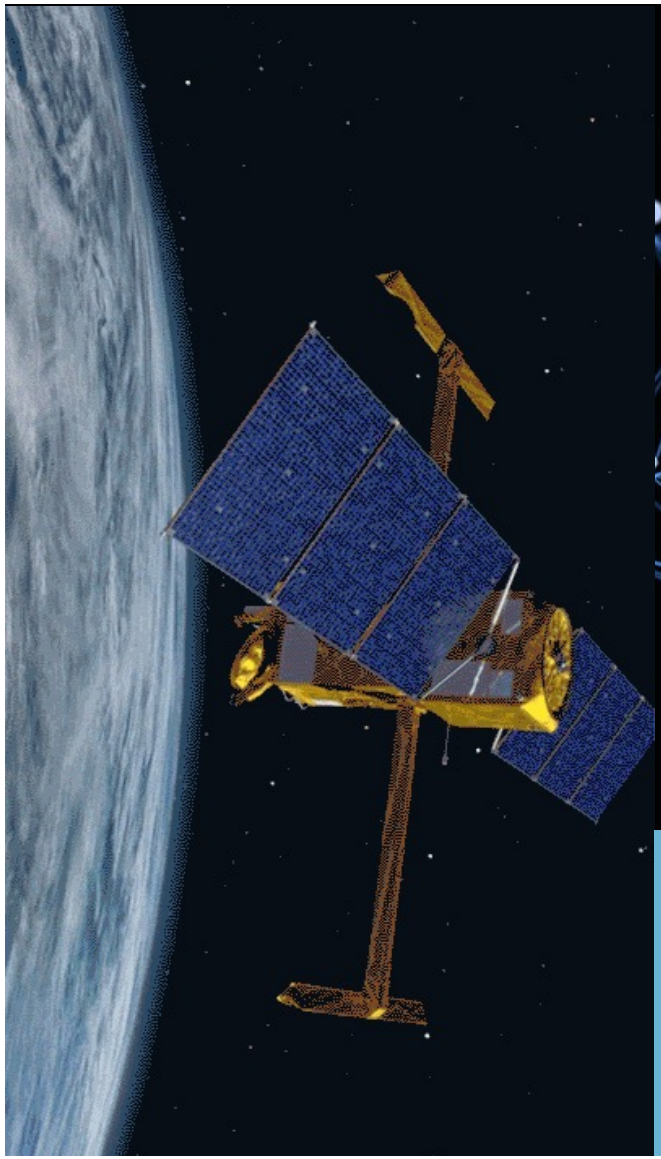


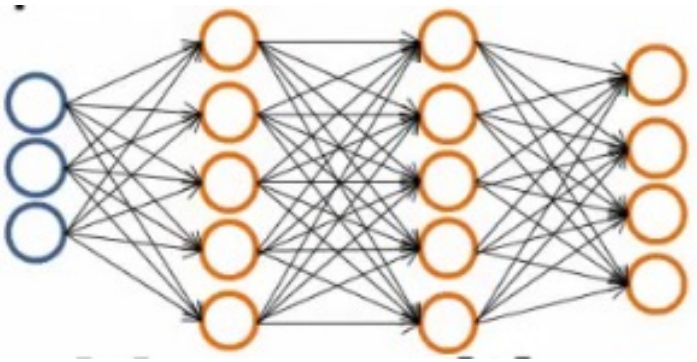
Lecture 10
Convolutional Neural Networks
For Computer Vision

Why yet another algorithm, *again*?

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

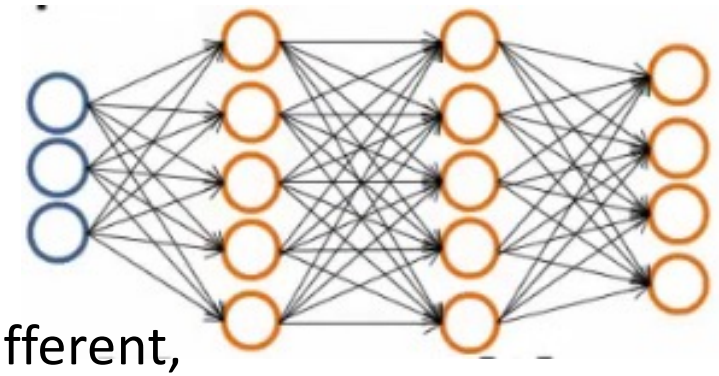
- Linear numerical and logistic regression often insufficiently complex
- Polynomial features quickly become too expensive, especially due to cross-terms between many features
- Need more efficient way to introduce high degree of non-linearity: neural networks

Limitations of ANN

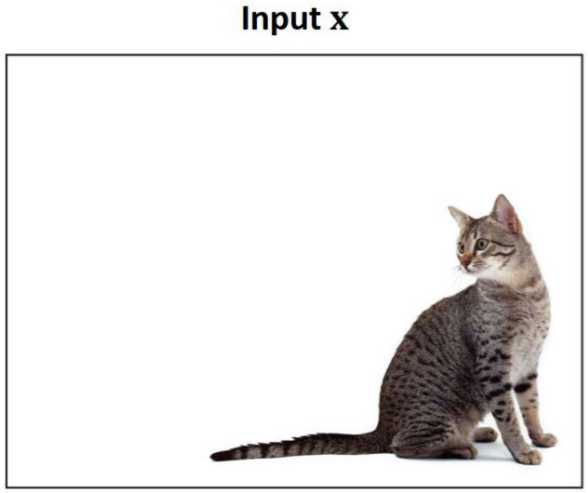


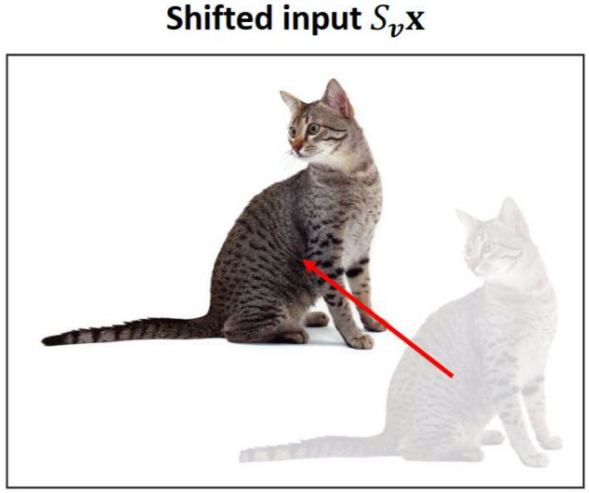
- *Too many* trainable parameters
- For tiny 40x40 pixel images, first hidden layer of 25 nodes (lab) has already 401 x 25 = 10,025 fitting parameters (= **weights**)
- Same ANN for typical RGB phone picture has 3 x 12 x 10^6 pixel values,
- 36 x 10^6 x 25 $\approx$ 10^9 training weights for just first 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



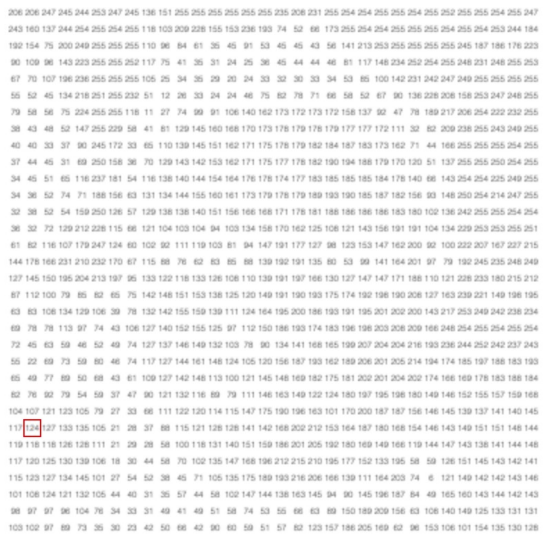


Inspiration for Solution

- 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**



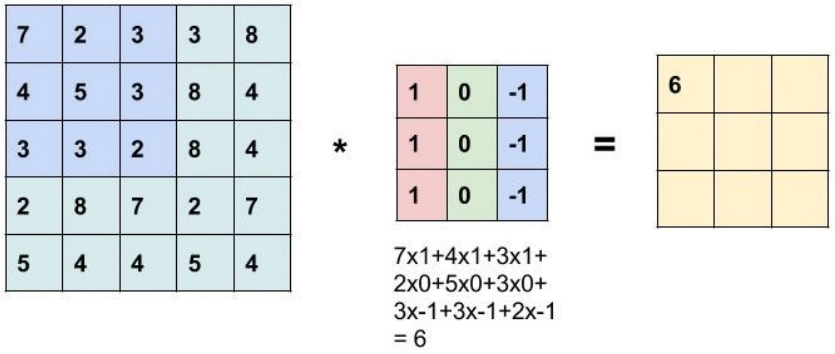
What is a kernel?

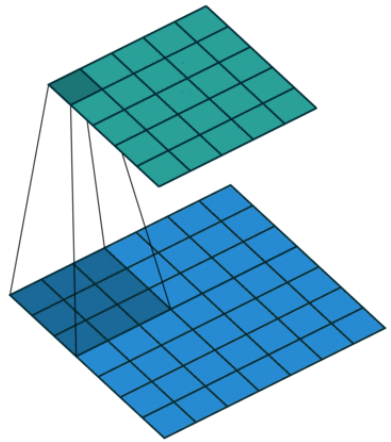
- Just a small matrix, typically square and typically of odd dimensions, e.g., 3x3, 5x5, or –less common– larger.
- Example, Left-Sobel kernel:

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

What is a Convolution?

