

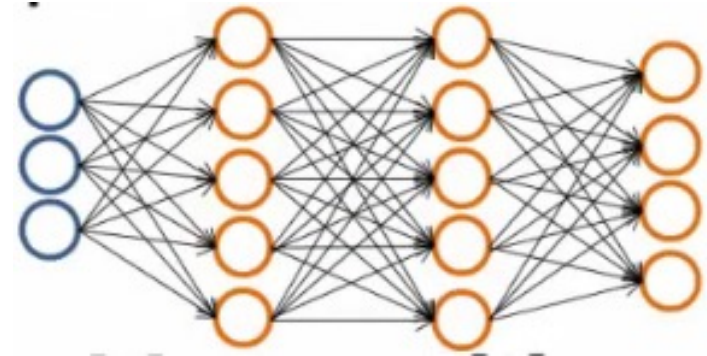
Lecture 10
Convolutional Neural Networks
For Computer Vision

Why yet another algorithm, *again*?

Last week(s): Full Connected **Artificial Neural Networks** (ANN)

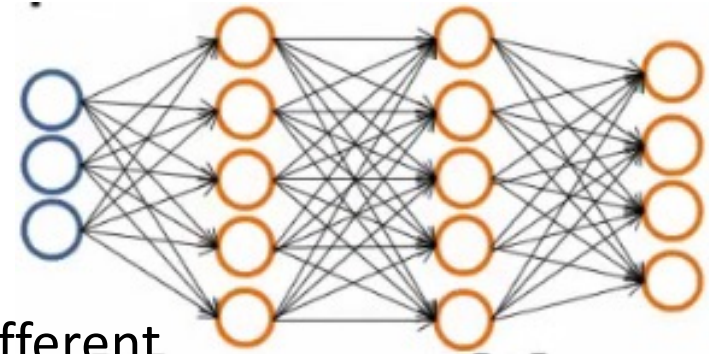
- Linear numerical and logistic regression often insufficiently complex
- Polynomial features quickly become too expensive, especially due to cross-terms between many features
- Need more efficient way to introduce high degree of non-linearity: neural networks

Limitations of ANN

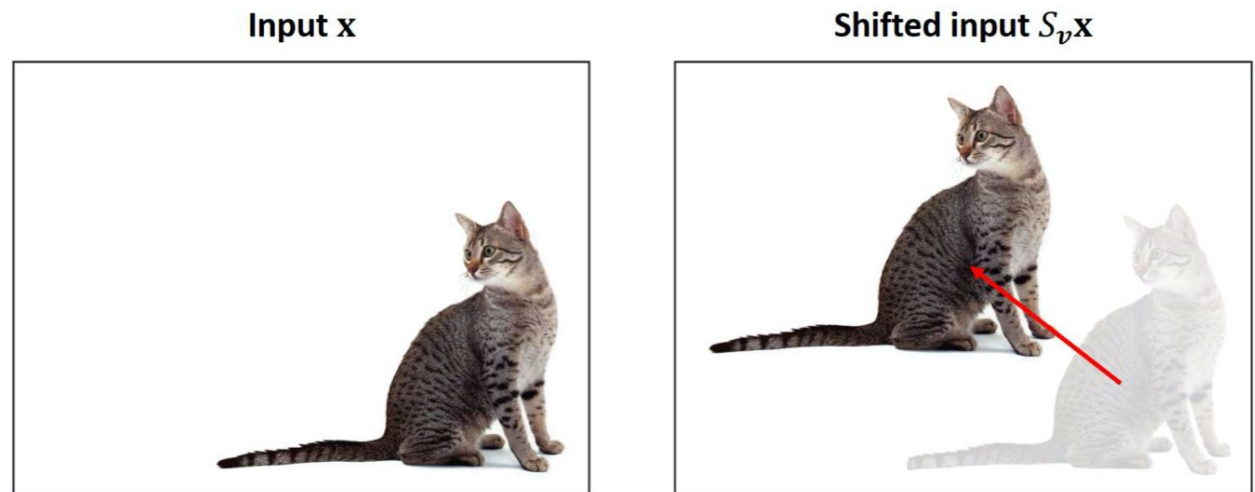


- *Too many* trainable parameters
- For tiny 40x40 pixel images, first hidden layer of 25 nodes (lab) has already $401 \times 25 = 10,025$ fitting parameters (= **weights**)
- Same ANN for typical RGB phone picture has $3 \times 12 \times 10^6$ pixel values,
- $36 \times 10^6 \times 25 \approx 10^9$ training weights for just first layer!
- Satellite images typically have 10^8 pixels & 13 spectral bands ('colors')
- ANN not feasible.

Limitations of ANN



- To an ANN, two image below are completely different,
- Every pixel is different; difficult for ANN to have same weights for each pixel to detect cat in both pictures
- We'd like a NN that is '**translation invariant**'
- Same for **size** of cat



Inspiration for Solution

- Take cue from 'old school' image processing techniques
- **Filters** in Photoshop, Gimp, and now your phone, are all based on **convolutions** of your photo with some small **kernel**.
- What does all this mean?

What is an image..?

- Image/photo is just light intensity for each image pixel, i.e. an array of numbers; often stored as integers 0 – 255 (8-bit = 1 byte)
- 1 value per pixel for grayscale, 3 values for Red-Green-Blue (RGB), many more for multi- and hyper-spectral satellite imagery: **channels/bands**

```
208 206 247 245 244 253 247 245 136 151 255 255 255 255 255 255 235 208 231 255 254 254 255 255 254 255 252 255 255 254 244 184
243 160 137 244 254 255 254 255 118 103 209 228 155 153 236 193 74 52 66 173 255 254 254 255 255 255 254 255 254 253 244 184
192 154 75 300 249 255 255 255 110 98 84 61 35 45 91 53 45 45 43 56 141 213 253 255 255 255 245 187 186 176 233
90 109 96 143 223 255 255 252 117 75 41 35 31 24 25 36 45 44 44 46 81 117 148 234 252 254 255 248 231 248 255 253
67 70 107 196 236 255 255 255 105 25 34 35 29 20 24 33 32 30 33 34 53 85 100 142 231 242 247 249 255 255 255 255
55 52 45 134 218 251 255 232 51 12 26 33 24 24 46 75 82 78 71 66 58 52 67 90 136 228 208 158 253 247 248 255
79 58 56 75 224 255 255 118 11 27 74 99 91 106 140 162 173 172 173 172 158 137 92 47 78 189 217 206 254 222 232 255
38 43 48 52 147 245 229 58 41 81 129 145 160 168 170 173 176 179 179 177 177 177 111 32 62 209 238 255 243 249 255
40 40 33 37 90 245 172 33 65 110 139 145 151 162 171 175 178 179 182 184 187 183 173 162 71 44 166 255 255 254 255
37 44 45 31 69 250 158 36 70 129 143 142 153 162 171 175 177 178 182 190 194 188 179 170 120 51 137 255 255 250 254 255
34 45 51 65 116 237 181 54 116 138 140 144 154 164 176 178 174 177 183 185 185 185 184 178 140 66 143 254 254 225 249 255
34 36 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 57 129 138 138 140 151 156 166 168 171 176 181 188 186 186 186 183 180 102 136 242 255 255 254 254
36 32 72 129 212 228 115 66 121 104 103 104 94 103 134 158 170 162 125 106 121 143 156 191 191 104 134 229 253 253 255 251
61 82 116 107 179 247 134 60 102 92 111 119 103 81 94 147 191 177 127 98 123 153 147 163 200 90 100 232 207 167 227 215
144 176 166 231 210 232 170 67 115 68 76 62 63 65 88 139 192 191 135 80 53 99 141 164 201 97 79 192 245 233 248 249
127 145 150 195 204 213 197 95 133 132 118 133 126 106 110 130 191 197 166 130 127 147 147 171 188 110 121 228 233 149 215 212
87 112 100 79 85 82 65 75 142 148 151 153 136 125 120 140 191 190 193 175 174 192 196 190 208 127 163 239 221 149 198 195
63 83 106 134 129 106 39 78 132 142 155 159 139 111 124 164 185 200 186 193 191 195 201 202 200 143 217 253 249 242 239 234
69 78 78 113 97 74 43 106 127 140 152 155 125 97 112 150 186 193 174 183 196 198 203 208 209 166 248 254 255 254 255 254
72 45 63 56 46 52 49 74 127 137 146 149 132 103 78 90 134 141 168 165 199 207 204 204 216 193 236 244 252 242 237 243
52 22 69 73 59 86 46 74 117 127 144 161 148 124 105 120 156 167 193 162 189 206 201 205 214 194 174 185 197 188 183 193
65 49 77 89 50 68 43 61 109 127 142 148 113 100 121 145 146 169 189 182 175 181 202 201 204 202 174 166 169 178 183 188 184
82 76 90 79 54 58 37 47 90 121 132 116 89 79 111 146 163 149 122 124 180 197 195 196 180 149 146 152 155 157 159 168
104 107 121 123 105 79 27 33 66 111 122 120 114 115 147 175 190 196 163 101 170 200 187 187 156 148 145 139 137 141 140 145
113 127 133 135 105 101 28 37 88 115 121 128 128 141 142 168 202 212 153 164 187 180 166 154 146 143 149 151 151 144 144
119 118 118 126 126 111 21 29 28 58 100 118 131 140 151 159 186 201 205 192 180 169 149 166 119 144 147 143 138 141 144 148
117 120 125 130 139 106 18 30 44 58 70 102 135 147 168 196 212 215 210 195 177 152 133 195 58 58 126 151 145 143 142 141
115 123 127 134 145 101 27 54 52 38 45 71 105 135 175 189 193 216 206 166 139 111 164 203 74 6 121 149 142 142 143 146
101 108 124 121 132 105 44 40 31 35 57 44 58 102 147 144 138 163 145 94 90 145 196 187 64 49 165 160 143 144 142 143
98 97 97 96 104 76 34 33 31 49 41 49 51 58 74 53 55 66 63 89 150 189 209 156 63 108 140 149 125 133 131 131
103 102 97 89 73 35 30 23 42 50 66 42 90 69 51 57 82 123 157 186 205 169 62 96 153 106 101 154 135 130 128
```



What is a kernel?

