Convolutional Networks

Overview

1. What’s difficult about images?
2. Review of signal convolution
3. Review of image convolution
4. Convolutional Neural networks
    a. Kernels
    b. Strides, Padding
    c. Pooling

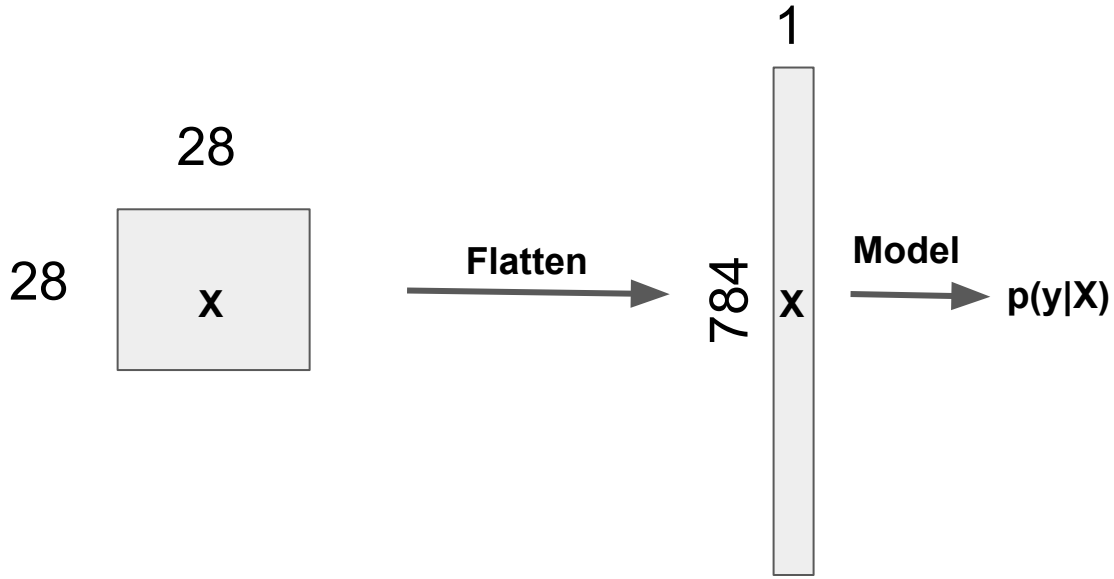
Image Analysis

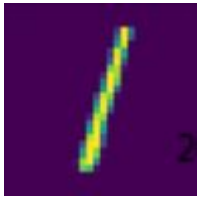
What’s difficult about images?

- The analysis and processing of images has historically been very challenging from a statistical modeling perspective
  - Entirely unstructured data source
  - Extremely high dimensional
  - Complex distribution over the inputs
  - Not amenable to classical modeling methods
- Yet, identifying objects in images (or the visual world) is supremely easy, even for children

What's difficult about images?

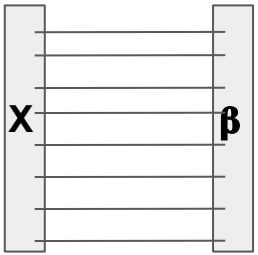
- What is even the right way to begin to model images?
  - Input features must be the raw pixel values



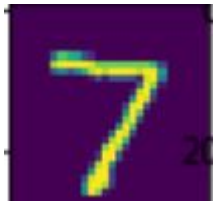


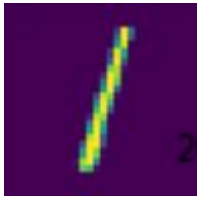
What's difficult about images?

- What is even the right way to begin to model images?
  - Input features must be the raw pixel values
  - **Regression** or **Logistic Regression** model?
    - Each input pixel gets a coefficient?
    - But some pixels are "active" on multiple or all classes
    - And combinations of pixels must be important
    - And we know that the distribution of pixel intensity values is certainly not Gaussian, so we're far away from the assumptions/theory that underlie these linear methods



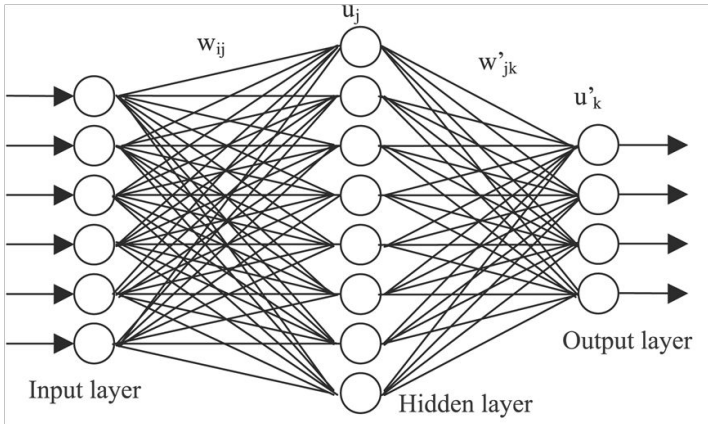
$$\hat{y} = X^T \vec{\beta}$$

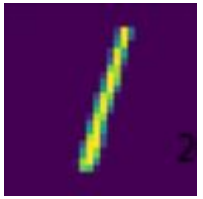




What's difficult about images?

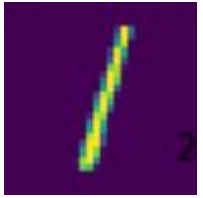
- What is even the right way to begin to model images?
  - Input features must be the raw pixel values
  - **ANN** model?
    - We get weights for every possible pixel interaction
    - This can work fairly well, but will hit a limit
    - Lots of parameters, model gets huge





What's difficult about images?

- These models treat all inputs equally, and all combinations of inputs equally
- Ignores some basic common intuition we have about the visual world

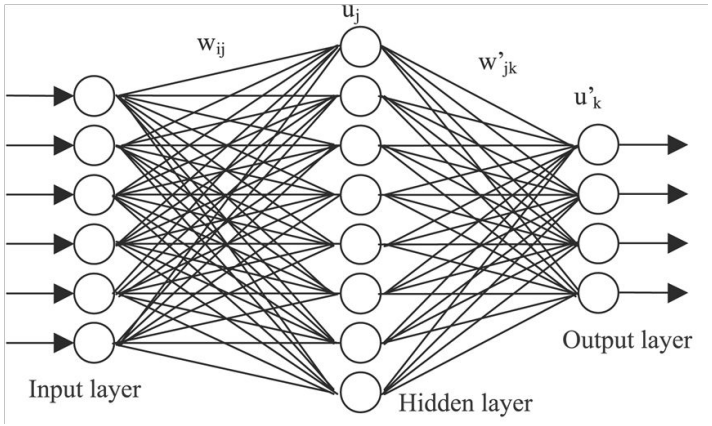


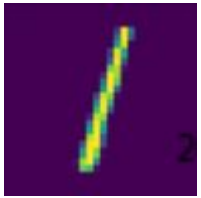
The two pixels are spatially far apart. They probably correspond to different parts of a visual world. They are less likely to be related to each other.

The two pixels are spatially close together. They are likely to be correlated.

What's difficult about images?

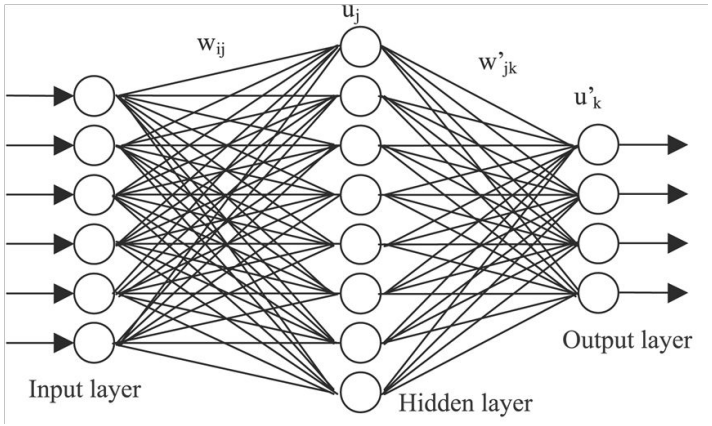
- Our basic intuition about spatial correlation is not captured by our ANN model
- The ANN model **could** learn about which pixel combinations are important are which are not, but there are so many parameters in this model, it turns out to be difficult to learn

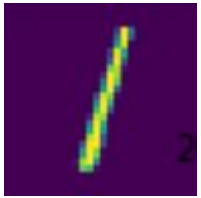




What's difficult about images?

- What if we just impose our prior belief that *close neighborhoods of pixels should be analyzed together*, but that far-away relations between pixels can likely be ignored?
- This is achieved through **kernel convolution**



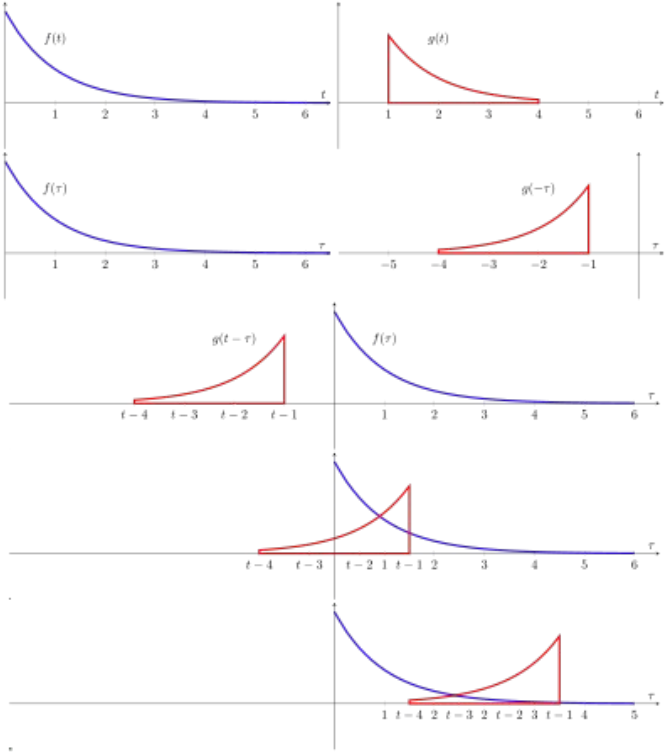


Review of Signal Convolution

Convolution for 1-D Signals

$$f(t) \circledast g(t) = \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau$$

- Pass one signal from left to right across the other
- At each point, integrate the area under both functions
- The output (not shown here) is a new function of t that highlights where the two input functions had similar “overlap”
- This area is very important in linear filter theory, signal processing, Fourier theory...