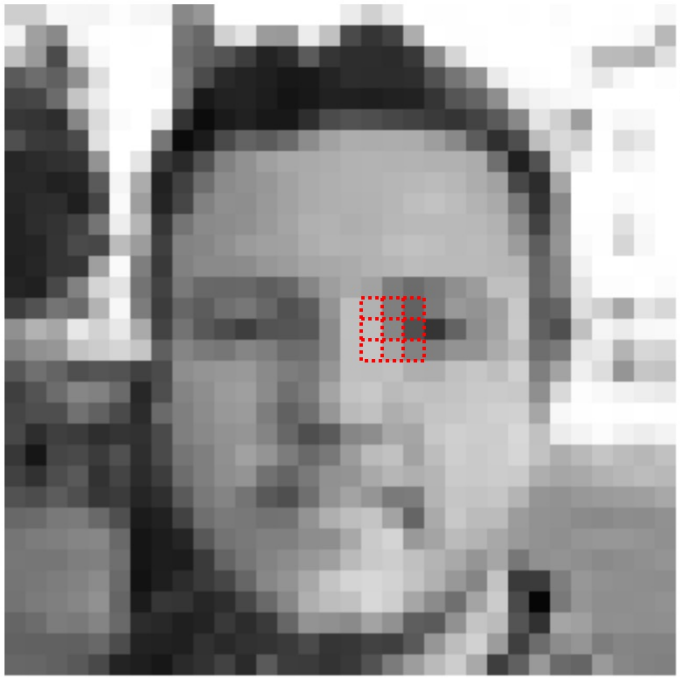
- ‘Slide’ kernel over image, one (or more) pixel at a time horizontally, then vertically
- Multiply each element of kernel with each element in image and sum
- For (square) image width n_w and kernel size n_f, output is $w = n_w - n_f + 1$
- But why convolutions??



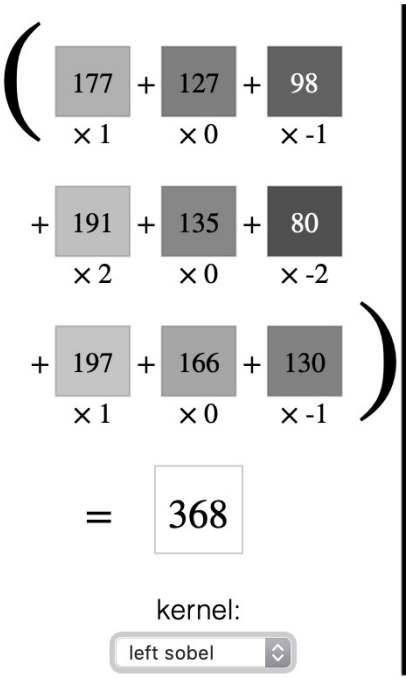


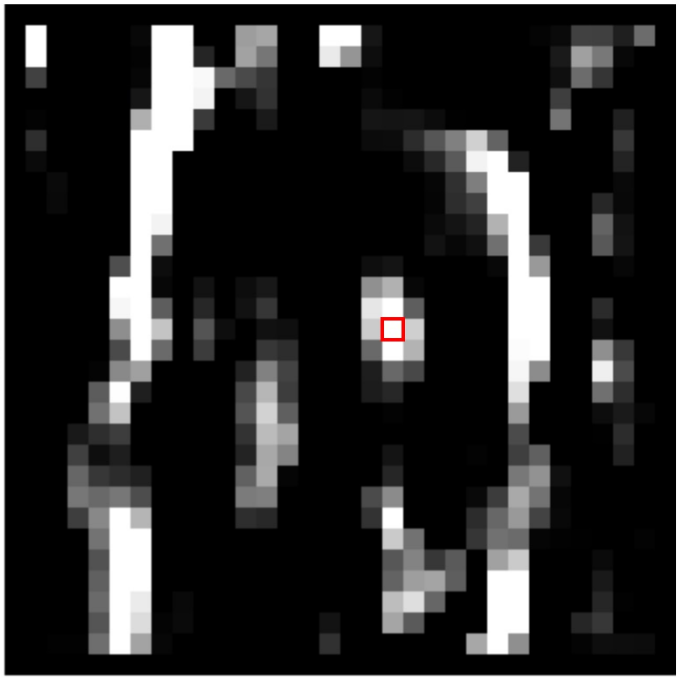
This is how image processing **filters** work!

- Vertical edge detection:



input image

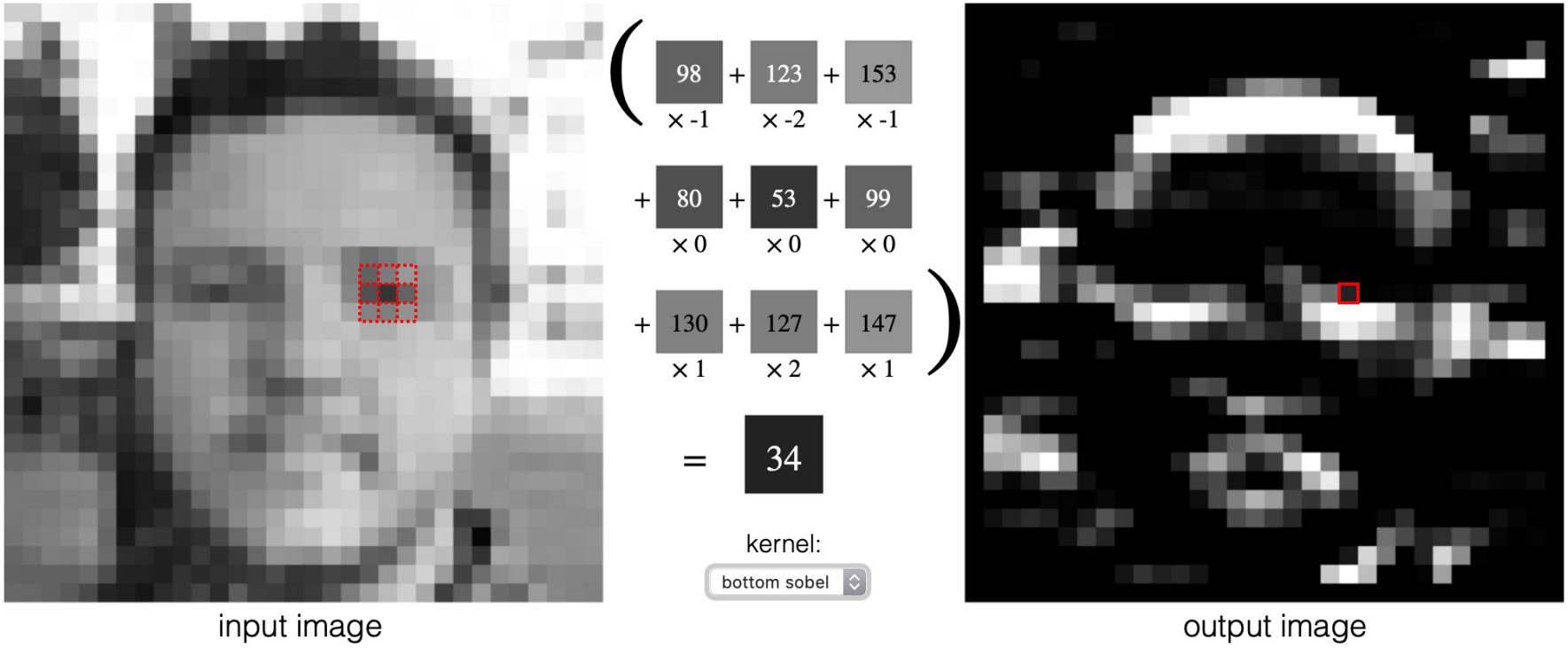




output image

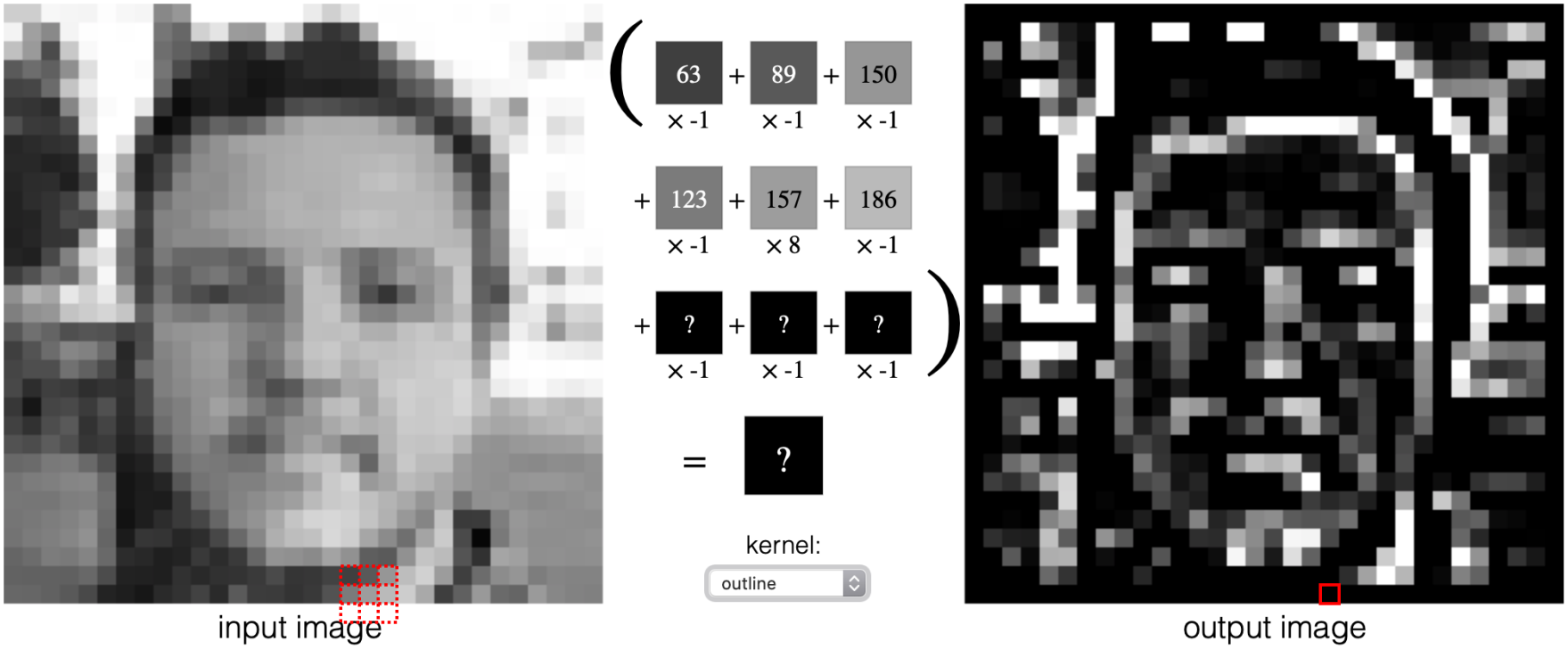
This is how image processing **filters** work!

- Horizontal edge detection:



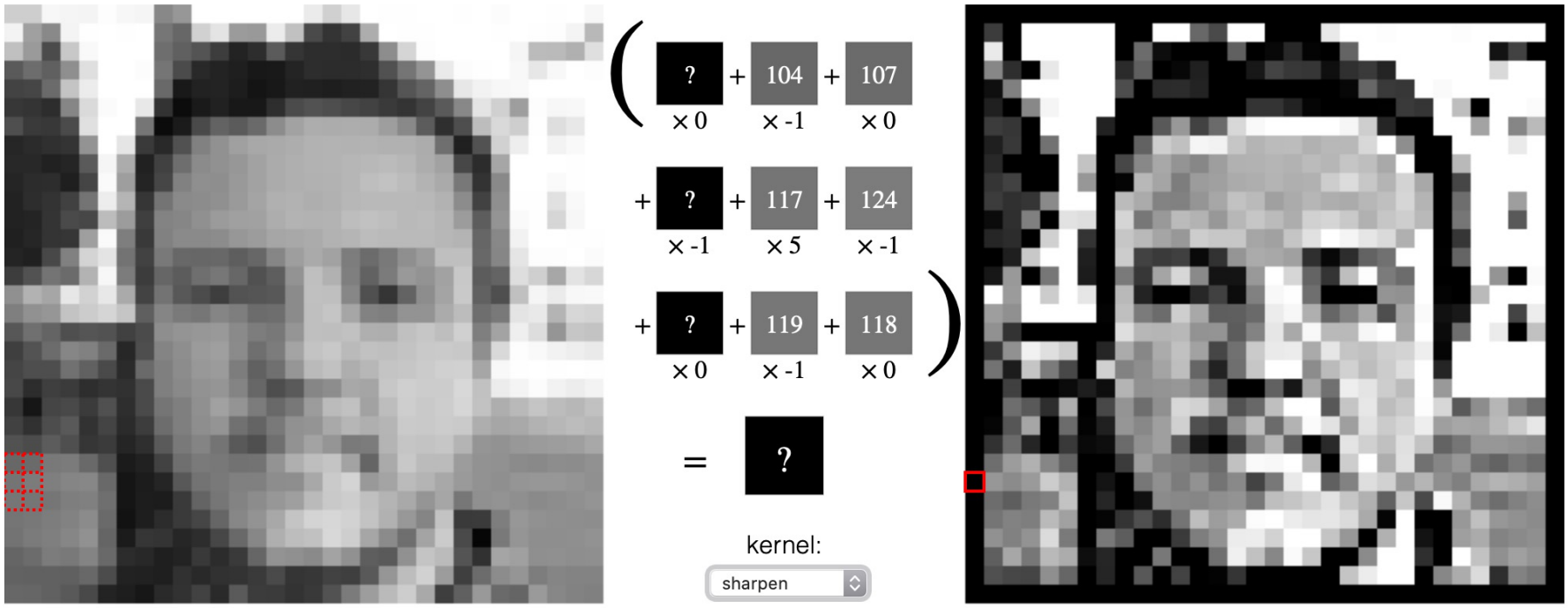
This is how image processing **filters** work!

- Outline:



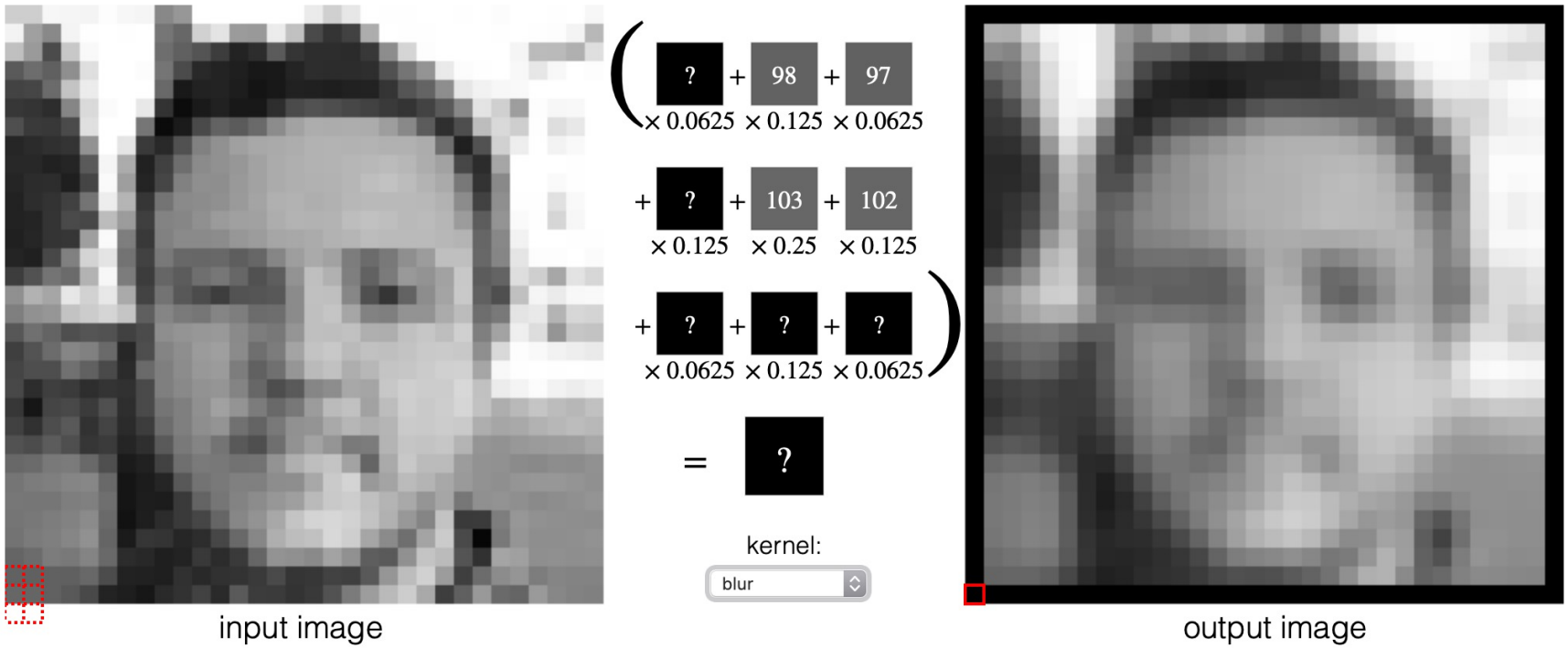
This is how image processing **filters** work!

- Sharpen:



This is how image processing **filters** work!

- Blur:



This is how image processing **filters** work!

