AI - Neural Networks

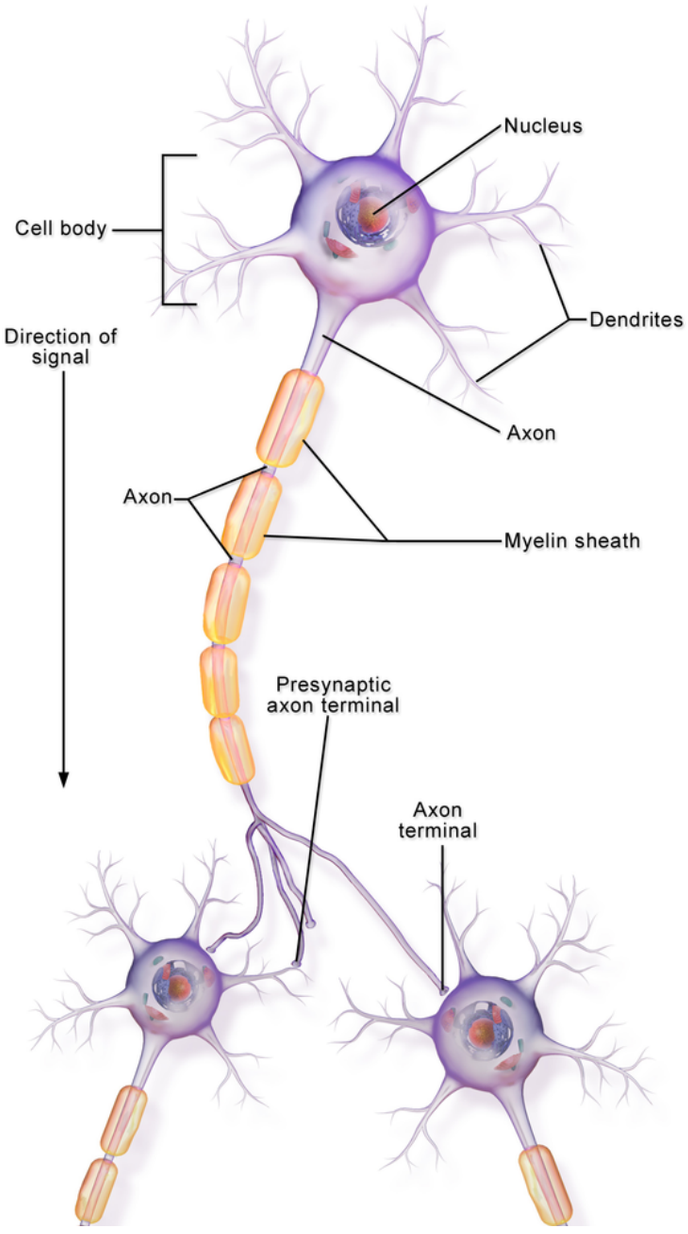
CS 246

Objectives

- Students will be able to:
  - enumerate the main categories of AI
  - understand basic terminology (so they can carry on a coherent conversation during a job interview)
  - speak, from personal experience, about the advantages and limitations of AI (so they can carry on a coherent conversation during a job interview)
  - explain the basics of how machine learning works
  - describe how neural networks work
  - recognize the importance of life-long learning

Biological Neurons

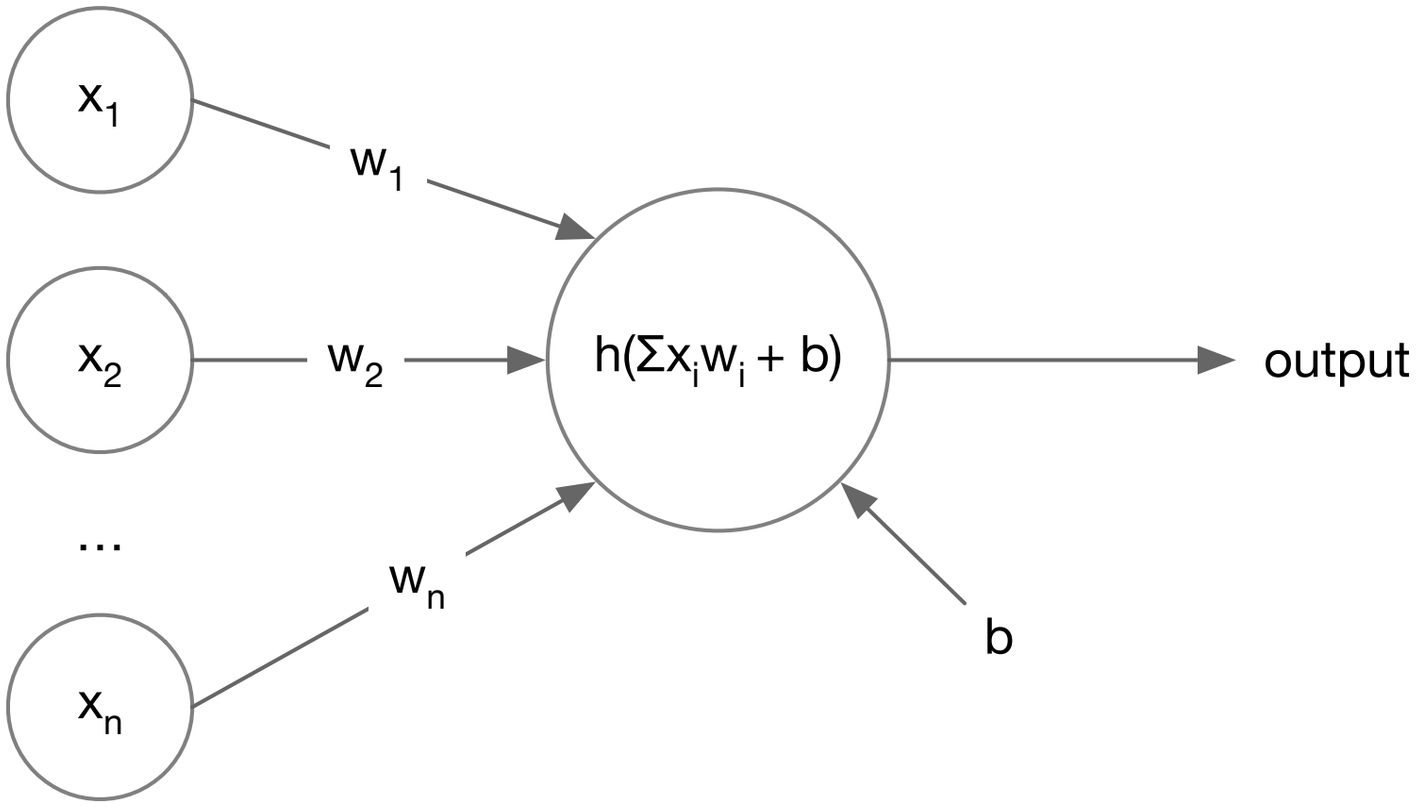
- A biological neuron receives electrical inputs from dendrites
- When a sufficient number of inputs are active, it fires a signal along its axon
- In the central nervous system, neurons connect to other neurons, forming a network (your brain)



