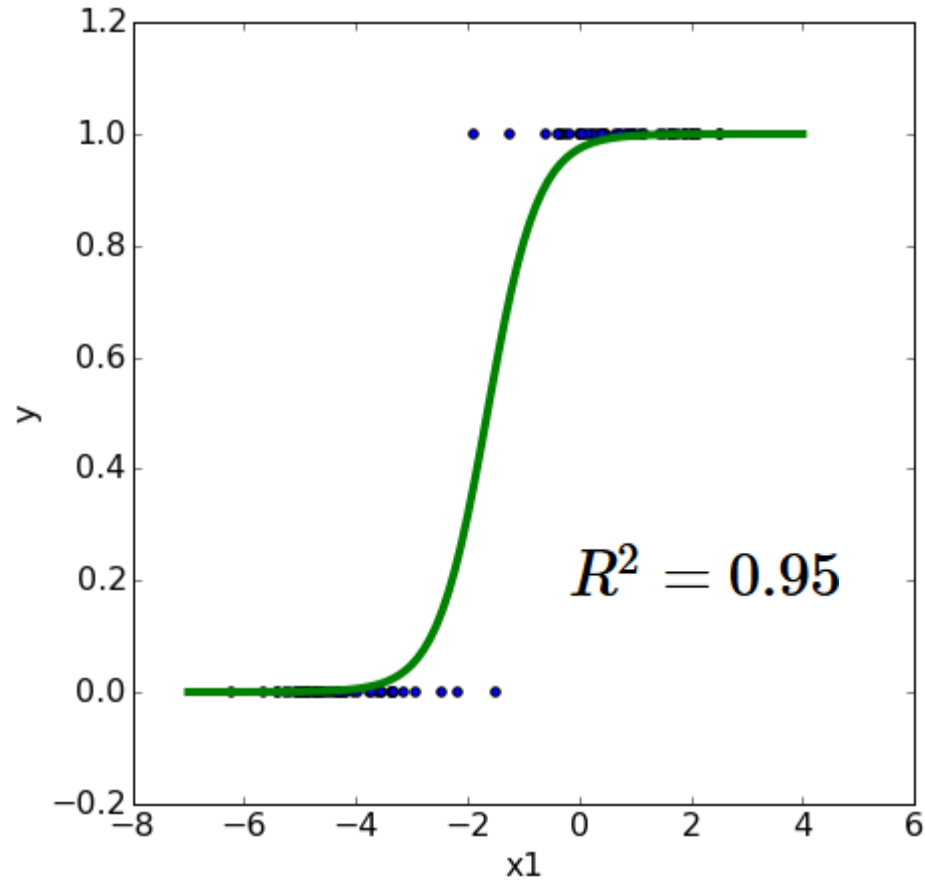
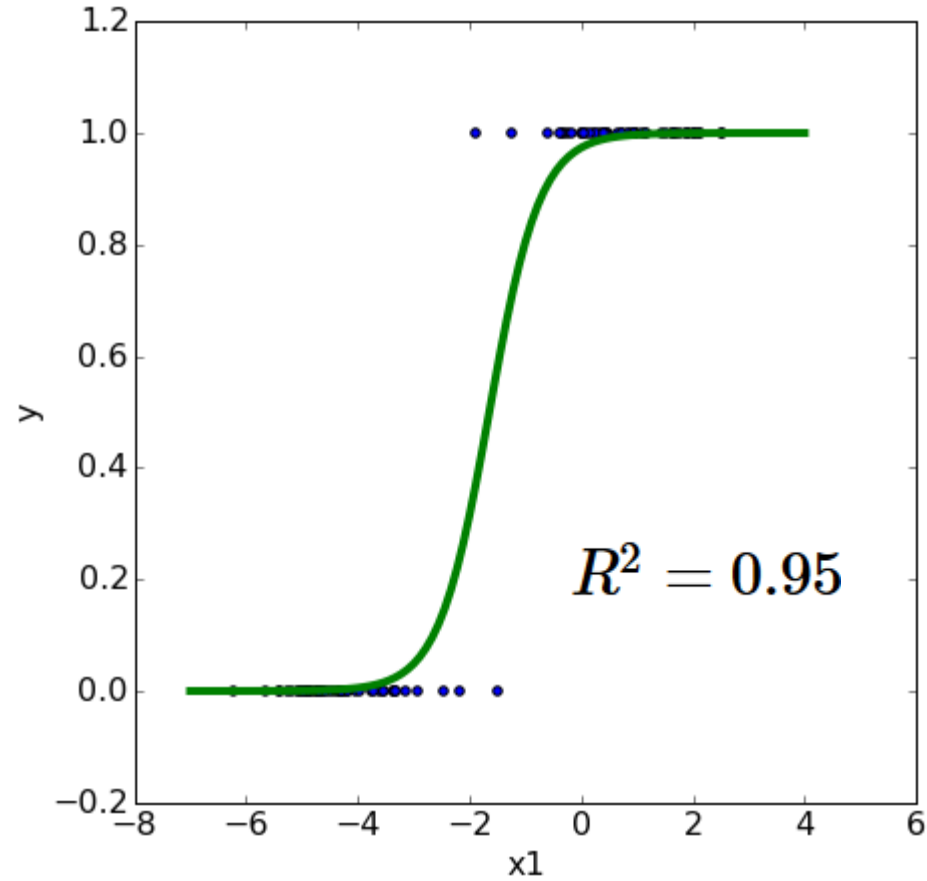


# *logistic regression*



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*

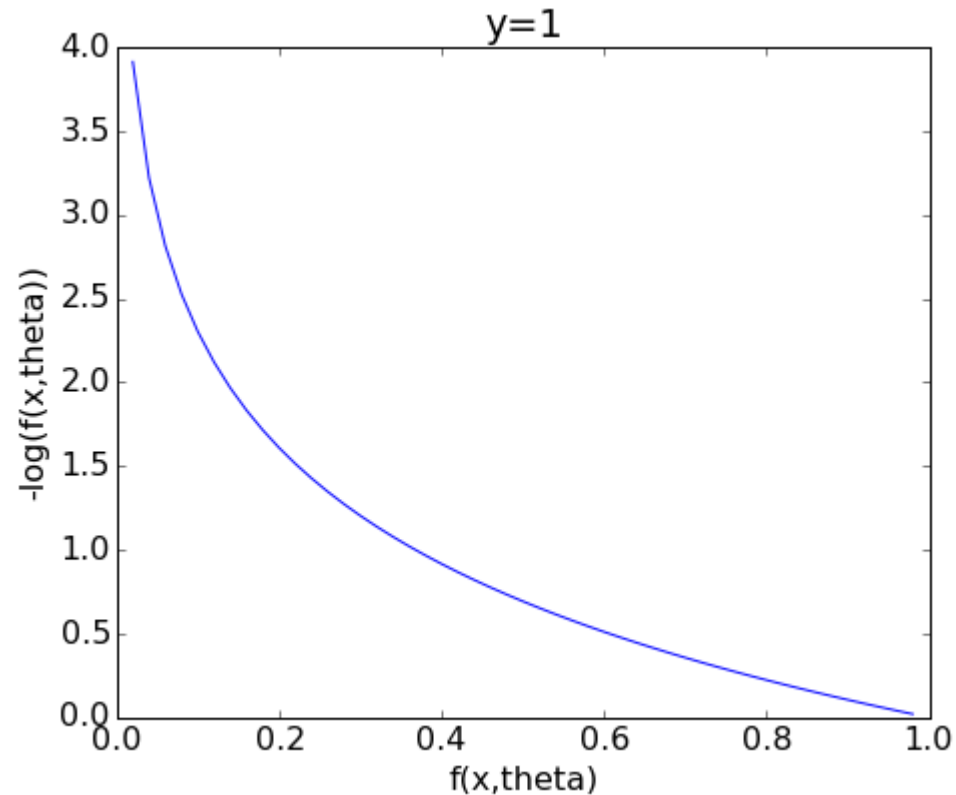
$$J(\theta) = -\left[\frac{1}{n} \sum_{i=1}^n y^{(i)} \log(f(x^{(i)}, \theta)) + (1 - y^{(i)}) \log(1 - f(x^{(i)}, \theta))\right]$$

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

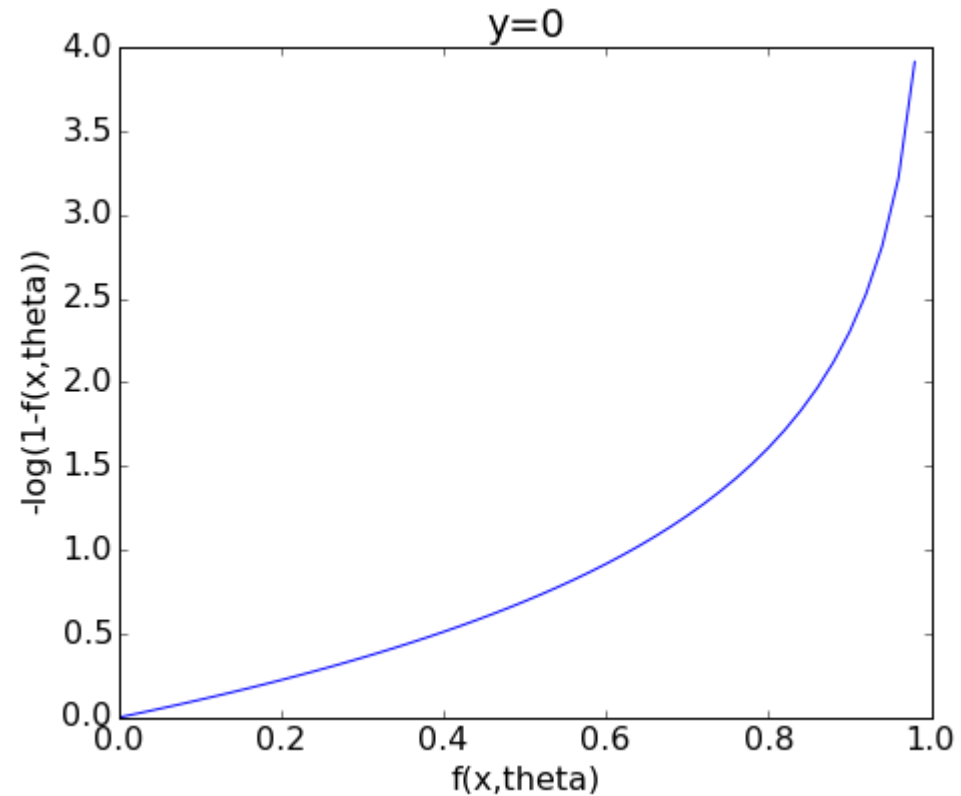
## *logistic regression: cost function*

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

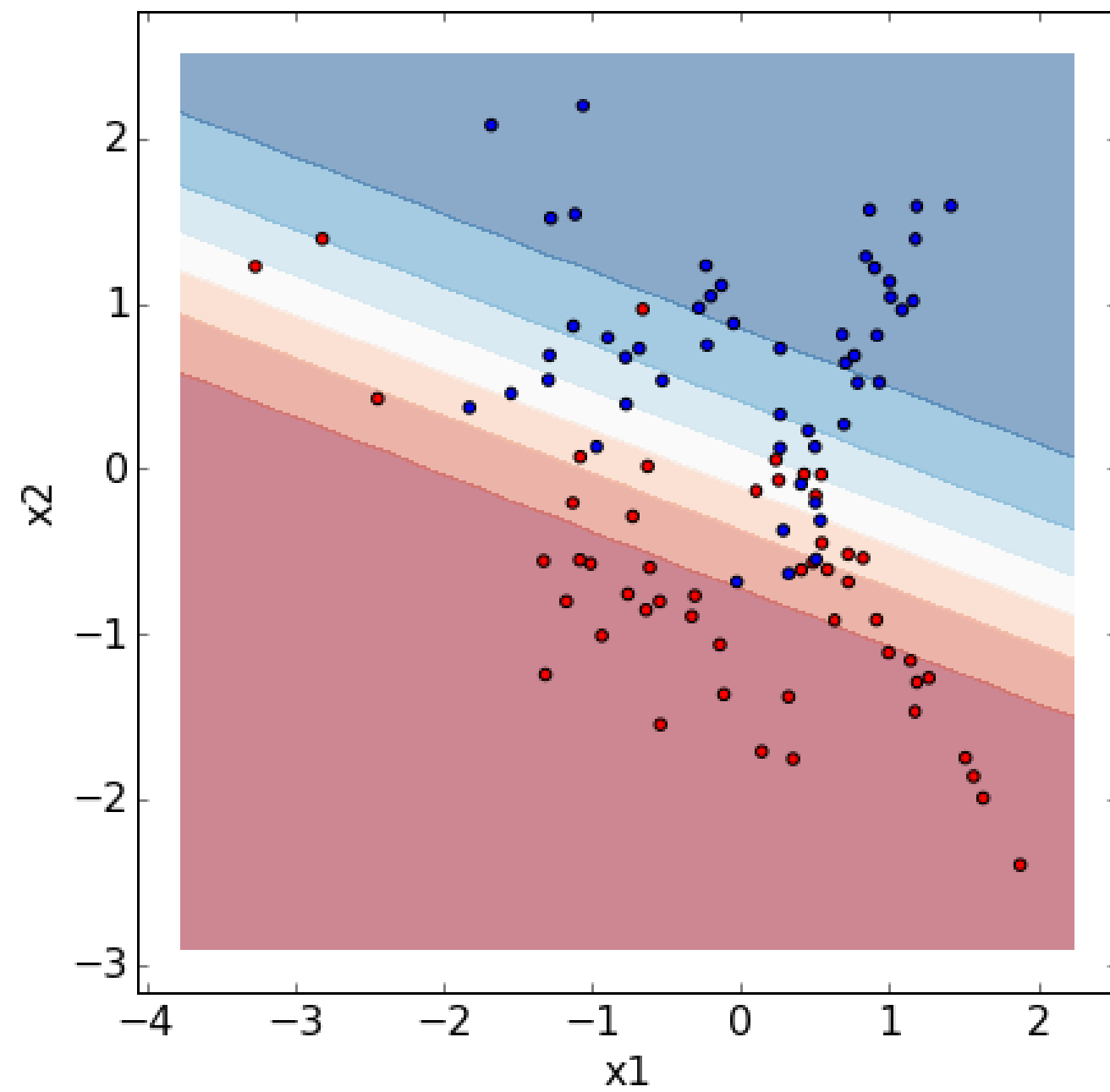
$$f(x, \theta) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_m x_m) = g(\theta' x)$$

