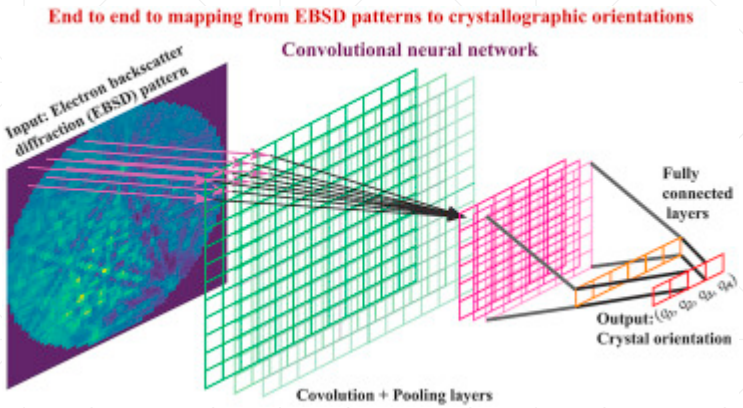



# Convolutional Neural Network

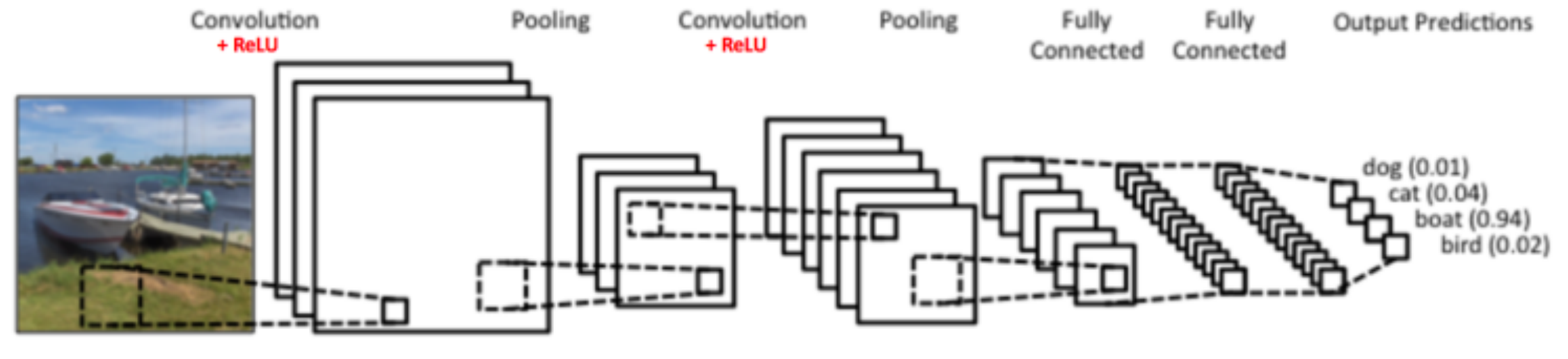
~Abhishek Kumar

# Convolutional Neural Network (CNN)

- Image recognition
- Classification
- Natural Language Processing tasks
- Multi Layer Perceptron's = Fully Connected Layers

# The LeNet Architecture (1990s)

Character recognition tasks such as reading zip codes, digits,

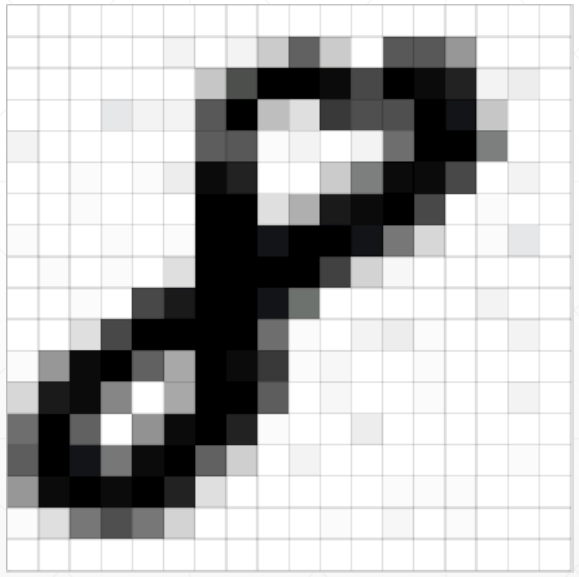


# Operations/ Basic building blocks

- Convolution
- Non Linearity (ReLU)
- Pooling or Sub Sampling
- Classification (Fully Connected Layer)