Artificial Neurons

- In a neural network, a neuron's **activation function** h() acts on the weighted sum of its inputs (w_1, w_2, ...) plus a bias factor b to produce an output - all quantities are numeric

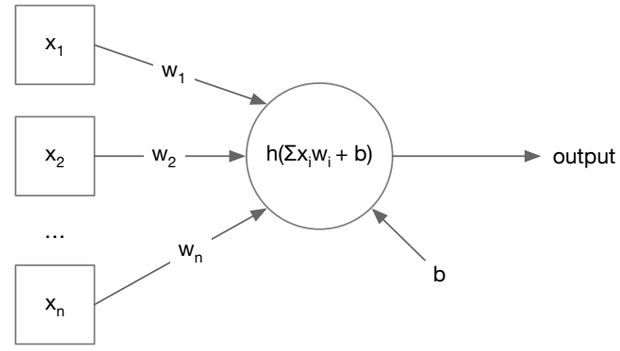
- h() introduces non-linearity into the network, making it possible to model complex relationships



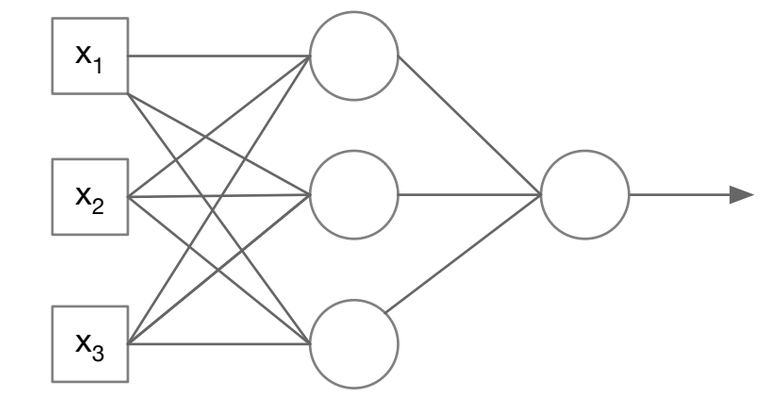
Activation Functions

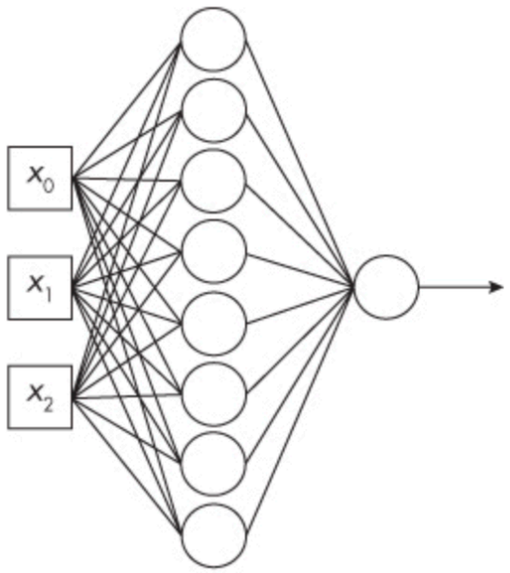
Name	Formula	Notes
ReLU (Rectified Linear Unit)	$f(x) = \max(0,x)$	Most hidden layers in modern deep nets - ensures that the output is non-negative
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	Output layer for binary classification - produces a number in [0,1], which is construed as a probability
Softmax	$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$	Output layer for **multi-class** classification - scales multiple outputs so each is in [0,1] and their sum is 1, i.e., they act like probabilities
Swish / GELU	Smooth, learned activations	Transformers, some cutting-edge models

Neuron Layers



- Neurons are usually arranged in **layers**, each layer feeding into the next layer
- The layers between the input and output neurons are called hidden layers
- If all inputs connect to all neurons in the hidden layer, they are said to be fully connected
- Each input to the neuron has a weight (except for the bias)
- How many weights and biases (parameters) do we for the network at left? At right?

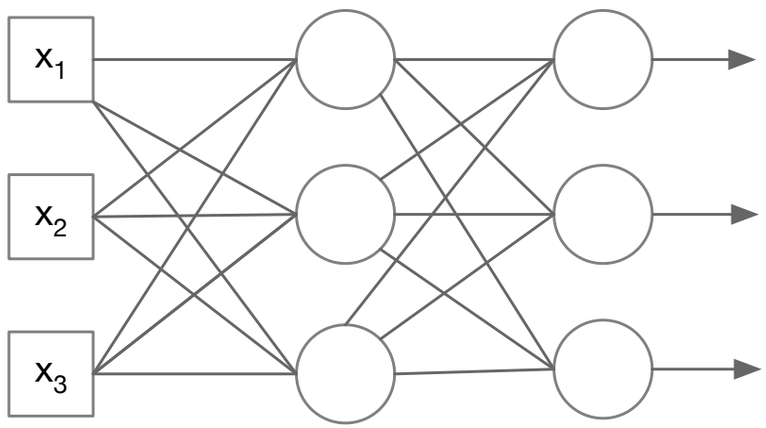




Model	# of Parameters
GPT-2	~1.5 billion
GPT-3	~175 billion
GPT-4	~1 trillion (rumored)
GPT-4o	Not disclosed

