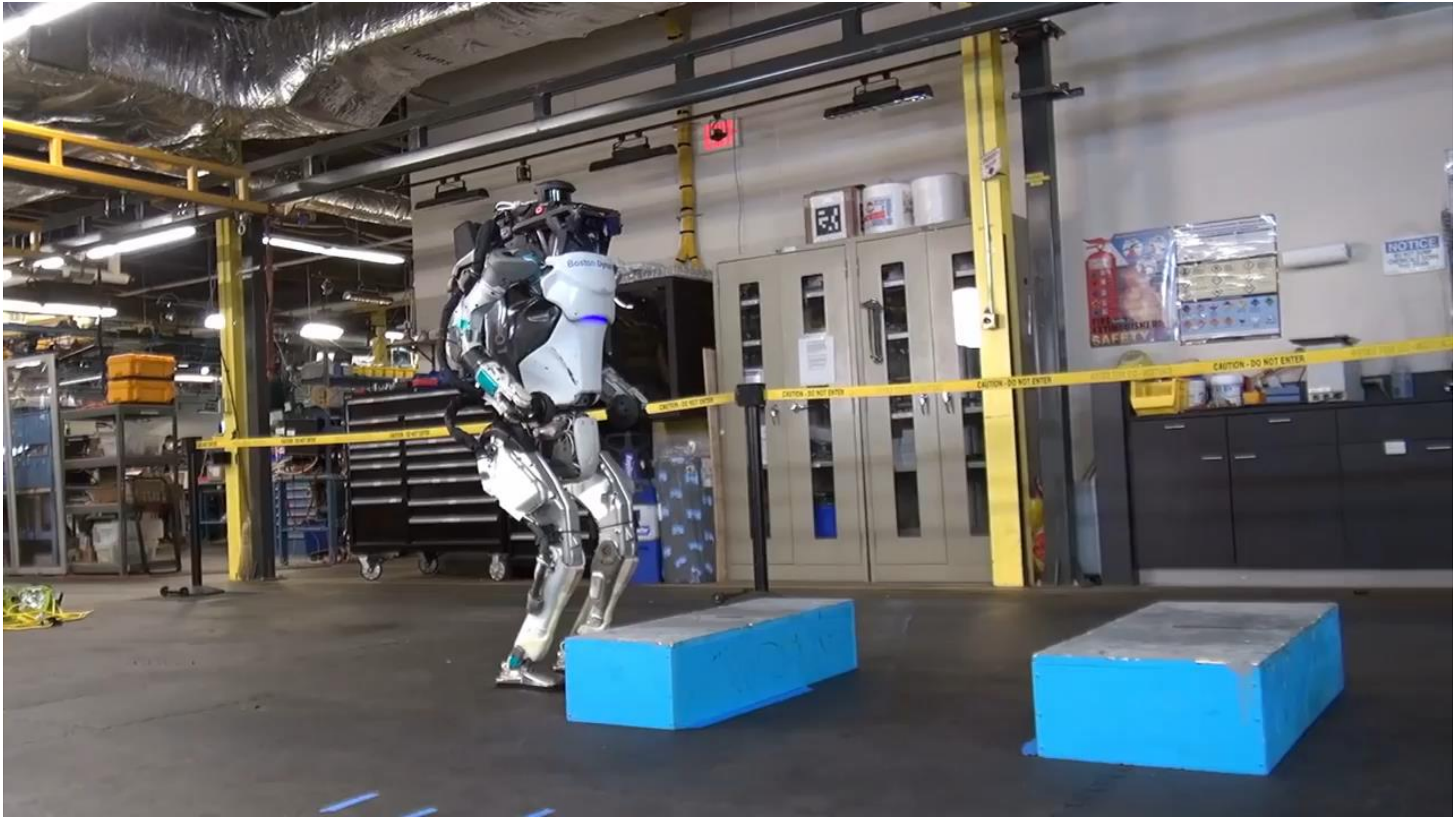
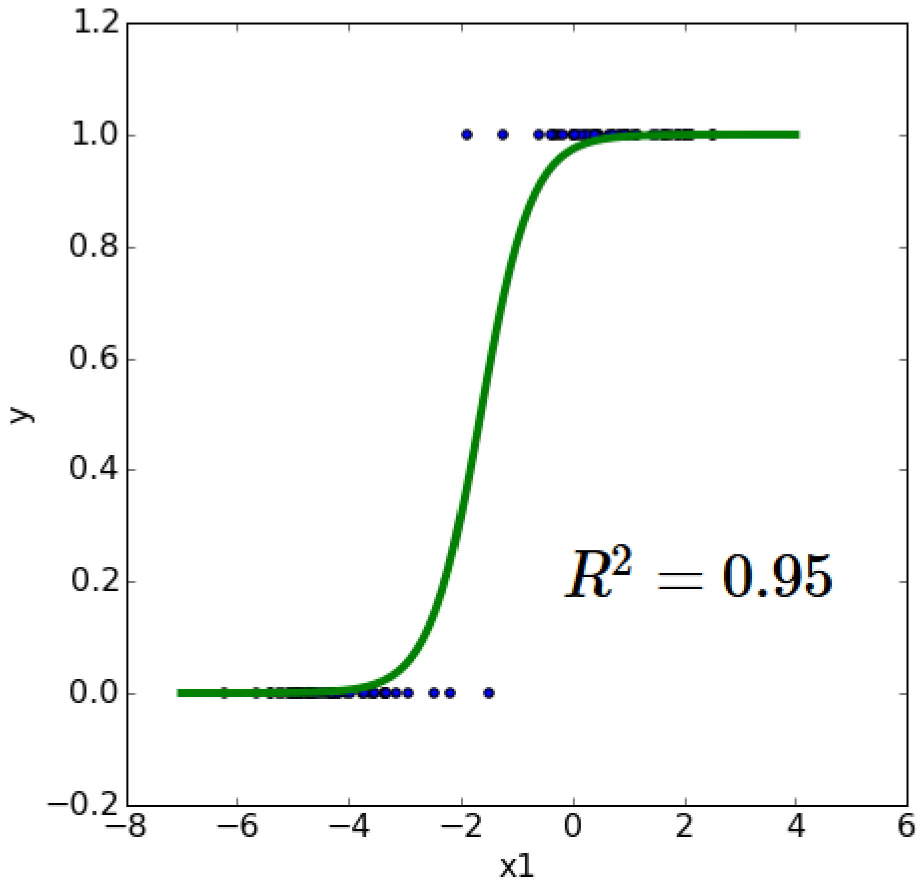

CAUTION - DO NOT ENTER
NOTICE

logistic regression



We need to make

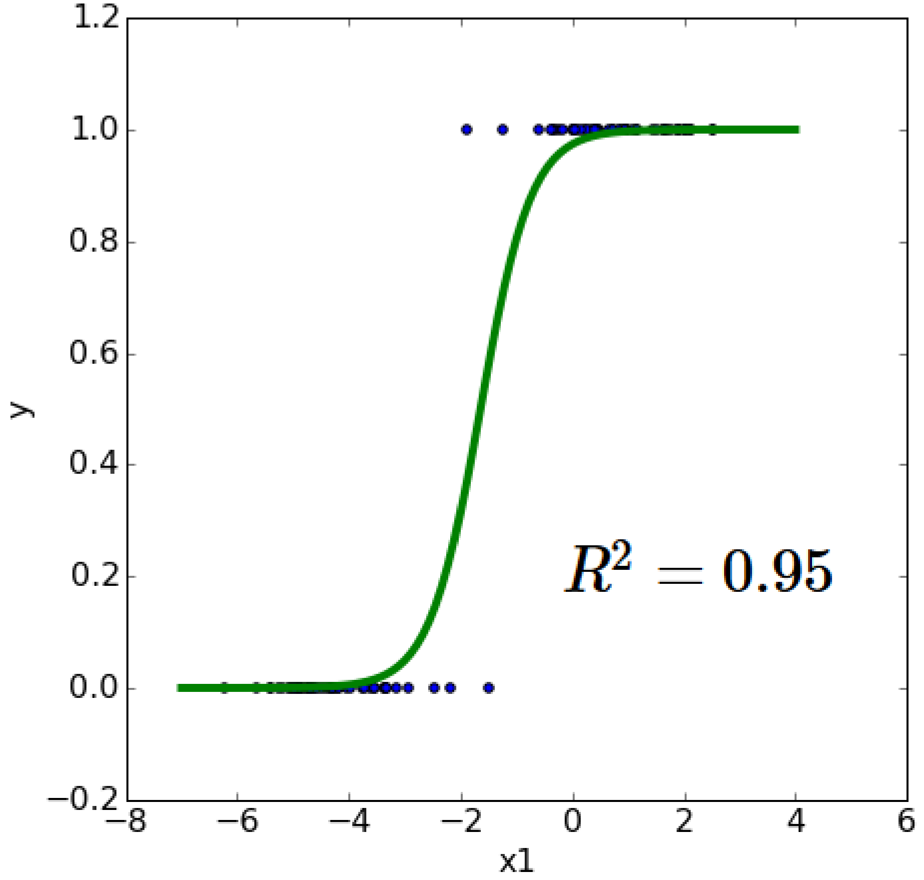
assumptions *linearly separable*

about the

model *logistic model*

that generated the data.

logistic regression: logistic model



$$f(x, \theta) = g(\theta_0 + \theta_1 x_1)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

logistic regression: cost function

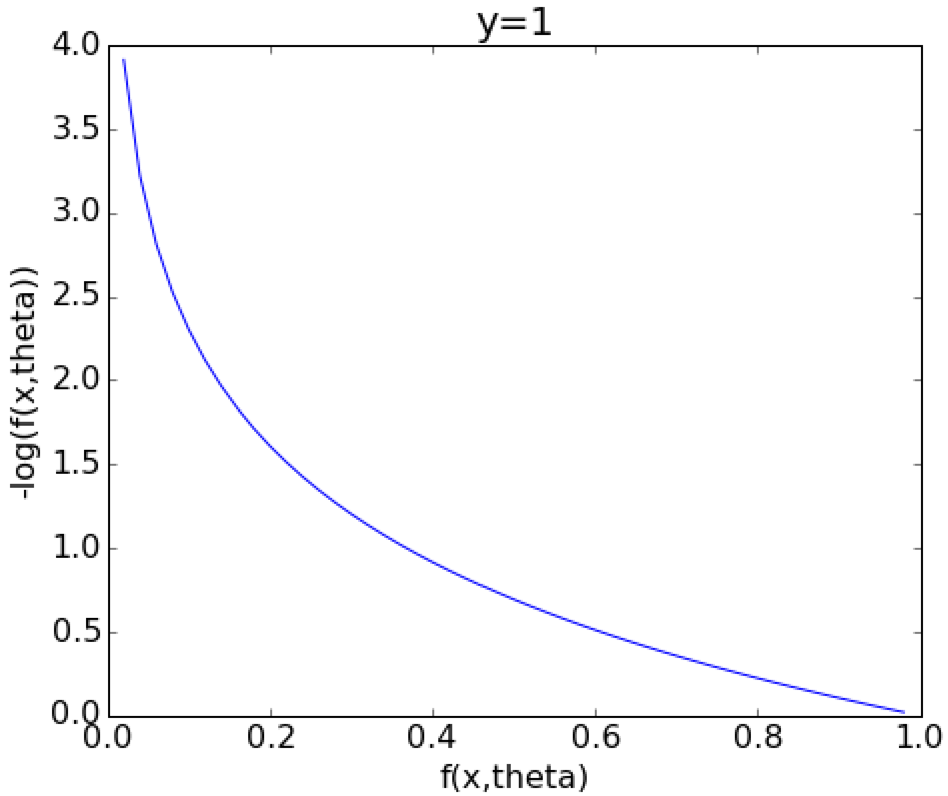
$$J(\theta) = -[\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)},\theta)) + (1 - y^{(i)}) log(1 - f(x^{(i)},\theta))]$$

We know that $y^{(i)}$ is either 0 or 1. If $y^{(i)} = 1$ then the cost function $J(\theta)$ is incremented by $-log(f(x^{(i)},\theta))$.

Similarly, if $y^{(i)} = 0$ then the cost function $J(\theta)$ is incremented by $-log(1 - f(x^{(i)},\theta))$.

logistic regression: cost function

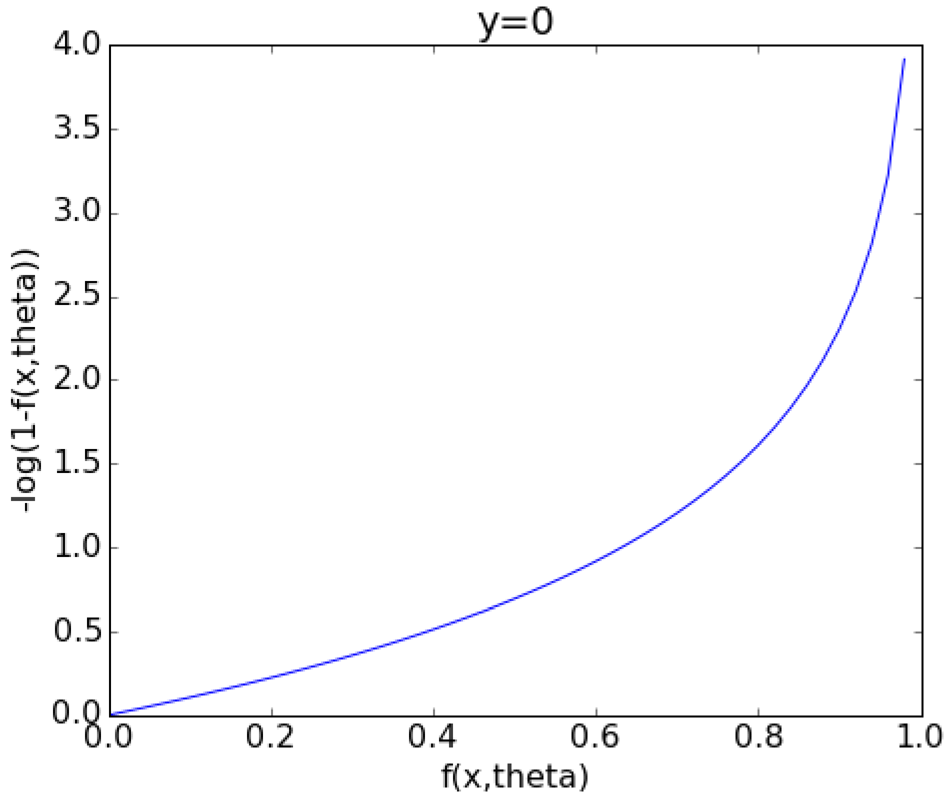
We know that $y^{(i)}$ is either 0 or 1. If $y^{(i)} = 1$ then the cost function $J(\theta)$ is incremented by $-log(f(x^{(i)}, \theta))$.