- Just a small matrix, typically square and typically of odd dimensions, e.g., 3x3, 5x5, or –less common– larger.
- Example, Left-Sobel kernel:

$$\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$

What is a Convolution?

- ‘Slide’ kernel over image, one (or more) pixel at a time horizontally, then vertically
- Multiply each element of kernel with each element in image and sum
- For (square) image width n_w and kernel size n_f , output is $w = n_w - n_f + 1$
- But why convolutions??

7	2	3	3	8
4	5	3	8	4
3	3	2	8	4
2	8	7	2	7
5	4	4	5	4

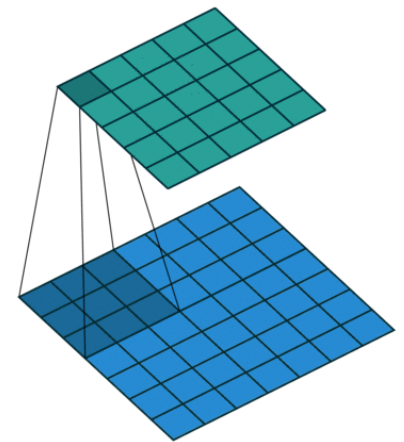
*

1	0	-1
1	0	-1
1	0	-1

=

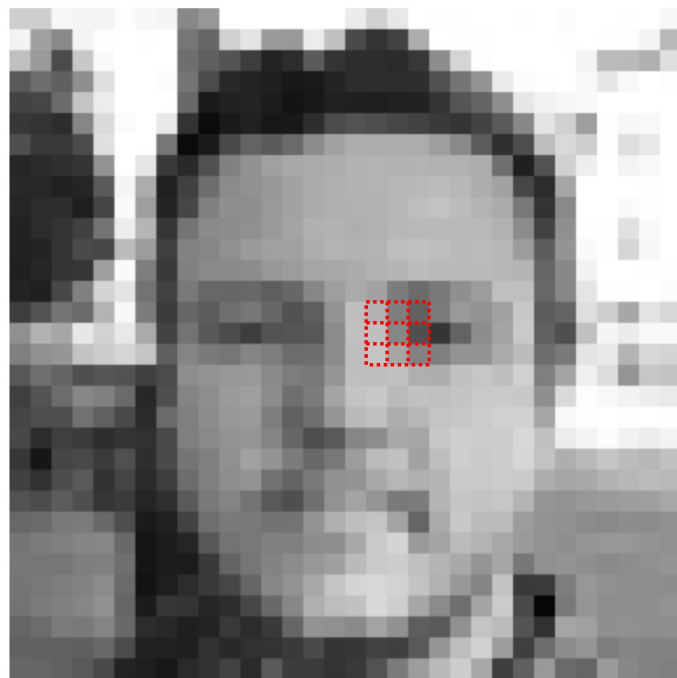
6		

$7 \times 1 + 4 \times 1 + 3 \times 1 + 2 \times 0 + 5 \times 0 + 3 \times 0 + 3 \times -1 + 3 \times -1 + 2 \times -1 = 6$



This is how image processing filters work!

- Vertical edge detection:

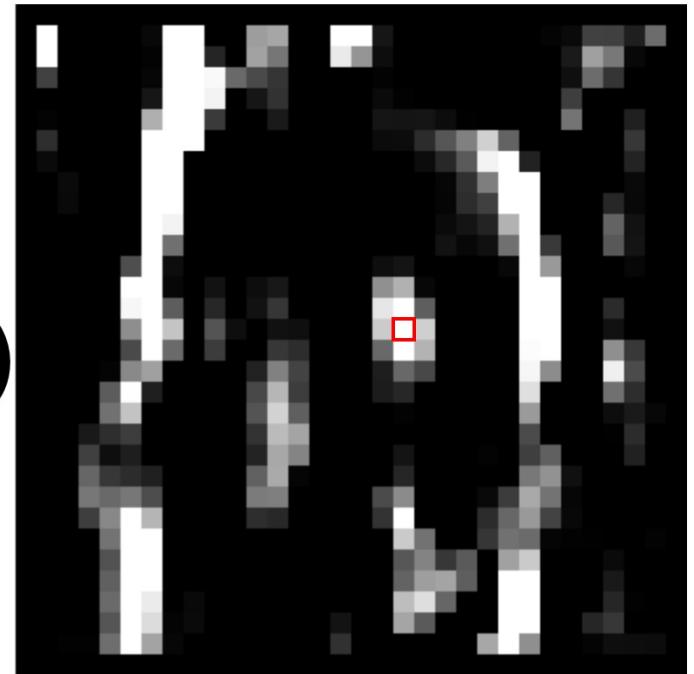


input image

$$\left(\begin{array}{ccc} 177 & + & 127 & + & 98 \\ \times 1 & & \times 0 & & \times -1 \\ + & 191 & + & 135 & + & 80 \\ \times 2 & & \times 0 & & \times -2 \\ + & 197 & + & 166 & + & 130 \\ \times 1 & & \times 0 & & \times -1 \end{array} \right)$$

$$= 368$$

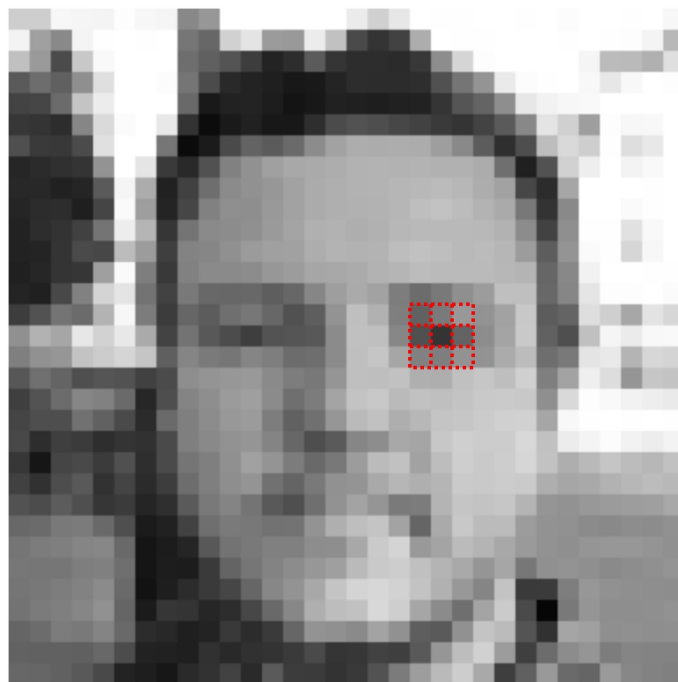
kernel:



output image

This is how image processing filters work!

- Horizontal edge detection:

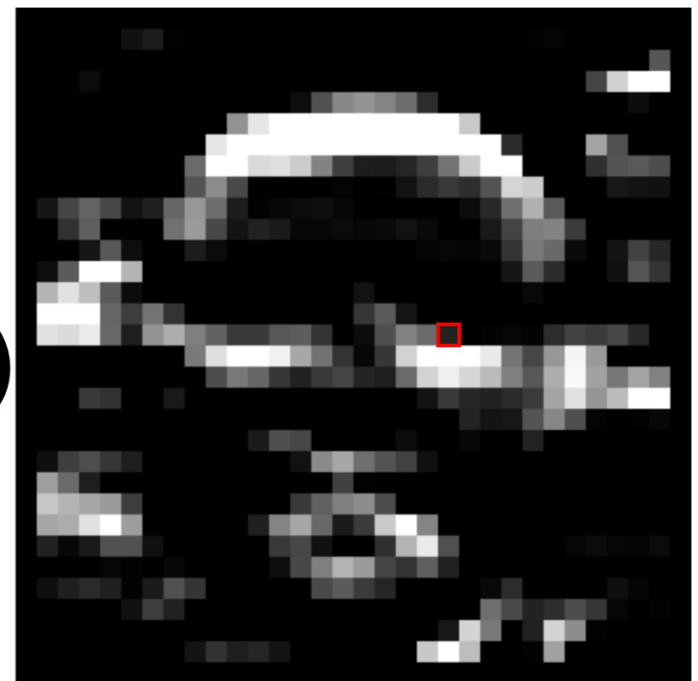


input image

$$\left(\begin{array}{ccc} 98 & + & 123 & + & 153 \\ \times -1 & & \times -2 & & \times -1 \\ + & 80 & + & 53 & + & 99 \\ \times 0 & & \times 0 & & \times 0 \\ + & 130 & + & 127 & + & 147 \\ \times 1 & & \times 2 & & \times 1 \end{array} \right)$$

