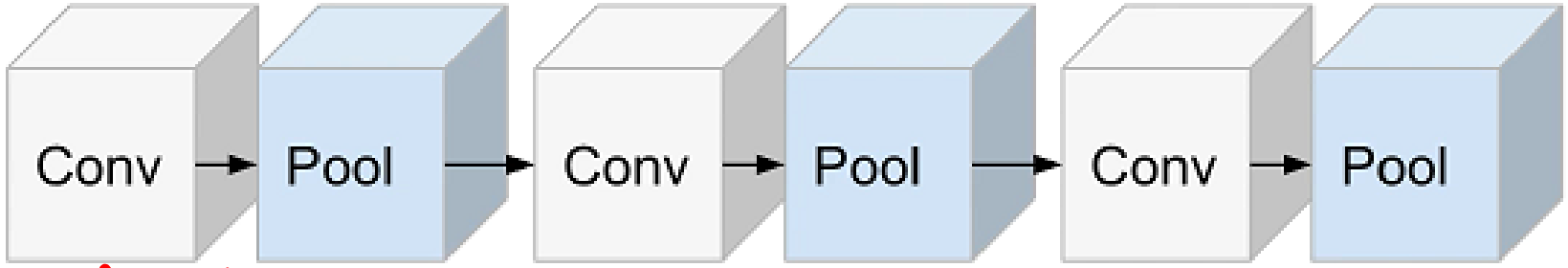
Different Pool Sizes

- It's possible to have a non-square window, e.g. 2x3 or 3x2, but this is unconventional
- It's also possible for boxes to overlap (this is called "stride")
  - Previously, we looked at a pool size of 2 with a stride of 2 (common)
  - If you had a stride of 1, the boxes would overlap (not common)

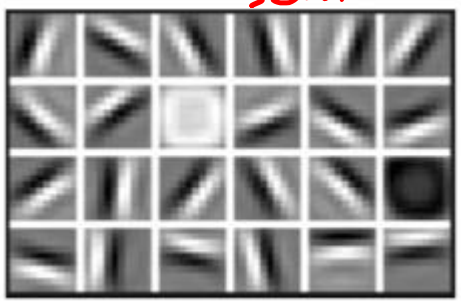




Why Convolution followed by pooling?



Amitabh Barchan SCM



First Layer Representation

crises, texture, mole, pimples

Second Layer Representation

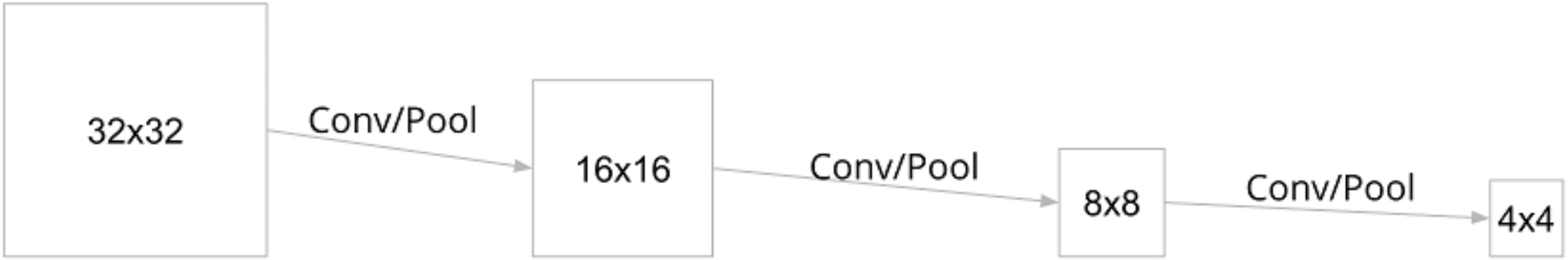
eye, nose, lips, ears

Third Layer Representation

faces

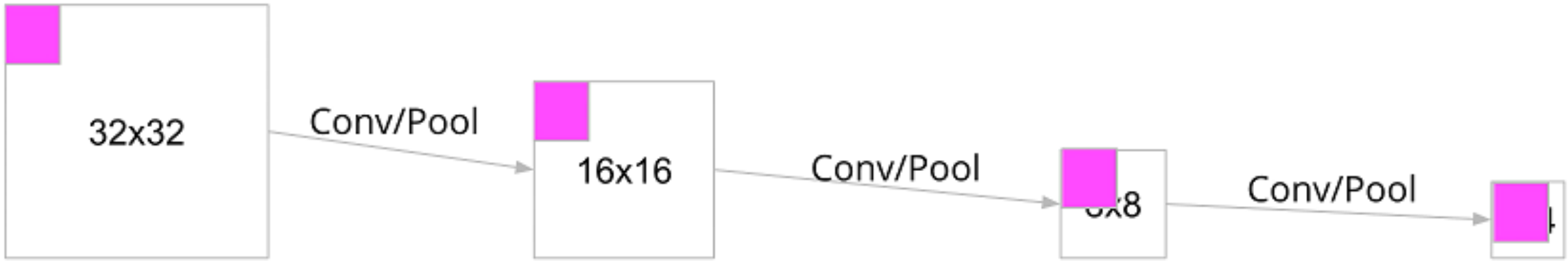
Why Convolution followed by pooling?