logistic regression: cost function

Similarly, if $y^{(i)} = 0$ then the cost function $J(\theta)$ is incremented by

$-log(1 - f(x^{(i)}, \theta))$.



logistic regression

Fit a logistic model

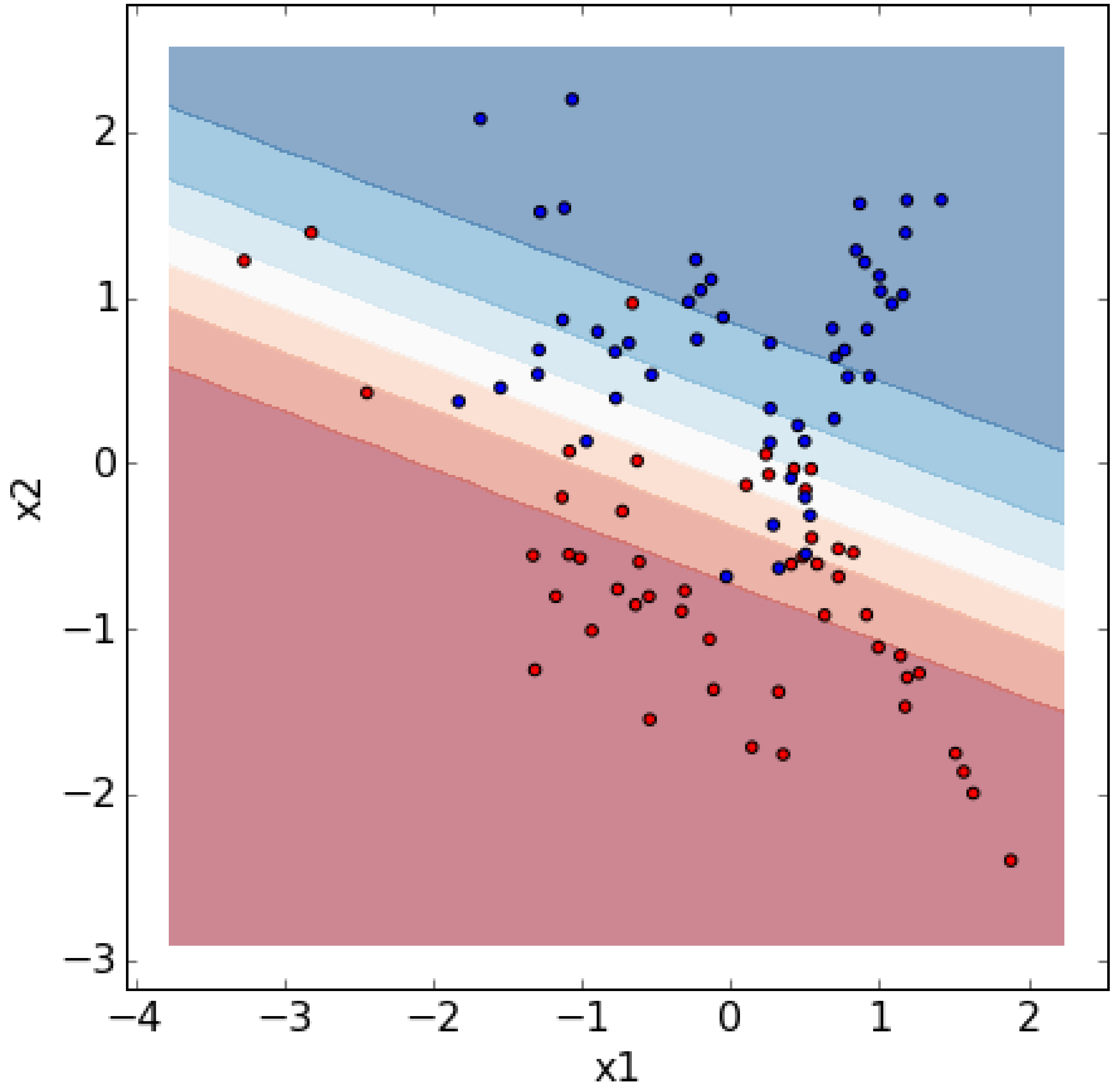
$$f(x,\theta) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m) = g(\theta' x)$$

to the data set such that the cost function

$$J(\theta) = -[\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)},\theta)) + (1 - y^{(i)}) log(1 - f(x^{(i)},\theta))]$$

is minimal using gradient descent

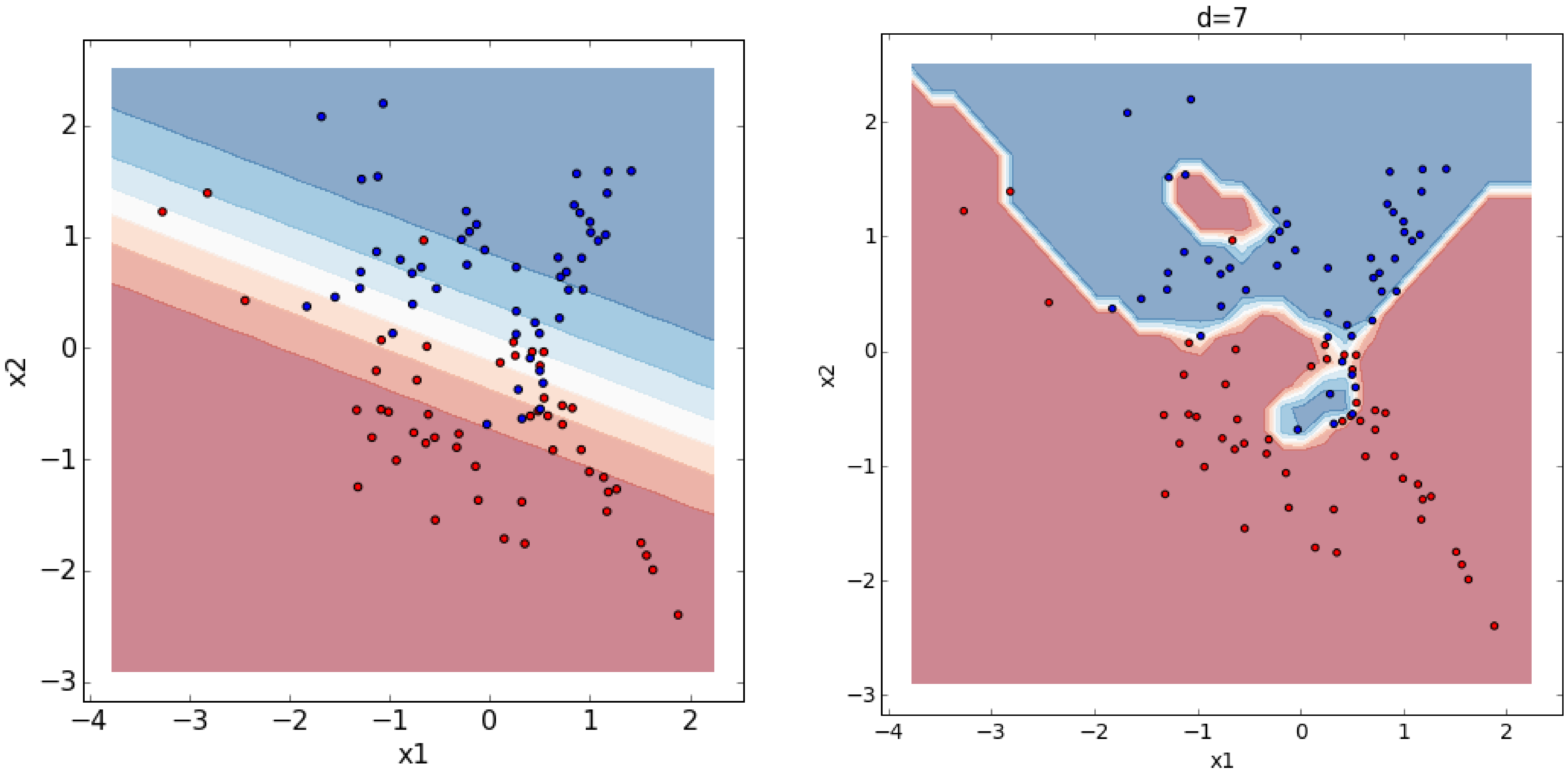
$$\theta_j := \theta_j - \alpha \frac{1}{n}\sum_{i=1}^{n}(f(x^{(i)},\theta) - y^{(i)}) x_j^{(i)}$$



2
1
0
−1
−2
−3
x2
−4
−3
−2
−1
0
1
2
x1

Let’s get real!

7. **Logistic regression**
8. **Non-linear transformations**


d=7
x1
x2
x1
x2

regularized logistic regression

$$f(x,\theta) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m)$$

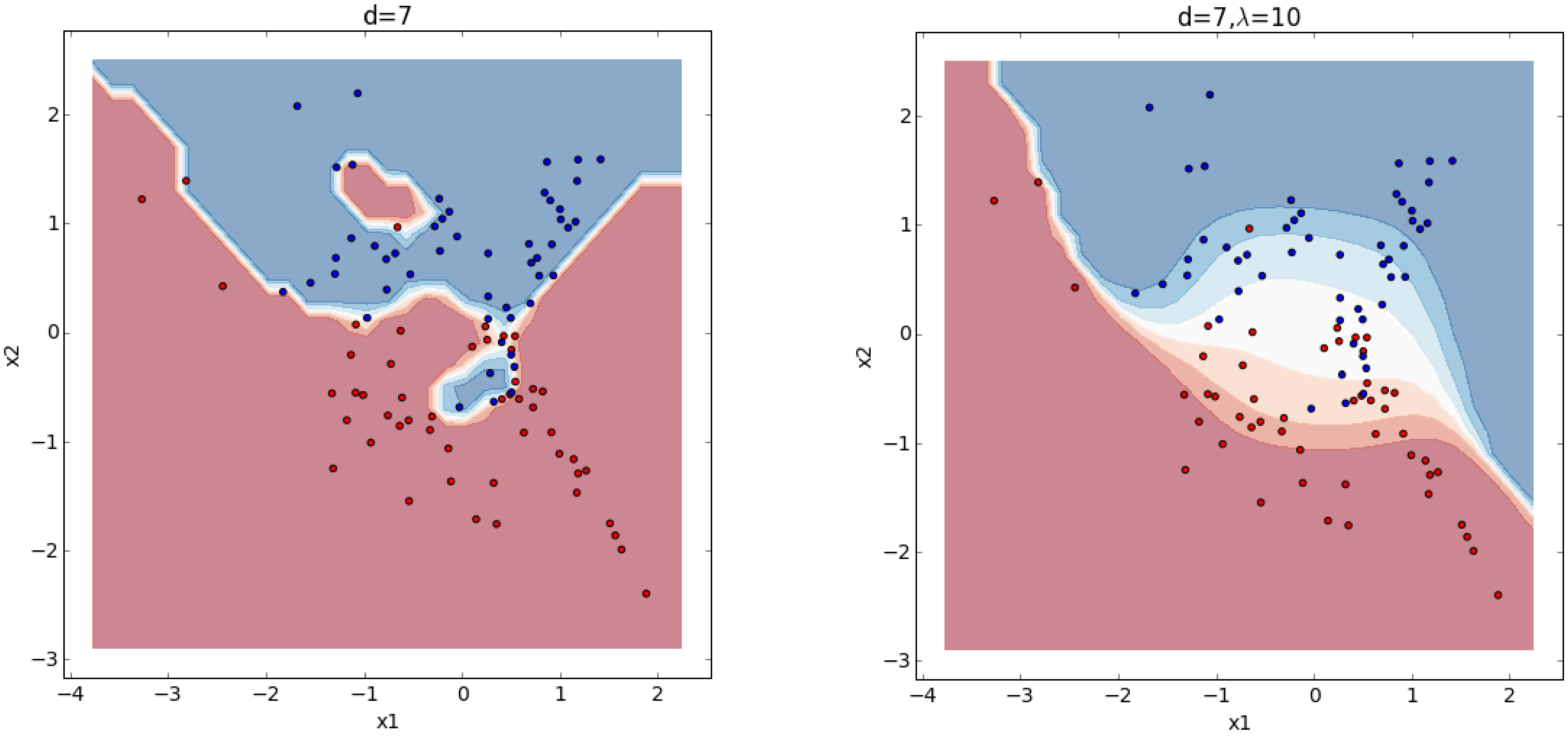
cost function

$$J(\theta) = -[\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)},\theta)) + (1-y^{(i)}) log(1-f(x^{(i)},\theta))]$$

$$J(\theta) = -[\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)},\theta)) + (1-y^{(i)}) log(1-f(x^{(i)},\theta))] + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

regularized cost function

regularized logistic regression



support vector machines

Fit a linear model

$$f(x,\theta) = \theta' x$$

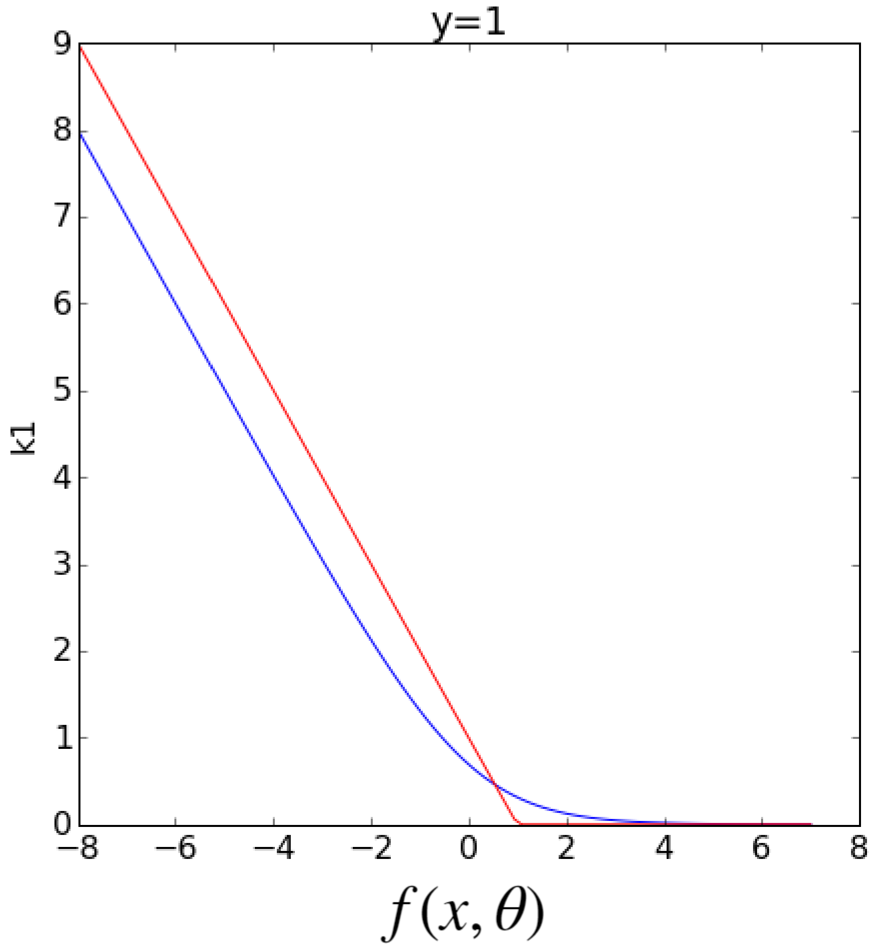
such that

$$J(\theta) = \left[C \sum_{i=1}^{n} y^{(i)} k_1(\theta' x^{(i)}) + (1 - y^{(i)}) k_0(\theta' x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$

with $k_1(\theta' x) = max(0, 1 - \theta' x)$ and $k_0(\theta' x) = max(0, 1 + \theta' x)$

is minimized.