= 34

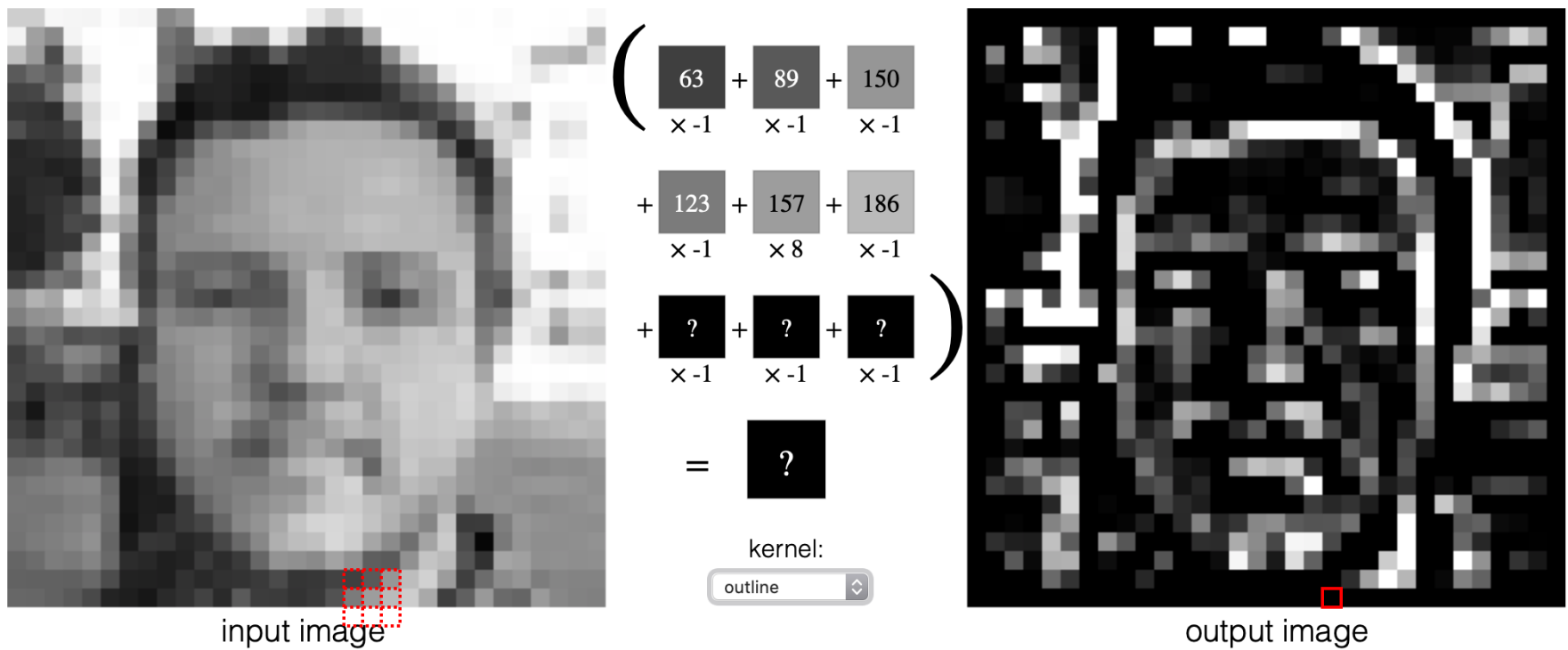
kernel:



output image

This is how image processing filters work!

- Outline:



This is how image processing filters work!

- Sharpen:



input image

$$\begin{pmatrix} ? & + & 104 & + & 107 \\ \times 0 & & \times -1 & & \times 0 \\ + & & & & \\ ? & + & 117 & + & 124 \\ \times -1 & & \times 5 & & \times -1 \\ + & & & & \\ ? & + & 119 & + & 118 \\ \times 0 & & \times -1 & & \times 0 \end{pmatrix}$$

=

?

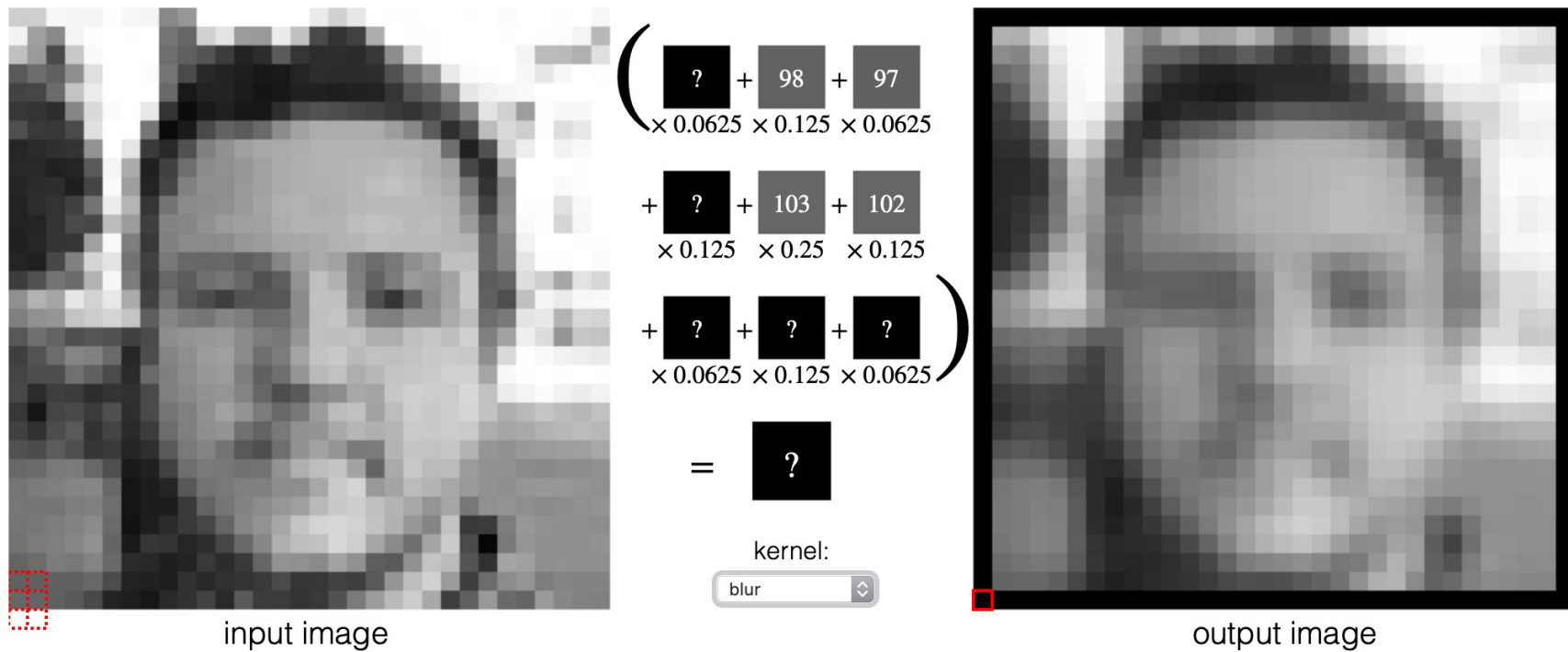
kernel: sharpen



output image

This is how image processing filters work!

- Blur:



This is how image processing filters work!

- Design your own:

Choose File kop_lipari_2021.jpg

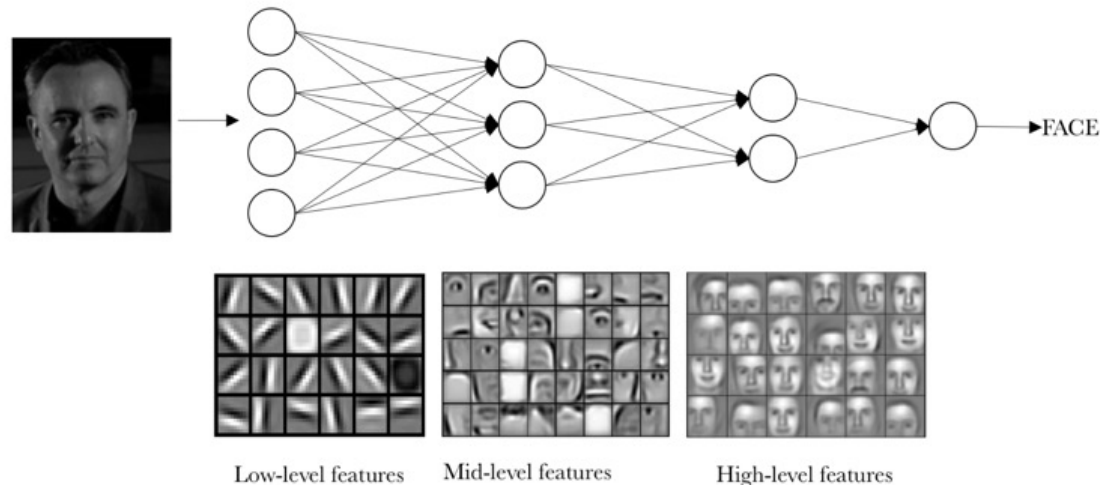
-2	-1	0
-1	1	1
0	1	2

emboss



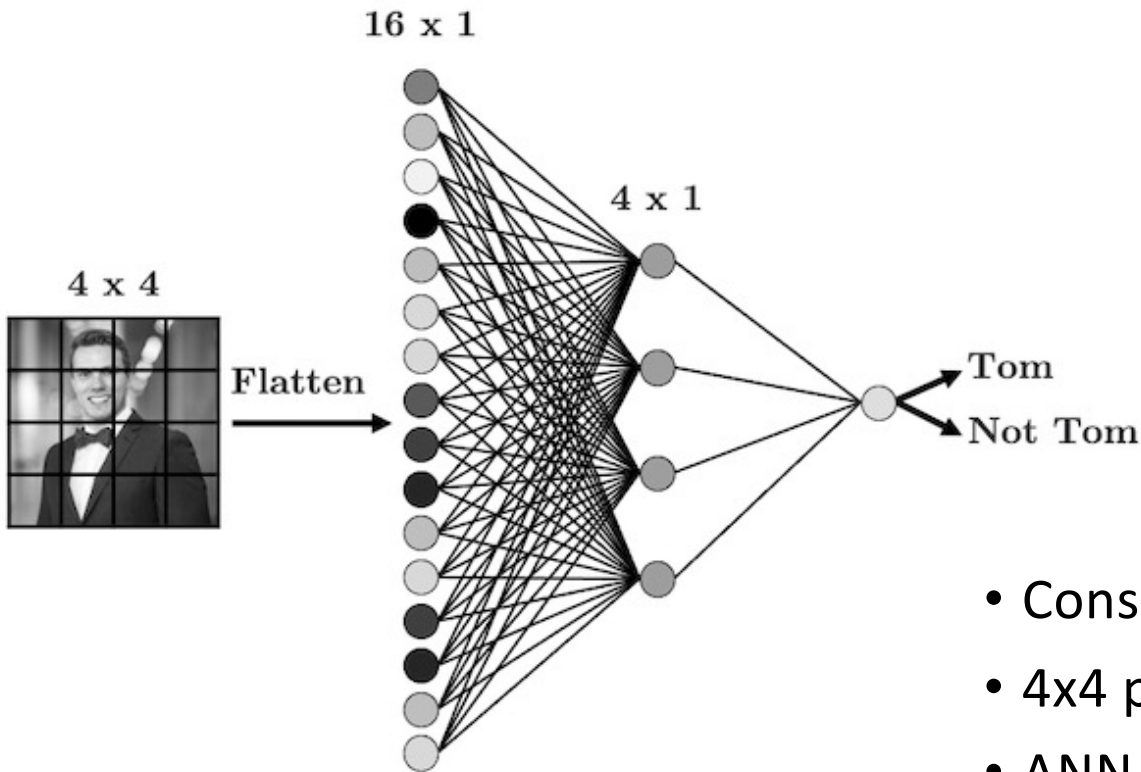
Image processing -> Convolutional NN

- **Observation:** tiny filter/kernel of, e.g., $3 \times 3 = 9$ parameters is able to extract important features from images of any size
- **Idea:** can we combine a whole bunch of such filters to extract enough features to identify what's in a picture (automatically, by a NN)?



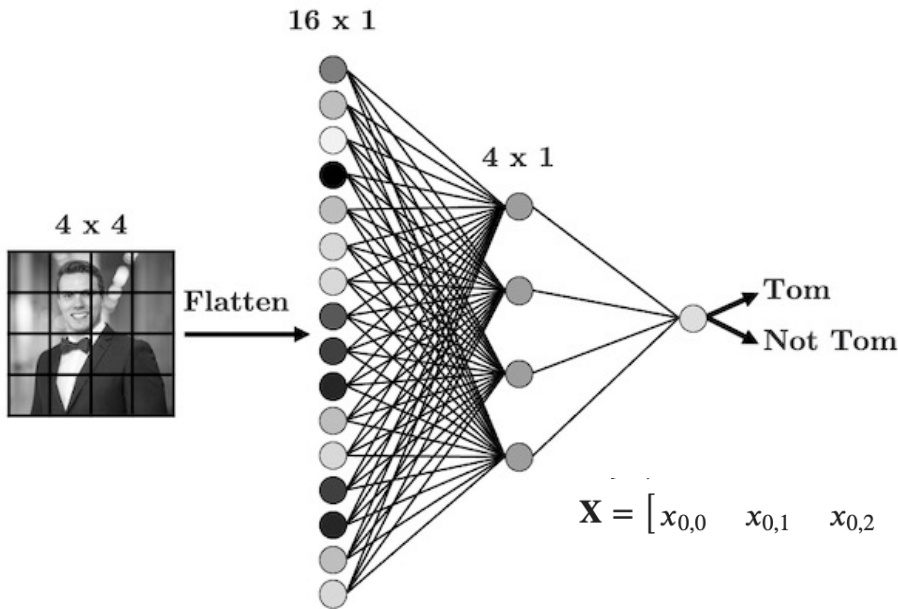
Equivalency between ANN & CNN

$$\text{Total Parameters} = (16 \times 4 + 4) + (4 \times 1 + 1) = 73$$



- Consider simplest possible example
- 4x4 pixel grayscale image
- ANN with just 1 hidden layer, 4 nodes
- Binary output
- **73 trainable parameters/weights**

$$\text{Total Parameters} = (16 \times 4 + 4) + (4 \times 1 + 1) = 73$$



$$\mathbf{X} = [x_{0,0} \ x_{0,1} \ x_{0,2} \ x_{0,3} \ x_{1,0} \ x_{1,1} \ x_{1,2} \ x_{1,3} \ x_{2,0} \ x_{2,1} \ x_{2,2} \ x_{2,3} \ x_{3,0} \ x_{3,1} \ x_{3,2} \ x_{3,3}]^T$$

- Remember process/steps of **ANN**
- ‘Unroll’ 2D image to 1D feature vector

$$\mathbf{X}' = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix} \in \mathbb{R}^{4 \times 4}$$

- to:

- Next layer, again linear combo:

$$\mathbf{z}^{[3]} = \Theta^{[2]} \cdot \mathbf{a}^{[2]}$$

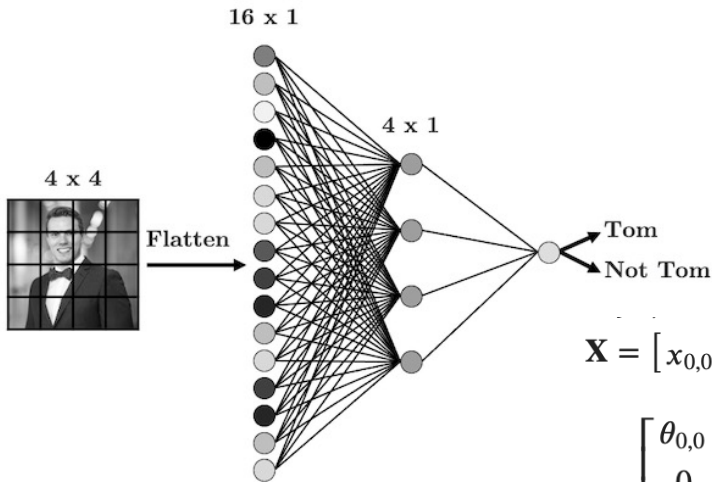
- Final (output) layer, non-linear activation function: $y = g(\mathbf{z}^{[3]})$

- Add bias and multiply with 17 x 4 matrix of weights for all connections

- **Linear combination:** $\mathbf{z}^{[2]} = \Theta^{[1]} \cdot \mathbf{X}^{[1]}$

- **Non-linear activation:** $\mathbf{a}^{[2]} = g(\mathbf{z}^{[2]})$.

Total Parameters = $(16 \times 4 + 4) + (4 \times 1 + 1) = 73$

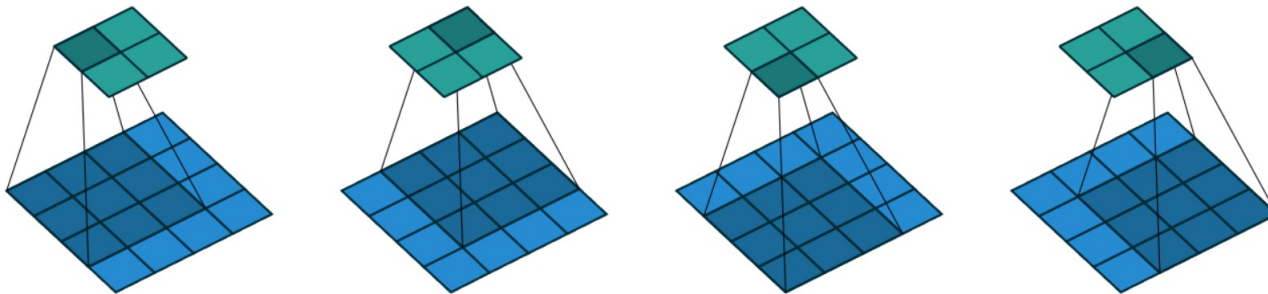


$$\mathbf{X} = [x_{0,0} \quad x_{0,1} \quad x_{0,2} \quad x_{0,3} \quad x_{1,0} \quad x_{1,1} \quad x_{1,2} \quad x_{1,3} \quad x_{2,0} \quad x_{2,1} \quad x_{2,2} \quad x_{2,3} \quad x_{3,0} \quad x_{3,1} \quad x_{3,2} \quad x_{3,3}]^T$$

$$\Theta^{\text{ANN}} = \begin{bmatrix} \theta_{0,0} & \theta_{0,1} & \theta_{0,2} & 0 & \theta_{1,0} & \theta_{1,1} & \theta_{1,2} & 0 & \theta_{2,0} & \theta_{2,1} & \theta_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \theta_{0,0} & \theta_{0,1} & \theta_{0,2} & 0 & \theta_{1,0} & \theta_{1,1} & \theta_{1,2} & 0 & \theta_{2,0} & \theta_{2,1} & \theta_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \theta_{0,0} & \theta_{0,1} & \theta_{0,2} & 0 & \theta_{1,0} & \theta_{1,1} & \theta_{1,2} & 0 & \theta_{2,0} & \theta_{2,1} & \theta_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \theta_{0,0} & \theta_{0,1} & \theta_{0,2} & 0 & \theta_{1,0} & \theta_{1,1} & \theta_{1,2} & 0 & \theta_{2,0} & \theta_{2,1} & \theta_{2,2} \end{bmatrix}$$

