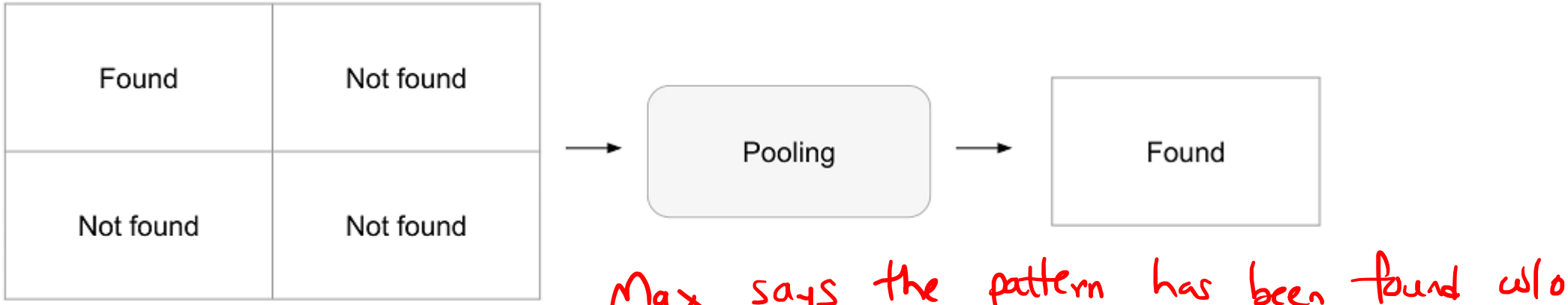
What is the Max Pooling doing?

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

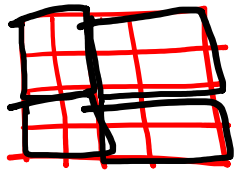


Max says the pattern has been found w/o carrying info where it was found.

What about Average Pooling?

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

Different Pool Sizes

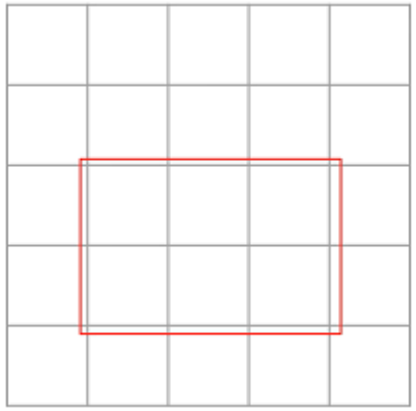


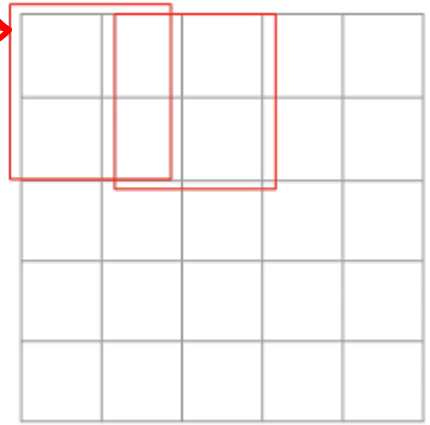
2x2 (SQUARE)

stride = 2

- 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)