support vector machines

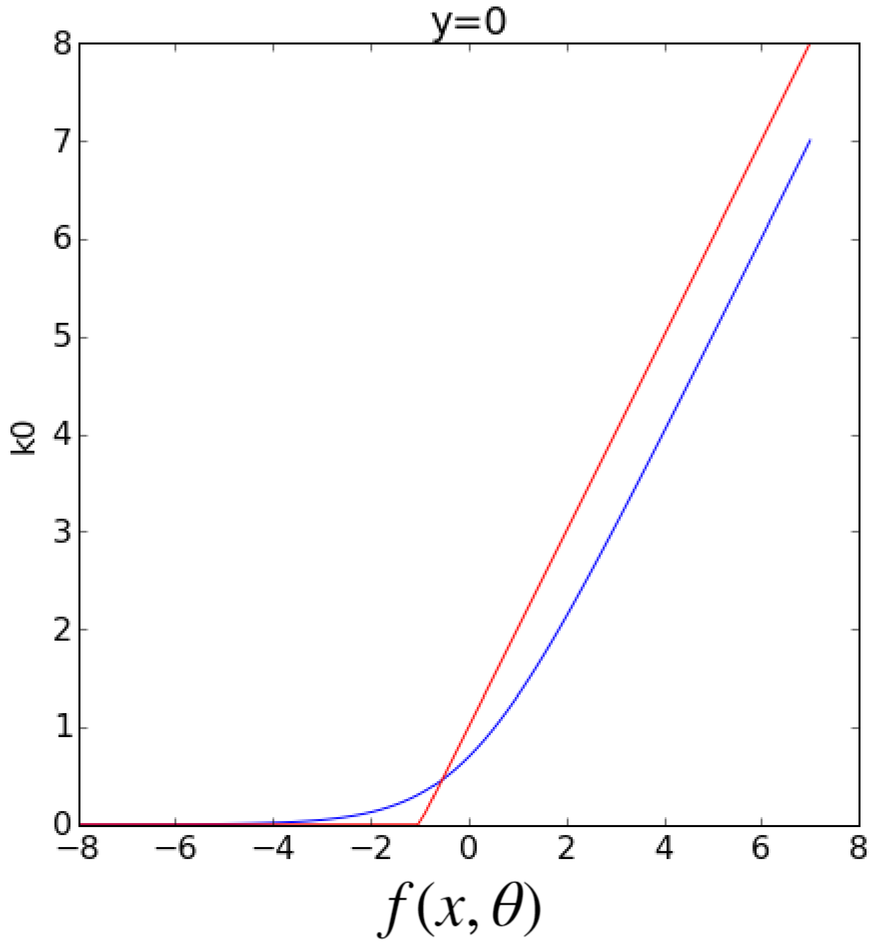


- replace cost function by piecewise linear function

- if y = 1 then the contribution to the cost is

$$k_1(f(x,\theta)) = max(0, 1 - f(x,\theta))$$

support vector machines



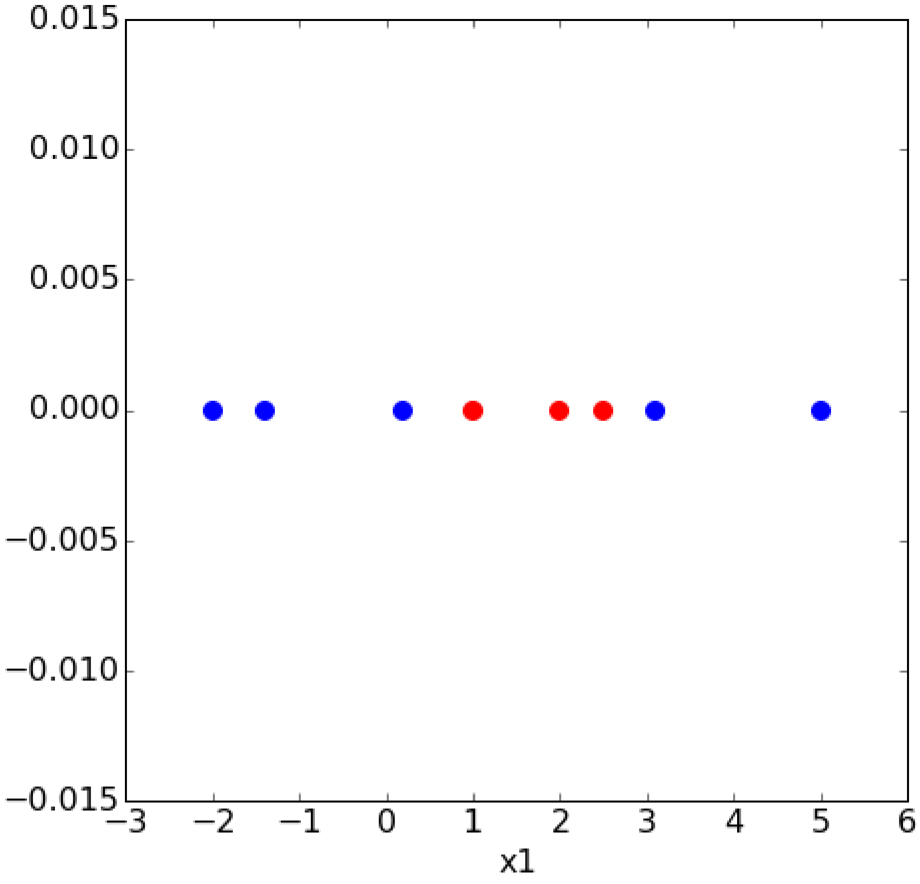
- replace cost function by piecewise linear function
- if $y = 1$ then the contribution to the cost is

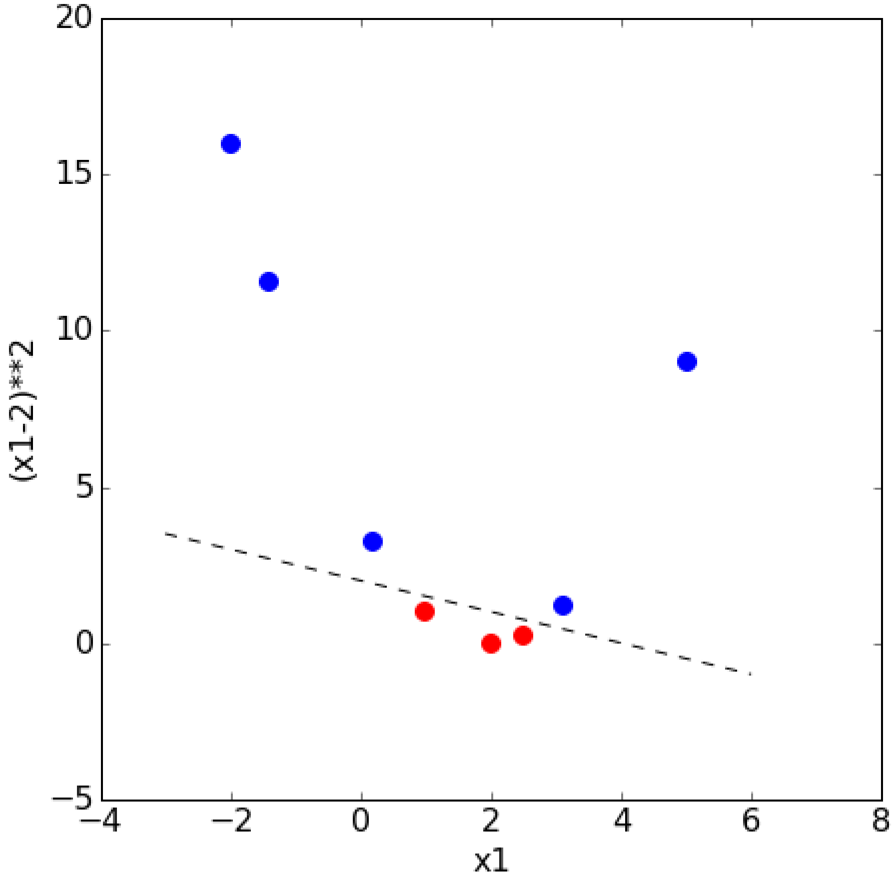
$$k_1(f(x, \theta)) = max(0, 1 - f(x, \theta))$$

- if $y = 0$ then the contribution to the cost is

$$k_0(f(x, \theta)) = max(0, 1 + f(x, \theta))$$

kernel support vector machines





kernel support vector machines

SVMs can also be formulated as a linear function of the samples (dual form) instead of the features as

$$f(x,\theta) = \sum_{i=1}^{n} \theta_i (x \cdot x^{(i)}) + \theta_0$$

that can be reformulated as a non-linear function using what is know as a kernel function

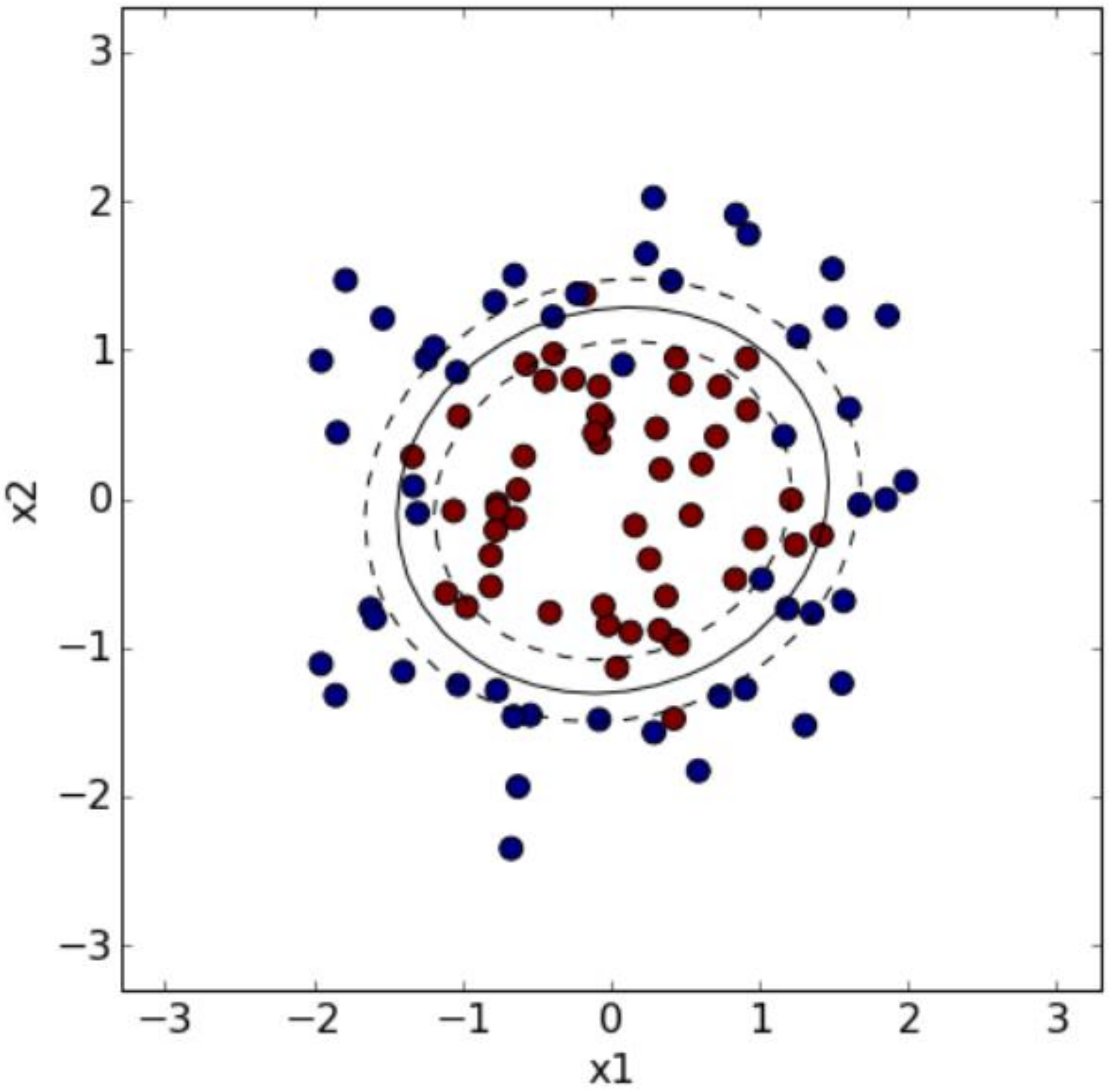
$$K(x^{(i)}, x^{(j)})$$

to become

$$f(x,\theta) = \sum_{i=1}^{n} \theta_i K(x, x^{(i)}) + \theta_0$$

The data points $x^{(i)}$ for which $\theta_i > 0$ are called the support vectors.

