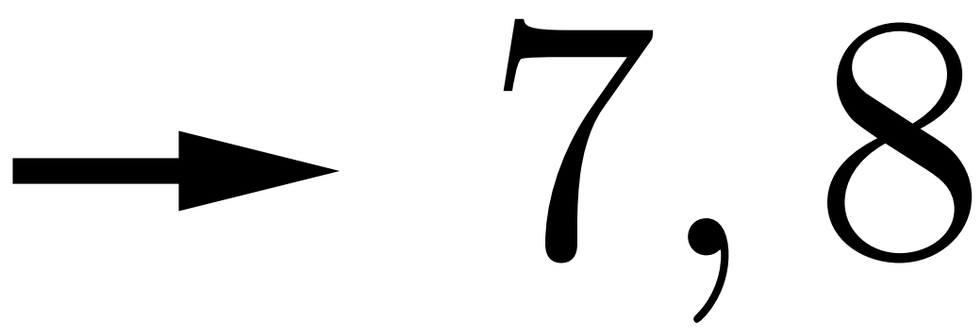
Convolutions

Recap

MNIST



28 x 28

Input

Digit from [0, 9]

Output

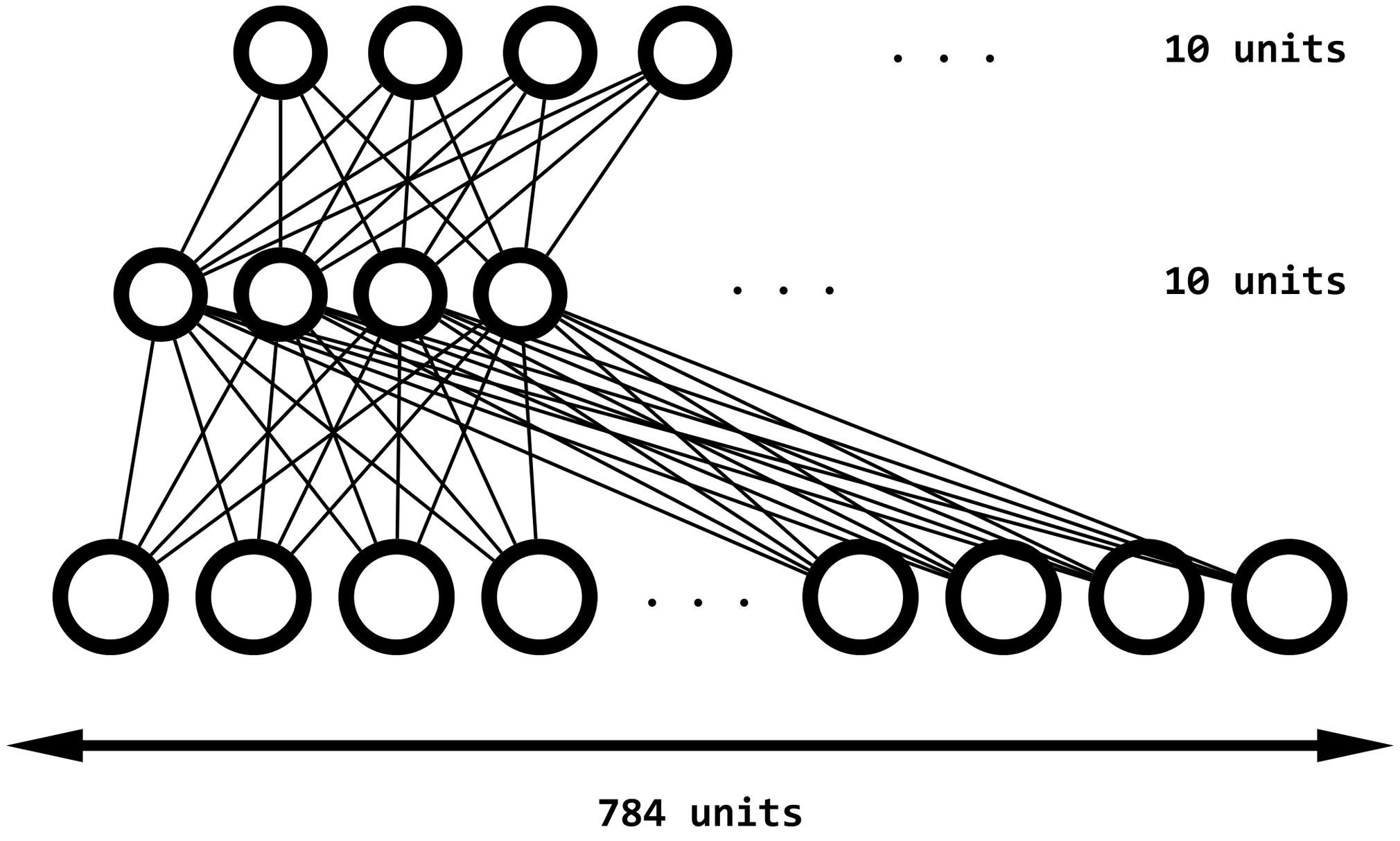
Our Approach

1. Flatten image -> 784 dimensional vector
2. Transform to 10 dim vector
    1. Linear Transformation
    2. ReLU activation
3. Transform to another 10 dim vector
4. Softmax
5. Compare to one-hot labels with cross-entropy