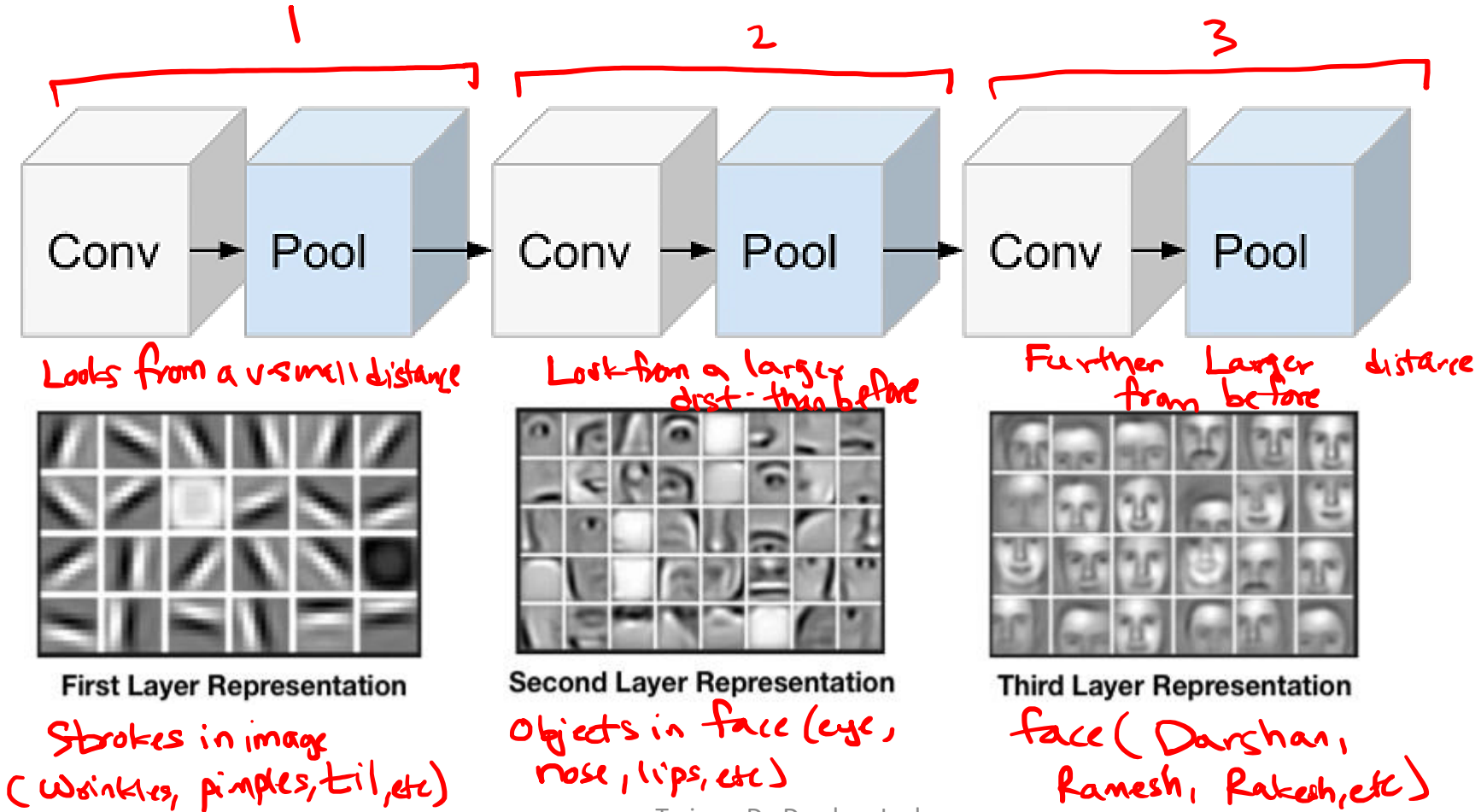
stride/slide by which u wnt to move the pool





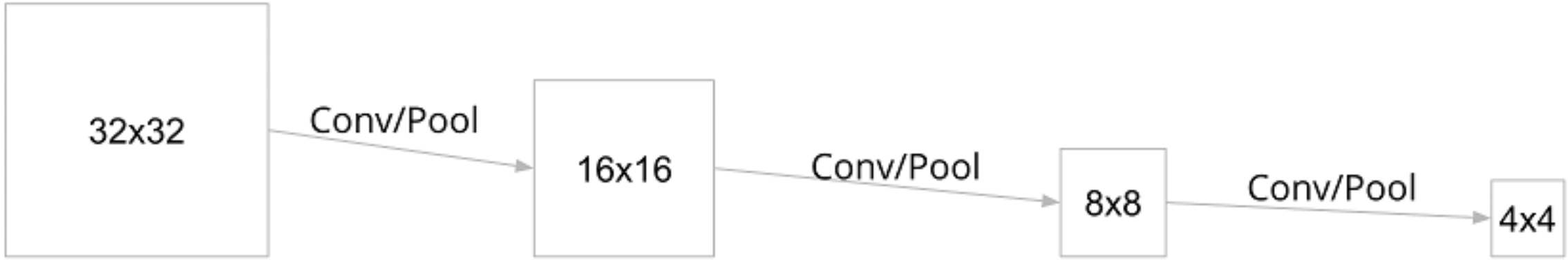
stride = 1

Why Convolution followed by pooling?



