Expert Systems

PROFESSOR DARIAN M. ONCHIS

HTTPS://STAFF.FMI.UVT.RO/~DARIAN.ONCHIS/

EUROPEAN LABORATORY FOR LEARNING AND INTELLIGENT SYSTEMS (ELLIS)

DARIAN.ONCHIS@E-UVT.RO

S·I·M·T = Signal, Image and Machine Learning Team


Is there a cat in this image?

Suppose we have a ‘cat-filter’ W

Suppose we have a ‘cat-filter’ W

Suppose we have a 'cat-filter' W

C-S Inequality

Suppose we have a ‘cat-filter’ W

C-S Inequality

For any

$$X = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix}$$ we have

$$X \cdot W \leq (X \cdot X)^{1/2} (W \cdot W)^{1/2}$$

with equality if and only if X is parallel to a cat.

Suppose we have a 'cat-filter' W

(Cat-Selection) C-S Inequality

For any

$$X = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix}$$ we have

$$X \cdot W \leq (X \cdot X)^{1/2}(W \cdot W)^{1/2}$$

with equality if and only if X is parallel to a cat.

Suppose we have a 'cat-filter' W

(Cat-Selection) C-S Inequality

For any

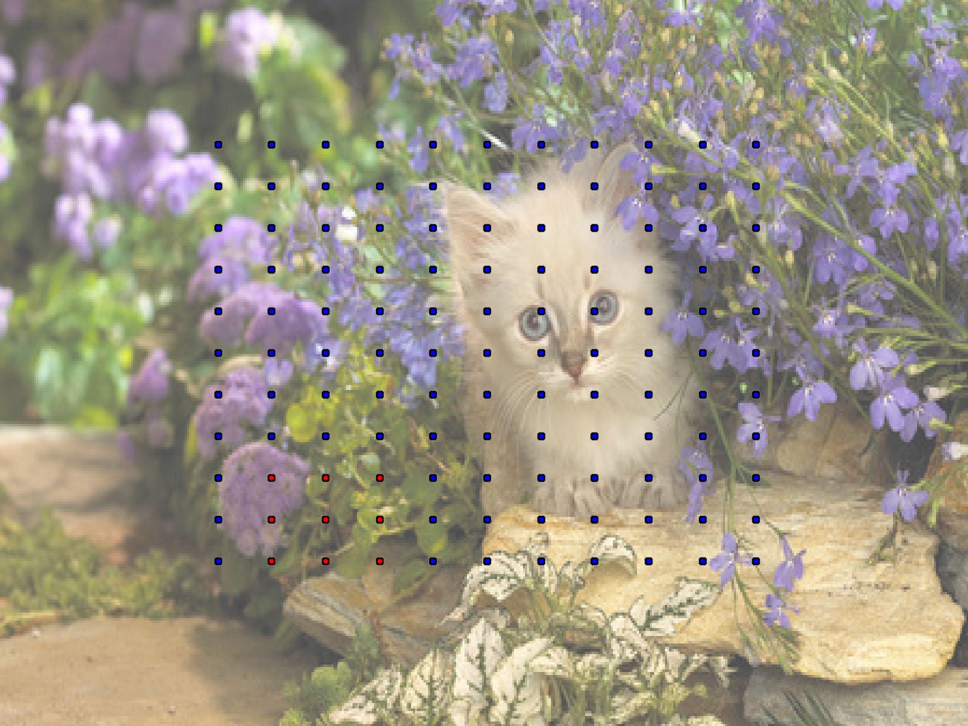
$$X = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix}$$ we