Convolution for 1-D Signals

$$f(t) \circledast g(t) = \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau$$

- Suppose the blue function is some data we've collected and care about
- The red function is some shape that we think might be relevant (call it a **kernel**)
- Convolving our function with this kernel will highlight which parts of our blue function have a high "match" with our kernel

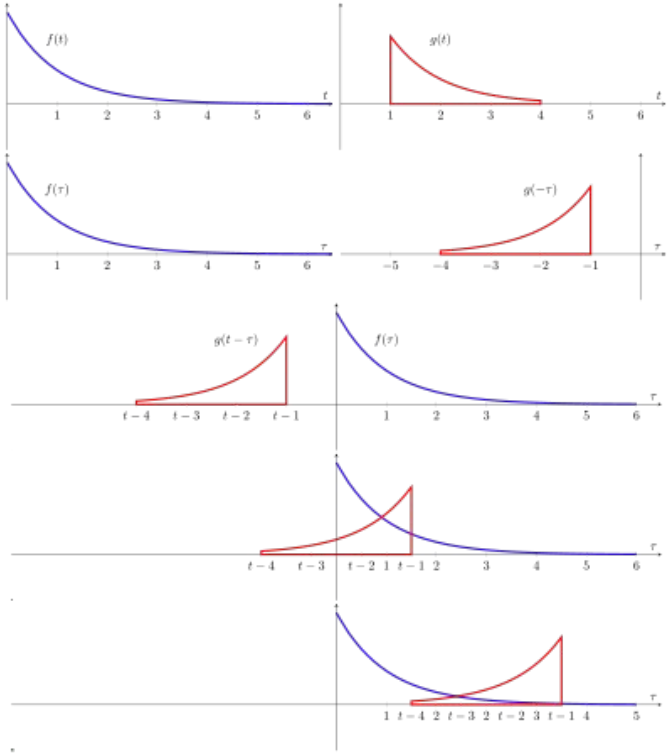


Image Convolution

Convolution for 2-D Signals

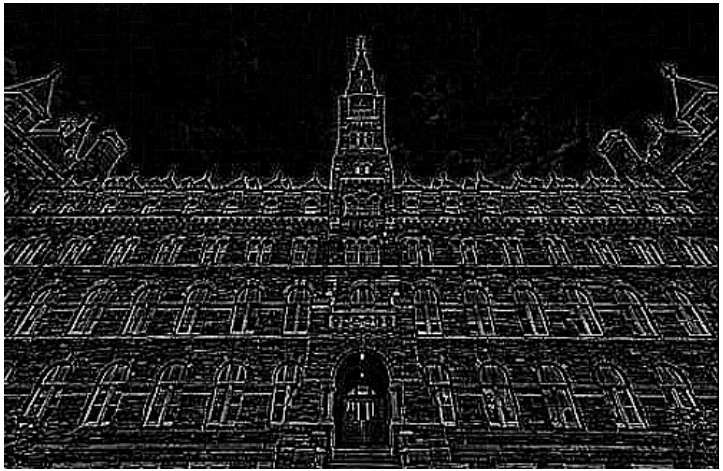
- We can repeat the same idea, but for a 2-D array of data and for a 2-D kernel.
- Here, we extend the idea of integration by sliding the kernel left-to-right and top-to-bottom across the input image, and computing the output.

A peek at where we are headed

Convolving an image with a kernel:

Edge Detection Kernel

-1	-1	-1
-1	8	-1
-1	-1	-1



Convolution for 2-D Signals

- Hadamard Product

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}\, b_{11} & a_{12}\, b_{12} & a_{13}\, b_{13} \\ a_{21}\, b_{21} & a_{22}\, b_{22} & a_{23}\, b_{23} \\ a_{31}\, b_{31} & a_{32}\, b_{32} & a_{33}\, b_{33} \end{bmatrix}$$