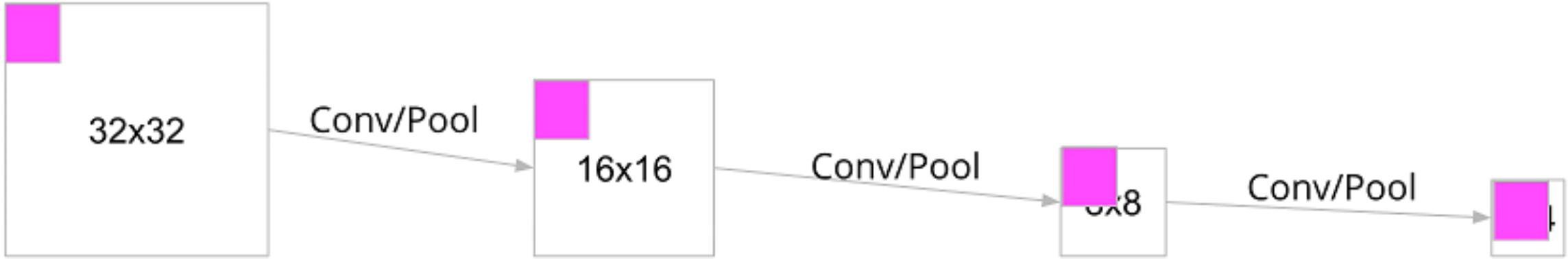
Why Convolution followed by pooling?

- 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



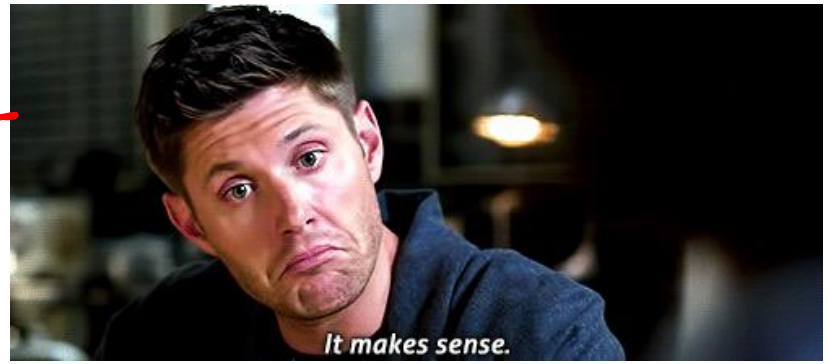
Why Convolution followed by pooling?

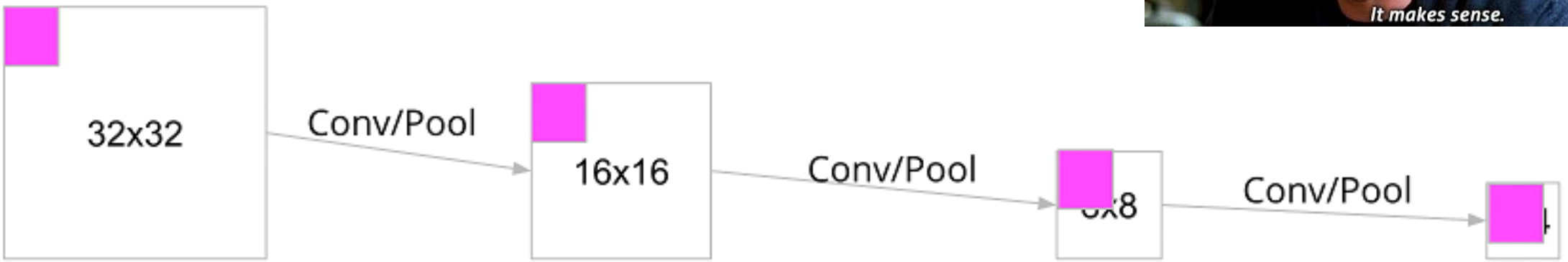
- If the filter size stays the same, but the image shrinks, then the portion of the image that the filter covers increases!



Why Convolution followed by pooling?