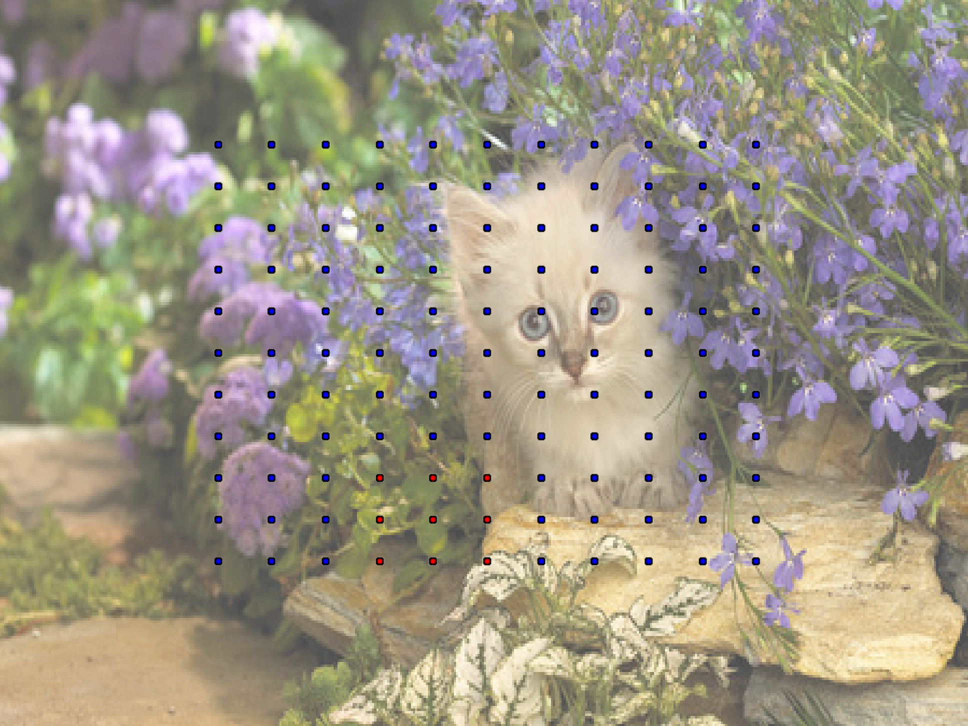
have

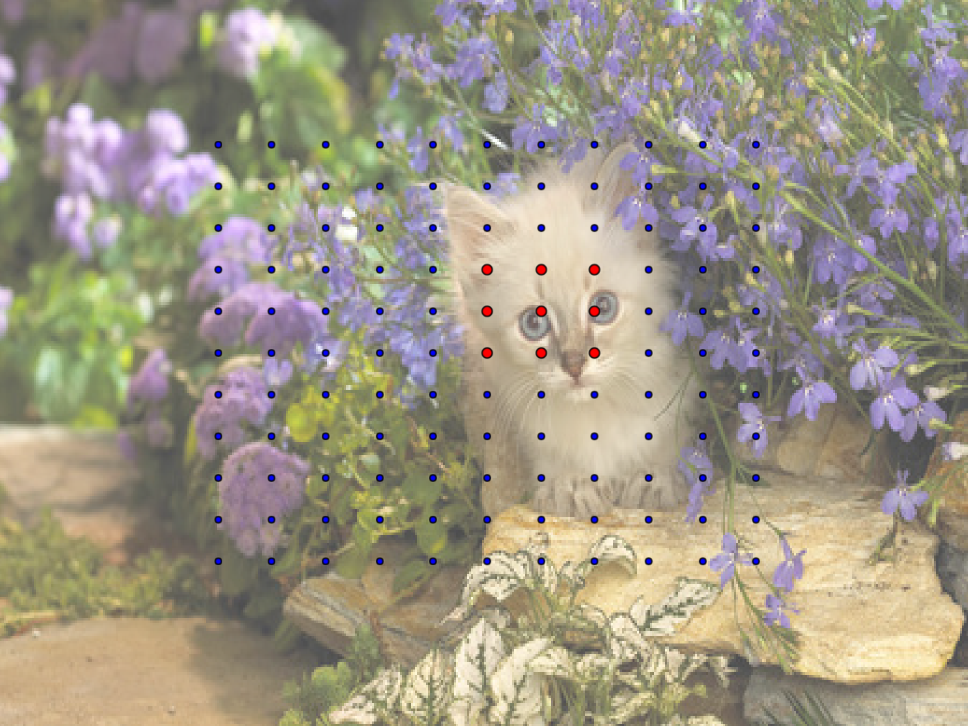
$$X \cdot W \leq (X \cdot X)^{1/2} (W \cdot W)^{1/2}$$

with equality if and only if X is parallel to a cat.

perform cat-test on all 3×3 image subpatches!







Convolution

Definition

Suppose that $X, Y \in \mathbb{R}^{n\times n}$. Then $Z = X * Y \in \mathbb{R}^{n\times n}$ is defined as

$$Z[i,j] = \sum_{k,l=0}^{n-1} X[i-k, j-l]Y[k,l],$$

where periodization or zero-padding of X, Y is used if $i-k$ or $j-l$ is not in $\{0,\ldots,n-1\}$.