# Image is a matrix of pixel values

- Three channels – red, green and blue
- 0 to 255

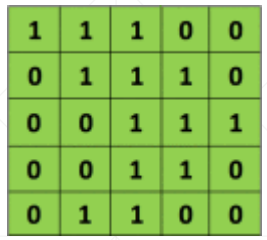


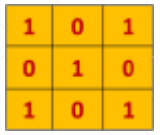
| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 12 | 0 | 11 | 39 | 137 | 37 | 0 | 152 | 147 | 84 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 41 | 168 | 250 | 255 | 235 | 162 | 255 | 238 | 206 | 11 | 13 | 0 |
| 0 | 0 | 0 | 16 | 9 | 9 | 150 | 251 | 45 | 21 | 184 | 159 | 154 | 255 | 233 | 40 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 145 | 146 | 3 | 10 | 0 | 11 | 124 | 253 | 255 | 107 | 0 | 0 |
| 0 | 0 | 3 | 0 | 4 | 15 | 236 | 216 | 0 | 0 | 38 | 109 | 247 | 240 | 169 | 0 | 11 | 0 |
| 1 | 0 | 2 | 0 | 0 | 0 | 253 | 253 | 23 | 62 | 224 | 241 | 255 | 164 | 0 | 5 | 0 | 0 |
| 6 | 0 | 0 | 4 | 0 | 3 | 252 | 250 | 228 | 255 | 255 | 234 | 112 | 28 | 0 | 2 | 17 | 0 |
| 0 | 2 | 1 | 4 | 0 | 21 | 255 | 253 | 251 | 255 | 172 | 31 | 8 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 4 | 0 | 163 | 225 | 251 | 255 | 229 | 120 | 0 | 0 | 0 | 0 | 0 | 11 | 0 | 0 |
| 0 | 0 | 21 | 162 | 255 | 255 | 254 | 255 | 126 | 6 | 0 | 10 | 14 | 6 | 0 | 0 | 9 | 0 |
| 3 | 79 | 242 | 255 | 141 | 66 | 255 | 245 | 189 | 7 | 8 | 0 | 0 | 5 | 0 | 0 | 0 | 0 |
| 26 | 221 | 237 | 98 | 0 | 67 | 251 | 255 | 144 | 0 | 8 | 0 | 0 | 7 | 0 | 0 | 11 | 0 |
| 125 | 255 | 141 | 0 | 87 | 244 | 255 | 208 | 3 | 0 | 0 | 13 | 0 | 1 | 0 | 1 | 0 | 0 |
| 145 | 248 | 228 | 116 | 235 | 255 | 141 | 34 | 0 | 11 | 0 | 1 | 0 | 0 | 0 | 1 | 3 | 0 |
| 85 | 237 | 253 | 246 | 255 | 210 | 21 | 1 | 0 | 1 | 0 | 0 | 6 | 2 | 4 | 0 | 0 | 0 |
| 6 | 23 | 112 | 157 | 114 | 32 | 0 | 0 | 0 | 0 | 2 | 0 | 8 | 0 | 7 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# 1. Convolution

- Extract features from the input image

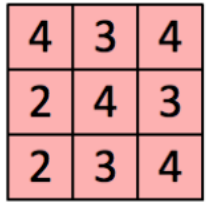
- Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.