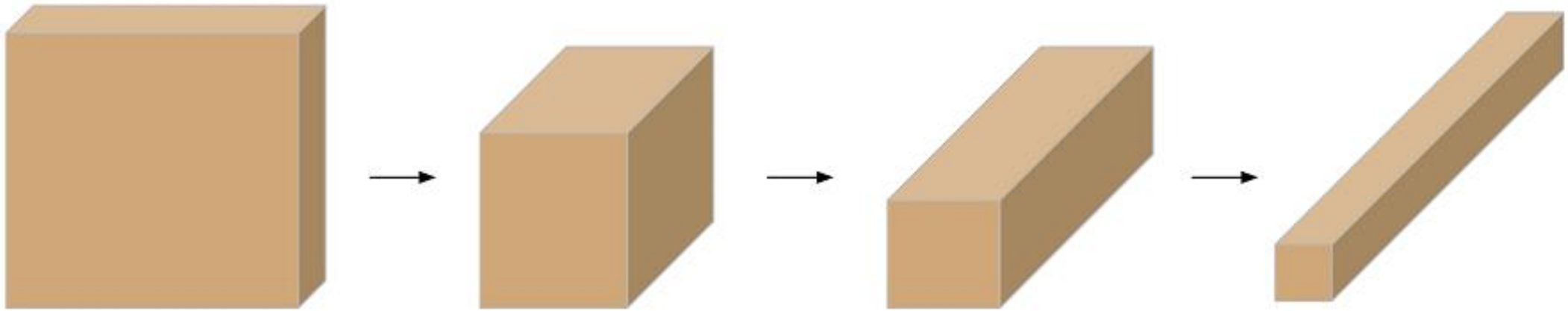
- The input image shrinks
- Since filters stay the same size, they find increasingly large patterns (relative to the image)
- This is why CNNs learn *hierarchical* features