- then Sum
- then move on to the next location

Convolution for 2-D Signals

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

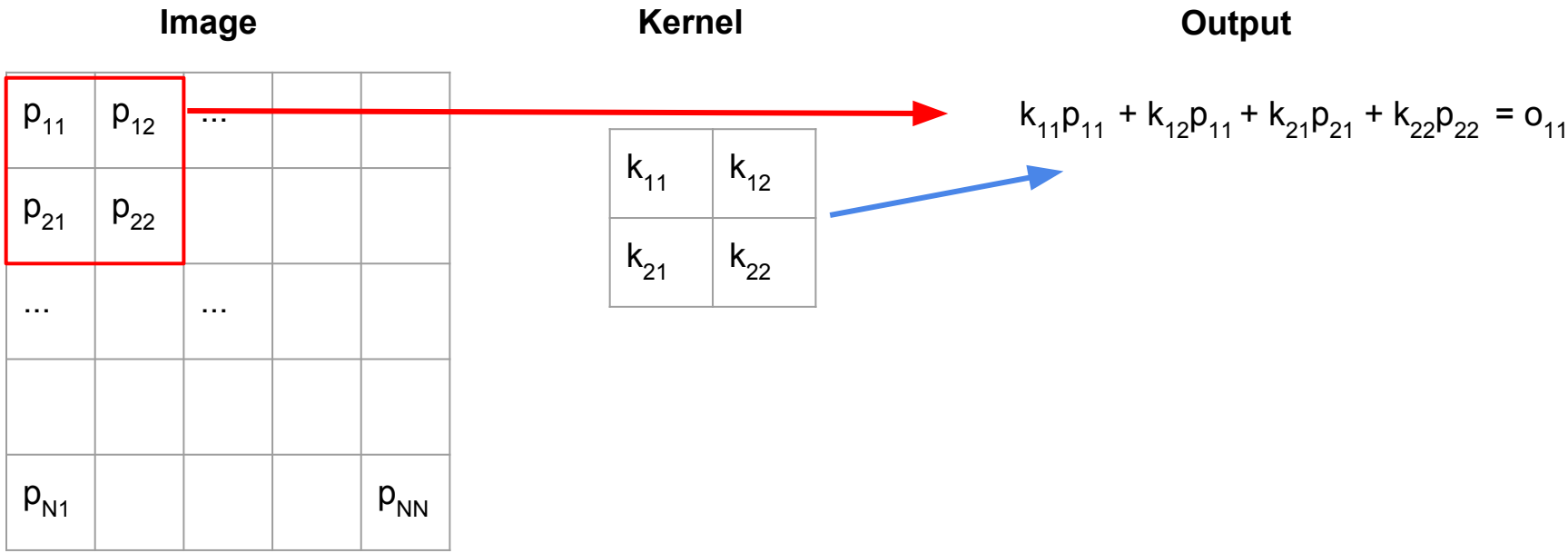
Image

p_{11}	p_{12}	...		
p_{21}	p_{22}			
...		...		
p_{N1}				p_{NN}

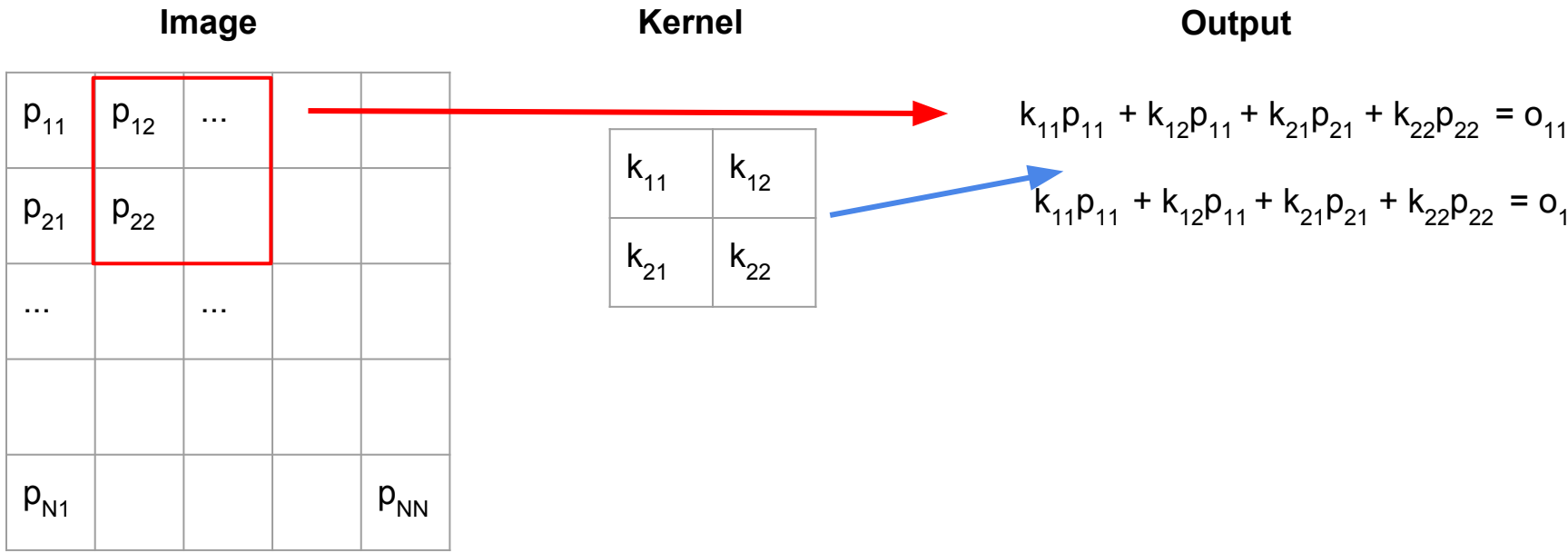
Kernel

k_{11}	k_{12}
k_{21}	k_{22}

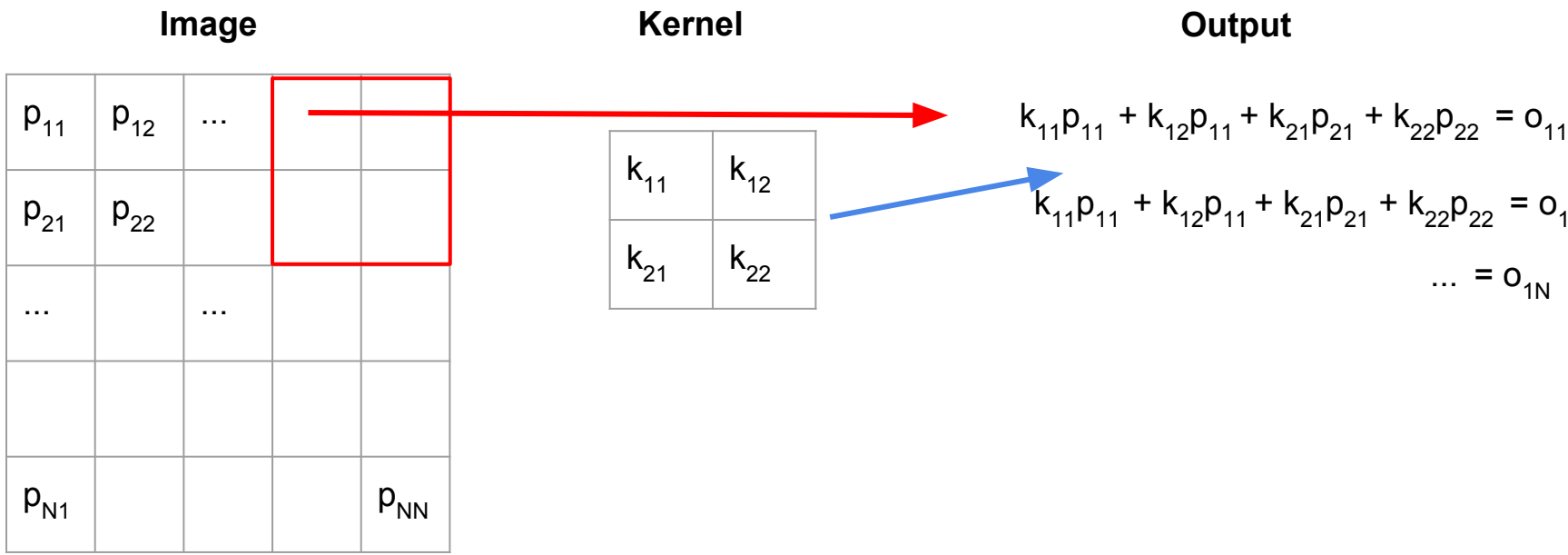
Convolution for 2-D Signals



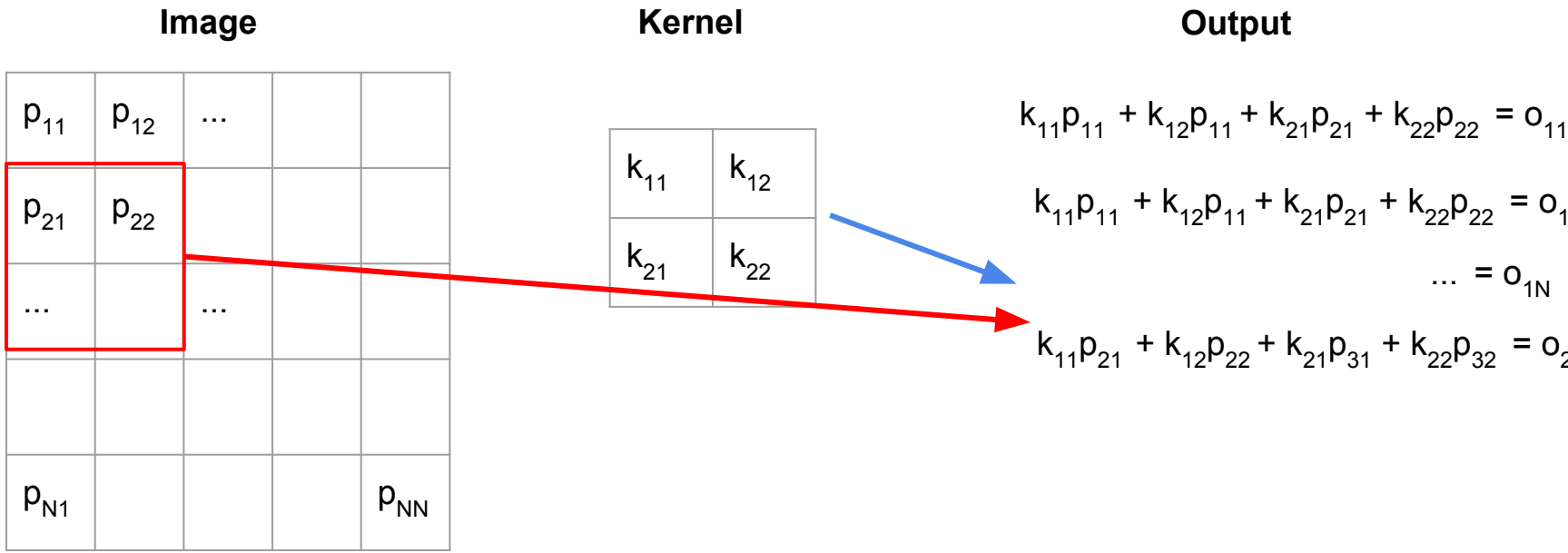
Convolution for 2-D Signals



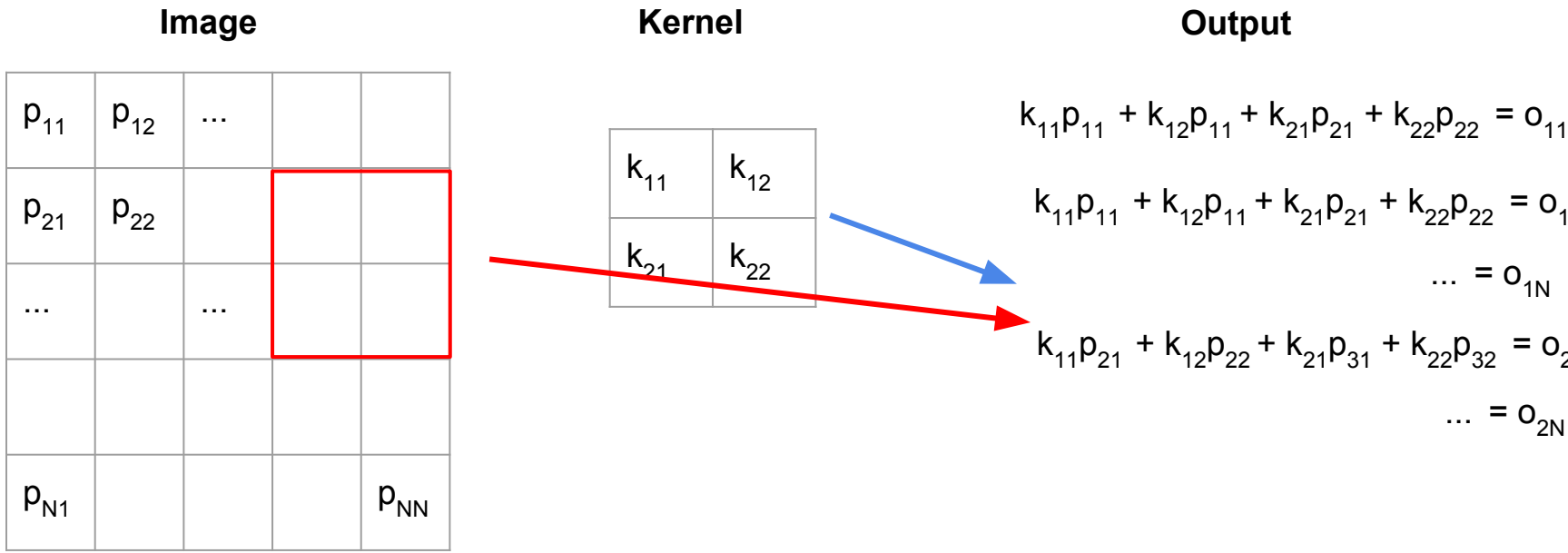
Convolution for 2-D Signals



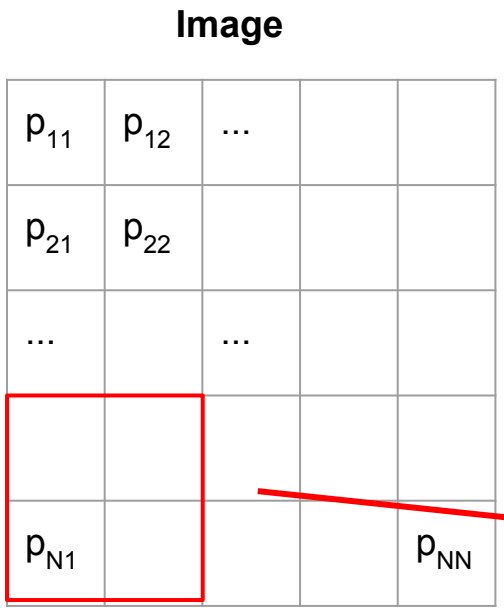
Convolution for 2-D Signals

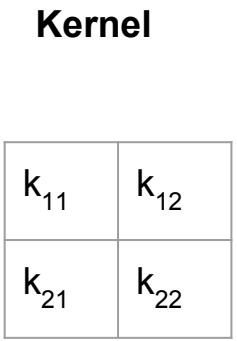


Convolution for 2-D Signals



Convolution for 2-D Signals





Output

$$k_{11}p_{11} + k_{12}p_{11} + k_{21}p_{21} + k_{22}p_{22} = o_{11}$$

$$k_{11}p_{11} + k_{12}p_{11} + k_{21}p_{21} + k_{22}p_{22} = o_{1}$$

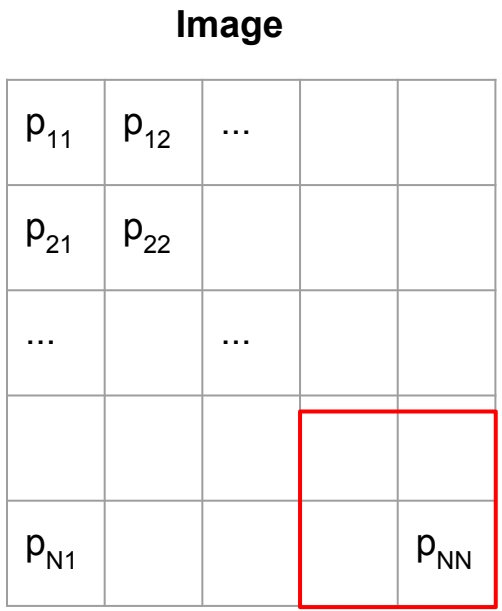
$$\ldots = o_{1N}$$

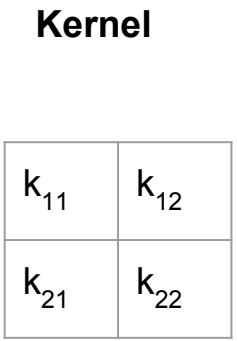
$$k_{11}p_{21} + k_{12}p_{22} + k_{21}p_{31} + k_{22}p_{32} = o_{2}$$