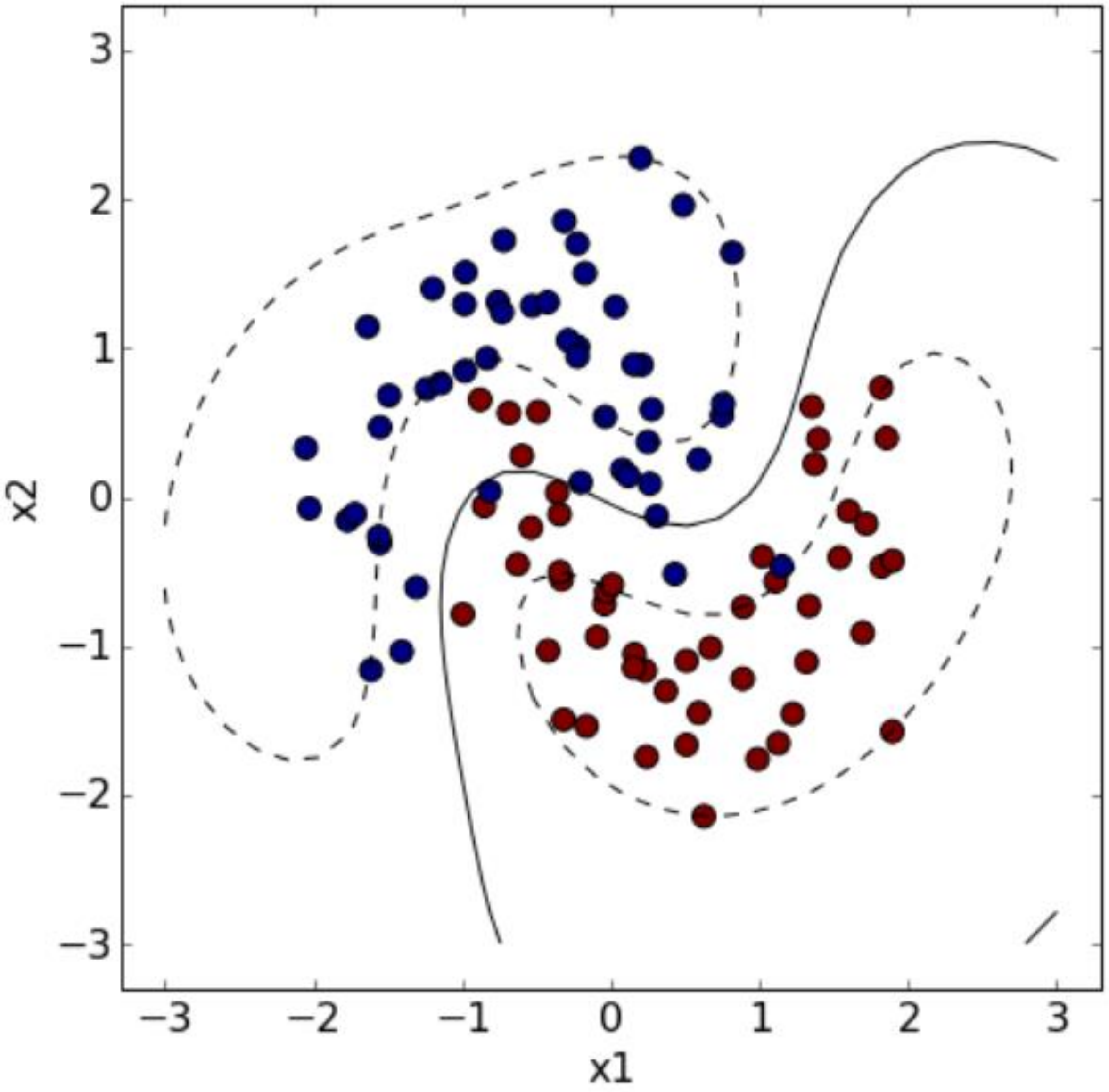
kernel support vector machines



$$f(x,\theta) = \sum_{i=1}^{n} \theta_i K(x, x^{(i)}) + \theta_0$$

$$K(x^{(i)}, x^{(j)}) = (x^{(i)} \cdot x^{(j)} + c)^d$$

kernel support vector machines



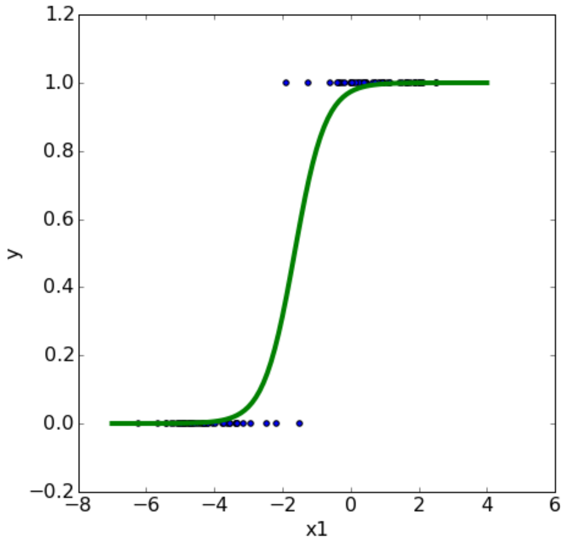
$$f(x, \theta) = \sum_{i=1}^{n} \theta_i K(x, x^{(i)}) + \theta_0$$

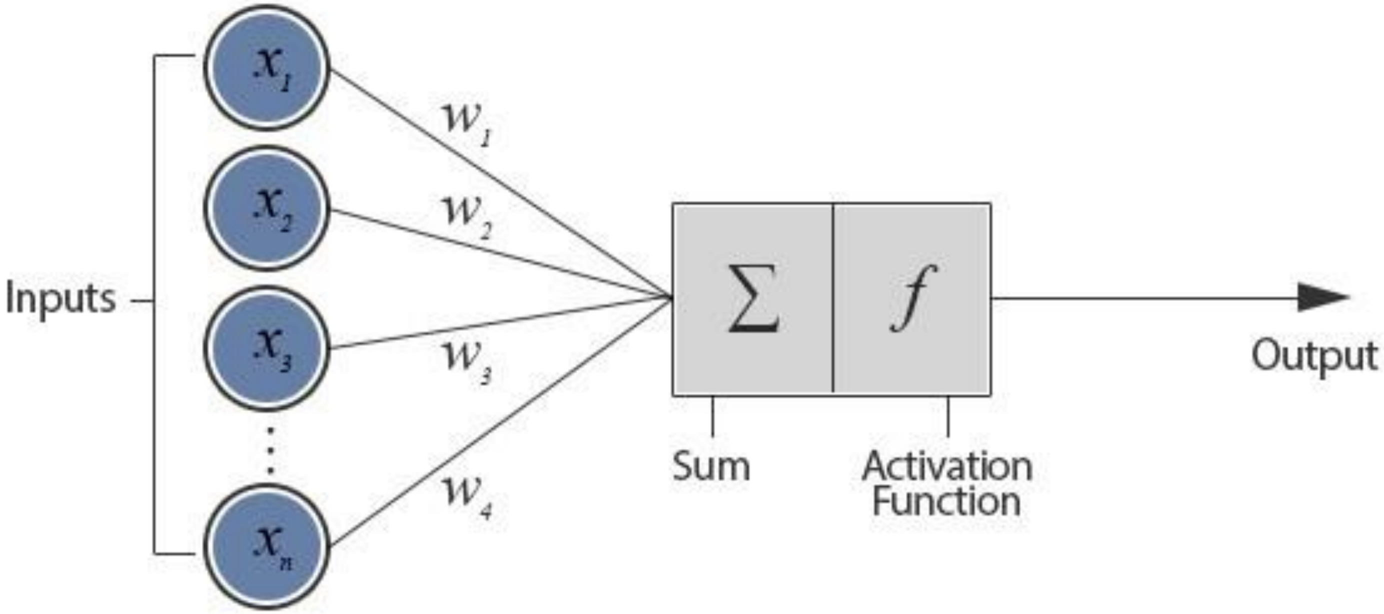
$$K(x^{(i)}, x^{(j)}) = \exp\left[-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\right]$$

Let’s get real!

9. Support Vector Machines

$$f(x,\theta) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m) = g(\theta' x)$$



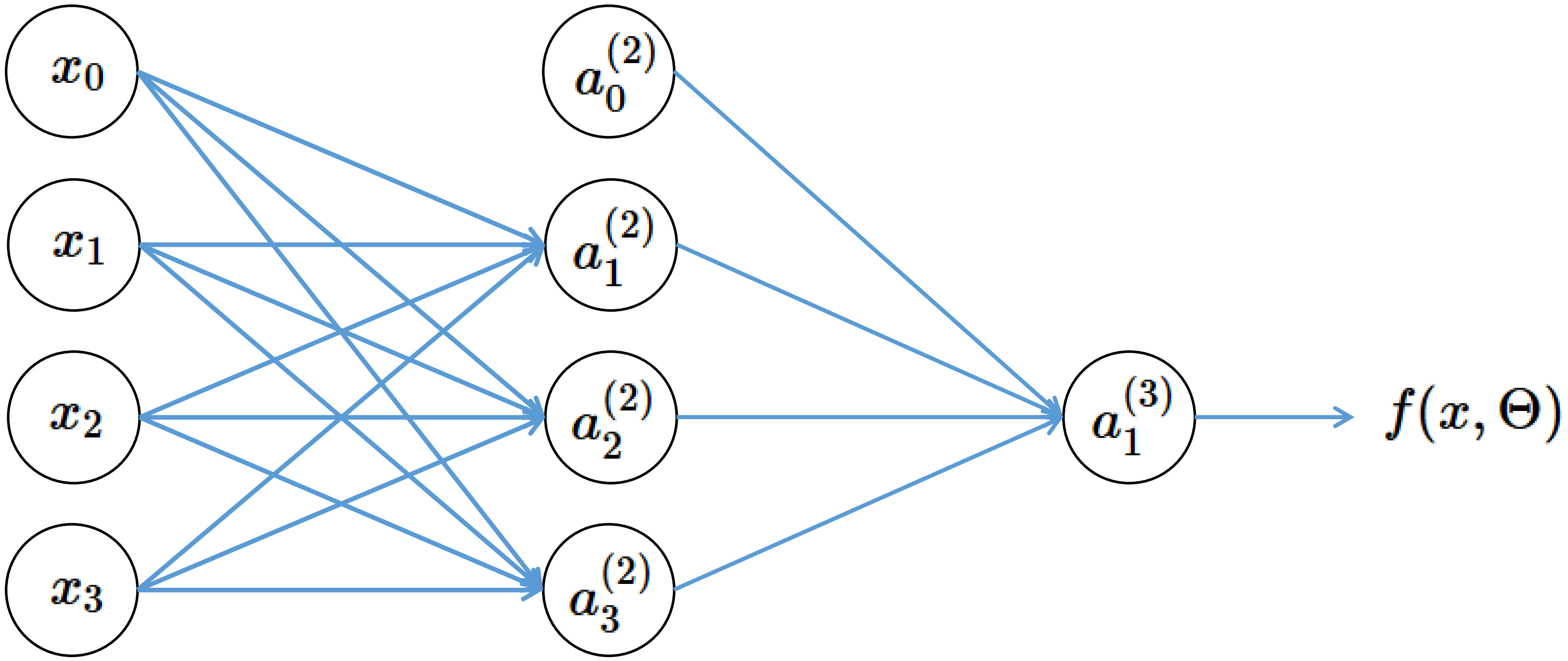


Model: $f(x,\Theta) = g(\Theta_{10}^{(2)} a_0 + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$



Cost function logistic regression:

$$J(\theta) = -[\frac{1}{m}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)},\theta) + (1-y^{(i)}) log(1-f(x^{(i)},\theta))] + \frac{\lambda}{2n}\sum_{j=1}^{n}\theta^2$$

Cost function feedforward neural network:

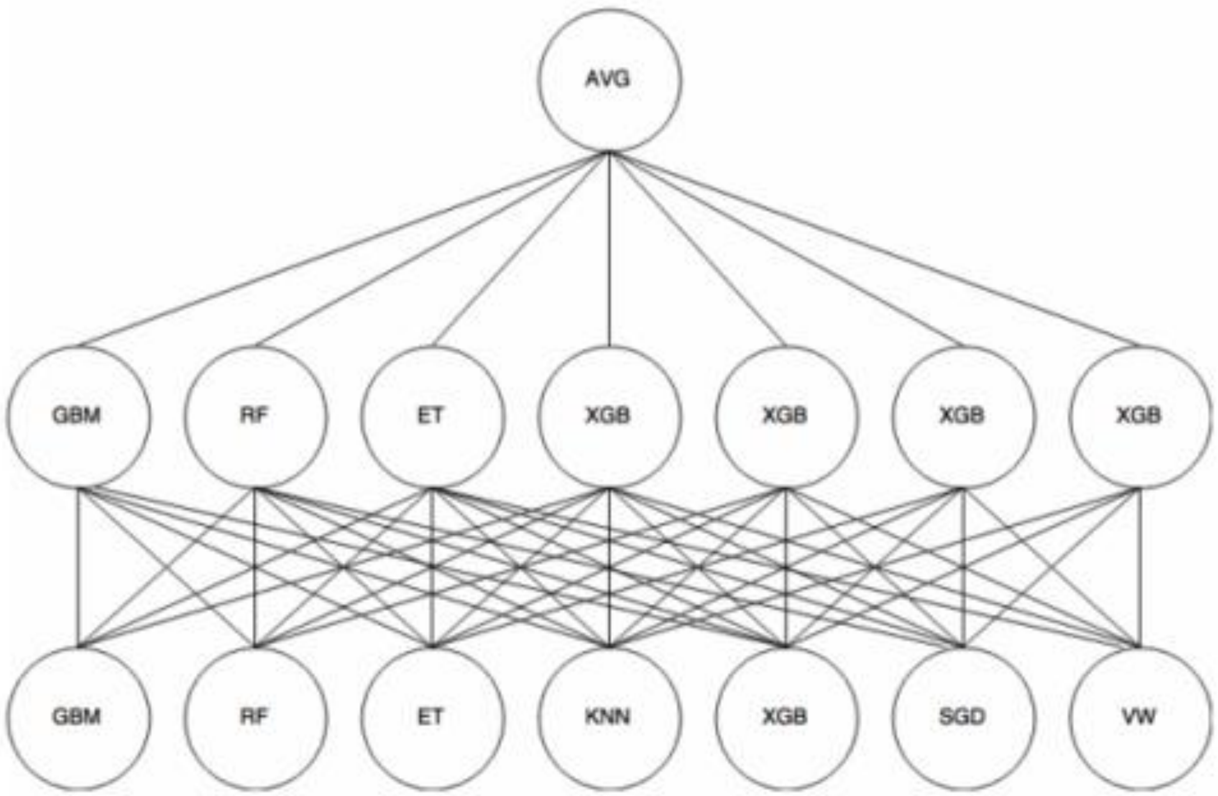
$$J(\theta) = -[\frac{1}{m}\sum_{i=1}^{n}\sum_{k=1}^{K} y_k^{(i)} log(f(x^{(i)},\Theta)_k) + (1-y_k^{(i)}) log(1-f(x^{(i)},\Theta)_k)] + \frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

https://playground.tensorflow.org/

Let’s get real!