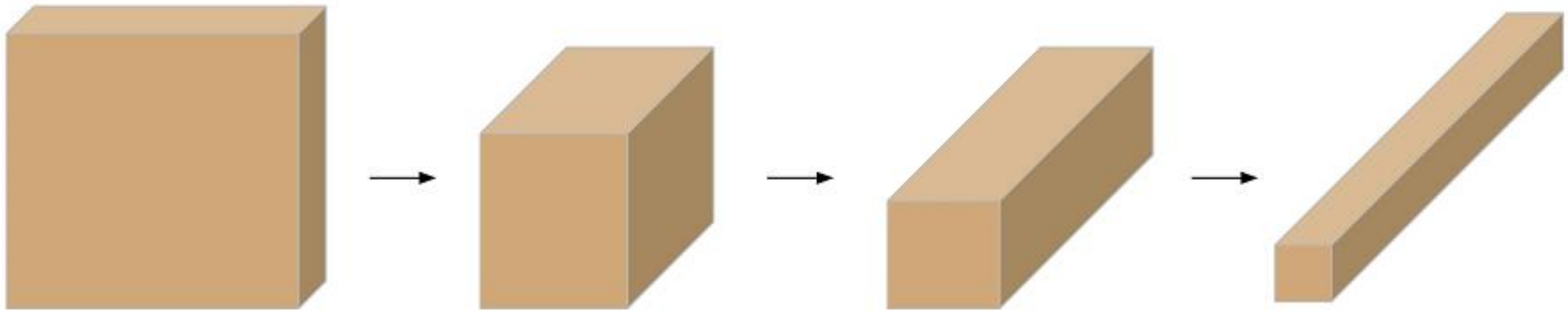
Losing Information

- 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



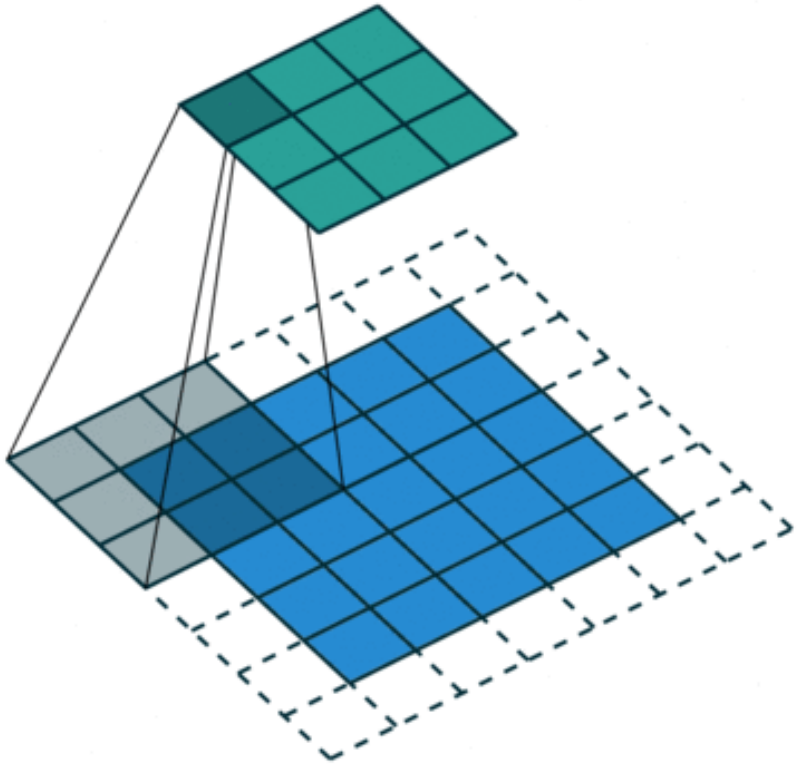
What Hyperparameters to choose?

- 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! (Research Papers)



Alternative to Pooling: Stride

Conv + Pool ⟹ Conv. func. also has Stride option.