- Design your own:

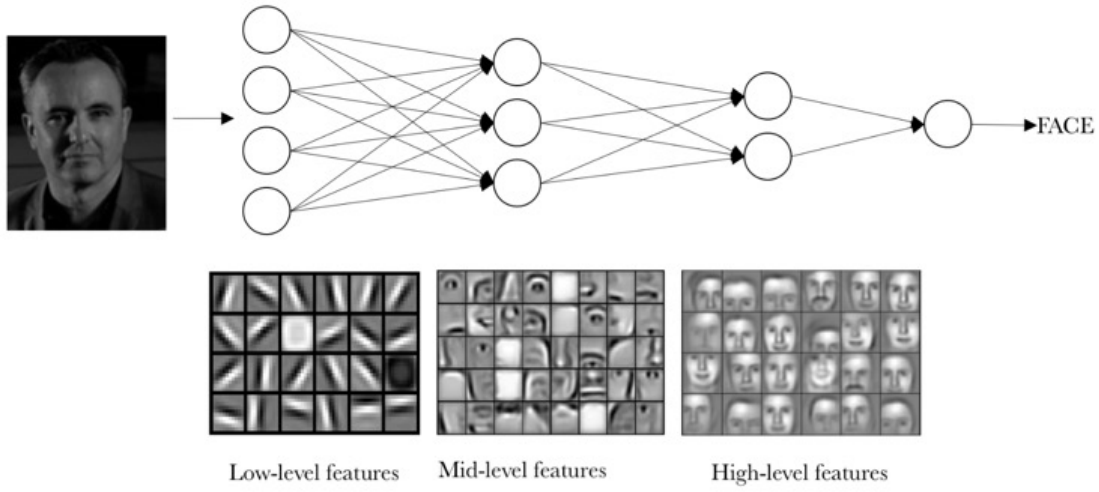
Choose File kop_lipari_2021.jpg

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

emboss

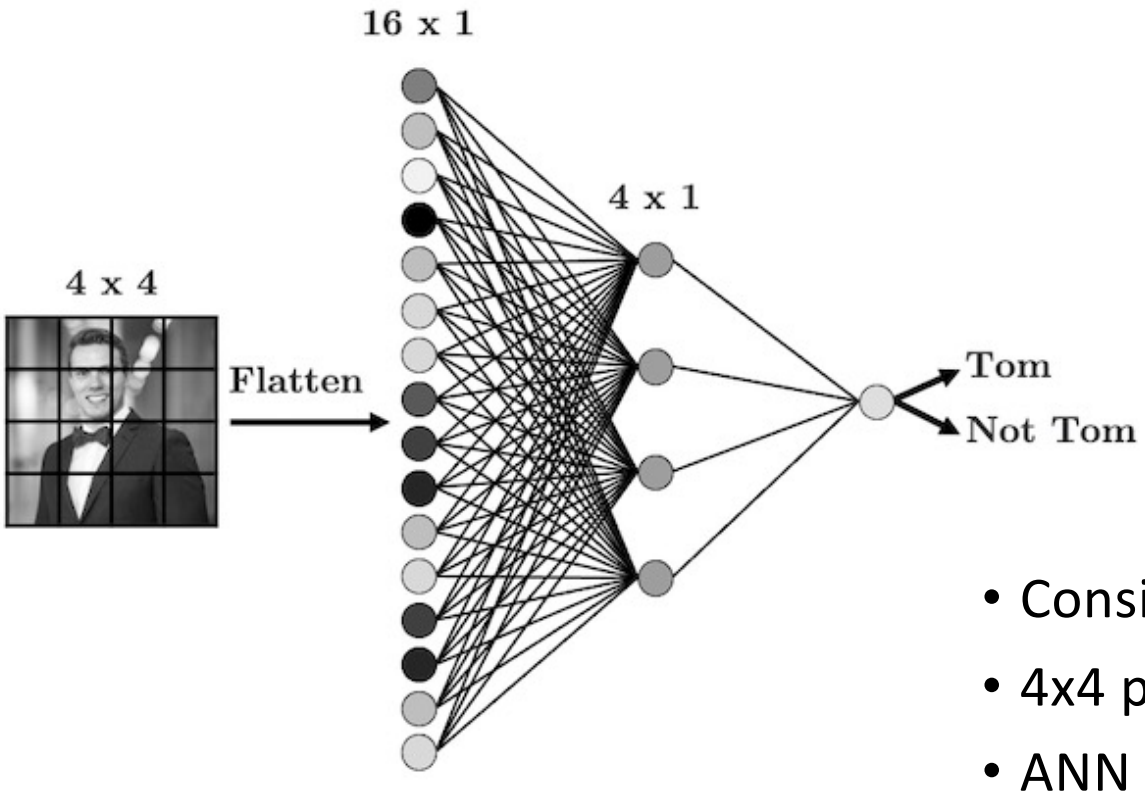
Image processing -> Convolutional NN

- **Observation**: tiny filter/kernel of, e.g., 3 x 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)?



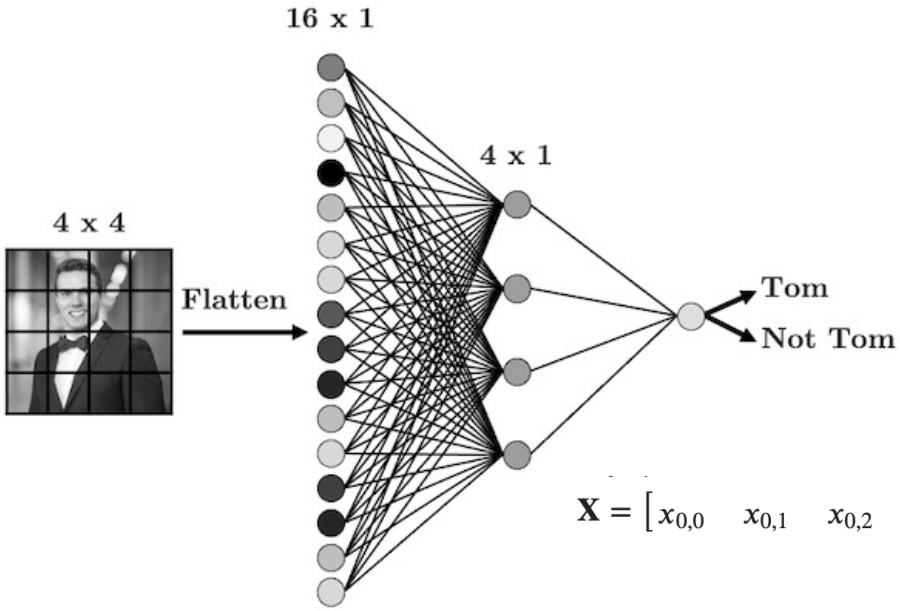
Equivalency between ANN & CNN

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



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

Total Parameters = (16 x 4 + 4) + (4 x 1 + 1) = 73



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

$$\mathbf{X}' = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix} \in \mathbb{R}^{4\times4}$$

- to:

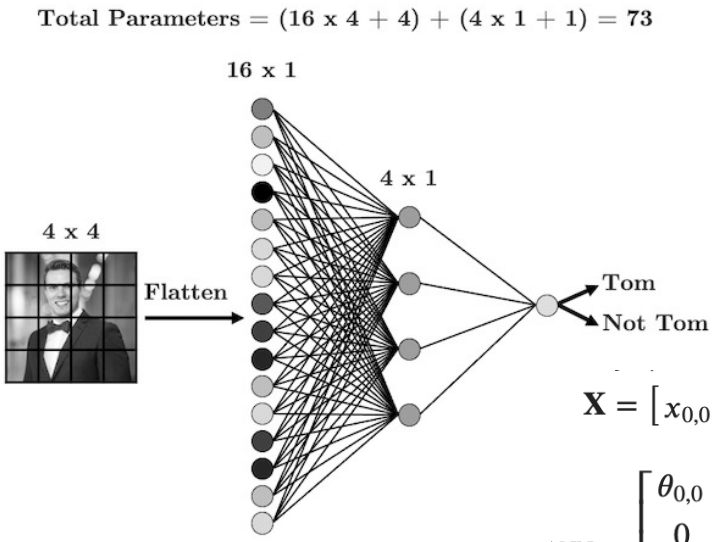
$$\mathbf{X} = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix}^T$$

- 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: $\mathbf{z}^{[2]} = \mathbf{\Theta}^{[1]} \cdot \mathbf{X}^{[1]}$
- **Non-linear activation**: $\mathbf{a}^{[2]} = g(\mathbf{z}^{[2]})$.



- **CNN**
- 3x3 kernel:

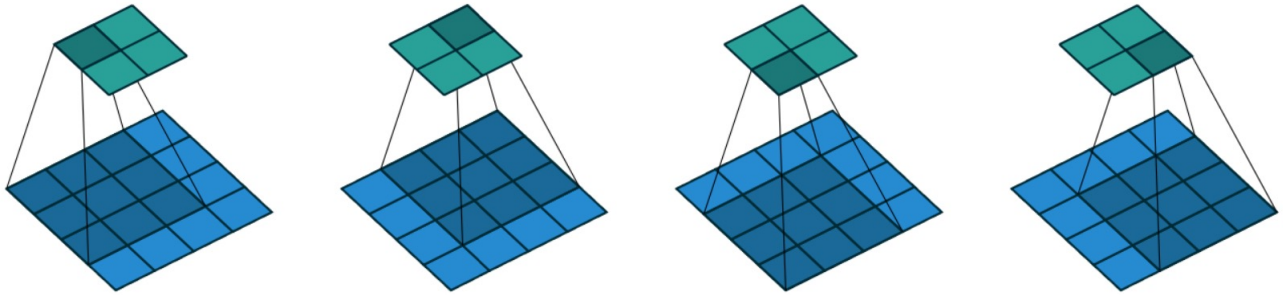
$$\Theta^{\text{CNN}} = \begin{bmatrix} \theta_{0,0} & \theta_{0,1} & \theta_{0,2} \\ \theta_{1,0} & \theta_{1,1} & \theta_{1,2} \\ \theta_{2,0} & \theta_{2,1} & \theta_{2,2} \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

- Can be re-written as 16 x 4 matrix (special kind, circulant/Toeplitz)

$$\mathbf{X} = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix}^T$$

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

- Everything else the same, *but* above matrix only has 9 unique values (vs. 73)
- 4x9 = 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.