6. SGD to discover the right transformations

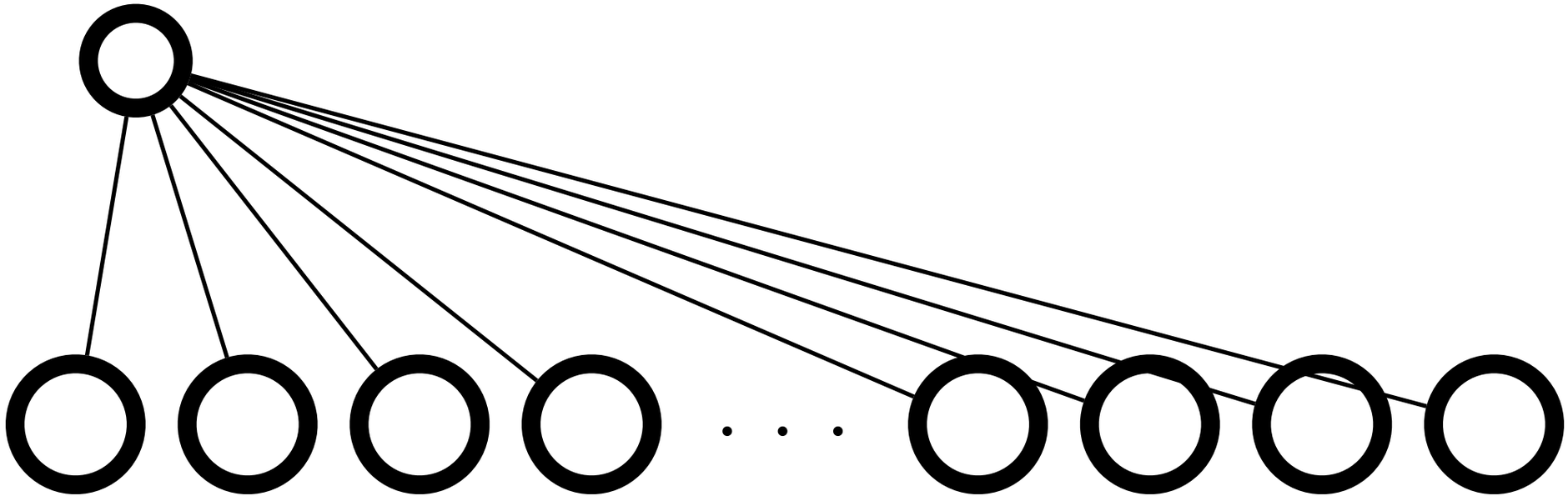
Keras Code

```
model = Sequential()
model.add(Dense(10, input_dim=784)) # INTRODUCE NON-LINEARITY
model.add(Activation('relu'))
model.add(Dense(10))

# linear separator bits
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])

model.fit(train_images, to_categorical(train_labels), epochs=20)
```