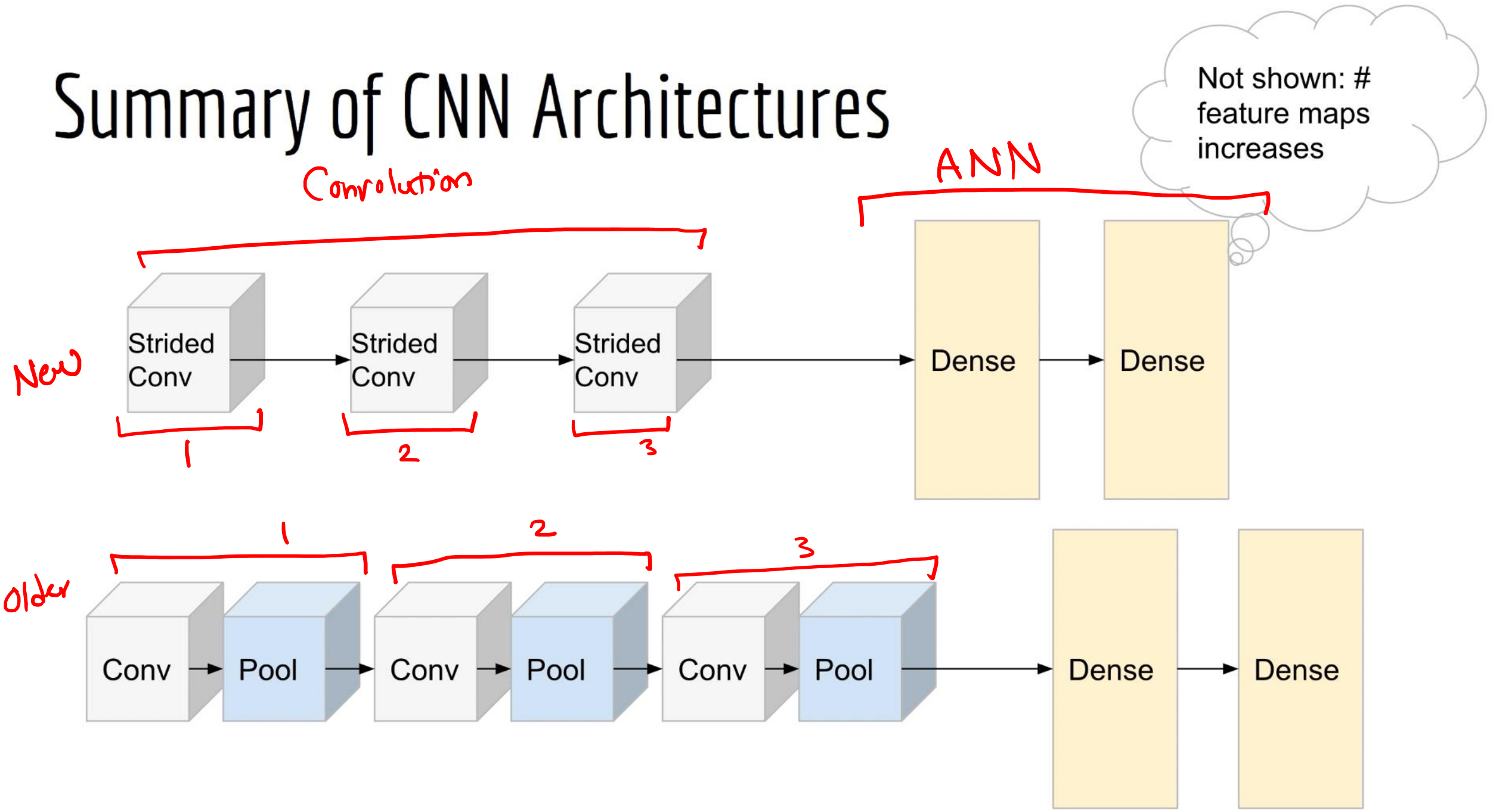


Why does it work?

- 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



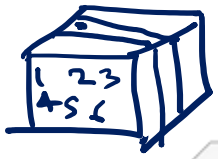
Dense Neural Network

when an image comes out of convolution, it will be a 3D
i.e. H x W x #featuremaps

while NN Dense layer expects a 1D array as input
shape shud be appropriate.

∴ We go for flatten()

3D → 1D



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Feed Fwd NN

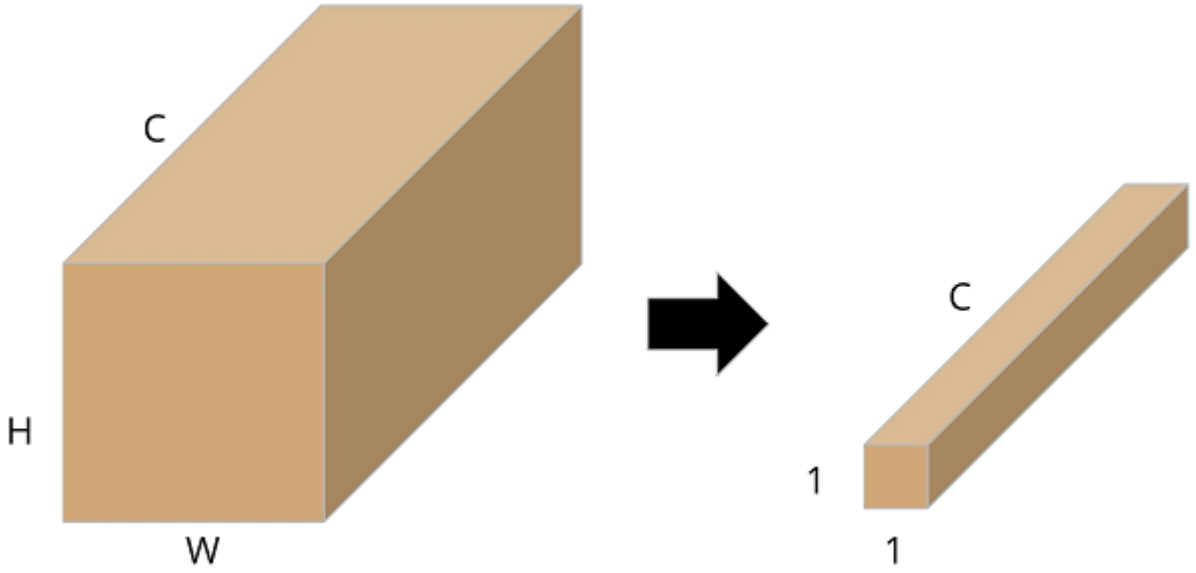
Conv → Pool → Conv → Pool → Conv → Pool → 3D → 1D → Dense → Dense

Global Max Pooling : It is alternative to flatten()

Internet : Each image is of different size.

Is CNN capable of handling different sized images ?

Answer: Yes. How? Using Global Max Pooling



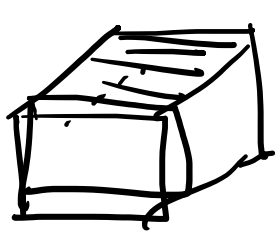
What's wrong with Flatten?