Convolution

Definition

Suppose that $X, Y \in \mathbb{R}^{n \times n}$. Then $Z = X * Y \in \mathbb{R}^{n \times n}$ is defined as

$$Z[i,j] = \sum_{k,l=0}^{n-1} X[i-k, j-l] Y[k,l],$$

where periodization or zero-padding of X, Y is used if $i-k$ or $j-l$ is not in $\{0, \ldots, n-1\}$.

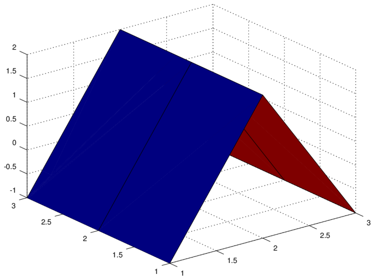
Efficient computation possible via FFT (or directly if X or Y are sparse)!

Example: Detecting Vertical Edges

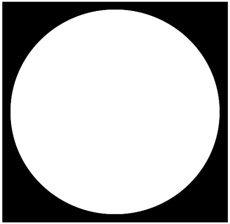
Given 'vertical-edge-detection-filter' $W = \begin{pmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{pmatrix}$

Example: Detecting Vertical Edges

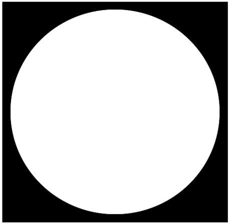
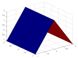
Given 'vertical-edge-detection-filter' $W = \begin{pmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{pmatrix}$



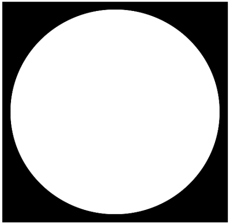
Example: Detecting Vertical Edges



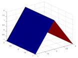
Example: Detecting Vertical Edges

 $*$ 

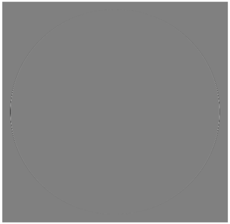
Example: Detecting Vertical Edges



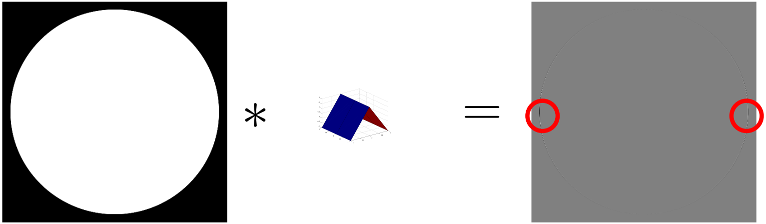
$*$



$=$



Example: Detecting Vertical Edges

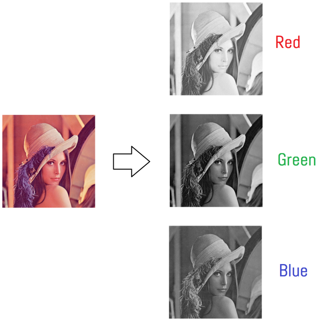


Introducing Convolutional Nodes

- A convolutional node accepts as input a stack of images, e.g. $X \in \mathbb{R}^{n_1 \times n_2 \times S}$.

Introducing Convolutional Nodes

- A convolutional node accepts as input a stack of images, e.g. $X \in \mathbb{R}^{n_1 \times n_2 \times S}$.



Introducing Convolutional Nodes

- A convolutional node accepts as input a stack of images, e.g. $X \in \mathbb{R}^{n_1 \times n_2 \times S}$.
- Given a filter $W \in \mathbb{R}^{F \times F \times S}$, where F is the *spatial extent* and a *bias* $b \in \mathbb{R}$, it computes a matrix

$$Z = W *_{12} X := \sum_{i=1}^{S} X[:,:,i] * W[:,:,i] + b.$$

Introducing Convolutional Layers

- A convolutional node accepts as input a stack of images, e.g. $X \in \mathbb{R}^{n_1 \times n_2 \times S}$.
- Given a filter $W \in \mathbb{R}^{F \times F \times S}$, where F is the *spatial extent* and a *bias* $b \in \mathbb{R}$, it computes a matrix