Filter/
Kernel/
Feature detector

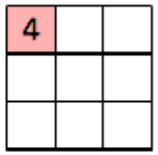

Image



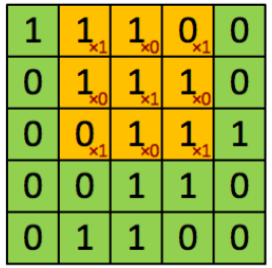

Convolved
Feature

/ Activation Map
/ Feature Map

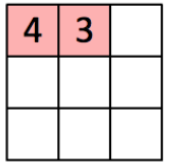

Image



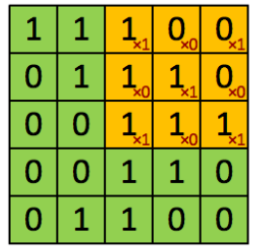

Convolved
Feature



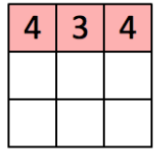

Image




Convolved
Feature

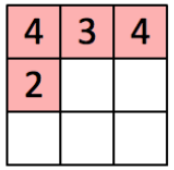



Image



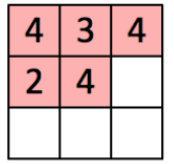

Convolved
Feature


Image




Convolved
Feature


Image




Convolved
Feature

- Different values of the filter matrix will produce different Feature Maps for the same input image.
- Edge Detection, Sharpen and Blur
- This means that different filters can detect different features from an image,
for example edges, curves etc

| Operation | Filter | Convolved Image |
|---|---|---|
| Identity | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |  |
| Edge detection | $\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$ |  |
| | $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ |  |
| | $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ |  |
| Sharpen | $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ |  |
| Box blur (normalized) | $\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ |  |
| Gaussian blur (approximation) | $\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ |  |

- CNN *learns* the values of these filters on its own during the training process

- Specify parameters such as number of filters, filter size, architecture of the network

- More number of filters = More image features get extracted = better our network at recognizing patterns in unseen images.

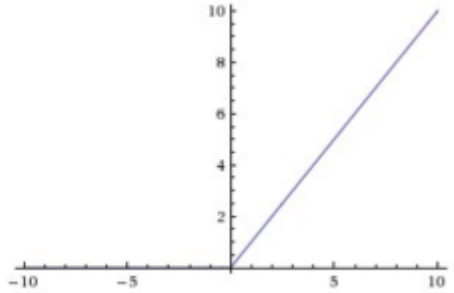
# Size of the Feature Map

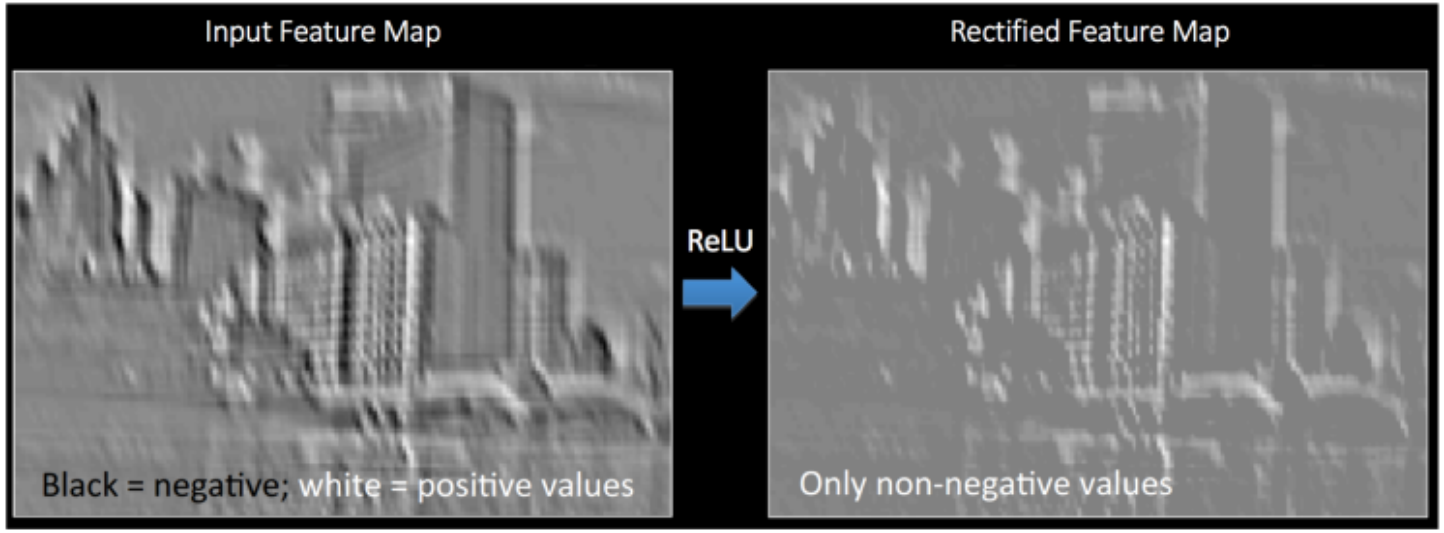
- **Depth:** number of filters we use for the convolution operation.
- **Stride:** number of pixels by which we slide our filter matrix over the input matrix.
- **Zero-padding:** pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix.
  - Wide Convolution
  - Narrow Convolution