to the data set such that the cost function

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

is minimal using gradient descent

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



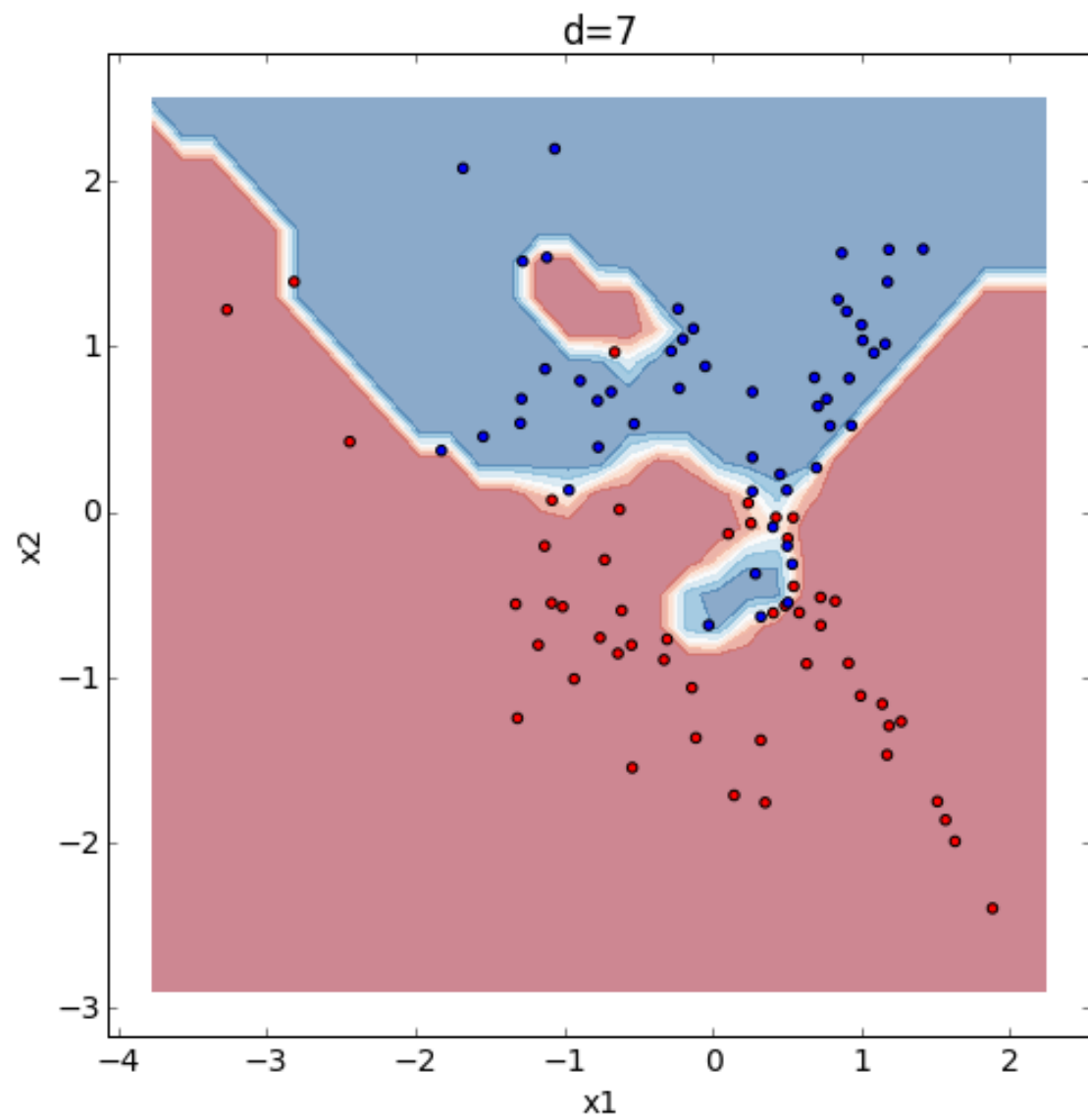
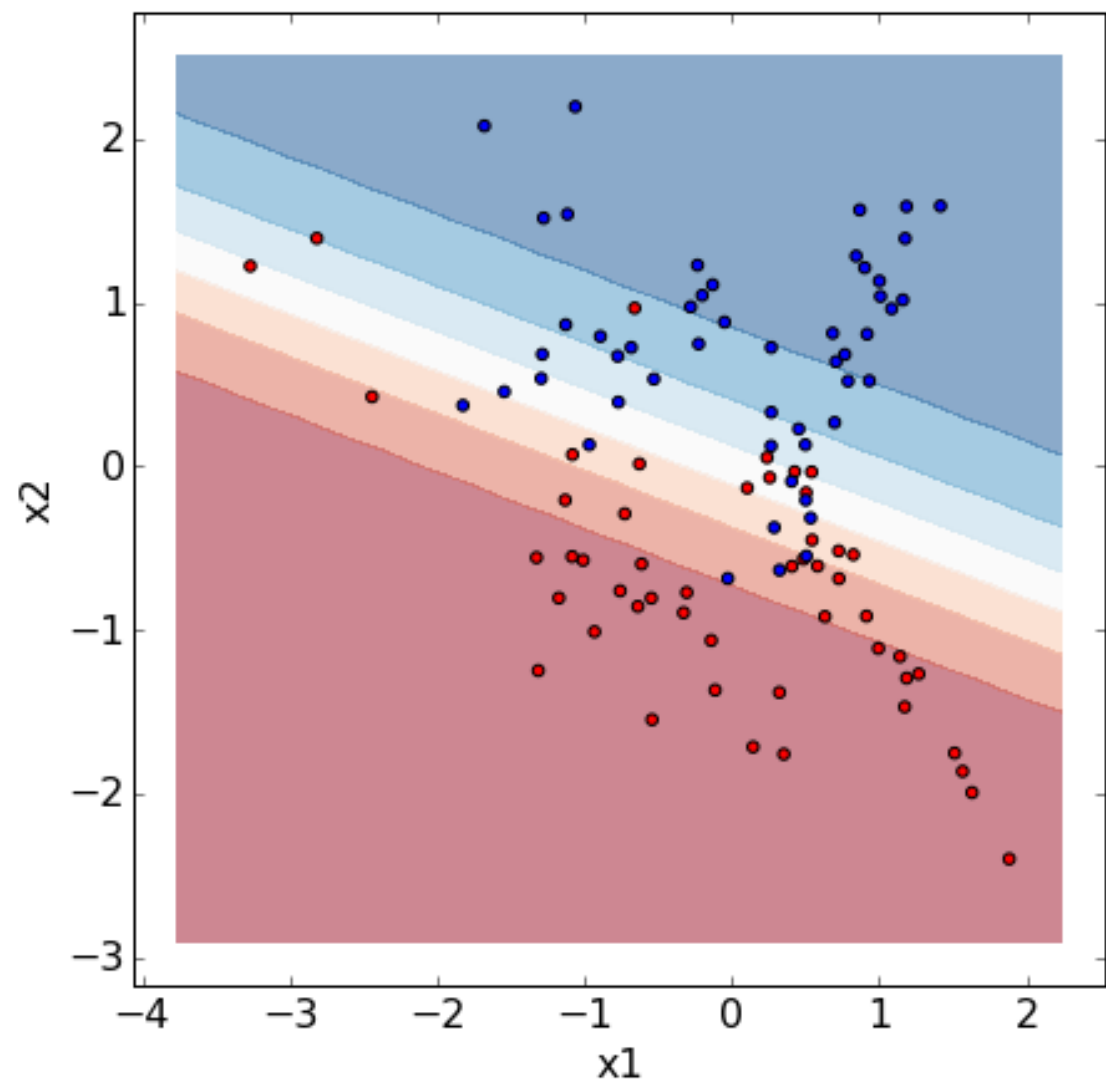




# Let's get real!

**7. Logistic regression**

**8. Non-linear transformations**



# *regularized logistic regression*

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

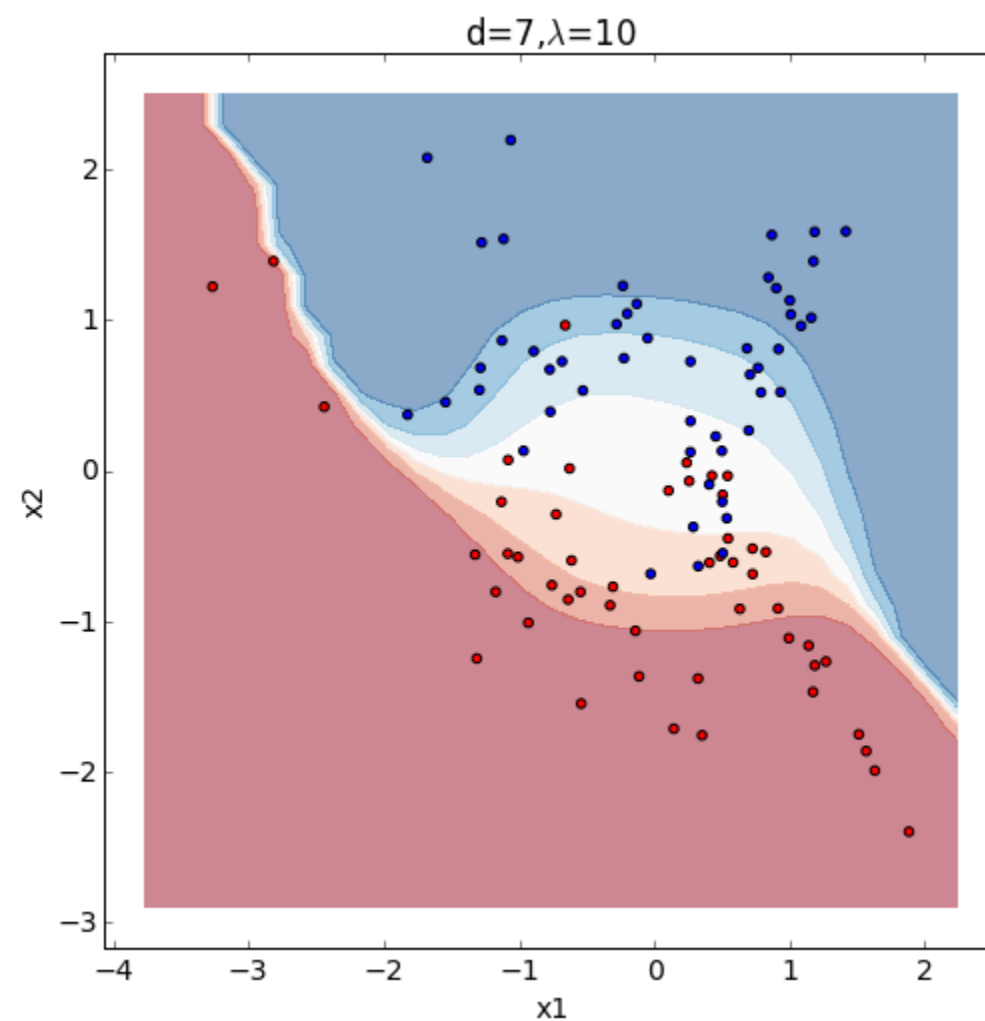
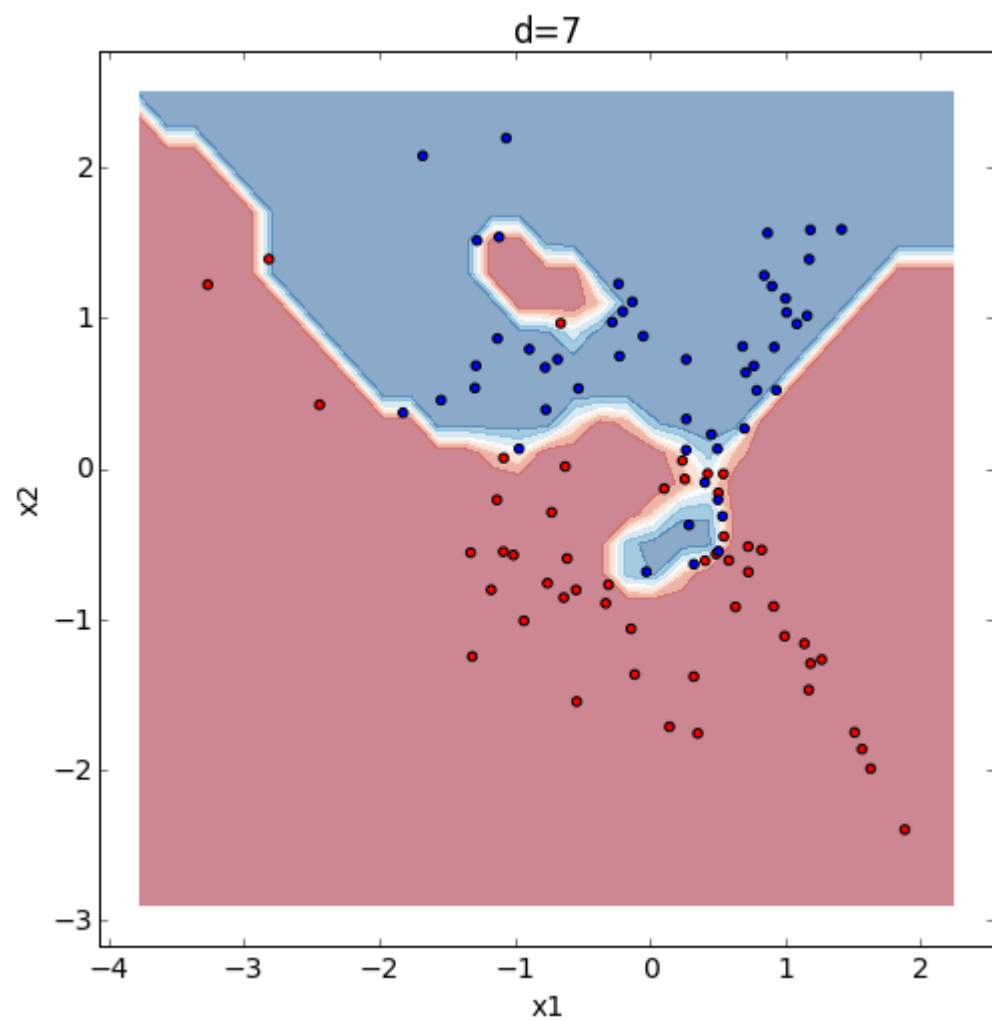
*cost function*