10. Neural Networks

Blending/Stacking



https://mlwave.com/kaggle-ensembling-guide/

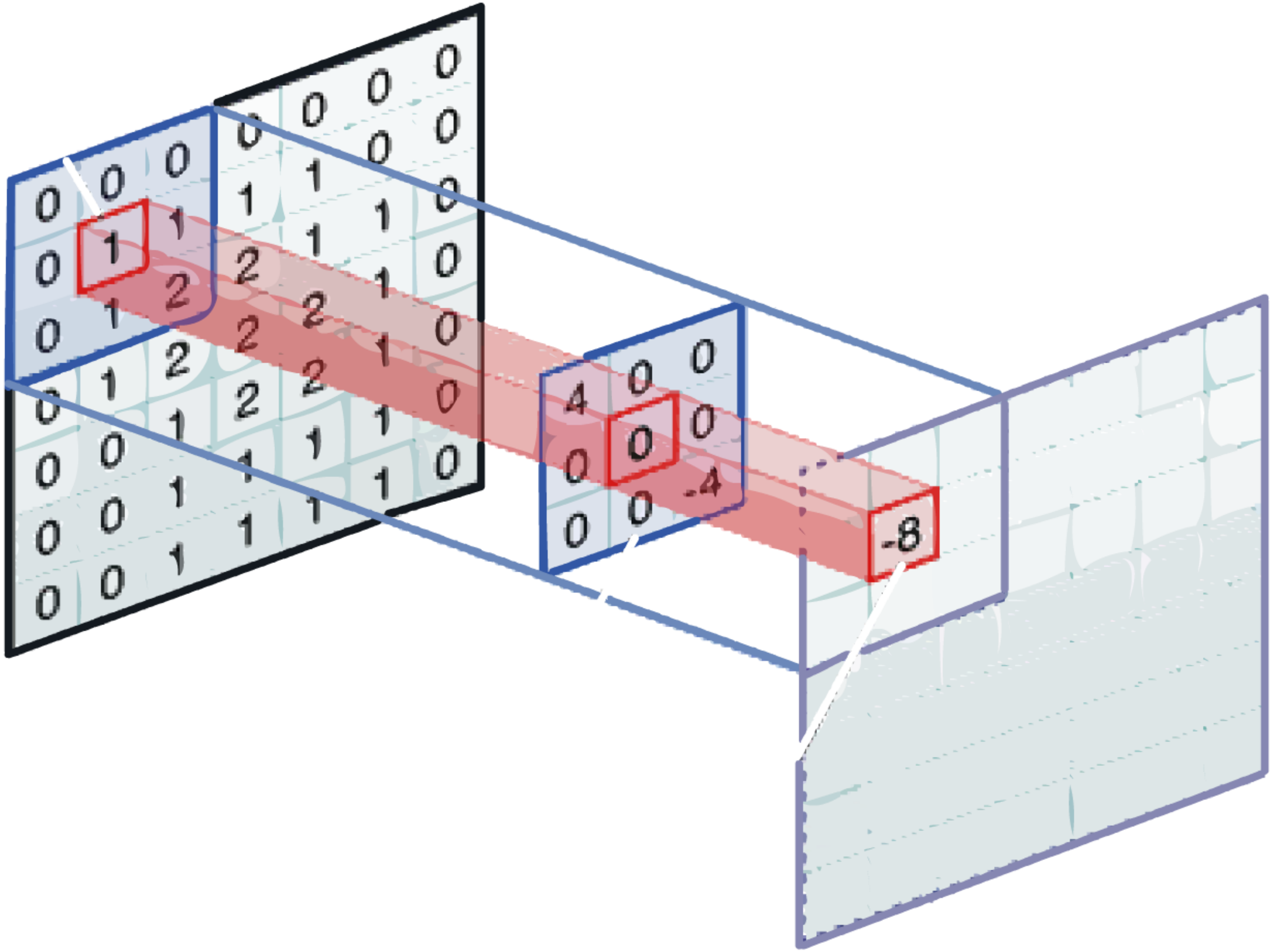
Convolutional Neural Networks

What We See

08	02	22	97	38	15	00	40	00	75	04	05	07	78	52	12	50	77	91	08
49	49	99	40	17	81	18	57	60	87	17	40	98	43	69	48	04	56	62	00
81	49	31	73	55	79	14	29	93	71	40	67	53	88	30	03	49	13	36	65
52	70	95	23	04	60	11	42	69	24	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	67	63	89	41	92	36	54	22	40	40	28	66	33	13	80
24	47	32	60	99	03	45	02	44	75	33	53	78	36	84	20	35	17	12	50
32	98	81	28	64	23	67	10	26	38	40	67	59	54	70	66	18	38	64	70
67	26	20	68	02	62	12	20	95	63	94	39	63	08	40	91	66	49	94	21
24	55	58	05	66	73	99	26	97	17	78	78	96	83	14	88	34	89	63	72
21	36	23	09	75	00	76	44	20	45	35	14	00	61	33	97	34	31	33	95
78	17	53	28	22	75	31	67	15	94	03	80	04	62	16	14	09	53	56	92
16	39	05	42	96	35	31	47	55	58	88	24	00	17	54	24	36	29	85	57
86	56	00	48	35	71	89	07	05	44	44	37	44	60	21	58	51	54	17	58
19	80	81	68	05	94	47	69	28	73	92	13	86	52	17	77	04	89	55	40
04	52	08	83	97	35	99	16	07	97	57	32	16	26	26	79	33	27	98	66
88	36	68	87	57	62	20	72	03	46	33	67	46	55	12	32	63	93	53	69
04	42	16	73	38	25	39	11	24	94	72	18	08	46	29	32	40	62	76	36
20	69	36	41	72	30	23	88	34	62	99	69	82	67	59	85	74	04	36	16
20	73	35	29	78	31	90	01	74	31	49	71	48	86	81	16	23	57	05	54
01	70	54	71	83	51	54	69	16	92	33	48	61	43	52	01	89	19	67	48

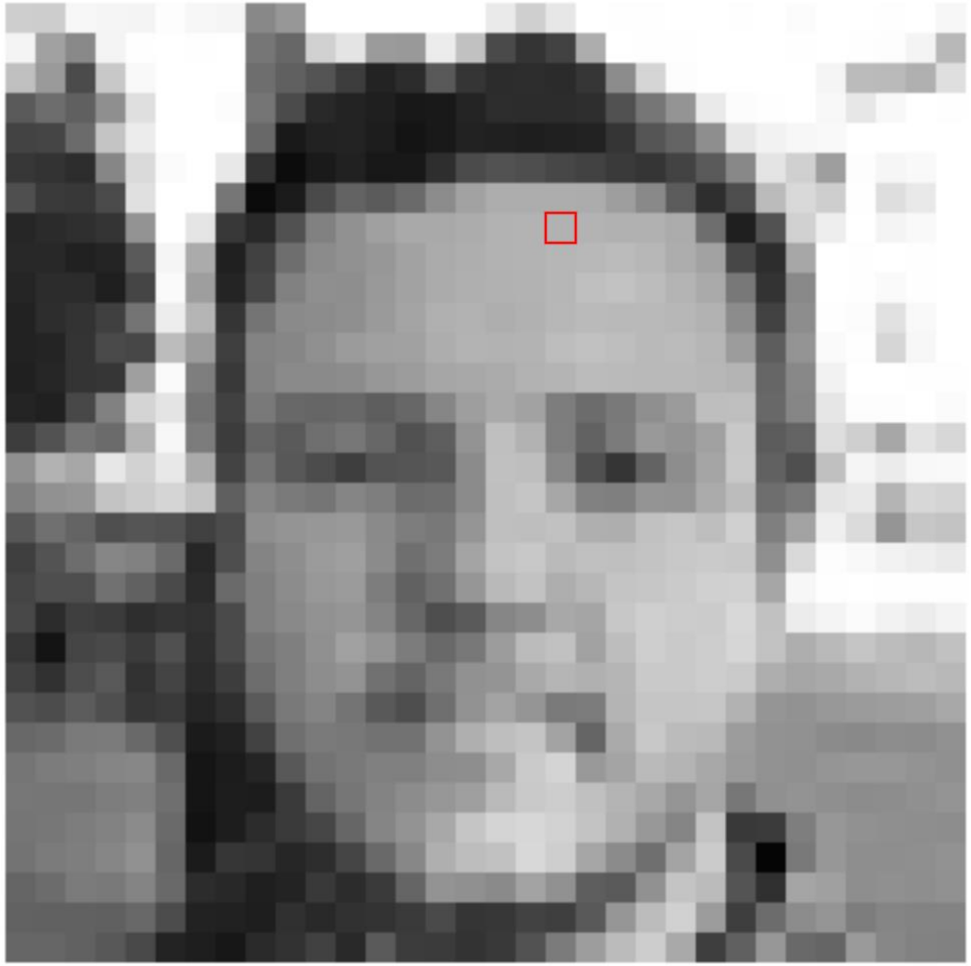
What Computers See

https://github.com/leriomaggio



https://github.com/leriomaggio

206	205	247	245	244	253	247	245	138	151	255	255	255	255	255	255	234	207	231	255	254	254	255	255	254	255	252	255	255	254	255	247
244	161	137	244	254	255	254	255	118	103	209	228	155	153	238	193	74	52	68	173	255	254	254	255	255	255	254	255	254	253	244	184
192	154	75	200	249	255	255	255	110	98	84	61	35	44	89	53	44	45	43	54	140	213	253	255	255	255	255	245	187	188	178	223
90	109	98	143	223	255	255	252	117	75	41	35	31	24	25	38	45	44	44	48	81	118	148	234	252	254	255	248	231	248	255	254
67	69	107	198	238	255	255	255	104	25	34	35	29	20	25	34	32	30	32	34	53	85	100	142	231	242	247	249	255	255	255	255
55	51	45	134	218	251	255	232	51	12	28	33	24	24	48	75	82	78	71	68	58	53	67	90	138	228	208	158	253	246	249	255
79	58	58	75	224	255	255	118	11	27	74	99	91	108	140	162	173	173	173	172	158	137	92	48	78	187	217	208	254	222	233	255
38	43	47	52	147	255	229	58	41	81	129	145	160	169	169	172	178	179	178	179	177	177	172	110	31	82	209	238	255	244	249	255
40	40	33	38	90	245	171	32	65	110	139	145	151	162	171	174	178	179	182	184	187	183	173	162	71	45	187	255	254	255	254	255
37	44	44	31	69	250	158	38	70	129	143	142	153	162	171	175	177	178	182	191	194	188	180	170	120	51	137	255	254	250	254	255
34	45	51	64	116	237	181	53	116	138	140	143	154	164	178	178	174	177	183	188	185	185	183	178	140	68	141	254	252	225	249	255
34	38	52	74	71	188	156	63	131	134	144	155	160	161	173	179	178	179	189	193	190	185	187	182	156	93	148	250	254	214	247	255
32	38	52	54	159	250	126	57	129	138	138	140	151	156	168	168	171	178	180	187	188	185	185	183	180	102	136	242	255	255	254	254
38	32	72	129	212	228	115	65	121	104	102	104	94	103	134	158	170	162	125	108	121	143	155	190	191	104	134	230	253	253	255	251
61	82	116	107	179	247	124	60	101	90	111	119	103	81	94	147	191	178	128	98	123	153	147	161	200	92	100	222	207	167	227	215
144	178	167	231	210	232	170	67	115	88	78	62	83	85	88	139	192	190	135	80	53	99	141	165	201	97	79	192	245	235	248	249
127	145	149	195	204	213	197	95	133	122	117	133	128	108	110	139	191	197	167	129	127	148	147	171	188	110	121	228	233	180	215	212
87	112	100	79	85	82	65	75	142	148	151	153	138	125	120	149	191	190	193	175	174	193	198	190	208	127	163	239	219	149	198	195
63	83	109	134	129	108	39	78	132	142	155	159	139	111	124	164	195	200	188	192	191	195	200	202	200	143	217	253	249	242	238	234
69	78	78	113	97	74	43	106	127	140	152	155	125	97	112	150	185	194	174	183	198	198	202	208	209	168	247	254	255	254	254	254
72	44	63	59	48	52	49	74	127	137	148	149	132	103	78	90	134	141	168	165	199	207	204	203	216	193	238	244	251	242	238	243
55	20	69	73	59	80	48	74	117	127	144	161	148	124	105	120	156	187	193	182	189	208	201	205	214	194	174	185	197	188	183	193
65	49	77	89	50	68	43	61	109	127	141	147	113	100	121	145	148	169	181	178	181	201	201	205	202	174	168	169	178	183	188	184
82	78	92	79	54	58	37	47	90	121	132	116	89	78	111	146	163	149	122	124	180	197	197	198	178	149	146	152	155	157	159	168
104	107	122	123	105	79	27	33	68	111	122	120	114	114	147	175	190	198	163	101	170	200	187	185	156	146	145	139	137	141	140	145
117	124	127	133	135	105	21	28	37	88	115	121	128	128	141	142	168	202	212	153	164	186	180	168	154	146	144	149	151	151	147	144
119	118	118	125	128	111	21	29	28	58	100	118	131	140	151	159	188	201	205	192	180	168	149	168	119	144	147	143	140	141	144	148
117	119	125	130	139	108	18	29	44	58	70	102	133	147	168	197	212	215	210	195	177	152	133	195	57	59	128	151	145	143	142	141
115	123	128	134	145	102	27	54	52	38	45	69	105	135	175	189	193	218	208	168	139	111	164	203	74	5	121	151	142	142	143	146
101	108	123	121	132	105	44	40	31	35	57	44	58	101	147	144	138	163	145	94	90	145	198	187	84	48	165	160	142	144	142	145
98	97	97	98	104	78	34	33	30	48	41	49	51	58	74	53	55	68	63	89	150	188	209	156	62	108	140	149	125	133	131	131
102	102	97	88	73	35	30	23	42	50	65	41	90	60	59	51	57	82	123	157	187	205	169	62	98	151	105	101	154	136	130	129