- Key point: after each “conv-pool”, the image shrinks, but filter sizes generally stay the same
- Common filter sizes are 3x3, 5x5, 7x7
- Assume “same mode” convolution and pool size = 2



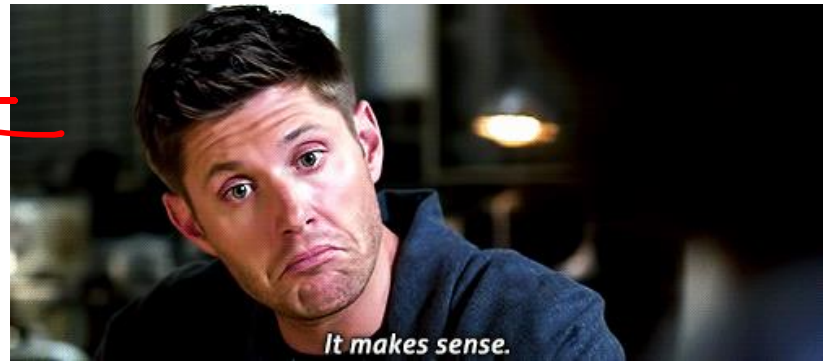
Why Convolution followed by pooling?

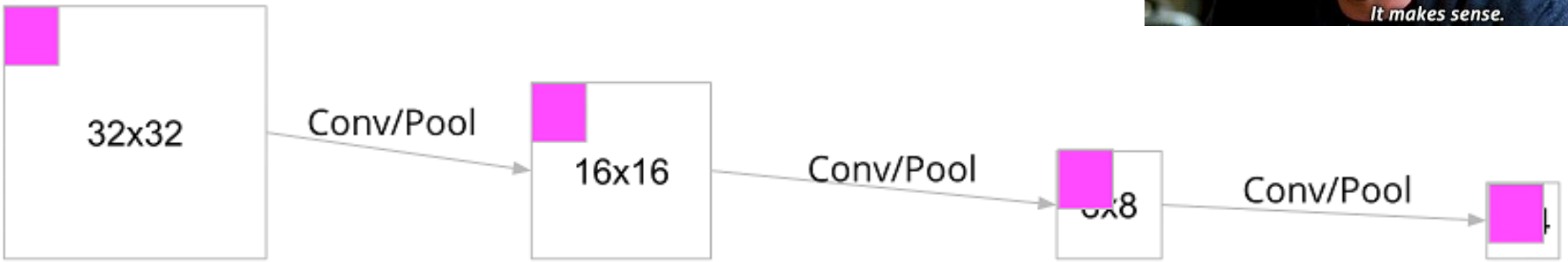
- If the filter size stays the same, but the image shrinks, then the portion of the image that the filter covers increases!



Why Convolution followed by pooling?

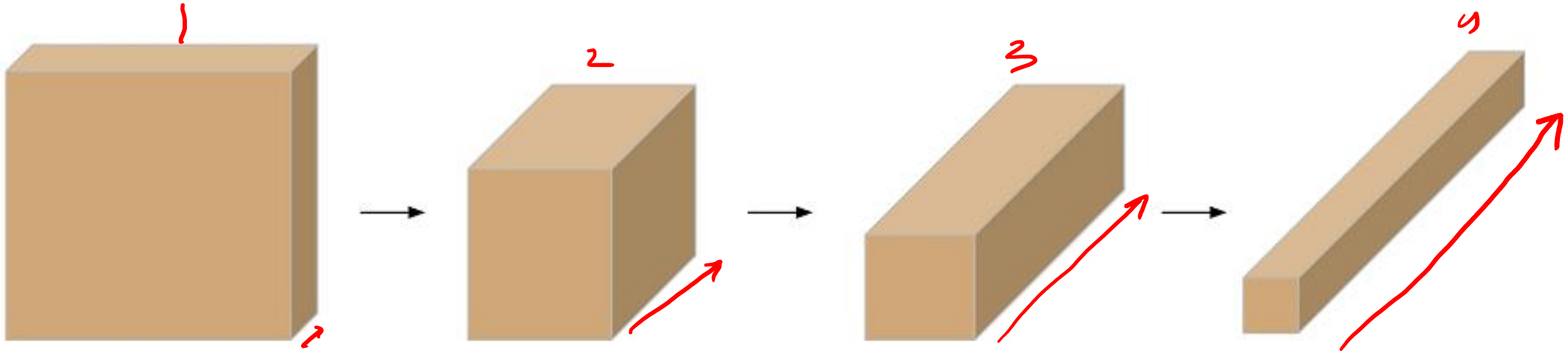
- The input image shrinks
- Since filters stay the same size, they find increasingly large patterns (relative to the image)
- This is why CNNs learn *hierarchical* features