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

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

*regularized cost function*

# *regularized logistic regression*



# *support vector machines*

Fit a linear model

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

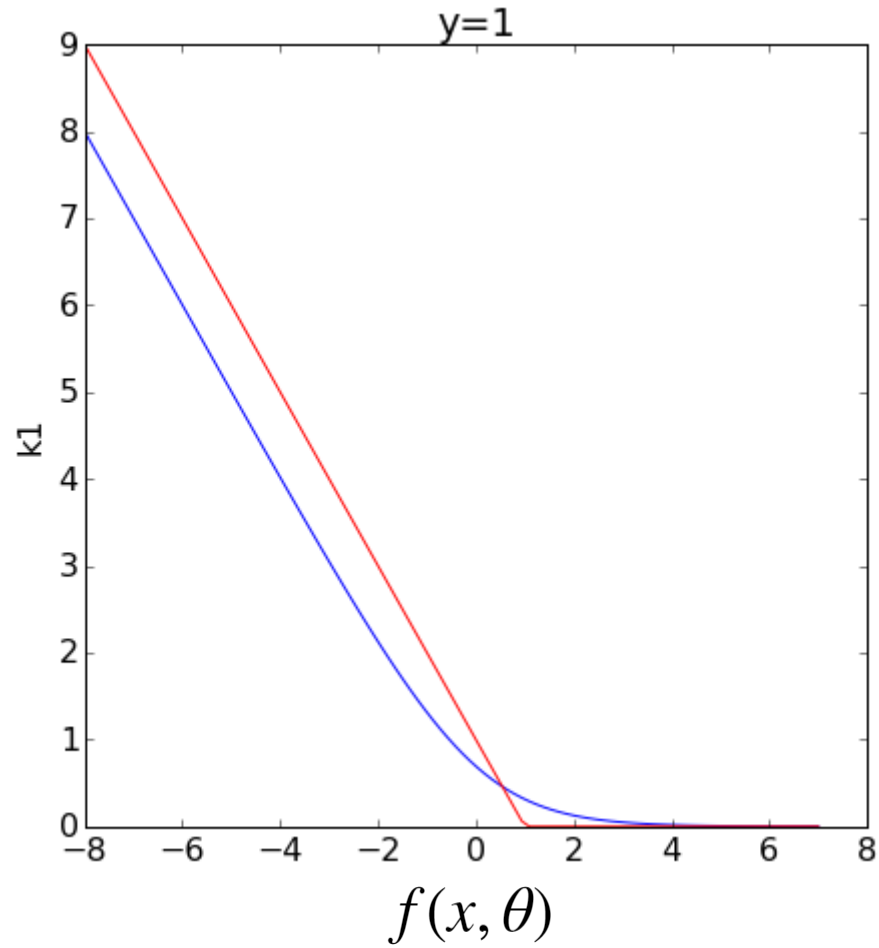
such that

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

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*

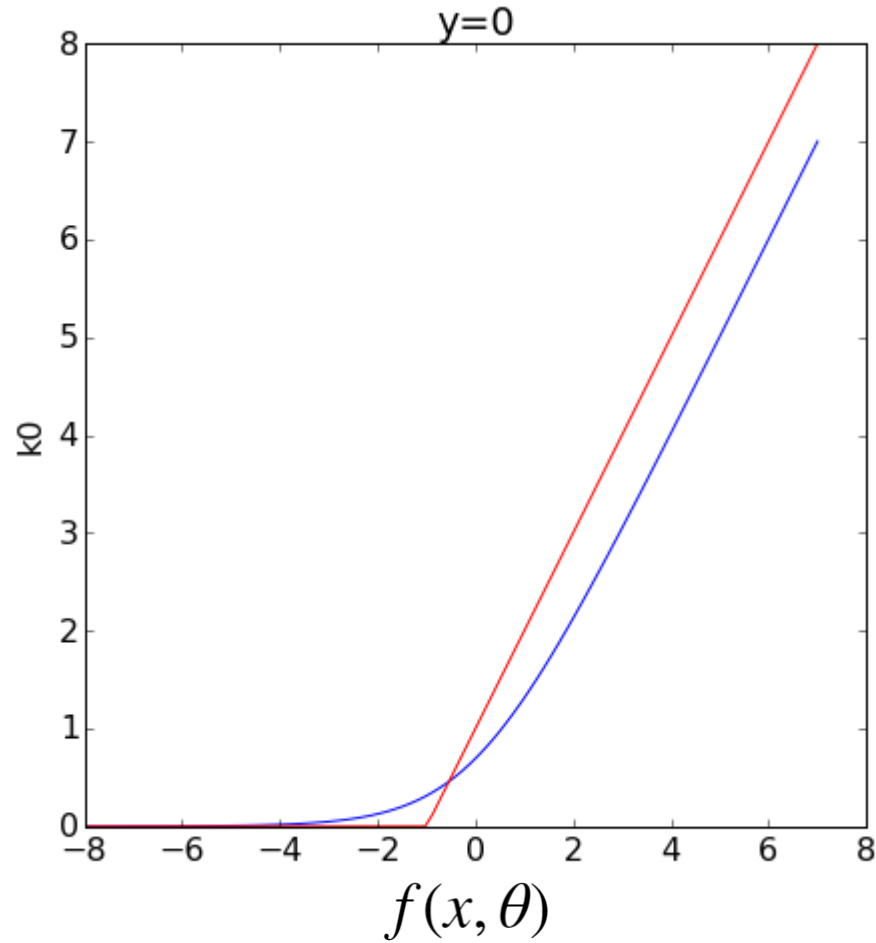


- replace cost function by piecewise linear function

- 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

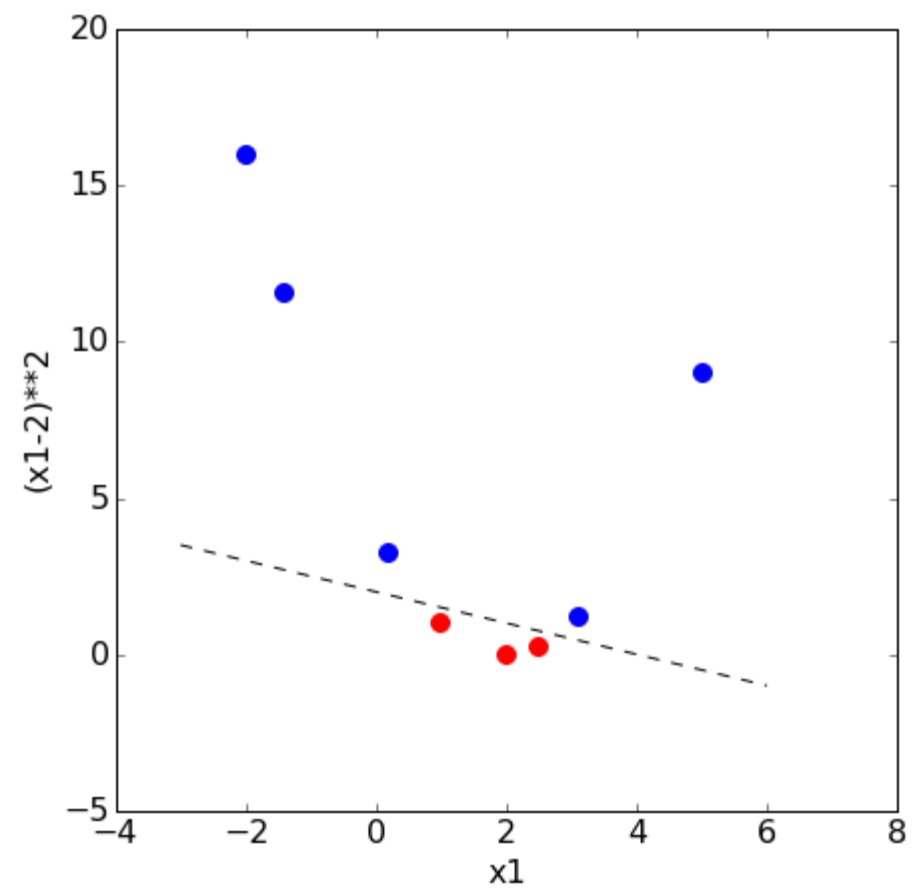
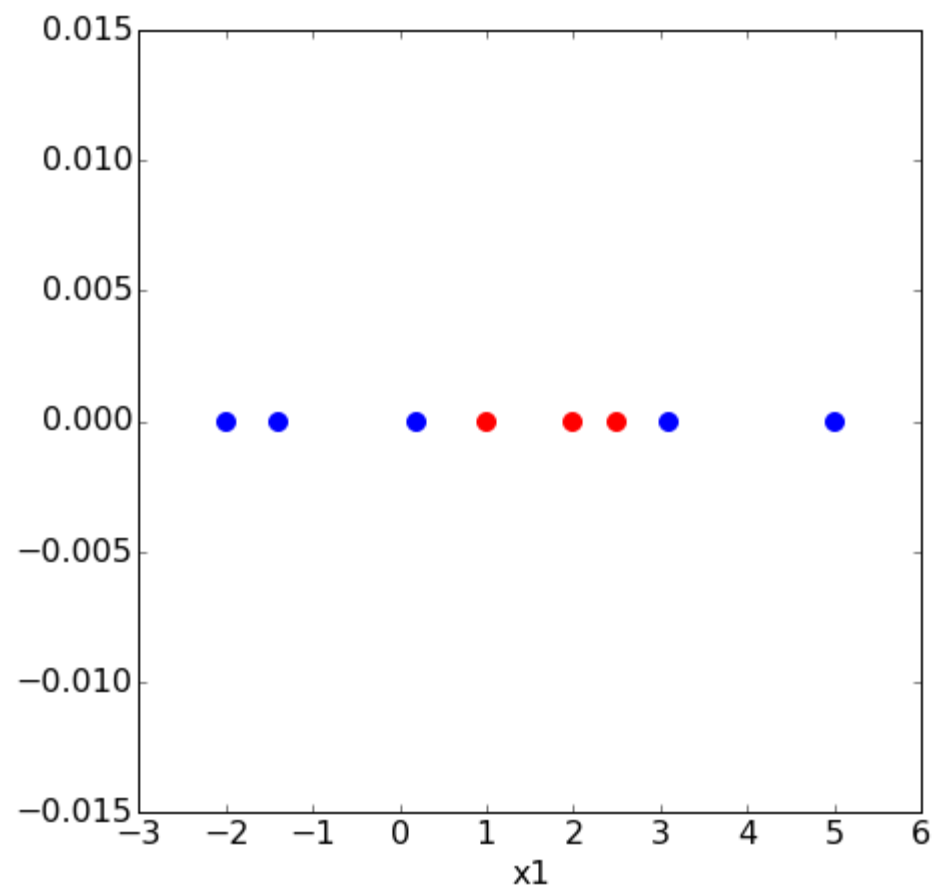
- if  $y = 1$  then the contribution to the cost is

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

- 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 known as a kernel function

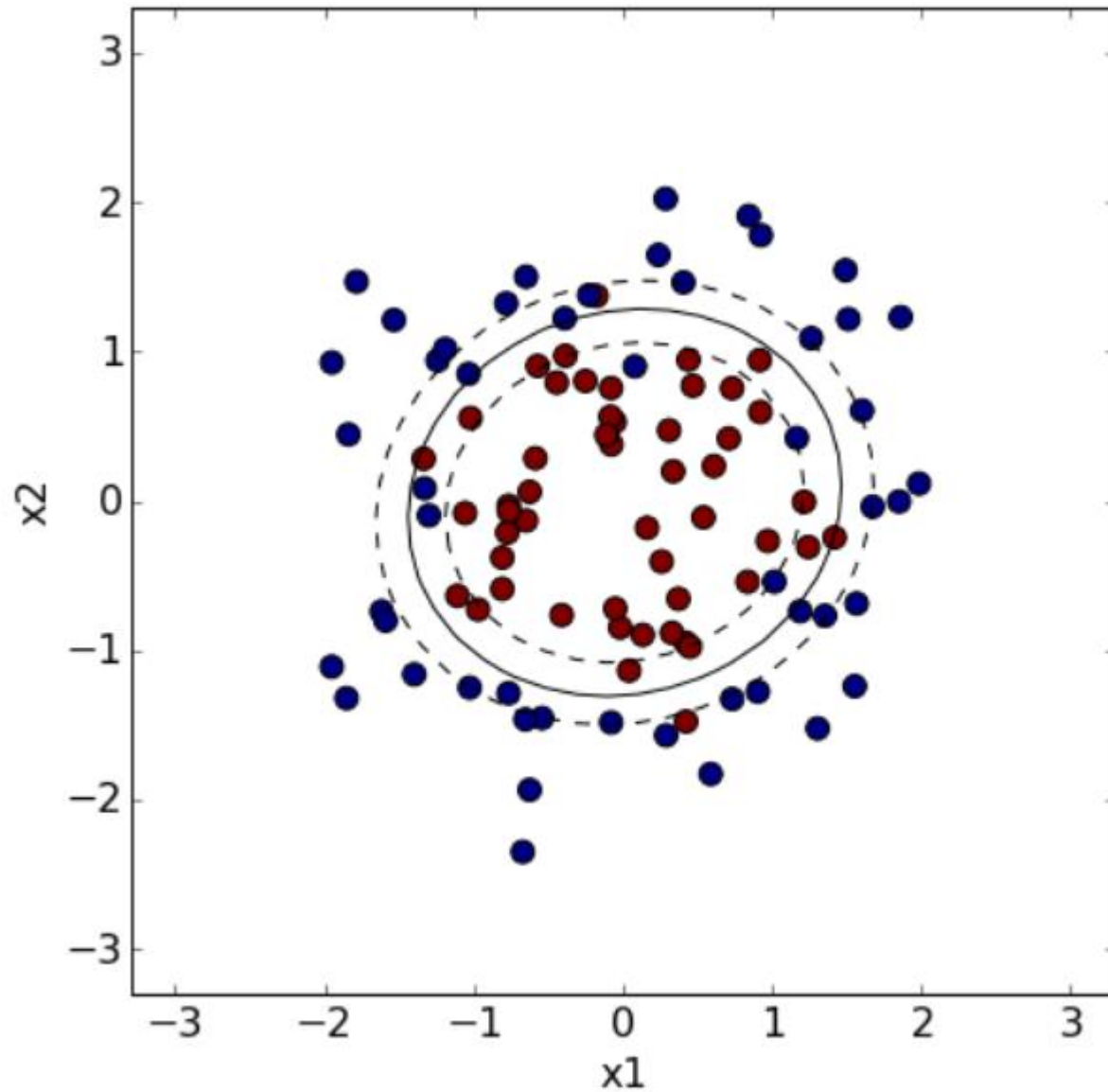
$$K(x^{(i)}, x^{(j)})$$

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.