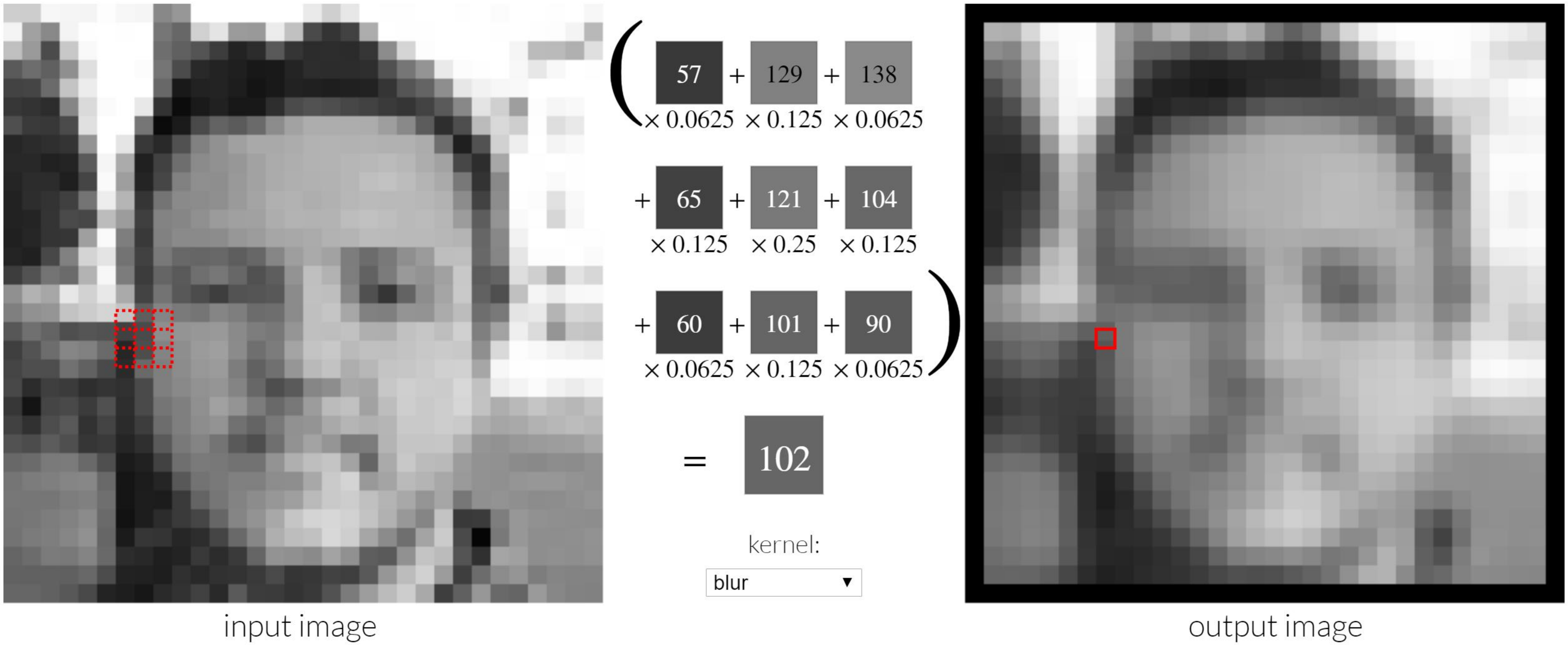


http://setosa.io/ev/image-kernels/

blur

$$\begin{pmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{pmatrix}$$

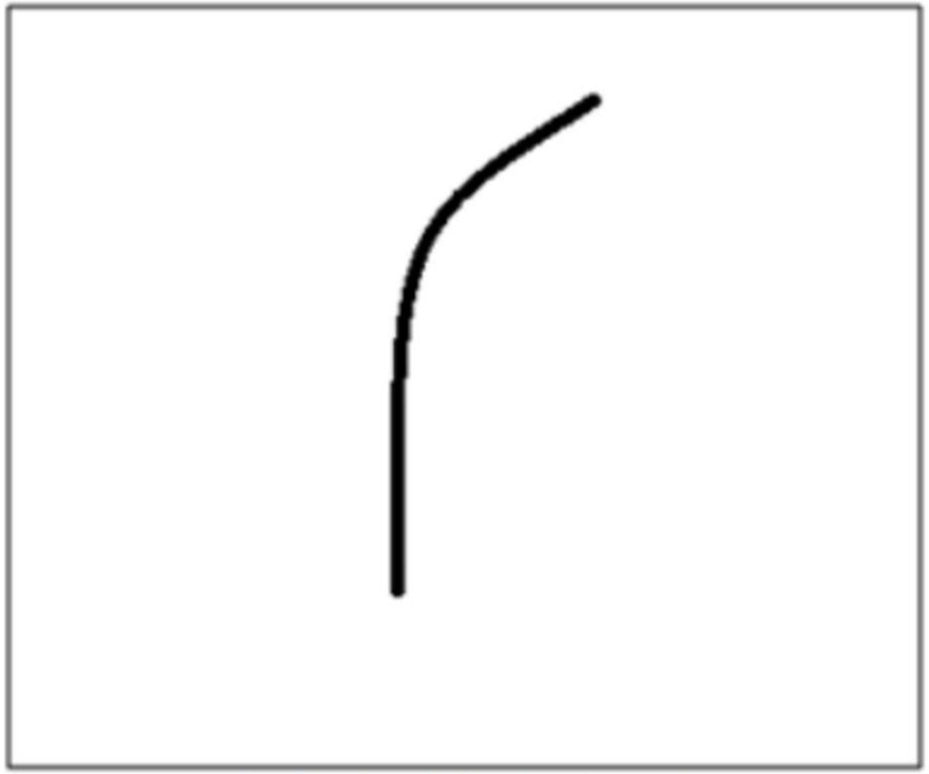
http://setosa.io/ev/image-kernels/



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

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

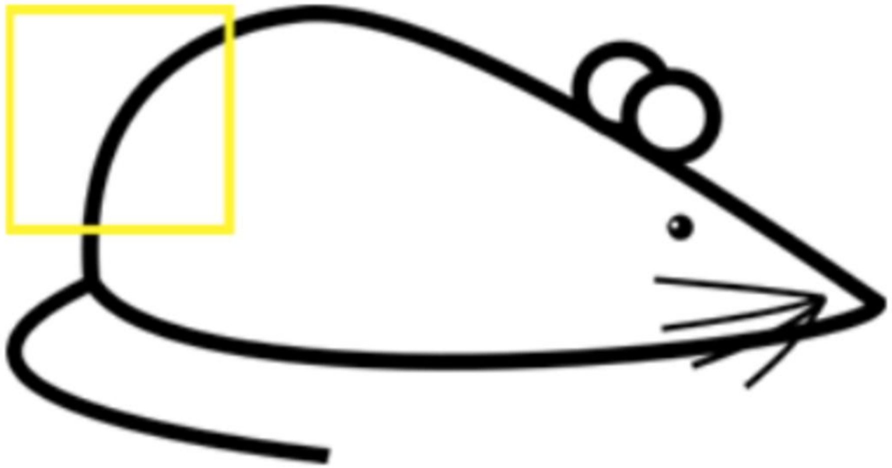
Pixel representation of filter



Visualization of a curve detector filter

https://github.com/leriomaggio

Original image



Visualization of the filter on the image

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

https://github.com/leriomaggio

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

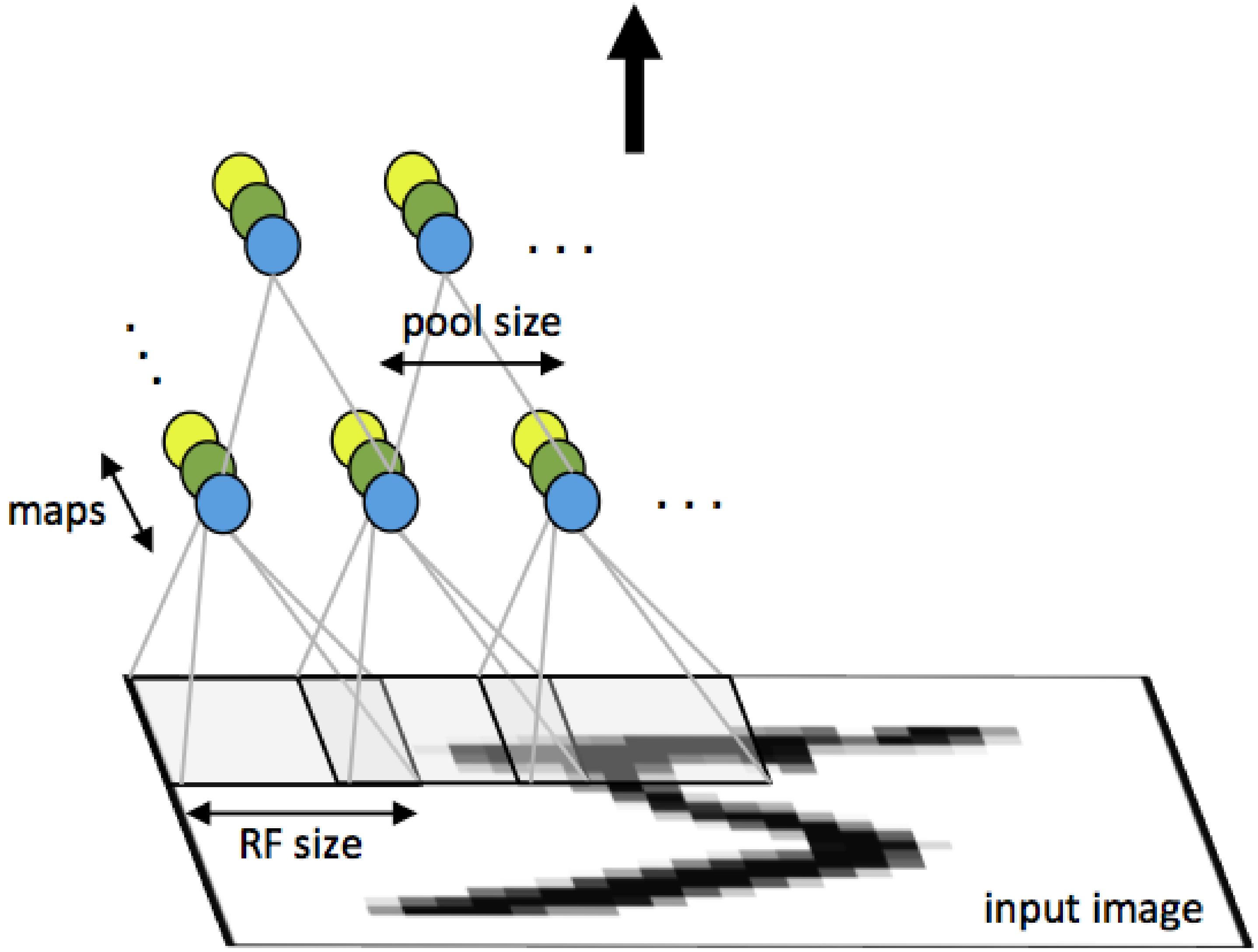
Pixel representation of receptive field

$*$

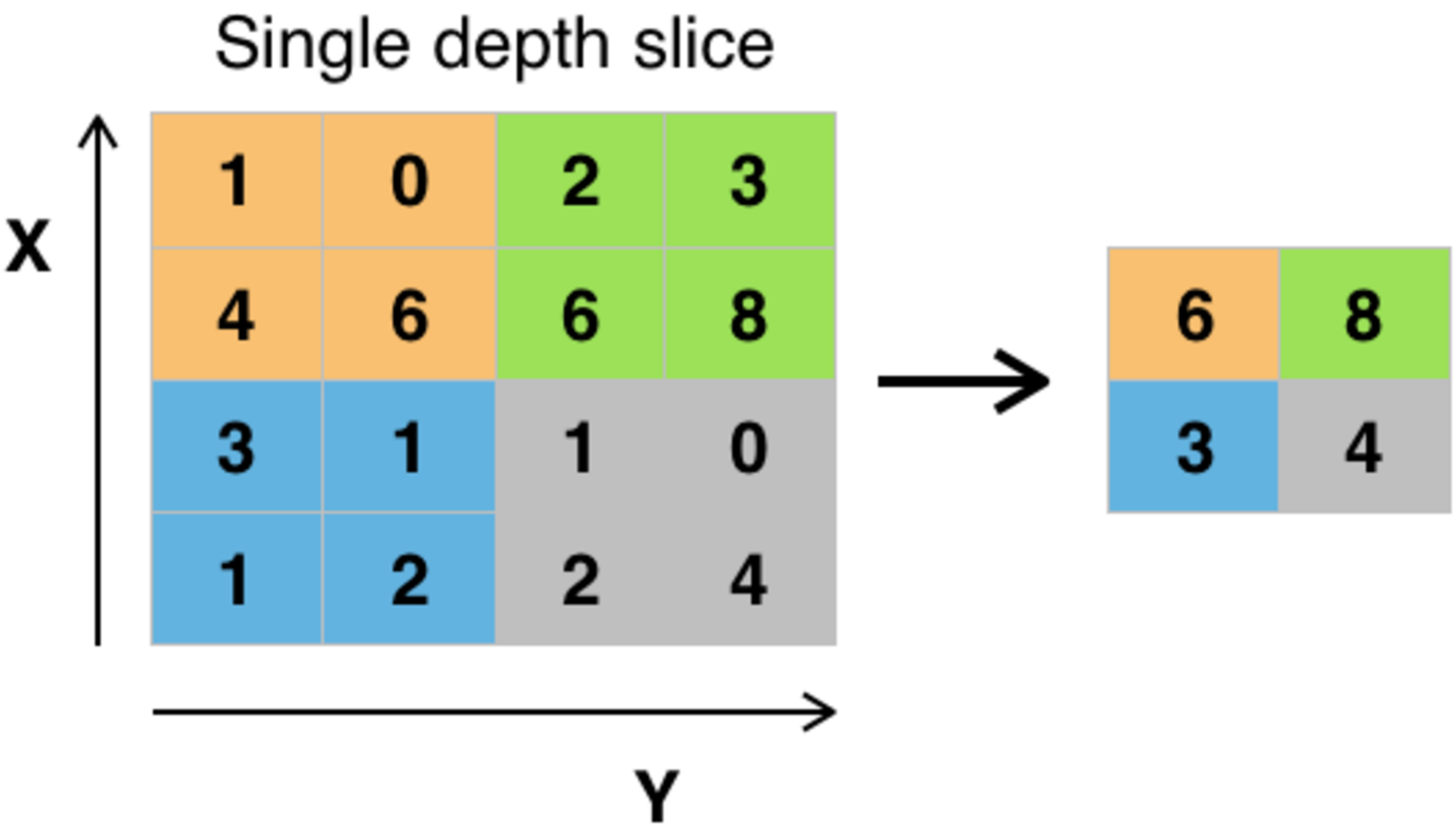
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