# 2. ReLU (Rectified Linear Unit)

- Introducing Non Linearity
- Perform better in most situations

Output = Max(zero, Input)

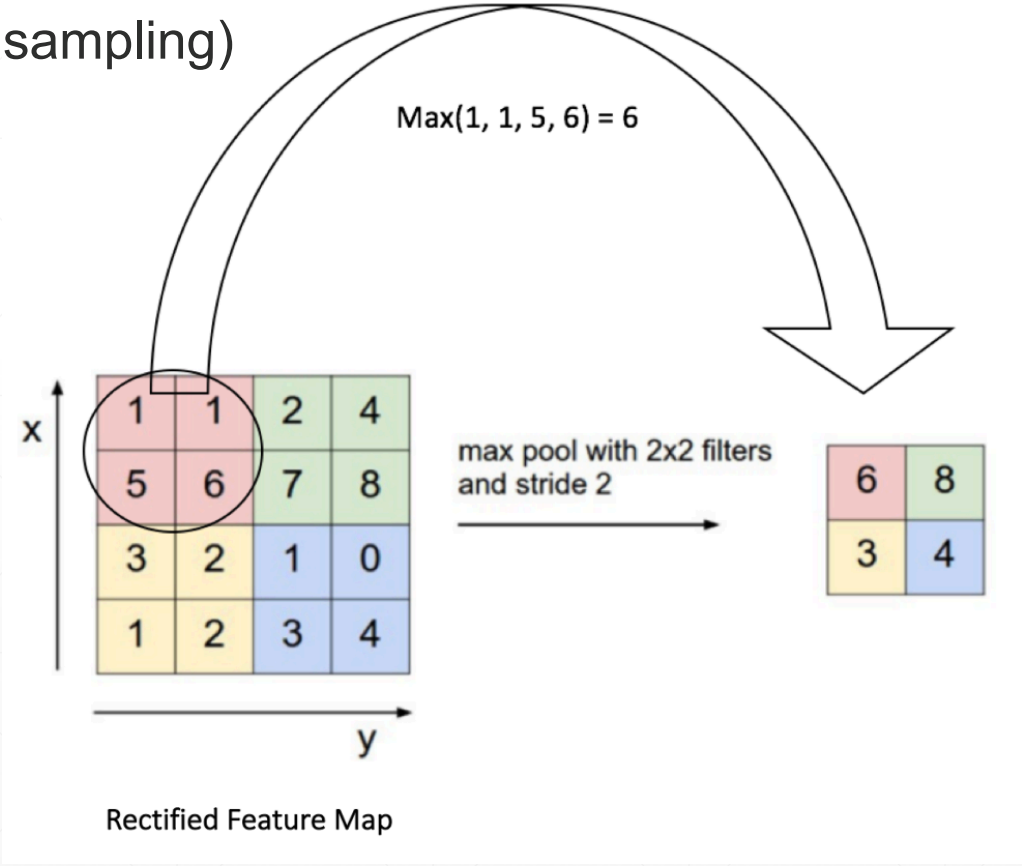


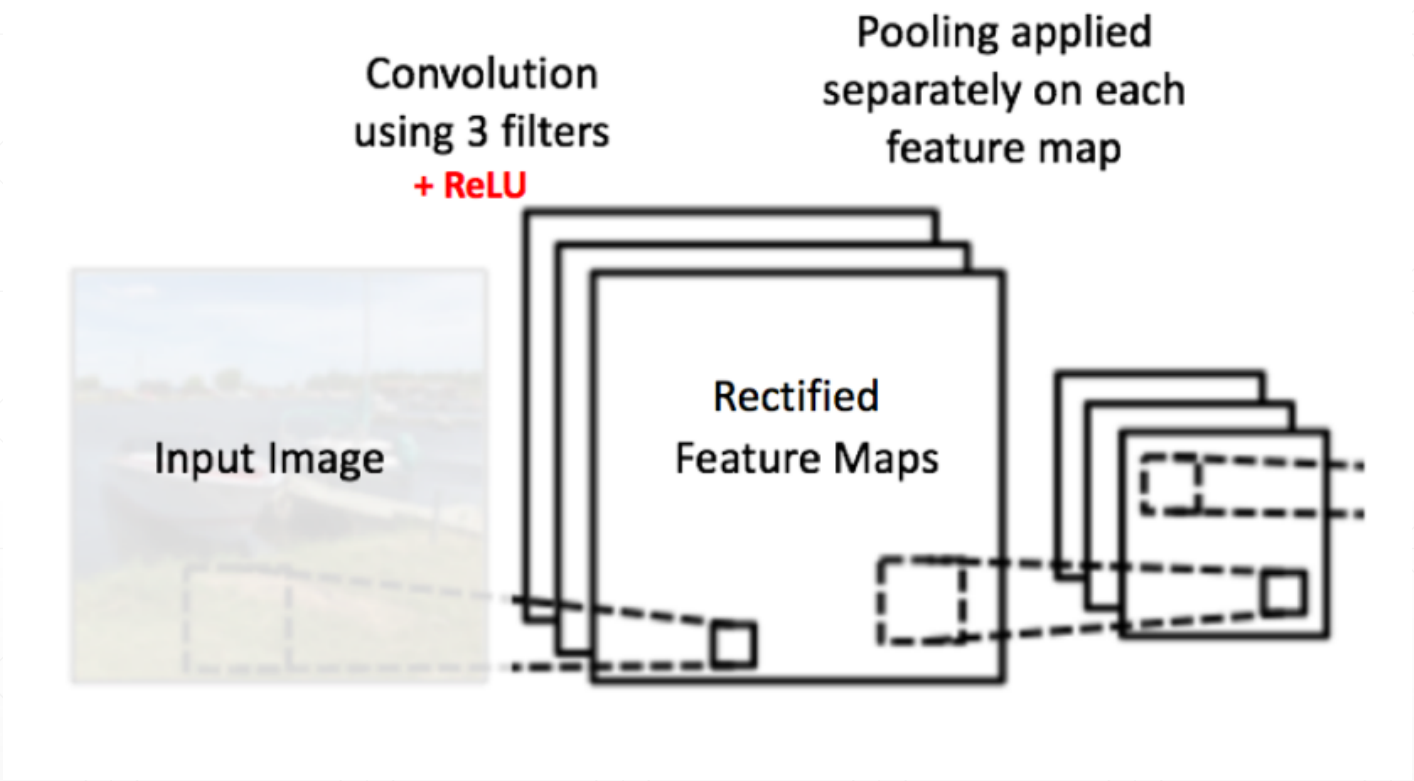



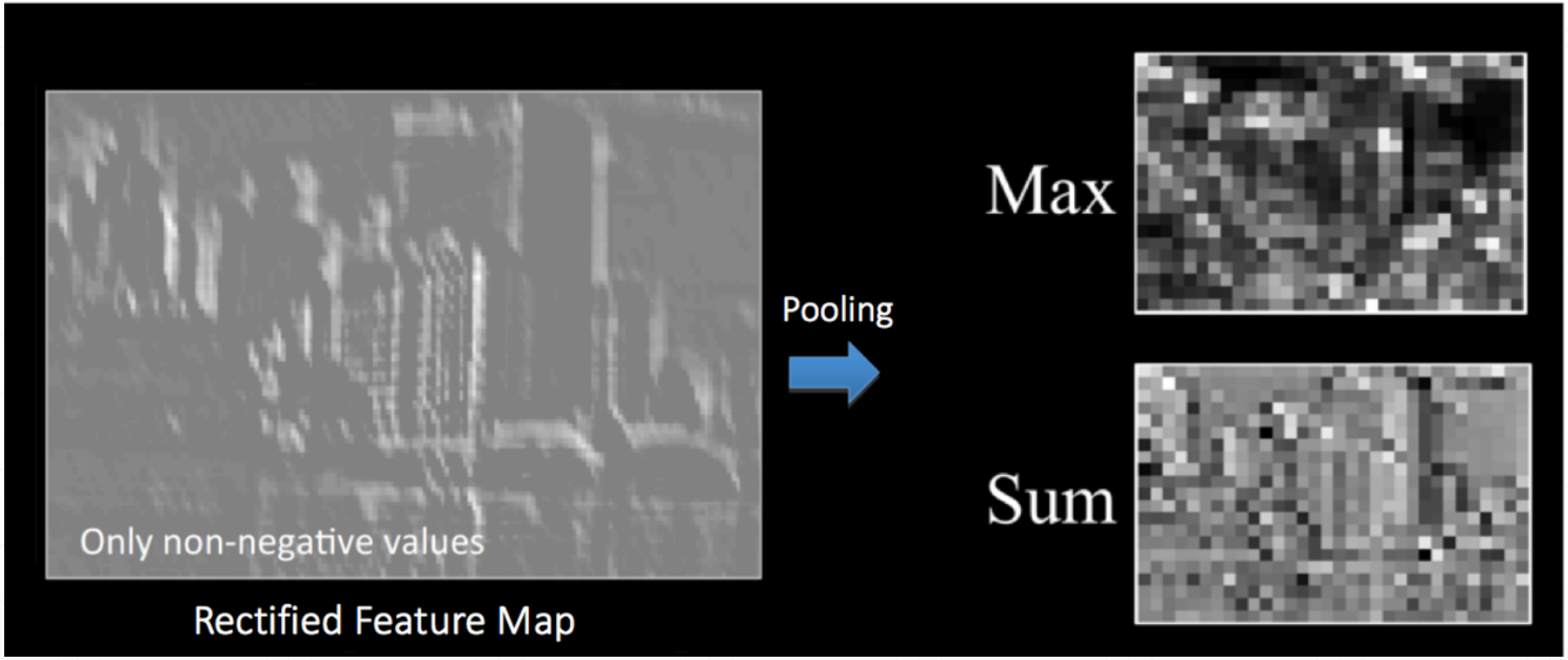