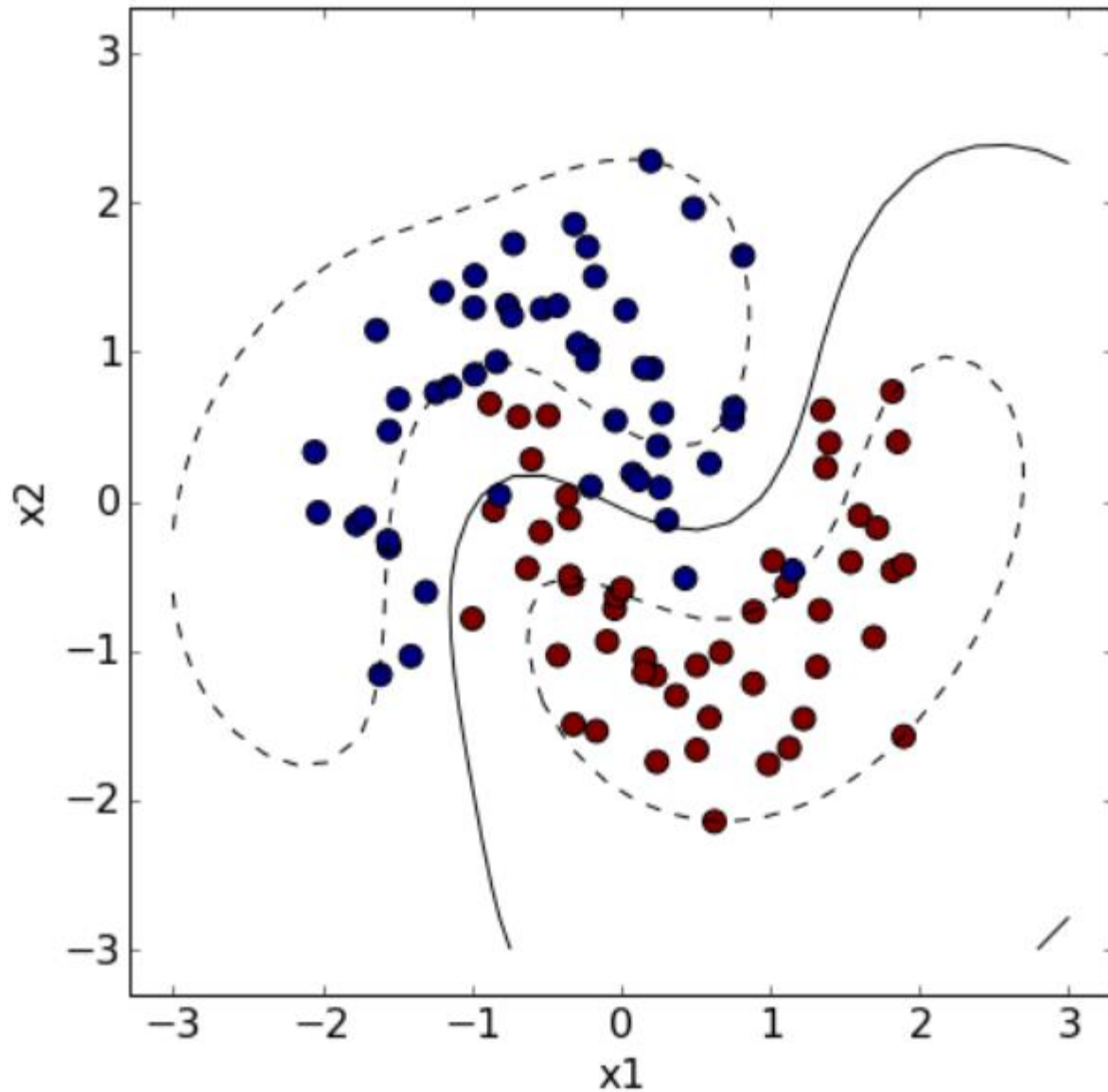
# *kernel support vector machines*



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

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

# *kernel support vector machines*



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

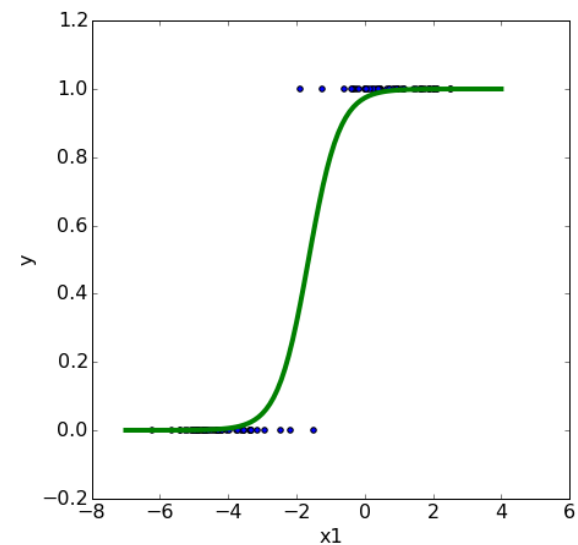
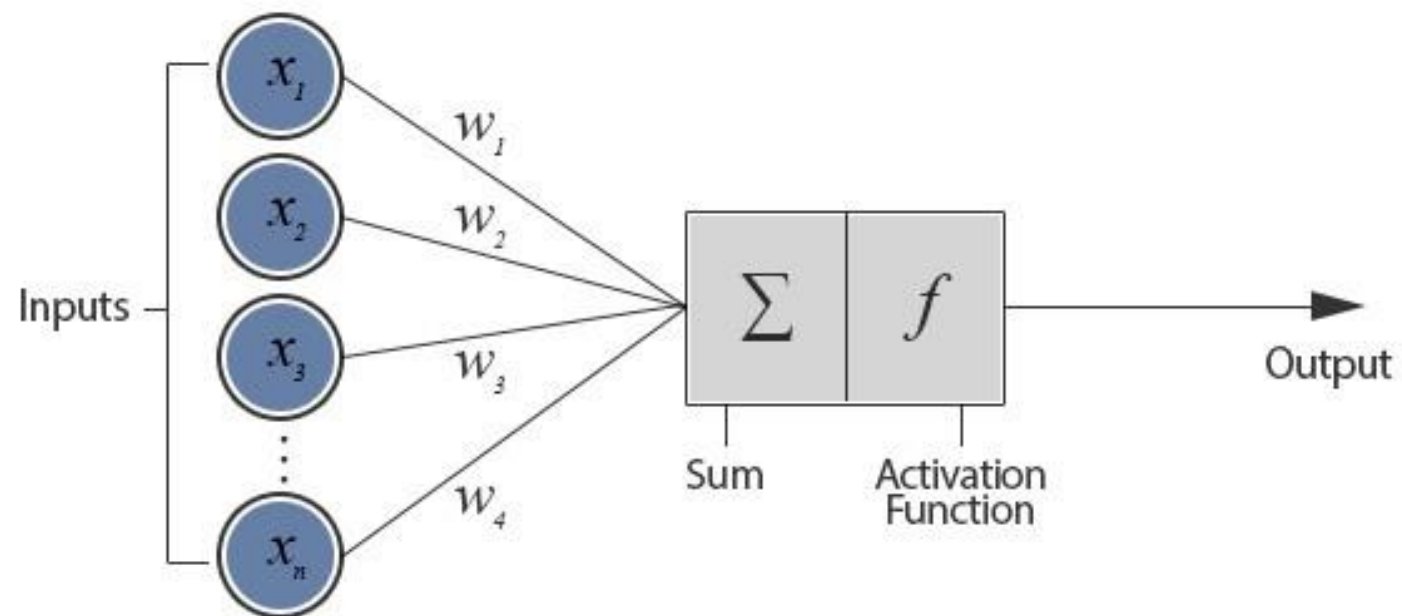
$$K(x^{(i)}, x^{(j)}) = \exp \left[ -\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2} \right]$$



Let's get real!

## 9. Support Vector Machines

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

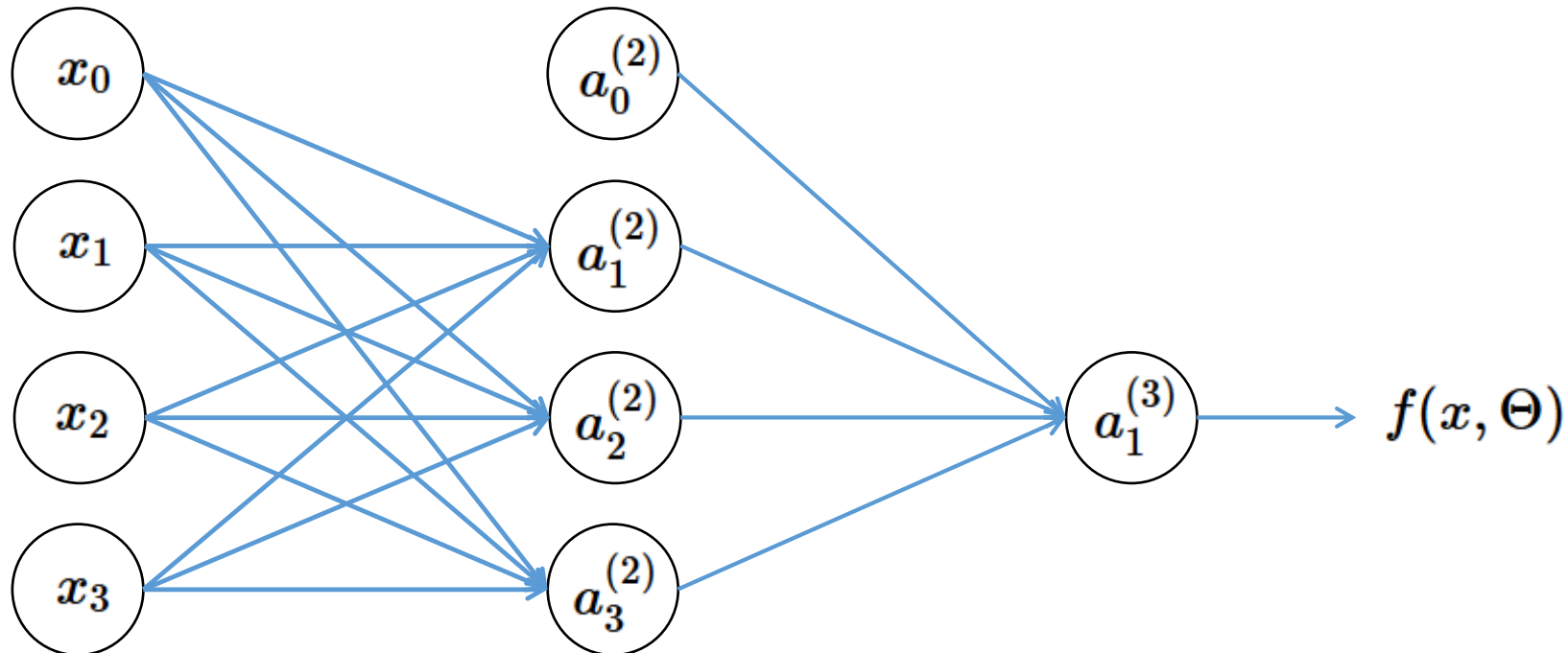


Model:  $f(x, \Theta) = g(\Theta_{10}^{(2)} a_0 + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$

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

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

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



Cost function logistic regression:

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

Cost function feedforward neural network:

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

<https://playground.tensorflow.org/>

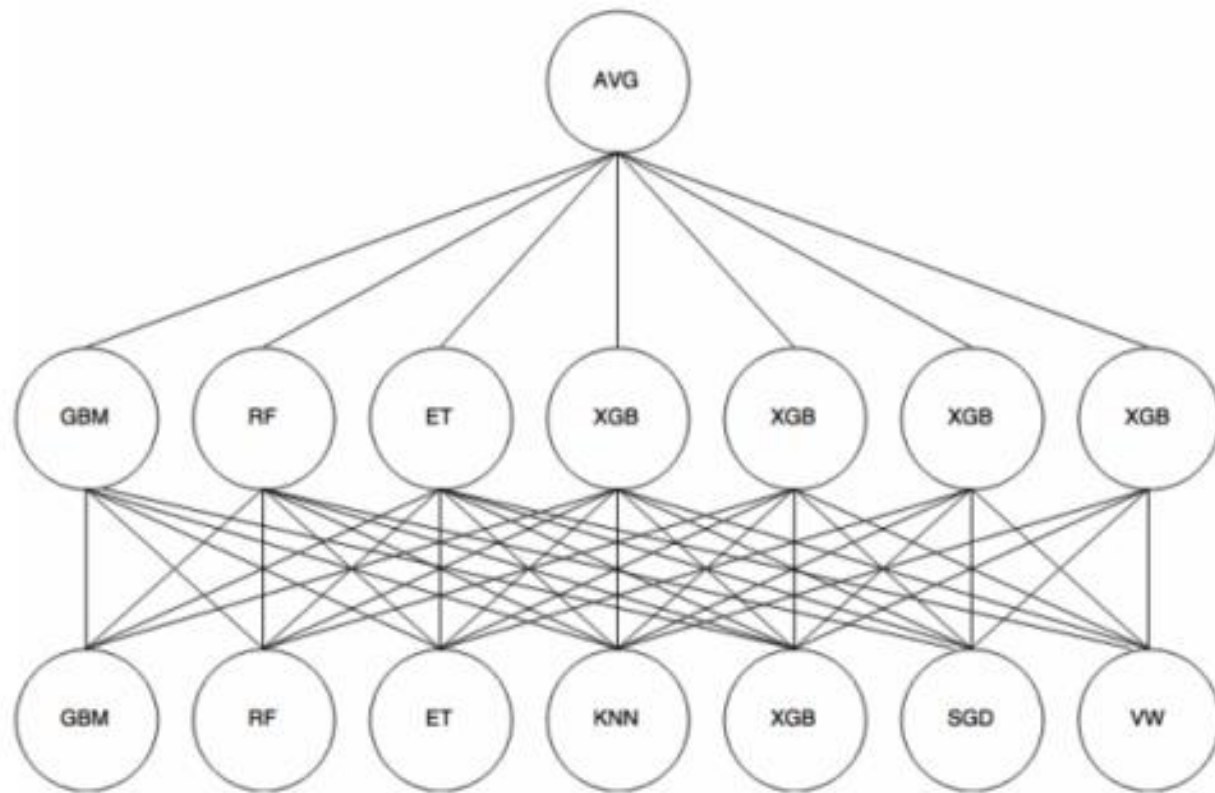




Let's get real!