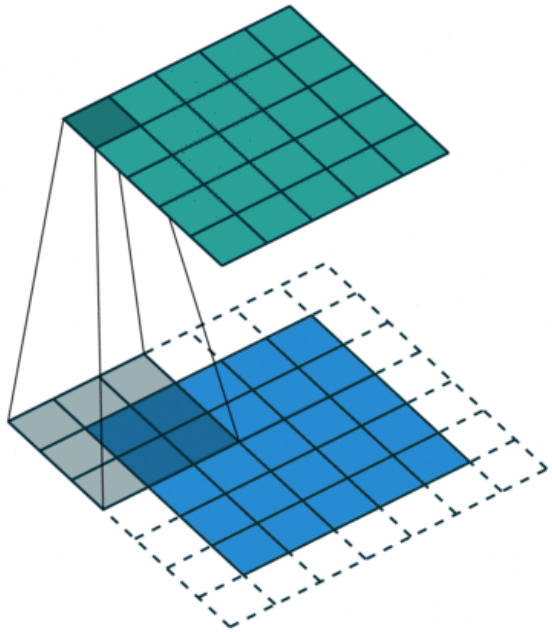
Technical Details

Padding

- Sometimes, we may want output to be same size as input
- ‘Pad’ image with one or more ‘dummy’ pixels, typically zero-valued

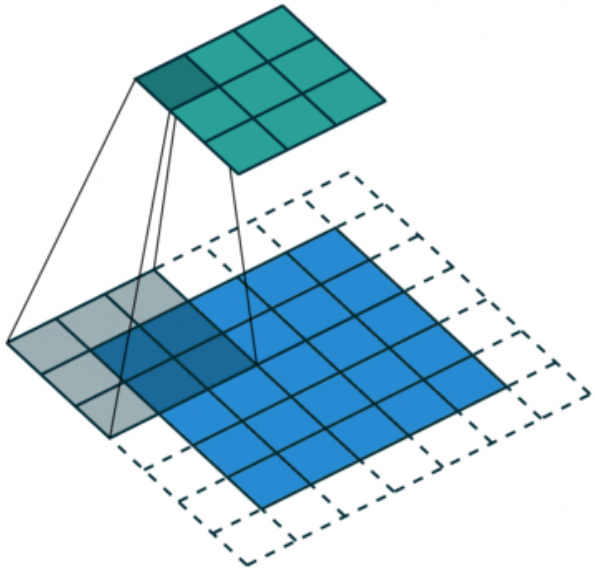
- This also applies filter ‘better’ near edges
- Sometimes we like to have, say, even pixel dimensions

- 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)



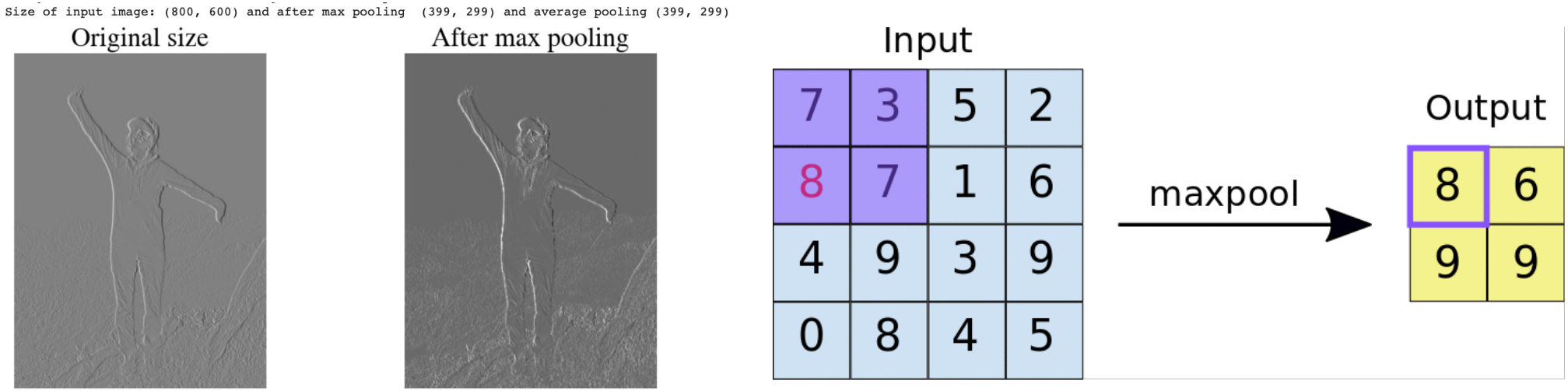
Stride

- 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**

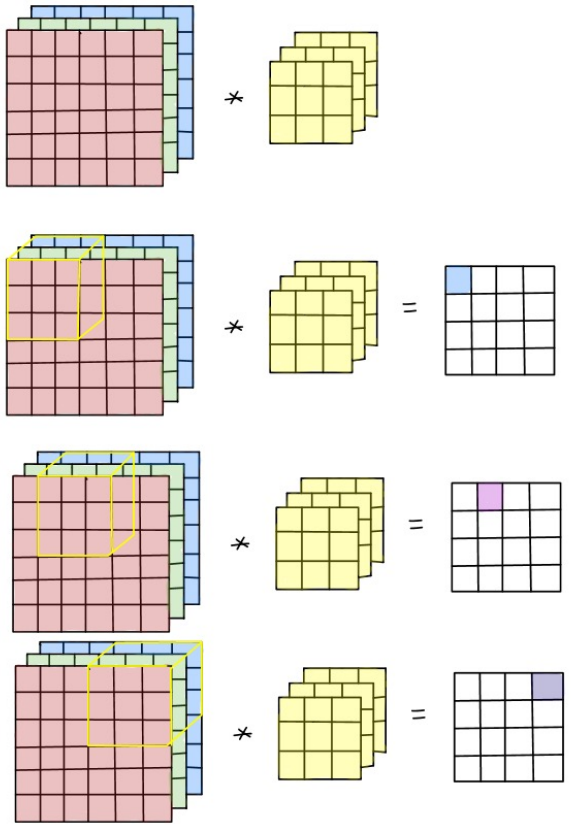


Max-Pooling (and Average Pooling)

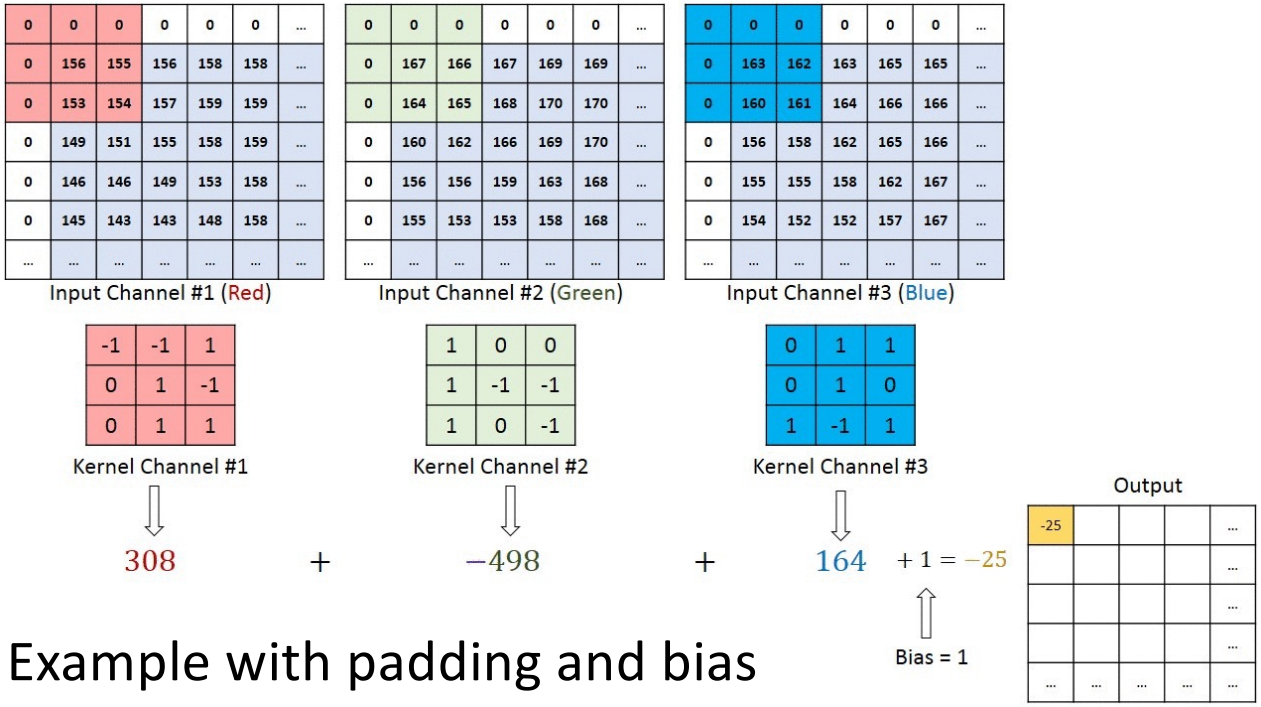
- Special type of convolution that takes maximum (or average) value inside kernel window (no adjustable weights!)
- Usually, stride is same as max-pool size such that there is no overlap
- Pooling again reduces data (compression), while enhancing features