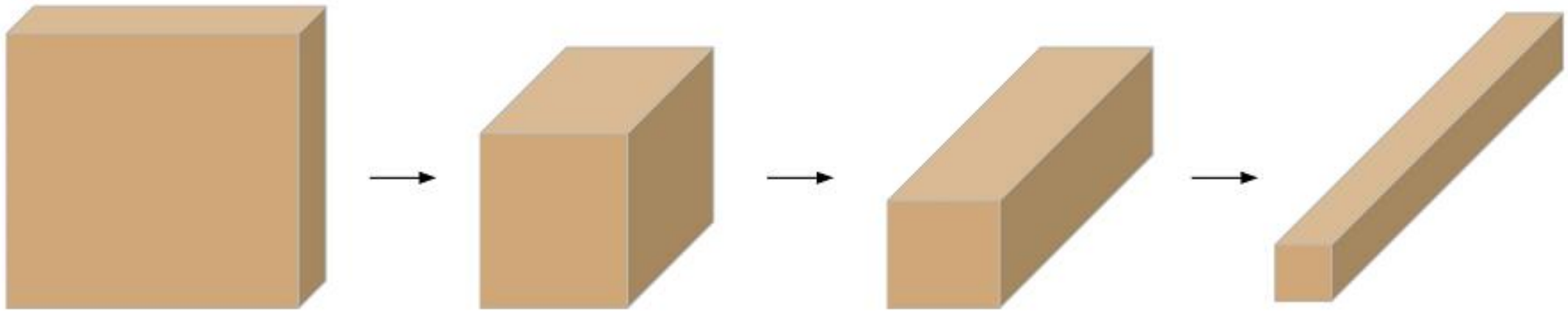
Losing Information

- Do we lose information if we shrink the image? Yes!
- We lose spatial information: we don't care *where* the feature was found
- We haven't yet considered the # of feature maps
- Generally, these increase at each layer
- So we *gain* information in terms of what features were found



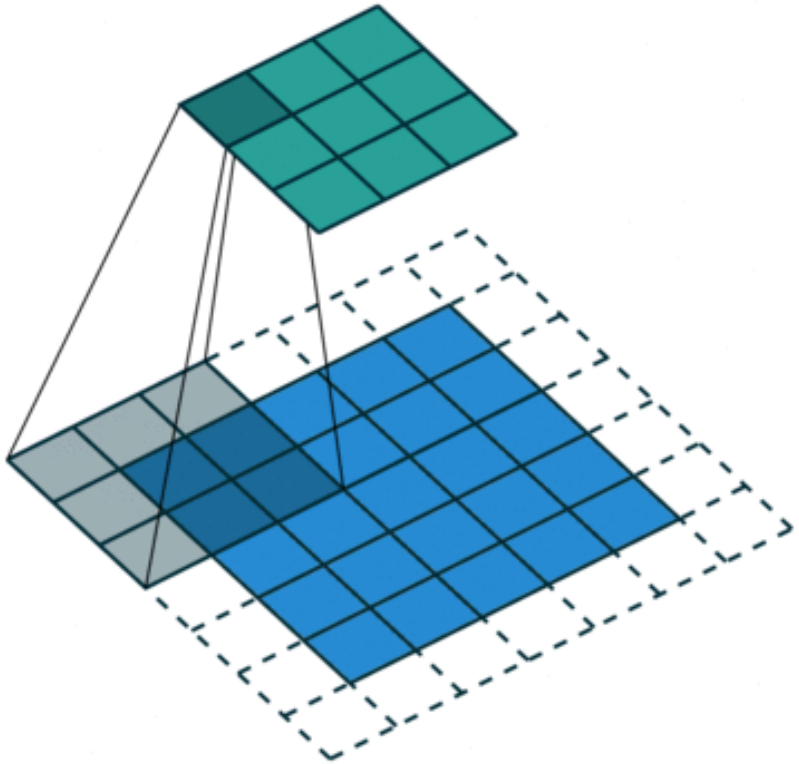
What Hyperparameters to choose?