$$Z = W *_{12} X := \sum_{i=1}^{S} X[:,:,i] * W[:,:,i] + b.$$

- A convolutional layer consists of K convolutional nodes $((W_i, b_i))_{i=1}^{K} \subset \mathbb{R}^{F \times F \times S} \times \mathbb{R}$ and produces as output a stack $Z \in \mathbb{R}^{n_1 \times n_2 \times K}$ via

$$Z[:,:,i] = W_i *_{12} X + b_i.$$

Introducing Convolutional Layers

- A convolutional node accepts as input a stack of images, e.g. $X \in \mathbb{R}^{n_1 \times n_2 \times S}$.
- Given a filter $W \in \mathbb{R}^{F \times F \times S}$, where F is the *spatial extent* and a *bias* $b \in \mathbb{R}$, it computes a matrix

$$Z = W *_{12} X := \sum_{i=1}^{S} X[:, :, i] * W[:, :, i] + b.$$

- A convolutional layer consists of K convolutional nodes $((W_i, b_i))_{i=1}^{K} \subset \mathbb{R}^{F \times F \times S} \times \mathbb{R}$ and produces as output a stack $Z \in \mathbb{R}^{n_1 \times n_2 \times K}$ via

$$Z[:, :, i] = W_i *_{12} X + b_i.$$

A convolutional layer can be written as a conventional neural network layer!

Activation Layers

The activation layer is defined in the same way as before, e.g., $Z \in \mathbb{R}^{n_1 \times n_2 \times K}$ is mapped to

$$A = \mathsf{ReLU}(Z)$$

where ReLU is applied component-wise.

Pooling Layers

- Reduce dimensionality after filtering.

Pooling Layers

Reduce dimensionality after filtering.

Definition

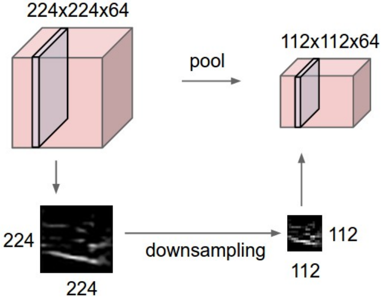
A *pooling operator* $\mathbf{R}$ acts layer-wise on a tensor $X \in \mathbb{R}^{n_1 \times n_2 \times S}$ to result in a tensor $\mathbf{R}(X) \in \mathbb{R}^{m_1 \times m_2 \times S}$, where $m_1 < n_1$ and $m_2 < n_2$.

Pooling Layers

Reduce dimensionality after filtering.

Definition

A *pooling operator* $\mathbf{R}$ acts layer-wise on a tensor $X \in \mathbb{R}^{n_1 \times n_2 \times S}$ to result in a tensor $\mathbf{R}(X) \in \mathbb{R}^{m_1 \times m_2 \times S}$, where $m_1 < n_1$ and $m_2 < n_2$.



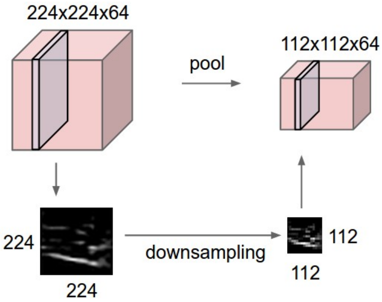
Downsampling

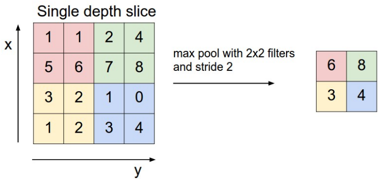
Pooling Layers

Reduce dimensionality after filtering.

Definition

A *pooling operator* $\mathbf{R}$ acts layer-wise on a tensor $X \in \mathbb{R}^{n_1 \times n_2 \times S}$ to result in a tensor $\mathbf{R}(X) \in \mathbb{R}^{m_1 \times m_2 \times S}$, where $m_1 < n_1$ and $m_2 < n_2$.





Max-pooling

Downsampling

Convolutional Neural Networks (CNNs)

Definition

A CNN with L layers consists of L iterative applications of a convolutional layer, followed by an activation layer, (possibly) followed by a pooling layer.

Convolutional Neural Networks (CNNs)

Definition

A CNN with L layers consists of L iterative applications of a convolutional layer, followed by an activation layer, (possibly) followed by a pooling layer.

Typical architectures consist of a CNN (as a *feature extractor*), followed by a fully connected NN (as a *classifier*)

Convolutional Neural Networks (CNNs)

Definition

A CNN with L layers consists of L iterative applications of a convolutional layer, followed by an activation layer, (possibly) followed by a pooling layer.