## 10. Neural Networks

Blending/Stacking



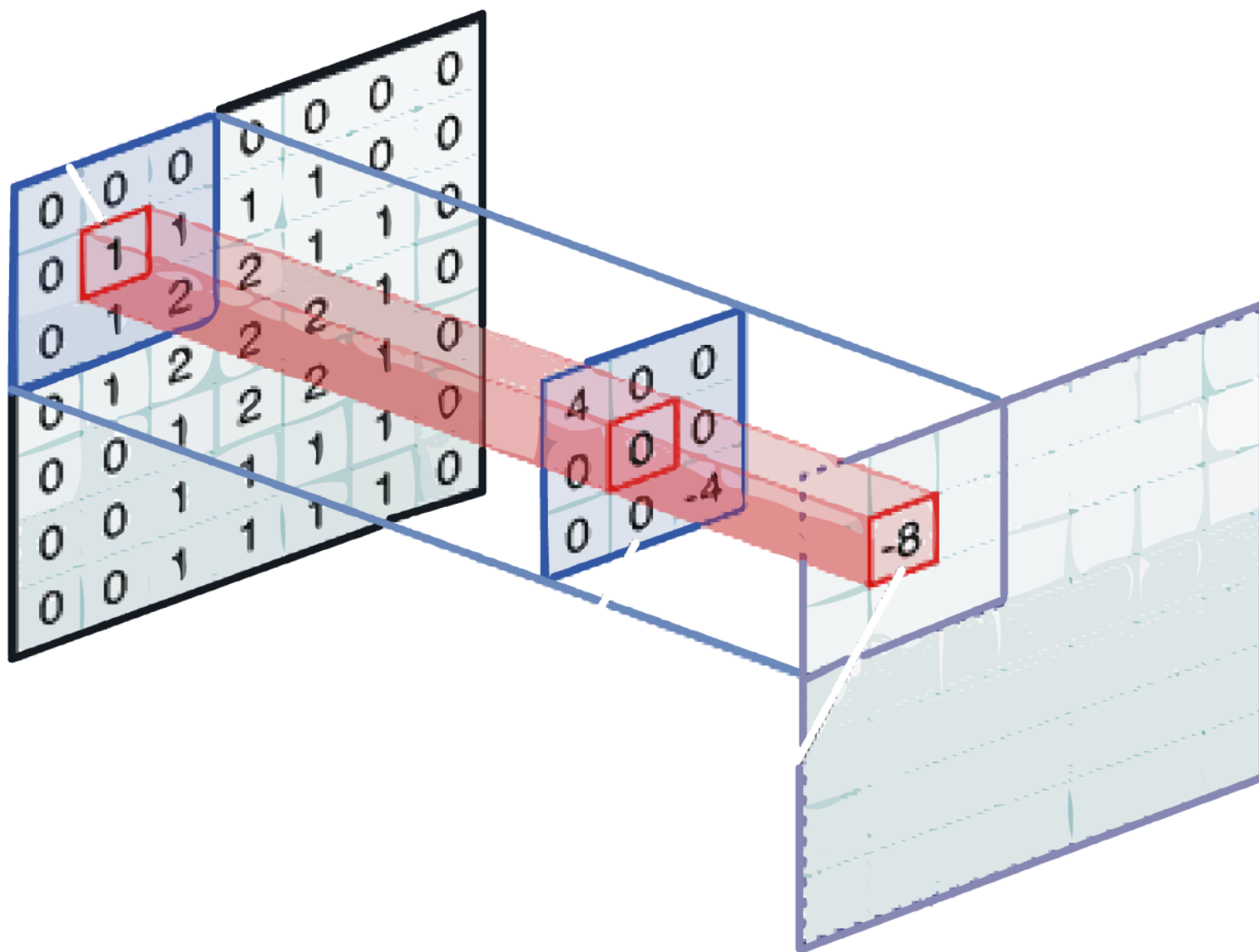
# Convolutional Neural Networks



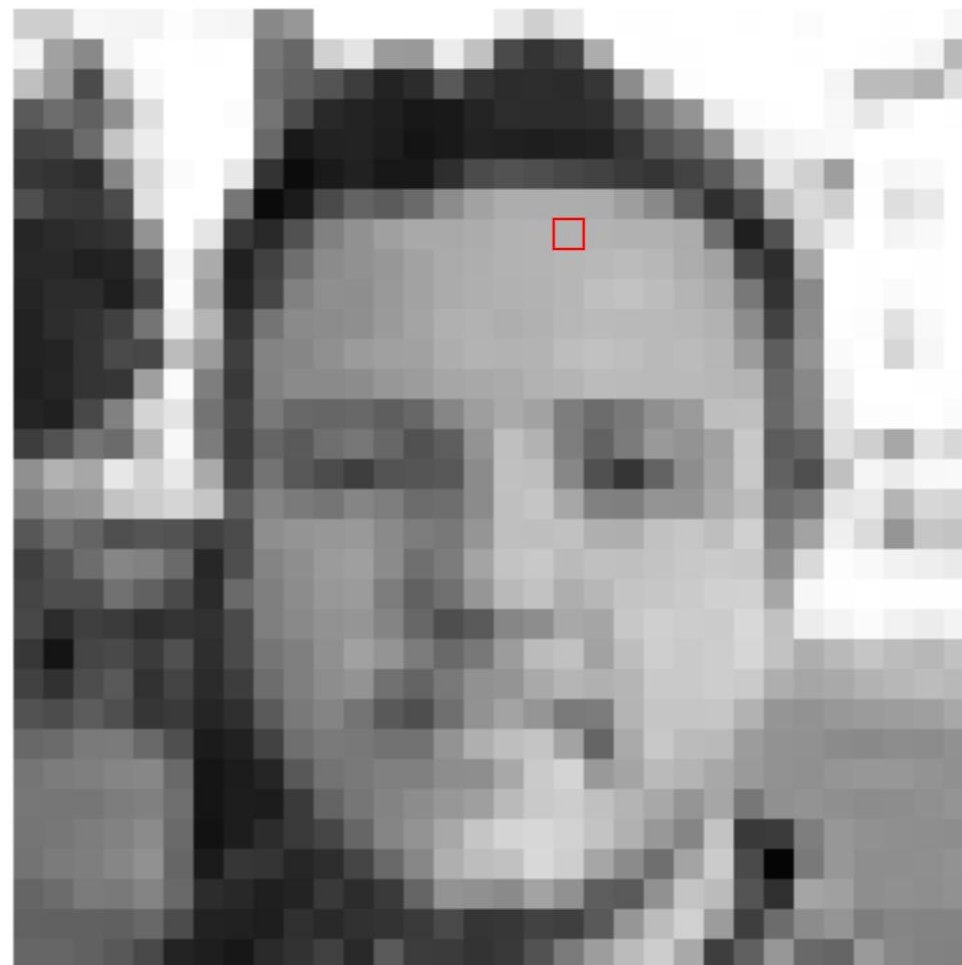
What We See

```
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



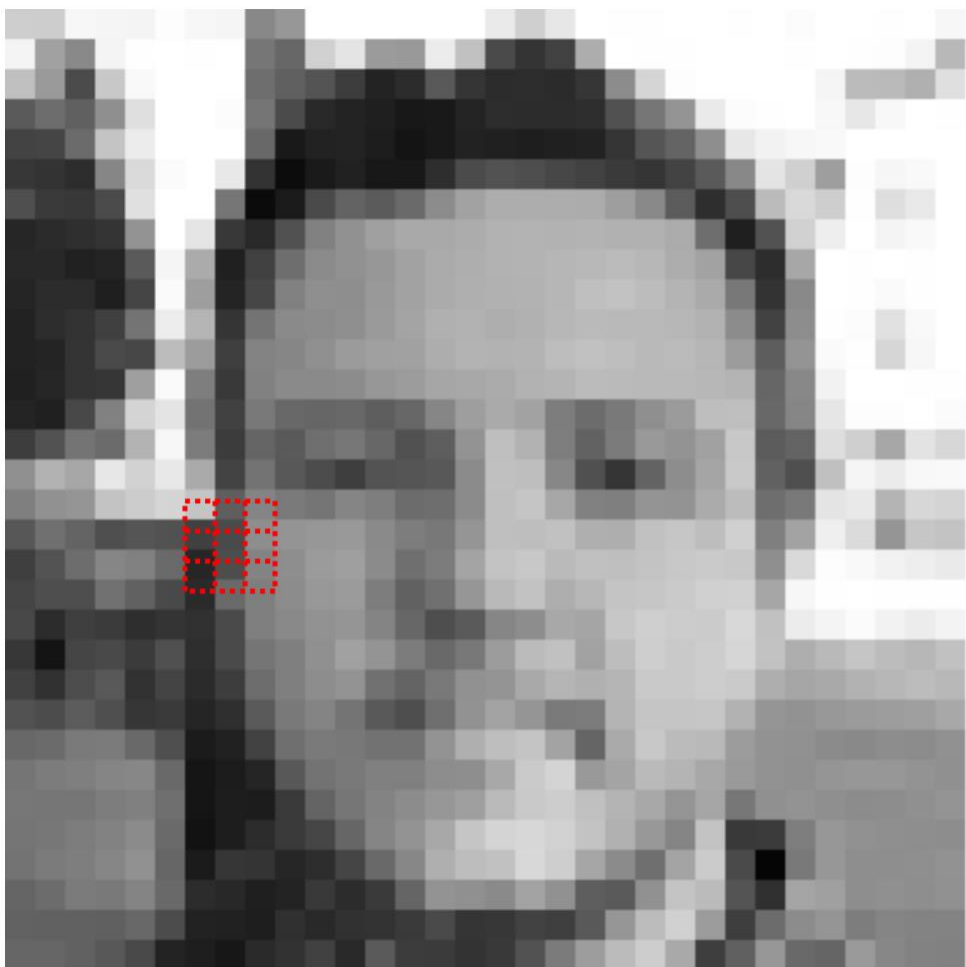
206 205 247 245 244 253 247 245 138 151 255 255 255 255 255 255 234 207 231 255 254 254 255 255 254 255 255 254 255 255 255 254 255 247  
244 181 137 244 254 255 254 255 118 103 209 228 155 153 238 103 74 52 88 173 255 254 254 255 255 255 254 255 254 253 244 184  
192 154 75 200 249 255 255 255 110 98 84 81 35 44 89 53 44 45 43 54 140 213 253 255 255 255 255 245 187 188 178 223  
90 109 98 143 223 255 255 255 117 75 41 35 31 24 25 38 45 44 44 48 81 118 148 234 252 254 255 248 231 248 255 254  
87 89 107 196 238 255 255 255 104 25 34 35 29 20 25 34 32 30 32 34 53 85 100 142 231 242 247 249 255 255 255  
55 51 45 134 218 251 255 232 51 12 28 33 24 24 48 75 82 78 71 88 58 53 87 90 138 228 208 158 253 248 249 255  
79 58 58 75 234 255 255 118 11 27 74 99 91 108 140 182 173 173 173 172 158 137 92 48 78 187 217 208 254 222 233 255  
38 43 47 52 147 255 229 58 41 81 129 145 180 189 189 172 178 175 178 179 177 177 172 110 31 82 209 238 255 244 249 255  
40 40 33 38 90 245 171 32 85 110 139 145 151 182 171 174 178 179 182 184 187 183 173 182 71 45 187 255 254 255 254 255  
37 44 44 31 89 250 158 38 70 129 143 142 153 182 171 175 177 178 182 191 194 188 180 170 120 51 137 255 254 250 254 255  
34 45 51 84 116 237 181 53 118 138 140 143 154 184 178 178 174 177 183 188 185 185 183 178 140 88 141 254 252 225 249 255  
34 38 52 74 71 188 158 83 131 134 144 155 180 181 173 179 178 179 189 193 190 185 187 182 158 93 148 250 254 214 247 255  
32 38 52 54 159 250 128 57 129 138 138 140 151 158 188 188 171 178 180 187 188 185 185 183 180 102 138 242 255 255 254 254  
38 32 72 129 212 228 115 85 121 104 102 104 94 103 134 158 170 182 125 108 121 143 155 190 191 104 134 230 253 253 255 251  