# 3. Pooling

- Spatial Pooling (also called subsampling or downsampling)

- Reduces the dimensionality of each feature map but retains the most important information. (Like PCA)
  - Max
  - Sum
  - Avg



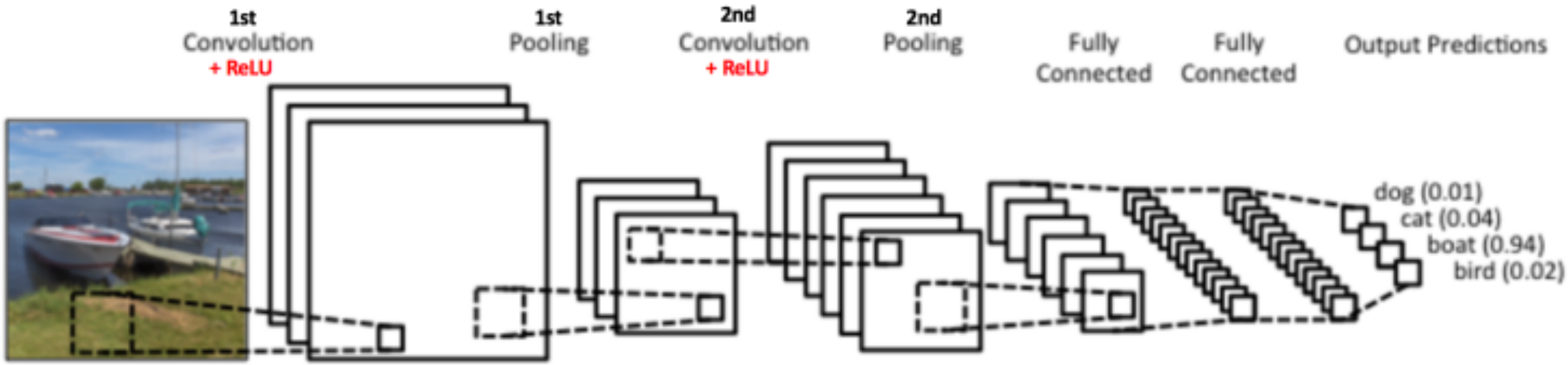

Convolution
using 3 filters
+ ReLU
Pooling applied
separately on each
feature map
Input Image
Rectified
Feature Maps


Only non-negative values
Rectified Feature Map
Pooling
Max
Sum

- Makes the input representations (feature dimension) smaller and more manageable

- reduces the number of parameters and computations in the network, therefore, controlling overfitting.