- There are so many choices!
- Previously: learning rate, # hidden layers, # hidden units per layer
- With CNNs, the conventions are pretty standard
  - Small filters relative to image, e.g. 3x3, 5x5, 7x7
  - Repeat: convolution → pooling → convolution → pooling → etc.
  - Increase # feature maps, e.g.: 32 → 64 → 128 → 128
  - Read lots of papers!



Alternative to Pooling: Stride

Modern CNN has "stride" option

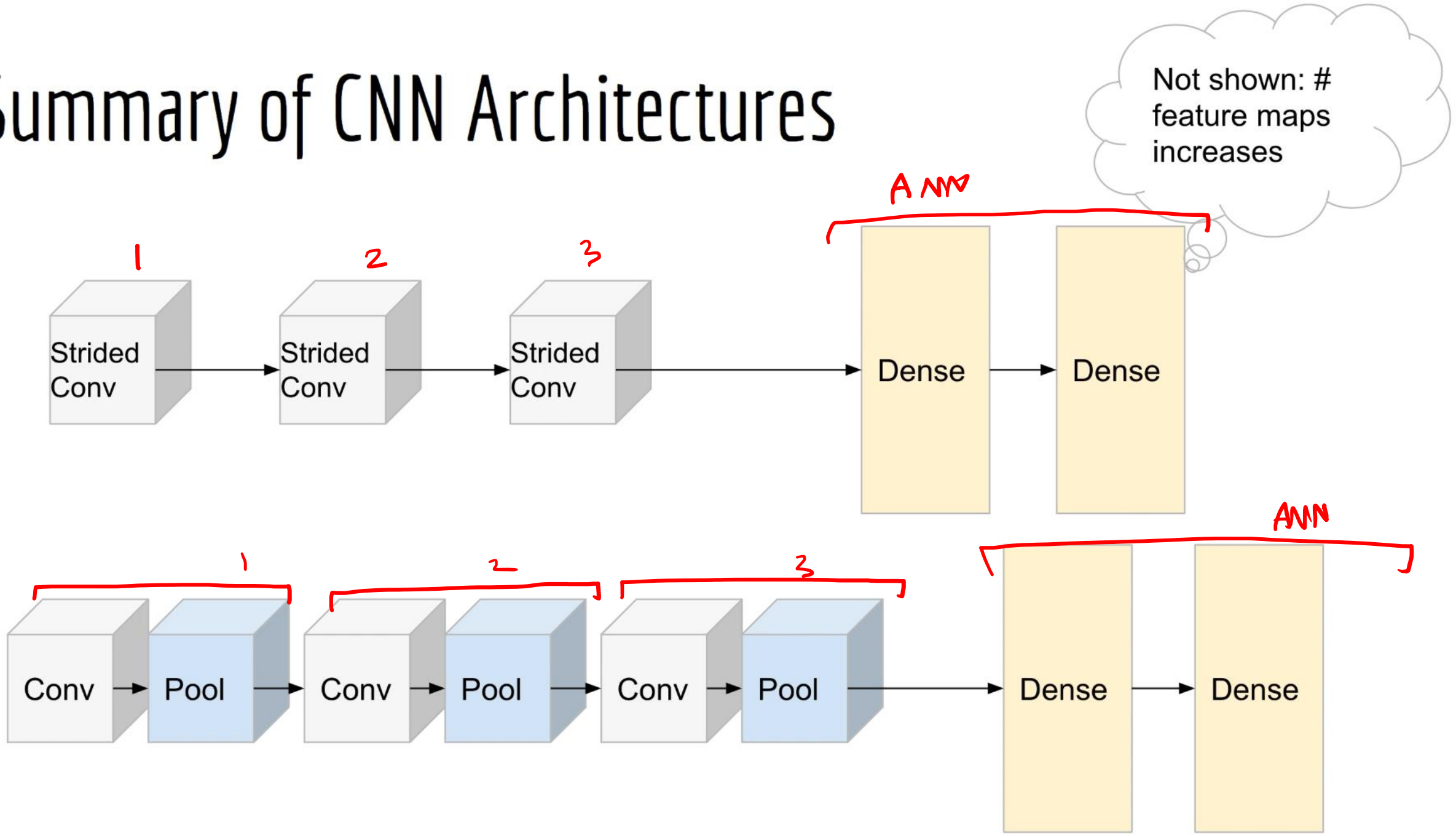


Why does it work?