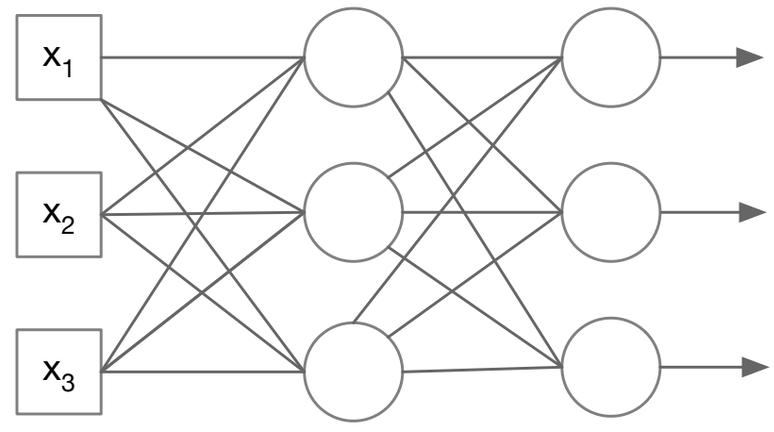
Classification

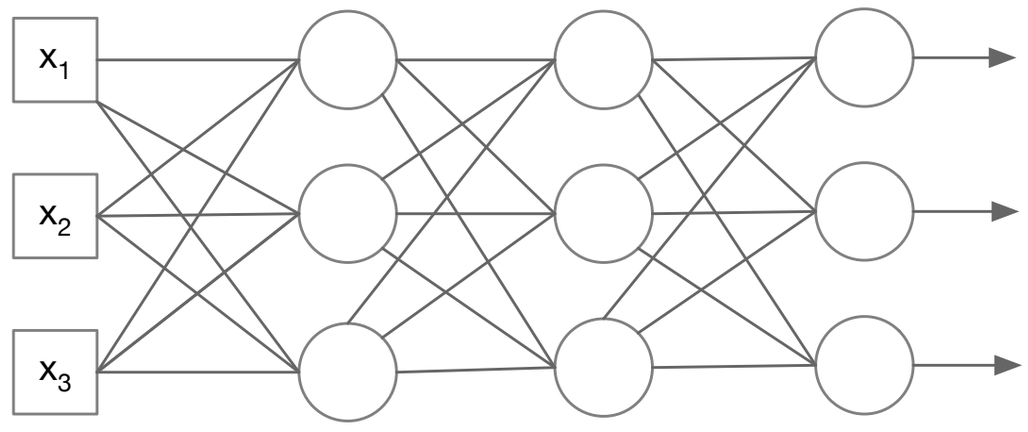
- Classification typically uses neural networks with 1 output node per class (label)
- So, for example, the iris dataset would have 3 output neurons
- The activation for a classification problem such as this would use softmax, normalizing the inputs so the outputs sum to 1, e.g.,
  [-2.5, 4.6, 3.1] -> ?
- In that case, each output would represent the probability that a given input belongs to that class



Network Size - Number of Layers

- How many layers should a network have?
- For simple problems (basic classification with linearly separable data*), 1-2 hidden layers will suffice
- For complex problems - image recognition, natural language processing, or playing games - dozens or even hundreds of layers, of different types, might be necessary





*in which a single line, for 2 features, or a single plane, for 3 features, etc., can separate the classes. This is very rare in real life datasets

Network Size - Number of Neurons

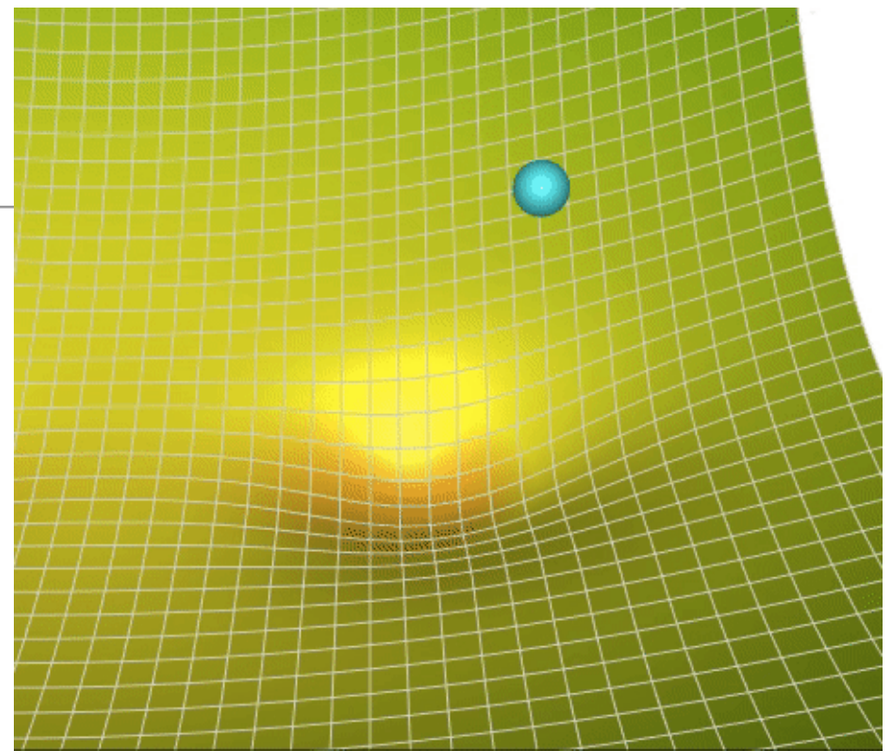
- A common *heuristic* is:
  - # hidden neurons = (# input features + # output neurons) / 2
- It comes down to trial-and-error
  - start small
  - use test datasets to evaluate accuracy

Training the Model

1. **Select a model architecture** — the number of hidden layers, the number of nodes per layer, and the h()'s
2. **Randomly initialize** all weights and biases.
3. **Run the training data through the model** (forward propagation) to compute the output.
4. **Calculate the error** — the difference between the predicted output and the actual (target) output as a function of the weights* - we call that the **loss function** $L(w_1, w_2, \ldots, w_n)$
5. **Use backpropagation** - compute how much each weight contributed to the error
   - It works **backwards** through the network — from output layer to input layer.
   - It uses the **chain rule** to compute **partial derivatives** of L( ) w.r.t. each weight -- $\partial L/\partial w_1$, $\partial L/\partial w_2$, ...
   - These derivatives tell us: if I tweak this weight a little, how will it affect the final error?
6. **Update the weights** using the gradient descent algorithm.
   - The "gradient" is a vector, $[\partial L/\partial w_1, \partial L/\partial w_2, \ldots]$, pointing in the direction of steepest *increase* in error.
   - To minimize error, we take a step in the *opposite* direction — the negative gradient
   - The size of the step is controlled by the learning rate (too big: you overshoot, too small: slow learning).
   - In short, $w_i = w_i - \partial L/\partial w_i$ * learningRate (for all weights)
7. **Repeat steps 3–6** until the error is acceptably low (note: it will never be exactly zero).

*and biases - but we can treat biases as just weights with an input of 1, so we'll ignore them henceforth

An Example in 3D



- Consider a 3D space, where the weights, w1 and w2, are on the x and y axes, and z represents the loss function L(w1,w2).
- We want to train the model, tweaking w1 and w2 until the blue ball ends up in the center, where L(w1,w2) is smallest
- The blue ball represents the current values for w1 and w2.
- The curved arrows show the partial derivatives:
  - $\partial L/\partial w_1$ - slope in the x direction
  - $\partial L/\partial w_2$ - slope in the y direction
- The black arrow points downhill - the negative direction of the slopes - showing the direction we'll update the weights to minimize loss
- At each iteration, we move closer to the minimum by updating the weights as shown
  - $w1 = w1 - \partial L/\partial w_1 * learningRate$
  - $w2 = w2 - \partial L/\partial w_2 * learningRate$
- We have shown this in 3D, but the same principle applies no matter how many weights are involved