(Partial) Diagram



A Single Hidden Unit



Observations

- Each hidden layer unit:
  - **Uses each unit from previous layer**

AKA

Fully Connected Layer

Parameters

- Need to learn:
  - Hidden Layer 1:
    - Linear transformation - 784 x 10 matrix
    - Bias term - 10 (1 per unit)
  - Hidden Layer 2:
    - Linear transformation - 10 x 10 matrix
    - Bias term - 10 (1 per unit)

Parameters

$$7840 + 10 + 100 + 10$$

$$=$$

$$7960$$

We Learned That

more units

=

more representational power

Consider

- **200 x 200** Image
- **1** hidden layer
- **40,000 units** in hidden layer

Parameters

- 200 x 200 Flattened - 40,000 input units
- Linear transform to 40,000 hidden units:
  - 40,000 * 40,000 = 1.6e9

Problem

1 Billion Parameters

Primate PreFrontal Cortex

25 billion neurons

Our model is pretty

pathetic

in comparison

Need To

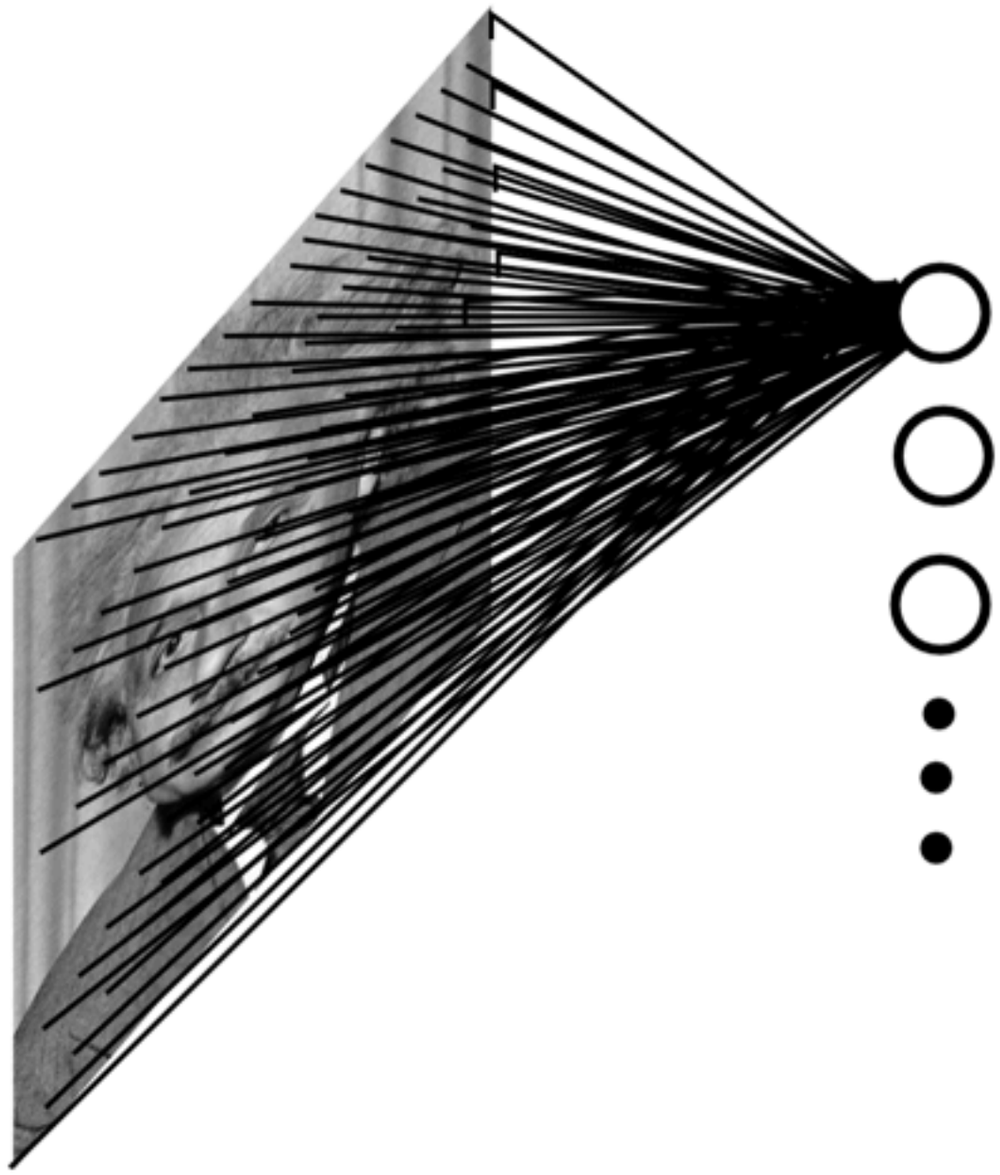
Cut

Parameters

Observations

- Each **hidden unit** considers **all input units**

The Hidden Unit



Is This Necessary?