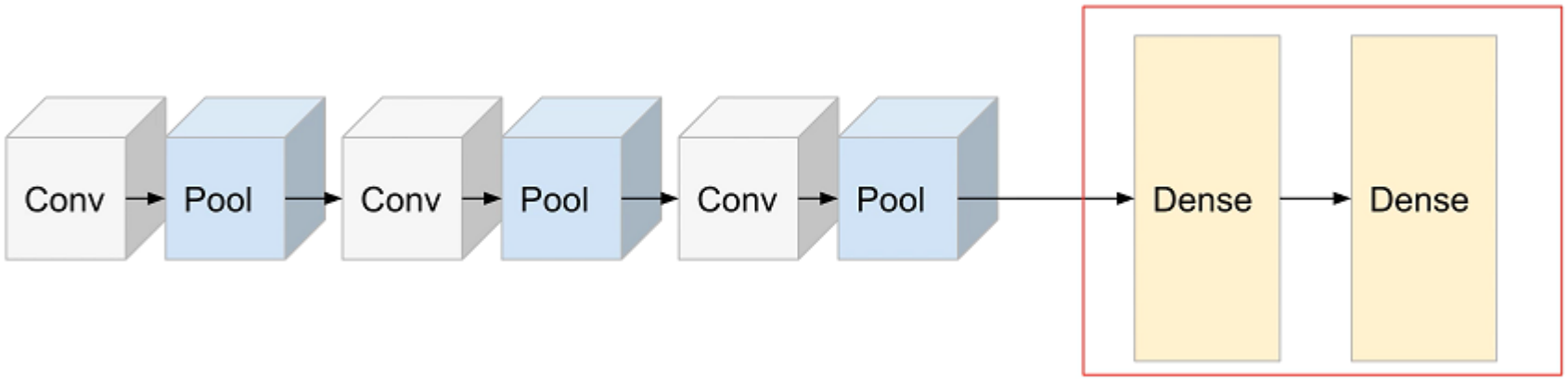
- An image is just large patches of “stuff”

- A red pixel’s neighbors (up / down / left / right) are *probably* also red

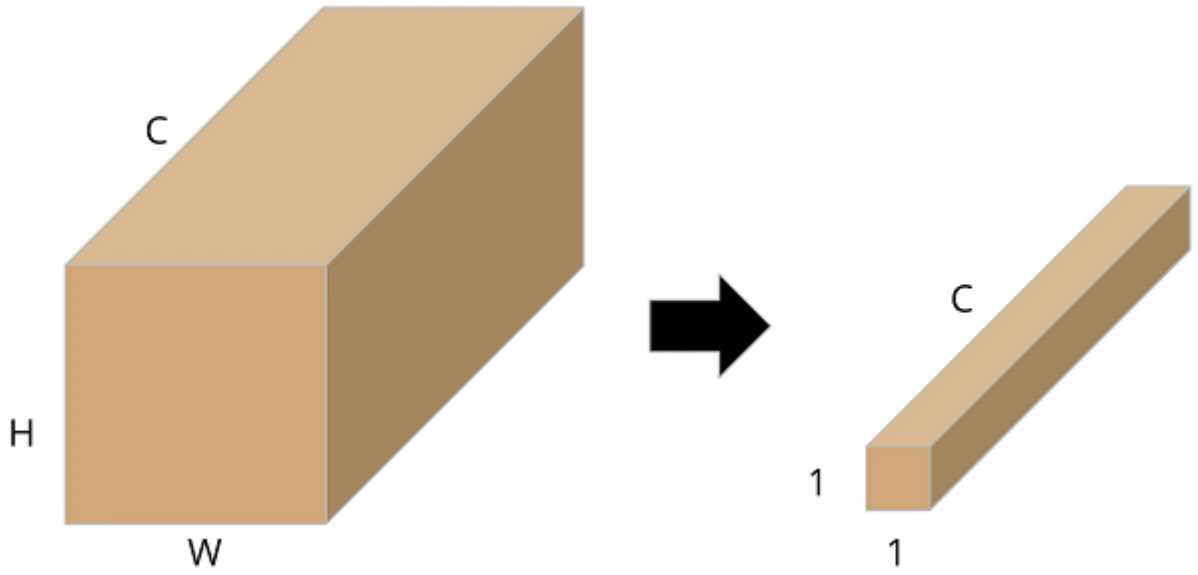
Summary of CNN Architectures



Dense Neural Network



Global Max Pooling



What's wrong with Flatten?