$$\ldots = o_{2N}$$

$$k_{11}p_{(N-1)1} + k_{12}p_{(N-1)2} + k_{21}p_{N1} + k_{22}p_{N2} = o_{N}$$

Convolution for 2-D Signals





Output

$$k_{11}p_{11} + k_{12}p_{11} + k_{21}p_{21} + k_{22}p_{22} = o_{11}$$

$$k_{11}p_{11} + k_{12}p_{11} + k_{21}p_{21} + k_{22}p_{22} = o_{1}$$

$$\ldots = o_{1N}$$

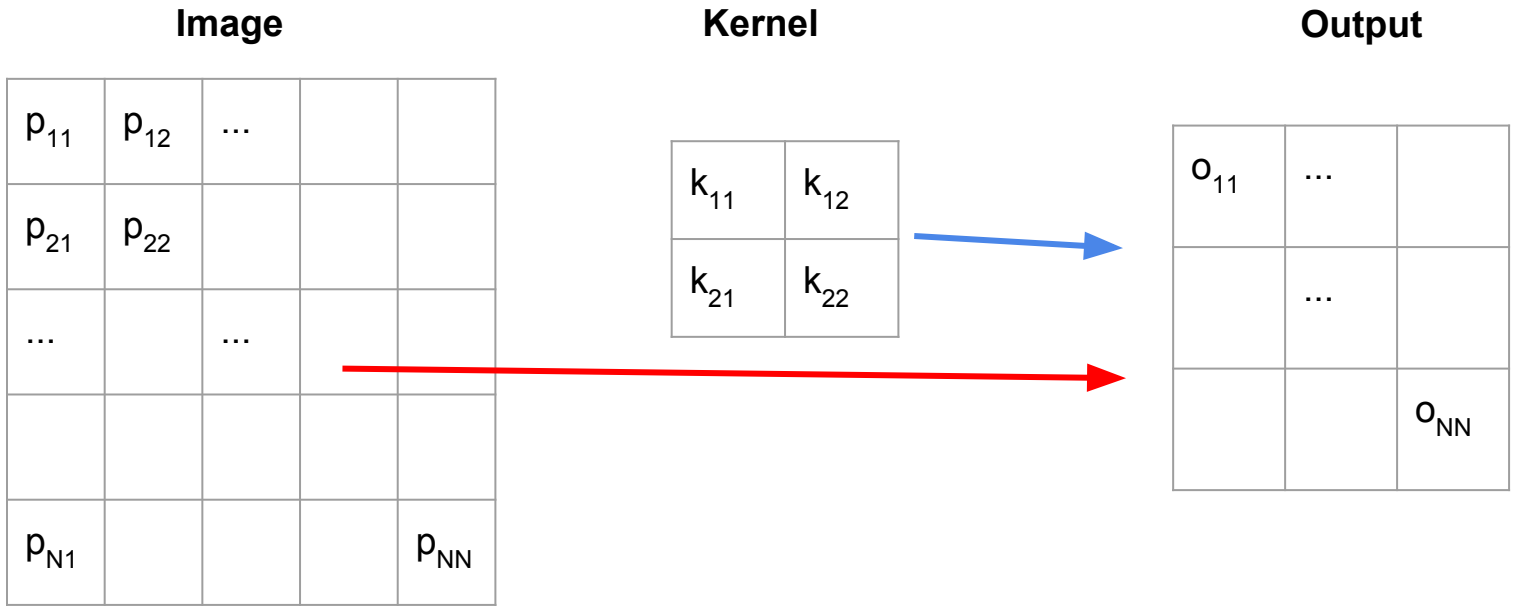
$$k_{11}p_{21} + k_{12}p_{22} + k_{21}p_{31} + k_{22}p_{32} = o_{2}$$

$$\ldots = o_{2N}$$

$$k_{11}p_{(N-1)1} + k_{12}p_{(N-1)2} + k_{21}p_{N1} + k_{22}p_{N2} = o_{N}$$

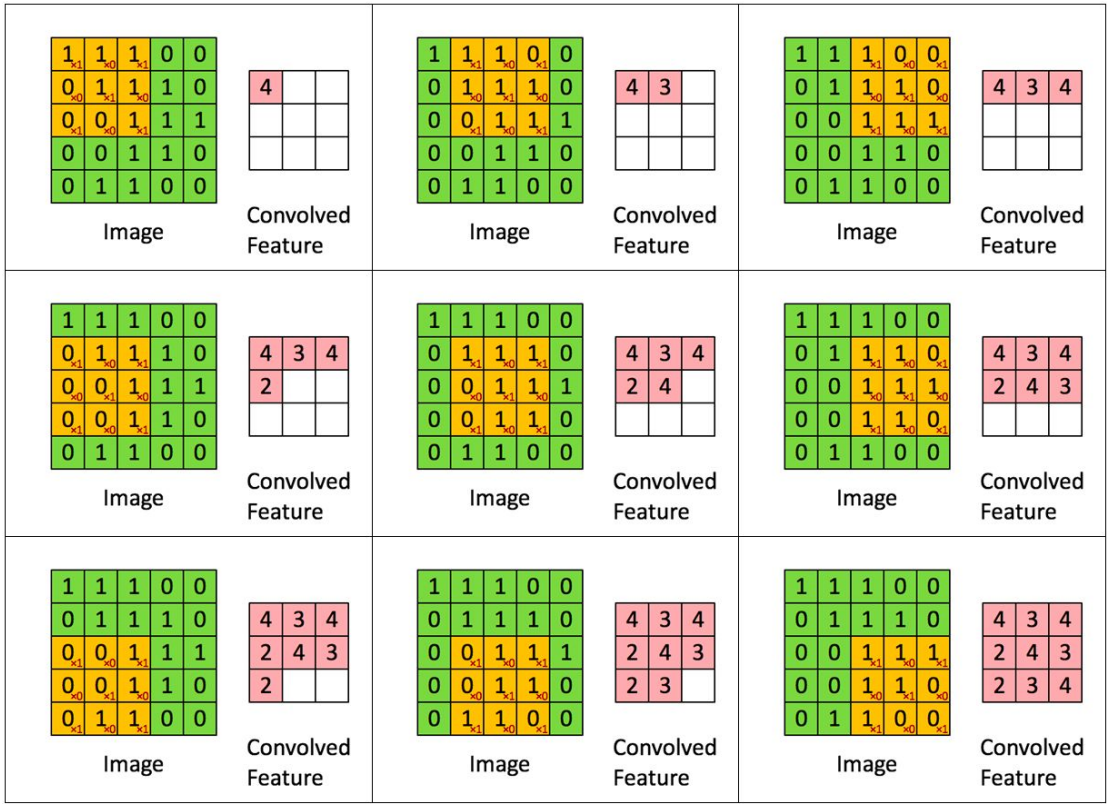
$$\ldots = o_{NN}$$

Convolution for 2-D Signals



After convolving an image with a kernel, the output is a new “image” that has higher values in some areas that another. The output will depend on the value of the kernel, but generally the output will highlight which areas of the image has a high “match” with the kernel.

Convolution for 2-D Signals



Some additional gifs to bring home the idea

https://cdn-images-1.medium.com/max/1600/0*9J3MK1gd2zrFDzDN.gif

https://i.stack.imgur.com/YyCu2.gif

https://i.stack.imgur.com/FjvuN.gif

Convolution for 2-D Signals

http://setosa.io/ev/image-kernels/

Convolutional Kernels

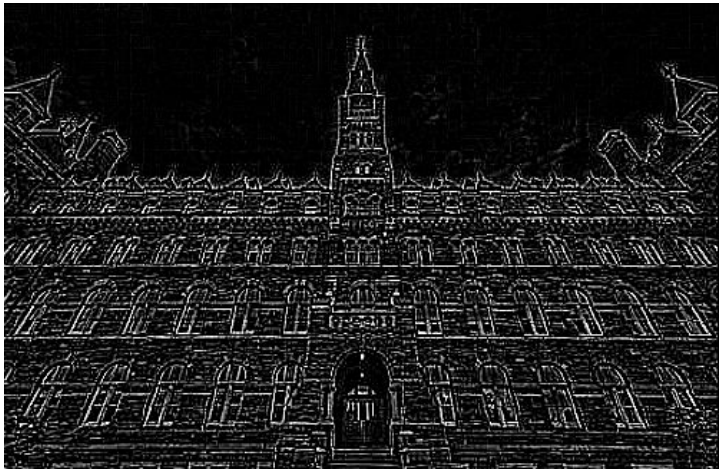
Blurring Kernel

.06	.12	.06
.12	.25	.12
.06	.12	.06

Convolutional Kernels

Edge Detection Kernel

-1	-1	-1
-1	8	-1
-1	-1	-1



Convolutional Kernels

Emboss Kernel

-2	-1	0
-1	1	1
0	1	2

Image Convolution

Historically, the fields of Image Processing and Computer Vision had been dominated by devising image kernels that were useful for solving specific tasks.

- Edge detection
- Texture detection
- Color & object detection

But much of this centered around **identifying** useful linear kernels. This was often done with much research and manual effort. And often tuned for a specific task.

Convolution-based Modeling

If we **know** we want edge detection, we can use a well-known kernel and our problem is done.

Further, we could build a model that stacks together these previously discovered kernels and uses this information for our task.