- Let us swap a couple of rows in each image

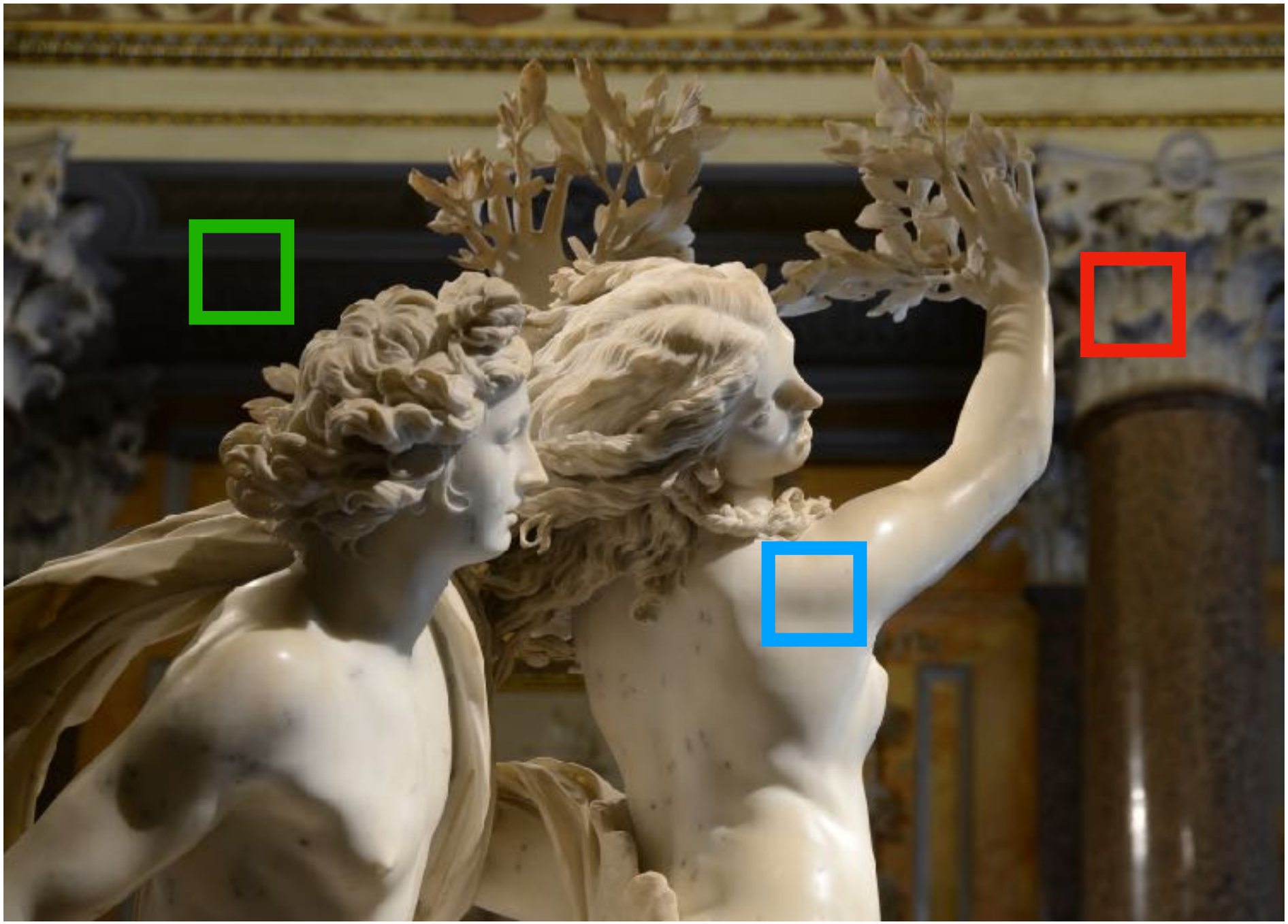
IPython NB Demo

Attack

- Exploit **locality**

- Possibly **share parameters** (?)