81 82 118 107 179 247 124 80 101 90 111 119 103 81 94 147 191 178 128 98 123 153 147 181 200 92 100 222 207 187 227 215  
144 178 187 231 219 232 170 87 115 88 78 82 83 85 88 139 192 190 135 80 53 99 141 185 201 97 79 192 245 235 248 249  
127 145 149 195 204 213 197 95 133 122 117 133 128 108 110 139 191 197 187 129 127 148 147 171 188 110 121 228 233 180 215 212  
87 112 100 79 85 82 85 75 142 148 151 153 138 125 120 149 191 190 193 175 174 193 198 190 208 127 183 239 219 149 198 195  
83 83 109 134 129 108 39 78 132 142 155 159 139 111 124 184 195 200 188 192 191 195 200 202 200 143 217 253 249 242 238 234  
89 78 78 113 97 74 43 108 127 140 152 155 125 97 112 150 185 194 174 183 198 198 202 208 209 188 247 254 255 254 254 254  
72 44 83 59 48 52 49 74 127 137 148 149 132 103 78 90 134 141 188 185 199 207 204 203 216 193 238 244 251 242 238 243  
55 20 89 73 59 80 48 74 117 127 144 181 148 124 105 120 158 187 193 182 189 208 201 205 214 194 174 185 197 188 183 193  
85 49 77 89 50 88 43 81 109 127 141 147 113 100 121 145 148 189 181 178 181 201 201 205 202 174 188 189 178 183 188 184  
82 78 92 79 54 58 37 47 90 121 132 118 89 78 111 148 183 149 122 124 180 197 197 198 178 149 148 152 155 157 159 188  
104 107 122 123 105 79 27 33 88 111 122 120 114 114 147 175 190 198 183 101 170 200 187 185 158 148 145 139 137 141 140 145  
117 124 127 133 135 105 21 28 37 88 115 121 128 128 141 142 188 202 212 153 184 188 180 188 154 148 144 149 151 151 147 144  
119 118 118 125 128 111 21 29 28 58 100 118 131 140 151 159 188 201 205 192 180 188 149 188 119 144 147 143 140 141 144 148  
117 119 125 130 139 108 18 29 44 58 70 102 133 147 188 197 212 215 210 195 177 152 133 195 57 59 128 151 145 143 142 141  
115 123 128 134 145 102 27 54 52 38 45 89 105 135 175 189 193 218 208 188 139 111 184 203 74 5 121 151 142 142 143 148  
101 108 123 121 132 105 44 40 31 35 57 44 58 101 147 144 138 183 145 94 90 145 198 187 84 48 185 180 142 144 142 145  