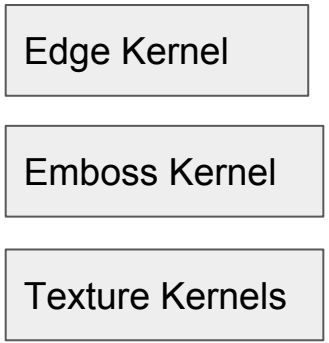
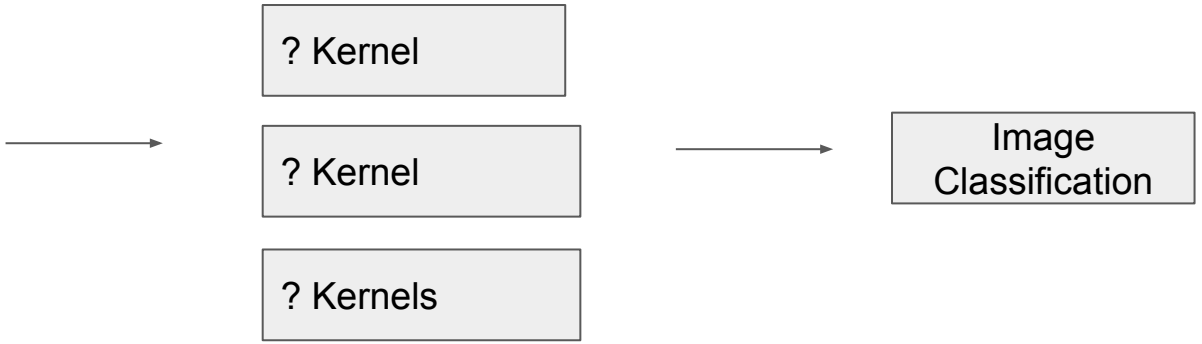


Image Convolution

But what if we **don't** use previously-discovered kernels?

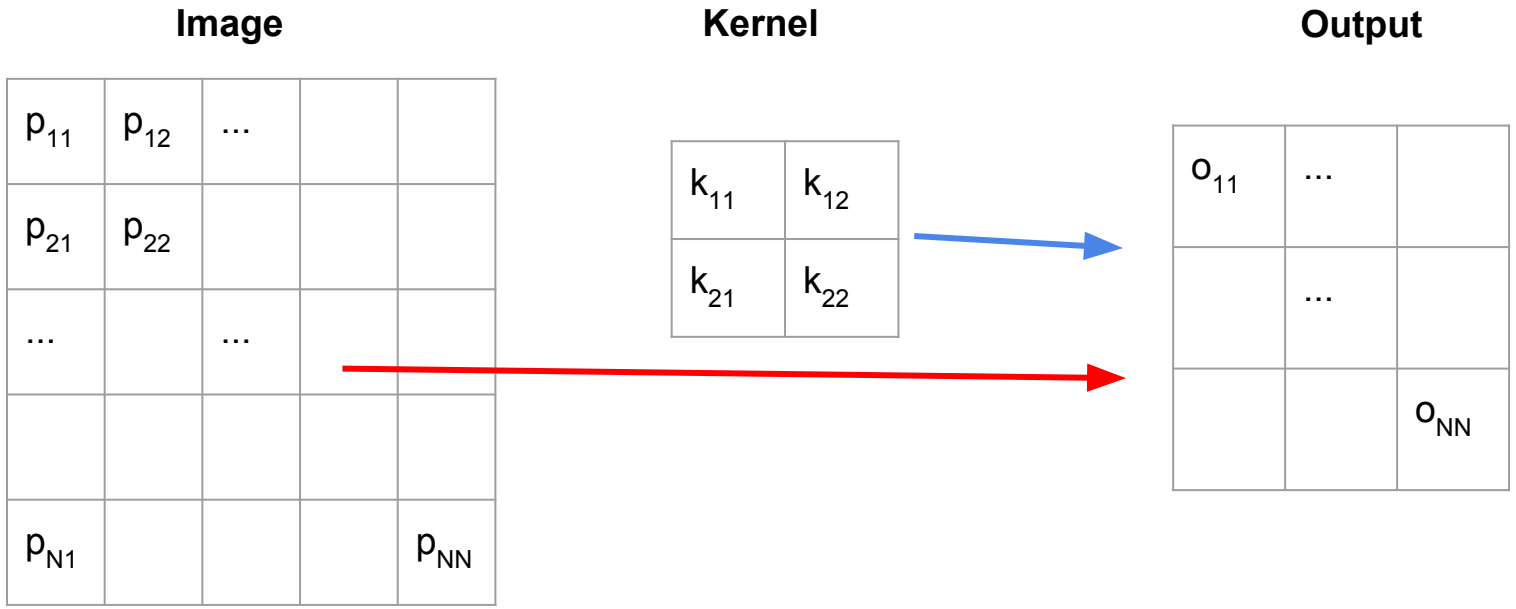
Maybe we have an ANN where the free parameters are the kernel weights.

Convolution Neural Networks: **Learn** the kernels that solve the classification task



Convolutional Neural Networks

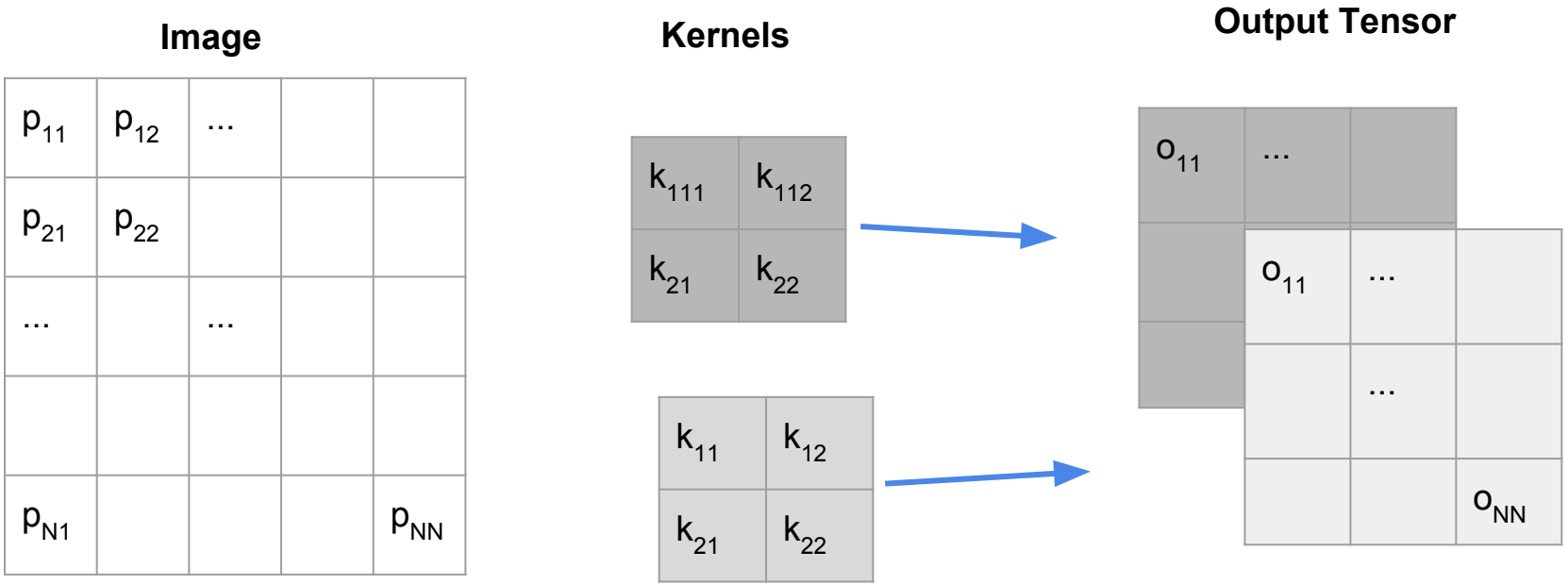
CNNs: Kernel Weights are the free parameters



In this instance, our model has 1 kernel (only 4 free weights).

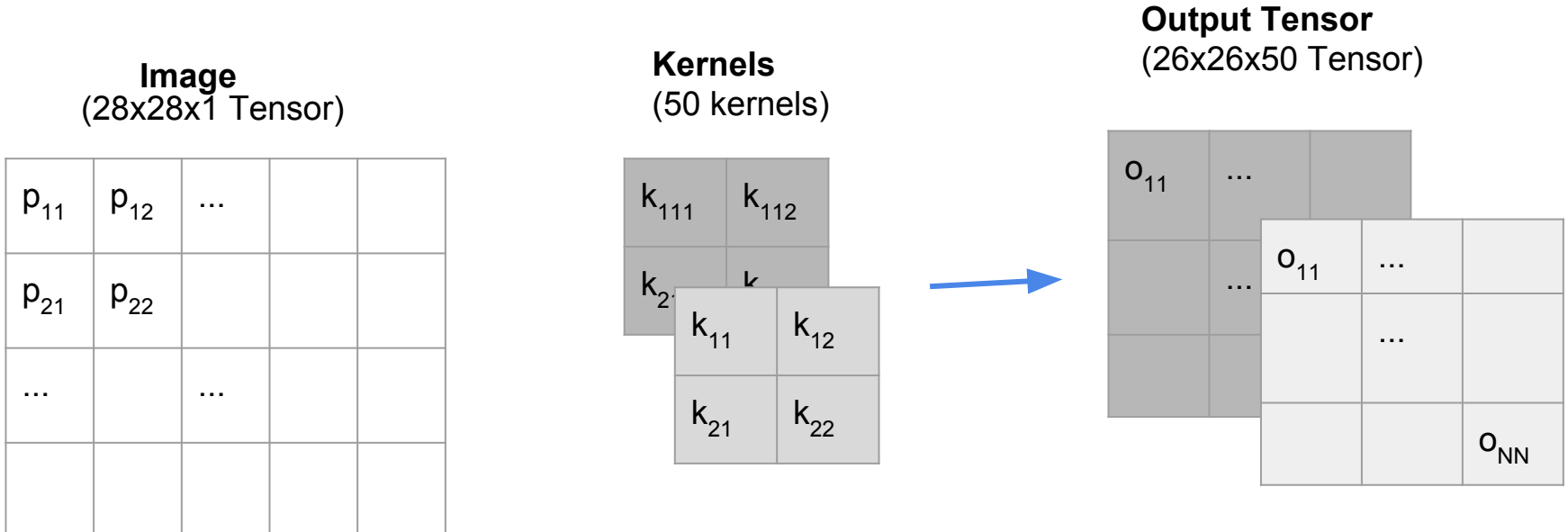
CNNs: Kernel Weights are the free parameters

But we can (and should) learn multiple kernels, not just one! Each kernel can detect different important image features.