Locality



One Possible Solution

- Hidden unit considered everything
- ***Consider just patch***

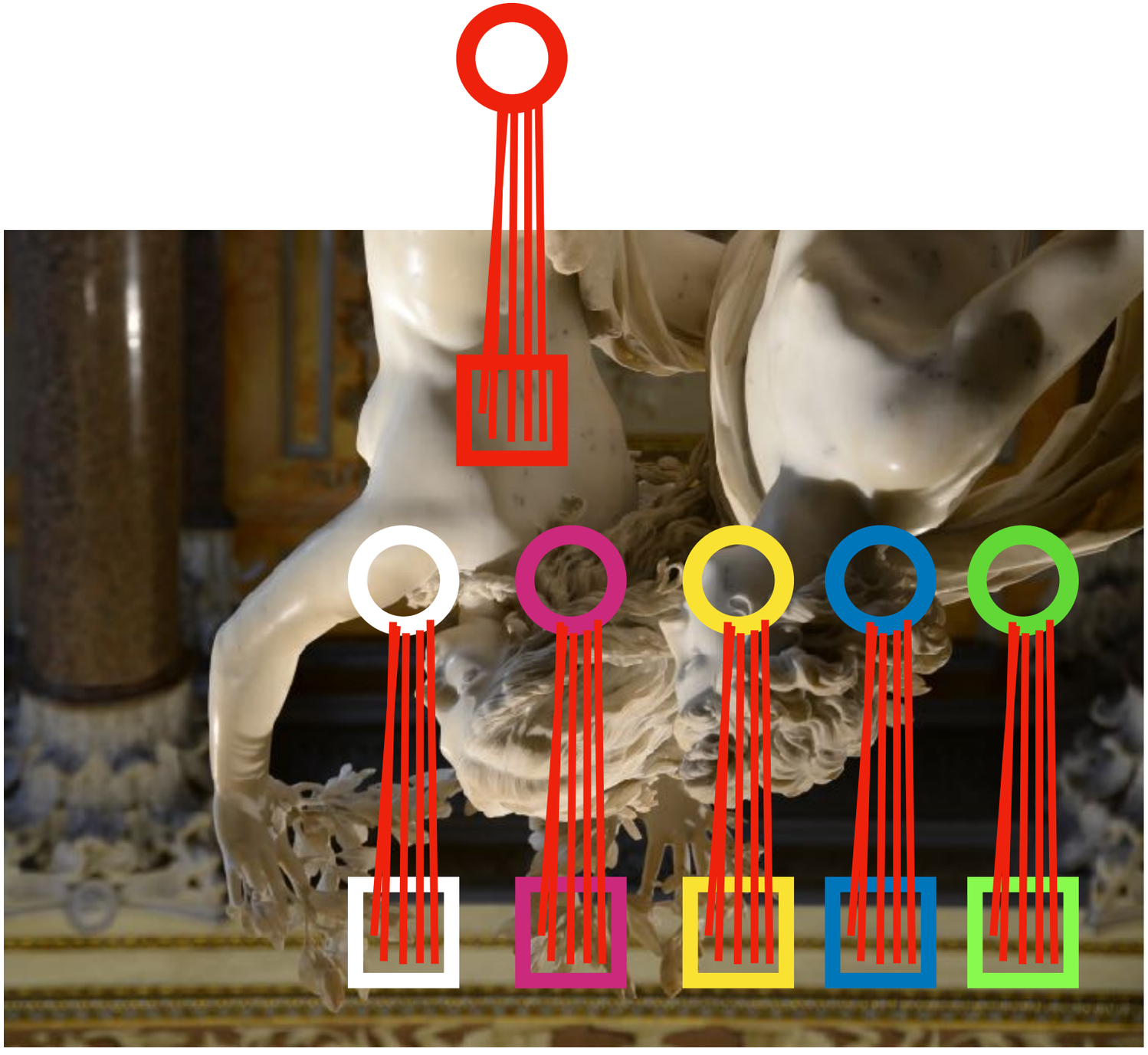
Looking just at this patch

Assume

- 10 x 10 patch
- Consider just pixels in this patch
- 100 parameters per hidden unit
- Previously = # of pixels in image

We've Cut Parameters

What’s next?