- **CNN**
- 3x3 kernel: $\Theta^{\text{CNN}} = \begin{bmatrix} \theta_{0,0} & \theta_{0,1} & \theta_{0,2} \\ \theta_{1,0} & \theta_{1,1} & \theta_{1,2} \\ \theta_{2,0} & \theta_{2,1} & \theta_{2,2} \end{bmatrix} \in \mathbb{R}^{3 \times 3}$
- Can be re-written as 16 x 4 matrix (special kind, circulant/Toeplitz)

- Everything else the same, *but* above matrix only has 9 unique values (vs. 73)
- $4 \times 9 = 36$ connections instead of 73; other steps basically the same.



Two (or more) filters in CNN

- Typically, hidden layer has multiple **channels** for different filters
- Example: two 3x3 kernels (say, horizontal and vertical edges)
- For our 4x4 input image, hidden layer has 8 nodes but **stacked** as 2 x 4 nodes
- Algebraically, we can vertically stack 2 similar Toeplitz matrices into a 8x16 matrix
- Or use 2 separate 4x16 Toeplitz matrices (more efficient)

- In real applications: many automatically determined filters

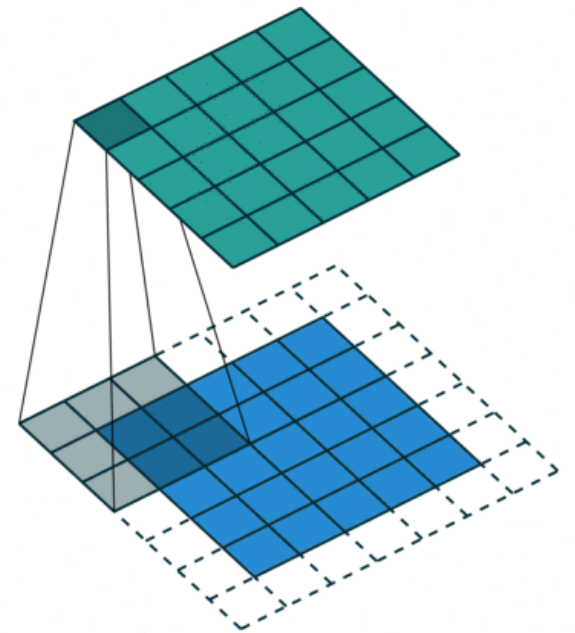
CNN for image classification

- After one or more convolutional layers, output of channels are **flattened**
- One or more fully connected layers combine features in different parts of image
- Combination (with trainable weights) gives final image class.

Technical Details

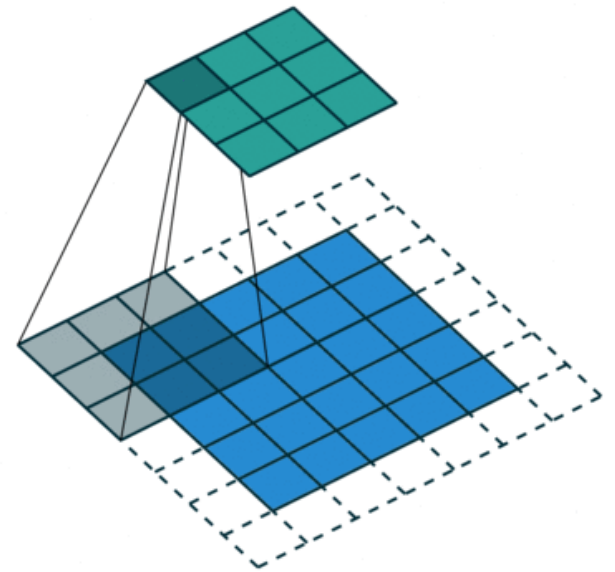
Padding

- Sometimes, we may want output to be same size as input
- ‘Pad’ image with one or more ‘dummy’ pixels, typically zero-valued
- This also applies filter ‘better’ near edges
- Sometimes we like to have, say, even pixel dimensions
- Padding can be
 - **valid** (no padding),
 - **same** (giving same size of output image), or
 - **full** (making sure every pixel is covered n_p^2 times)



Stride

- Instead of moving kernel 1 pixel at a time, we can move 2 or more pixels (n_s) at a time; this is called the **stride**
- This reduces output size roughly by a factor n_s in each dimension (e.g. n_s^2 in total): **compression**

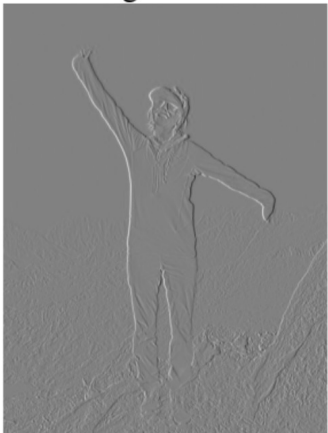


Max-Pooling (and Average Pooling)

- Special type of convolution that takes maximum (or average) value inside kernel window (no adjustable weights!)
- Usually, stride is same as max-pool size such that there is no overlap
- Pooling again reduces data (compression), while enhancing features

Size of input image: (800, 600) and after max pooling (399, 299) and average pooling (399, 299)

Original size



After max pooling



Input

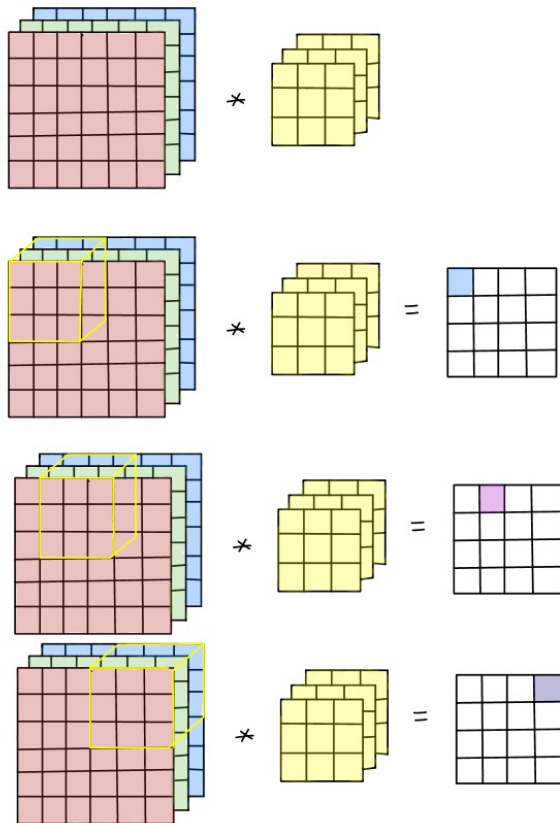
7	3	5	2
8	7	1	6
4	9	3	9
0	8	4	5

maxpool

Output

8	6
9	9

Volume convolutions



- When input has n_c channels, such as RGB ($n_c=3$), filter/kernel is $n_f \times n_f \times n_c$ and also summed over n_c

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

↓
308

+

↓
-498

+

↓
164 + 1 = -25

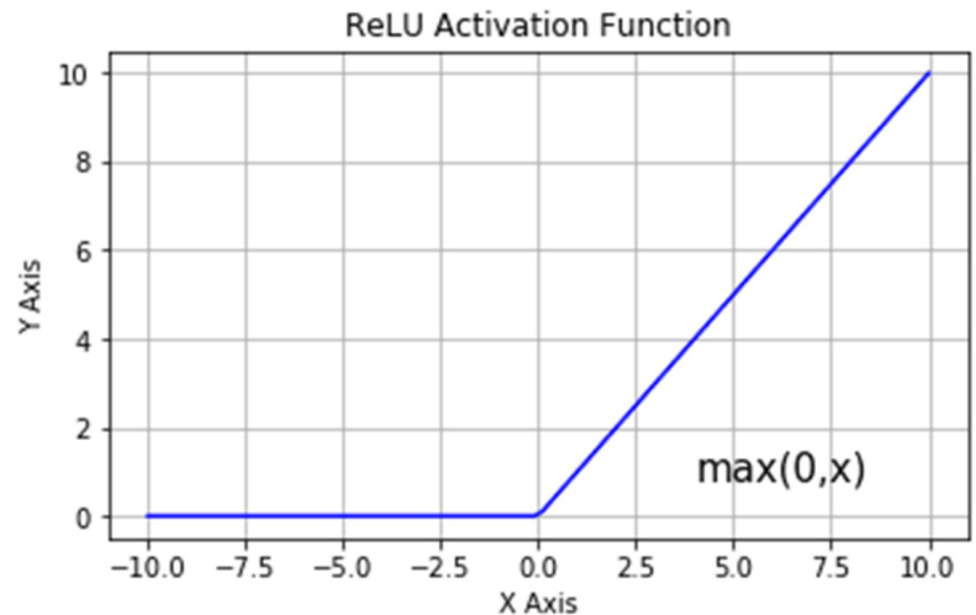
↑
Bias = 1

- Example with padding and bias

-25			...
			...
			...
			...
...