① Suppose the input image size = 32 × 32
We do 4 convolutions with stride = 2

$32\times32 \xrightarrow[\text{Conv.}]{\text{Strided}} 16\times16 \xrightarrow{S.C.} 8\times8 \xrightarrow{S.C.} 4\times4 \xrightarrow{S.C.} 2\times2$

② Suppose the input image size = 64 × 64
— " — 4 — " — stride = 2

$64\times64 \xrightarrow{S.C.} 32\times32 \xrightarrow{S.C.} 16\times16 \xrightarrow{S.C.} 8\times8 \xrightarrow{S.C.} 4\times4$



If final #feature maps = 100

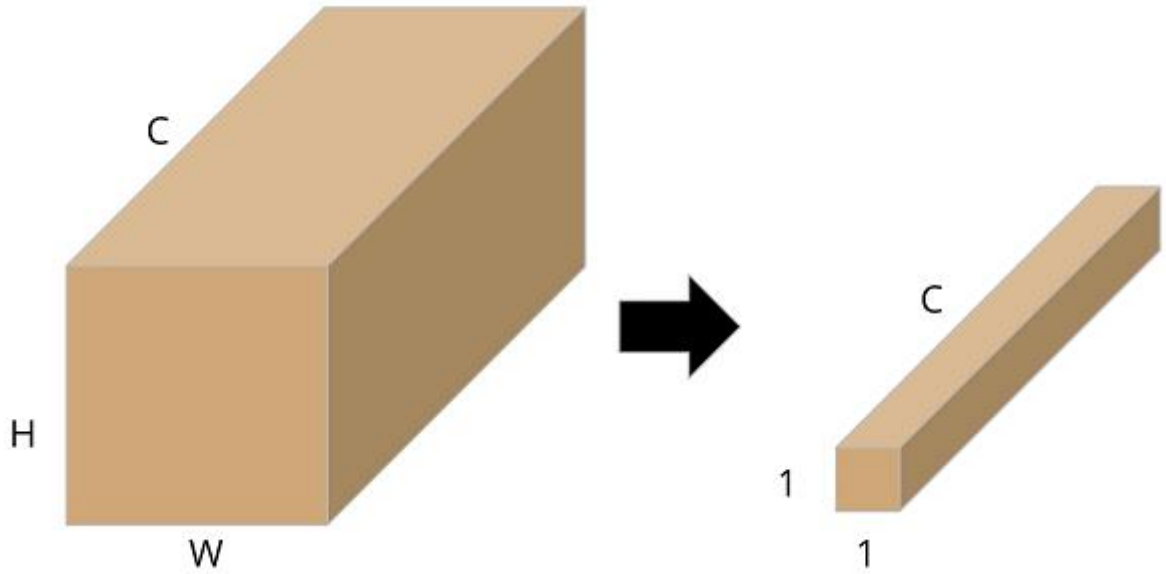
① for my 32×32 image = 400 D vector $\xrightarrow{\text{flatten()}}$ (400, 1)

② for my 64×64 image = 1600 D vector $\xrightarrow{\text{flatten()}}$ (1600, 1)

∴ NN cannot handle i/p vectors of different size.

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:
- Conv > Pool > Conv > Pool > ....
- Strided Conv > Strided conv. > ....

Step 2:
- flatten()
- GlobalMaxPooling2D()

Step 3: Dense > Dense > ....

3 tasks that determine final activation/# of nodes!

1. Regression : None
2. Binary Classification : sigmoid
3. Multi-class Class : softmax

CNN Code Preparation

Step 1: Load the data

Fashion MNIST (difficult than MNIST)

It consists of 60000 images

28 × 28 grayscale

Images are of different types of clothing

eg: tshirts, shoes, pants, etc.