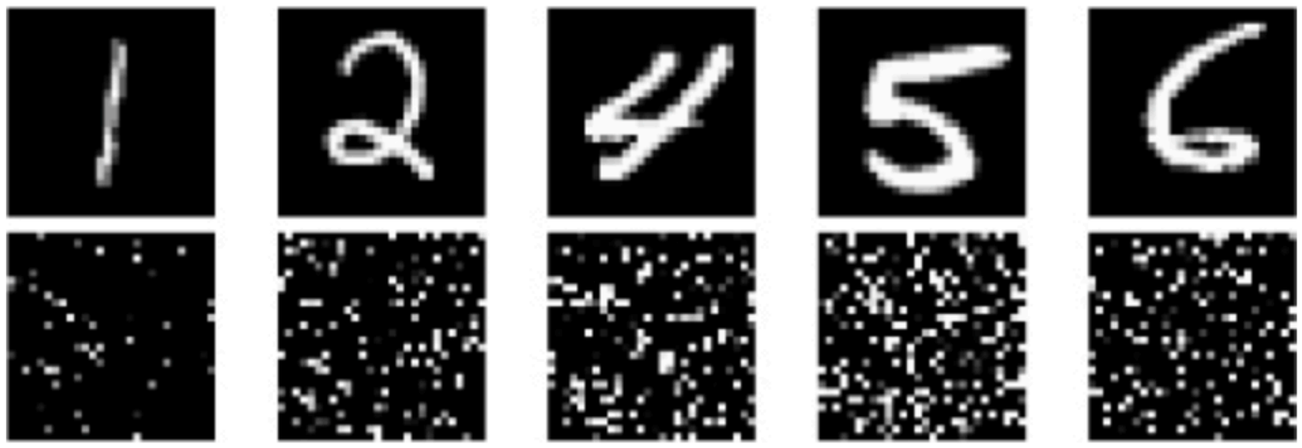
Computer Vision

- In traditional neural networks, the order in which features are presented in the training dataset does not matter

- For example, take the MNIST dataset and rerun it with the pixels moved to different locations

- The structure that *we* use to identify digits is gone, but the model learns in exactly the same way

- Convolutional Neural Networks (CNNs) do take structure into account -- they discover it, and then use it to allow computers to "see"



Convolutional Neural Networks

- CNNs are multi-layer neural networks that *learn* how to identify structure in the inputs
- Early layers learn simple geometric shapes - edges (horizontal, vertical, diagonal), corners, gradients, simple textures
- Mid layers learn parts of objects - simple shapes (L-shapes, circles, ovals, blobs), curves, angles, texture patterns (fur, grass, stripes)
- Deeper layers learn semantics - wheels, eyes, noses, handwritten digits, etc.
- Final layers: whole-object concepts - cat, 7, dog face
- It does so by applying a kernel to enhance various aspects of an image - a process known as convolution

You Say You Want a Convolution 🎶

- A convolution is easy to do, difficult to describe
- For **each** pixel in the image
  - Center a **kernel** - a small matrix (e.g., 3x3 or 5x5) - over it

1	3	1
0	0	0
-1	-3	-1

  - Calculate the sum of products of the overlapping values
  - That sum becomes a pixel in the convoluted image

52	27	2	44	44	31	28	38
53	43	20	77	89	24	14	68
35	63	68	61	81	48	26	57
49	22	15	50	39	40	50	62
29	87	15	25	71	36	21	27
65	62	77	59	41	38	82	64
74	23	14	41	68	62	55	19
79	63	59	82	42	38	55	66

X

1	3	1
0	0	0
-1	-3	-1

= 1(43) + 3(20) + 1(77) + ... + 1(50) = **63**

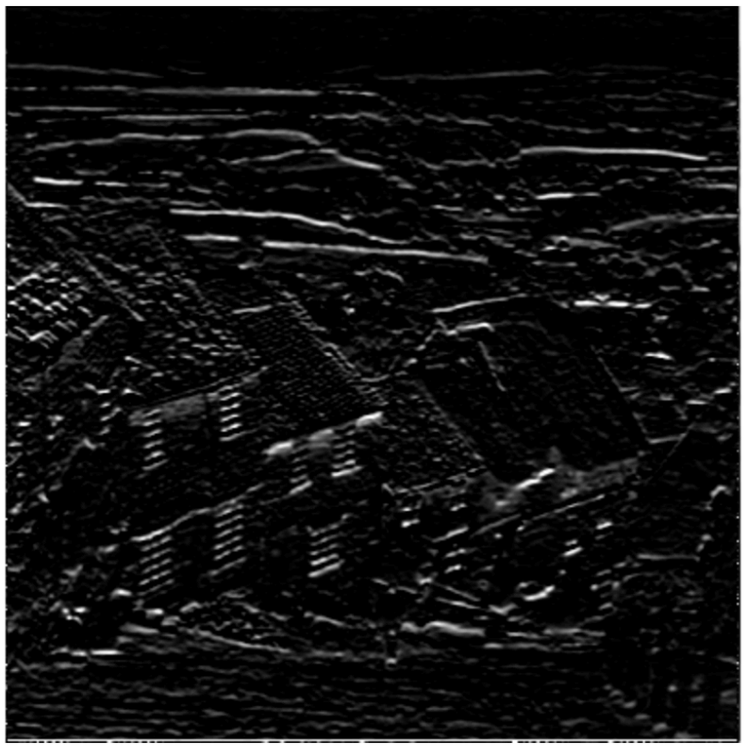
-202	-202	-180	-340	-368	-175	-134	-218
15	-157	-251	-154	-145	-86	-30	-55
33	72	**63**	136	161	-34	-118	-18
-6	-13	171	171	78	51	57	95
-88	-198	-235	-91	-13	-28	-96	-38
-71	148	51	-44	-33	-109	-120	-10
-43	1	30	-52	-26	26	79	21
245	157	106	205	307	309	246	112

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

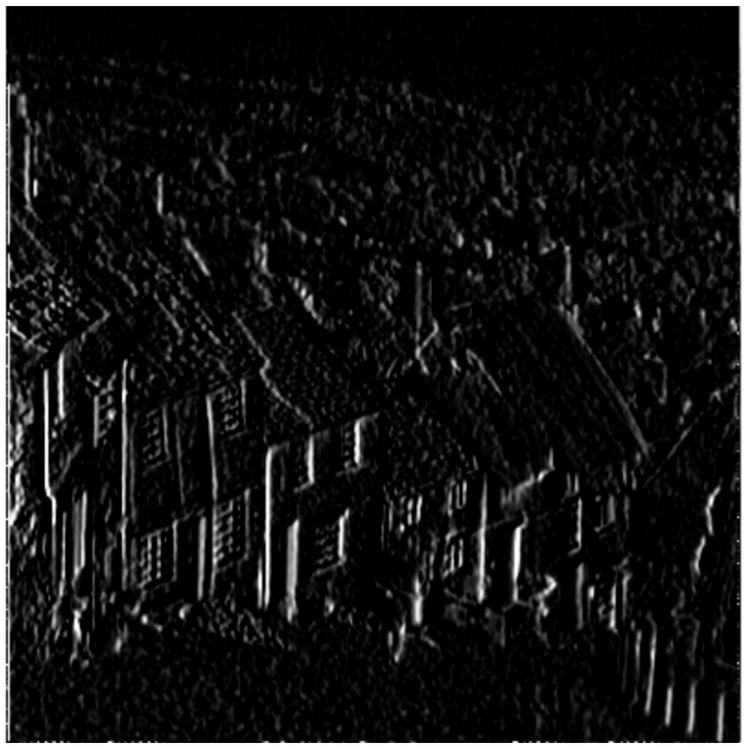
Identifying Structure with Kernels

- With an appropriate kernel, we can extract useful structures out of an image (in white). What shapes do you see?