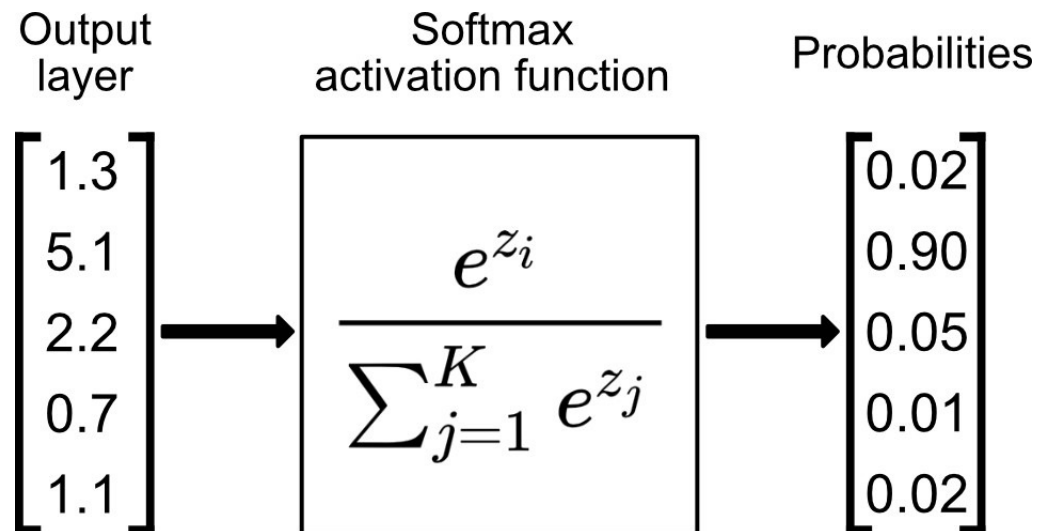
Activation function

- Rectified Linear Units (ReLU) more common than sigmoids in CNN
- For ReLU $g(z)$, with $z = x \cdot \Theta$ and $z < 0$, $g(z)$ is exactly zero. Zero elements in weights matrix: sparse.
- For $z > 0$, $\frac{\partial g(z)}{\partial z}$ is constant
- Sigmoid has *vanishing gradients*
- Gradient descent more efficient



Softmax

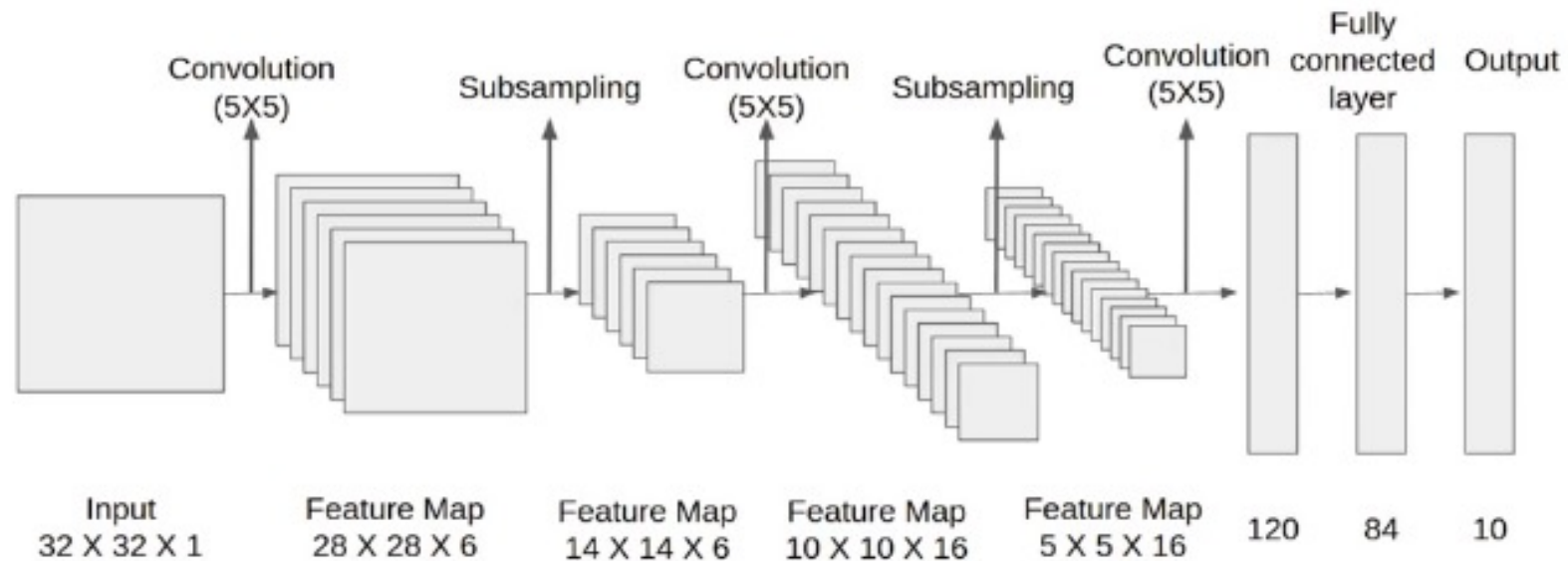
- For multi-class CNN, we use **softmax** instead of sigmoid activation function for the output layer
- Instead of binary 0/1, outputs **probabilities** for any nr of classes



Putting it all Together

LeNet-5

- One of the first CNN (1998, much more limited CPU power)



Python code

- Nowadays, unbelievably easy, thanks to, e.g, Keras:

```
1 model = keras.Sequential()
2
3 model.add(layers.Conv2D(filters=6, kernel_size=(5, 5), activation='relu', input_shape=(32,32,1)))
4 model.add(layers.AveragePooling2D())
5
6 model.add(layers.Conv2D(filters=16, kernel_size=(5, 5), activation='relu'))
7 model.add(layers.AveragePooling2D())
8
9 model.add(layers.Flatten())
10
11 model.add(layers.Dense(units=120, activation='relu'))
12
13 model.add(layers.Dense(units=84, activation='relu'))
14
15 model.add(layers.Dense(units=10, activation = 'softmax'))
```

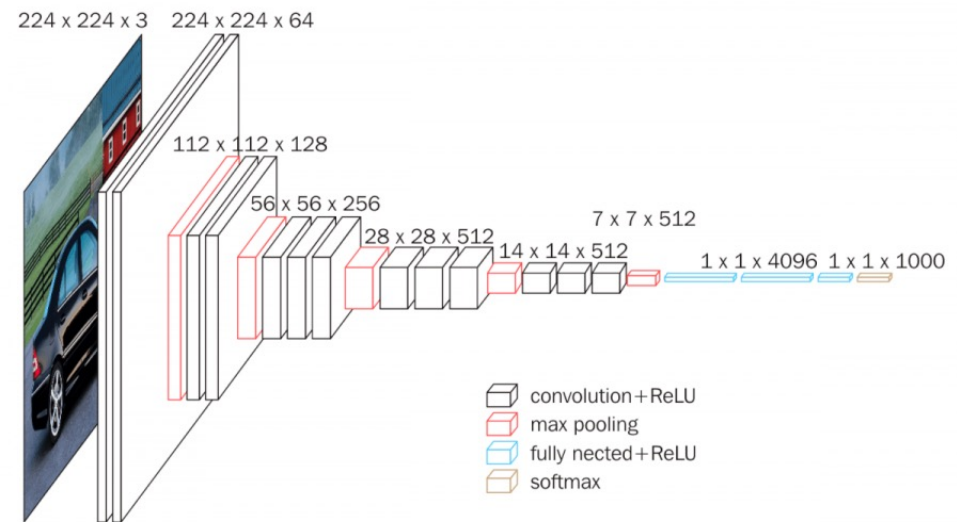
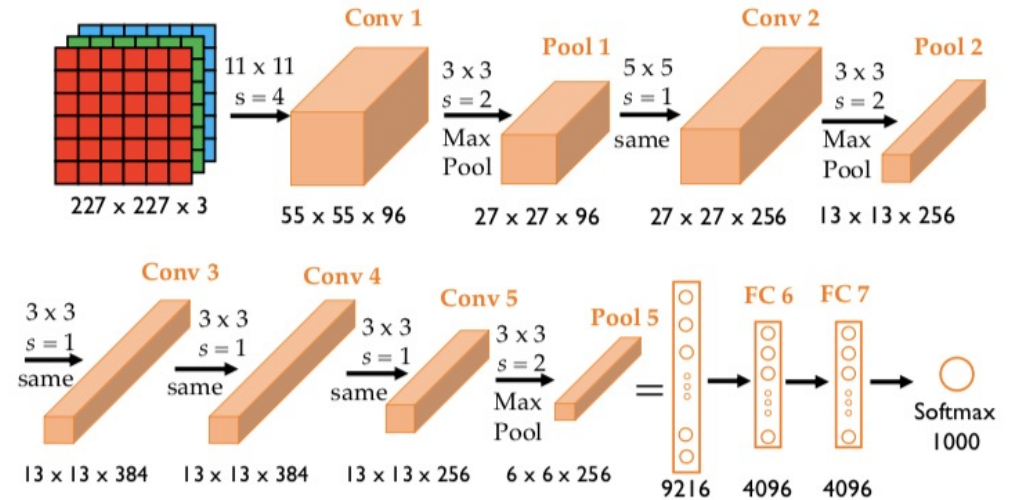
Python code