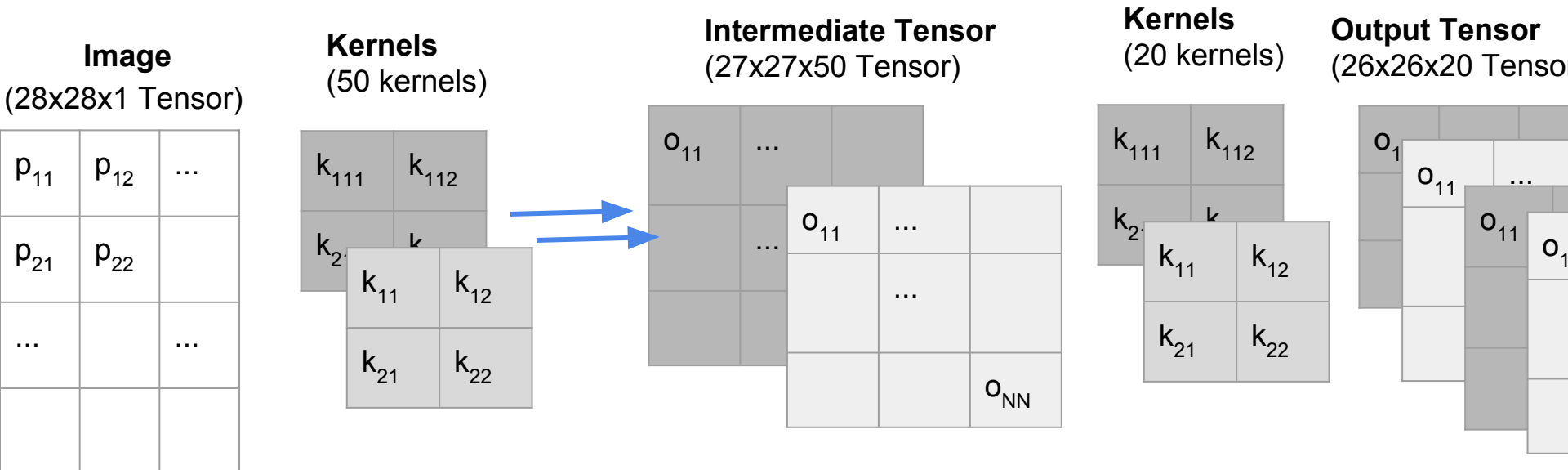
Convolutional Neural Networks

- For each kernel, we can apply common ANN activation functions (like ReLU)
- For each kernel, we get a corresponding output array, commonly called a **Feature Map**



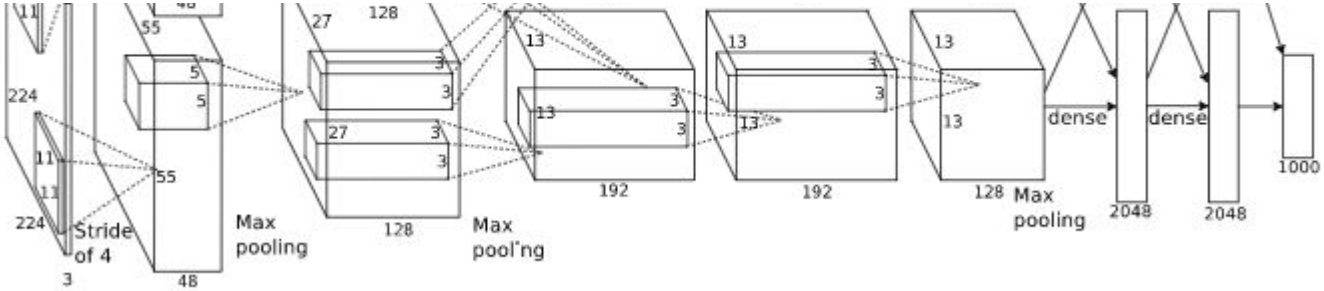
Deep Convolutional Neural Networks

- We can apply convolution again to the Feature Maps



Deep Convolutional Neural Networks

- This process is continued until we Flatten the output and have a final Dense layer(s) into a Softmax for our classification problem



AlexNet

Kernels

- We need to specify the size/shape of convolutional kernels
- Typically something like 2x2 or 3x3 or 5x5 or 7x7

Conv2D

```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid',
```

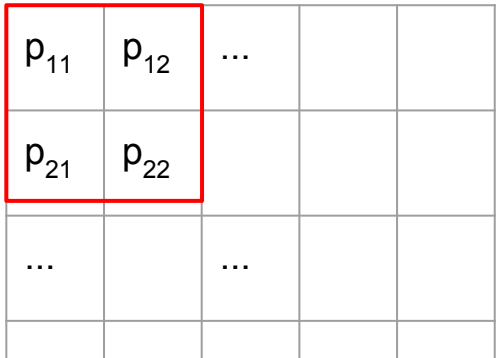
2D convolution layer (e.g. spatial convolution over images).

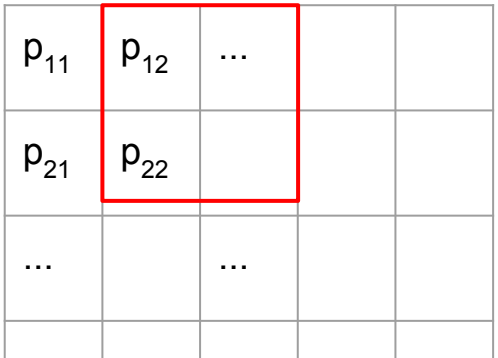
Arguments

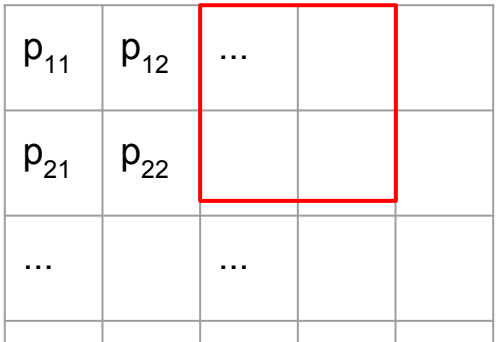
- **filters**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size**: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

Stride

- How should our kernel move across the image? Should it move as a sliding window, or jump past pixels?
- Stride of 1 is like a sliding window

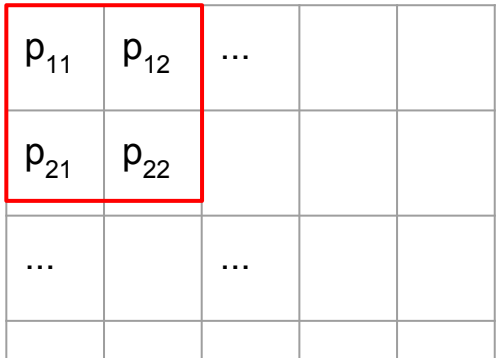


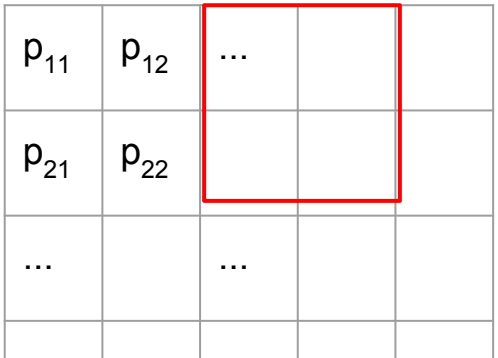


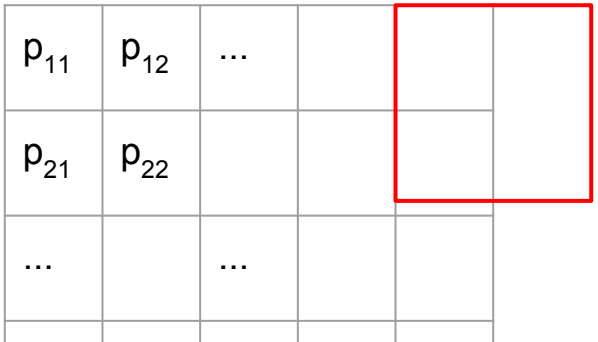


Stride

- How should our kernel move across the image? Should it move as a sliding window, or jump past pixels?
- Stride of 2 or more will skip some pixels, in case you don't want to double-count (will result in down-sampling in the output)

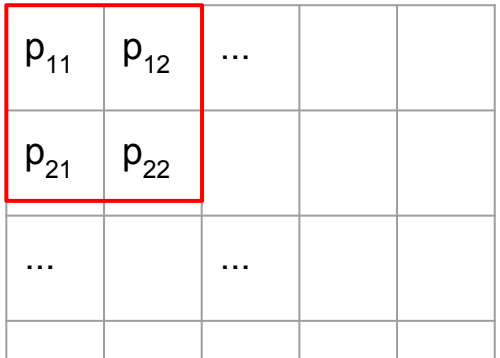


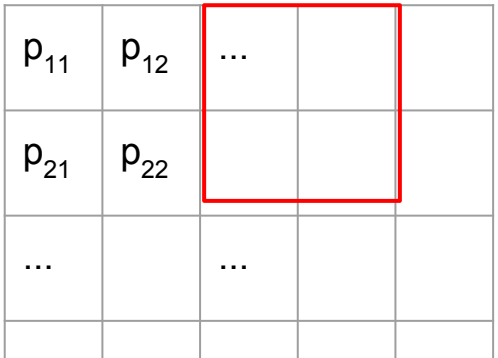


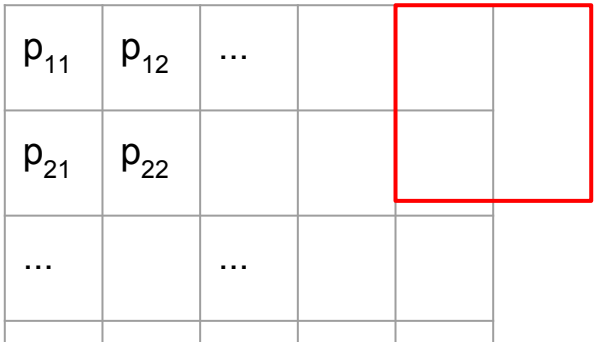


Padding

- How should we handle the border of the image where the kernel has no pixels to overlap?
- “Valid” padding adds no padding to the input and the output shape inevitably must be smaller than the input shape