https://github.com/leriomaggio



https://github.com/leriomaggio



Example of Maxpool with a 2x2 filter and a stride of 2

https://github.com/leriomaggio

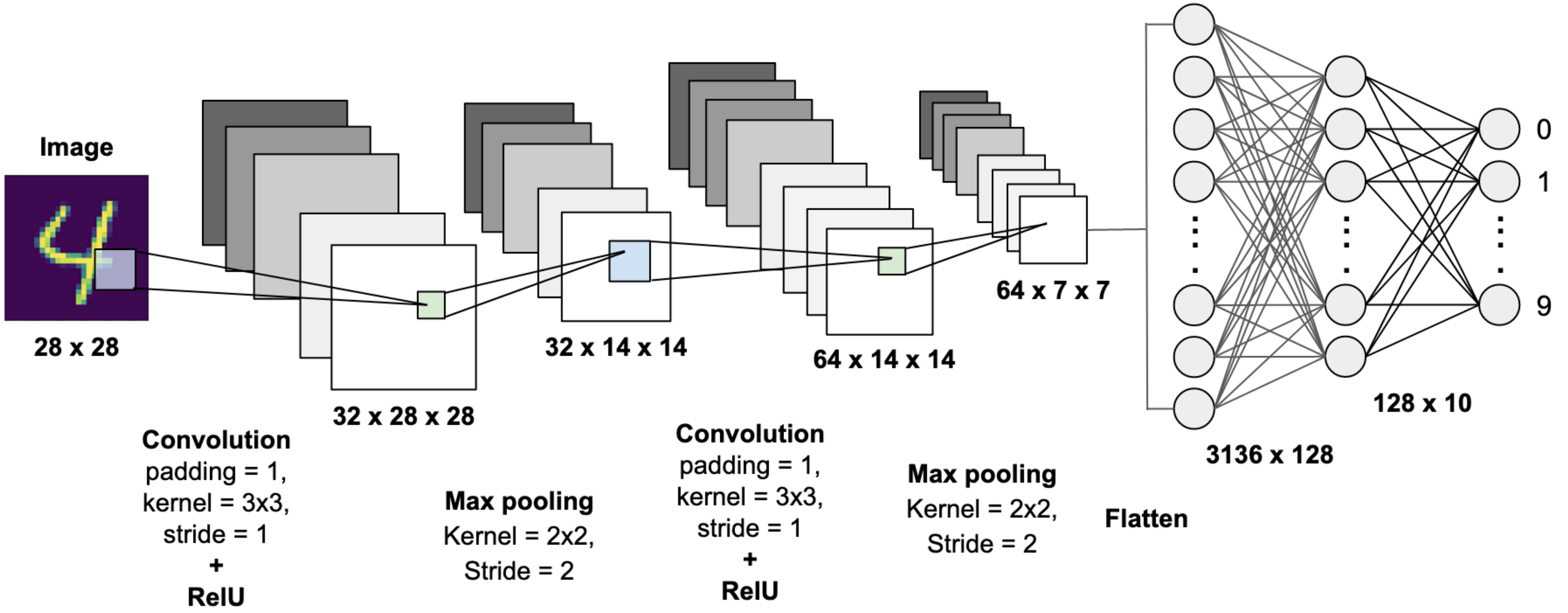
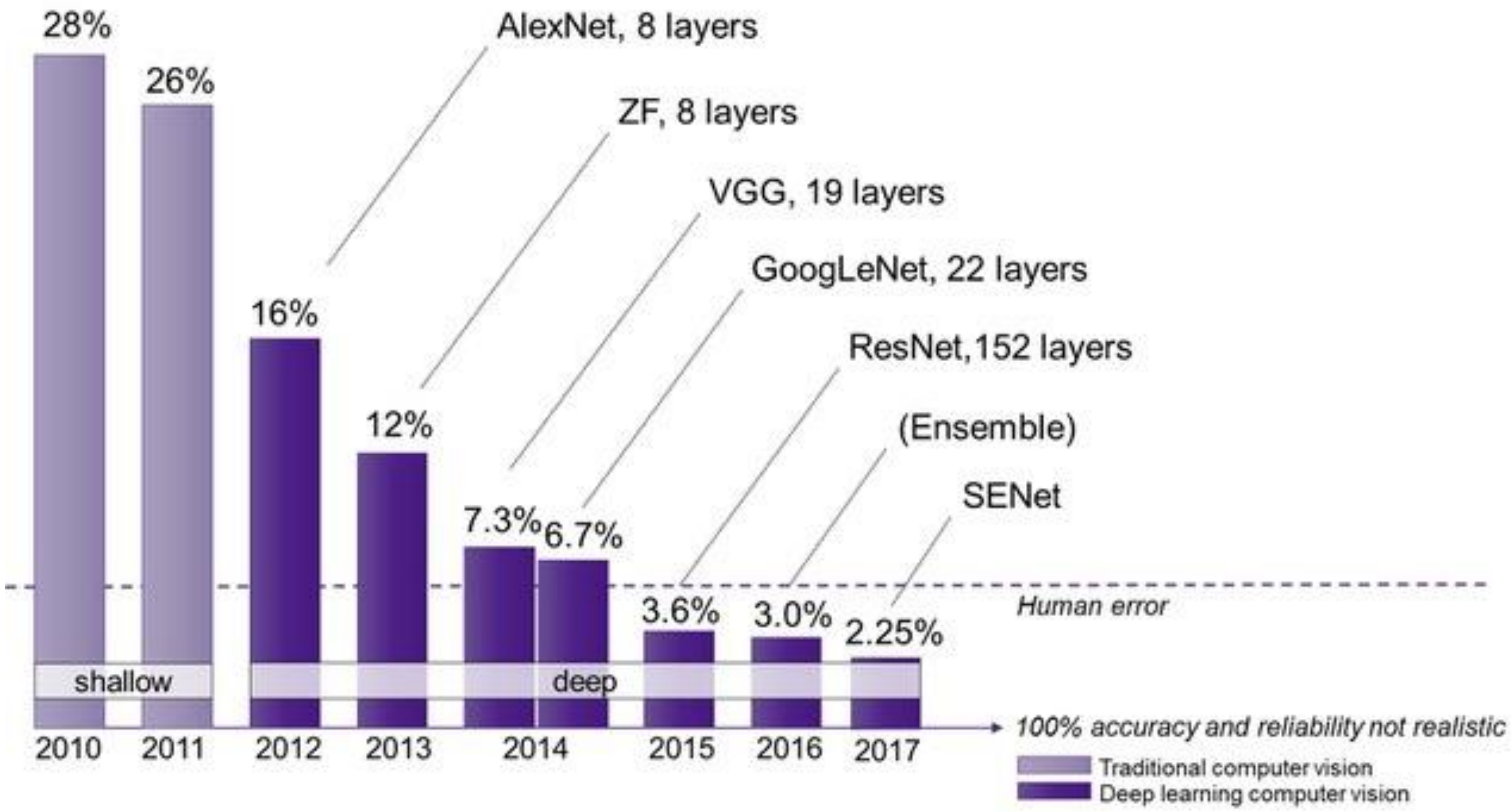
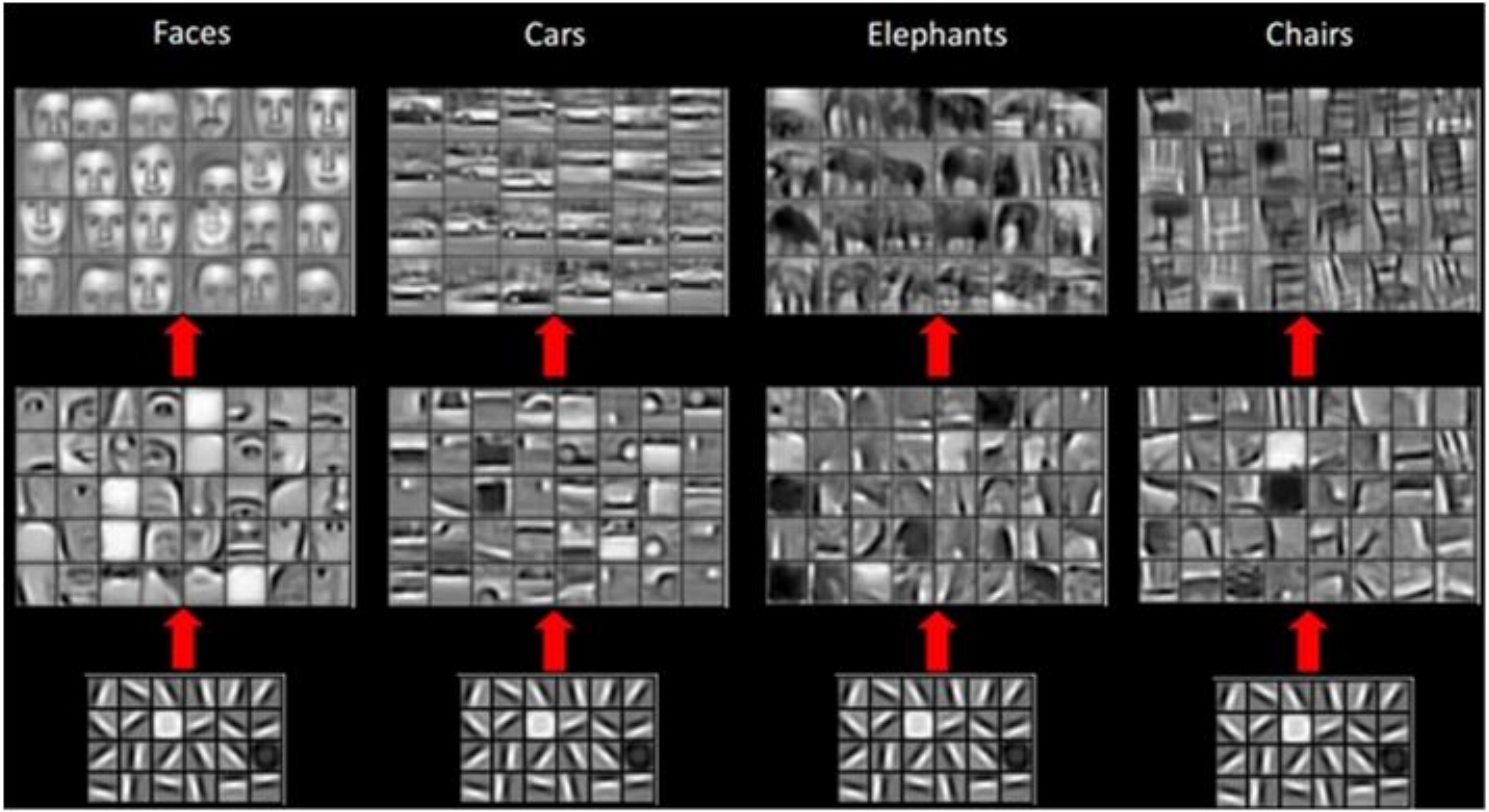


Image
28 x 28
32 x 28 x 28
32 x 14 x 14
64 x 14 x 14
64 x 7 x 7
0
1
9
128 x 10
3136 x 128
Convolution
padding = 1,
kernel = 3x3,
stride = 1
+
RelU
Max pooling
Kernel = 2x2,
Stride = 2
Convolution
padding = 1,
kernel = 3x3,
stride = 1
+
RelU
Max pooling
Kernel = 2x2,
Stride = 2
Flatten

- 14.197.122 images
- 21.841 subcategories
- 27 high-level categories

ImageNet Large Scale Visual Recognition Challenge




Faces
Cars
Elephants
Chairs

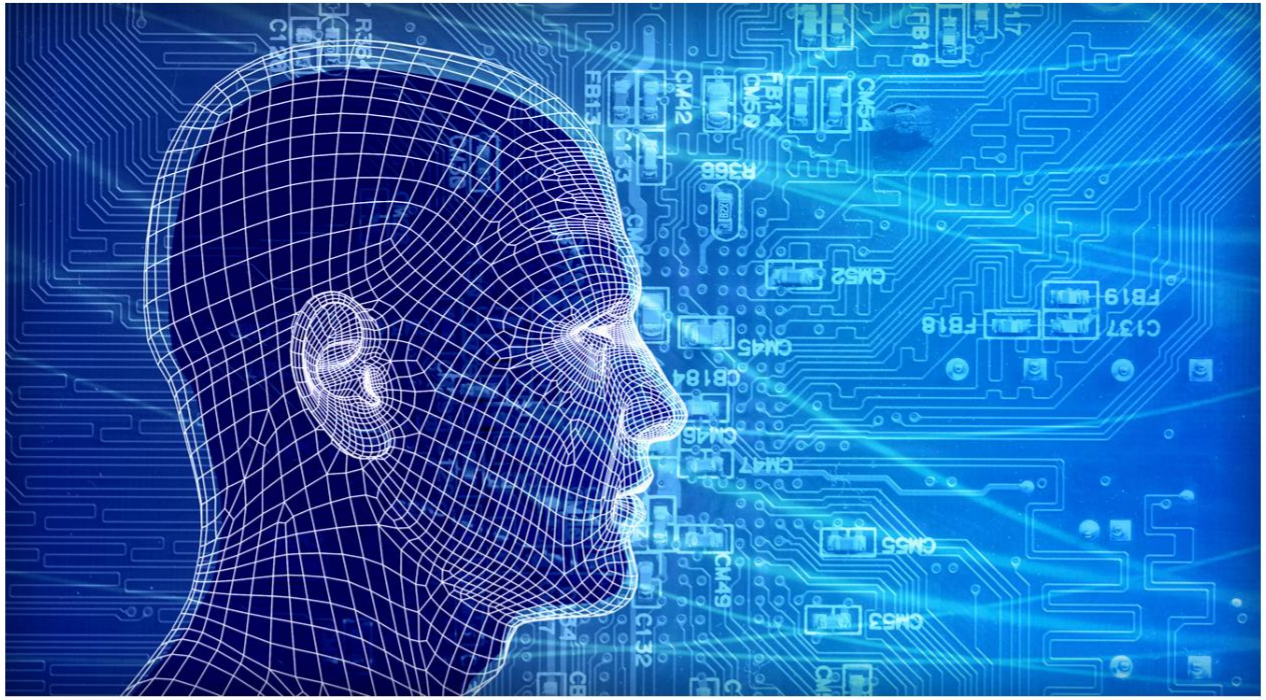
Let’s get real!