Task: Classify (10 category)

CIFAR-10 is old, but it is more difficult than Fashion MNIST

- Contains color images : 32×32×3 (3 is for color)

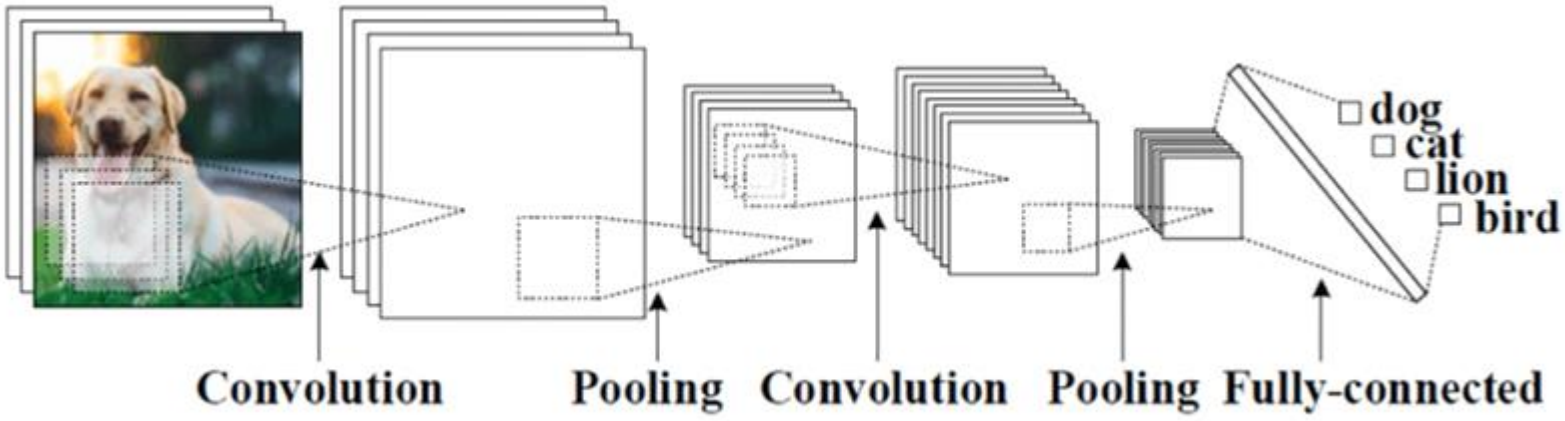
Images: Autombiles, Frog, horse, dog, cat, etc.

Task: Classification

CNN Code Preparation

Step 2: Build the model

- CNN: just an ANN with Conv layers
- Functional API: a better way of creating models (cleaner & compact)



CNN Code Preparation

Step #3: Train the model

Step #4: Evaluate the model

Step #5: Make predictions

Fashion MNIST

N=60,000

Drop-in replacement for MNIST, exact same format

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

N x H x W x ?

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

N x H x W x C

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 → (N,)

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"

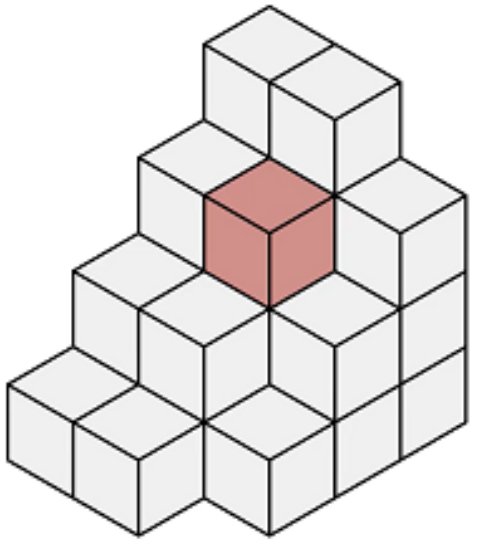
No need to *translate*

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

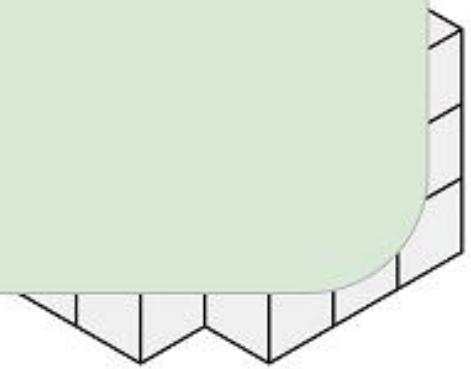
```
i = Input(shape=x_train[0].shape)
x = Conv2D(32, (3, 3), strides=2, activation='relu')(i)
x = Conv2D(64, (3, 3), strides=2, activation='relu')(x)
x = Conv2D(128, (3, 3), strides=2, activation='relu')(x)
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
x = Dense(K, activation='softmax')(x)

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)
```