0	0	0
0	1	0
0	0	0



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



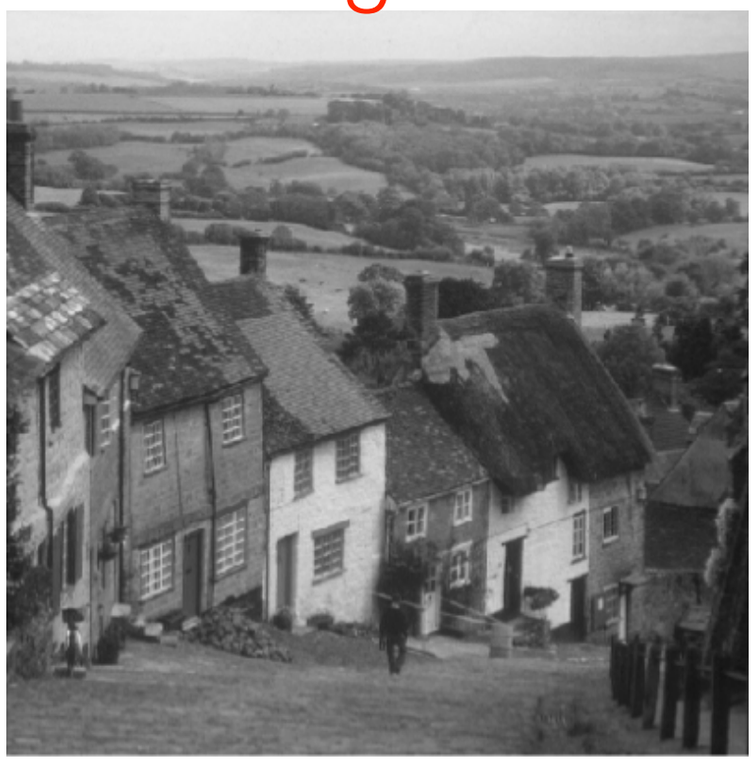
-1	0	1
-2	0	2
-1	0	1

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

Identifying Structure with Kernels

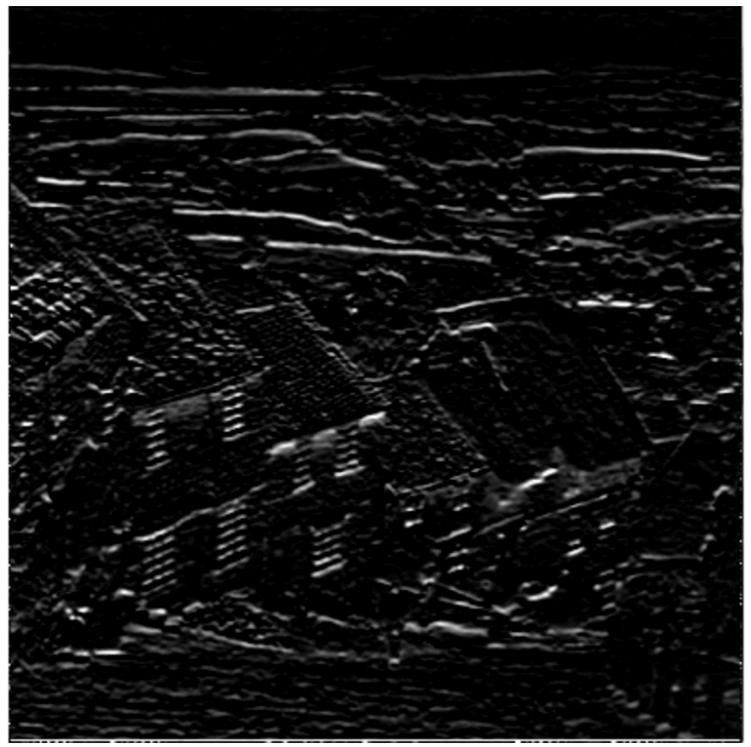
- By choosing kernels wisely, we can isolate structures that are germane to the problem of identifying images.

Original



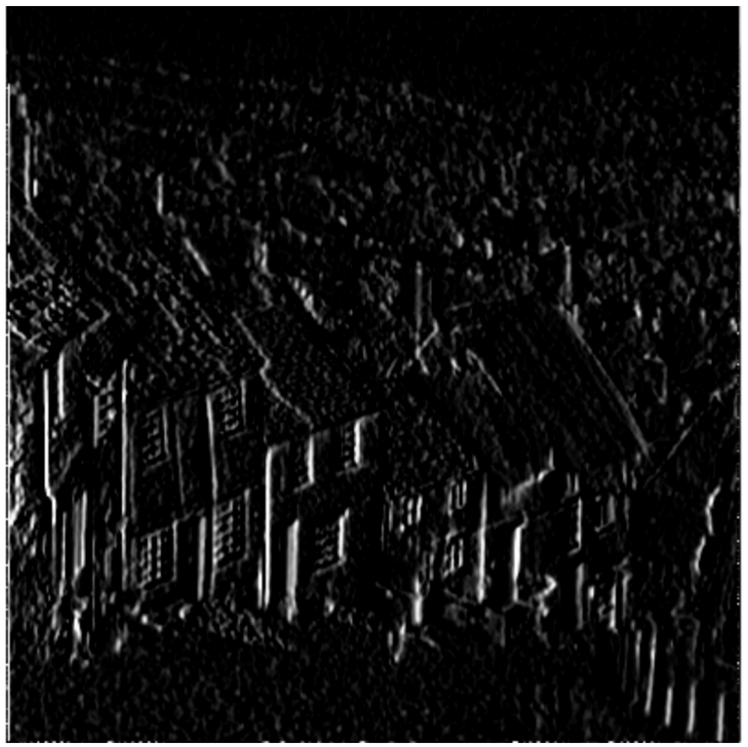
0	0	0
0	1	0
0	0	0

Horizontal lines



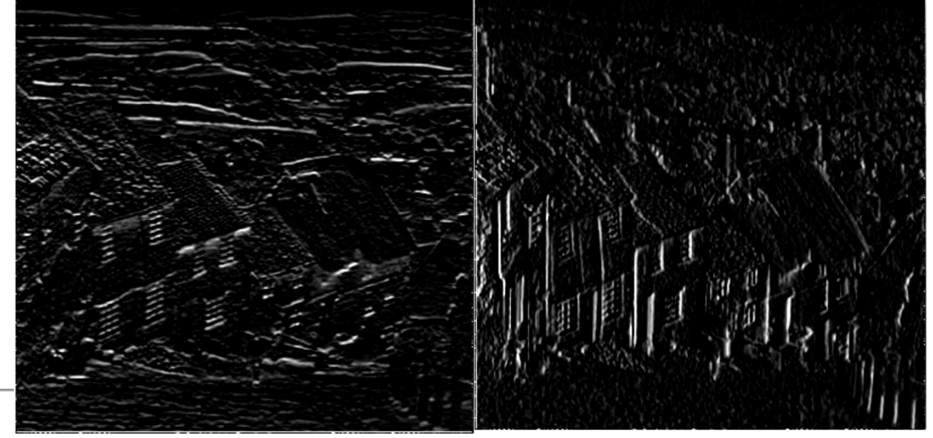
-1	-2	-1
0	0	0
1	2	1

Vertical lines



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

CNN Layer Types



- **Dense layers** - layers in which each set of neurons is connected to each set of outputs from the layer beneath it