Typical architectures consist of a CNN (as a *feature extractor*), followed by a fully connected NN (as a *classifier*)

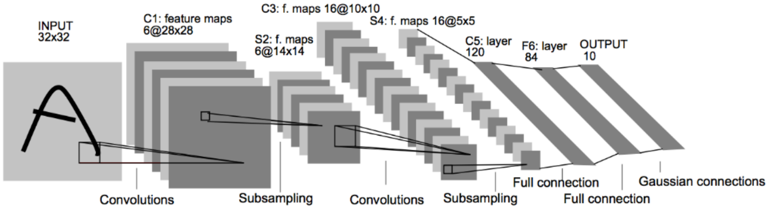


Figure: LeNet (1998, LeCun etal): the first successful CNN architecture, used for reading handwritten digits

Feature Extractor vs. Classifier












Feature Extractor vs. Classifier

Feature Extractor vs. Classifier

D. Trump

B. Sanders

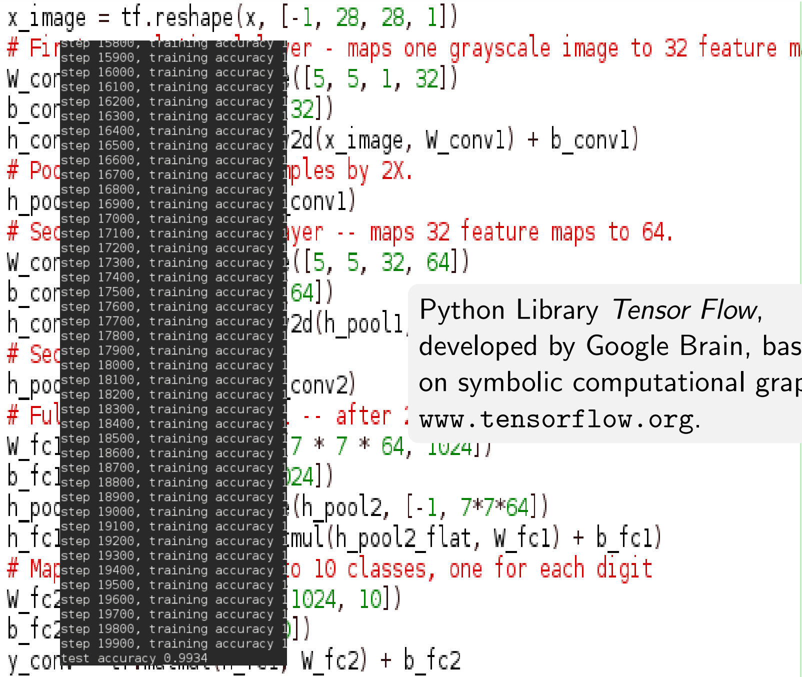
A. Merkel

B. Johnson

```
x_image = tf.reshape(x, [-1, 28, 28, 1])
# First convolutional layer - maps one grayscale image to 32 feature maps.
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
# Pooling layer - downsamples by 2X.
h_pool1 = max_pool_2x2(h_conv1)
# Second convolutional layer -- maps 32 feature maps to 64.
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
# Second pooling layer.
h_pool2 = max_pool_2x2(h_conv2)
# Fully connected layer 1 -- after 2 round of downsampling, our 28x28 image
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
# Map the 1024 features to 10 classes, one for each digit
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
y_conv = tf.matmul(h_fc1, W_fc2) + b_fc2
```

```
x_image = tf.reshape(x, [-1, 28, 28, 1])
# First convolutional layer - maps one grayscale image to 32 feature maps.
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
# Pooling layer - downsamples by 2X.
h_pool1 = max_pool_2x2(h_conv1)
# Second convolutional layer -- maps 32 feature maps to 64.
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1,
# Second pooling layer.
h_pool2 = max_pool_2x2(h_conv2)
# Fully connected layer 1 -- after 2
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
# Map the 1024 features to 10 classes, one for each digit
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
y_conv = tf.matmul(h_fc1, W_fc2) + b_fc2
```

Python Library *Tensor Flow*, developed by Google Brain, based on symbolic computational graphs `www.tensorflow.org`.



Python Library *Tensor Flow*, developed by Google Brain, based on symbolic computational graphs `www.tensorflow.org`.