Huh?

Hidden Unit

- Linear combination of inputs
  - (and some nonlinearity but forget that for now)

We Are

- **Fixing the coefficients** (parameters)

- **Using these same parameters** to process different patches

Input Volume (+pad 1) (7x7x3)

`x[:,:,0]`

0	0	0	0	0	0	0
0	0	1	2	0	0	0
0	2	1	0	1	0	0
0	0	0	2	2	1	0
0	2	0	0	2	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

`w0[:,:,0]`

1	0	1
1	1	1
-1	0	0

So…

- 1 Image Processed with
  - **100** parameters (10x10 patch)
- Previously
  - # of pixels - **40,000**

This is a

Discrete

convolution

2D Discrete Convolution

Input

Parameters

Linear combination weights for a patch

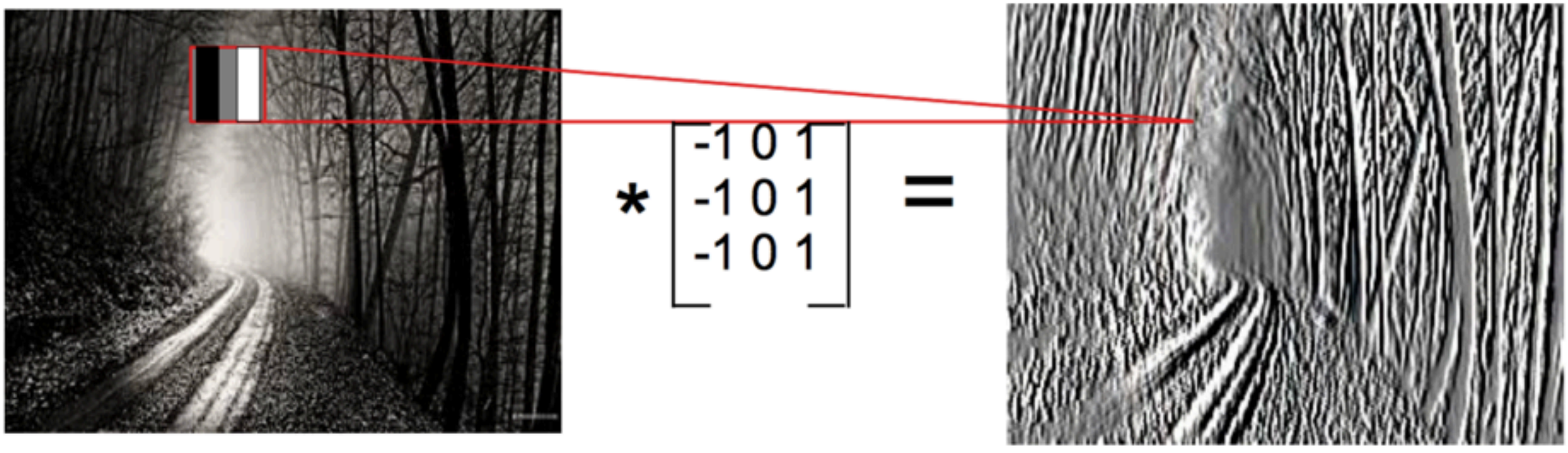
+

Biases

Hyperparameters

- Size of patch
- Stride
  - How far apart are adjacent patches

Output?