- **Convolutional layers** - apply multiple kernels to their input to extract multiple outputs (aka feature sets), like the images above

  - values of the kernels are the weights for this layer

- **Pooling layers** - reduces the size of the image by sliding a 2x2 matrix over the image (without overlapping) and selecting the largest element

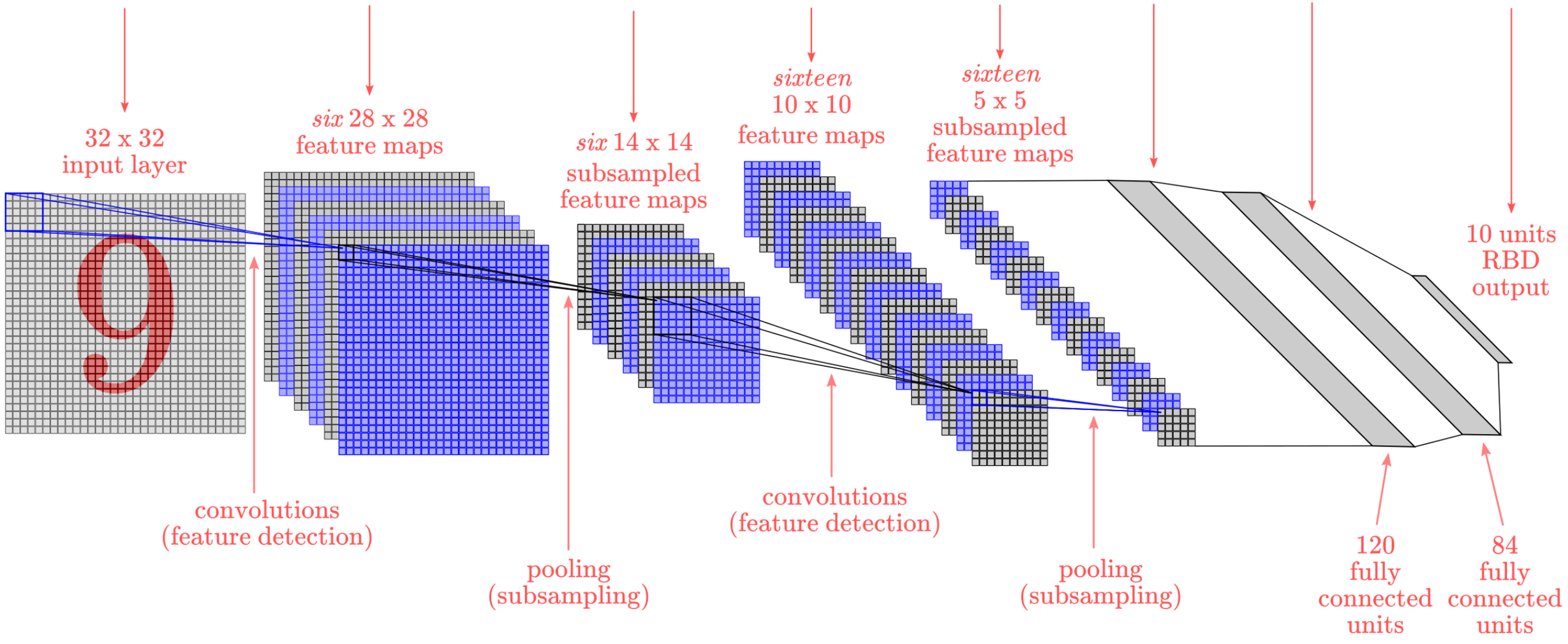
L0 - padded to 32x32 for convenience L1 - 6 filters of size 5x5, produces 6 feature maps of size 28x28 --

L2 - each 2x2 block from L1 becomes a single number (the average)

LeNet

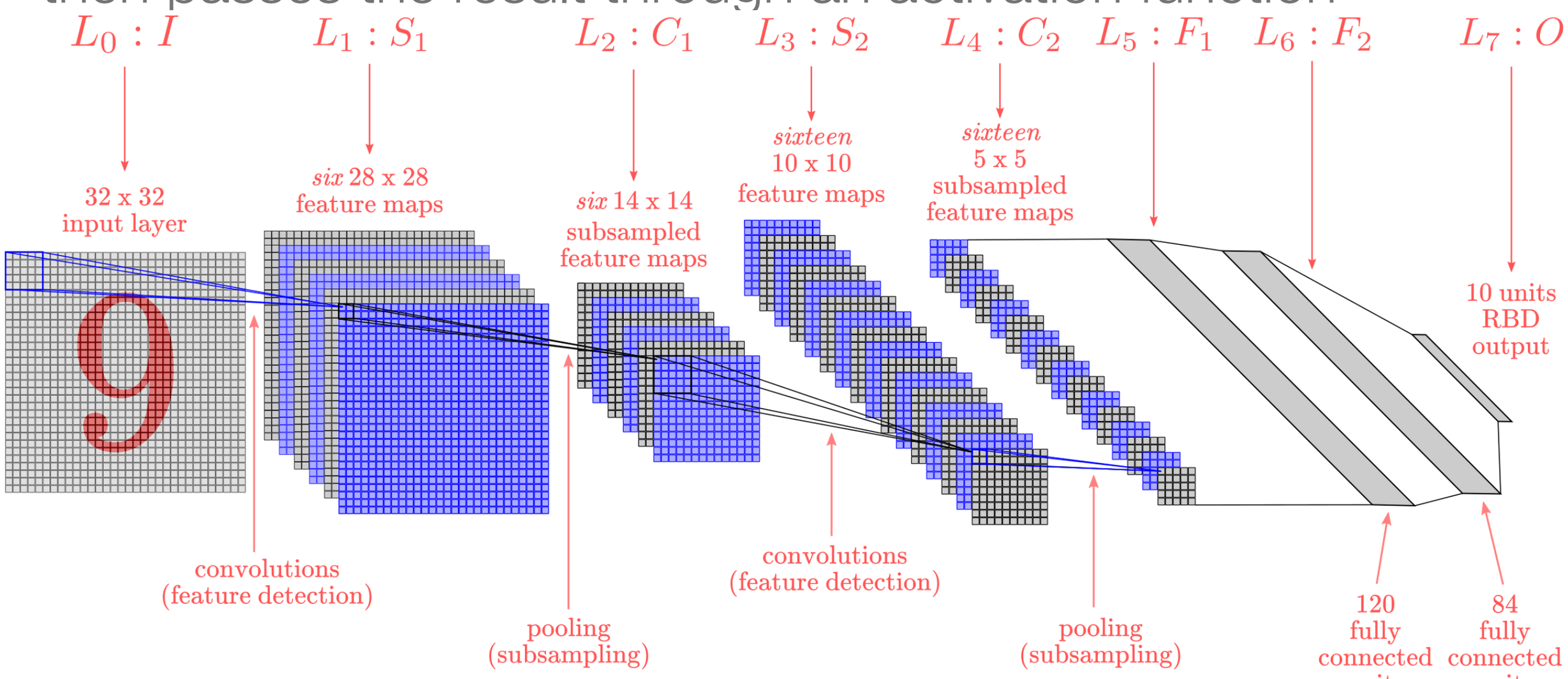
- LeNet is a classic CNN architecture, devised in the 1990s
- Training works as described before - forward propagation gets us to the classification; we get the error, and use backward propagation to determine how to adjust the weights (in this case, weights and kernels). Note that initially all the kernels values are randomly chosen.