Volume convolutions



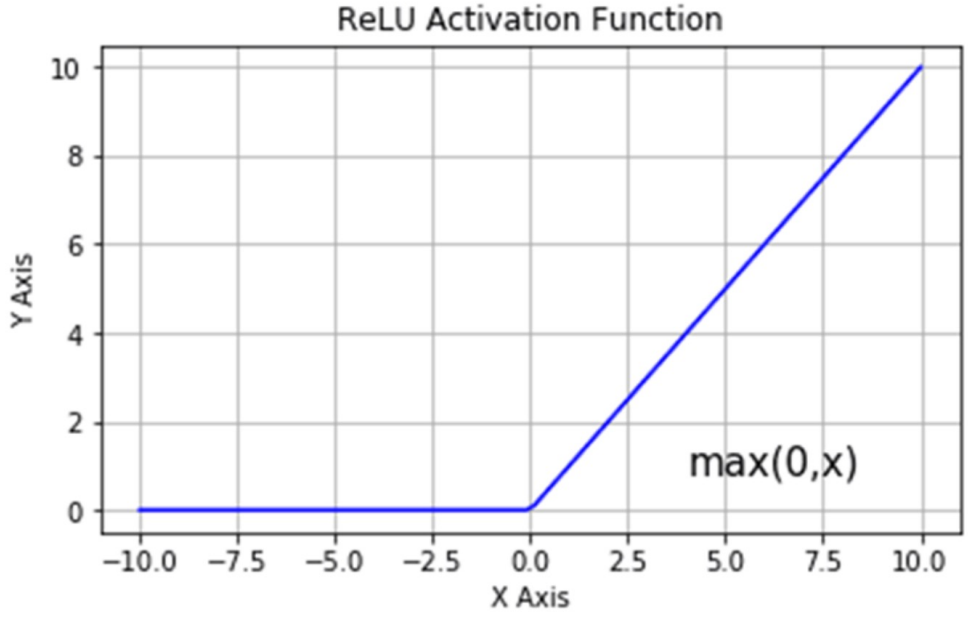
- When input has n_c channels, such as RGB (n_c=3), filter/kernel is n_f x n_f x n_c and also summed over n_c



- Example with padding and bias

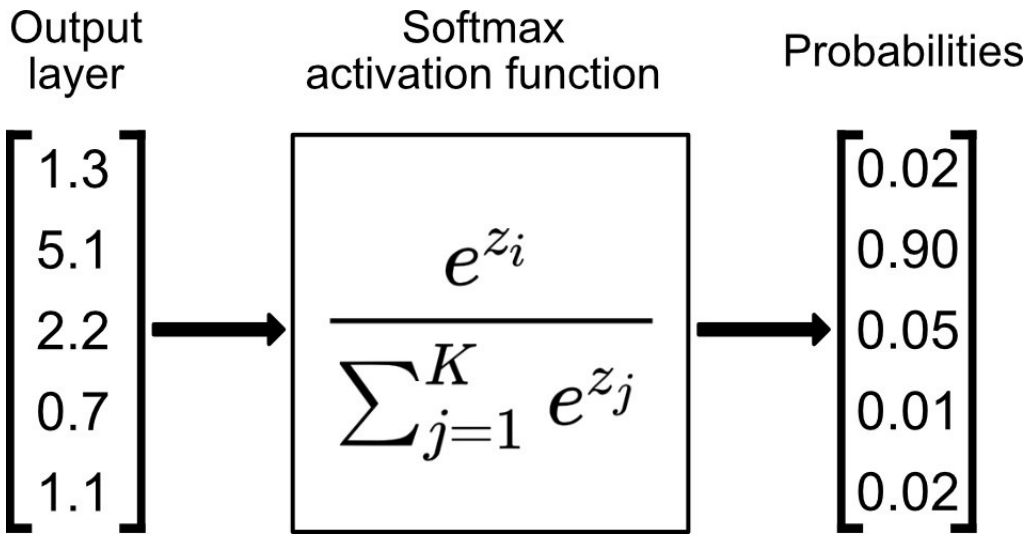
Activation function

- 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

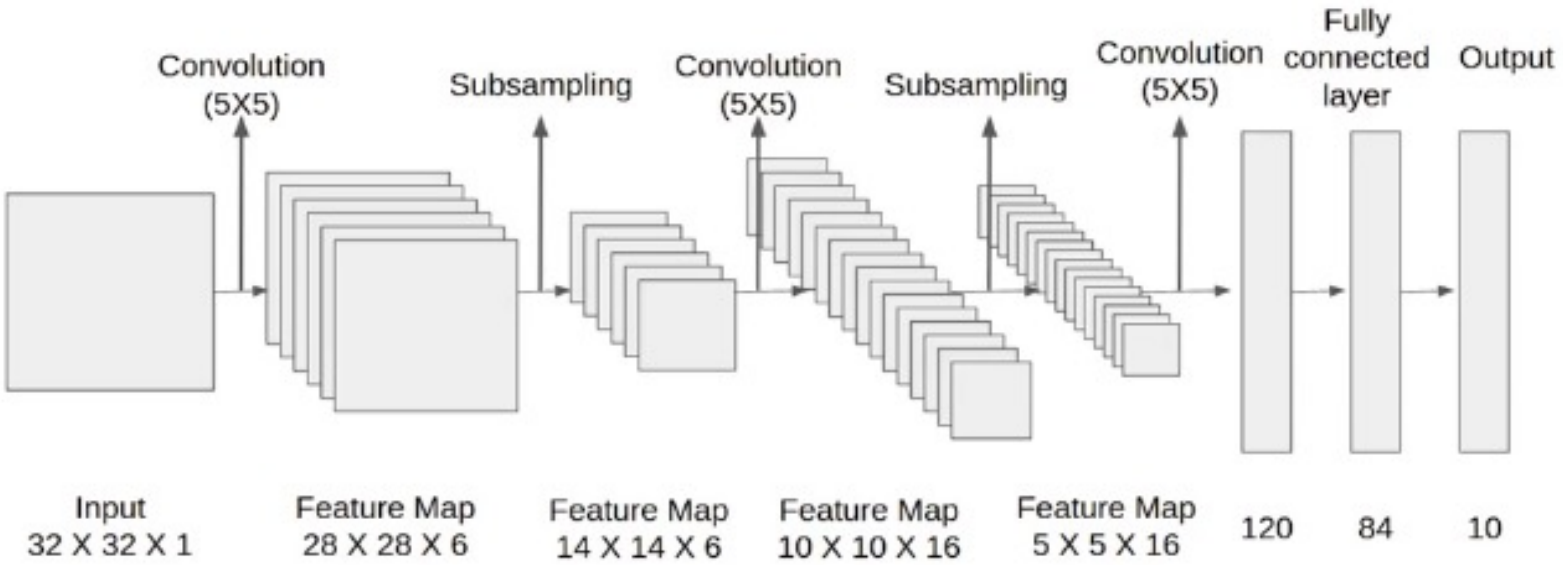
- For multi-class CNN, we use **softmax** instead of sigmoid activation function for the output layer
- Instead of binary 0/1, outputs **probabilities** for any nr of classes



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:

```
model = keras.Sequential()

model.add(layers.Conv2D(filters=6, kernel_size=(5, 5), activation='relu', input_shape=(32,32,1)))
model.add(layers.AveragePooling2D())

model.add(layers.Conv2D(filters=16, kernel_size=(5, 5), activation='relu'))
model.add(layers.AveragePooling2D())

model.add(layers.Flatten())

model.add(layers.Dense(units=120, activation='relu'))

model.add(layers.Dense(units=84, activation='relu'))

model.add(layers.Dense(units=10, activation = 'softmax'))
```