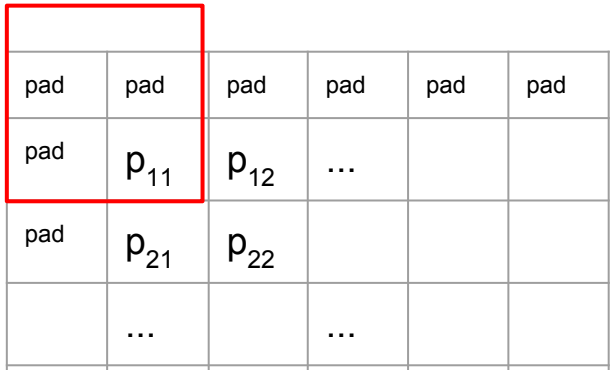


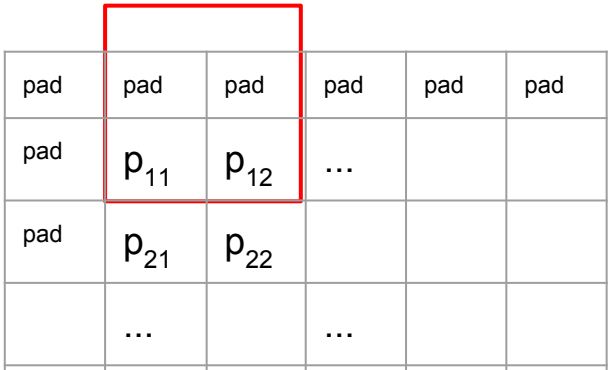


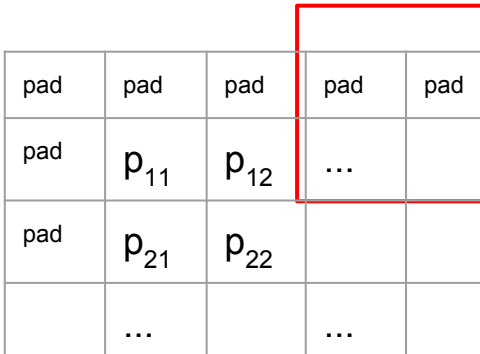


Padding

- How should we handle the border of the image where the kernel has no pixels to overlap?
- “Same” padding will add a perimeter of zero-padded values to the image, so that the output array has the same shape as the input





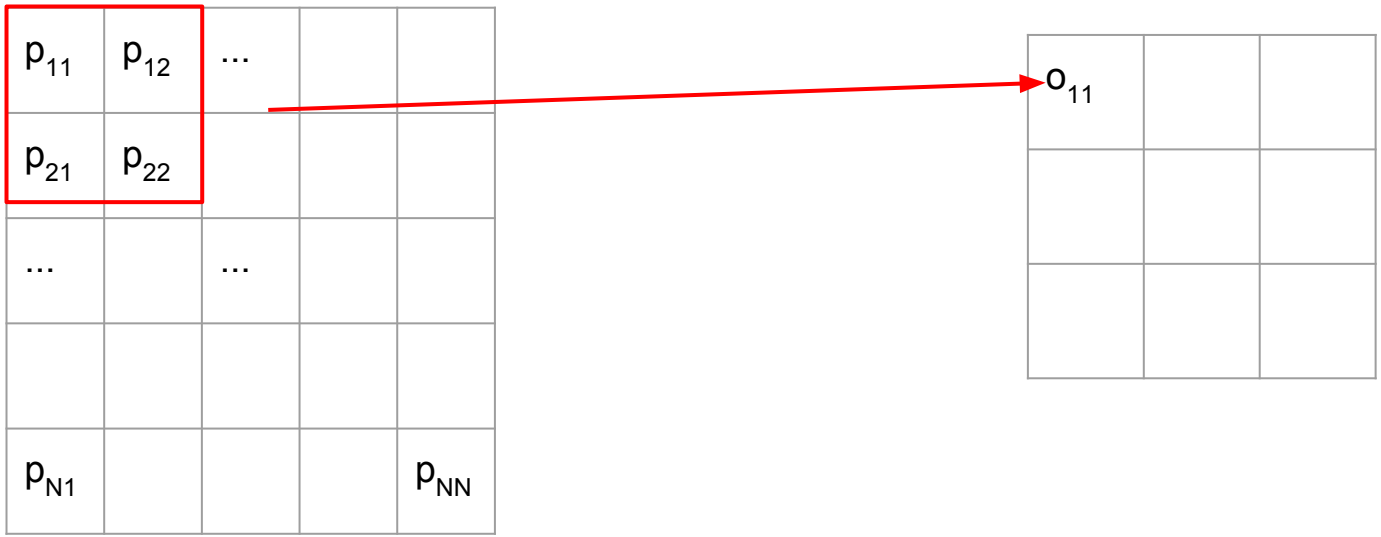


Pooling

- We can quickly down-sample a feature map (or an image) by applying pooling
- Here, we consider a collection of pixels (maybe 2x2 or 4x4) and output a summary of that collection
- Common techniques are AveragePool or MaxPool
- The goal is to highlight something like “did this kernel have high activation **anywhere** in this region?”, while losing spatial locational details
- Forces downstream layers to act upon wide patches of input image (receptive field)
- This has a big effect on the output shape: much down-sampling

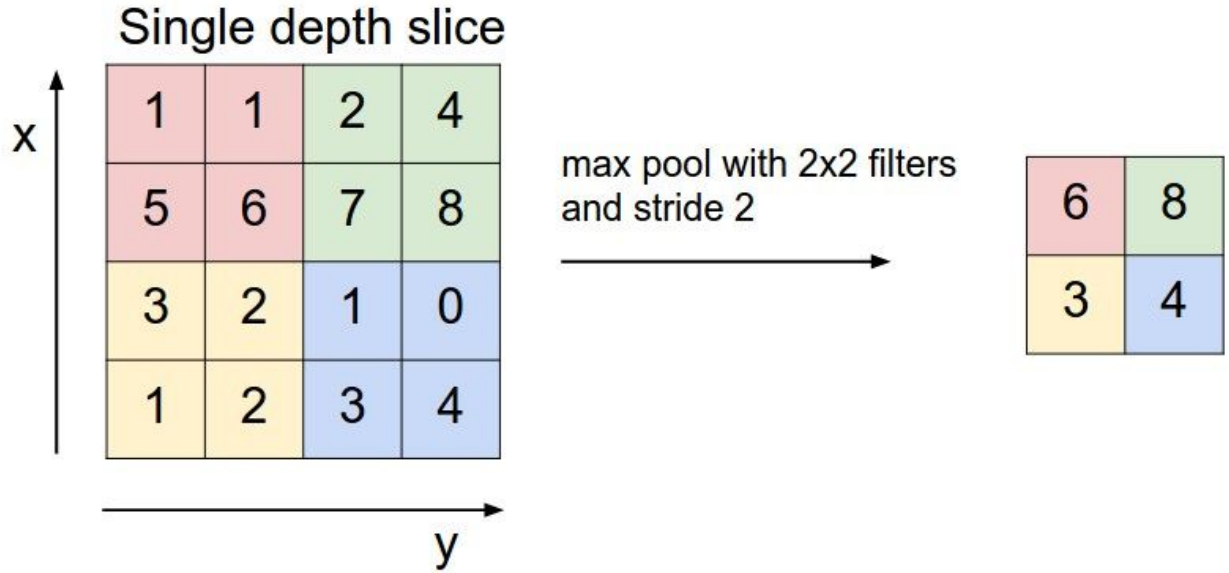
Pooling

A 2x2 MaxPool

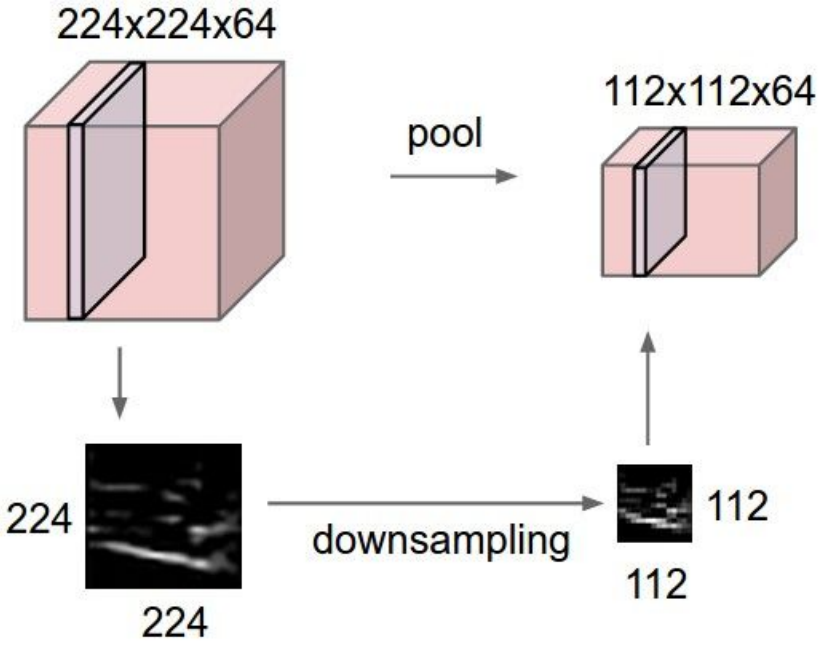


Note that a pooling layer is a type of kernel, but just has **0 free parameters**

Pooling

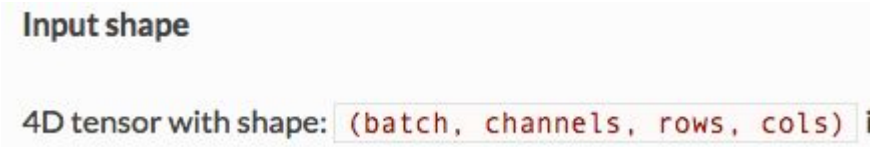

Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Pooling



Summary

- The design choices we make for a convolutional layer include kernel shape, number of kernels, stride, padding and pooling
- Each of these has impacts on the shape of the output tensors that results from your operations
- **Make sure you double check the output shapes of each Keras layer, make you sure understand why these outputs are taking a certain shape**