Cats & dogs

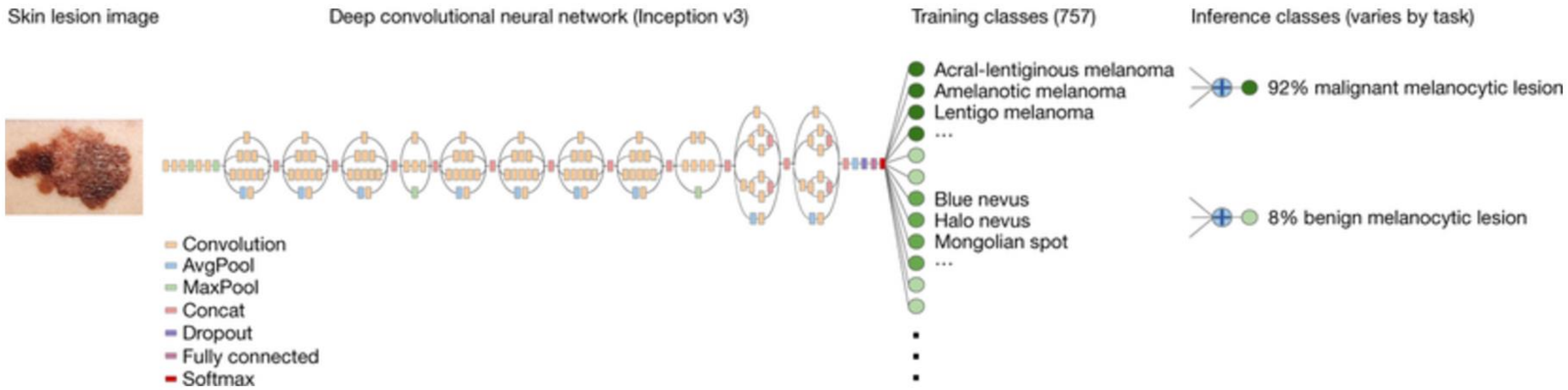
Deep learning algorithm diagnoses skin cancer as well as seasoned dermatologists

By Jessica Hall on January 25, 2017 at 1:25 pm | 16 Comments

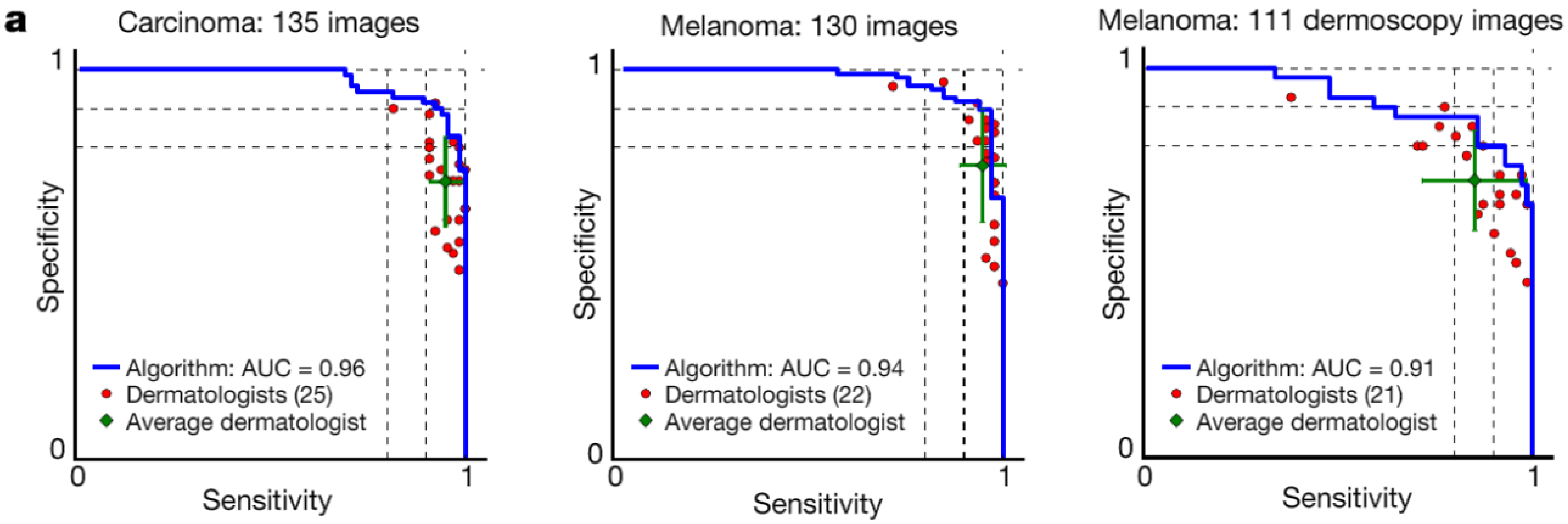
422 shares



Thrun et al. Dermatologist-level classification of skin cancer with deep neural networks. Nature, January 2017 DOI: 10.1038/nature21056



Thrun et al. Dermatologist-level classification of skin cancer with deep neural networks. Nature, January 2017 DOI: 10.1038/nature21056



keratinocyte carcinomas: 65
benign seborrheic keratoses: 75

malignant melanomas: 33
benign nevi: 97

malignant melanomas: 71
benign nevi: 40

Branch: master ▾ models / research / inception /

Create new file | Upload files | Find file | History

ema987 and shlens Added missing python interpreter directive (#6015) Latest commit 47a7d7c on 11 Jan

..		
g3doc	Move the research models into a research subfolder (#2430)	2 years ago
inception	Added missing python interpreter directive (#6015)	10 months ago
.gitignore	Move the research models into a research subfolder (#2430)	2 years ago
README.md	Make the redirect to slim notice more prominent	2 years ago
WORKSPACE	Move the research models into a research subfolder (#2430)	2 years ago

README.md

NOTE: For the most part, you will find a newer version of this code at models/research/slim. In particular:

- `inception_train.py` and `imagenet_train.py` should no longer be used. The slim editions for running on multiple GPUs are the current best examples.
- `inception_distributed_train.py` and `imagenet_distributed_train.py` are still valid examples of distributed training.

For performance benchmarking, please see https://www.tensorflow.org/performance/benchmarks.

Inception in TensorFlow

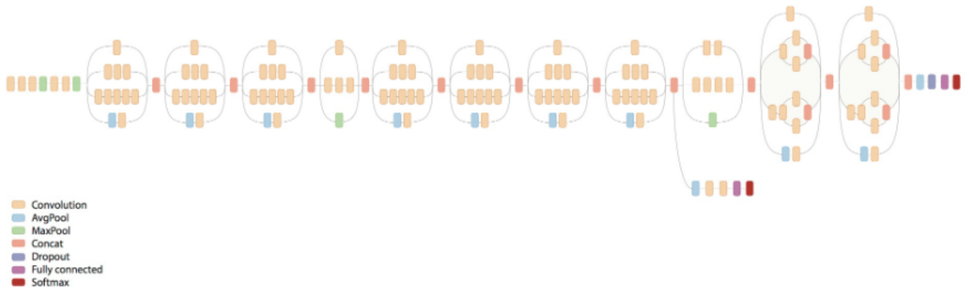
ImageNet is a common academic data set in machine learning for training an image recognition system. Code in this directory demonstrates how to use TensorFlow to train and evaluate a type of convolutional neural network (CNN) on this academic data set. In particular, we demonstrate how to train the Inception v3 architecture as specified in:

Rethinking the Inception Architecture for Computer Vision

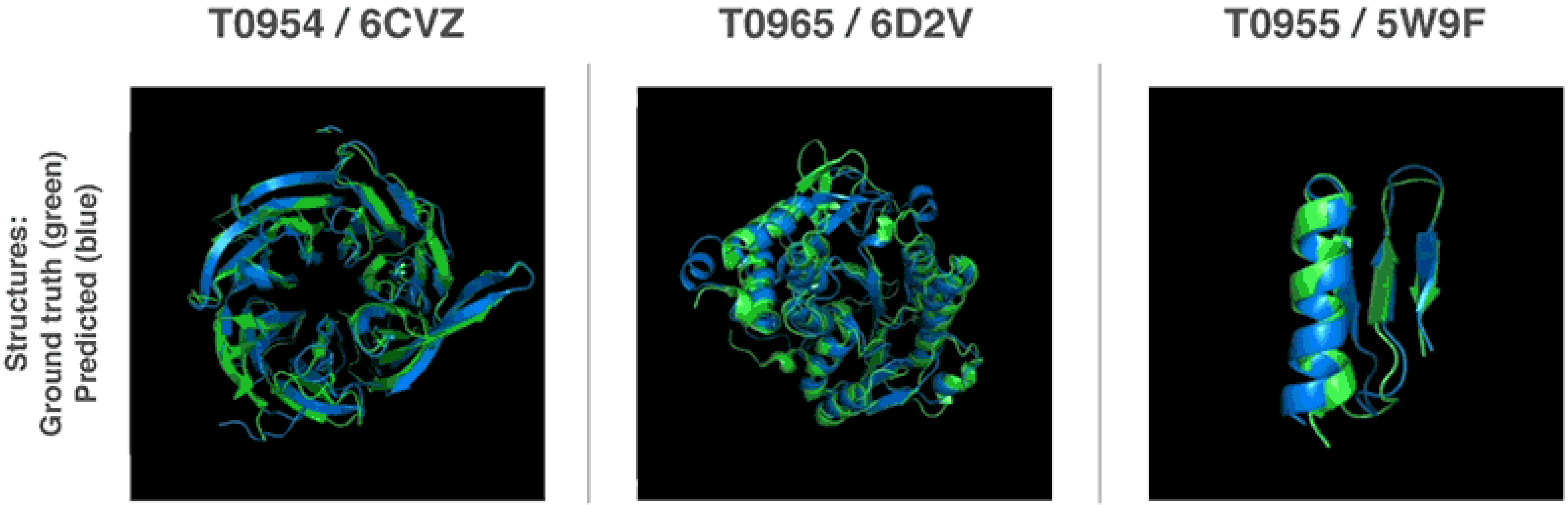
Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna

http://arxiv.org/abs/1512.00567

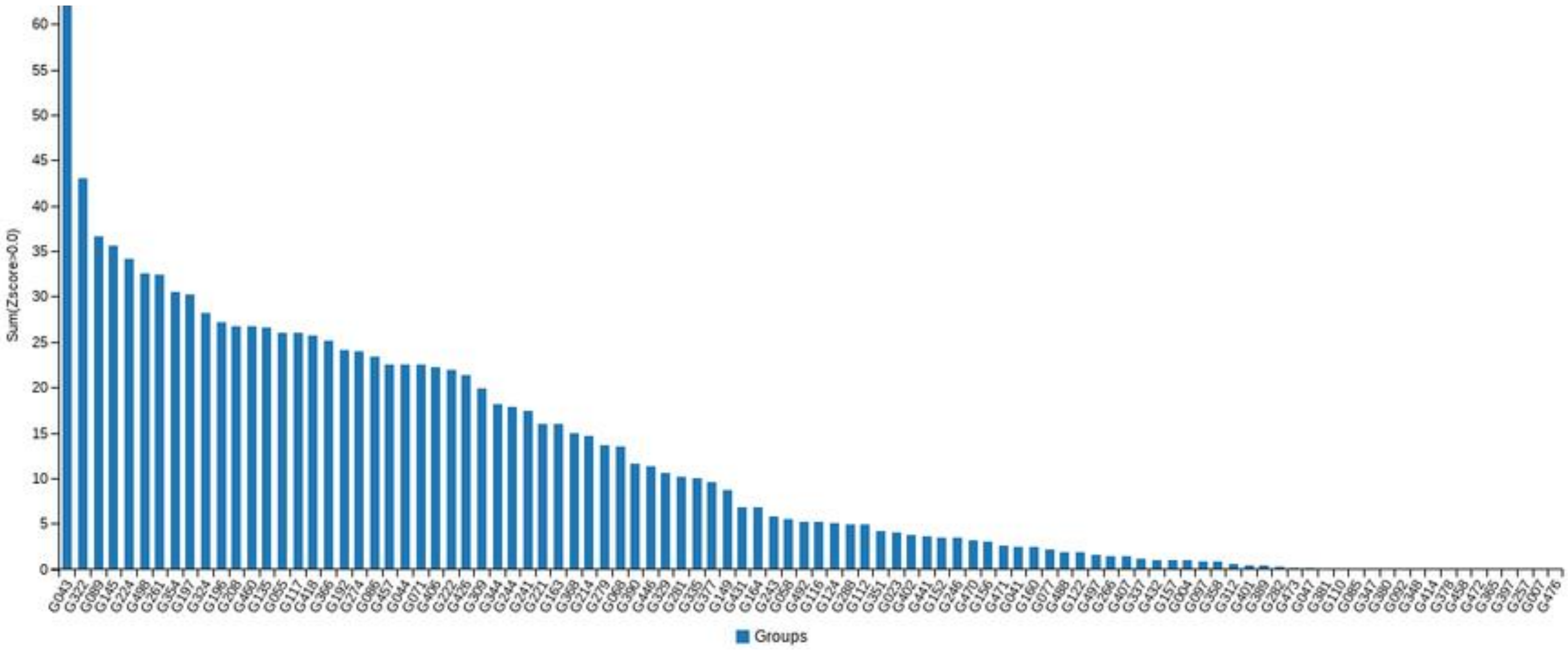
This network achieves 21.2% top-1 and 5.6% top-5 error for single frame evaluation with a computational cost of 5 billion multiply-adds per inference and with using less than 25 million parameters. Below is a visualization of the model architecture.



Critical Assessment of Techniques for Protein Structure Prediction



deepmind.com/blog/article/alphafold @ CASP13 (2018)



predictioncenter.org