Input **filter** **output**

Remember

- We said deep learning solutions = shape specifications
- Convolution:
  - Input = Image
  - Output = Something image-like
  - With a different shape

Example

$$\begin{bmatrix} 1 & 0 & 2 \\ 2 & -1 & 4 \\ 0 & -5 & 3 \end{bmatrix} * \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -2 & 3 \\ -2 & 3 \end{bmatrix}$$

Input **filter** **output**

Stride 1

What About Channels

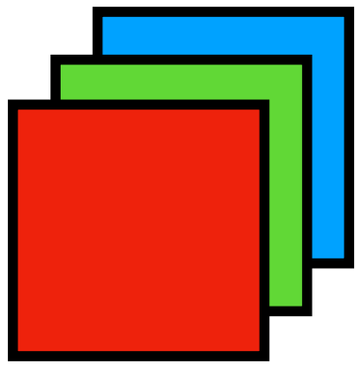
- An RGB Image:
  - Width x Height x 3
  - 3:
    - Red
    - Green
    - Blue

Tensor

Numpy shape

- (width, height, 3)

The convolution operator



A Patch

So

- Just a linear combination of all R, G, B pixels in a batch

And

- Same for n-channel “images”

We Just Implemented

- A Layer That:
  - Exploits locality
  - Cuts parameters (drastically)
  - Shares parameters

Mission Accomplished

Code

2D Convolution layer:

- Keras:

```
Conv2D(
    filter_size=24,
    kernel_size=(2,2),
    strides=(1,1),
    padding='valid',
    input_shape=(128, 128, 3),
    channels_last=True
)
```

What Does This Mean

- 24 Filters
- Each filter is 2x2 - (patch size, FOV)
- Stride 1x1 - The adjacent patches are one over in both directions.
- Valid padding - don’t add any padding (tensorflow bs)
- input_shape - 128x128 color image (R, G ,B channels)
- Channels_last - tensor shape - see explanation

Parameters

- 24 Filters
- 2x2 filter
  - So patch has 4 pixels
  - Linear combination so **4 weights**
- **24 * 4 = 96**

Typically

- Conv2D is followed by:
  - Pooling layers
    - MaxPool
    - AvgPool
    - . . .

Pooling

- Also Image -> Image
- Operates at patch level as well
- Patches **Don’t Overlap**