- Can provide nice summary of, e.g., trainable parameters:

```
1 model.summary()
Model: "sequential_17"
Layer (type)                Output Shape                Param #
-----
conv2d (Conv2D)             (None, 28, 28, 6)          156
average_pooling2d_5 (Averag (None, 14, 14, 6)          0
ePooling2D)
conv2d_1 (Conv2D)           (None, 10, 10, 16)         2416
average_pooling2d_6 (Averag (None, 5, 5, 16)           0
ePooling2D)
flatten (Flatten)           (None, 400)                 0
dense (Dense)                (None, 120)                 48120
dense_1 (Dense)              (None, 84)                 10164
dense_2 (Dense)              (None, 10)                 850
-----
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
```

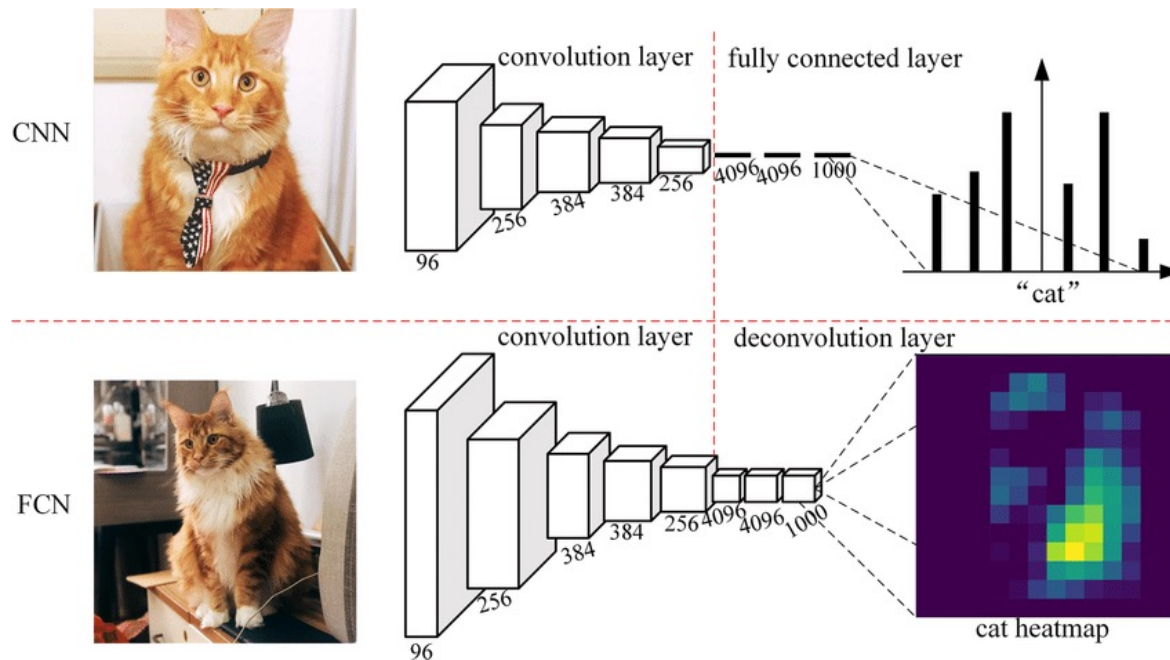
AlexNet & VGG-16

- Other 'classic', deeper CNN
- AlexNet: 62 million weights
- VGG-16: 138 million



Fully Convolutional Neural Networks (FCN)

- No fully-connected layers anywhere in NN
- Allows for any input size



U-Nets for Segmentation

- Auto-encoder -> bottleneck -> decoder

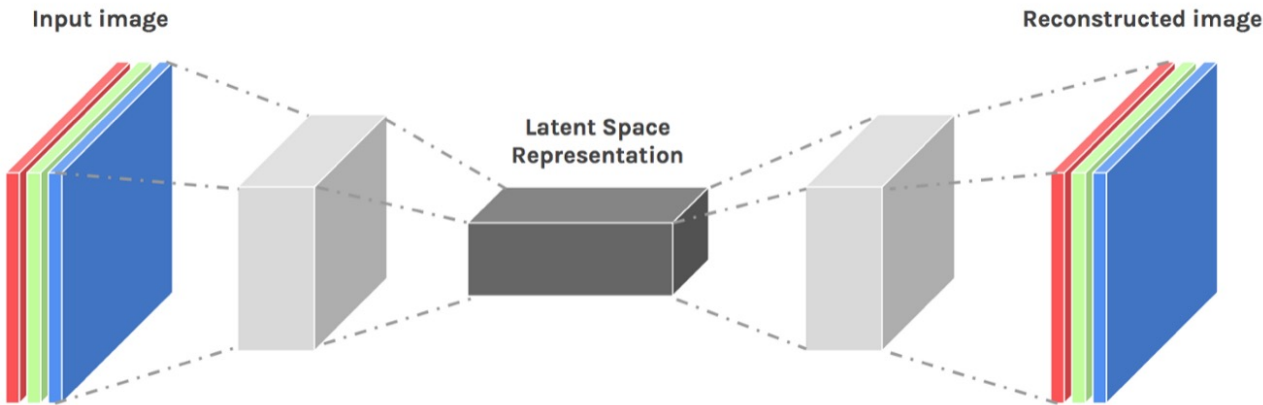
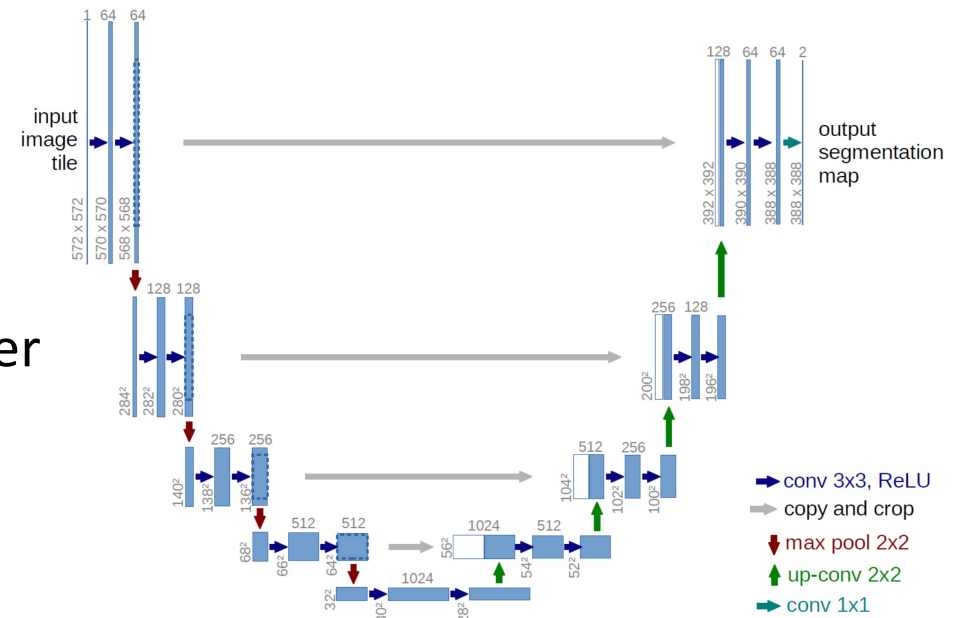


Figure 1: Auto-encoding an RGB image with two Conv2D followed by two Conv2DTranspose. (source)

Different Computer Vision Tasks

Classification



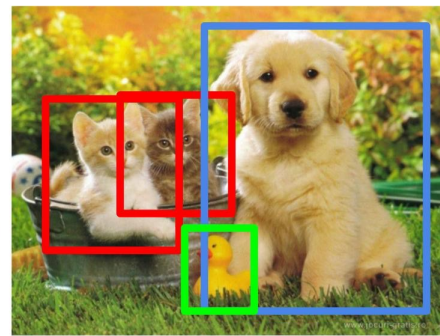
CAT

**Classification
+ Localization**



CAT

Object Detection



CAT, DOG, DUCK

**Semantic or
Instance
Segmentation**



CAT, DOG, DUCK

Single object

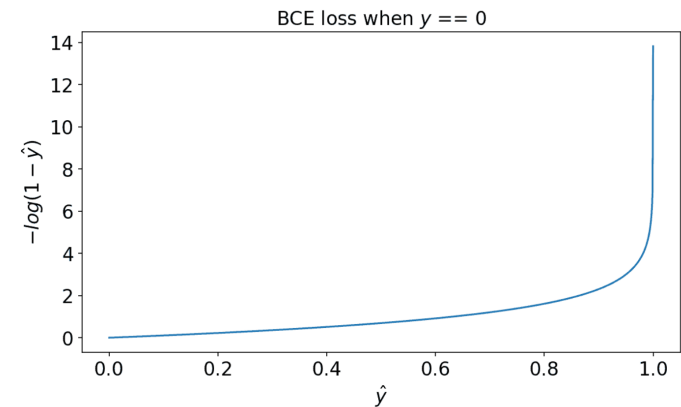
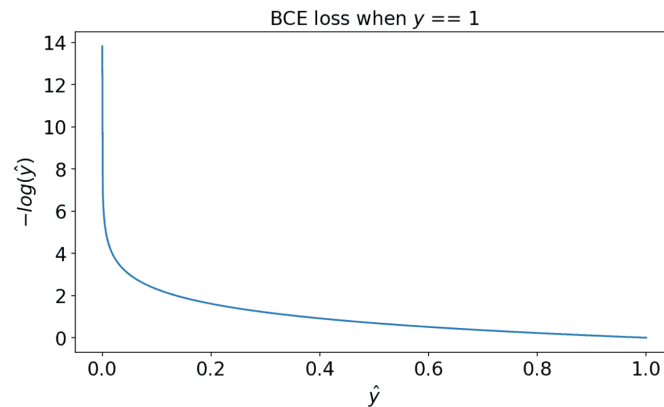
Multiple objects