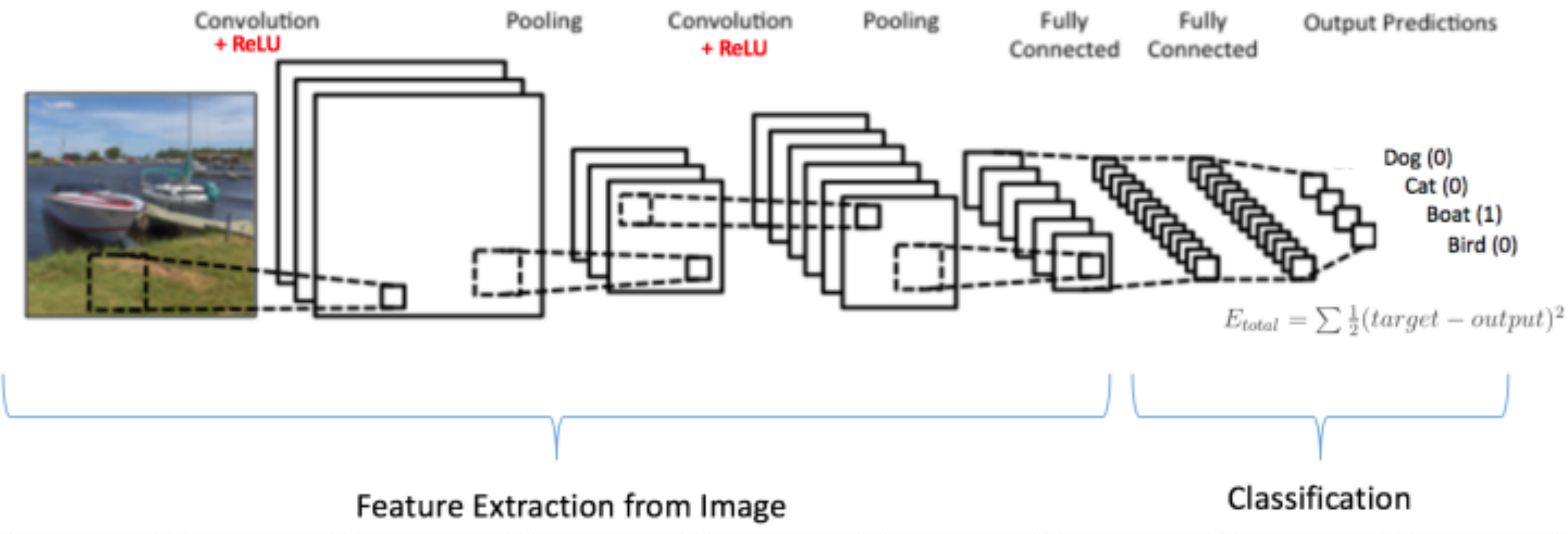
- Makes the network invariant to small transformations, distortions and translations in the input image. (since we take the maximum / average value in a local neighborhood).