Python code

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

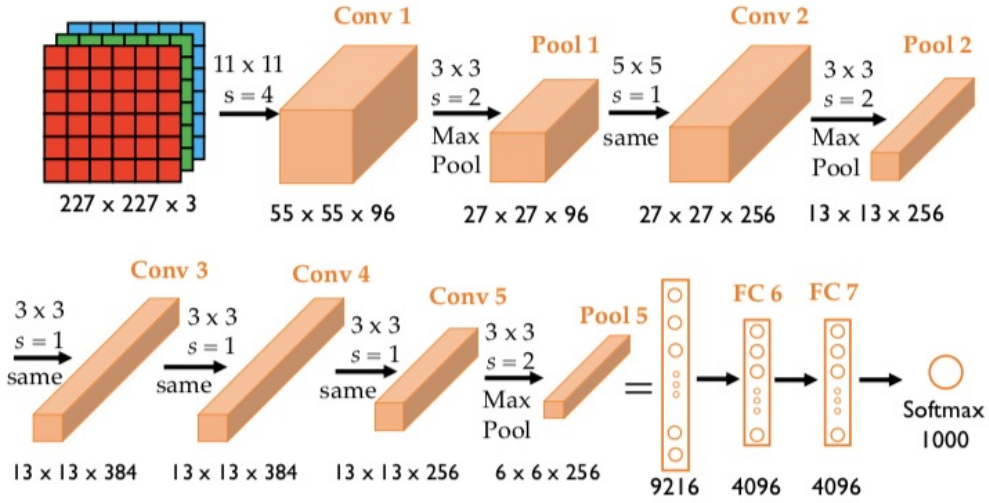
```
1  model.summary()
```

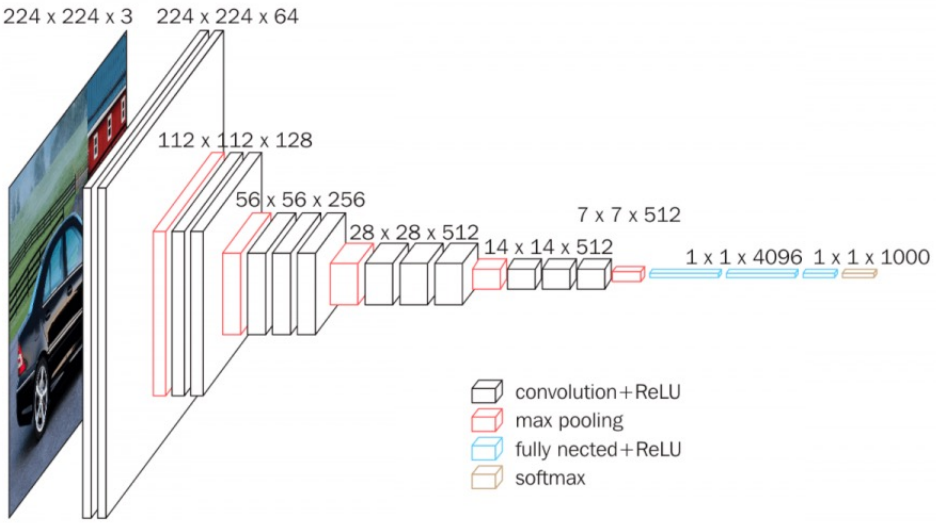
```
Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 6)         156

 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

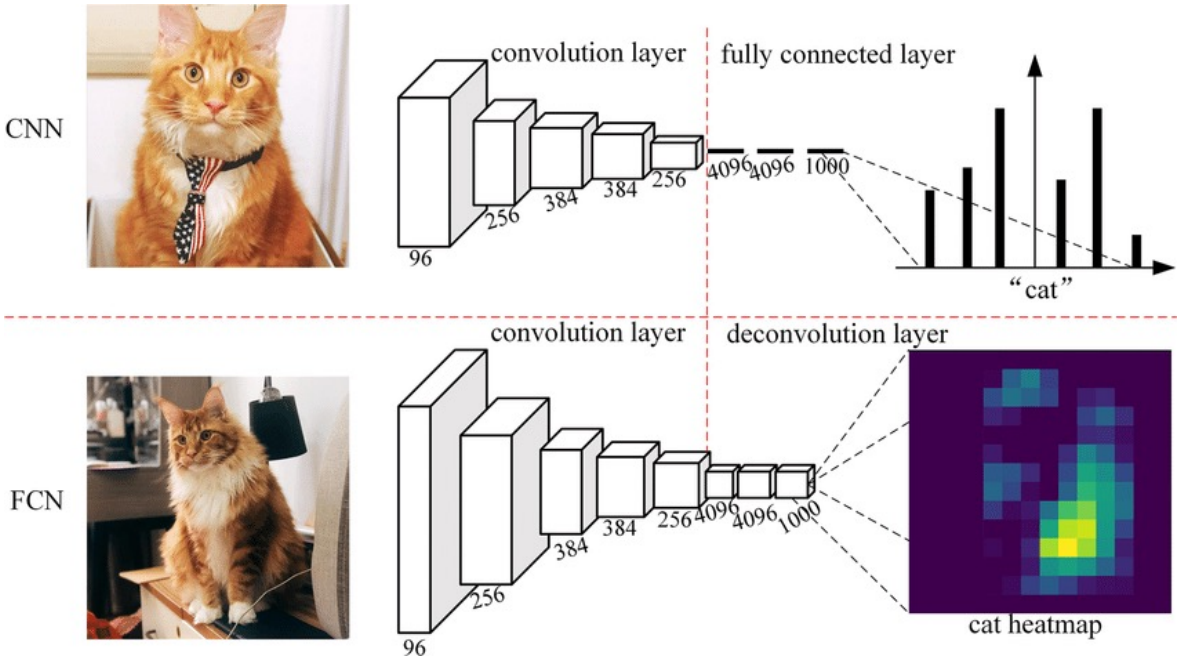
- AlexNet: 62 million weights
- VGG-16: 138 million





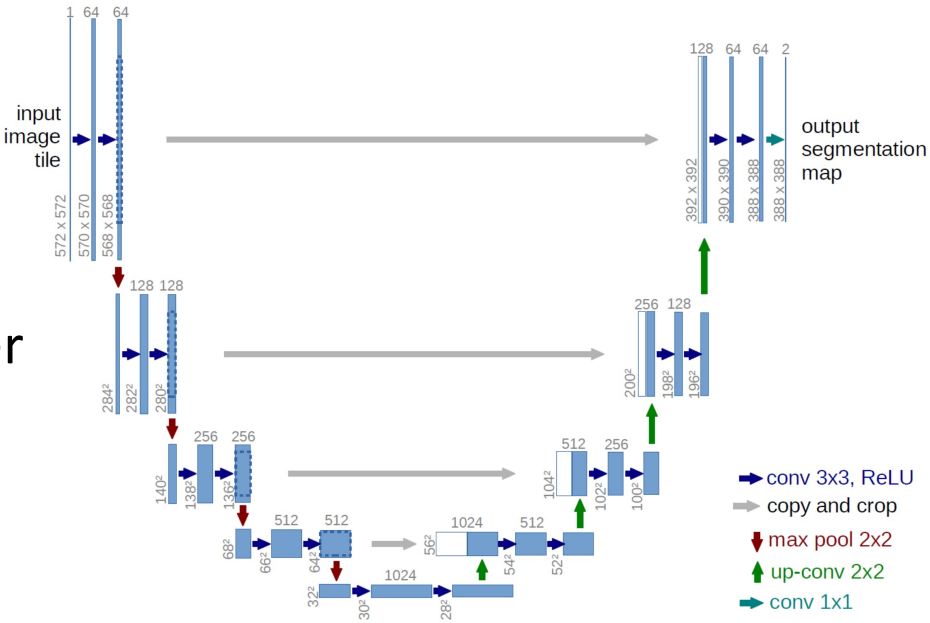
Fully Convolutional Neural Networks (FCN)

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



U-Nets for Segmentation

- Auto-encoder -> bottleneck -> decoder



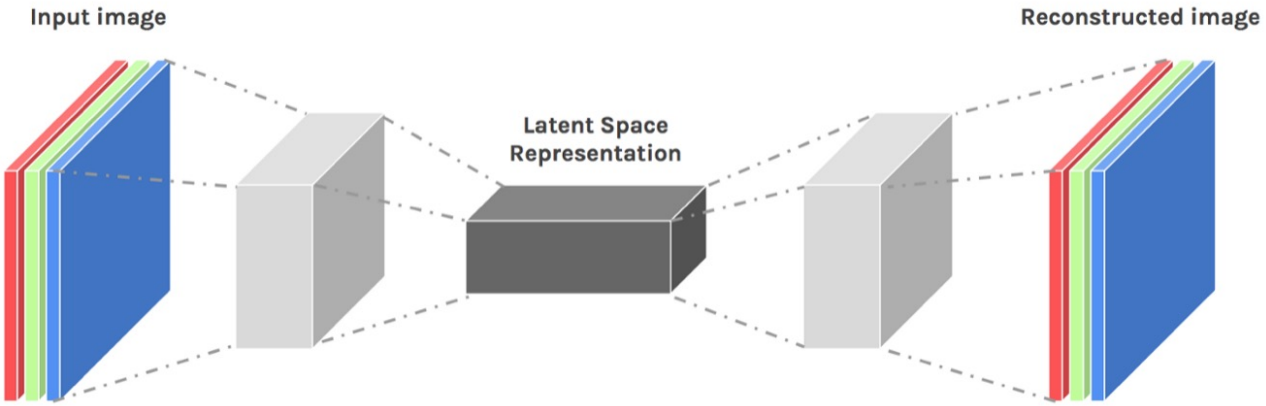
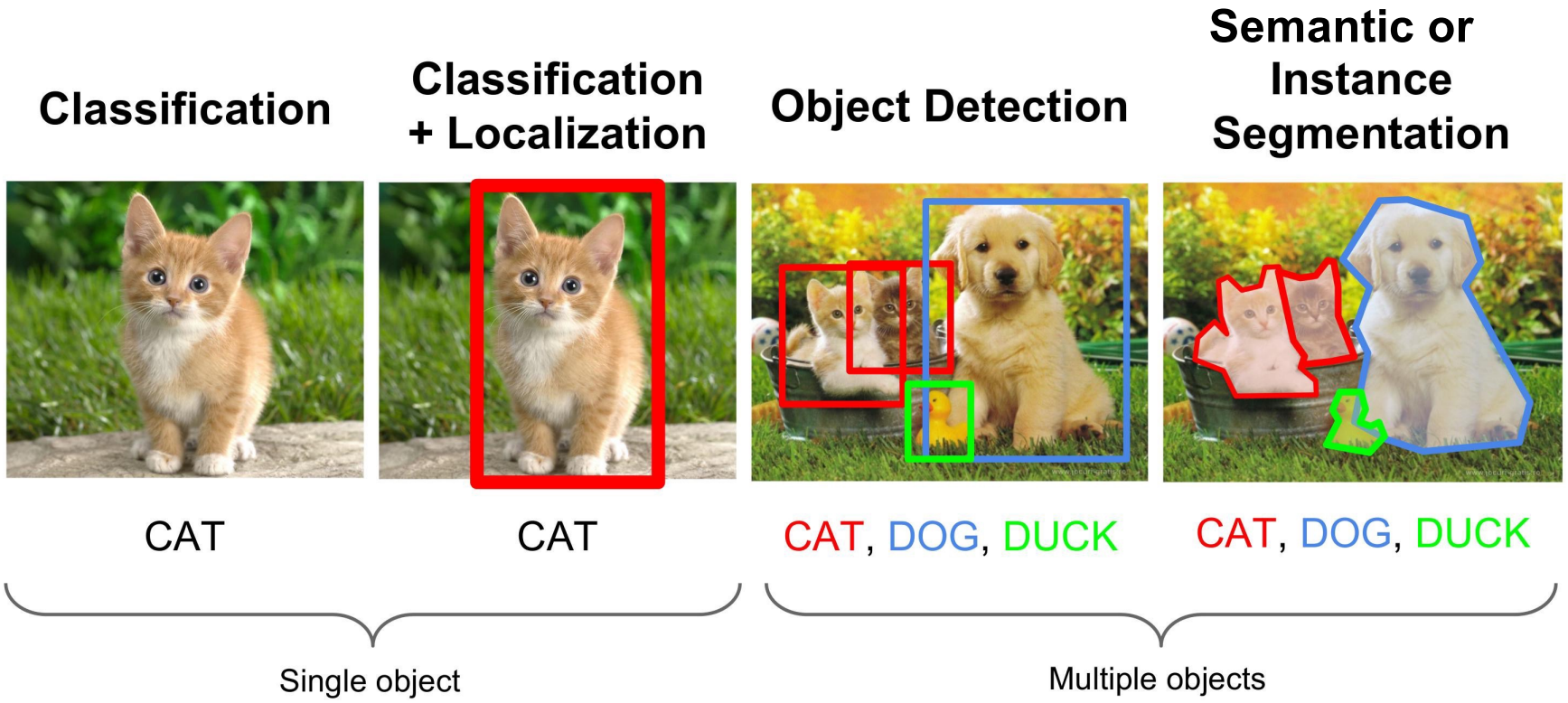


Figure 1: Auto-encoding an RGB image with two Conv2D followed by two Conv2DTranspose. (source)

Different Computer Vision Tasks

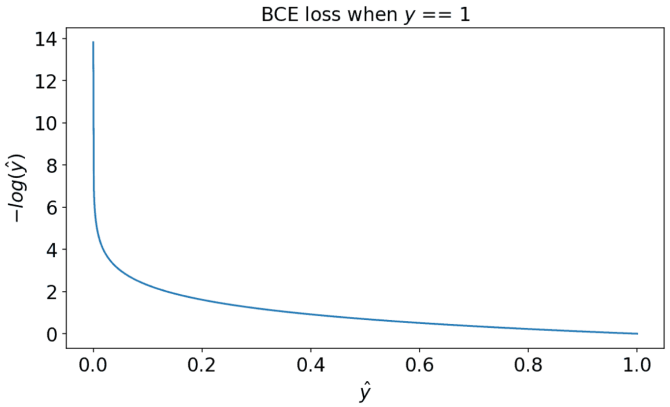


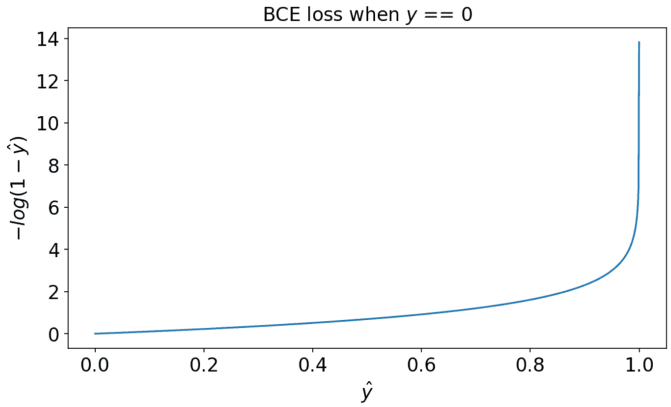
‘Accuracy’ Metrics for Segmentation

- 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			
	Total population $= P + N$	**Positive (PP)**	**Negative (PN)**	Informedness, bookmaker informedness (BM) $= TPR + TNR - 1$	Prevalence threshold (PT) $= \frac{\sqrt{TPR \times FPR} - FPR}{TPR - FPR}$
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 $= \frac{TP}{P} = 1 - FNR$	False negative rate (FNR), miss rate $= \frac{FN}{P} = 1 - TPR$
	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 $= \frac{FP}{N} = 1 - TNR$	True negative rate (TNR), specificity (SPC), selectivity $= \frac{TN}{N} = 1 - FPR$
	Prevalence $= \frac{P}{P + N}$	Positive predictive value (PPV), precision $= \frac{TP}{PP} = 1 - FDR$	False omission rate (FOR) $= \frac{FN}{PN} = 1 - NPV$	Positive likelihood ratio (LR+) $= \frac{TPR}{FPR}$	Negative likelihood ratio (LR−) $= \frac{FNR}{TNR}$