'Accuracy' Metrics for Segmentation

- Binary cross entropy (BCE) loss = logistic loss function

$$L_{BCE} = -\frac{1}{n} \sum_{i=1}^n (Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log (1 - \hat{Y}_i))$$

- Common choice, e.g., to minimize by gradient descent
- Not that intuitive



'Accuracy' Metrics for Segmentation

Sources: [6][7][8][9][10][11][12][13][14] view · talk · edit

		Predicted condition			
		Positive (PP)	Negative (PN)		
Total population = P + N				Informedness, bookmaker informedness (BM) = TPR + TNR - 1	Prevalence threshold (PT) $= \frac{\sqrt{\text{TPR} \times \text{FPR}} - \text{FPR}}{\text{TPR} - \text{FPR}}$
Actual condition	Positive (P)	True positive (TP), hit	False negative (FN), type II error, miss, underestimation	True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power $= \frac{\text{TP}}{\text{P}} = 1 - \text{FNR}$	False negative rate (FNR), miss rate $= \frac{\text{FN}}{\text{P}} = 1 - \text{TPR}$
	Negative (N)	False positive (FP), type I error, false alarm, overestimation	True negative (TN), correct rejection	False positive rate (FPR), probability of false alarm, fall-out $= \frac{\text{FP}}{\text{N}} = 1 - \text{TNR}$	True negative rate (TNR), specificity (SPC), selectivity $= \frac{\text{TN}}{\text{N}} = 1 - \text{FPR}$
Prevalence $= \frac{\text{P}}{\text{P} + \text{N}}$		Positive predictive value (PPV), precision $= \frac{\text{TP}}{\text{PP}} = 1 - \text{FDR}$	False omission rate (FOR) $= \frac{\text{FN}}{\text{PN}} = 1 - \text{NPV}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Negative likelihood ratio (LR-) $= \frac{\text{FNR}}{\text{TNR}}$
Accuracy (ACC) $= \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}}$		False discovery rate (FDR) $= \frac{\text{FP}}{\text{PP}} = 1 - \text{PPV}$	Negative predictive value (NPV) = $\frac{\text{TN}}{\text{PN}} = 1 - \text{FOR}$	Markedness (MK), deltaP (Δp) = PPV + NPV - 1	Diagnostic odds ratio (DOR) = $\frac{\text{LR}+}{\text{LR}-}$
Balanced accuracy (BA) = $\frac{\text{TPR} + \text{TNR}}{2}$		F ₁ score $= \frac{2\text{PPV} \times \text{TPR}}{\text{PPV} + \text{TPR}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}$	Fowlkes–Mallows index (FM) $= \sqrt{\text{PPV} \times \text{TPR}}$	Matthews correlation coefficient (MCC) $= \sqrt{\text{TPR} \times \text{TNR} \times \text{PPV} \times \text{NPV}} - \sqrt{\text{FNR} \times \text{FPR} \times \text{FOR} \times \text{FDR}}$	Threat score (TS), critical success index (CSI), Jaccard index = $\frac{\text{TP}}{\text{TP} + \text{FN} + \text{FP}}$

'Accuracy' Metrics for Segmentation

- **Confusion matrix**

		Predicted condition	
		Positive (PP)	Negative (PN)
Actual condition	Total population = P + N		
	Positive (P)	True positive (TP), hit	False negative (FN), type II error, miss, underestimation
	Negative (N)	False positive (FP), type I error, false alarm, overestimation	True negative (TN), correct rejection

'Accuracy' Metrics for Segmentation

- **Precision:** $\frac{TP}{TP+FP}$
- What fraction of pixels **predicted** as 1 is actually **labeled** as 1.

		Predicted condition	
		Positive (PP)	Negative (PN)
Actual condition	Total population = P + N		
	Positive (P)	True positive (TP), hit	False negative (FN), type II error, miss, underestimation
	Negative (N)	False positive (FP), type I error, false alarm, overestimation	True negative (TN), correct rejection

'Accuracy' Metrics for Segmentation

- **Recall:** $\frac{TP}{TP+FN}$
- What fraction of pixels **labeled** as 1 is actually **predicted** as 1.

		Predicted condition	
		Positive (PP)	Negative (PN)
Actual condition	Total population = P + N		
	Positive (P)	True positive (TP), hit	False negative (FN), type II error, miss, underestimation
	Negative (N)	False positive (FP), type I error, false alarm, overestimation	True negative (TN), correct rejection

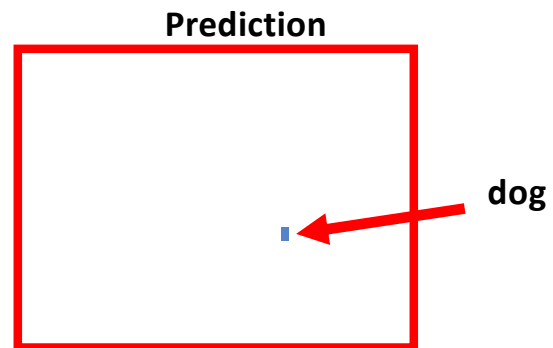
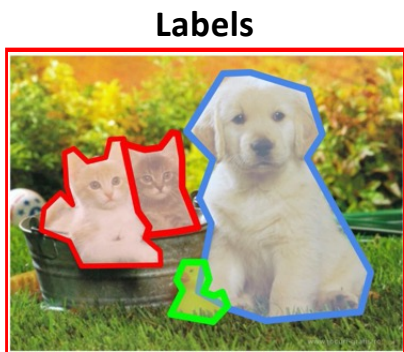
'Accuracy' Metrics for Segmentation

- **F1**: $\frac{2 TP}{2 TP + FN + FP} = \frac{2 \text{ precision} * \text{recall}}{\text{precision} + \text{recall}}$
- Harmonic average of precision and recall

		Predicted condition	
		Positive (PP)	Negative (PN)
Actual condition	Total population = P + N		
	Positive (P)	True positive (TP), hit	False negative (FN), type II error, miss, underestimation
	Negative (N)	False positive (FP), type I error, false alarm, overestimation	True negative (TN), correct rejection

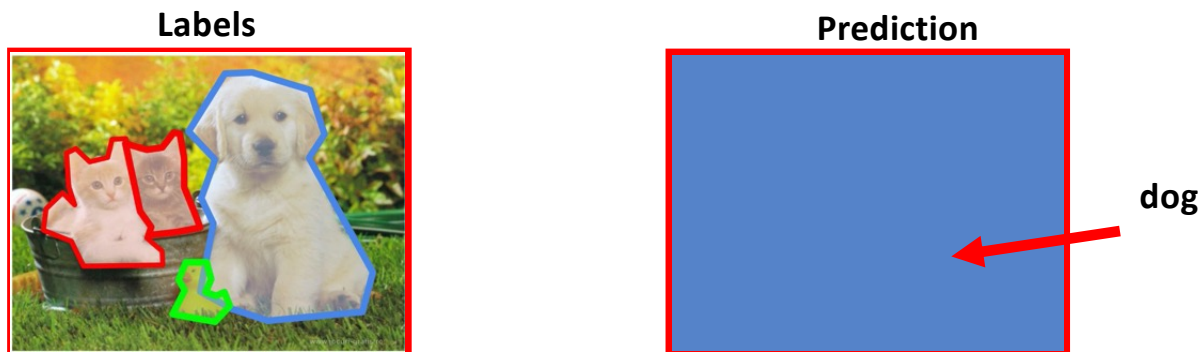
Questions:

- What is precision, recall, and F1 score of this prediction:



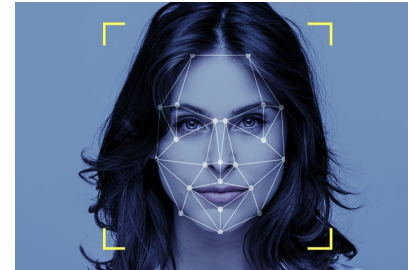
Questions:

- What is precision, recall, and F1 score of this prediction:



- This is why F1 score is useful single metric of accuracy
- Possible to directly optimize for F1: **Dice loss** = $1 - F1$

Conclusions



- CNN are more efficient than ANN for computer vision tasks
- Inspired by image/signal processing: extract features by small kernels
- Let CNN model learn its *own* kernels/filters and keep combining those
- Same parameters used in different parts of images: translation invariant
- CNN learns complex features that we, humans, cannot even interpret...
- CNN are (my) best example of true, almost magical, deep learning

Next weeks

- AI in practice / research
- Short black-board lectures on other ML/DL methods
- Work on individual AI projects