$L_0 : I$ $L_1 : S_1$ $L_2 : C_1$ $L_3 : S_2$ $L_4 : C_2$ $L_5 : F_1$ $L_6 : F_2$ $L_7 : O$



Where are the Neurons?

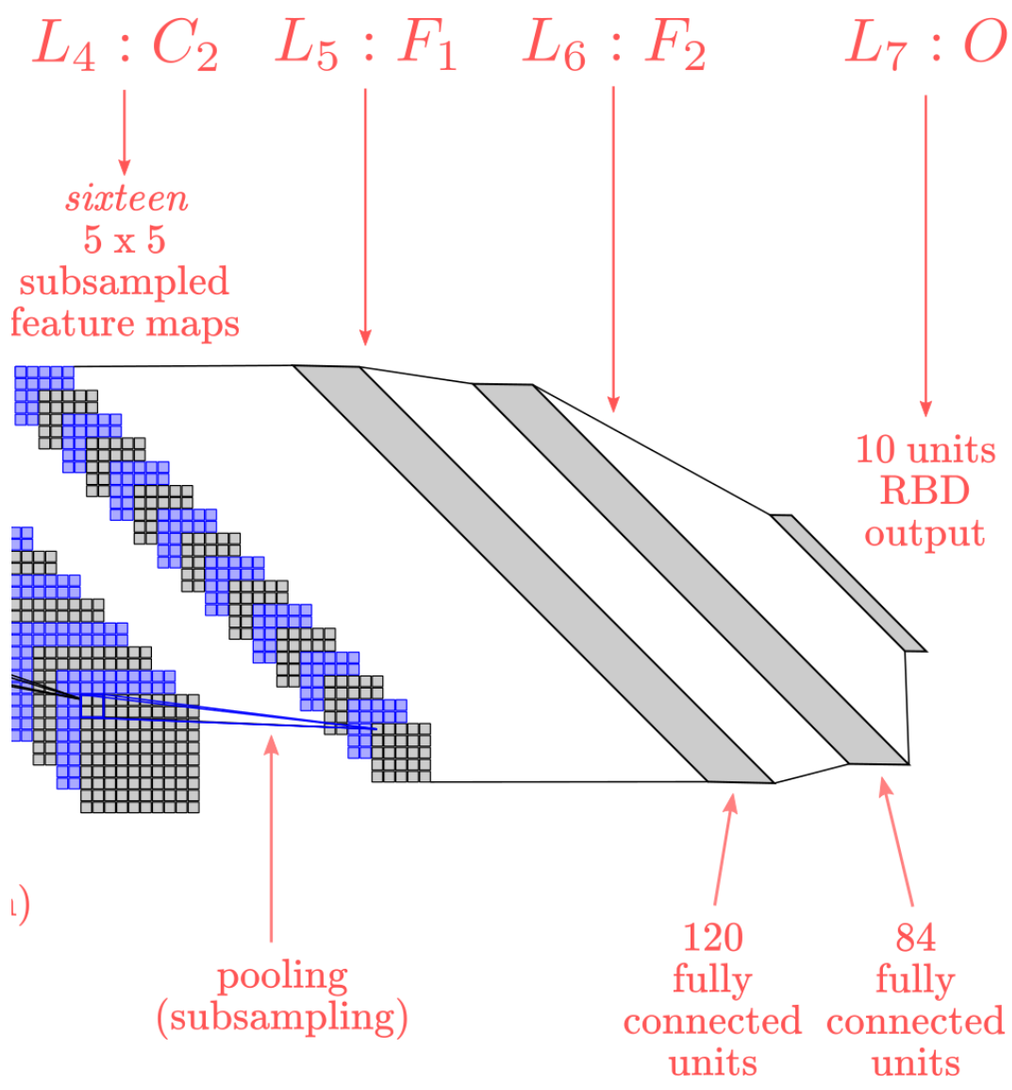
Each position in a feature map is a neuron - it receives input from a small region (like a 5x5) of the previous layer. It applies a dot product (weighted sum) using kernel widgets, then passes the result through an activation function



Flattening the Feature Maps: the L4-L5 Transition

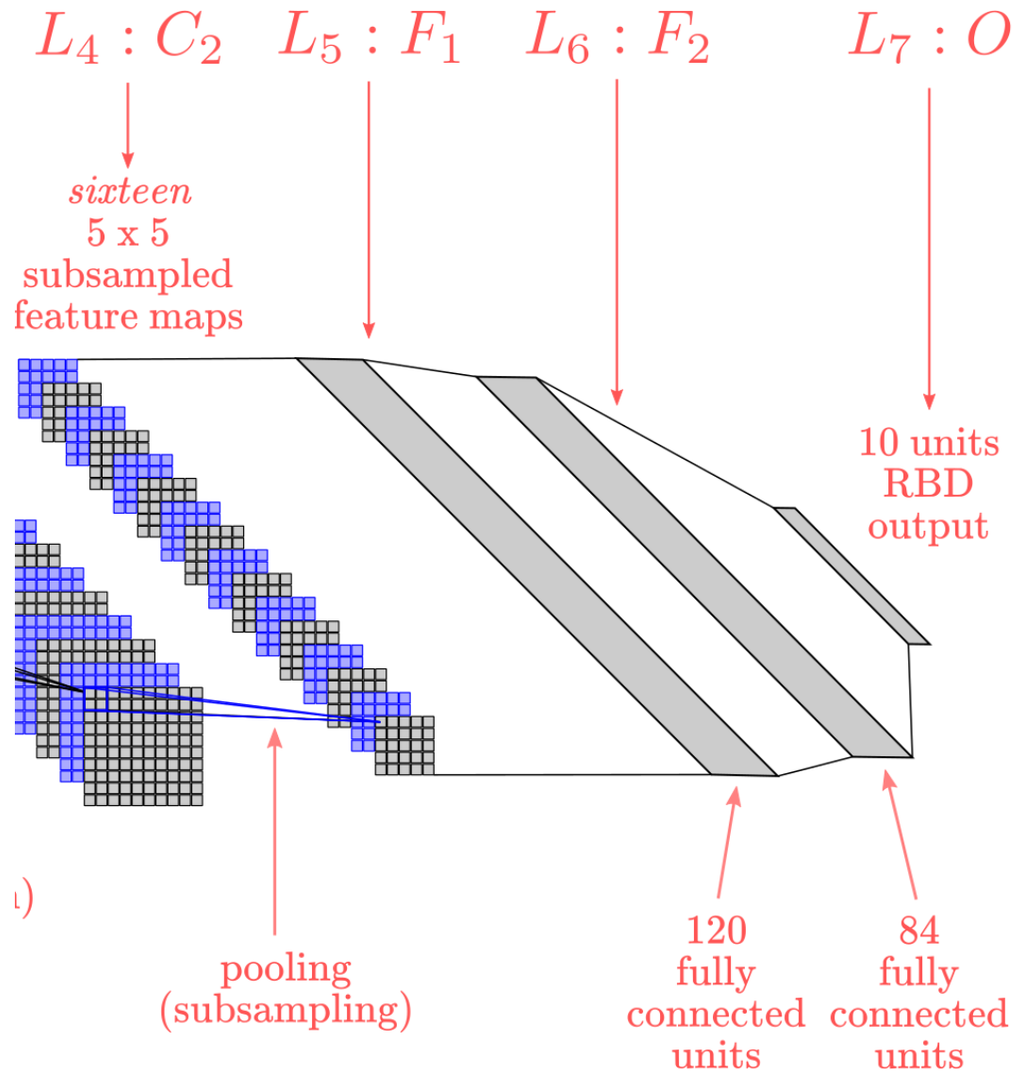
- The 16 x 25 = 400 neurons coming out of L4 are flattened into a 1D vector, [n1,... n400]