Together these layers extract the useful features from the images,
introduce non-linearity in our network
and reduce feature dimension while aiming to make the features somewhat equivariant to scale and translation

# 4. Fully Connected Layer

- Output of the 2nd Pooling Layer = Input to the Fully Connected Layer
- Multi Layer Perceptron that uses a activation function in the output layer (eg. Softmax)
- The sum of output probabilities from the Fully Connected Layer is 1


Convolution
+ ReLU
Pooling
Convolution
+ ReLU
Pooling
Fully
Connected
Fully
Connected
Output Predictions
Dog (0)
Cat (0)
Boat (1)
Bird (0)
$E_{total} = \sum \frac{1}{2}(target - output)^2$
Feature Extraction from Image
Classification

# Step1:

- We initialize all filters and parameters / weights with random values

# Step2:

- The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.
  - Lets say the output probabilities for the boat image above are [0.2, 0.4, 0.1, 0.3]
  - Since weights are randomly assigned for the first training example, output probabilities are also random

# Step3:

- Calculate the total error at the output layer (summation over all 4 classes)
  - **Total Error = $\sum$ ½ (target probability – output probability) $^2$**

# Step4:

- Use Backpropagation to calculate the *gradients* of the error with respect to all weights in the network and use *gradient descent* to update all filter values / weights and parameter values to minimize the output error.

  - Parameters like number of filters, filter sizes, architecture of the network etc. have all been fixed before Step 1 and do not change during training process – only the values of the filter matrix and connection weights get updated.

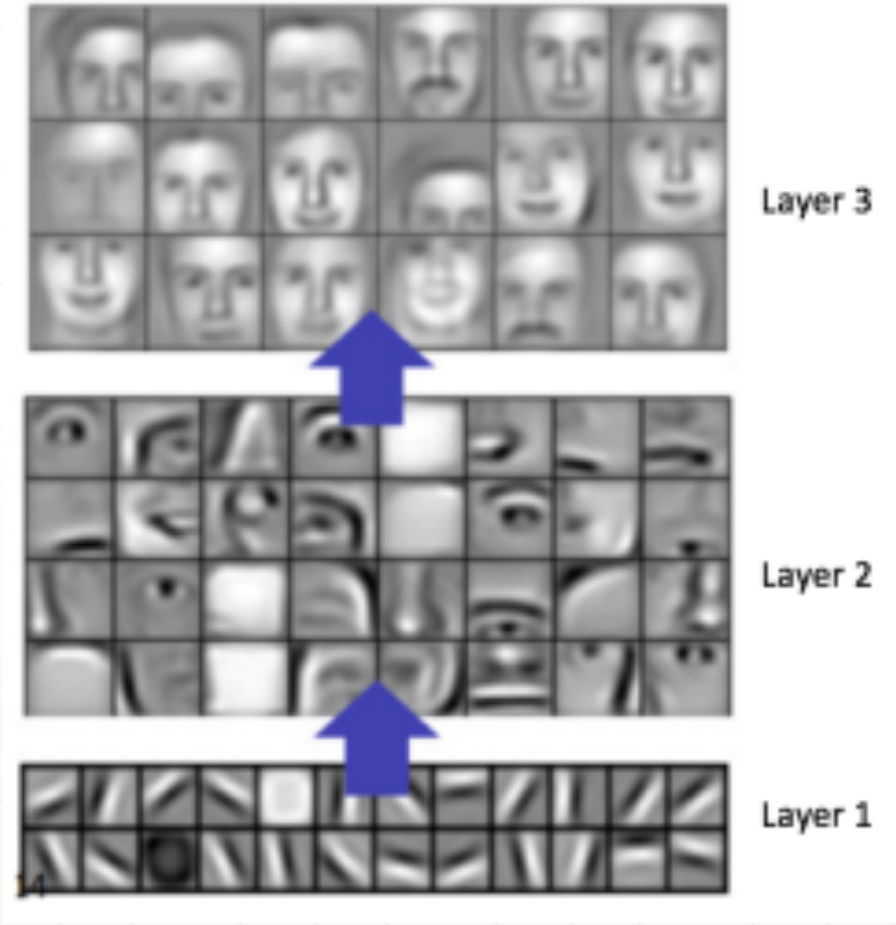
# Step5:

- Repeat steps 2-4 with all images in the training set.

- In general, the more convolution steps we have, the more complicated features our network will be able to learn to recognize.

For example, in Image Classification

- detect edges from raw pixels in the first layer,
- use the edges to detect simple shapes in the second layer,
- and then use these shapes to deter higher-level features, such as facial shapes in higher layers



# Adam Harley's example on MNIST database

- http://scs.ryerson.ca/~aharley/vis/conv/flat.html

# Python code implementation

- Jupyter Notebook

# Discussion


# Thank you!