98 97 97 98 104 78 34 33 30 48 41 49 51 58 74 53 55 88 83 89 150 188 209 158 82 108 140 149 125 133 131 131  
102 102 97 88 73 35 30 23 42 50 85 41 90 80 59 51 57 82 123 157 187 205 189 82 98 151 105 101 154 135 130 129



blur ▼

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



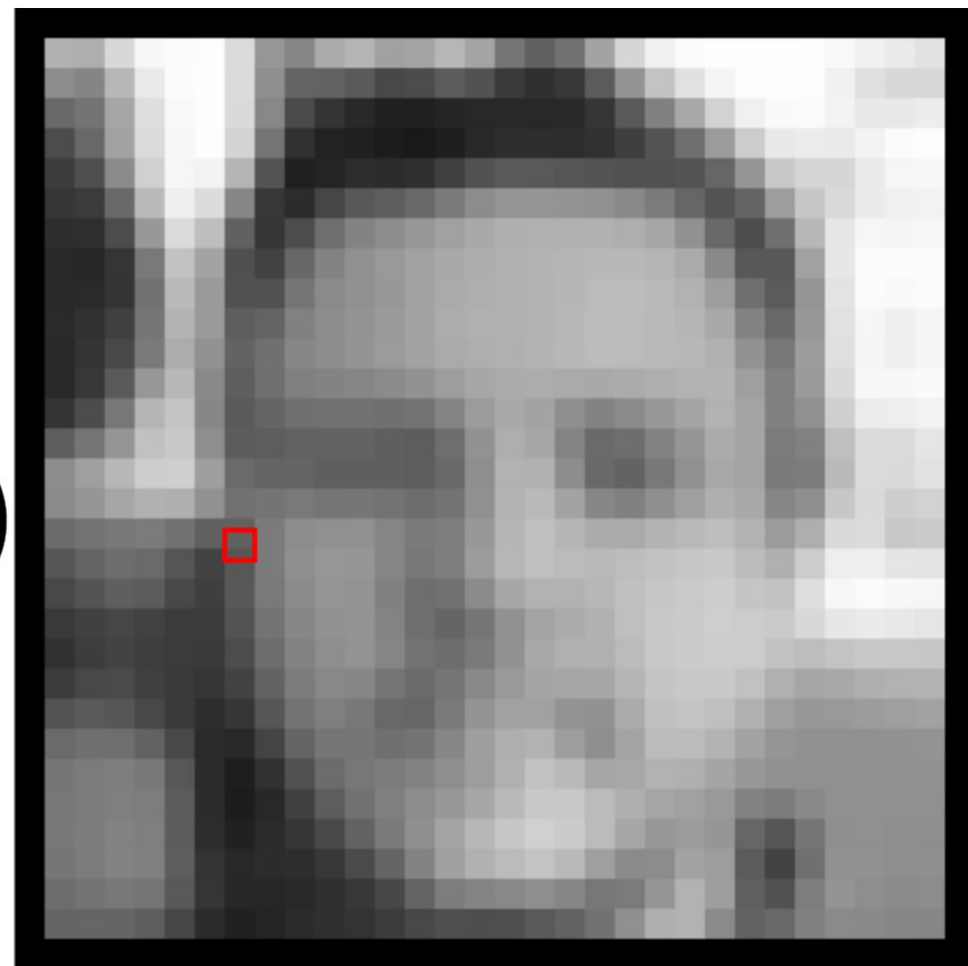


input image

$$\begin{aligned}
 & \left( \begin{array}{ccc} 57 & + & 129 & + & 138 \\ \times 0.0625 & \times 0.125 & \times 0.0625 \end{array} \right. \\
 & + \begin{array}{ccc} 65 & + & 121 & + & 104 \\ \times 0.125 & \times 0.25 & \times 0.125 \end{array} \\
 & \left. + \begin{array}{ccc} 60 & + & 101 & + & 90 \\ \times 0.0625 & \times 0.125 & \times 0.0625 \end{array} \right) \\
 & = 102
 \end{aligned}$$

kernel:

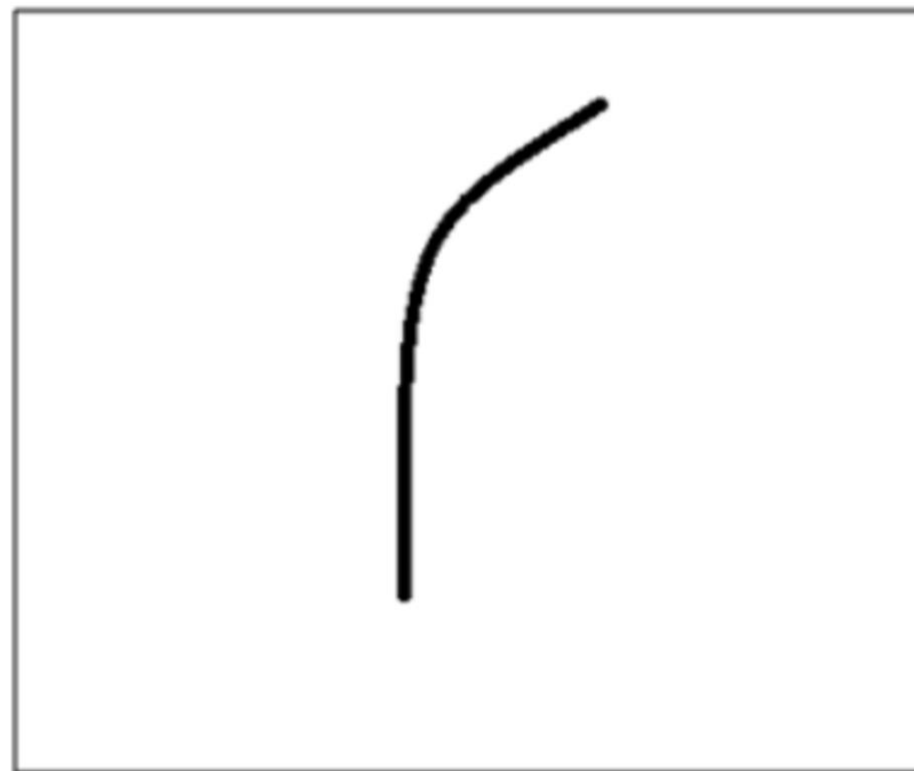
blur ▼



output image

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

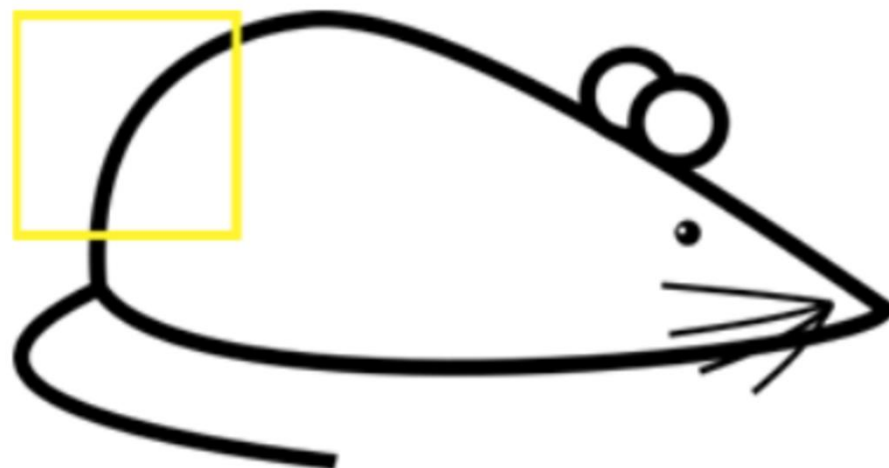
Pixel representation of filter



Visualization of a curve detector filter



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



Visualization of the filter on the image

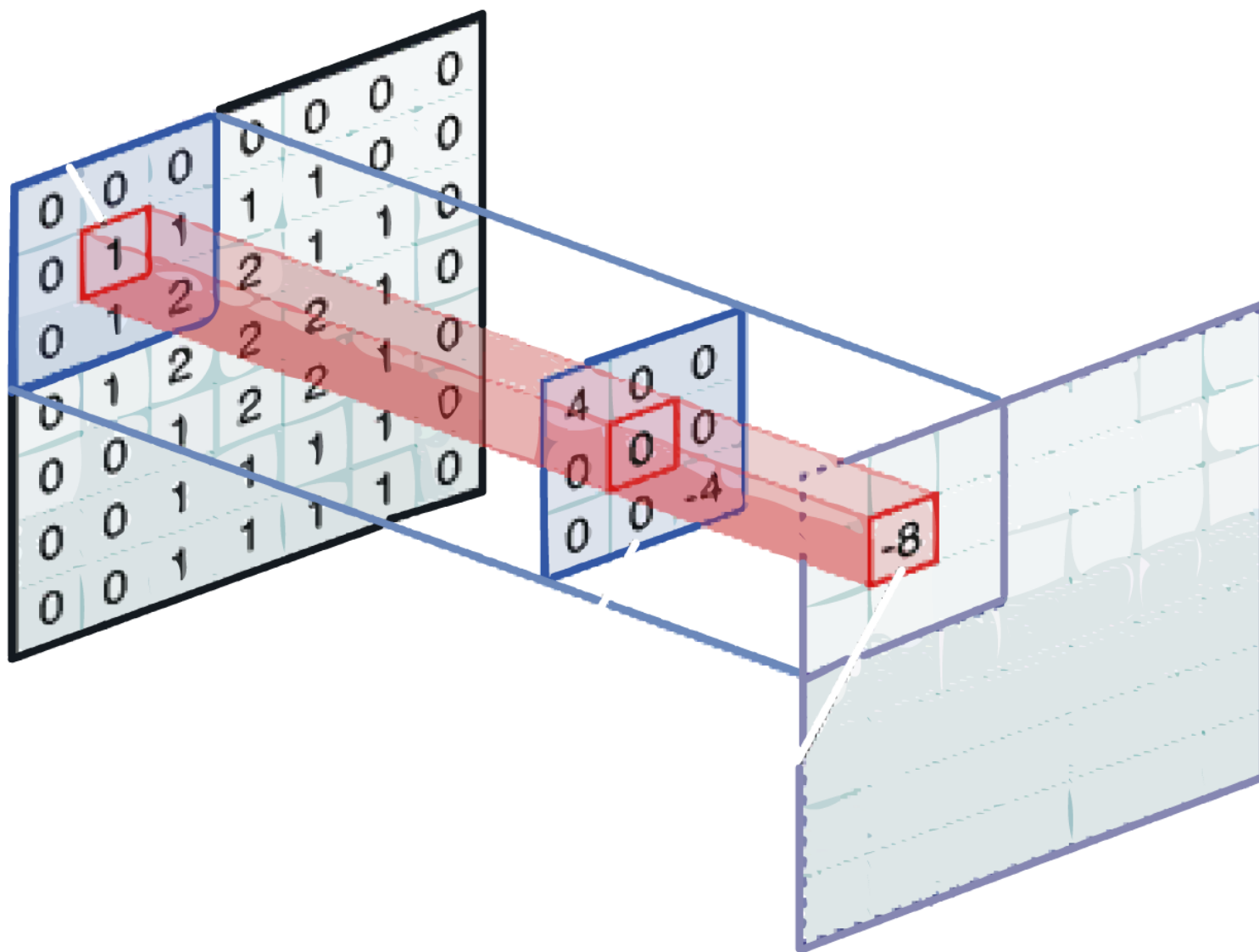
0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

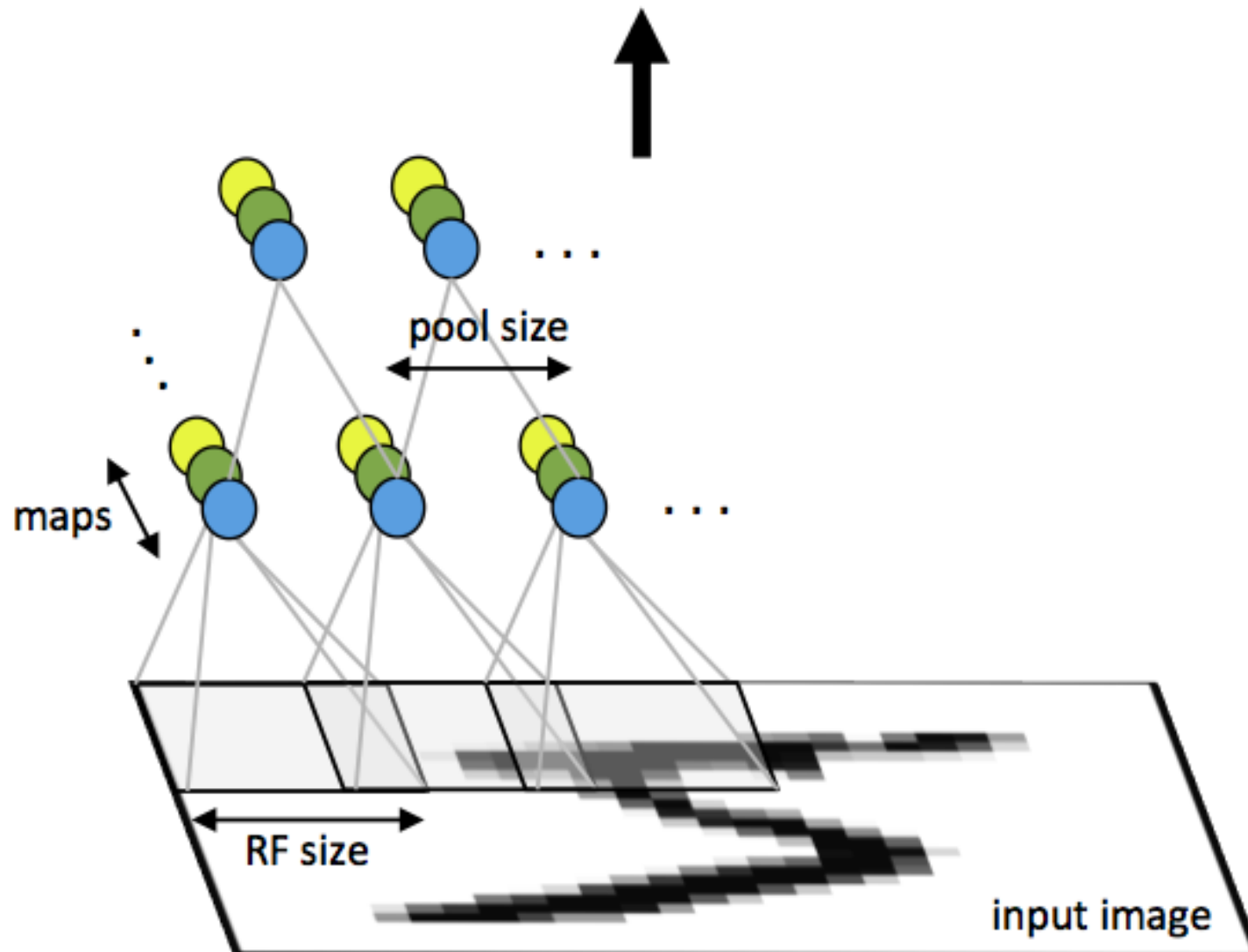
Pixel representation of receptive field

\*

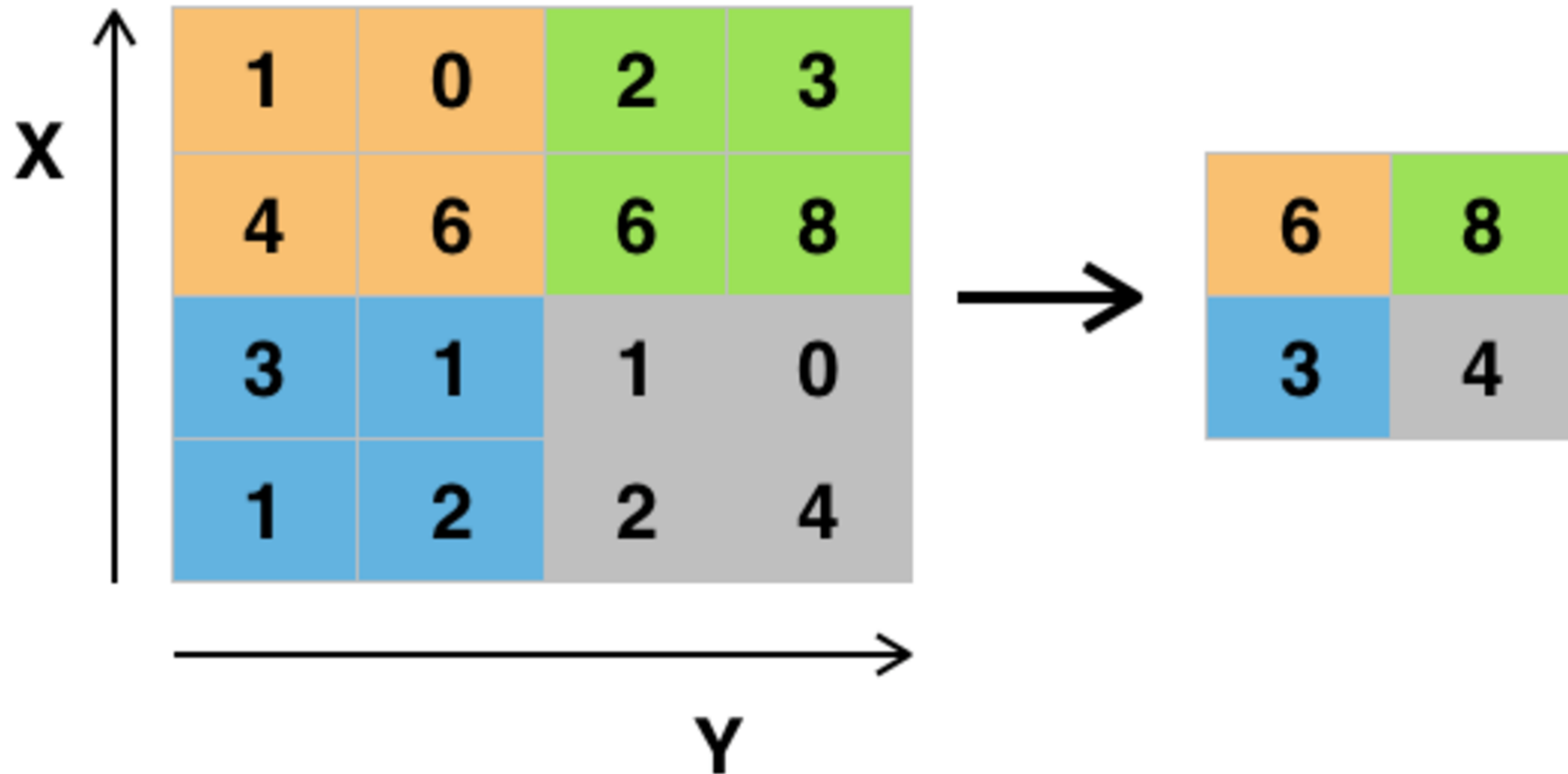
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

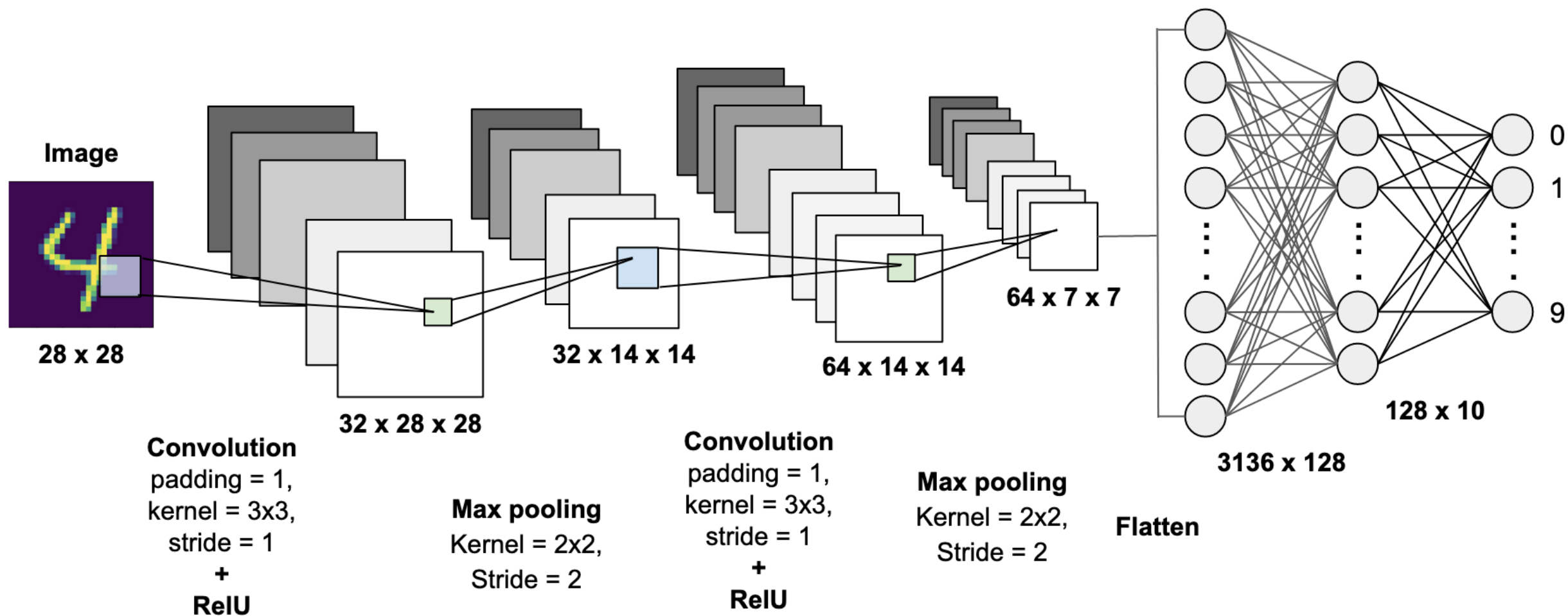




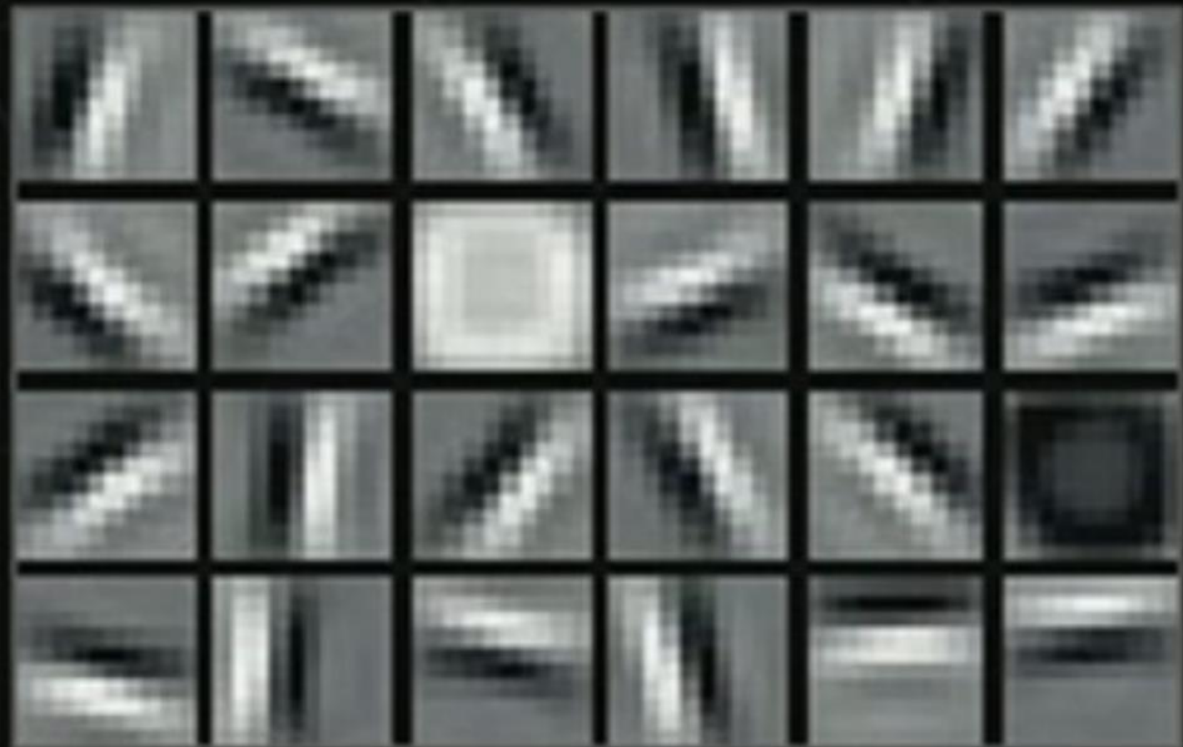
## Single depth slice



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



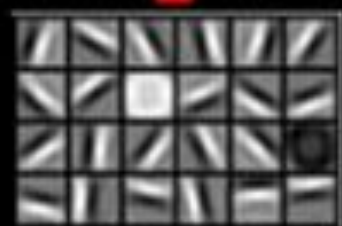




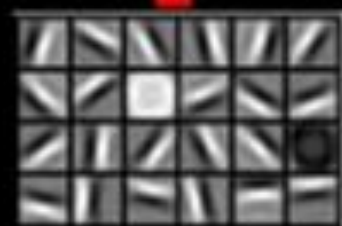
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24
25	26	27	28	29
30	31	32	33	34
35	36	37	38	39
40	41	42	43	44
45	46	47	48	49
50	51	52	53	54
55	56	57	58	59
60	61	62	63	64

65	66	67	68	69
70	71	72	73	74
75	76	77	78	79
80	81	82	83	84
85	86	87	88	89
90	91	92	93	94
95	96	97	98	99
100	101	102	103	104
105	106	107	108	109
110	111	112	113	114
115	116	117	118	119
120	121	122	123	124