Case I:

Input = 32x32

4 Convolutions with stride = 2

32x32 > 16x16 > 8x8 > 4x4 > 2x2

Case II:

Input = 64x64

—11—

64x64 > 32x32 > 16x16 > 8x8 > 4x4

If final #feature maps = 100

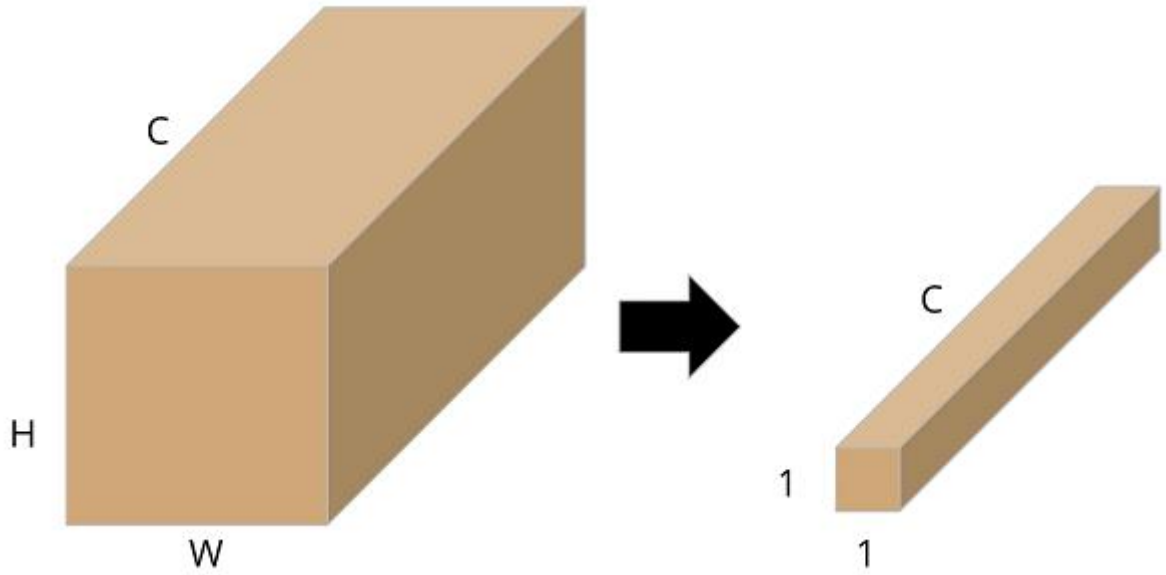
400D

1600D

NN cannot handle input vectors of different sizes.

Global Max Pooling

- We *always* get an output of 1x1xC (or just a vector of size C) regardless of H and W
- Takes the max over each feature map
- We don't care *where* the feature was found
- Also: global average pooling



Exception to CNN

- If you pass in images that are too small, you will get an error
- E.g. if you start with a 2x2 image, you cannot halve it 4 times
- You'll encounter this if you try to add too many conv layers to your CNN

Summary

Step 1: C > P > C > P - - -

or

S.C. > S.C. > - - - ..

Step 2: Flatten()

or

Global Max Pooling 2D()

Step 3: Dense > Dense > - - .