Keras Example

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

32 kernels, will result in 32 Feature Maps output

3x3 kernel shape, will impact output shape

Another convolution layer, now with 64 kernels

MaxPool over 2x2 windows

Flatten tensor into vector (length corresponds to number of elements in tensor)

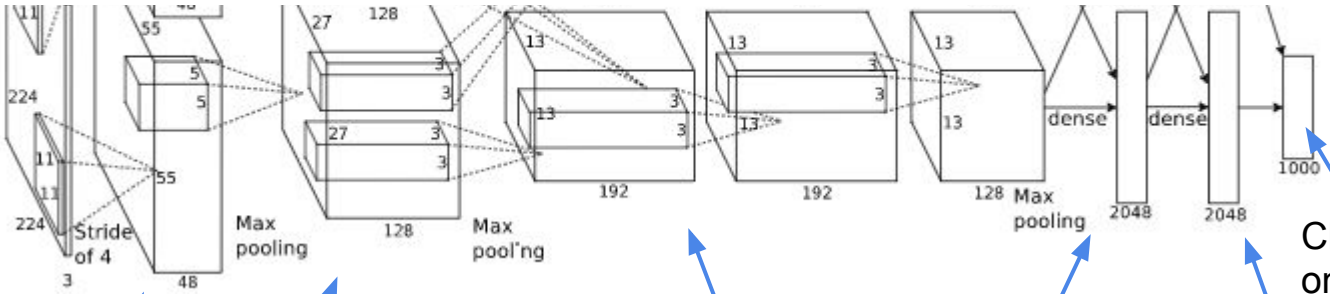
A couple Dense layers for final classification

Keras Example

```
model = Sequential()
# input: 100x100 images with 3 channels -> (100, 100, 3) tensors.
# this applies 32 convolution filters of size 3x3 each.
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd)
```

What are CNNs learning?

- We'll come back to this in more detail later, but it is useful to pause and ask why deep CNNs are so powerful



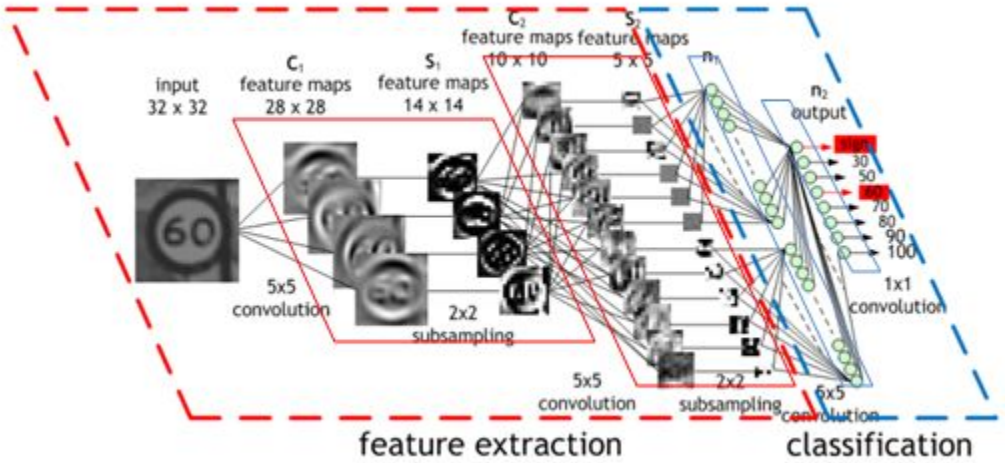
Extraction of simple low-level features (colors, edges)

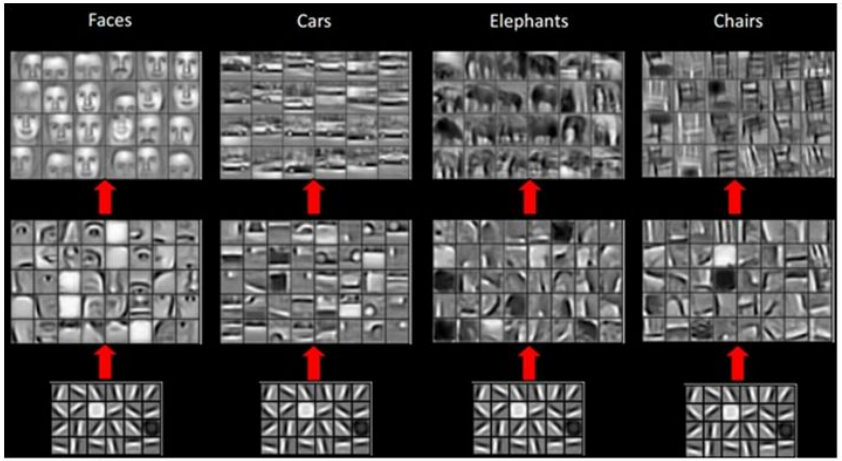
Combining low-level features in complex hierarchical ways, across whole image

A rich feature vector that tells us what is in the image

Classification base on those extracted features

Understanding CNNs





Visualizing outputs of CNNs

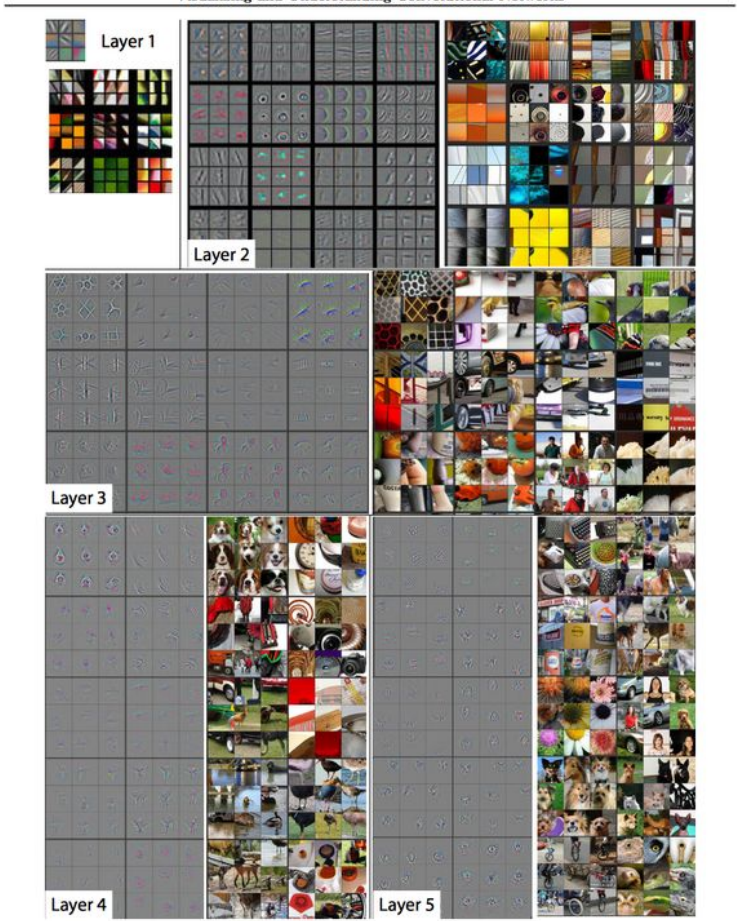
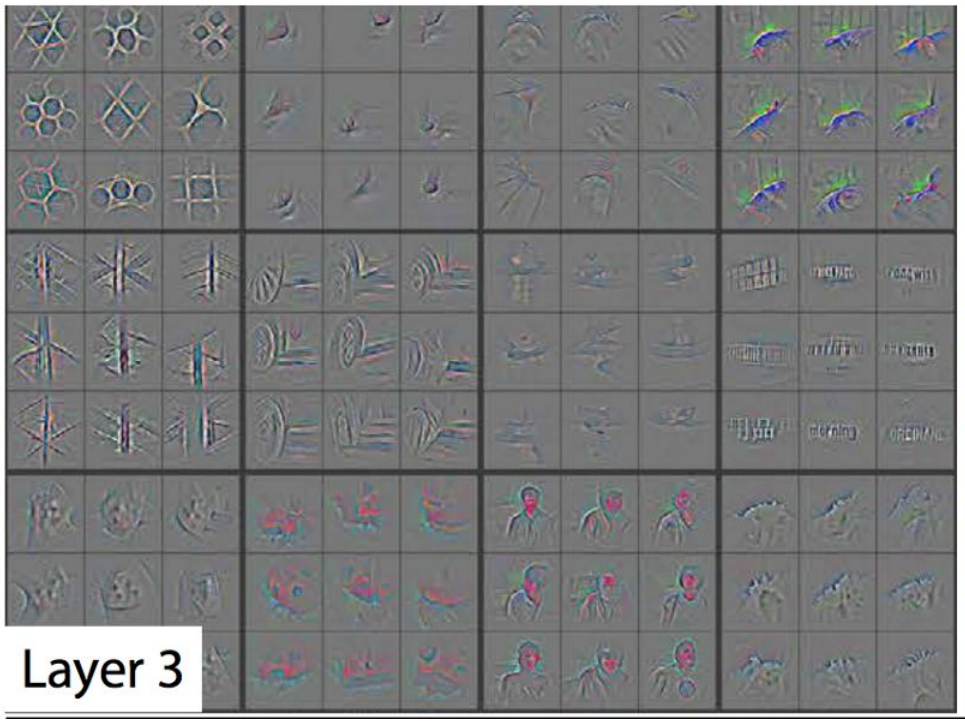


Figure 2. Visualization of features in a fully trained model. For layers 2-5 we show the top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using our deconvolutional network approach. Our reconstructions are *not* samples from the model: they are reconstructed patterns from the validation set that cause high activations in a given feature map. For each feature map we also show the corresponding image patches. Note: (i) the the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1). Best viewed in electronic form.

Visualizing CNNs



Convolutional Networks Summary

- Image convolution is a core operation for extracting features from images
- With CNNs, we can **learn** the best kernels for our task, instead of deciding them beforehand
- We can stack convolutional layers together - low level image features are combined in complex ways - abstraction and generalization
- Extra reading - http://cs231n.github.io/convolutional-networks/