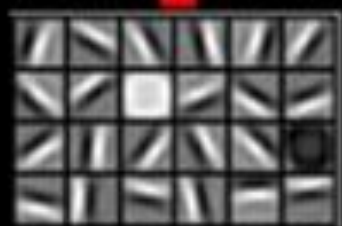
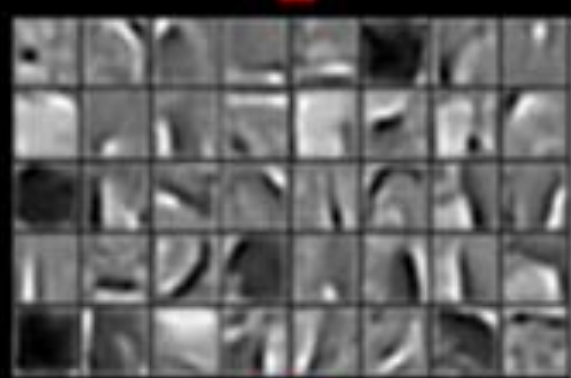
Faces



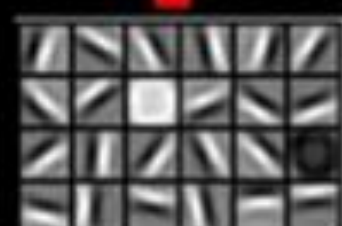
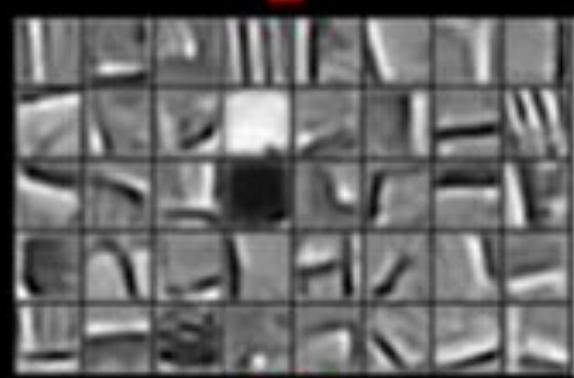
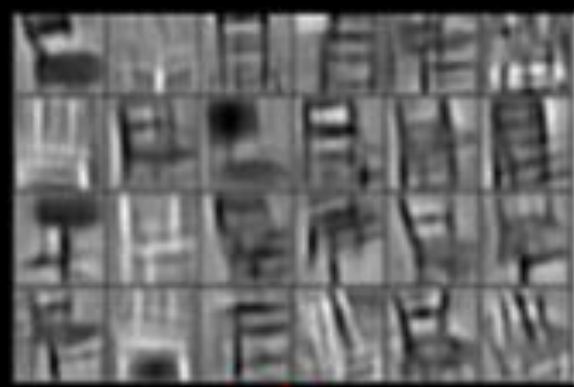
Cars



Elephants



Chairs

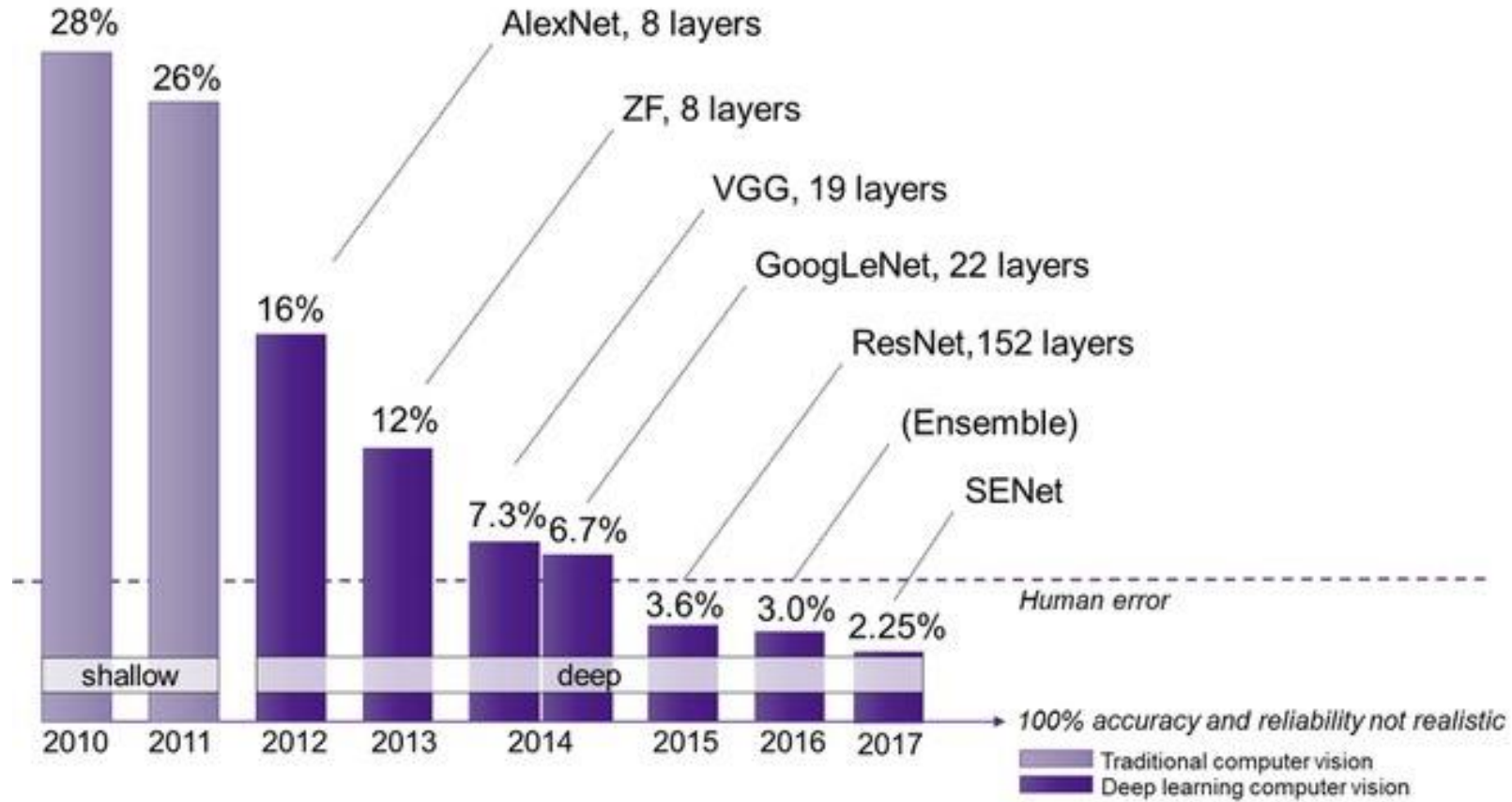






- 14.197.122 images
- 21.841 subcategories
- 27 high-level categories

## ImageNet Large Scale Visual Recognition Challenge







Let's get real!

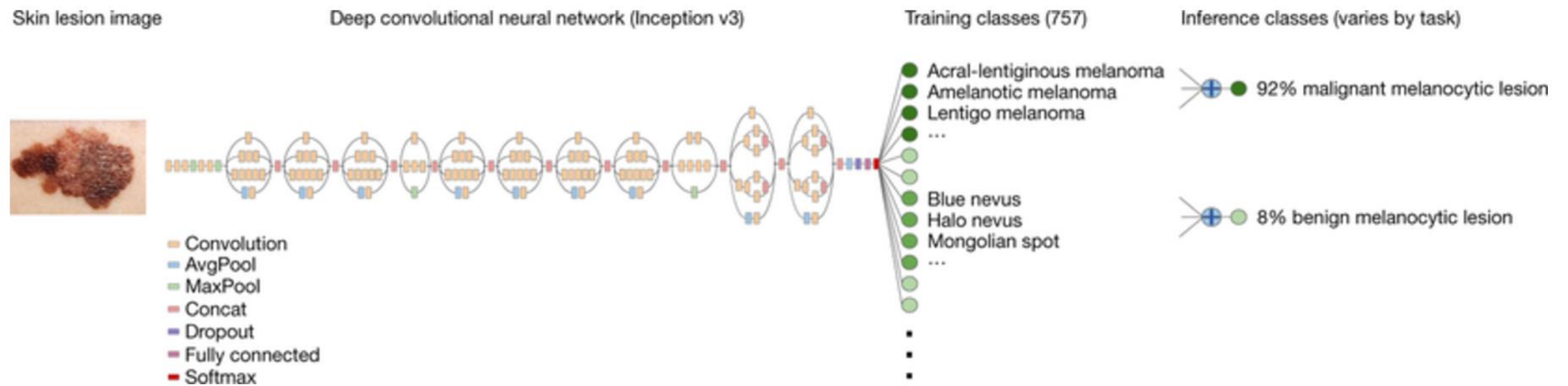
Cats & dogs

# Deep learning algorithm diagnoses skin cancer as well as seasoned dermatologists

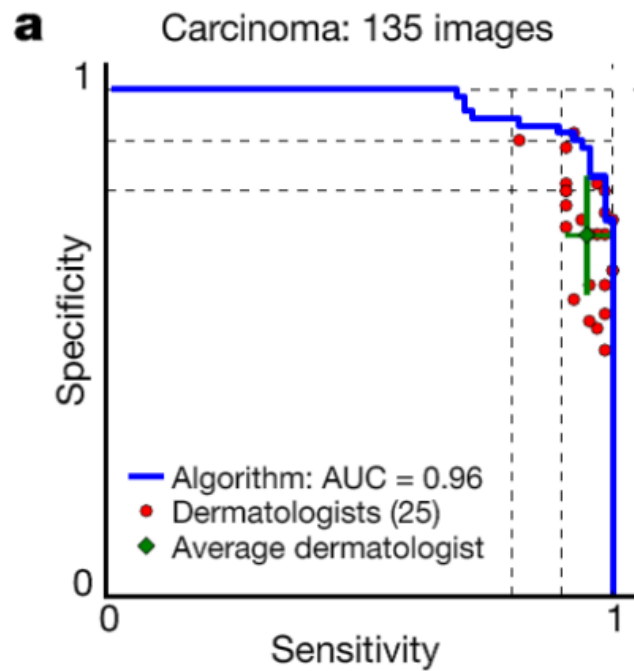
By Jessica Hall on January 25, 2017 at 1:25 pm | 16 Comments

422  
shares

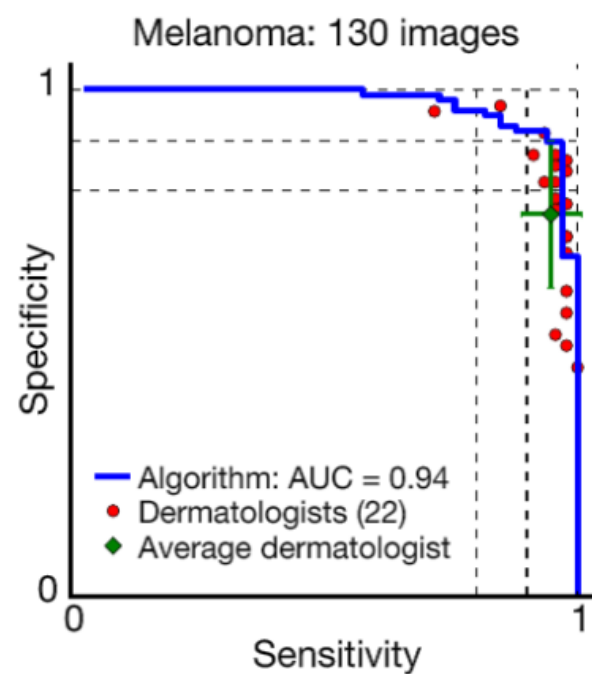




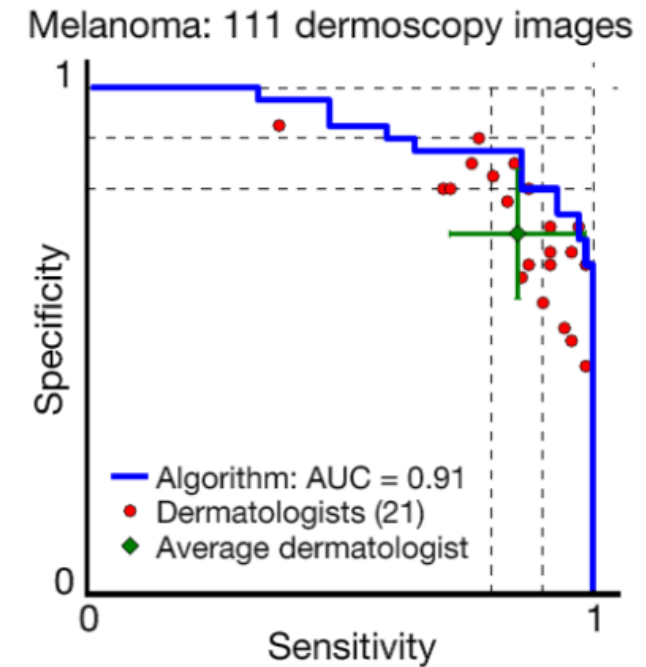




keratinocyte carcinomas: 65  
benign seborrheic keratoses: 75



malignant melanomas: 33  
benign nevi: 97



malignant melanomas: 71  
benign nevi: 40

Branch: master ▾	<a href="#">models</a> / <a href="#">research</a> / <a href="#">inception</a> /	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

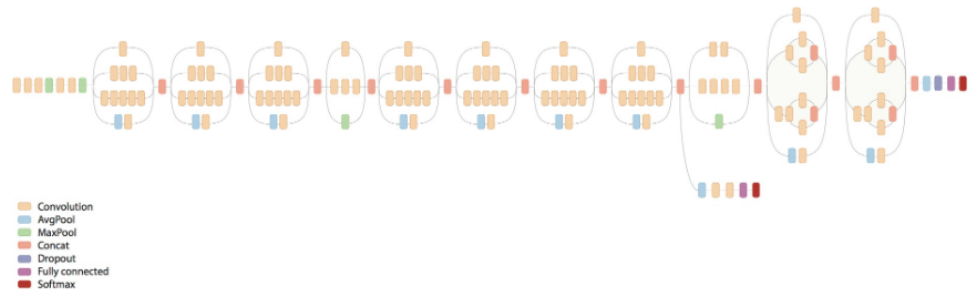
[ImageNet](#) is a common academic data set in machine learning for training an image recognition system. Code in this directory demonstrates how to use TensorFlow to train and evaluate a type of convolutional neural network (CNN) on this academic data set. In particular, we demonstrate how to train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

*Rethinking the Inception Architecture for Computer Vision*

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

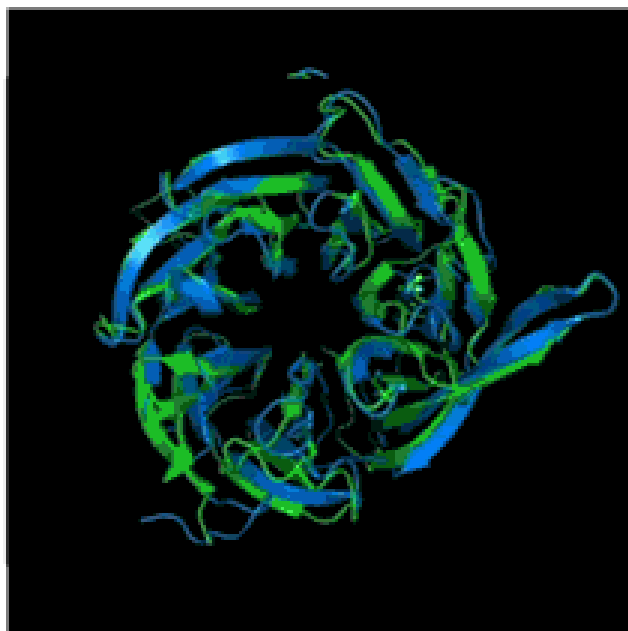
<http://arxiv.org/abs/1512.00567>

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.

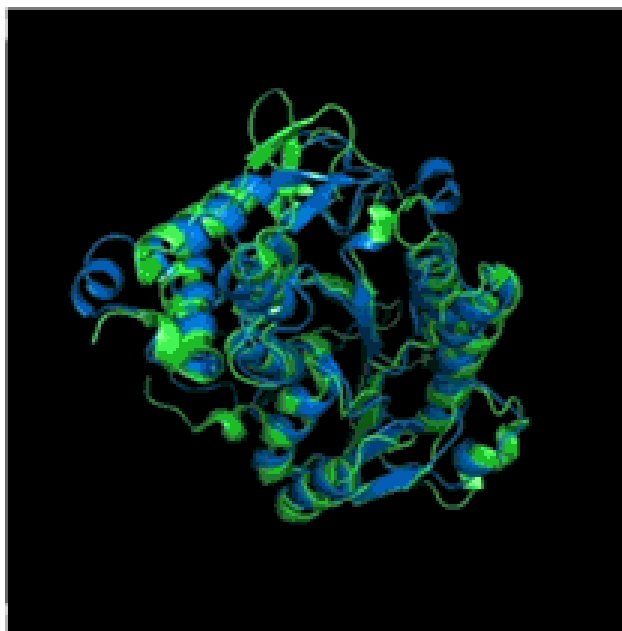


## Critical Assessment of Techniques for Protein Structure Prediction

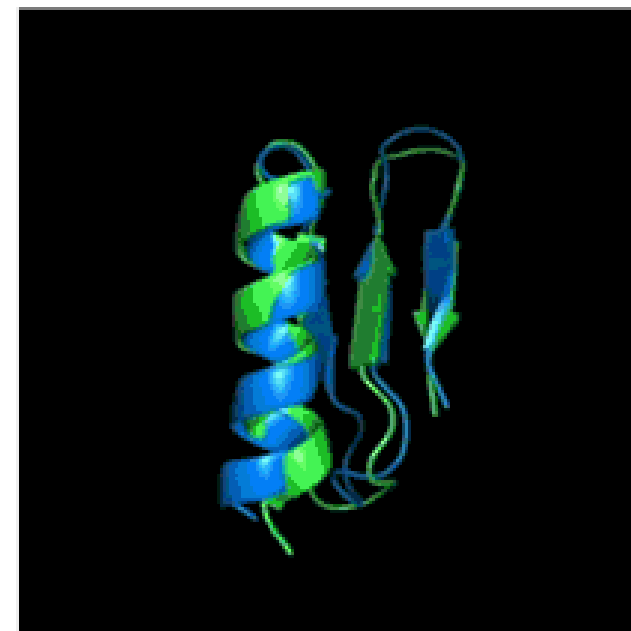
**T0954 / 6CVZ**



**T0965 / 6D2V**



**T0955 / 5W9F**



Structures:  
Ground truth (green)  
Predicted (blue)