	Accuracy (ACC) $= \frac{TP + TN}{P + N}$	False discovery rate (FDR) $= \frac{FP}{PP} = 1 - PPV$	Negative predictive value (NPV) $= \frac{TN}{PN} = 1 - FOR$	Markedness (MK), deltaP (Δp) $= PPV + NPV - 1$	Diagnostic odds ratio (DOR) $= \frac{LR+}{LR-}$
	Balanced accuracy (BA) $= \frac{TPR + TNR}{2}$	F_1 score $= \frac{2PPV \times TPR}{PPV + TPR} = \frac{2TP}{2TP + FP + FN}$	Fowlkes–Mallows index (FM) $= \sqrt{PPV \times TPR}$	Matthews correlation coefficient (MCC) $= \sqrt{TPR \times TNR \times PPV \times NPV} - \sqrt{FNR \times FPR \times FOR \times FDR}$	Threat score (TS), critical success index (CSI), Jaccard index $= \frac{TP}{TP + FN + FP}$

‘Accuracy’ Metrics for Segmentation

- **Confusion matrix**

		Predicted condition	
	Total population = P + N	**Positive (PP)**	**Negative (PN)**
Actual condition	**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	
	Total population = P + N	**Positive (PP)**	**Negative (PN)**
Actual condition	**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**: $\frac{TP}{TP+\boldsymbol{FN}}$
- What fraction of pixels **labeled** as 1 is actually **predicted** as 1.

		Predicted condition	
	Total population = P + N	**Positive (PP)**	**Negative (PN)**
Actual condition	**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

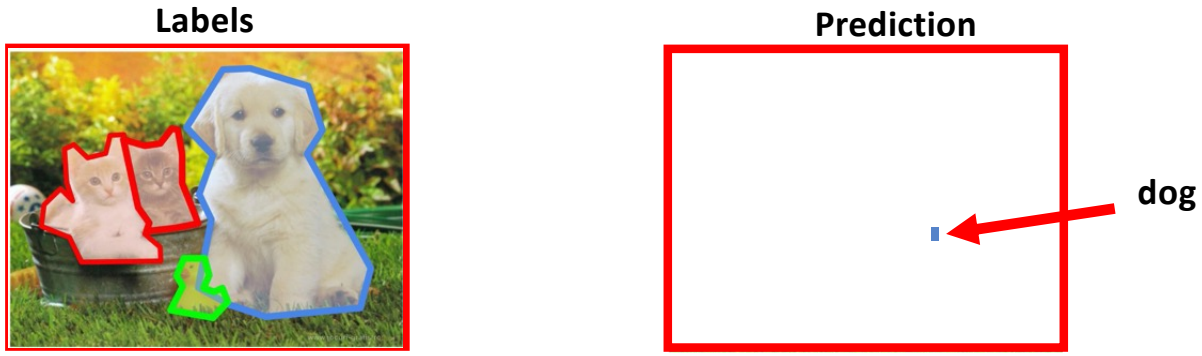
‘Accuracy’ Metrics for Segmentation

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

		Predicted condition	
	Total population = P + N	**Positive (PP)**	**Negative (PN)**
Actual condition	**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

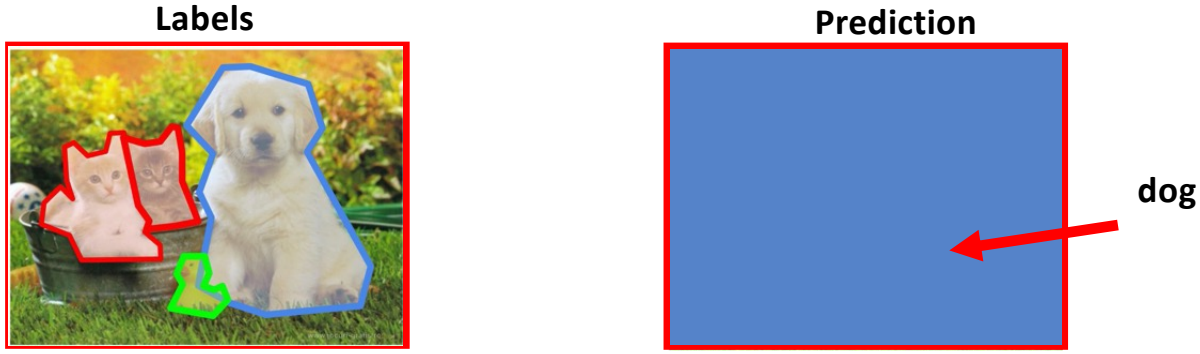
Questions:

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



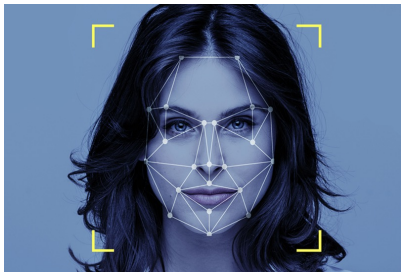
Questions:

- 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