CNN Code Preparation

Step1: Load in data

Dataset: Fashion MNIST and CIFAR-10

60000 x 28 x 28 Grayscale

Different types of clothing

CIFAR-10

32 x 32 x 3

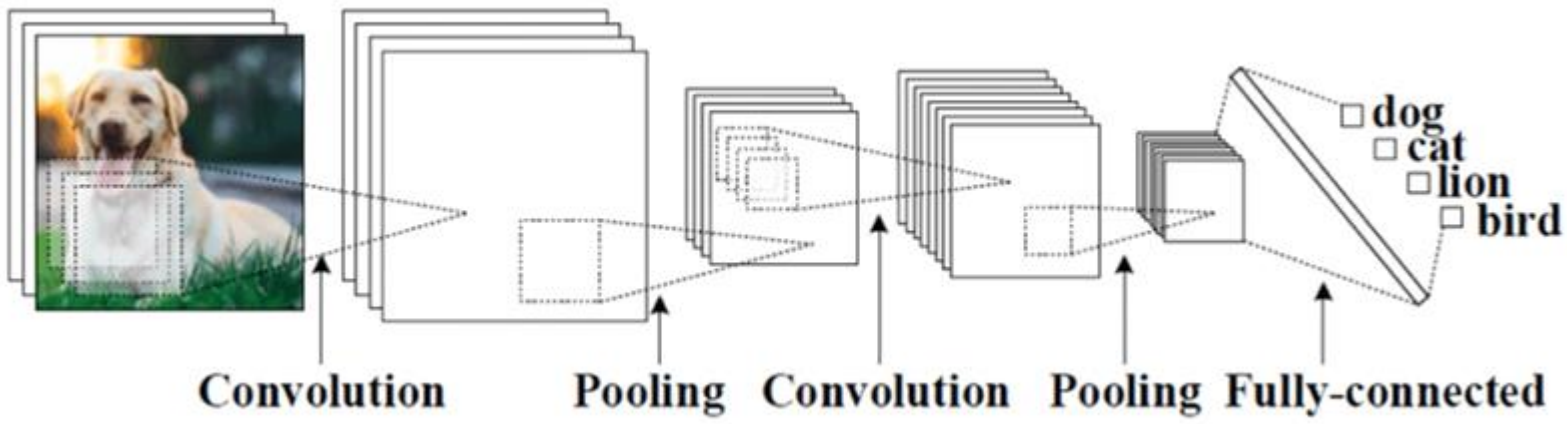
Classes: Horse | Cat, Dog, etc.

CNN Code Preparation

Step 2: Build the model

- CNN: just an ANN with Conv layers
- Functional API: a better way of creating models

→ makes code cleaner
→ more compact



CNN Code Preparation

Step #3: Train the model

Step #4: Evaluate the model

Step #5: Make predictions

Fashion MNIST

Drop-in replacement for MNIST, exact same format

X.shape = N x 28 x 28 (grayscale)

Pixel values 0...255

Not the right shape for CNN! CNN expects N x H x W x C

We must reshape to N x 28 x 28 x 1

CIFAR-10

model.fit(X, y)

X = 2D (100, 28)

y = 1D (100,)

Data is N x 32 x 32 x 3, pixel values 0...255

Slight inconvenience: labels are N x 1

Just call flatten() to fix it

Build the model

We will use the Keras Functional API

"What's the difference between Keras the library and Keras the module inside the Tensorflow library?"

Keras is an API specification

Anyone can make their own deep learning library (that doesn't depend on Theano, Tensorflow, PyTorch, etc.) and wrap it with the Keras API

Someone could then use Keras with *your* library as a backend

Creator of Keras went on to work at Google, so it's "closer" to Tensorflow

Keras the *library* is installed separately from its backend

Functional API

- Easiest to learn by example

```
model = Sequential([
  Input(shape=(D,)),
  Dense(128, activation='relu'),
  Dense(K, activation='softmax'),
])

...
model.fit(...)
model.predict(...)
```

```
i = Input(shape=(D,))
x = Dense(128, activation='relu')(i)
x = Dense(K, activation='softmax')(x)

model = Model(i, x)

...
model.fit(...)
model.predict(...)
```

Functional API

Isn't this "bad style"?

Important rule of software engineering: conform to the same style guide as your team

E.g. don't use 4-space indent if everyone else uses 2-space

Don't use tab if they use space

Esp. for math, less cognitive load to use "one-letter variable names"

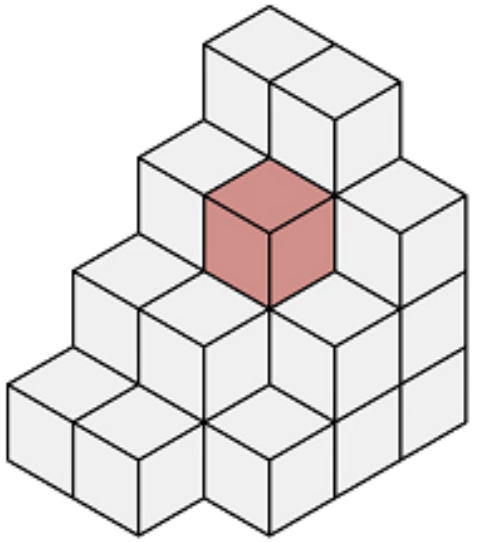
No need to *translate*

In any case, stick to conventions!

```
i = Input(shape=(D,))
x = Dense(128, activation='relu')(i)
x = Dense(K, activation='softmax')(x)

model = Model(i, x)

...
model.fit(...)
model.predict(...)
```