- Each is connected to all of the 120 neurons in L5 (fully-connected)



What's Happening at L7?

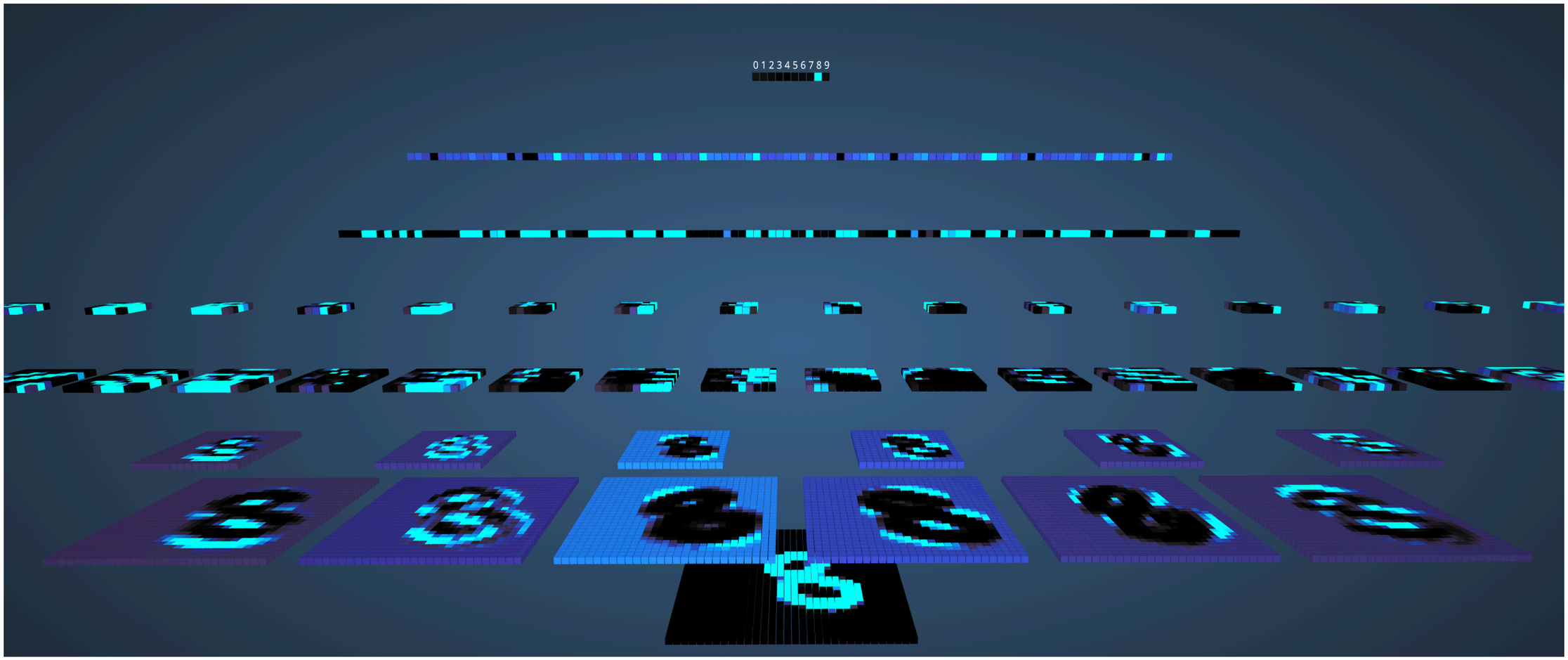
- At L7, there are 10 neurons [n0 ... n9]
- Their weights are random to start, e.g., n3 doesn't "know" it's for the digit 3 yet.
- For an input labeled "3", the *desired* output is [0,0,0,1,0,0,0,0,0,0]. The cost increases if the wrong neuron "lights up" (e.g., if output is [0.1, 0.2, 0.1, 0.4, 0.0, 0.1, 0.2, 0.0, 0.0, 0.0]
- The network updates weights so that the next time, n3 lights up more* when it sees a "3". Eventually n3 learns to respond strongly to features of a 3.
- Assignment is not enforced by the programmer: it emerges because of consistent training labels



*e.g., [0.1, 0.15, 0.1, 0.45, 0.0, 0.05, 0.15, 0.0, 0.0, 0.0]

A CNN, Visualized

- (Click on the image to get to a live version)



CNNs in Practice

- The architecture of a CNN is decided by the person who builds the model

- Architecture includes the following:

Component	Example (VGG16)
🧱 Number of layers	16 weight layers (13 conv + 3 dense)
🧠 Layer types and order	Conv → ReLU → Conv → ReLU → MaxPool →
🔍 Kernel sizes	3×3 for all conv layers in VGG16
🔢 Number of filters per	Increases from 64 → 128 → 256 → 512 → 512
⬇️ Pooling strategy	Max pooling, 2×2, stride 2
🔢 Dense layer sizes	4096, 4096, and 1000 (for 1000-class ImageNet)
🎯 Output layer	Softmax activation for classification

CNNs in Practice

- Developing software, you will likely rely on a pre-trained model

Model Name	**Purpose**	**Example**
mobilenet	Image classification	cat, dog, car, etc.
coco-ssd	Object detection	Detect bounding boxes in images
face-landmarks-detection	Facial features tracking	Eyes, nose, mouth
pose-detection	Pose estimation	Identify body keypoints
toxicity	NLP model to classify	Profanity, insults, etc.
universal-sentence-encoder	Text embedding	Convert sentences to numerical vectors
qna	Question answering	Answer questions from given context
handpose	Hand tracking	Detect finger positions and gestures
speech-commands	Audio classification	Recognize "yes", "no", "stop", etc.
blazeface	Face detection	Fast face bounding box detection

References

- https://youtu.be/i62czvwDlsw?si=g9nAU_JZRrvlSwEQ
- https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/
- https://setosa.io/ev/image-kernels/
- https://pabloinsente.github.io/the-convolutional-network