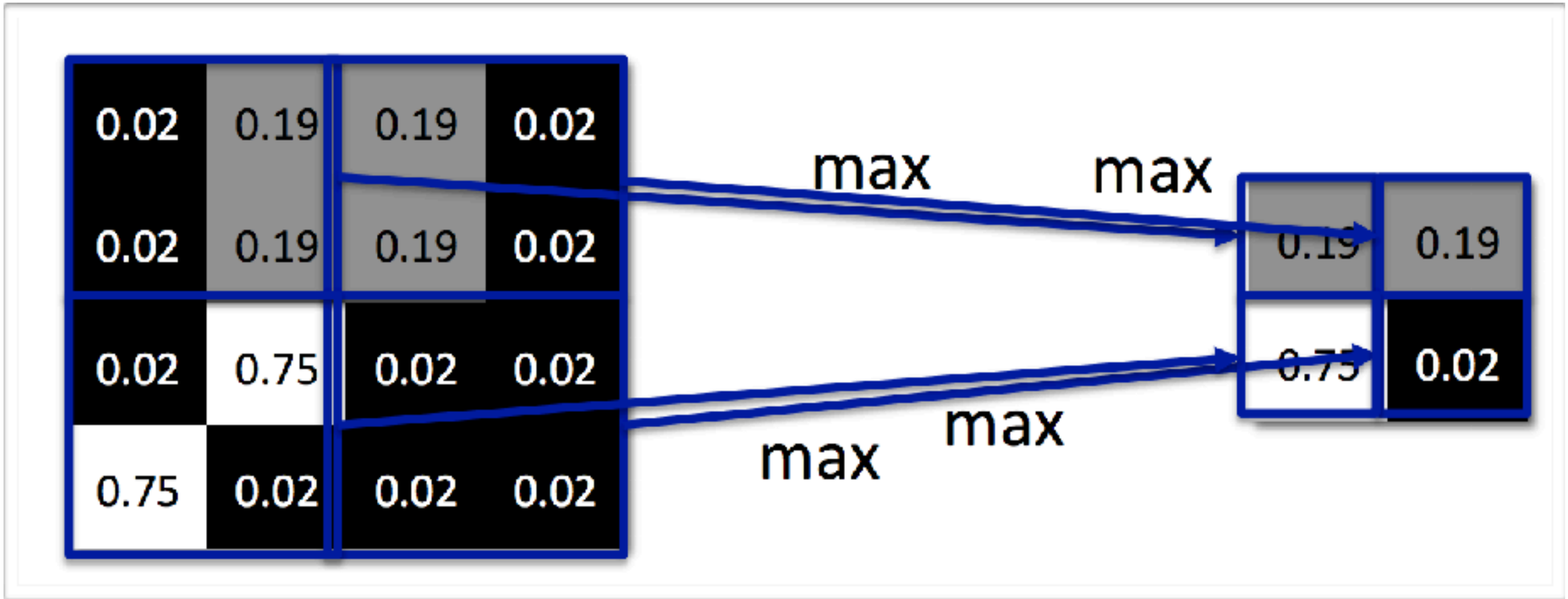
MaxPool

- Just take the max in a patch

AvgPool

- Just take the average in a patch

MaxPool


0.02
0.19
0.19
0.02
0.02
0.19
0.19
0.02
0.02
0.75
0.02
0.02
0.75
0.02
0.02
0.02
max
max
max
max
0.19
0.19
0.75
0.02

Code

- keras:

```
MaxPooling2D(
    pool_size=(2, 2),
    strides=None,
    padding='valid',
    data_format=None
)
```

What Do You Get

A New Shrunken(ish) image

Parameters

- None

Hyperparameters

- Stride

- Patch size

Now

- We discussed
- Image -> Image transformations
- Pooling, Conv2D

How does it fit in

- Cross entropy not defined for image
- Regression not defined for image
- Etc. Etc.

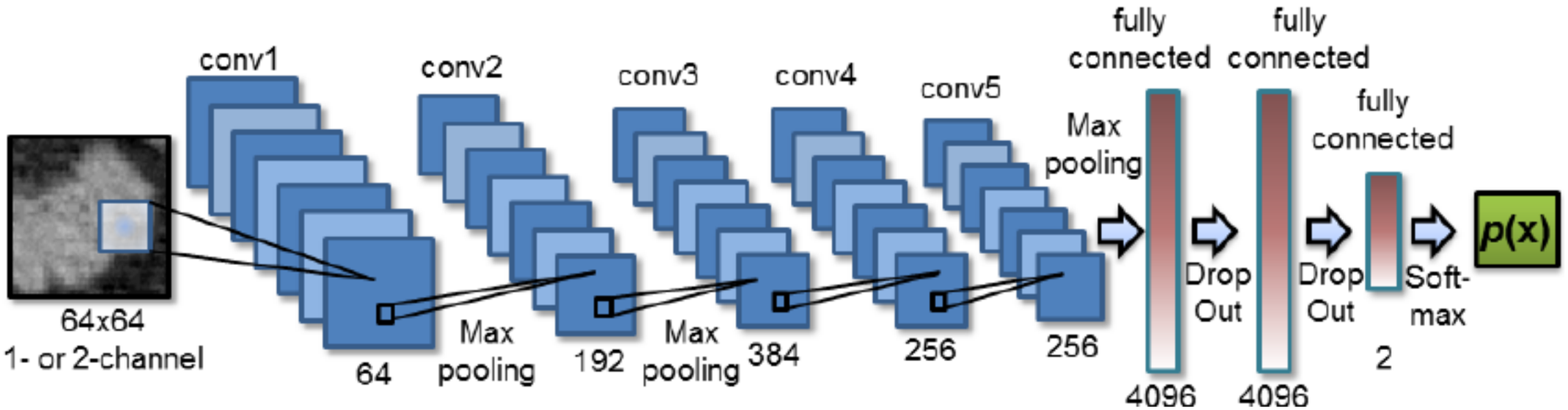
Deep Convnet

- Series of image transformations

- At the end you get an image

Just

- Flatten the resulting image
- Use normal fully connected layers from there on


conv1
conv2
conv3
conv4
conv5
fully connected
fully connected
fully connected
Max pooling
64x64
1- or 2-channel
64
Max pooling
192
Max pooling
384
256
256
4096
Drop Out
4096
Drop Out
2
Soft-max
p(x)

Code Example

- MNIST

Acknowledgements

- Ruslan Salakhudtinov - http://www.cs.cmu.edu/~rsalakhu/10707/Lectures/Lecture_Conv1.pdf

- Some figures from CS231N @ Stanford

- Good vibes from the Keras and TF teams