CNN using Functional API

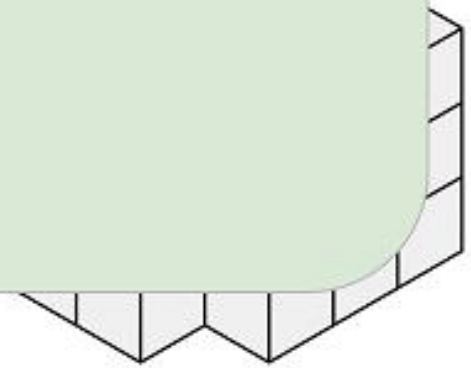
```
i = Input(shape=x_train[0].shape)
x = Conv2D(32, (3, 3), strides=2, activation='relu')(i)
x = Conv2D(64, (3, 3), strides=2, activation='relu')(x)
x = Conv2D(128, (3, 3), strides=2, activation='relu')(x)
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
x = Dense(K, activation='softmax')(x)

model = Model(i, x)
```



CNN using Functional API

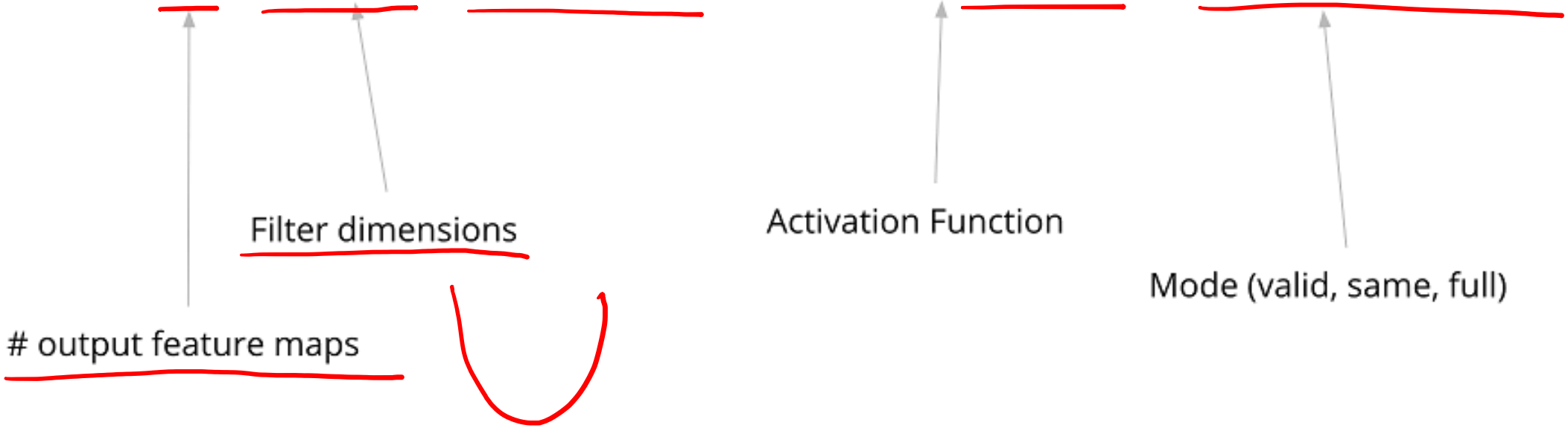
```
i = Input(shape=x_train[0].shape)
x = Conv2D(32,
x = Conv2D(64
x = Conv2D(12
x = Flatten()
x = Dense(512
x = Dense(K,

model = Model
```

- Note: it's Conv2D because there are 2 spatial dimensions
- Also have Conv1D and Conv3D
- A time-varying signal would use Conv1D
- A video (Height, Width, Time) would use Conv3D
- Voxels (Height, Width, Depth) would use Conv3D
- E.g. medical imaging data
- Pixel = "Picture Element"
- Voxel = "Volume Element"



Conv2D Arguments

```
Conv2D(32, (3, 3), strides=2, activation='relu', padding='same')
```



CNN using Functional API

```
i = Input(shape=x_train[0].shape)
x = Conv2D(32, (3, 3), strides=2, activation='relu')(i)
x = Conv2D(64, (3, 3), strides=2, activation='relu')(x)
x = Conv2D(128, (3, 3), strides=2, activation='relu')(x)
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
x = Dense(K, activation='softmax')(x)

model = Model(i, x)
```

Dropout for Convolution

```
x = Conv2D(32, (3, 3), strides=2, activation='relu')(i)
x = Dropout(0.2)(x)
x = Conv2D(64, (3, 3), strides=2, activation='relu')(x)
x = Dropout(0.2)(x)
x = Conv2D(128, (3, 3), strides=2, activation='relu')(x)
x = Dropout(0.2)(x)
```